Titre: Title:	Planification de trajectoire dans un atlas de cartes
Auteur: Author:	Damien Garcia
Date:	2008
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Garcia, D. (2008). Planification de trajectoire dans un atlas de cartes [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/8251/

UNIVERSITÉ DE MONTRÉAL

PLANIFICATION DE TRAJECTOIRE DANS UN ATLAS DE CARTES

DAMIEN GARCIA
DÉPARTEMENT DE GÉNIE ELECTRIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(AUTOMATION ET SYSTÈMES)
MARS 2008

Your file Votre référence
ISBN: 978-0-494-41557-3
Our file Notre référence
ISBN: 978-0-494-41557-3

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

PLANIFICATION DE TRAJECTOIRE DANS UN ATLAS DE CARTES

présenté par: GARCIA Damien
en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de:

M. DESANTIS Romano, Ph.D., président
M. GOURDEAU Richard, Ph.D., membre et directeur de recherche
M. DUPUIS Erick, Ph.D., membre et codirecteur de recherche
M. HURTEAU Richard, Ph.D., membre

REMERCIEMENTS

Ce projet de maîtrise et mon séjour au Québec ont été une expérience très enrichissante, non seulement sur le plan des études mais également sur le plan personnel. Je tiens à remercier tout d'abord mon directeur de recherche, M. Richard Gourdeau, pour m'avoir fait confiance pendant toute la durée de ce projet en m'aidant financièrement et techniquement, et en me donnant l'opportunité d'être chargé de laboratoire dans ses cours. Je remercie ensuite mon co-directeur de recherche M. Erick Dupuis pour m'avoir donné la chance d'apporter peut-être ma modeste contribution à l'immense projet qu'est l'exploration martienne. Je souhaite remercier également les employés du département Automation et Système pour leur amabilité et leur efficacité. Il est évident qu'un tel projet ne peut être réalisé sans le soutien des proches et amis du Québec et de France. Je pense tout d'abord à mes compagnons de voyage Julien, Laurent, David et Bertrand avec qui j'ai partagé cette expérience, ainsi qu'aux nombreux colocataires avec qui j'ai pu passé d'excellents moments. Je ne peux terminer sans remercier plus particulièrement ma copine Karen, mon frère et par dessus tout mes parents qui ont su m'accompagner et me soutenir malgré la distance.

RÉSUMÉ

L'augmentation de l'autonomie des robots est un domaine de recherche très important dans l'exploration spatiale. Ce projet, en collaboration avec l'Agence Spatiale Canadienne, a pour objectif d'améliorer la prise de décision d'un robot dans le choix des trajectoires. Le robot mobile possède, au départ, plusieurs cartes sous la forme de triangles irréguliers avec des résolutions différentes obtenues à partir de ses propres capteurs ou de sources externes (prises de vue satellite, autre robot mobile...). Afin de ne pas perdre d'information et de ne pas accumuler d'erreurs, on évite de ramener notre atlas de cartes en une unique carte. De plus, à cause des limitations en termes de mémoire et de capacité de calcul, la trajectoire devra être calculée à partir des nuages de points bruts modélisant l'environnement sous forme de triangles irréguliers. On souhaite planifier une trajectoire passant par plusieurs cartes. Nous connaissons leur relation en termes de position et d'orientation avec une certaine erreur mais il n'y a pas de relation entre les sommets des triangles dans chaque modèle. L'objectif de ce projet est de déterminer un chemin dont l'origine et la destination seront situées sur des cartes différentes. L'exploration spatiale impose également de nombreuses contraintes comme la sécurité, la fiabilité et la capacité de calcul, qu'il faut absolument prendre en considération.

On considère que l'on a déjà en notre possession un ensemble de cartes de différentes résolutions et que l'on connaît leur position entre elles. On veut définir une trajectoire optimale et sécuritaire joignant deux points qui ne sont pas situés sur une même carte. Il va falloir mettre en oeuvre des algorithmes qui vont, dans un premier temps, trouver des relations et une hiérarchie entre ces cartes. On pourra ainsi passer aisément d'une carte à l'autre, dans le but de trouver le plus court chemin . Ce « plus court chemin » sera calculé à l'aide d'algorithmes de recherche du type Dijkstra ou A*, adaptés et modifiés. Il reste également à déterminer les fonctions de coût qui vont nous permettre de construire le graphe dans lequel on

va rechercher notre plus court chemin. A cause de la taille des graphes et de la longueur de la trajectoire à calculer, il faudra avoir une approche globale de la recherche de trajectoire, en définissant par exemple des points de passages qui pourront ensuite être traités par d'autres algorithmes pour trouver une trajectoire locale plus précise.

ABSTRACT

Increasing rover autonomy is a huge domain of research in space exploration. This project in collaboration with the Canadian Space Agency aims to improve rovers decisions in path planning. At first, the rover has several maps of irregular triangles in several resolutions obtained by his own sensors or external sources (satellites, another robot...). In order not to loose information and not to accumulate errors, we try to keep our maps in an atlas, instead of creating one single big map. Limitations in terms of memory and computing speed impose that the path should be planned directly using the points of the irregular triangle mesh. We wish to plan a trajectory that cross several maps. We know their relative position and orientation assuming some error but there is no relation between two triangles of two different maps. The goal of this project is to find a trajectory starting and ending on two different maps. Space exploration impose also constraints like safety, reliability, and computing speed, that have to be considered.

We assume that we have already a set of maps with different resolutions and that we know their global positions. We want to plan an optimal and safe path joining two points located on different maps. We need to implement algorithms that will create relations between those maps. Then, we will be able to jump from a map to another easily in order to find the shortest path. This shortest path will be computed with algorithms adapted and modifief from Dijkstra or A*. We also have to create cost functions needed to build the graph so as to search for the shortest path. Because of large graph and trajectory length, we will have a global approach to plan the path, computing way points that will be comuted then by other algorithms to find a more precise continuous path.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION

La robotique est sans aucun doute un des domaines représentant le mieux l'évolution technologique des ces dernières décennies ainsi que les futures. Les domaines concernés sont nombreux : la mécanique, l'informatique, l'électronique... Les bénéfices apportés sont immenses : production des industries, augmentation des capacités de l'Homme, précision, puissance, rapidité... La robotique est en perpétuelle évolution et la recherche permet d'inventer des robots de plus en plus habiles, rapides, autonomes et intelligents. L'augmentation de la capacité de calcul des ordinateurs a permis de concevoir des systèmes d'Intelligence Artificielle de plus en plus sophistiqués et performants. Il n'est plus étonnant de voir à présent des robots humanoïdes monter des escaliers, des super-ordinateurs battre les plus grands champions d'échec, et des robots explorer le système solaire...

Les axes de recherche de la robotique sont divers et variés. L'exploration spatiale ne forme qu'une petite partie du vaste domaine de recherche de la robotique mais nécessite déjà, à elle toute seule, énormément de moyens pour atteindre ses objectifs. Les exploits réalisés par ces robots sont remarquables : atteindre des distances telles que Mars, ensuite pouvoir se déplacer dans un milieu aussi hostile que le sol martien, et réaliser de nombreuses tâches permettant de mieux connaître notre planète voisine. Tout cela est effectué avec une intervention de l'homme minimale. Ainsi, l'exploration spatiale se différencie par son besoin d'augmenter sans cesse l'autonomie des robots d'exploration, à cause des contraintes de communication avec la Terre. Il est demandé aux robots d'effectuer de plus en plus de tâches sur des distances de plus en plus importantes.

Pour qu'un robot soit autonome, il doit pouvoir prendre toutes les décisions possibles permettant de mener à bien ses missions, en garantissant sa propre sécu-

rité, et sans l'intervention de l'homme. Ces décisions seront prises par un système d'intelligence artificielle qui devra planifier les tâches à effectuer, les exécuter, et transmettre les informations. Ces tâches peuvent être par exemple : la prise de clichés, le prélèvement d'échantillons, l'analyse de roches, et des déplacements vers des endroits toujours plus éloignés. Le Département de Recherche en Robotique de l'Agence Spatiale Canadienne est actuellement en train de développer un robot d'exploration permettant de parcourir de longues distances avec la plus grande autonomie possible. Dans ce projet, nous allons nous préoccuper plus particulièrement de la planification de trajectoire. Leur recherche s'est orientée vers l'utilisation d'un capteur Laser de type LIDAR (*LIght Detection And Ranging*) pour pouvoir observer l'environnement du robot et déterminer les trajectoires.

Nous allons présenter dans ce mémoire une solution permettant à un robot de se déplacer dans une zone partiellement connue. Ainsi, pour pouvoir se déplacer de façon autonome sur de très longues distances, le robot doit successivement s'informer sur sa localisation et son environnement, planifier son mouvement, puis se déplacer. En effet, le robot ne dispose pas, au départ, de toutes les informations nécessaires à son déplacement. Pour dépasser son horizon visible initial, il doit donc s'informer, seul, au fur et à mesure qu'il avance. Le robot va ainsi construire un atlas de cartes dans lequel il devra être capable de planifier une trajectoire. Nous verrons, dans un premier temps, quelques algorithmes permettant de résoudre ce problème, puis nous tenterons d'analyser leur performance et leurs limites à l'aide de simulations sous Matlab.

CHAPITRE 1

PROBLÉMATIQUE

1.1 L'exploration sur Mars

Les premières sondes spatiales dirigées vers Mars ont été envoyées dans les années 60, celles qui suivirent étaient avant tout destinées à prendre des clichés et faire des analyses depuis l'orbite. Il aura fallu attendre le 4 juillet 1997 et la mission *Path Finder* avec son robot *Sojourner* pour enfin pouvoir poser une roue sur le sol martien. Ce premier *rover* télécommandé aura permis d'explorer 84m sur mars dans toute la mission (figure 1.1). Les missions *Mars Sample Return* prévoyaient d'explorer 100m par jour. Initialement prévues pour 2003 et 2005, elles ont été reportées à 2020. Les robots *Spirit* et *opportunity* de la mission *Mars Exploration Rover* ont permis cet exploit de parcourir 100m par jour à eux deux et d'atteindre le sommet de la colline *Husband* à 3500m du lieu d'atterrissage de *Spirit.* Ces objectifs nécessitent une plus grande autonomie des robots, compte tenu du fait qu'il y a seulement deux fenêtres de communication par jour (Volpe, 1999). A ce jour, les principales missions prévoyant envoyer un *rover* sur Mars sont *ExoMars* prévue pour 2013 par l'Agence Spatiale Européenne (figure 1.2) et *Mars Science Laboratory* prévue pour 2009 (figure 1.3). Le premier partira pour une longue mission d'analyse des sols pour trouver d'éventuelles traces d'eau. Le second aura l'ambitieuse tâche de trouver des signes de vie passée ou présente à la surface du sol.

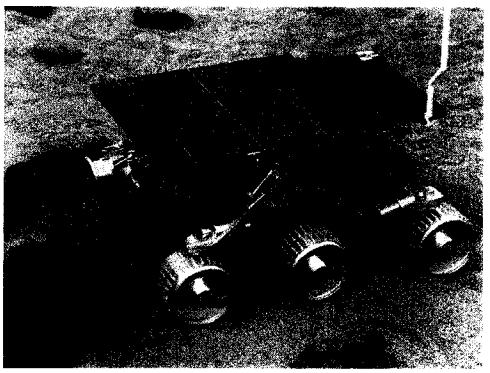


FIG. 1.1 Robot *Sojourner*

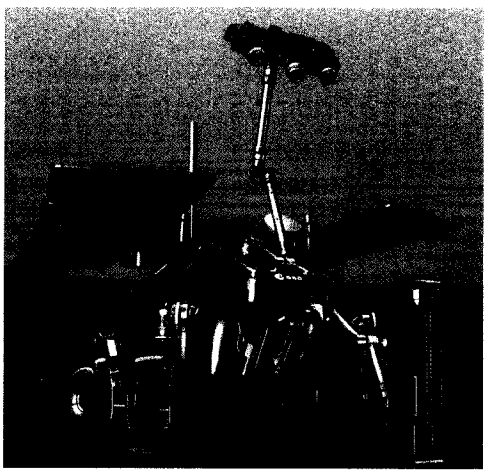


FIG. 1.2 Robot de la mission *ExoMars*

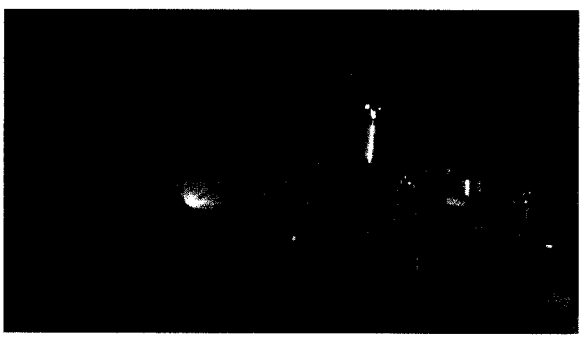


FIG. 1.3 Robot de la mission *Mars Science Laboratory*

1.2 Contexte

L'Agence Spatiale Canadienne est un acteur très important dans la recherche spatiale à l'échelle mondiale (*http ://www.espace.gc.ca*). De nombreux projets sont en collaboration très étroite avec la NASA et l'ESA. L'Agence spatiale est notamment reconnue pour le fameux *CanadArm* équipant les navettes spatiales américaines ou encore la Station Spatiale Internationale associé au robot *Dextre* depuis Mars 2008. L'exploration martienne est également un domaine important de recherche pour l'ASC, non seulement pour les instruments scientifiques embarqués dans les sondes, mais également dans le domaine qui nous intéresse plus particulièrement ici, la robotique mobile.

Actuellement, les efforts de recherche de l'Agence Spatiale concernant les *rovers* d'exploration martienne portent sur l'augmentation de la distance quotidienne que peut parcourir un robot. L'article de E. Dupuis *et al.*, *Towards Autonomous Long Range Navigation* (Dupuis, 2005) décrit très bien les axes de recherche actuels de l'Agence Spatiale Canadienne. Nous allons décrire les grandes étapes de la méthode d'exploration vue par l'Agence Spatiale, depuis l'acquisition des données, jusqu'au guidage du robot.

Dans un premier temps, le terrain est scanné à plusieurs reprises à partir d'endroits différents à l'aide d'un laser type LIDAR permettant d'obtenir plusieurs nuages de points (figure 1.5). Plusieurs traitements de ces données fournissent par la suite un maillage sous la forme de triangles irréguliers (figure 1.4). C'est à partir de ces cartes que l'on cherche à déterminer la trajectoire optimale.

Ici, les cartes sont en fait fusionnées, puis on utilise un algorithme de plus court chemin de type Dijkstra disponible dans la librairie *Java JGraphT*, cherchant à minimiser certaines fonctions de coût. Un exemple de résultat tiré de l'article est

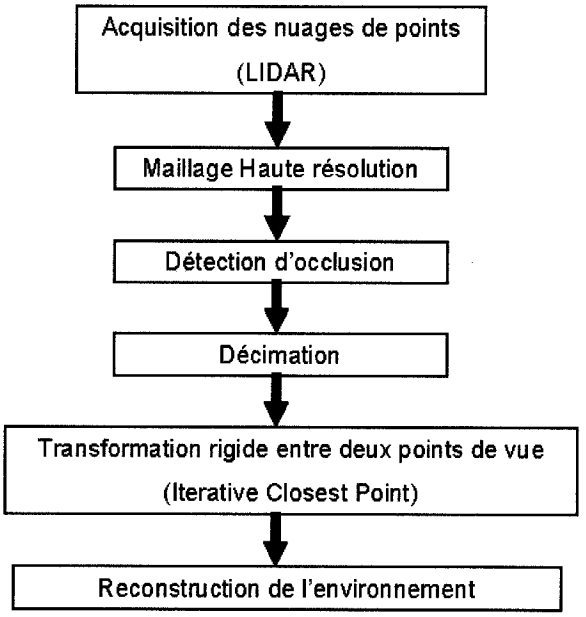


FIG. 1.4 Étapes de construction des cartes

visible sur la figure 1.6.

La trajectoire est ensuite simplifiée pour pouvoir effectuer un suivi de trajectoire le plus robuste et le plus sécuritaire possible.

Les résultats présentés par E. Dupuis *et al.* (Dupuis, 2005) montrent l'efficacité de leur démarche, avec une erreur de positionnement de l'ordre de moins de 1%, pour des trajectoires fermées mesurant jusqu'à 50m. La recherche est maintenant axée vers la localisation et la cartographie en simultanée et une autonomie permettant d'explorer sur des distances supérieures à 100m.

1.3 Objectifs

On a pu voir dans l'exemple précédent que les cartes étaient fusionnées entre elles. Ceci introduit une perte de données et une accumulation d'erreurs puisqu'on ne peut pas se permettre de garder toutes les cartes en mémoire pour la planification

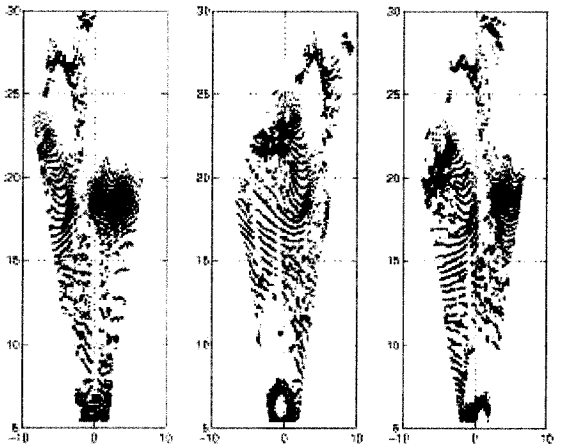


FIG. 1.5 Nuages de points

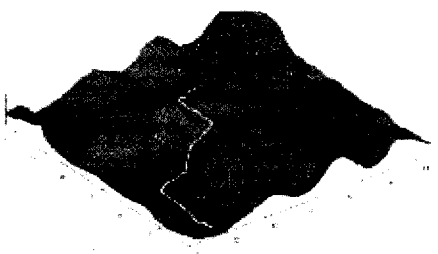


FIG. 1.6 Exemple de trajectoire possible (Dupuis, 2005)

de trajectoire (cartes initiales et cartes fusionnées). Si on souhaite recaler une carte, il faudra supprimer l'actuelle carte fusionnée, puis recommencer. L'avantage d'un atlas de carte est de pouvoir faire des ajustements de position des cartes à tout moment, sans avoir besoin de traiter l'ensemble des données. De plus, avant d'obtenir des scans de terrains à haute résolution, on dispose généralement de cartes à plus faible résolution réalisées à partir de données satellite ou lors de l'amarsissage. Il serait donc très intéressant de pouvoir planifier une trajectoire en diminuant le nombre de cartes intermédiaires fusionnées et en utilisant les avantages de l'étendue des cartes basse résolution et de la précision des cartes haute résolution. Le principal objectif va donc être de déterminer un moyen de passer de cartes en cartes

afin d'évoluer dans l'atlas. Il va falloir également créer des outils permettant de réaliser des essais pour tester nos solutions. Les performances de nos algorithmes seront évaluées selon différents critères. Non seulement on tentera d'obtenir des trajectoires optimales en termes de distance et de sécurité, mais on cherchera à optimiser le plus possible les algorithmes afin d'obtenir des temps de calcul raisonnables et une occupation de mémoire minimale. Il faut bien entendu remarquer que l'optimalité d'une trajectoire peut avoir de nombreuses significations. Il existe une infinité de paramètres pouvant être pris en considération. Bien entendu, la sécurité et la distance sont les principales valeurs à mesurer. La sécurité est elle-même une valeur ayant des définitions différentes, plus ou moins complexes. Il va donc falloir implémenter des fonctions de coûts suffisamment complexes pour pouvoir obtenir des trajectoires les plus sécuritaires possibles. Cela nous permettra également de quantifier les performances de nos algorithmes avec de telles fonctions, demandant beaucoup plus de ressources de calcul.

1.4 Formulation du problème

Nous disposons au départ de quelques cartes du terrain d'essai. L'ASC nous a fourni trois cartes obtenues dans trois positions différentes du robot. Ces cartes ont été à la base utilisées pour les essais vus précédemment en les fusionnant (figure 1.7). Nous disposons ensuite de la carte fusionnée, qui ne sera pas utilisée dans notre cas puisqu'on veut éviter de passer par ce type de cartes. Nous possédons également deux cartes d'ensemble, basse (figure 1.8) et haute résolution (figure 1.9), du terrain martien sous forme de maillage régulier. Ces cartes pourront être intéressantes pour produire des cartes supplémentaires ou de simuler la navigation du robot sur l'ensemble du terrain d'essai.

Chaque carte est, en fait, une liste de points dans l'espace dans les coordonnées

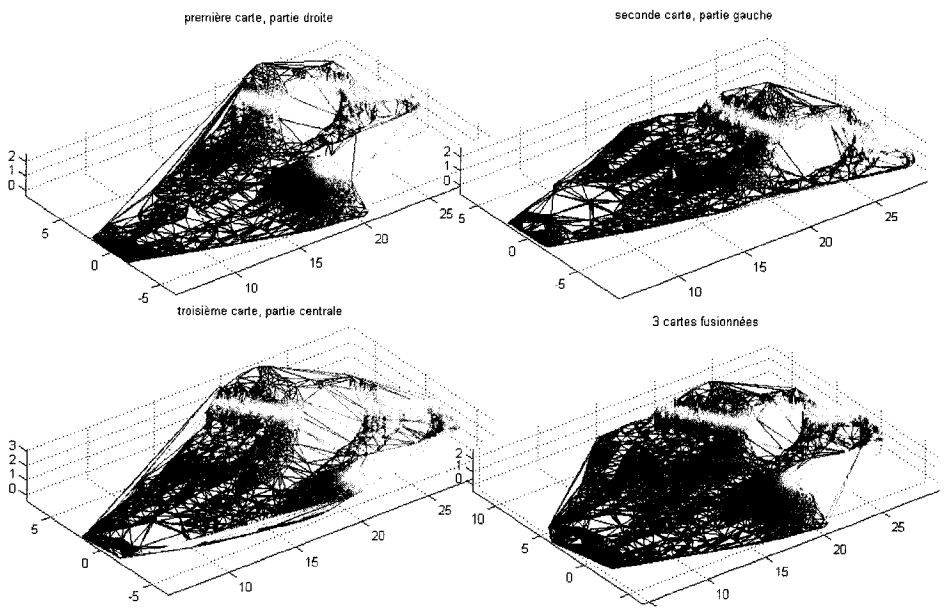


FIG. 1.7 Maillage triangulaire irrégulier, cartes obtenues par le robot

cartésiennes X,Y et Z. Nous utilisons ensuite la triangulation de Delaunay disponible dans Matlab pour réaliser le maillage représenté dans les figures 1.7, 1.8 et 1.9. L'ASC utilise d'autres algorithmes pour la triangulation, nous nous contenterons de ceux de Matlab pour les essais. Le problème rencontré dans notre cas est l'apparition de triangles qui modélisent mal le terrain réel. Par exemple, une portion du terrain peut être occultée par une roche du point de vue du robot. Ainsi, les triangles créés derrière sont obtenus à partir des points autour de l'ombre de la roche. Ces triangles ne sont qu'une interpolation du terrain comme étant entièrement plat derrière la roche. Pour des raisons de sécurité, dans le doute, on ne devrait pas tenir compte de ces triangles, puisqu'ils correspondent finalement à une zone inconnue du terrain. On supposera donc que ces triangles n'existeront plus dans la réalité lors de la réalisation des cartes. Nous les utiliserons tout de même dans notre cas, dans un premier temps, faute d'avoir les algorithmes appropriés de détection d'occlusion.

Concernant la position des cartes dans l'espace, pour les deux cartes globales, les

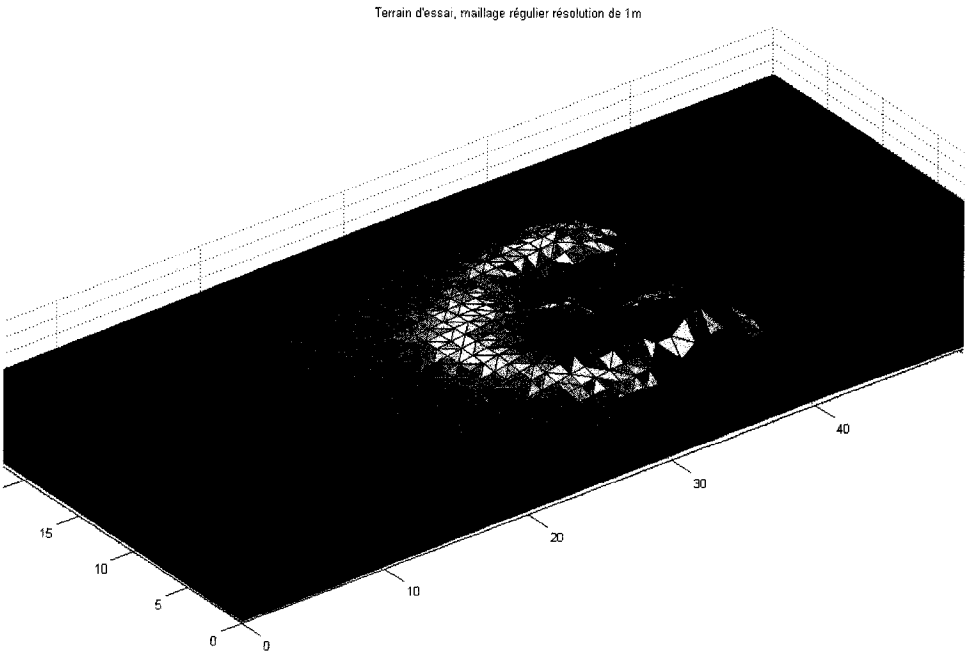


FIG. 1.8 Maillage régulier avec une résolution de 1m

origines correspondent à une des extrémités du terrain. Dans le cas des trois cartes à maillage irrégulier obtenues directement à partir du robot, l'origine correspond au LIDAR. Il faudra donc, dans un premier temps, positionner ces cartes entre elles. Nous ne disposons pas de l'algorithme basé sur *l'Iterative Closest Point* pour le faire. Ce positionnement des cartes n'étant pas l'objectif du projet, nous tenterons de le réaliser sans pour autant rentrer dans les détails. Ce positionnement est en pratique facilité par les systèmes de navigation et de localisation du robot.

Enfin, il serait impossible de conclure sur les performances réelles de nos algorithmes si nous nous contentons d'essais sur ces quelques cartes. Il va donc être indispensable de mettre en place des outils qui nous permettrons d'obtenir de nouvelles cartes à partir des deux cartes globales et plus particulièrement la carte haute résolution. Nous tenterons donc de simuler un scan du terrain comme on le ferait avec le LIDAR en conditions réelles. Ces scans ainsi obtenus seront une bonne base pour faire les essais des algorithmes de planification.

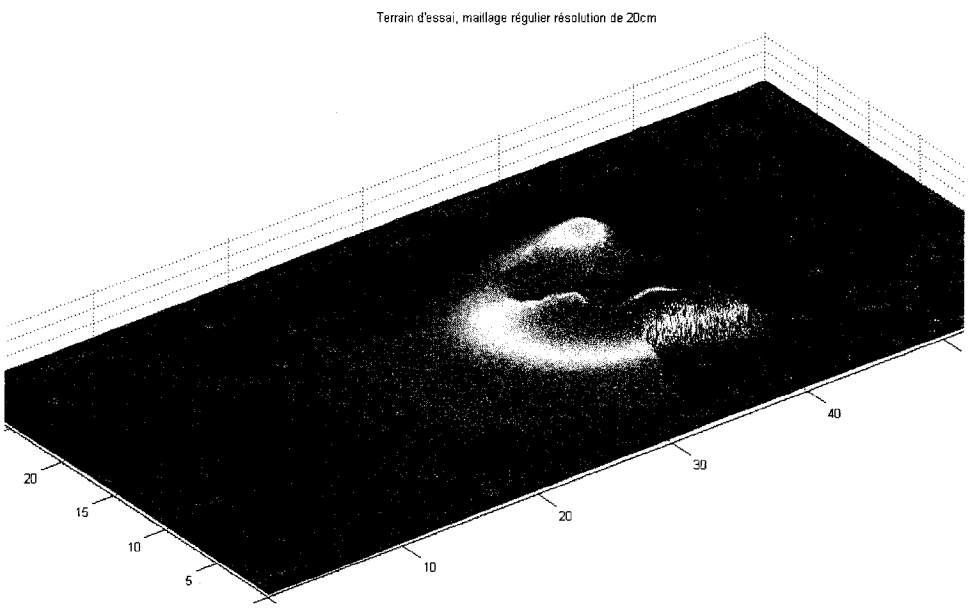


FIG. 1.9 Maillage régulier avec une résolution de 20cm

1.5 Eléments de recherche ((Lavalle 2006) et (Latombe, 2006))

1.5.1 Représentation de l'environnement

Bien que la définition du problème limite énormément les axes de recherche, nous allons passer en revue quelques solutions et notions de base généralement utilisées pour la planification de trajectoire.

La planification de mouvement ne concerne pas seulement la robotique mobile. Elle est également présente dans les robots tels que les bras manipulateurs, en vision, en imagerie par ordinateur, ou encore pour la résolution de problèmes complexes tels que *rubicube, taquin, etc...* Pour tout problème de planification, il est nécessaire dans un premier temps de modéliser l'environnement. Dans le cas de la robotique mobile, l'environnement va être modélisé par des éléments géométriques. Un environnement peut être composé d'objets structurés ou non, solides ou déformables, fixes ou mobiles. Certains modèles sont plus adaptés pour certaines situations. Voici une liste de représentations généralement utilisées :

- Polygones
- Polyèdres
- Modèles algébriques et semi-algébriques
- Triangles 3D
- NURBS (Nonuniform rational B-Splines)
- Superquadrics
- Cylindres généralisés

Dans notre cas, nous utiliserons un modèle sous-forme de Maillage Triangulaire Irrégulier en 2.5D. Pour pouvoir être cohérent avec les hypothèses de base convenues avec L'Agence Spatiale Canadienne, il faudrait généraliser à un modèle 3D, avec

des structures telles que des cavernes ou des viaducs. Les données initiales étant sous forme d'un nuage de points donné par le LIDAR, ce type de modèle est le plus simple permettant d'obtenir une représentation convenable du terrain. En effet, nous avons affaire à un terrain sans cavités, non structuré et fixe. Une représentation de ce type est donc largement préférable par rapport aux autres.

1.5.2 Espace de configuration

Une fois que nous avons un modèle de l'environnement, il faut pouvoir définir les états possibles de notre robot. On considère que notre robot possède un seul corps rigide. Ainsi, un état va être défini dans l'espace par une certaine pose. Une pose χ est composée d'une position $[x, y, z]$ et d'une orientation $[\psi, \theta, \phi]$ si on considère la représentation des angles d'Euler. Cependant, notre robot est limité à des déplacements dans un plan, même si la dénivélation du terrain va forcément modifier l'altitude z et l'orientation. On peut ainsi ramener la pose de notre robot à des coordonnées $[x, y, \theta]$ (figure 1.10).

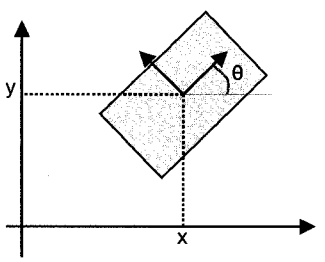


FIG. 1.10 Coordonnées $[x, y, \theta]$ du modèle du robot

L'environnement est représenté sous forme d'espace des configuration C qui dans notre cas sera $\mathbb{R}^2 \times [0, 2\pi[$ où tout état $q \epsilon C$ avec $q = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$. Cet espace des configuration va être composé d'obstacles définissant un sous-espace C_{obs} et d'un

sous-espace libre C_{free}. C'est la forme du robot et ces capacités de franchissement qui vont déterminer quels états q forment les sous-espaces C_{obs} et C_{free}.

1.5.3 Modélisation du robot

L'espace libre étant créé, la planification d'un mouvement nécessite la prise en compte des moyens de déplacement du robot pour pouvoir tracer des trajectoires acceptables entre un état de départ q_d et un état final q_f. Là encore, il existe un trés grand nombre de types de robot mobile. On ne parlera pas des robots mobiles « marcheurs », ceux-ci peuvent certes franchir toute sorte d'obstacle mais font l'objet de peu de recherche à cause de leur complexité en terme de commandabilité et de conception. Le *rover* à roues est donc sans aucun doute le moyen le plus efficace pour se déplacer sur un sol tel que Mars. La figure 1.11 illustre quelques modèles couramment utilisés en robotique mobile. L'utilisation de roues comme moyen de déplacement introduit des contraintes non-holonomes qui vont limiter les possibilités de déplacements. Ainsi, un robot ne pourra généralement pas se déplacer « en travers », ce qui limite les possibilités pour la planification de trajectoire. Seuls quelques robots dits « omnidirectionnels » peuvent être commandés suivant toutes les dimensions de l'espace libre C_{free}.

Dans notre cas, il sera nécessaire de simplifier notre espace de configuration C à $\mathbb{R}^2$, c'est à dire que nous ne considèrerons que les coordonnées x et y d'un état q. Ceci est acceptable dans un premier temps pour déterminer une trajectoire globale. Une trajectoire est normalement définie comme une fonction du temps $q(t)$ avec $t\epsilon[t_d, t_f]$ tel que $q(t_d) = q_d$ et $q(t_f) = q_f$. Dans notre cas, nous assimilerons une trajectoire comme une succession d'états discrets $[q_d, \ldots, q_i, \ldots, q_f]$ indépendants du temps. On parle alors dans ce cas de chemin plutôt que de trajectoire . Le fait de ne pas tenir compte de l'orientation θ et du temps permet de ne pas ce

soucier des contraintes non-holonomes lors de la planification du mouvement. Ces contraintes sont généralement prises en compte dans un second temps en définissant une trajectoire continue avec des rayons de courbures appropriés puis dans un dernier temps lors du guidage du robot au moment où on introduit l'orientation θ et le temps t.

1.5.4 Méthodes de planification

Différentes approches sont utilisées pour réaliser la planification, une fois que l'environnement et le robot sont modélisés. On retrouve souvent les méthodes représentées dans la figure 1.12. Les méthodes sous forme de décomposition spatiale, à savoir, la décomposition trapézoïdale, les *roadmaps,* et les *quadtrees,* permettent de nous ramener finalement à une recherche en graphe, semblable à ce que nous décrirons plus tard. La méthode du champ de potentiel va nécessiter des algorithmes d'optimisation du type « descente du gradient »par exemple.

1.5.5 Modèles utilisés pour la planification dans un atlas de cartes

En ce qui nous concerne, nous allons utiliser la décomposition spatiale déjà faite par le maillage triangulaire irrégulier. Cela va éviter de rajouter d'autres étapes pour produire le graphe dans lequel on va planifier notre chemin. Chaque triangle représente un noeud et aura au maximum trois voisins. On ne fera donc pas d'analyse préalable de l'espace de configuration pour détecter les obstacles. Il suffira d'associer un coût ∞ au lien amenant à un noeud infranchissable par le robot. L'espace de configuration sera donc décomposé en même temps que la création du graphe. Les coûts seront déterminés à partir d'une forme de robot circulaire à priori représentant une zone de sécurité autour de celui-ci (figure 1.13).

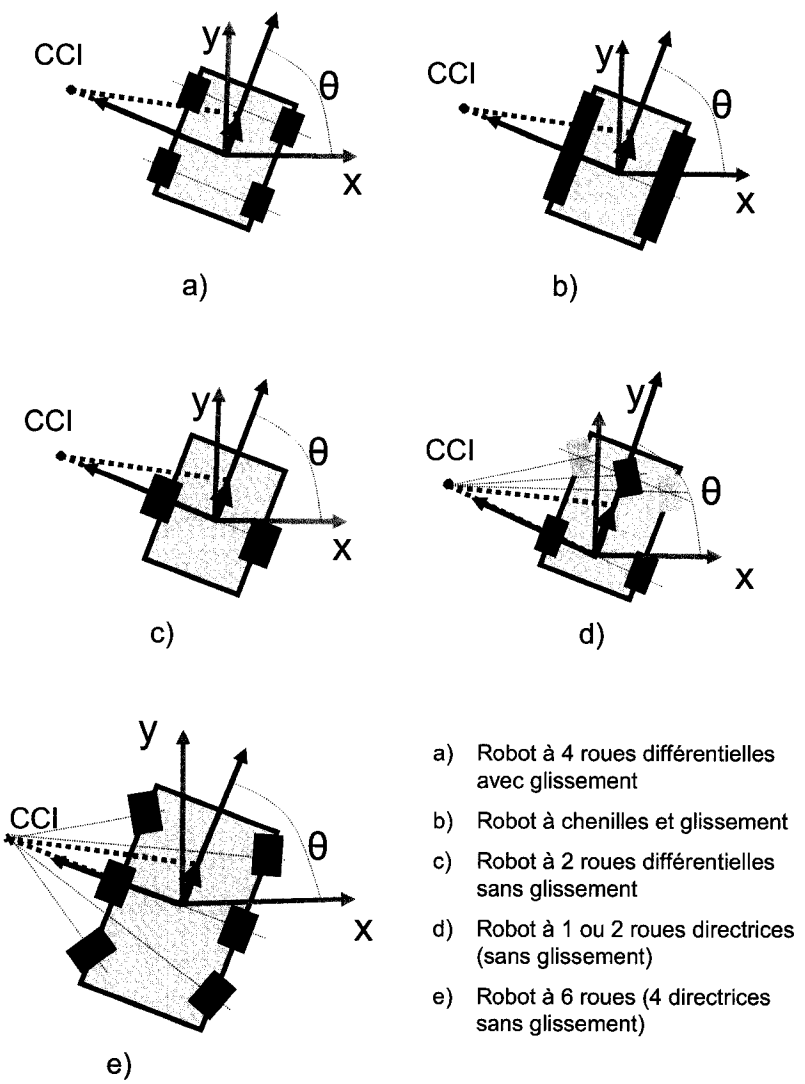


FIG. 1.11 Quelques types de *rovers*

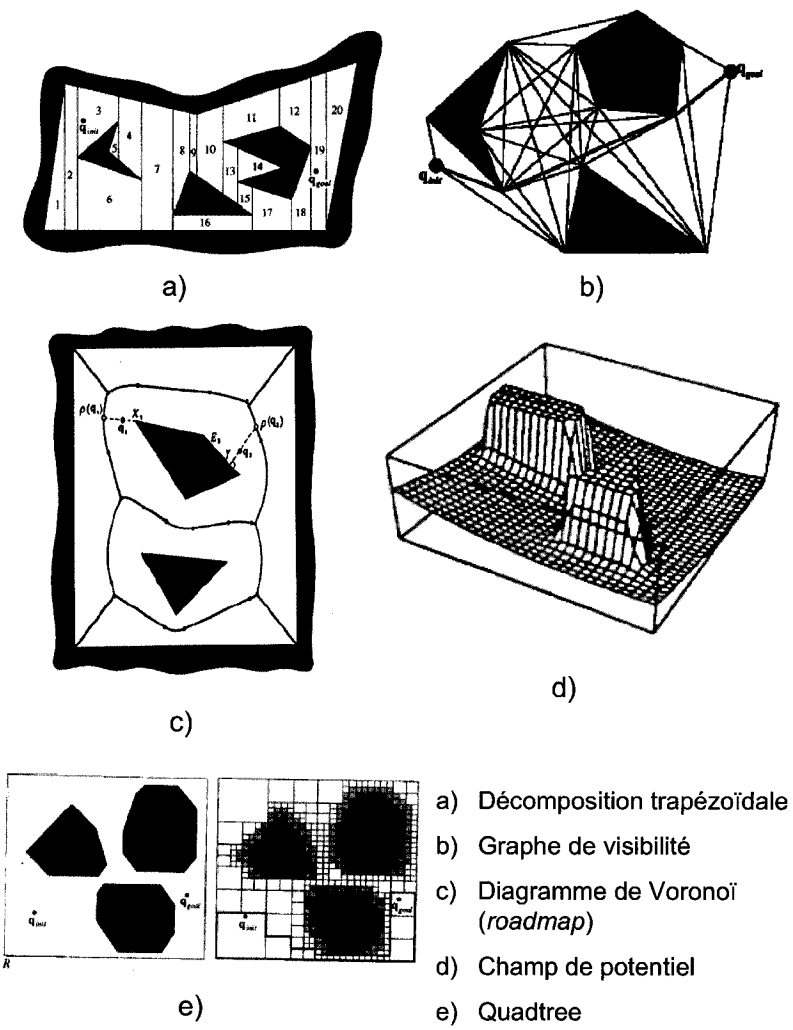


FIG. 1.12 Méthodes de planification (Latombe, 2006)

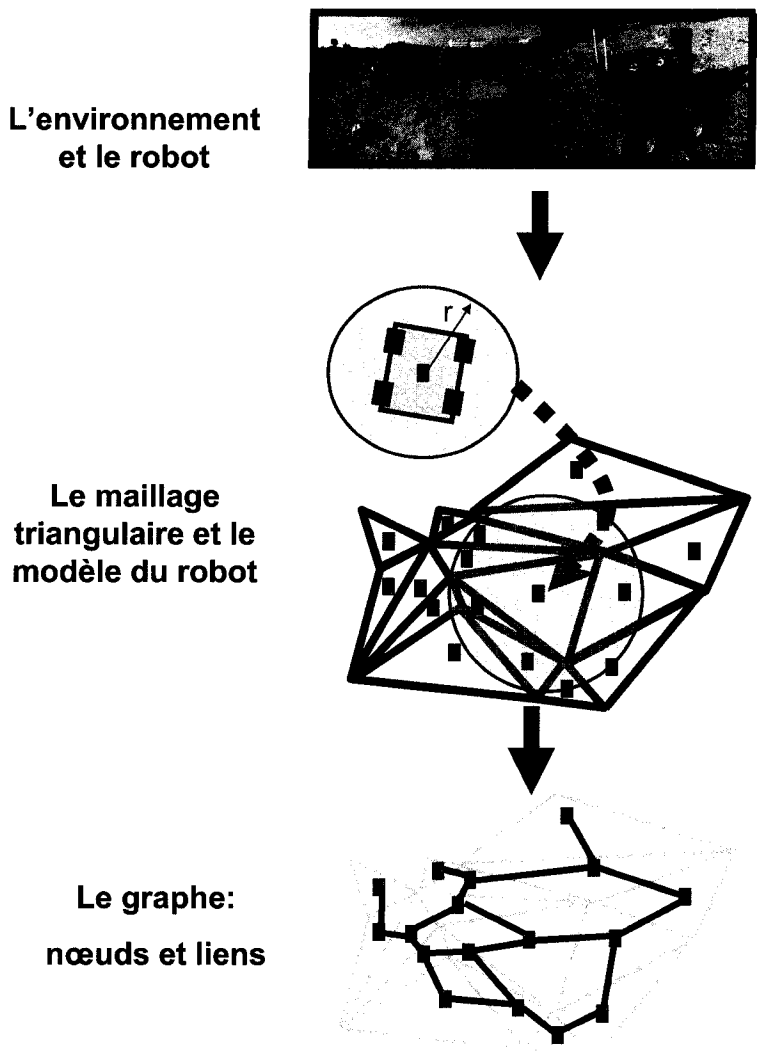


FIG. 1.13 Résumé des représentations retenues

1.5.6 Guidage

Nous allons donc présenter un moyen de planifier un chemin composé d'un suite d'états sous la forme $\tau = ([x_d, y_d], \ldots, [x_i, y_i], \ldots, [x_f, y_f])$. Ce chemin global, sous forme de dents de scie, sera ensuite la base pour planifier une trajectoire continue grâce à des fonctions du type B-Splines (Agudelo, 2008). Une telle trajectoire est indispensable pour pouvoir ensuite contrôler le robot plus efficacement. Nous ne nous attarderons pas ici sur les moyens de contrôle de suivi de trajectoire. Nous invitons le lecteur à ce documenter sur les lois de contrôle d'*A.Astolfi* (Astolfi, 1999). L'Agence Spatiale Canadienne utilise un contrôleur basé sur un retour d'état discontinue proposé par ce dernier (Dupuis, 2005). Cela nécessite donc, bien évidemment, un système de localisation permettant de situer le robot lorsque cela est nécessaire. L'ASC, pour les essais réalisés sur le terrain, utilise un robot équipé de capteurs tels qu'une centrale inertielle, un odomètre, ou encore un compas (boussole indiquant l'orientation du robot, remplacée par un capteur solaire sur Mars) (Dupuis, 2005). Un gyroscope permet d'estimer l'orientation du robot, puis un accéléromètre est utilisé afin de recaler les orientations par rapport à la gravité. L'odomètre sur les roues, permet d'obtenir une estimée de la position. Ces capteurs ont besoin d'être recalibrés régulièrement par un autre système de localisation basé sur les cartes obtenus par le LIDAR. Des algorithmes tels qu'*ICP* (*Iterative Closest Point*) permettent de faire coïncider plusieurs nuages de points par transformation. En effet une erreur est accumulée au fur et à mesure de l'avancement, au niveau de l'odomètre à cause du glissement des roues sur le sol, et au niveau de la centrale inertielle à cause de la dérive du gyroscope. Il est donc indispensable de prendre le temps de remettre à jour la position estimée et de recalibrer les capteurs. Ce recalibrage prend en revanche du temps, il necessite un scan de terrain suivi d'une superposition des cartes. Il doit donc pouvoir être utilisé un minimum de fois. Ces estimations de la position sont donc combinées entre elles puis filtrées à l'aide

d'un filtre de *Kalman* afin de diminuer l'erreur produite par le bruit de mesure de l'inertie et par le glissement des roues. Ces méthodes font partie du concept *SLAM* (*Simultaneous Localization And Mapping*). C'est le nom généralement donné à tout système de déplacement d'un robot mobile, dans un environnement inconnu, où on construit une carte de cet environnement en même temps qu'on l'explore.

CHAPITRE 2

ÉTUDE DE LA PLANIFICATION DANS UN ATLAS DE CARTES

2.1 État de l'art

Nombreux sont les articles traitant de l'exploration longue distance sur Mars (Nagatani, 1999)(Parrish, 2001)(Volpe, 1999). Cependant, pour la plupart d'entre eux, les méthodes décrites ne correspondent pas exactement à nos attentes. Par exemple, de nombreux robots naviguent à l'aide de données provenant de caméras stéréo. D'autres utilisent des cartes topologiques pour planifier la trajectoire, avec l'inconvénient de devoir être recalculées à chaque fois que l'on obtient de nouvelles données de terrain. On peut parler par exemple de cartes de type DEM (*Digital Elevation Maps),* ou les *quad-tree* déjà utilisés par l'Agence Spatiale (Dupuis, 2005) mais peu appropriés à nos objectifs. On cherche donc plutôt à utiliser directement les cartes de différentes résolutions sous forme de maillage en triangles irréguliers. Il va donc falloir déterminer une solution basée directement sur la théorie des graphes et apporter des modifications aux nombreux algorithmes déjà existants de recherche de plus court chemin dans un espace d'états.

2.2 Solutions envisagées

Nous allons tout d'abord résumer le déroulement d'un voyage d'exploration sur Mars. On dispose dans un premier temps d'une carte de mauvaise résolution obtenue pendant l'amarsissage ou à l'aide de satellite en orbite, voire même de précédentes explorations. Ces cartes sont utilisées au départ pour se situer lors de

l'amarsissage. En effet le point désiré et l'emplacement réel d'arrivée peuvent se situer à des centaines de mètres l'un de l'autre, voire même des kilomètres. Une fois cette localisation effectuée, un objectif est donné. Les fenêtres de communication étant très réduites, il faut donc un maximum d'autonomie au robot. Celui-ci doit être capable d'effectuer des centaines de mètres seul, sans intervention humaine. Ainsi, l'objectif donné ne sera généralement pas visible par le robot, il doit donc rechercher une trajectoire à l'aveugle, au-delà de son horizon visible, et se diriger vers son objectif, en tenant compte des obstacles à éviter à une échelle locale tels que rochers et trous ainsi qu'à une échelle plus globale tels que collines ou cratères. Le robot doit donc avancer dans les zones suffisamment connues et sécuritaires puis prendre de nouvelles informations lorsque cela est nécessaire. Ainsi, le robot devrait avancer de carte en carte, alternant prise de donnée, analyse, planification, déplacement, prise de donnée, etc...

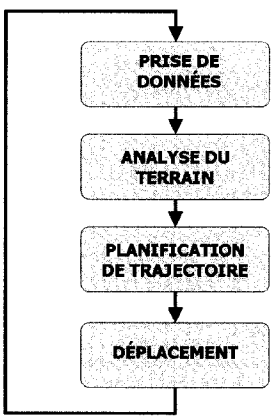


FIG. 2.1 Déroulement d'une séquence de navigation dans un terrain partiellement connu

Dans notre cas, la prise de donnée est en fait un balayage du terrain grâce au LIDAR. L'analyse du terrain correspond au traitement des données laser pour cartographier l'environnement. L'analyse des données va donc consister à détecter les zones d'ombre, décimer puis finalement créer le maillage triangulaire. Une fois la carte créée, il faut l'intégrer dans le graphe. Ainsi, chaque carte doit pouvoir

être insérée dans le graphe indépendamment des autres, afin d'éviter de recalculer à chaque fois un nouveau graphe et tenir compte de toutes les cartes. Dans un tel cas, plus le robot avancera, plus la quantité de données augmentera, plus les calculs seront lourds. Le graphe sera donc sous forme d'un atlas de cartes positionnées entre elles, dans lequel l'algorithme de recherche pourra se déplacer de cartes en cartes pour déterminer la trajectoire optimale. Dans un dernier traitement, cette trajectoire sera lissée et adaptée aux conditions réelles, pour que le robot puisse effectivement se déplacer le long de cette trajectoire.

Notre problème se situe donc au niveau de la création du graphe et de l'algorithme permettant d'explorer sur toutes les cartes en même temps. Pour cela, nous devons trouver un moyen pour passer d'une carte à l'autre, tout en les gardant indépendantes les unes des autres. Le seul lien entre elles est, finalement, leur position relative. Avec une certaine erreur due au système de localisation, les cartes étant bien positionnées entre elles, on peut se positionner plus ou moins précisément sur chaque carte en même temps. Ainsi, à chaque instant de la planification, on doit savoir où l'on se situe sur le terrain et trouver l'endroit correspondant sur chaque carte. Il est inutile de déterminer ceci dans le graphe, et faire correspondre à chaque position une autre position possible dans les autres cartes, les calculs deviendraient de plus en plus lourd avec le temps. Une solution serait donc de quadriller le terrain à explorer pour classer les triangles dans chaque case, et faciliter ainsi la recherche de la position dans les autres cartes disponibles à cet endroit là. Ceci peut s'effectuer facilement lors de la création du graphe et ne demande pas beaucoup de puissance de calcul puisqu'on peut savoir immédiatement dans quelle case le triangle va se situer en regardant ses coordonnées. La recherche de la position semblable sur les autres cartes s'effectuera donc uniquement au moment de la planification, et pour les positions qui nous intéressent réellement.

Un nouveau problème se pose, maintenant que l'on sait se déplacer de carte en

carte : il va falloir déterminer à quel moment changer de cartes et également quelle carte choisir. Ceci revient finalement au même que choisir tel point pour une trajectoire : il faut donc mettre en place des fonctions de coût qui vont permettre d'effectuer ce choix.

2.3 Le graphe

Nous allons tout d'abord préciser notre vision du graphe dans notre situation. Un graphe est composé de noeuds et de liens. Chaque noeud correspond à un état. Dans notre cas, afin de rester proche du mode de fonctionnement des algorithmes de l'ASC, nous prendrons comme état, le centre d'un triangle de la carte. Il faut ensuite définir les liens. Un lien possède un noeud de départ, un noeud d'arrivée et un coût. Ces liens permettent de définir les voisins d'un noeud. Dans notre cas, les liens sont bidirectionnels et correspondent au passage d'un triangle à un triangle voisin, c'est à dire un triangle avec une arête commune. On pourrait également étendre les voisins à ceux ayant un sommet commun mais cela ne change rien à la structure du graphe : ceci augmenterait le nombre de voisins possibles d'un noeud, passant de trois à une infinité de voisins possibles. Ceci augmenterait énormément ce que l'on appelle le facteur de branchement b du graphe. Or, si ce facteur de branchement devient trop important, le parcours de l'arbre peut devenir extrêmement long.

A partir des cartes, nous devons donc calculer le barycentre du triangle, puis déterminer les voisins et calculer le coût du passage d'un voisin à l'autre. Ce coût doit bien entendu tenir compte de la distance mais doit également prendre en considération la pente et l'éventuelle présence d'un obstacle par exemple. Là encore cette fonction de coût peut être aisément modifiée pour être plus complète. On pourrait penser par exemple à prendre en compte les dimensions réelles du robot en regardant l'environnement de chaque triangle, mais ceci, bien entendu, au détriment du

temps de calcul. Dans un premier temps, à des fins d'essais, nous utiliserons des fonctions de coûts simples telles que décrites ci-dessous. Nous étudierons plus tard la seconde solution.

$$cout(noeud_1, noeud_2) = \sum c_i(noeud_1, noeud_2)$$

avec par exemple

$$c_1 = \begin{cases} \infty & \text{si } |z_1 - z_2| > 0.5 \\ 0 & \text{sinon} \end{cases}$$

$$c_2 = \begin{cases} \infty & \text{si } \frac{|z_1 - z_2|}{\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}} > 0.4 \\ 0 & \text{sinon} \end{cases}$$

$$c_3 = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Ici, notre fonction de coût va être la somme de trois coûts différents. Le premier coût c_1 permet d'identifier un franchissement semblant trop brusque. Le coût c_2 pénalise les pentes trop élevées qui risqueraient de faire glisser le robot voire de rendre impossible son déplacement. Enfin c_3 est un simple calcul de la distance. Ces trois fonctions vont être peu efficaces en réalité puisqu'elles ne tiennent pas compte des dimensions du robot. Elles ont en revanche l'avantage d'être très simples, et nous permet de décrire les principes de la planification dans un atlas de cartes sans ce préoccuper dans un premier temps de la fonction de coût. Nous verrons plus tard si nous pouvons implémenter des coûts plus complexes (section 3.3.2).

Alors que nous créons le graphe qui va permettre le calcul de la trajectoire, nous devons également créer le quadrillage qui va permettre de se situer du mieux possible sur toutes les cartes en même temps à tout moment. Chaque carré de terrain peut optionnellement être composé d'un nom du type *Ri-Fi* ou *Li-Bi* pour *right, left, front* et *back,* afin de faciliter la localisation pour l'utilisateur. Ce carré sera surtout défini par sa position dans l'espace et aura une taille fixe. Cette taille peut être modifiée aisément pour optimiser la planification (il serait possible de pouvoir modifier cette taille alors que le graphe est créé, il faudrait alors réorganiser entièrement le classement des noeuds dans chaque case). Plus le quadrillage sera grand, plus le nombre de points dans ce cadre sera important et la recherche des points similaires d'une carte à l'autre sera donc plus longue. A l'inverse, si le maillage est trop petit, le choix pour ces points similaires sera minime voire nul et diminuera la qualité de la planification. Enfin, un carré sera composé de noeuds de chaque carte selon les disponibilités. Cette liste de noeuds permettra, lors de la planification, de retrouver facilement les noeuds similaires dans les autres cartes.

Ces données seront modélisées dans Matlab à l'aide de structures nommées *main_ mesh* et *graph* :

main_mesh		place	$[x_{min}, x_{max}, y_{min}, y_{max}]$
		name	*'Ri-Fi'*
		maps	nodes $[n_1, n_2, ...]$
graph	xyz		$[x_i, y_i, z_i]$
	ref	carte	n_{carte}
		num	n_{noeud}
		case	n_{case}
	next		$[v_1, v_2, v_3]$
	link		$[cout(i, v_1), cout(i, v_2), cout(i, v_3)]$

TAB. 2.1 Structure du graphe

Le logigramme de la figure 2.2 décrit l'algorithme dans sa version simple permet-

tant de créer le graphe et le quadrillage. Nous verrons dans une seconde phase les modifications à apporter pour optimiser l'algorithme et augmenter les performances.

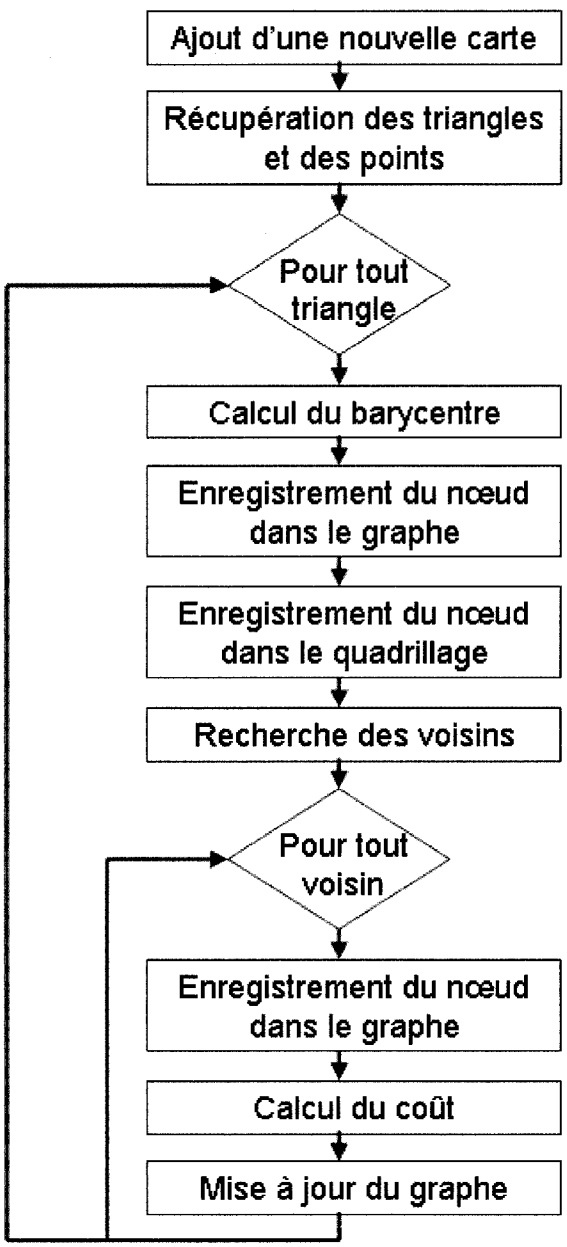


FIG. 2.2 Algorithme de réalisation du graphe

2.4 L'algorithme de planification (Russel, 1995)

Une fois le graphe créé, nous devons à présent déterminer l'algorithme qui va permettre de parcourir le graphe pour trouver une trajectoire optimale. Il existe de nombreux algorithmes d'exploration de graphes, on peut citer les algorithmes simples de type « recherche en profondeur » ou « en largeur », mais ceux-ci sont bien évidemment inappropriés à nos objectifs de rapidité et d'optimalité. Ces algorithmes ne possèdent pas une stratégie d'exploration suffisamment évoluée. La stratégie d'exploration est en fait ce qui permet de choisir le noeud suivant, désigné à être développé. Sans stratégie d'information, on pourrait développer une grande quantité de noeuds qui ne nous mèneront jamais à une solution, notamment dans le cas d'une exploration en graphe comme dans notre cas (à l'opposé d'une exploration en arbre qui ne possède pas de répétition d'état). L'ASC utilise l'algorithme de Dijkstra pour réaliser l'exploration. Cet algorithme possède cependant quelques inconvénients. En effet, dans sa version simple, l'algorithme de Dijkstra est un algorithme dit « non-informé », c'est-à-dire qu'il n'utilise pas d'autre information que celle donnée par le graphe et les coûts de chaque lien pour explorer le graphe. Son principe est de choisir le noeud suivant ayant le coût le plus faible, c'est à dire le plus « proche » de l'origine. Ce n'est pas la meilleure stratégie d'exploration. Nous aboutirons certes vers la solution optimale, mais avec une vitesse d'exécution encore trop lente.

Dans notre cas nous savons finalement où aller, c'est une information supplémentaire que l'on pourrait utiliser en notre faveur. En effet, pour aller de Montréal à Québec, il est inutile de regarder du coté de Ottawa, c'est le principe de la recherche dite « informée ». Le meilleur algorithme pour notre cas est l'algorithme A*, celui-ci est en effet complet et optimal sous certaines conditions, et est surtout beaucoup plus rapide que tout autre algorithme de recherche « non-informée ».

Son fonctionnement est plus ou moins le même que l'algorithme de Dijkstra, à la différence qu'on ne cherche pas à choisir le noeud le plus « proche » du noeud de départ, mais celui qui semble être le plus dans la direction du noeud d'arrivée.

Dans une exploration en graphe comme la notre, à savoir une application dans le domaine de la planification de trajectoire, il est relativement facile d'utiliser la distance par rapport au noeud d'arrivée comme heuristique. Une heuristique $h(n)$ est une estimée du coût minimal pour aller du noeud courant n jusqu'au noeud d'arrivée. Connaissant le coût $g(n)$ entre le noeud de départ et le noeud courant n, on possède donc une estimée du coût total de la solution $f(n)$. La stratégie d'exploration de l'algorithme A* va donc être de choisir le noeud suivant possédant le plus faible « coût total estimé » $f(n)$.

$$\begin{aligned} f(n) &= g(n) + h(n) \\ \text{où } h(n) &= distance(n, n_{final}) \\ \text{et } g(n) &= \sum_{depart}^{n} cout(noeud_i, noeud_j) \end{aligned}$$

Les figures 2.3, 2.4 et 2.5 décrivent le déroulement de l'exploration A*.

A chaque fois qu'on développe un noeud, c'est-à-dire qu'on regarde les noeuds suivants disponibles à ce noeud, on étend notre arbre de recherche. Cependant l'exploration en graphe doit prendre en compte la possibilité de répétition d'état, c'est pour cela qu'à chaque fois que l'on arrive à un état déjà visité, on ne garde que le chemin menant à ce noeud avec le chemin le moins coûteux.

Nous avons indiqué plus haut que l'algorithme A* est optimal sous certaines conditions, nous allons les décrire dans ce paragraphe. Dans le cas de l'exploration en

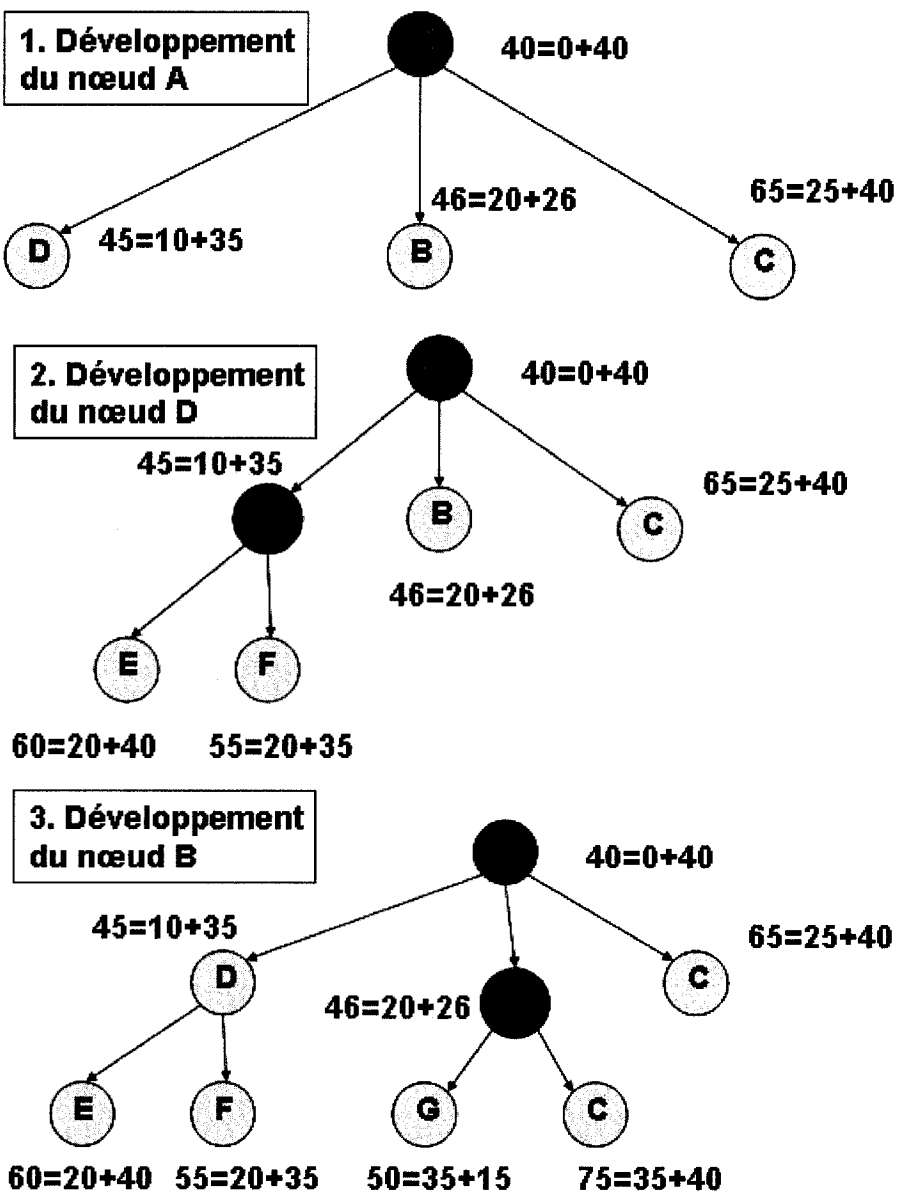


FIG. 2.3 Fonctionnement de l'algorithme A* (1)

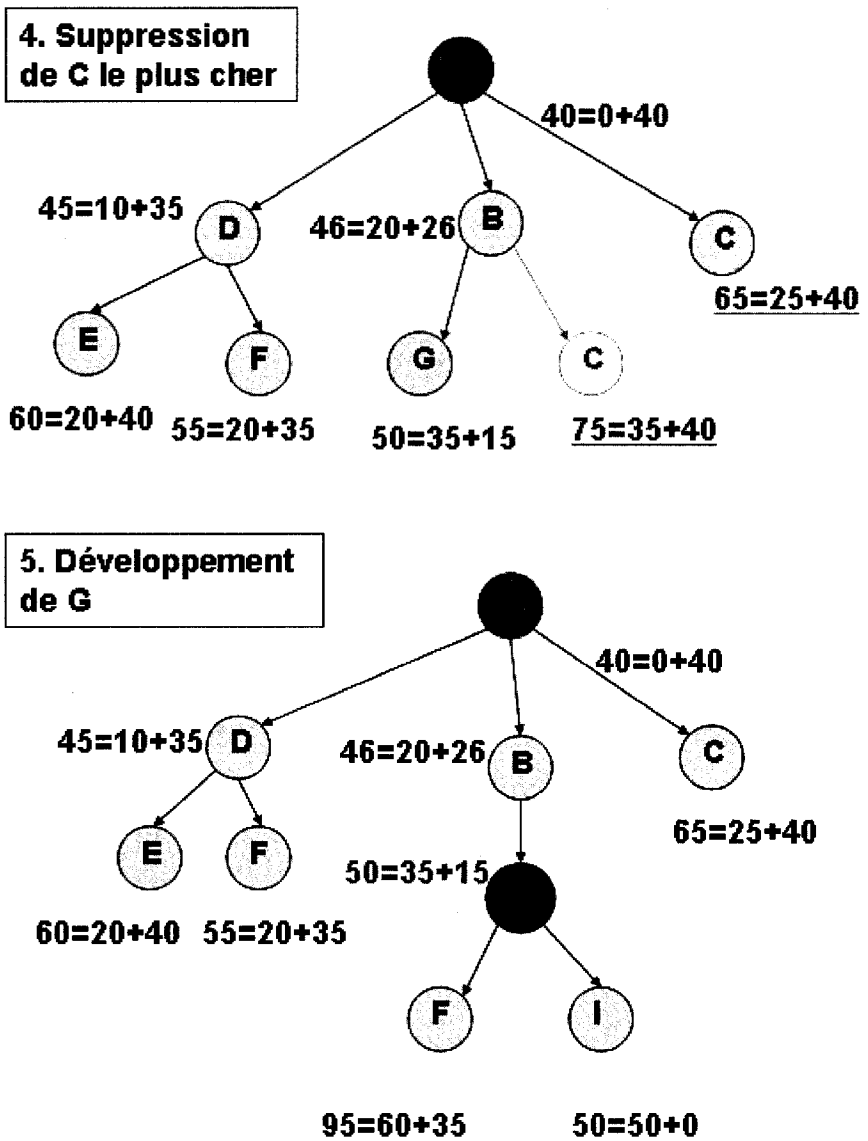


FIG. 2.4 Fonctionnement de l'algorithme A* (2)

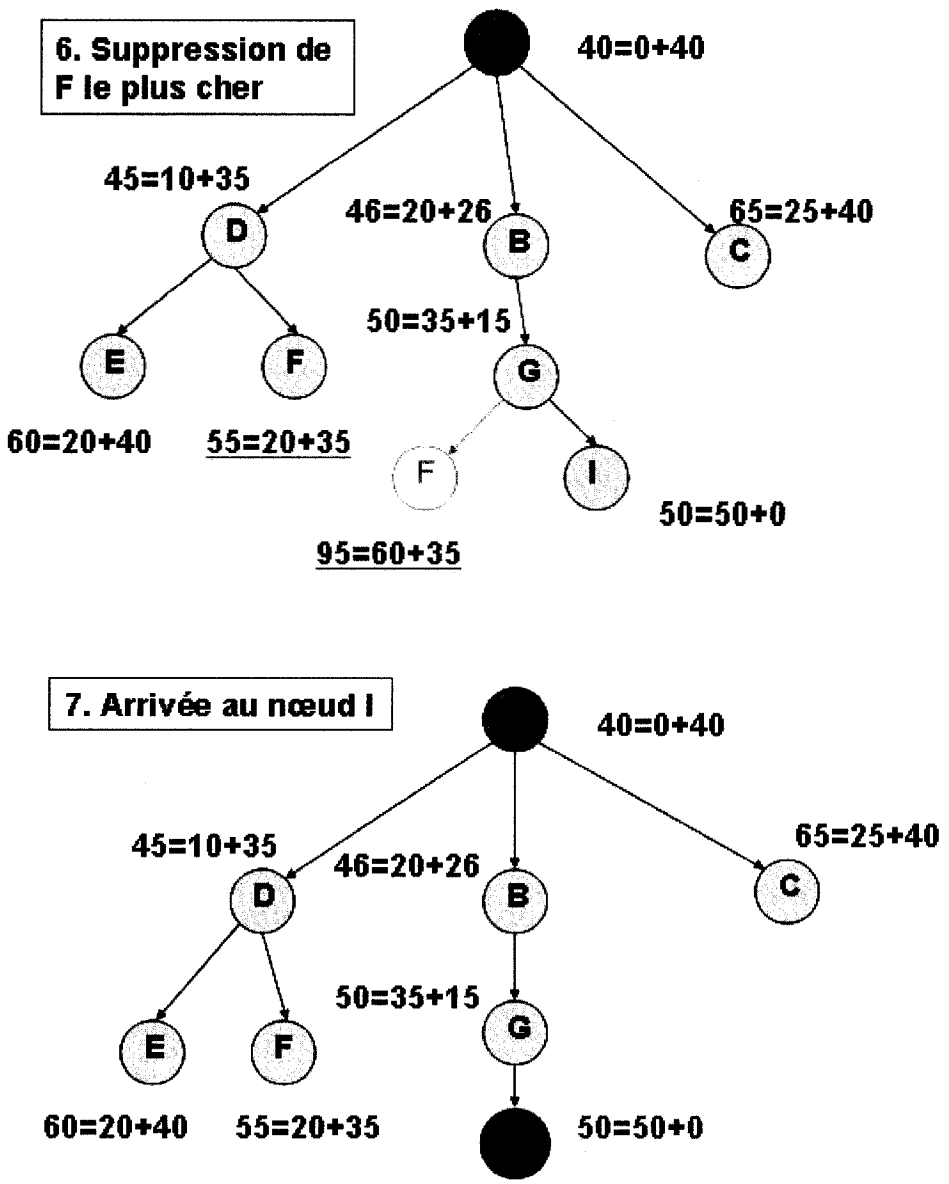


FIG. 2.5 Fonctionnement de l'algorithme A* (3)

graphe, la condition pour que A* soit optimal est que l'heuristique doit être consistante (Russel, 1995). Une heuristique $h(n)$ est consistante si pour tout noeud n_1 et pour tout voisin n_2 de n_1, le coût estimé $h(n_1)$ du chemin entre n_1 et le noeud d'arrivée n'est pas supérieur au coût de l'étape entre $n_1$et n_2 plus le coût estimé $h(n_2)$ du chemin entre n_2 et le noeud d'arrivée :

$$h(n_1) \leq cout(n_1, n_2) + h(n_2)$$

Cette condition peut être respectée si on prend par exemple la distance cartésienne entre le noeud n_1 et le noeud d'arrivée :

$$h(n) = dist(n, n_{final}) = \sqrt{(x - x_{final})^2 + (y - y_{final})^2 + (z - z_{final})^2}$$

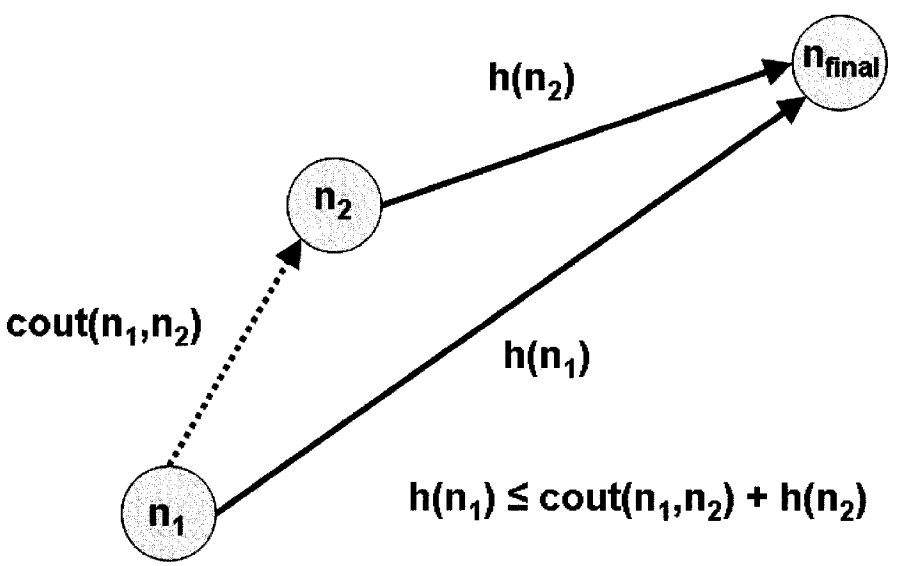


FIG. 2.6 Consistance de l'exploration A*

Dans ce cas, l'inégalité triangulaire impose que $dist(n_1, n_{final}) \leq dist(n_1, n_2) + dist(n_2, n_{final})$, et vu que $cout(n_1 n_2) \geq dist(n_1, n_2)$ notre heuristique respecte bien

la condition de consistance (figure 2.6). Il serait facile de vérifier cela expérimentalement, en effet, la condition de consistance impose que la suite de noeuds développés est dans l'ordre non décroissant des valeurs de $f(n)$ (comme on peut le voir dans les figures 2.3, 2.4 et 2.5). Il en découle aussi que la première fois que l'on voudra développer le noeud final, on aura atteint notre objectif avec le chemin optimal. Comme on peut l'observer dans la figure 2.7, supposons que le noeud I fut déjà parmi les voisins du noeud A. L'algorithme d'exploration ne se serait jamais arrêté à ce moment là vu que l'on dispose encore de noeuds avec des meilleures estimées.

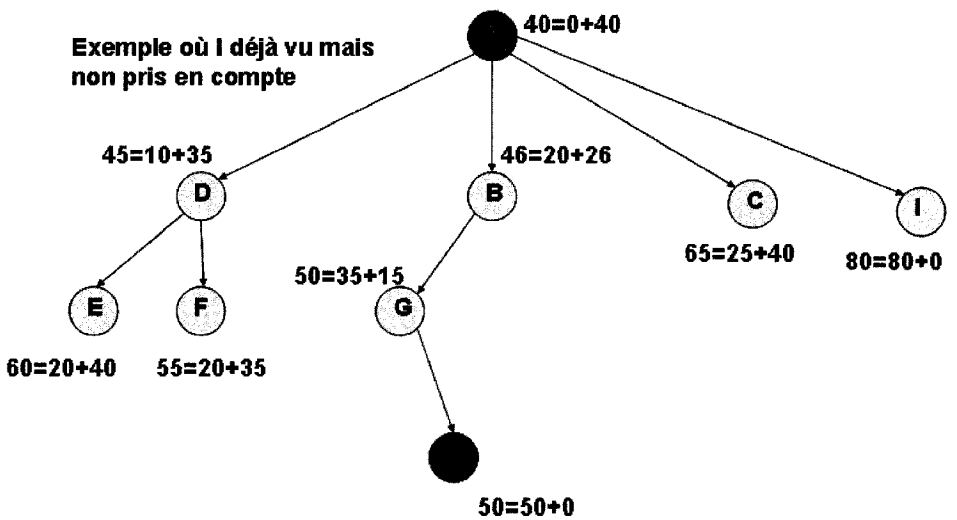


FIG. 2.7 Illustration de l'optimalité de A*

Il nous faut à présent tenir compte d'un graphe particulier sous forme d'atlas. Nous savons que les cartes sont disjointes, il n'existe pas de lien direct entre deux noeuds de cartes différentes. On doit pourtant trouver un moyen pour passer d'une carte à l'autre. Nous allons donc pouvoir utiliser notre quadrillage pour se situer à chaque instant pendant l'exploration. Ainsi, lorsqu'on développe un noeud, il suffit d'observer dans quel case de notre quadrillage il se situe pour ensuite faire une recherche de chaque noeud similaire au noeud courant dans chaque carte disponible à cette case. Il suffit donc de rajouter une fonction que l'on nommera *similaire* qui va

parcourir tous les noeuds présents dans la case et garder pour chaque carte ce noeud que l'on qualifiera de similaire. Un nouveau problème se pose à présent au niveau du classement de ces nouveaux points dans la file pour la stratégie d'exploration. En effet, il n'y a pas de lien directs entre deux noeuds similaires, nous ne pouvons donc pas déterminer de coût pour passer d'un noeud à son noeud similaire. Il faut donc déterminer une nouvelle fonction de coût pour cette situation afin de faire entrer ce noeud similaire dans la file, à une position appropriée.

Une stratégie serait de pénaliser le fait que l'ont vient de changer de carte. Ceci éviterait de basculer de carte en carte plusieurs fois pendant la trajectoire. Ainsi lorsqu'on réalise l'expansion du noeud courant i, on va donc récupérer ces voisins directs indicés j à partir du graphe. Dans un second temps, on recherche les noeuds similaires i' à i. Il faut maintenant les classer dans la file Q de façon à bien choisir le prochain noeud à développer. Les voisins j seront insérés dans la file Q suivant l'estimation que l'on calculera $f(j) = g(j) + h(j)$. En revanche, pour les noeuds similaires i', le calcul de l'estimation va être différent de façon à tenir compte du changement de carte et aussi à cause du fait qu'il n'y a pas de coût entre les noeuds i et i', vu qu'ils ne sont pas voisins mais seulement similaires.

On prendra donc pour estimée, $f(i') = g(i) + k_{penalite} * cout(i, i') + h(i')$, où $k_{penalite}$ est un coefficient qui permet de pénaliser le changement de carte. On peut donc avec cette fonction assimiler également le nouveau coût du chemin entre le noeud de départ i_{dep} et le noeud i' comme étant $g(i') = g(i) + k_{penalite} * cout(i, i')$. La figure 2.8 représente l'algorithme d'exploration du graphe sous la forme d'un logigramme.

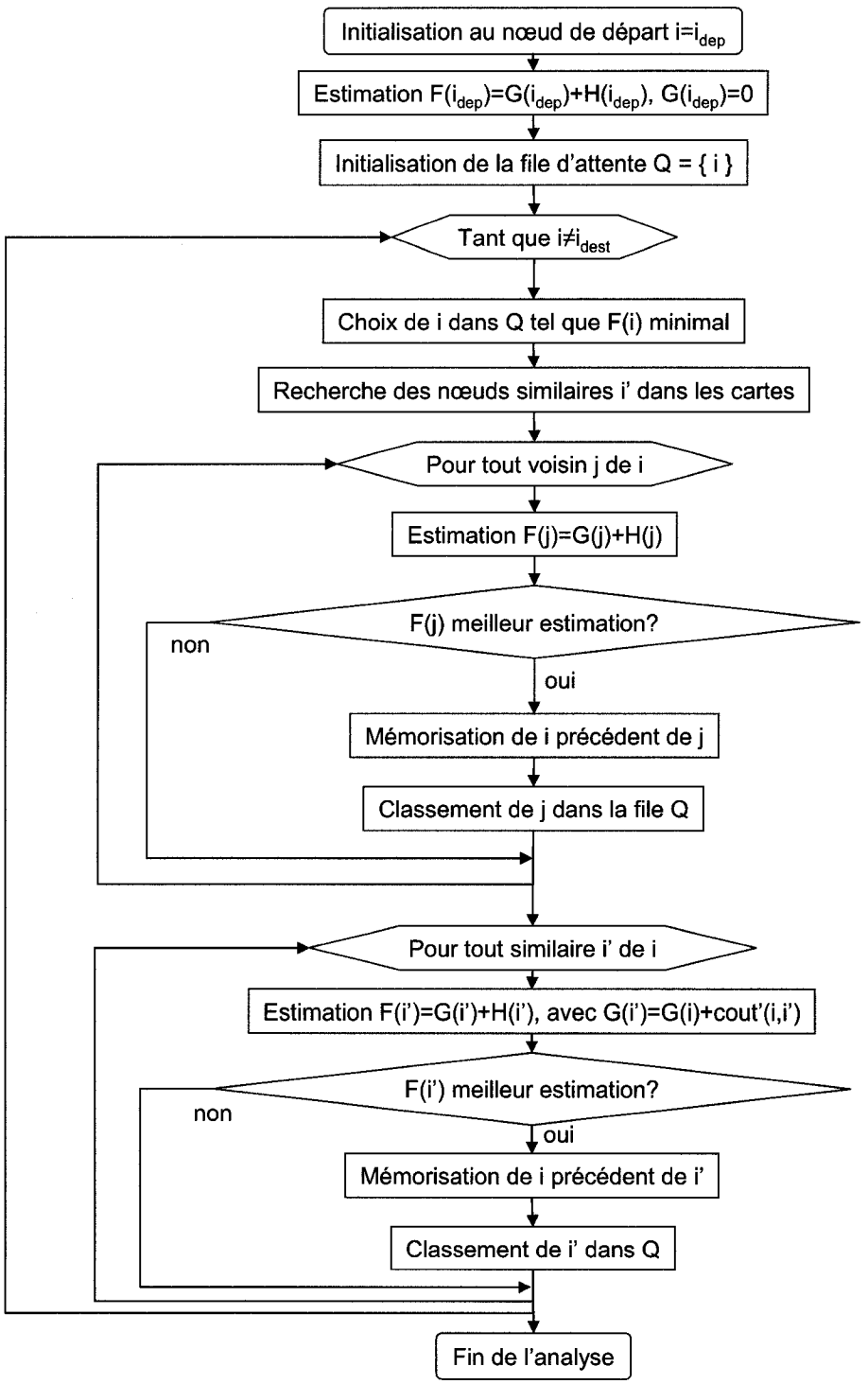


FIG. 2.8 Algorithme d'exploration du graphe

2.5 Expériences

Les tests ont été effectués à partir des trois cartes de relevés à maillage triangulaire irrégulier du robot. Il faut noter que les cartes de départ avaient des origines et des orientations correspondant à celles du robot aux trois moments où les relevés ont été effectués. Il a donc fallu repositionner entre elles ces trois cartes. Ceci a été possible à partir de la carte fusionnée fournie par l'ASC. Il a suffit de comparer quelques points judicieusement choisis dans chaque carte pour calculer les matrices de transformation de deux cartes par rapport à la troisième. Ainsi on a pu construire un graphe de plus de 29000 noeuds à partir des trois cartes à maillage triangulaire irrégulier fournis par l'ASC. L'algorithme de mise à jour met environ 4 minutes pour créer le graphe d'une carte. L'espace mémoire utilisé par le graphe est de 2.4 Mo pour les 3 cartes. Ce temps est relativement long mais une fois que le graphe est créé, il est définitif. Il n'est pas nécessaire de recréer le graphe en entier dés que l'on rajoute une nouvelle carte pendant l'exploration. La complexité de l'algorithme de création de graphe suit une loi en $o(n)$ où n sera le nombre de triangles dans la carte. Le temps de calcul sera donc une fonction linéaire du nombre de noeuds à mémoriser dans le graphe. Ainsi, en réalisant un essai sur la carte globale haute résolution, on obtient un graphe de la carte de 89700 triangles en 45 minutes ce qui confirme le comportement linéaire du temps de calcul. Ce temps peut sembler long à première vue, mais il faut noter qu'il correspond à un graphe de test pour une carte comportant prés de 90000 triangles. Les cartes devraient normalement comporter moins de 10000 triangles si on regarde les trois cartes à maillage triangulaire irrégulier, soit des temps de calculs de moins de 5 minutes.

Une fois ces graphes créés on peut à présent rechercher une trajectoire avec l'algorithme d'exploration. Les figures 2.9 et 2.10 représentent une trajectoire de 100 noeuds calculée en 60s en ayant visité environ 5800 noeuds. Le temps de calcul va

dépendre linéairement du nombre de noeuds visités. Ce nombre de noeuds visités peut varier selon la difficulté qu'aura l'algorithme à trouver une trajectoire en ce dirigeant vers son but. Pour illustrer ceci, on peut tenter de créer une trajectoire tentant de contourner un gros obstacle infranchissable (par exemple la colline sur le graphe de la carte haute résolution réalisé précédemment). On peut observer un tel essai sur les figures 2.11 et 2.12. Dans ce cas-ci, l'algorithme a dû visiter 50766 noeuds en 35 minutes. Ce temps de calcul peut sembler long, mais ce n'est qu'une illustration dans un cas extrême. On pourra utiliser les cartes de plus basse résolution pour accélérer ce temps de calcul. En effet, à partir d'une certaine distance du robot, il n'est pas encore nécessaire d'avoir la connaissance d'une trajectoire précise. Il suffira de savoir qu'il faut contourner l'obstacle et donc se diriger en conséquence,. La trajectoire exacte vers la destination se précisera au fur et à mesure de l'avancement du robot.

Concernant la recherche de trajectoire sur plusieurs cartes (figures 2.9 et 2.10), on peut déjà observer comment s'effectue le changement de carte. Ces premiers résultats semblent concluant. Il va falloir à présent renouveler d'autres essais et essayer d'agrandir l'espace de recherche, ainsi que la trajectoire à planifier pour analyser les performances de la planification.

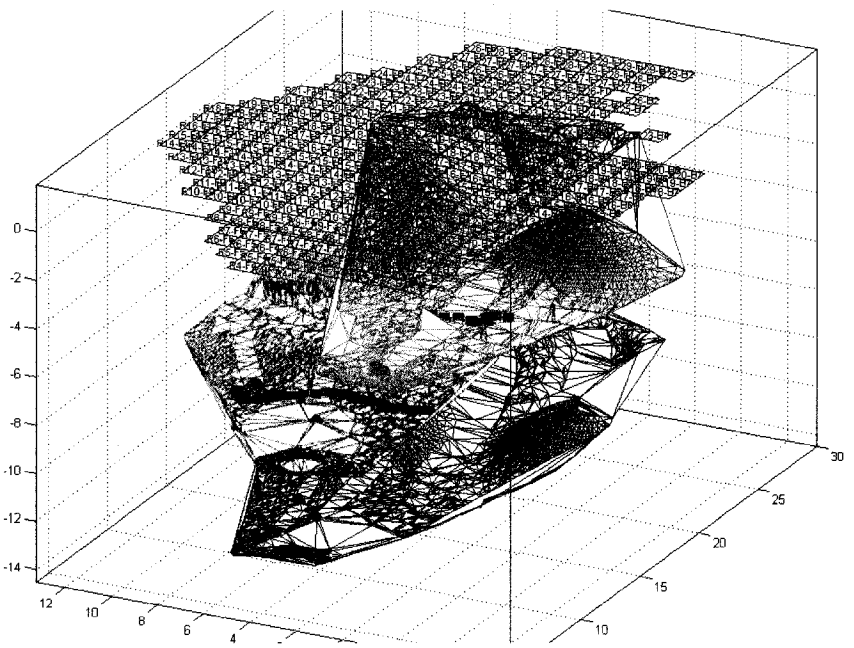


FIG. 2.9 Illustration d'une trajectoire sur deux cartes(1)

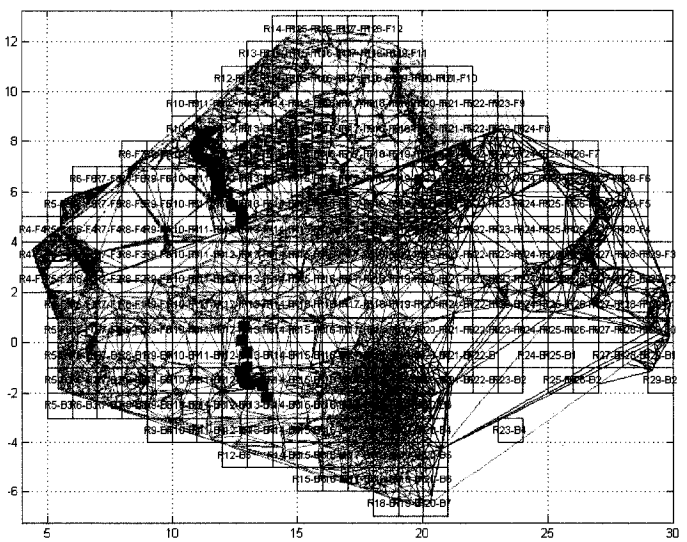


FIG. 2.10 Illustration d'une trajectoire sur deux cartes (2)

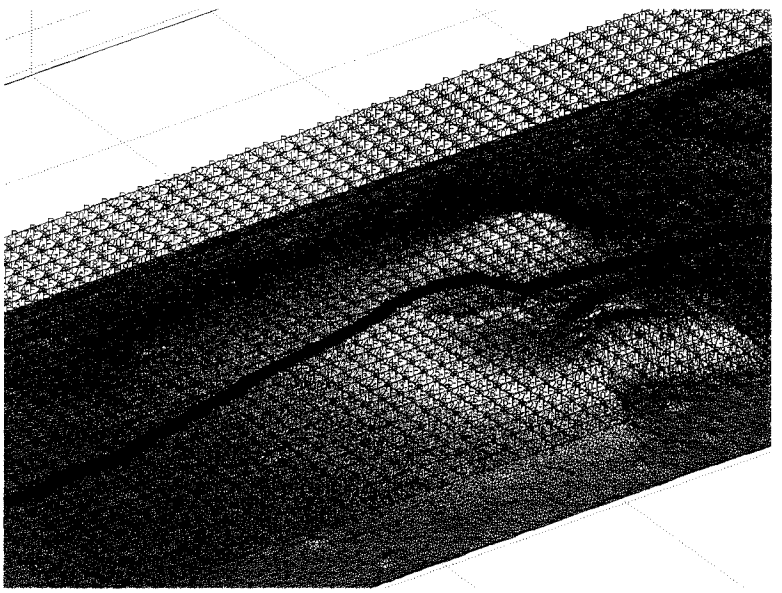


FIG. 2.11 Essai de calcul d'une trajectoire de contournement sur une longue distance

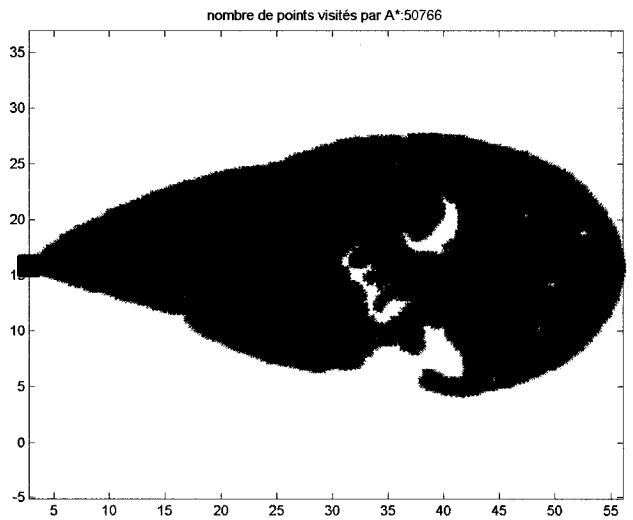


FIG. 2.12 Exemple de zone balayée par A* dans le cas d'un contournement sur une longue distance

2.6 Optimisation

Les algorithmes de base ont passé les premiers tests. Il reste de nombreux essais à réaliser pour analyser leur comportement dans différentes situations. La partie suivante concerne l'implémentation d'un outil de simulation qui nous permettra d'ajuster, voire de modifier, les fonctions de coût afin de respecter au mieux un cahiers des charges pour des trajectoires optimales et sécuritaires. Là encore les premiers résultats semblent prometteurs, la figure 2.13 illustre une trajectoire contournant une roche.

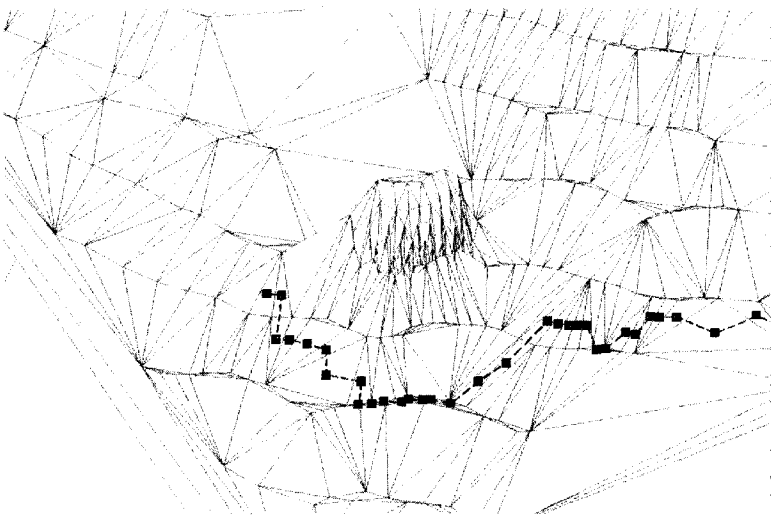


FIG. 2.13 Contournement d'un obstacle

Différentes valeurs sur $k_{penalite}$ pénalisant le changement de carte peut donner des résultats totalement différents (figure 2.14).

L'optimisation ne s'arrête pas seulement à l'allure de la trajectoire. Il faut penser également à l'optimisation au niveau de la puissance de calcul ainsi que la quantité de mémoire utilisée, si on veut penser à une application concrète dans un *rover* martien. Le matériel informatique embarqué et ses performances sont bien entendu limités.

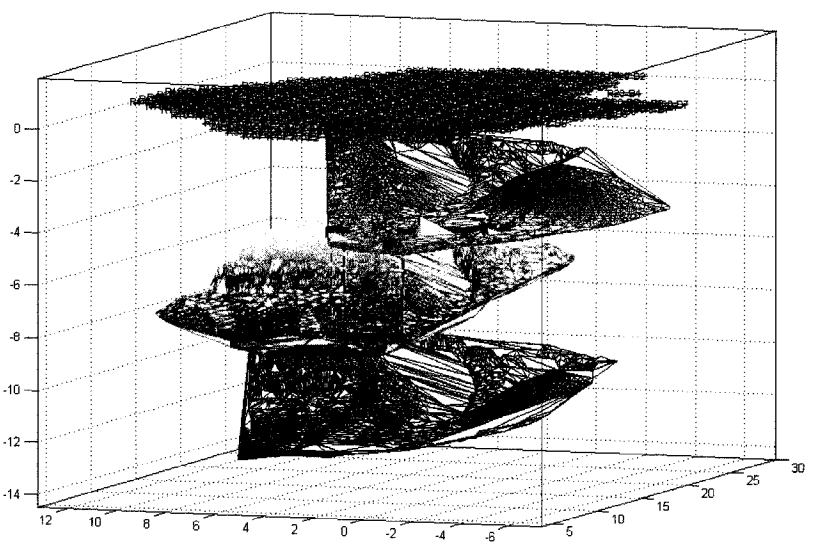


FIG. 2.14 Changement de carte insuffisamment pénalisé

CHAPITRE 3

SIMULATION

Nous avons à présent à notre disposition les principes de nos algorithmes ainsi qu'une première version. Nous avons pu déjà effectuer quelques essais. Cela n'est bien entendu pas suffisant pour prouver le bon fonctionnement de ceux-ci, et également quantifier leur performance. Cette partie va être consacrée à la réalisation plus poussée d'une exploration typique d'un territoire partiellement connu. Afin d'intégrer nos algorithmes de création du graphe et de planification de trajectoire, il va falloir simuler les autres étapes de la figure 2.1. Différents modules vont devoir être ajoutés afin de compléter le cycle de l'exploration. La première fonction à ajouter est l'acquisition de carte. On doit pouvoir simuler la prise de données de l'environnement du robot, à savoir l'acquisition LIDAR du nuage de points suivi d'une triangulation et d'une décimation en maillage triangulaire irrégulier. Le second module va être une Intelligence Artificielle minimale permettant de prendre des décisions telles que le choix des noeuds à atteindre, le choix des objectifs, la nécessité de faire une acquisition de l'environnement, ou encore observer et analyser les positions le long de la trajectoire planifiée. Une fois ces sous-systèmes réalisés, il va suivre une phase d'intégration pour former un système unique. Pour cela certaines adaptations ou corrections vont être réalisées : dans un premier temps, pour des soucis de compatibilité entre sous-systèmes, et enfin dans un second temps, dans un souci d'amélioration de performances.

3.1 Acquisition de cartes

Nous allons tout d'abord définir ce que l'on attend exactement de ce module. Cette étude est entièrement indépendante de la planification de trajectoire, mais elle est tout de même indispensable à sa réalisation. La simulation de l'exploration peut amener à n'importe quelle position sur notre terrain. Ce module doit donc être capable de créer une carte représentant au mieux le terrain à partir du point de vue du robot. Différents aspects seraient très intéressants à simuler pour observer le bon fonctionnement de notre planificateur de trajectoire. Un premier problème auquel on est confronté, lorsqu'on observe son environnement, est la visibilité. De nombreux obstacles masquent le champ de vision de notre robot. Il va donc falloir simuler le balayage du LIDAR dans une direction donnée, à une position donnée, et tenir compte des obstacles introduisant des zones d'ombres dans notre champ de vision. Un second aspect pouvant être simulé est l'erreur d'orientation et de position du robot, ainsi que le bruit de mesure du LIDAR. Cependant, cet aspect est intéressant uniquement pour observer le comportement du planificateur lorsque les données peuvent avoir une composante aléatoire. On ne peut malheureusement pas vérifier la robustesse de l'exploration puisque cela dépend fortement des systèmes de guidage, de déplacement et de localisation du robot. Enfin, un aspect indispensable à simuler dans l'acquisition d'une nouvelle carte est la décimation. En effet, la décimation introduit des triangles de tailles et de formes très variables, ce qui va donc naturellement influer sur les performances de notre planificateur. Certaines modifications sont donc à prévoir même si les premiers essais ont pu être concluants.

FIG. 3.1 Acquisition LIDAR réelle en (48.5,9) visualisée avec Paraview

FIG. 3.2 Acquisition LIDAR par simulation en (48.5,9) visualisée avec Matlab

3.1.1 Récupération des points

A partir d'une certaine position et d'une certaine orientation du robot sur notre carte, nous souhaitons obtenir un nuage des points visibles depuis ce point de vue. De nombreuses recherches ont déjà été effectuées dans le domaine de la visibilité. A la base, ces recherches sont motivées par la représentation 3D qui demande beaucoup de ressources et qui nécessite donc la simplification des objets 3D, mais également pour leur rendu graphique à l'écran. Un des algorithmes très répandus est le *z-buffer,* utilisé par les cartes graphiques pour déterminer quels objets, ou parties d'objets doivent être rendus à l'écran. L'article *Visibility and Dead-Zones in Digital*

Terrain Maps de *D.Cohen-Or* et *A.Shaked (Cohen-Or, 1995)* détaille les principes de la visibilité, dans le cas particulier des représentations 3D de terrains. On peut en tirer quelques avantages, en utilisant les particularités de ce type de données, comme leur régularité dans la répartition des points, ainsi que leur représentation relativement simple.

3.1.1.1 Principe

Pour identifier les points visibles et invisibles, il faut tout d'abord se placer dans une coupe transversale de notre terrain et contenant le point de vue (v_{eye}). Il faut ensuite pouvoir analyser dans cette coupe quels points sont visibles. Un point est visible s'il existe une ligne droite entre ce point et le point de vue, et ne traversant pas le profil du terrain (figure 3.3).

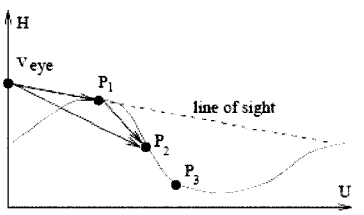


FIG. 3.3 Principe de la visibilité d'un point (Cohen-Or, 1995)

Il faut pouvoir maintenant calculer cette visibilité dans notre cas où le terrain est sous forme de maillage régulier. Considérons tout d'abord que l'on peut extraire une coupe transversale de notre terrain dans n'importe quelle direction. *D.Cohen-Or et al.* (Cohen-Or, 1995) décrit un algorithme incrémental permettant de calculer la visibilité d'un point. On parcourt les points du profil transversal à partir de v_{eye}. Un point P_i est dit visible si, à partir du vecteur défini par v_{eye} et par le dernier point visible P_v, on doit tourner dans le sens inverse des aiguilles d'une montre pour atteindre l'orientation du vecteur défini par v_{eye} et ce point P_i. Cela revient

à calculer le signe du produit vectoriel entre ces deux vecteurs.

$$visibilite(P_i) = \begin{cases} 1 \text{ si } sign[(P_v - v_{eye}) \wedge (P_i - v_{eye})] > 0 \\ 0 \text{ si } sign[(P_v - v_{eye}) \wedge (P_i - v_{eye})] < 0 \end{cases}$$

Il faut tout de même remarquer que l'on effectue une approximation par le fait que nos données soient discrètes. En effet toute coupe transversale de notre terrain suppose que les points du profil sont exactement dans le plan de coupe. Malheureusement, la nature discrète de ces points introduit un décalage évident entre le plan de coupe et la position réelle du point qui va dépendre de la résolution de la carte.

Pour réduire cette erreur due à l'échantillonnage, il convient de rechercher les points qui représenteront au mieux un plan de coupe donné. Cela revient à tracer un segment formé de pixels traversant la carte. Il existe plusieurs algorithmes permettant de tracer de tels segments dits *digitalisés,* et également différents types de segments digitalisés selon la connectivité entre pixels voisins (figure 3.4).

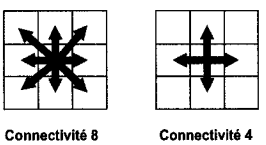


FIG. 3.4 Illustration de la connectivité

3.1.1.2 Implémentation

Il nous reste donc à choisir un algorithme permettant dans un premier temps de tracer un segment dans le plan (x, y) selon n'importe quelle orientation et passant par le point de vue v_{eye}. Nous verrons dans cette partie une description de l'algo-

rithme trés répandu de *Bresenham* (*http ://fr.wikipedia.org*). Pour pouvoir utiliser cet algorithme, il faut auparavant modifier la structure de notre carte. Celle-ci est à la base sous forme d'une liste, il faut donc auparavant représenter notre carte sous la forme d'une matrice $Z(i,j)$ représentant les valeurs de z selon les indices i et j tels que :

$$Z(i,j) = \begin{bmatrix} z(x_{min}, y_{min}) & \cdots & z(x_{min}, y_{max}) \\ & \ddots & \\ \vdots & z\left(\frac{i-1}{res} + x_{min}, \frac{j-1}{res} + y_{min}\right) & \vdots \\ & & \ddots \\ z(x_{max}, y_{min}) & \cdots & z(x_{max}, y_{max}) \end{bmatrix}$$

On considère tout d'abord que le segment à tracer a une pente inférieur à 1 en valeur absolue dans le plan (x,y). Il faut ensuite partir du point de départ correspondant à v_{eye}. On incrémente alors la valeur de l'abcisse i de 1 dans le sens des x, puis on vérifie à chaque itération s'il vaut mieux rester sur la même valeur j ou si on doit également l'incrémenter (ou décrementer dans le cas d'une pente négative). Pour savoir quelle valeur de j il faut choisir, nous utiliserons tout simplement l'arrondi de la valeur $y(x)$ définissant le segment à l'abcisse i (figure 3.5).

On est cependant restreint à un partie du plan, il faut donc généraliser notre algorithme. Pour cela, il suffit d'inverser les coordonnées et adapter le sens de l'incrémentation selon différentes valeurs de la pente α définissant l'orientation du segment à tracer (figure 3.6).

1. $\alpha = 0$
2. $\alpha = \pi$
3. $\alpha = \frac{\pi}{2}$

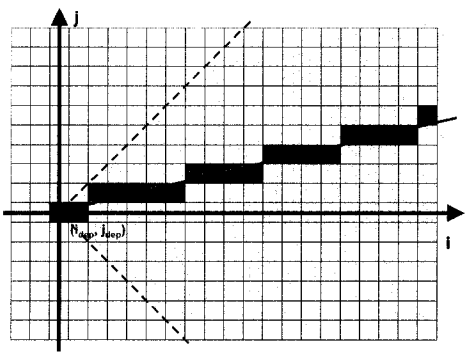


FIG. 3.5 Segment digitalisé

4. $\alpha = -\frac{\pi}{2}$
5. $-\frac{\pi}{4} < \alpha \leq \frac{\pi}{4}$ et $\alpha \neq 0$
6. $\frac{\pi}{4} < \alpha \leq \frac{3\pi}{4}$ et $\alpha \neq \frac{\pi}{2}$
7. $\frac{3\pi}{4} < \alpha \leq \frac{5\pi}{4}$ et $\alpha \neq \pi$
8. $\frac{5\pi}{4} < \alpha \leq \frac{7\pi}{4}$ et $\alpha \neq \frac{3\pi}{2}$

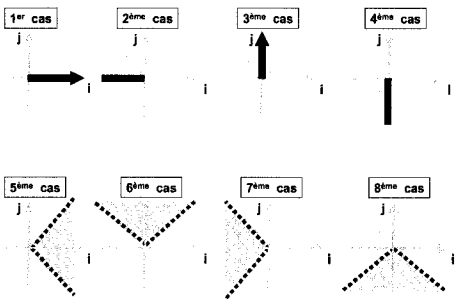


FIG. 3.6 Cas à différencier pour implémenter l'algorithme de Bresenham

Nous savons à présent tracer un segment dans le plan (x, y). Il faut maintenant effectuer le balayage autour du point de vue. Pour cela, il suffit de déplacer l'extrémité de notre segment située au bord de notre carte selon l'amplitude désirée, ainsi tous les points situés dans le champ de vision seront observés au moins une fois

(figure 3.7). Il suffira alors de le détecter comme visible une fois pendant le balayage pour le considérer comme visible (même s'il est détecté comme invisible dans un autre plan de coupe). Pour cela il suffit d'initialiser une matrice de visibilité des points à 0, puis de la compléter au fur et à mesure du balayage.

$$\text{initialisation} \quad : \quad Visibilite = \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix}$$

pour chaque point :

$$Visibilite(i,j) = \begin{cases} 1 & \text{si } sign[(P_v - v_{eye}) \wedge (P_i - v_{eye})] > 0 \\ 0 & \text{si } sign[(P_v - v_{eye}) \wedge (P_i - v_{eye})] < 0 \text{ et } Visibilite(i,j) = 0 \end{cases}$$

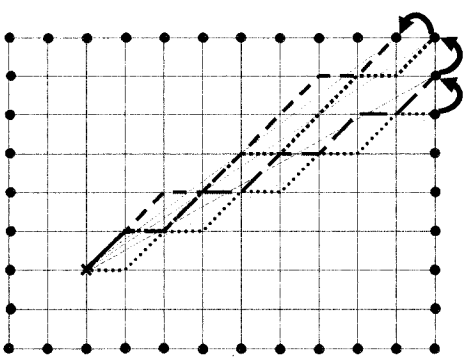


FIG. 3.7 Principe du balayage

On peut donc effectuer un balayage devant le robot permettant d'enregistrer les points visibles sous forme d'une matrice semblable à $Z(i,j)$ vue précédemment. Il reste alors à réaliser la triangulation des points visibles à l'aide de la fonction Matlab *delaunay()*. Cette fonction va cependant trianguler également les zones d'ombre, il faut donc les supprimer par la suite. On peut les détecter facilement puisque dans notre cas, ce sont tous les triangles ayant une aire de plus de $0,02m^2$.

3.1.2 Décimation du maillage triangulaire

La décimation consiste à diminuer la quantité de données nécessaires pour représenter un objet. Elle peut s'appliquer aussi bien sur des maillages rectangulaires, polyhédraux, *quadtree,* et dans notre cas, triangulaires. Nous allons décrire les principales notions utiles à la compréhension de la décimation. Puis nous verrons sa mise en oeuvre concrète dans notre cas, avec les simplifications apportées pour satisfaire au mieux nos exigences. Le but principal de cette partie est de pouvoir créer un Maillage Triangulaire Irrégulier semblable à ceux réalisés par l'ASC dans le cadre de leur recherche (Dupuis, 2005). Nous ne tenterons pas ici d'obtenir un algorithme optimal. Il existe en effet plusieurs critères permettant de définir la décimation. Différents critères peuvent agir sur la rapidité de calcul, sur la quantité de triangles décimés et également sur la qualité (fidélité entre le modèle original et décimé, cette notion est difficilement quantifiable puisque cela dépend du jugement de l'utilisateur sur le rendu visuel).

3.1.2.1 Principe

Il existe de nombreux algorithmes permettant de réaliser une décimation. On pourra se référer à l'article de *P.Cignoni et al. (Cignoni, 1998)* pour une comparaison de quelques algorithmes généralement utilisés dans la littérature. En ce qui concerne l'ASC, ils utilisent les librairies VTK (*http ://www.vtk.org*) déjà disponibles pour ce genre de traitement. Cette librairie étant difficilement adaptable pour Matlab, il est préférable et beaucoup plus intéressant d'implémenter notre propre outil de décimation. Nous allons décrire un peu plus en détail l'algorithme utilisé par *William J. Schroeder et al.* (Schroeder, 1992).

La décimation se déroule en trois phases, tout d'abord, il faut classifier chaque

point du nuage initial, puis calculer les critères de décimation et dans une dernière étape, il faut retrianguler l'ensemble des points restants.

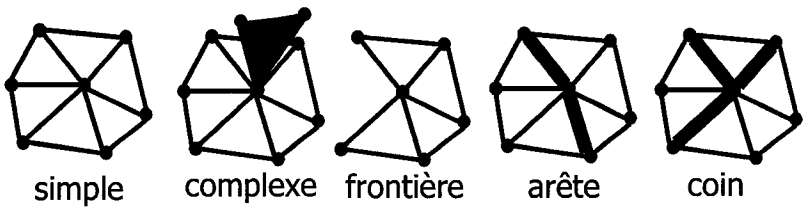


FIG. 3.8 Classification des sommets

La figure 3.8 représente les types de sommets que l'on peut rencontrer dans notre maillage triangulaire. Les sommets simples sont entièrement entourés par des triangles, et chaque arête est utilisée exactement par deux triangles. Dans le cas contraire, le sommet est dit complexe. Dans ces sommets complexes, on peut encore différencier des sommets représentant une frontière, des crêtes, ou des coins. Les sommets frontière sont ceux qui possèdent un cycle incomplet de triangles. Un sommet de crête possède deux arêtes formant une crête, c'est-à-dire qu'il compose la forme de l'objet. Il faut les considérer différemment, au risque d'altérer l'aspect visuel. Losqu'un sommet possède trois arêtes de ce type, il représente un coin. De telles arêtes caractéristiques sont calculées à partir de l'angle dièdre entre les deux triangles voisins. Cet angle dièdre est en fait l'angle entre chaque vecteur normal des deux triangles. Pour cela on doit tout d'abord calculer le vecteur normal du triangle (figure 3.9).

$$\overrightarrow{N_i} = \overrightarrow{P_1P_2} \wedge \overrightarrow{P_1P_3}$$

On souhaite cependant avoir un vecteur normal unitaire avec la même orientation pour tous les triangles. On choisit une orientation dirigée vers le haut. Cela est possible, puisque notre carte est en fait en 2.5D, c'est à dire que l'on est effectivement en 3D mais on ne peut pas avoir de faces orientées vers le bas comme un tunnel par exemple. Ainsi, si le signe de $\overrightarrow{N_{iz}}$ est négatif (le vecteur normal est dirigé vers

le bas), on multiplie par -1 pour obtenir le vecteur $\overrightarrow{n_i}$ dirigé vers le haut.

$$\overrightarrow{n_i} = sign(N_{iz}) \frac{\overrightarrow{N_i}}{\left\| \overrightarrow{N_i} \right\|}$$

Il nous reste ensuite, pour obtenir l'angle dièdre, à calculer le produit scalaire, et à extraire l'angle $\Phi_{ij} = acos(n \cdot n_j)$.

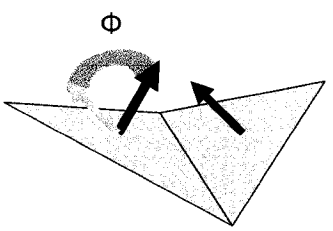


FIG. 3.9 Angle dièdre

Une fois que nous avons caractérisé chaque sommet, il nous faut calculer les critères de suppression. Il en existe de nombreux, et les méthodes de calcul peuvent différencier dans la littérature. Nous allons nous concentrer sur deux critères qui suffiront amplement dans notre situation. Le premier critère est la distance au plan moyen (figure 3.10). On pourrait également calculer un plan au sens des moindres carrés tel que décrit dans l'article de *M.Knapp (Knapp, 2002)* au sujet des outils utilisés par la librairie VTK.

A partir des vecteurs normaux calculés précédemment $\overrightarrow{n_i}$ et des barycentres des triangles $\overrightarrow{x_i}$ on peut obtenir le vecteur normal définissant le plan moyen des triangles entourant un sommet $\overrightarrow{N}$ avec son origine $\overrightarrow{x}$.

$$\begin{aligned} \overrightarrow{N} &= \frac{\sum \overrightarrow{n_i} A_i}{\sum A_i} \text{ et } \overrightarrow{n} = \frac{\overrightarrow{N}}{\left\| \overrightarrow{N} \right\|} \\ \overrightarrow{x} &= \frac{\sum \overrightarrow{x_i} A_i}{\sum A_i} \end{aligned}$$

On cherche à supprimer les sommets faisant partie du même plan formé par les triangles connexes. Il reste à définir une mesure permettant de quantifier cela. On va donc calculer la distance de chaque sommet par rapport au plan moyen associé. Elle correspond au produit scalaire du vecteur partant de l'origine du plan et allant jusqu'au sommet avec le vecteur normal.

$$d = |\vec{n} \cdot (\vec{v} - \vec{x})|$$

On choisira tout simplement de supprimer un sommet si on a $d < d_{max}$ où d_{max} est un critère arbitraire permettant de conserver au mieux la structure du maillage triangulaire et diminuant suffisamment le nombre de sommets.

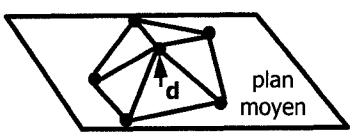


FIG. 3.10 Distance au plan moyen (Schroeder, 1992)

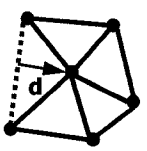


FIG. 3.11 Distance à une frontière (Schroeder, 1992)

On peut également chercher à supprimer un sommet situé sur une frontière ou une crête et où les deux arêtes sont quasiment alignées, ceci peut être mesuré par la distance entre ce sommet et la nouvelle arête éventuellement créée par sa suppression. Soient P_1et P_2 les deux sommets voisins du sommet P que l'on désire

supprimer, où les arêtes (P, P_1) et (P, P_2) forment la frontière ou la crête. Soit θ l'angle formé par les vecteurs $\overrightarrow{P_1P}$ et $\overrightarrow{P_1P_2}$.

$$\begin{aligned} \sin(\theta) &= \frac{d}{\left\|\overrightarrow{P_1P}\right\|} \\ \overrightarrow{P_1P} \wedge \overrightarrow{P_1P_2} &= \left\|\overrightarrow{P_1P}\right\| \cdot \left\|\overrightarrow{P_1P_2}\right\| \cdot \sin(\theta) \\ \text{d'où } d &= \frac{\overrightarrow{P_1P} \wedge \overrightarrow{P_1P_2}}{\left\|\overrightarrow{P_1P_2}\right\|} \end{aligned}$$

On compare ensuite de la même manière d à un d_{max} choisi arbitrairement.

On pourrait ajouter encore d'autres critères de suppression voire de conservation, cependant on considèrera ces deux critères suffisants pour réduire la quantité de points dans notre maillage triangulaire. On rappelle que le but de la décimation est d'analyser le comportement de nos algorithmes vis à vis d'un Maillage Triangulaire Irrégulier. Il n'a pas été jugé nécessaire de pousser l'analyse plus profondément pour améliorer la décimation. Les critères d_{max} ont été choisis afin de pouvoir diviser par deux environ la quantité de triangles dans notre maillage.

Il faut noter que dans la littérature, la triangulation se déroule habituellement en ligne, c'est à dire qu'on retriangule un trou laissé immédiatement après avoir supprimé un sommet. Afin de simplifier les algorithmes, il a été préférable encore une fois d'utiliser la fonction *delaunay()* de Matlab. Celle-ci est appliquée lorsque l'on a supprimé tous les sommets concernés. La fonction *delaunay()* comble en revanche les trous situés dans les zones d'ombres. Pour cela, il suffira de supprimer tous les triangles formés par trois sommets classés auparavant comme « situés sur une frontière ».

3.1.2.2 Algorithme (figure 3.12)

A la base, nous disposons d'une matrice contenant toutes les coordonnées de chaque point, ainsi qu'une seconde liste décrivant chaque triangle formé par trois points. Cependant pour effectuer la décimation, nous avons besoin de connaitre pour chaque sommet, quels triangles sont connectés à ce point. Une recherche en ligne pour chaque point de ses occurrences dans la liste des triangles possèderait une complexité en $o(n^2)$. Il faut donc parcourir la liste des triangles afin de créer une structure mémorisant les triangles disponibles pour un sommet donné. Il est également nécessaire de connaitre la normale et le barycentre des triangles, ceci peut donc être calculé en prétraitement lors du parcours précédent de la liste des triangles pour accélérer la seconde boucle de l'évaluation des critères de suppression des sommets. Au final on obtient une complexité en $o(n)$.

3.1.3 Ajout du bruit et de l'erreur d'orientation

Cette partie a pour but d'introduire des phénomènes présents dans une éventuelle expérimentation pratique. Les principales erreurs dans les données sont les bruits de mesure du LIDAR qui vont introduire des faibles décalages dans le nuage de points. Chaque erreur commise sur chaque point de mesure est indépendante des autres. L'erreur la plus importante va être celle introduite par les systèmes de guidage et de localisation. Lorsqu'on fait l'acquisition d'une nouvelle carte, le système de localisation va chercher à positionner cette nouvelle carte par rapport à celles déjà mémorisées. Ceci permettra également de mettre à jour l'estimation de la position du robot par le système de guidage lors du déplacement. Un algorithme tel que *l'ICP* (*Iterative Closest Point*) sert à positionner deux nuages de points en effectuant différentes transformations, translations et rotations successivement, jusqu'à trouver une erreur minimale. Cela nécessite un positionnement initial déjà

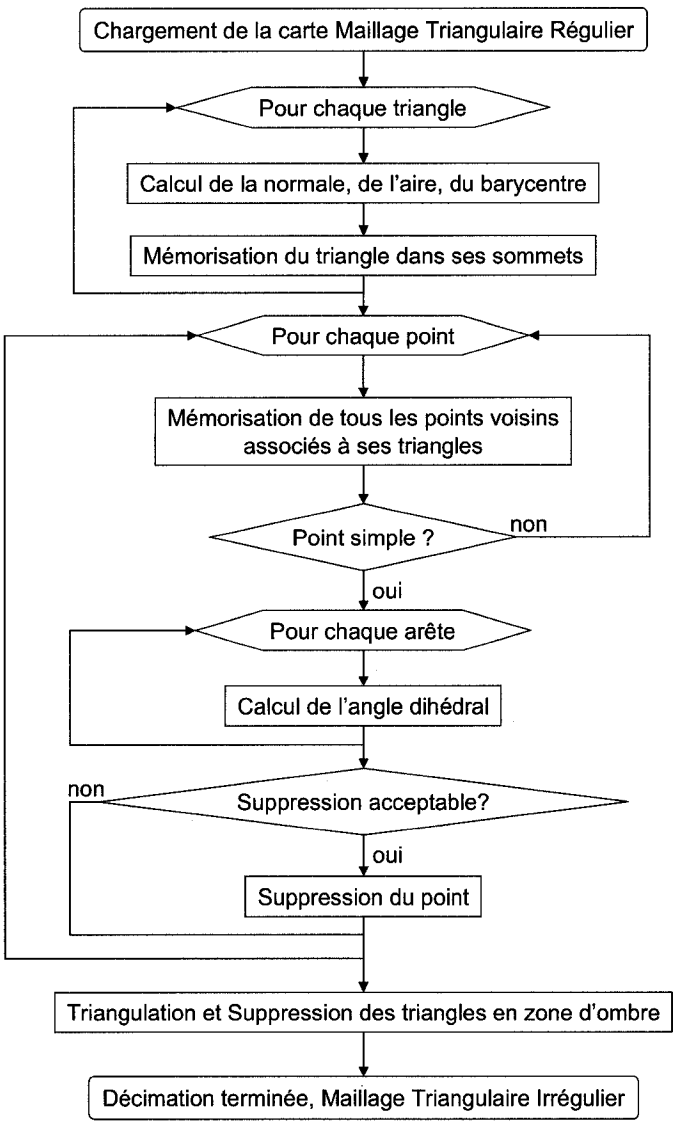


FIG. 3.12 Décimation en maillage triangulaire irrégulier

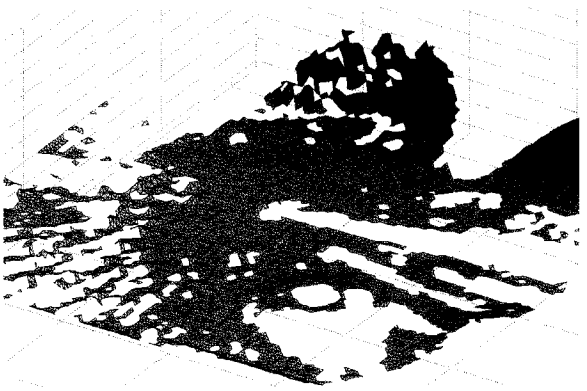


FIG. 3.13 Acquisition avant décimation

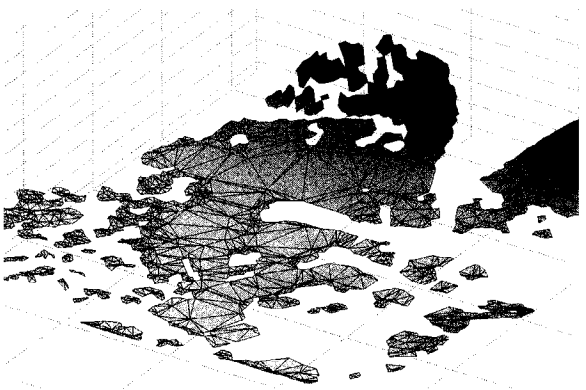


FIG. 3.14 Décimation

suffisamment proche en général, afin d'accélérer le processus et de trouver la solution optimale. Or, avoir un positionnement initial correct impose un bon système de guidage et un robot possédant des performances de franchissement limitant le glissement au maximum. L'erreur de positionnement initial combinée aux erreurs de mesure posent des problèmes pour superposer les cartes entre elles. On va donc introduire un faible bruit gaussien sur les coordonnées des points enregistrés après balayage puis on crée une matrice de transformation qui va modifier l'orientation et la position du nuage de points dans l'espace.

Si on considère des erreurs suffisamment petites d'ordre 1, nous pourrons simplifier la matrice de transformation en utilisant des mouvements incrémentaux. Supposons des décalages en translation d_x, d_y, d_z et en rotation δ_x, δ_y, δ_z. Suivant chaque axe, on aura donc $\cos(\delta) \approx 1$ et $\sin(\delta) \approx \delta$.

$$ROT(x, \delta_x) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -\delta_x & 0 \\ 0 & \delta_x & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$ROT(y, \delta_y) = \begin{bmatrix} 1 & 0 & \delta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\delta_y & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$ROT(z, \delta_z) = \begin{bmatrix} 1 & -\delta_z & 0 & 0 \\ \delta_z & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$TRANS(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

En négligeant les termes d'ordre 2, une transformation autour du centre du point de vue $(x_{eye}, y_{eye}, z_{eye})$ aura pour matrice T telle que :

$$T = \begin{bmatrix} 1 & -\delta_z & \delta_y & \delta_z y_{eye} - \delta_y z_{eye} + d_x \\ \delta_z & 1 & -\delta_x & -\delta_z x_{eye} + \delta_x z_{eye} + d_y \\ -\delta_y & \delta_x & 1 & \delta_y x_{eye} - \delta_x y_{eye} + d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Il faut alors appliquer cette transformation à chaque point du nuage et ajouter les bruits de mesure.

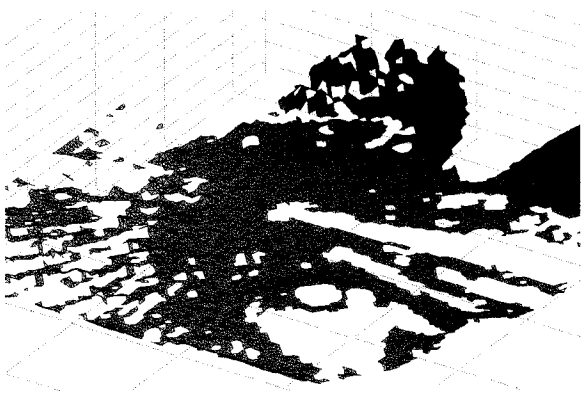


FIG. 3.15 Acquisition bruitée avant décimation

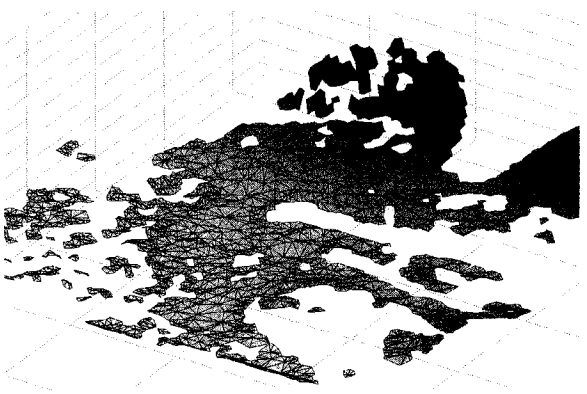


FIG. 3.16 Décimation de l'acquisition bruitée

3.2 Algorithme d'analyse de la trajectoire

Ce module représente en quelque sorte une simulation de l'Intelligence Artificielle du robot. Pour pouvoir tester nos algorithmes, nous avons besoin de simuler un système d'analyse qui va faire avancer le robot jusqu'à la position la plus sécuritaire de notre trajectoire calculée et déterminer la nécessité de faire une nouvelle acquisition de l'environnement. Ainsi, le système va parcourir la trajectoire et analyser chaque noeud parcouru afin d'observer si la solution trouvée est suffisamment sécuritaire. C'est-à-dire, la zone autour du noeud est suffisamment connue et la fonction de coût a assuré que cette zone est franchissable. De plus, avant de planifier une quelconque trajectoire, nous avons bien entendu besoin de préciser où on se trouve initialement et également où on doit se rendre. On voudra définir plusieurs objectifs, le système devra de façon autonome tenter de les atteindre successivement. Pour cela, il doit pouvoir détecter lorsqu'on est effectivement arrivé à destination. Dans le cas où l'objectif initial est hors de portée, il faut pouvoir en redéfinir un nouveau pour qu'il soit atteignable.

On pourra compléter ce système d'analyse par de nombreuses autres fonctionnalités. Dans le cas d'un LIDAR directionnel par exemple, comme cela était le cas dans un premier temps, ce système peut permettre de déterminer dans quelle direction regarder et depuis quel point de vue. Le robot peut également être amené à être bloqué lors de l'exploration, ce système doit alors être capable de prendre des décisions telles que avancer à l'aveugle considérant d'autres solutions de détection d'obstacles (capteurs de proximités par exemple). En effet, il peut arriver que la planification de trajectoire deviennent impossible lorsque il y'a une zone d'ombre et que le robot n'arrive pas à passer outre cette zone d'ombre. On simule alors que le robot avance tout de même légèrement en espérant que le robot puisse trouver un nouveau point de vue avec un meilleur champ de vision.

Il aurait été intéressant d'ajouter d'autres capacités plus évoluées. Par exemple, on se rend compte que l'algorithme de planification de trajectoire a tendance à passer au plus prêt des obstacles vu que l'on recherche à la base le plus court chemin. Ceci peut être gênant puisque lorsqu'on cherche à contourner de trop prêt un obstacle, celui-ci obstrue une partie du champ de vision. On pourrait donc imaginer que le robot puisse s'éloigner de sa trajectoire afin de faire l'acquisition d'une nouvelle carte de façon optimale permettant d'avoir la vue la plus dégagée possible, pour ensuite retourner sur sa route prévue. On se rend compte également lors des simulations qu'il est très difficile d'ajuster les différents coefficients intervenant dans le calcul de la fonction de coût. Par exemple, de nombreux compromis s'offrent à nous : doit-on préférer utiliser les zones déjà connues pour rejoindre un point ou au contraire prendre le risque de continuer en zone inconnue ? Doit-on privilégier la distance parcourue ou la prudence ? Quel détour sommes-nous prêt à faire pour éviter une zone dangereuse ? Pour solutionner ce problème, on pourrait concevoir un agent intelligent qui modifierait automatiquement les paramètres de sa fonction de coût permettant d'ajuster son *goût du risque.* On peut donc imaginer un robot qui va prendre tous les risques pour explorer une zone inconnue tout d'abord, puis se raviser petit à petit lorsqu'il ne peut plus avancer de façon sécuritaire, jusqu'à ce qu'il décide de faire finalement demi-tour et prendre une trajectoire hors de danger.

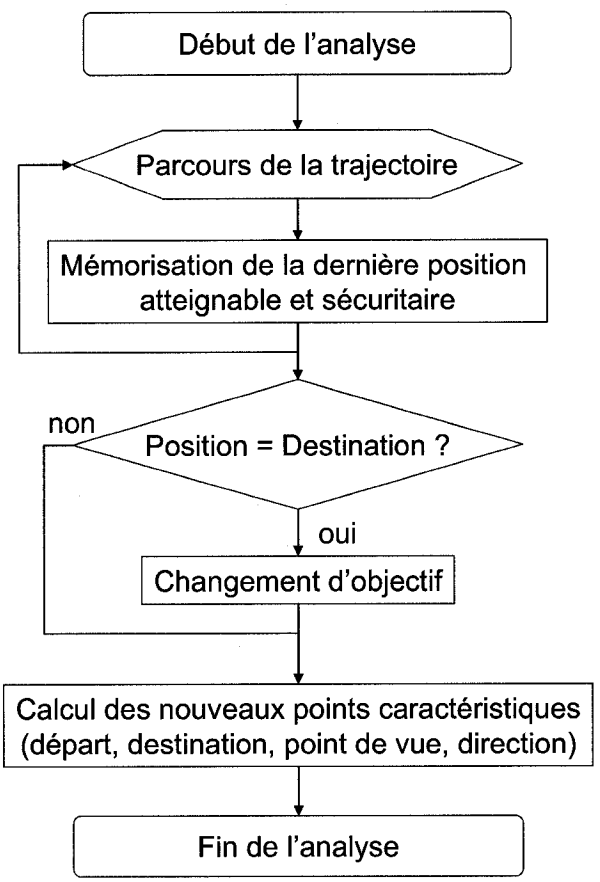


FIG. 3.17 Algorithme d'analyse de la trajectoire

3.3 Le Graphe

3.3.1 Algorithme de création du graphe

Nous avons déjà décrit suffisamment le principe de cet algorithme dans la section 2.3. Cette simulation a permis de mettre en évidence quelques soucis de rapidité d'exécution. Afin d'améliorer la vitesse de création du graphe, plusieurs modifications ont été apportées. La première version avait une structure de *main_mesh* sous forme de liste. Lorsqu'on voulait enregistrer un noeud dans une case il fallait rechercher dans la liste à quel endroit elle se situait. Il fallait donc parcourir cette liste de façon linéaire. Une solution pour remédier à cette recherche est de représenter la structure du quadrillage *main_mesh* sous la forme d'une matrice. Ainsi, en connaissant les extrêmes $X_{min} = dim \cdot E\left(\frac{min(x \epsilon Carte)}{dim}\right)$ (la fonction $E(x)$ étant la partie entière de x et dim la dimension de chaque case), $X_{max} = dim \cdot \left(E\left(\frac{max(x \epsilon Carte)}{dim}\right) + 1\right)$, $Y_{min} = dim \cdot E\left(\frac{min(y \epsilon Carte)}{dim}\right)$, et $Y_{max} = dim \cdot \left(E\left(\frac{max(y \epsilon Carte)}{dim}\right) + 1\right)$ on aura :

$$main\_mesh(i,j) = \begin{bmatrix} case(X_{min},Y_{min}) & \cdots & case(X_{min},Y_{max}-dim) \\ \vdots & case(X_{min}+(i-1)\cdot dim, Y_{min}+(j-1)\cdot dim) & \vdots \\ case(X_{max}-dim,Y_{min}) & \cdots & case(X_{max}-dim,Y_{max}-dim) \end{bmatrix}$$

L'élément $case(X,Y)$ est une structure du même type que dans le tableau 2.1 où X et Y correspondent aux coordonnées du coin inférieur de la case formée par les points (X,Y), $(X+dim,Y)$, $(X,Y+dim)$ et $(X+dim,Y+dim)$. Au moment où l'on souhaite enregistrer un nouveau noeud dans le maillage, si son barycentre a pour coordonnées (x_n,y_n), les indices i et j de cette case seront obtenues comme

ceci :

$$\begin{aligned} i &= E\left(\frac{x_n - X_{min}}{dim}\right) + 1 \\ j &= E\left(\frac{y_n - Y_{min}}{dim}\right) + 1 \end{aligned}$$

Ainsi, il n'y a plus besoin de rechercher la case dans une liste, l'enregistrement est donc instantané. En revanche, il faut implémenter un nouvel algorithme, qui va étendre cette matrice lorsqu'on doit rajouter des noeuds où la case ne se trouve pas dans la matrice *main_mesh*. Il va falloir ainsi rajouter suffisamment de lignes entières lorsqu'on a un noeud tel que $x_n > X_{max}$ et des colonnes lorsque $y_n > Y_{max}$. Lorsque le nouveau noeud a des coordonnées telles que $x_n < X_{min}$ ou $y_n < Y_{min}$, on obtient de nouvelles bornes X_{min} ou Y_{min}. Il faut rajouter également respectivement plusieurs lignes ou plusieurs colonnes mais décaler toutes les cases déjà existantes dans l'ancienne matrice de façon à pouvoir les insérer en haut ou à gauche de la matrice (figure 3.18).

Quelques optimisations de code ont également été réalisées. On peut voir dans la figure 2.2 que l'on parcourt tous les triangles une fois... Or on peut remarquer que cela est inutile. En effet, à chaque fois que l'on trouve un voisinage entre deux triangles voisins, il suffit de mémoriser ce lien à la fois dans le triangle courant et dans le triangle voisin. Il sera alors inutile de chercher des voisins pour des triangles pour lesquels on aura déjà trouvé les trois voisins.

Nous avons voulu également dissocier la partie concernant la création du graphe et la partie de calcul du coût. Deux avantages justifient ce choix : tout d'abord, si on veut implémenter des fonctions de coûts tenant compte de la taille du robot, il sera plus pratique d'avoir auparavant la structure du graphe avec tous les voisinages afin d'avoir accès rapidement aux triangles formant la zone où se situe le robot.

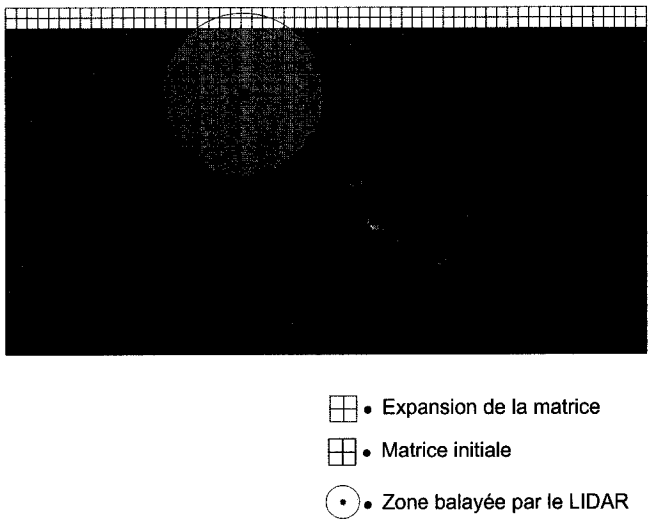


FIG. 3.18 Expansion du maillage

Cette solution permet également de recalculer les coûts sans toucher au graphe, au cas où l'on souhaite modifier notre stratégie.

D'autres améliorations peuvent encore être rajoutées. Par exemple, au niveau de la recherche des voisins, à la base, cette recherche est linéaire avec donc une complexité en $o(n)$. Or, au fur et à mesure que l'on avance dans la création du graphe, certains triangles ont déjà tous leurs voisins, il n'est donc plus nécessaire de regarder s'ils peuvent éventuellement être voisins du triangle courant. On pourrait encore aller plus loin, en modifiant la structure de la liste des triangles de façon à implémenter un algorithme de recherche dichotomique de complexité $o(\log(n))$.

Ces quelques modifications ont permis de diviser le temps de calcul par 4 environ sans tenir compte de la modification de la structure du quadrillage qui a permis de diminuer la complexité de l'algorithme.

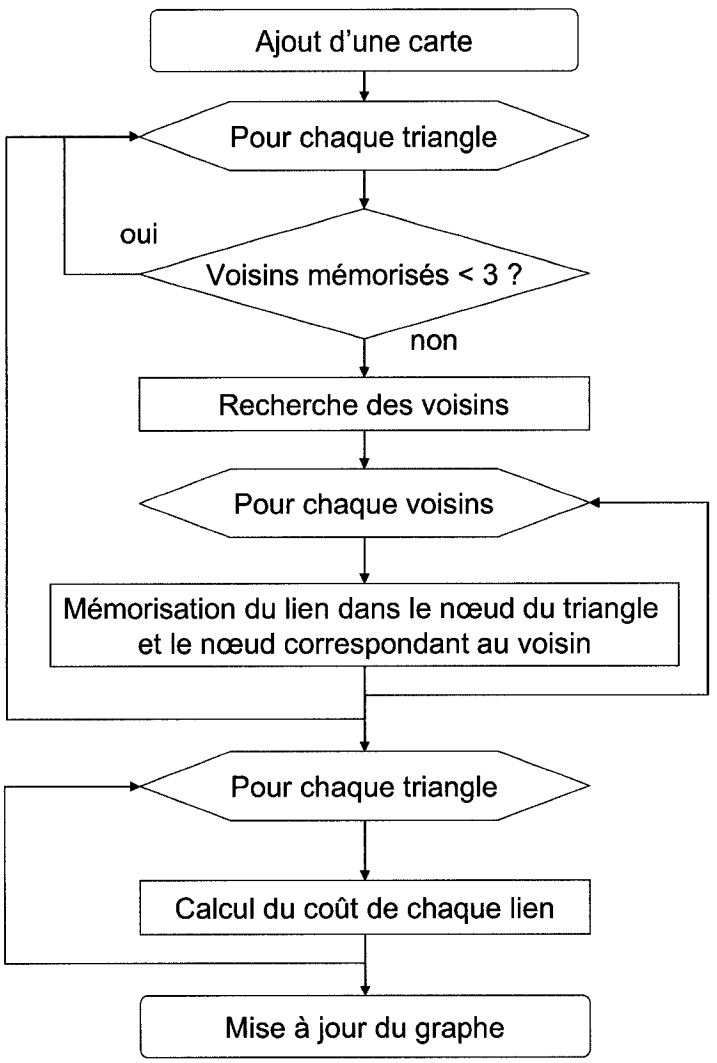


FIG. 3.19 Algorithme de mise à jour de la base de données

3.3.2 Fonctions de coût

Nous avons discuté dans la section 2.3 de quelques fonctions de coûts à titre d'exemples simples pour expliquer le principe de l'algorithme. Dans un second temps et afin d'obtenir des trajectoires plus réalistes, nous allons voir quelques fonctions de coût tenant compte des dimensions du robot, contrairement aux précédentes qui considéraient le robot comme un point. Ces fonctions sont basées sur des recherche précédentes de *E. Dupuis* au sein de l'Agence Spatiale Canadienne. Nous allons donc introduire les notions d'orientation et de rugosité dans nos fonctions de coûts. L'orientation va permettre de déterminer l'inclinaison latérale du robot ϕ (roulis), la pente dans le sens de la marche θ (tangage) et permettre de déterminer si un noeud est accessible ou non. La rugosité δ indique, quant à elle, si le terrain est plus ou moins accidenté, en mesurant la distance maximale des points sous l'empreinte du robot.

$$
\begin{aligned}
\text{distance} : c_1 &= \|X_1 - X_2\| \\
\text{roulis} : c_2 &= \frac{\phi}{\phi_{max}} \\
\text{tangage} : c_3 &= \frac{\theta}{\theta_{max}} \\
\text{orientation infranchissable} : c_4 &= \begin{cases} \infty & \text{si } \phi > \phi_{max} \text{ ou } \theta > \theta_{max} \\ 0 & \text{sinon} \end{cases} \\
\text{rugosité} : c_5 &= \frac{\delta}{\delta_{max}} \\
\text{rugosité infranchissable} : c_6 &= \begin{cases} \infty & \text{si } \delta > \delta_{max} \\ 0 & \text{sinon} \end{cases} \\
\text{dimension des triangles} : c_7 &= \exp\left(\frac{\|X_1 - X_2\|}{A_1 + A_2}\right)
\end{aligned}
$$

$$\text{changement de carte : } c_8 = \begin{cases} 1 & \text{si } carte_1 \neq carte_2 \\ 0 & \text{sinon} \end{cases}$$

$$\text{carte basse résolution (n° 1) : } c_9 = \begin{cases} 1 & \text{si } carte_1 = 1 \text{ ou } carte_2 = 1 \\ 0 & \text{sinon} \end{cases}$$

La distance c_1 est la distance entre les deux barycentres X_1 et X_2 des noeuds $n_1$et n_2.

Les roulis et tangage sont calculés tels qu'illustrés dans les figures 3.20 et 3.21. Supposons que l'on souhaite calculer l'orientation correspondant au lien entre deux noeuds n_1 et n_2. On mémorise dans une file Q tous les voisins du noeud n_2. Un voisin v_i est considéré comme faisant parti de l'empreinte du robot si un des trois sommets P_j de v_i ou son barycentre X_{v_i} est contenu dans une boule $\mathcal{B}(X_2, r)$ de centre le barycentre X_2 du noeud n_2 et de rayon r (paramètre à régler par l'utilisateur, dans notre cas des tests ont été effectués avec des valeurs de 50cm et 80cm, selon la marge de sécurité souhaitée). Puis pour chaque voisin v_i dans l'empreinte, on regarde si les voisins successifs sont eux aussi dans cette empreinte, jusqu'à ce que tous les triangles formant cette empreinte soient trouvés, c'est à dire lorsque la file Q est vide. L'empreinte sera donc un ensemble $\mathcal{F}$ composé du triangle n_2 et de tous les triangles alentours v_i respectant les conditions précédentes.

$$\mathcal{F} = \{n_2, v_i / \exists P_{j=1..3} \epsilon \mathcal{B}(X_2, r) \text{ ou } X_{v_i} \epsilon \mathcal{B}(X_2, r)\}$$

Connaissant $\mathcal{F}$ on peut calculer le vecteur normal moyen $\overrightarrow{N}$ au barycentre X_2 du triangle n_2 à partir des vecteurs normaux $\overrightarrow{n}$ ($n_z > 0$) et de l'aire A de chaque triangle de $\mathcal{F}$ tel que :

$$\overrightarrow{N} = \frac{\sum_{tri \epsilon \mathcal{F}} A_{tri} \overrightarrow{n_{tri}}}{\sum_{tri \epsilon \mathcal{F}} A_{tri}}$$

On doit ensuite calculer les vecteurs permettant d'orienter le robot suivant la direction du vecteur $\overrightarrow{X_1X_2}$. On définit donc $\overrightarrow{c}$ le vecteur transversal (perpendiculaire au sens de la marche du robot) et $\overrightarrow{a}$ le vecteur longitudinal (dans le sens de la marche) comme ceci :

$$\begin{aligned} \overrightarrow{c} &= \overrightarrow{N} \wedge \overrightarrow{X_1X_2} \\ \overrightarrow{a} &= \overrightarrow{c} \wedge \overrightarrow{N} \end{aligned}$$

L'angle de roulis ϕ et l'angle de tangage θ seront les angles formés respectivement par $\overrightarrow{c}$ et $\overrightarrow{a}$ avec le plan horizontal :

$$\begin{aligned} \phi &= \left|\text{atan2}\left(c_z, \sqrt{c_x^2 + c_y^2}\right)\right| \\ \theta &= \left|\text{atan2}\left(a_z, \sqrt{a_x^2 + a_y^2}\right)\right| \end{aligned}$$

On peut noter que nous ne tenons pas compte du signe de θ, on peut très bien tenir compte de ce signe et introduire une limite $\theta_{min} \neq -\theta_{max}$.

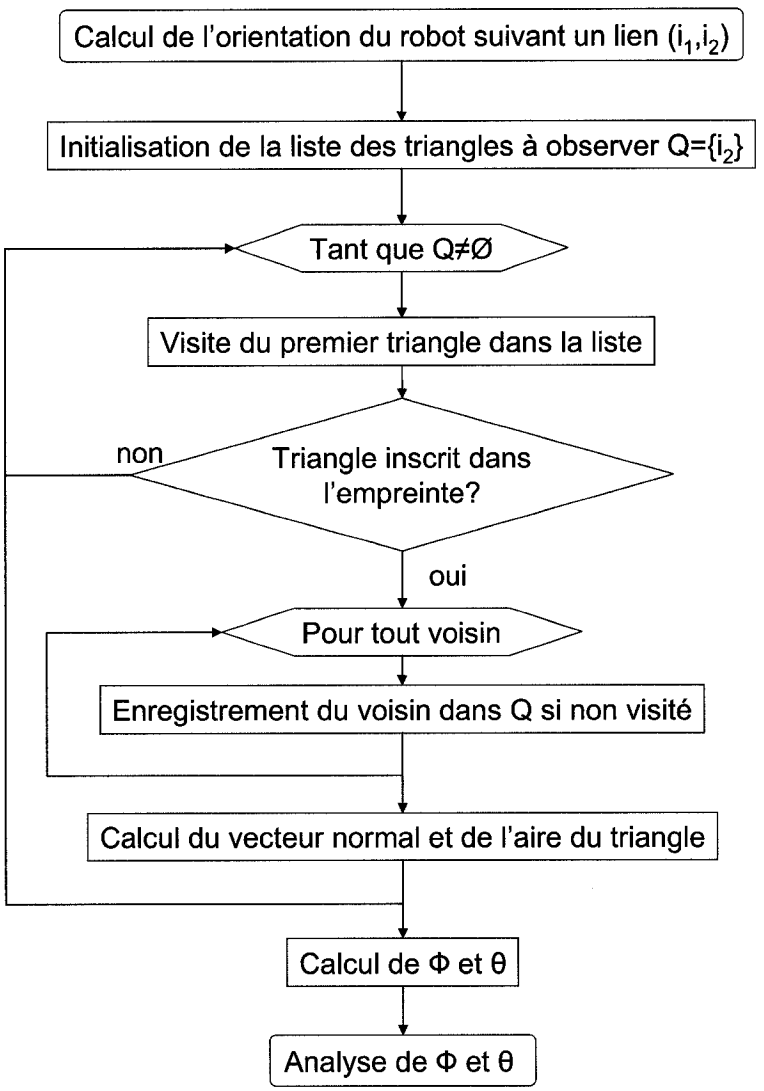


FIG. 3.20 Algorithme de cacul de l'orientation de l'empreinte du robot

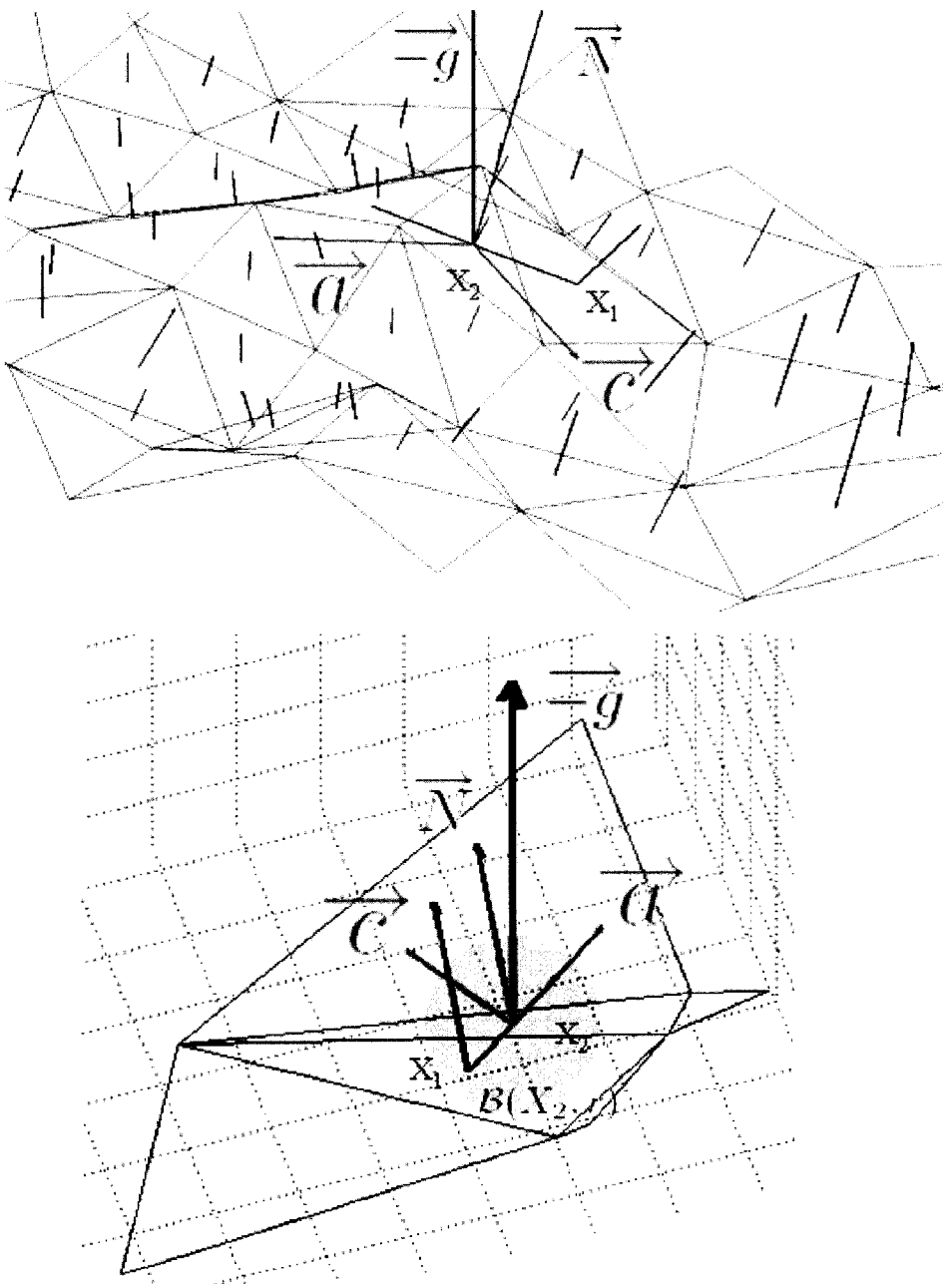


FIG. 3.21 Illustration des vecteurs et angles nécessaires dans le calcul de l'orientation

A partir des sommets P_j contenus dans la boule $\mathcal{B}(X_2, r)$ et du vecteur normal moyen $\overrightarrow{N}$, la rugosité est la distance du point P_j le plus éloigné du plan moyen :

$$\delta = \max\left(|\overrightarrow{N} \cdot \overrightarrow{P_j X_2}| / P_j \epsilon \mathcal{B}(X_2, r)\right)$$

Le coût c_7 permet d'avoir une mesure concernant la taille des triangles. En effet, il serait souhaitable de privilégier les triangles possédant une plus grande aire, ceux-ci indiquent que la zone traversée est plus sécuritaire que s'il y'avait plusieurs petits triangles. Il va en même temps pénaliser des liens trop longs. Cela revient finalement à privilégier les arêtes larges, et ainsi traverser les triangles dans le sens de la largeur.

Le coût c_8 permet de tenir compte du changement de carte. Comme on a pu le voir dans la section 2.6, il est nécessaire de rajouter un coût au passage d'une carte à une autre afin de ne pas effectuer des changements de cartes inutiles et éviter une accumulation les erreurs qui pourrait se produire lors d'un changement de carte. Quant au coût c_9 concernant les noeuds situés sur la carte basse résolution, il est nécessaire afin de ne pas basculer trop rapidement sur celle-ci : elle peut être bien souvent privilégiée lors de la planification parce qu'elle ne représente bien entendu pas tous les obstacles visibles sur une carte haute résolution. On souhaite donc pousser l'algorithme A* à rechercher le plus possible une solution dans les cartes haute résolution avant de décider de continuer le chemin sur la carte basse résolution. Le premier passage sur la carte basse résolution indique l'endroit où le robot doit s'arrêter et faire une nouvelle acquisition de carte (analyse traitée par le système décrit à la section 3.2). Les figures 3.22 et 3.23 sont un aperçu des liens considérés dangereux par les fonctions de coûts c_4 et c_6 de l'orientation et de la rugosité.

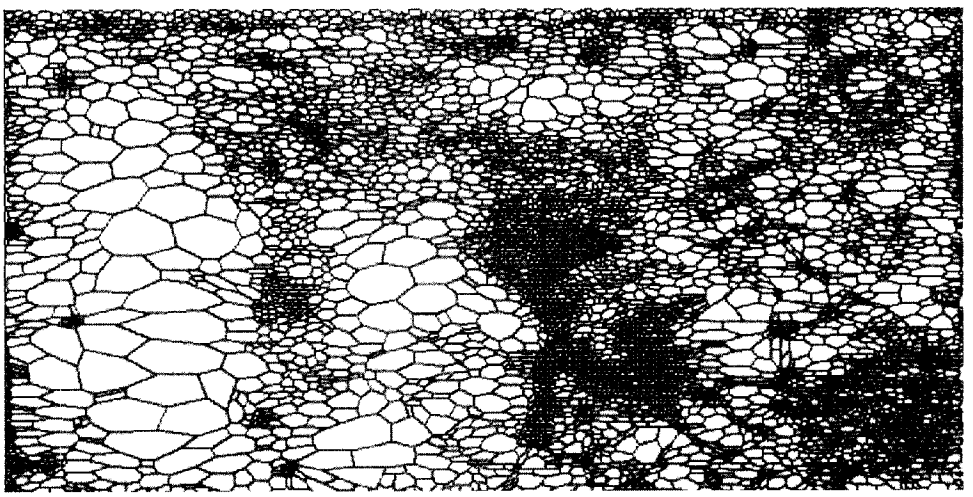


FIG. 3.22 Représentation des valeurs c_4 des orientations interdites des liens du graphe

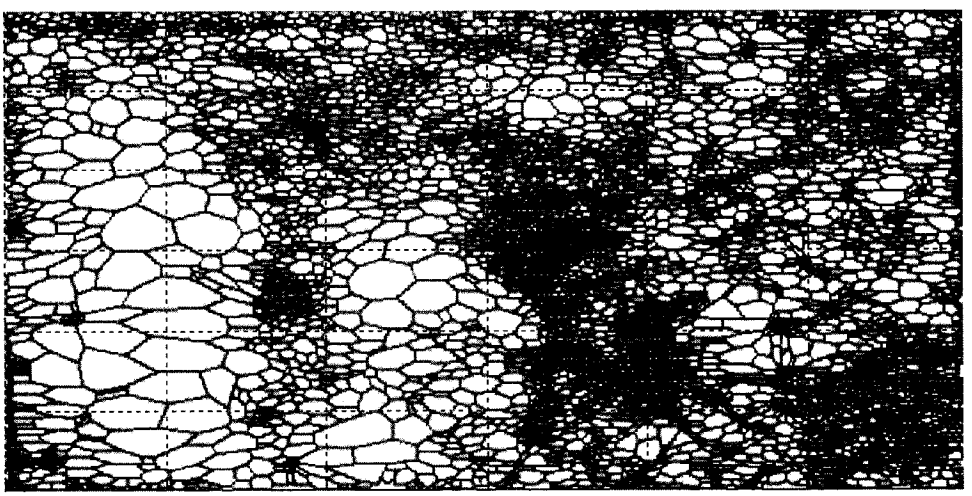


FIG. 3.23 Représentation des valeurs c_6 de la rugosité de plus de 20cm des liens du graphe

3.4 Algorithme A*

Nous ne reviendrons pas sur la description de l'algorithme déjà détaillée dans la section 2.4 et la figure 2.8. Nous allons discuter ici du calcul du coût $G(i)$ à partir des fonctions de coûts $c_{1..9}$. Il faut tout d'abord différencier deux cas. Le premier correspond à l'exploration dans un même graphe, le second au passage d'un graphe à un autre.

Dans le premier cas, pour calculer le coût $G(j)$ d'un noeud n_j voisin du noeud n_i, on a $G(j) = G(i) + cout(i,j)$ où la fonction $cout(i,j)$ va être une combinaison linéaire des $c_{1..9}$ avec pour coefficients, des scalaires associés $\lambda_{1..9}$. Ces poids λ_k vont représenter l'importance accordée à une fonction de coût c_k au sein de $cout(i,j)$. Il faut noter également qu'il est indispensable de multiplier par la distance c_1 tous les autres coûts $c_{2..9}$. Le cout d'un lien sera de la forme :

$$cout(i,j) = c_1 \cdot \left(1 + \sum_{k=2}^{9} \lambda_k \cdot c_k\right)$$

Ainsi pour deux liens consécutifs (h,i) et (i,j) de coûts $c_{1..9}$ et $c'_{1..9}$, leur somme préservera l'importance selon la distance et produira une moyenne des couts $\overline{c_k} = \frac{c_1 c_k + c'_1 c'_k}{c_1 + c'_1}$ le long de ces deux liens :

$$\begin{aligned} cout(h,i) + cout(i,j) &= c_1 \cdot \left(1 + \sum_{k=2}^{9} \lambda_k \cdot c_k\right) + c'_1 \cdot \left(1 + \sum_{k=2}^{9} \lambda_k \cdot c'_k\right) \\ &= (c_1 + c'_1) + \sum_{k=2}^{9} \lambda_k \cdot (c_1 \cdot c_k + c'_1 \cdot c'_k) \\ &= (c_1 + c'_1) \cdot \left(1 + \sum_{k=2}^{9} \lambda_k \cdot \frac{c_1 \cdot c_k + c'_1 \cdot c'_k}{c_1 + c'_1}\right) \end{aligned}$$

$$= \overline{c_1} \cdot \left(1 + \sum_{k=2}^{9} \lambda_k \cdot \overline{c_k}\right)$$

Il faut rappeler que nous sommes dans le cas où on réalise l'exploration sur un même graphe, on aura alors $c_8 = 0$ puisqu'il n'y a pas de changement de carte.

On se doit de différencier le calcul du coût lorsqu'on change de carte pour plusieurs raisons. Le calcul du coût entre deux noeuds similaires (voir section 2.4) est réalisé *en ligne*, c'est à dire qu'on ne recherche pas tous les points similaires lors de la création du graphe. On doit calculer le coût $cout(i, i')$ entre un noeud n_i, et son noeud similaire sur une autre carte $n_{i'}$, pendant l'exploration du graphe. On se rend compte également que les fonctions de coûts utilisées n'ont plus de signification lors du changement d'une carte. Deux points similaires sont sensés représenter un même point. Or, les erreurs de positionnement des cartes et la discrétisation du terrain introduisent une distance et une orientation qui n'ont pas lieu d'être. Ainsi on pourrait très bien avoir une distance c_1 nulle, ce qui produirait un coût $cout(i, i')$ nul. Il faut donc tenir compte dans ce cas, uniquement du coût du changement de carte ajouté à la distance. La fonction de coût générale aura donc la forme suivante :

$$cout(i,j) = \begin{cases} c_1 \cdot \left(1 + \sum_{k=2}^{9} \lambda_k \cdot c_k\right) & \text{si } carte_i = carte_j \\ c_1 + \lambda_8 \cdot c_8 & \text{si } carte_i \neq carte_j \end{cases}$$

Il faut à présent déterminer les combinaisons de λ_k permettant d'obtenir des trajectoires optimales au niveau de la distance et de la sécurité.

3.5 Résultats

3.5.1 Données et machine utilisées

Les simulations ont été effectuées sur un serveur LINUX 2.6 Red Hat 4.1 équipé de deux processeurs Intel Xeon 2,4GHz avec *hyperthreading* et 2Go de mémoire vive, la version de Matlab utilisée est la 7.2. Avant de commencer à simuler une exploration au complet, il faut tout d'abord régler les λ_k dans des bonnes conditions. Pour cela, dans un premier temps, nous considèrerons que nous avons en notre possession la carte entière en haute résolution (HR) décimée et la carte basse résolution (BR) (voir figures 1.9 et 1.8). Ces deux cartes contiennent respectivement 17503 et 3600 triangles, donc autant de noeuds dans les graphes à calculer. Une fois que nous obtiendrons des trajectoires correctes dans cette situation, nous pourrons adapter nos paramètres pour simuler l'exploration. Une telle simulation va permettre de créer une carte en haute résolution à l'aide des algorithmes vu précédemment. On pourra simuler ainsi l'acquisition d'une carte de la même manière qu'un scan LIDAR de l'environnement réel, depuis l'endroit où se situerais le robot. Nous analyserons les résultats finaux étape par étape.

La figure 3.24 indique les symboles utilisés pour visualiser la simulation ainsi que la position initiale du robot et les cinq objectifs à atteindre.

3.5.2 Réglages des paramètres λ_k

Le tableau 3.1 représente cinq combinaisons de valeurs pour le vecteur λ. Les trois premières simulations sont effectuées sur les cartes au complet. Le graphe possède alors 21103 noeuds sur deux cartes. Les simulation n° 1 et 2 ne prennent en compte que la distance et le changement de carte. On observe donc que le robot va tout

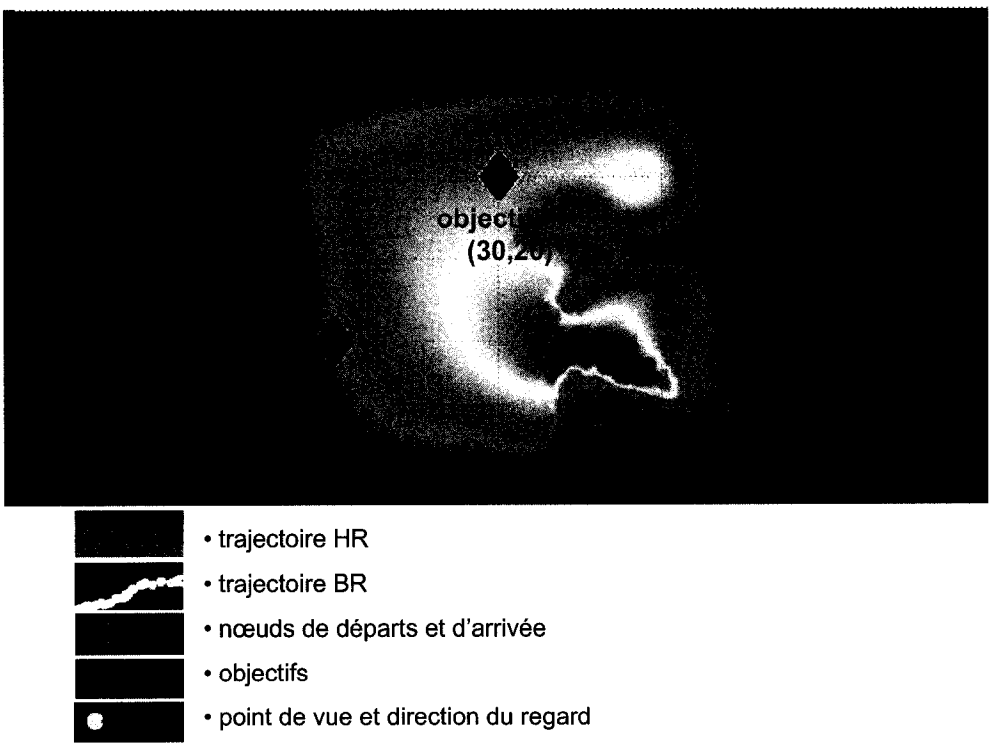


FIG. 3.24 Légende des graphiques

droit en direction de chaque objectif. On se rend compte qu'il faut naturellement augmenter les coefficients du changement de carte et de la carte BR (λ_8 et λ_9), en effet une même trajectoire va être moins coûteuse sur la carte BR puisqu'elle implique moins de *zig-zags* entre les triangles. Le troisième essai, par la suite, tient compte des liens considérés comme infranchissables lors de la création du graphe, on peut observer ainsi le contournement des obstacles.

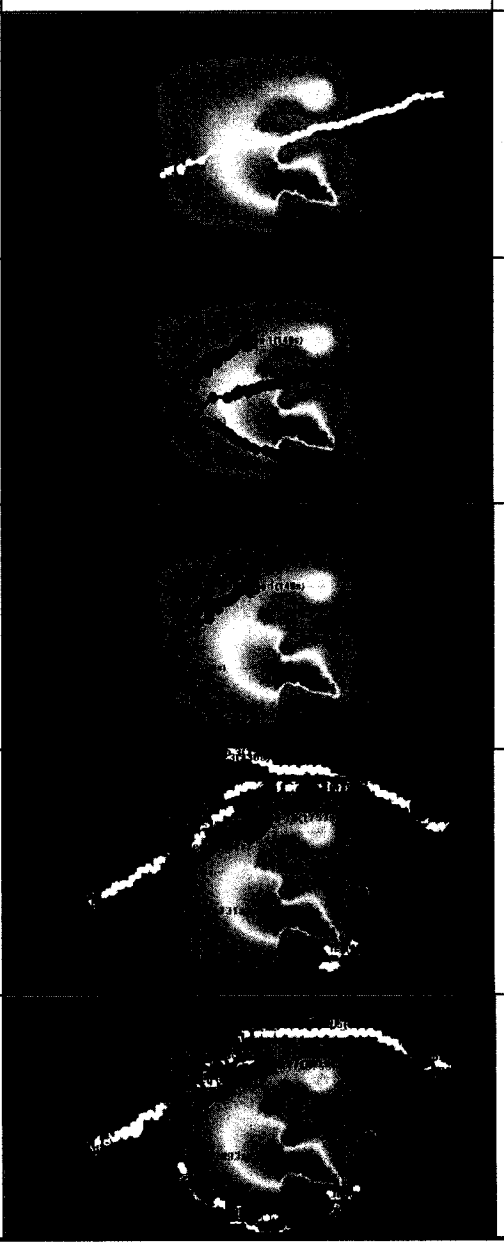
En ce qui concerne les simulation 4 et 5, elles correspondent à une exploration. C'est à dire que le robot avance pas à pas. Il ne possède à la base que la carte BR et fait l'acquisition LIDAR (simulée) dés que cela lui semble nécessaire. On obtient donc nécessairement des trajectoires sous-optimales au niveau global. Il faut cependant être conscient que les informations au départ sont très limitées, et que le robot doit

donc avancer avec ce qu'il connaît de son environnement. Or, la limite de la portée du LIDAR et les zones d'ombre forcent le robot à faire des détours afin d'avoir un meilleur champ de vision ou d'avancer petit à petit jusqu'à ce que la vue se dégage.

Observons les différences entre les simulations 4 et 5. Les objectifs ont été placés afin de faire un tour de la colline au centre de la carte. Or, dans la simulation 4, le robot préfère effectuer un demi-tour au détriment de la distance en utilisant les cartes connues, plutôt que de s'aventurer dans une zone inconnue. Pour, que le robot puisse prendre le risque de s'aventurer dans une zone inconnue, il faut donc diminuer le coefficient correspondant à la pénalité sur les noeuds de la carte BR. On peut observer également une différence introduite par les paramètres λ_2, λ_3 et λ_5 correspondant aux valeur de ϕ, θ et δ. Le robot va alors préférer s'éloigner de la colline pour retrouver un terrain plus plat (simulation n° 4) en s'éloignant d'une trajectoire plus courte. Ces paramètres sont optimisables à l'infini et leur réglage va dépendre énormément des essais sur le terrain et des capacités de franchissement du robot.

3.5.3 L'exploration étape par étape

Nous allons à présent décrire étape par étape le déroulement d'une exploration réalisée avec $\lambda = [1, 0.1, 0.1, 1, 0.1, 1, 0.005, 50, 5]$ (figures 3.25 et 3.26). A l'étape n° 1, on calcule tout d'abord le graphe de 3600 noeuds pour planifier une première trajectoire globale. On peut alors faire l'acquisition d'une première carte (zones avec un contraste plus élevé sur les figures 3.25 et 3.26). On calcule le graphe correspondant puis un nouveau chemin local au niveau de la carte la plus récente puis globale jusqu'au premier objectif. On simule ainsi que le robot est parti de sa position initiale en $(10, 10)$ pour atteindre la position $(22, 18)$ en toute sécurité. Il doit alors à nouveau faire un scan de l'environnement puis continuer à avancer

cas	λ	trajectoire	notes
1	$\lambda = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$		Temps de calcul : 1100s 2 cartes (21103 nœuds) : -carte BR et HR décimée **Commentaires :** -Trajectoire en ligne droite -passage rapide sur la carte BR
2	$\lambda = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$		Temps de calcul : 1200s 2 cartes (21103 nœuds) : -carte BR et HR décimée **Commentaires :** -trajectoires en ligne droite -trajectoires sur la carte HR
3	$\lambda = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0{,}1 \\ 20 \\ 20 \end{bmatrix}$		Temps de calcul : 1300s 2 cartes (21103 nœuds) : -carte BR et HR décimée **Commentaires :** -contournement des obstacles -augmentation de λ_8 et λ_9 pour ne pas basculer sur la carte BR à cause du contournement -*zig-zags* à cause de λ_7
4	$\lambda = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0{,}01 \\ 20 \\ 20 \end{bmatrix}$		Temps de calcul : 1800s 11 cartes (30274 nœuds) : -carte BR -cartes LIDAR simulées **Commentaires :** -contournement des obstacles -utilise les cartes déjà existante (demi-tour) entre les étapes 2 et 3 - λ_2 et λ_3 trop grands
5	$\lambda = \begin{bmatrix} 1 \\ 0{,}1 \\ 0{,}1 \\ 1 \\ 0{,}1 \\ 1 \\ 0{,}005 \\ 20 \\ 5 \end{bmatrix}$		Temps de calcul : 2500s 16 cartes (38009 nœuds) : -carte BR -cartes LIDAR simulées **Commentaires :** -distance privilégiée

TAB. 3.1 Trajectoires selon les λ_k

ainsi de suite. Il atteint alors les objectifs 1 et 2. Pour aller à l'objectif n° 3 situé en $(38, 5)$, à partir de l'étape 13, le robot explore le bas de la carte puis se retrouve finalement en zone connue à partir de l'étape 18, zone scannée à l'étape 2. On peut ainsi terminer les objectifs 3,4 et 5 en planifiant nos trajectoires dans les graphes calculés dans les premières étapes. Le tableau 3.2 récapitule les chiffres concernant chaque étape de l'exploration afin d'avoir une idée des distances parcourues, des temps de calculs, et de la taille des données.

3.5.4 Critiques et analyses

On peut tirer plusieurs conclusions des résultats précédents. Tout d'abord on peut affirmer que notre système fonctionne parfaitement. Les objectifs sont atteints avec des performances convenables et les trajectoires semblent être satisfaisantes. Il faudrait bien sûr réaliser des essais sur le terrain pour pouvoir valider de façon certaine nos algorithmes. Un des points importants à noter est le temps mis par notre système pour calculer l'ensemble des coûts du graphe. La complexité est en $o(n)$ et peut difficilement être améliorée. On devrait pouvoir sans doute optimiser un peu l'algorithme pour diminuer le temps de calcul, mais la complexité est au mieux $o(n)$.

Le temps de calcul est donc sans doute le principal souci de nos algorithmes. D'autant plus que la plupart des cartes scannées en simulation possèdent au maximum 4000 triangles après décimation qui demandent ensuite moins de 3 minutes à l'algorithme de création de graphe pour effectuer ses calculs (voir étape 12 dans le tableau 3.2, 161s pour 3845 noeuds). Or d'après quelques données fournies par l'ASC, un scan obtenu dans les conditions réelles peut contenir jusqu'à prés de 10000 triangles. On peut donc imaginer par interpolation qu'un graphe de 10000 noeuds serait créé en environ 7 minutes. Cela peut sembler raisonnable, mais il fau-

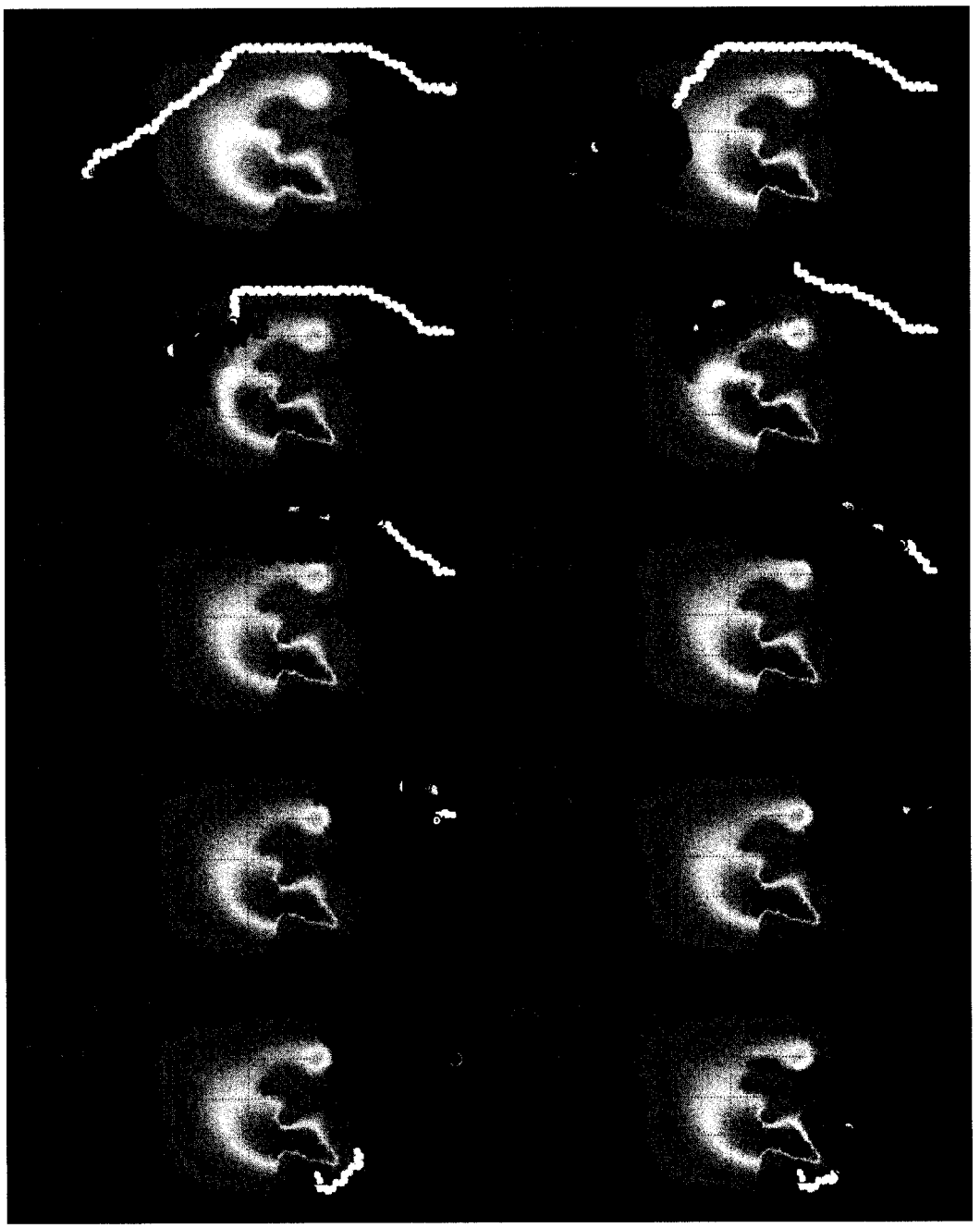


FIG. 3.25 Etapes 1 à 10 de l'exploration

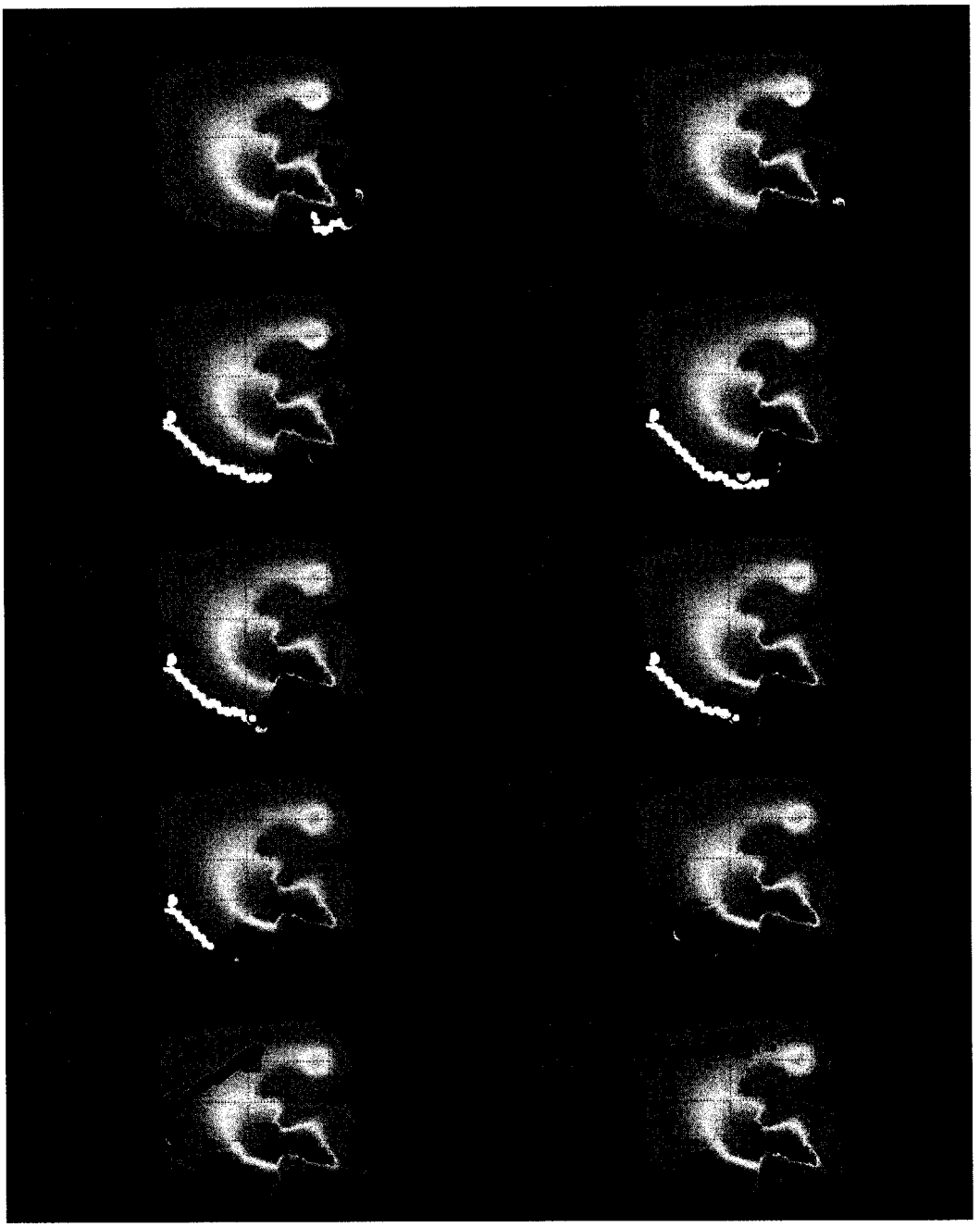


FIG. 3.26 Etapes 11 à 20 de l'exploration

étape	objectif	départ	scan		décimation		graphe		trajectoire			arrivée
n°	x,y	x,y	triangles	temps (s)	triangles	temps (s)	noeuds (total)	temps (s) graph-coût	avance (total) (m)	temps (s)	noeuds visités	x,y
1	55,20	10,10	3600	-	-	-	3600 (3600)	9s-59s	0m (0m)	4s	3028	10,10
2	55,20	10,10	24901	15s	1880	34s	1880 (5480)	3s-57s	20m (20m)	6s	4818	22,18
3	55,20	22,18	21285	13s	2392	29s	2392 (7872)	5s-72s	8m (28m)	11s	6704	27,21
4	55,20	27,21	13911	11s	2830	19s	2830 (10702)	7s-98s	21m (49m)	17s	8660	37,28
5	55,20	37,28	9494	11s	1672	13s	1672 (12374)	3s-51s	13m (62m)	14s	6367	46,28
6	55,20	46,28	10066	11s	1794	14s	1794 (14168)	3s-63s	9m (71m)	6s	3136	50,24
7	55,20	50,24	13480	11s	2717	19s	2717 (16885)	5s-97s	6m (77m)	8s	3182	53,20
8	55,20	53,20	15301	13s	2997	21s	2997 (19882)	7s-101s	2m (79m)	1s	24	55,20
9	38,5	55,20	-	-	-	-	-	-	23m (102m)	23s	8227	44,10
10	38,5	44,10	13968	13s	3559	20s	3559 (23441)	10s-140s	4m (106m)	20s	6681	44,7
11	38,5	44,7	13323	13s	3547	19s	3547 (26988)	10s-139s	2m (108m)	23s	6872	43,6
12	38,5	43,6	14003	13s	3845	20s	3845 (30833)	12s-149s	4m (112m)	1s	187	38,5
13	20,10	38,5	-	-	-	-	-	-	8m (120m)	95s	21786	34,3
14	20,10	34,3	1757	7s	998	3s	998 (31831)	1s-53s	1m (121m)	62s	13386	35,3
15	20,10	35,3	5447	8s	1701	8s	1701 (33532)	3s-75s	4m (125m)	76s	16261	33,2
16	20,10	33,2	7106	9s	1743	10s	1743 (35275)	3s-72s	1m (126m)	80s	17035	32,2
17	20,10	32,2	7375	9s	1759	10s	1759 (37034)	3s-71s	7m (133m)	46s	10139	27,4
18	20,10	27,4	7721	9s	975	11s	975 (38009)	1s-29s	10m (143m)	21s	5760	20,10
19	30,20	20,10	-	-	-	-	-	-	9m (152m)	26s	9282	30,20
20	10,10	30,20	-	-	-	-	-	-	18m (170m)	20s	6640	10,10

TAB. 3.2 Chiffres d'une exploration étape par étape

drait pouvoir le comparer et l'ajouter au temps que chaque étape de l'exploration peut prendre dans la pratique : acquisition LIDAR, décimation, calcul de la spline, guidage, déplacement...

Quelques optimisations pourraient être apportées. Tout d'abord dans la recherche des voisins, on considère que la décimation en Maillage Triangulaire Irrégulier et que la création du graphe sont deux systèmes séparés. Or, on peut remarquer que l'on effectue déjà cette recherche de voisinage dans la décimation. Il suffirait de partager l'information créée lors de la décimation, on éviterait ainsi d'avoir à recalculer les voisinages lors de la réalisation du graphe. On gagnerait ainsi quelques dizaines de secondes, puisqu'il ne restera alors qu'à référencer chaque noeud dans notre matrice *main_mesh*.

Une optimisation beaucoup plus importante serait de considérer au départ uniquement les relations de voisinage dans le graphe et de ne pas calculer dans un premier temps les coûts de chaque liens. En effet, il est facile de se rendre compte que beaucoup de noeuds pour lesquels on a calculé ces coûts ne seront jamais visités. Ainsi, ces coûts seraient calculés en ligne au moment de la planification de trajectoire avec l'algorithme A*, puis mémorisés au fur et à mesure dans le graphe. Certes on augmentera énormément le temps de la planification de trajectoire en elle même, qui passera de quelques dizaines de secondes à plusieurs minutes, mais ce temps sera très largement récupéré au moment de la création du graphe qui ne prendra que quelques secondes.

L'algorithme A* pourrait également être amélioré. Il est possible d'implémenter une variante appelée IDA* (*Iterative Deepening A**) moins gourmande en ressource mémoire (Russel, 1995). Il est très important de remarquer que les paramètres λ_k vont également jouer énormément sur les performances de l'algorithme A*, en termes de rapidité de calcul. En effet, il faut faire attention à ne pas obtenir

des coûts trop supérieurs à l'heuristique, c'est-à-dire au simple coût de distance. Plus nos fonctions de coûts sont proches de l'heuristique, plus l'heuristique sera performante. La quantité de noeuds visités sera donc réduite. Ceci serait d'autant plus important si on met en oeuvre l'optimisation d'un calcul en ligne des coûts des liens du graphe qui va augmenter considérablement le temps de calcul pour chaque noeud visité.

L'outil de simulation qui a été développé ici avait pour but de démontrer le bon fonctionnement d'une planification de chemin dans un atlas de cartes. Deux idées principales permettent de réaliser efficacement cette planification. D'une part, il faut référencer chaque noeud de nos graphes dans une matrice. Cette matrice nous sert ensuite à trouver immédiatement quels noeuds similaires peuvent nous permettre de passer d'une carte à une autre. D'autre part, il faut modifier notre algorithme de planification afin de tenir compte de ce changement de carte. Il faut introduire une seconde notion de coût entre deux noeuds situés sur deux cartes différentes. Ce second coût doit bien entendu être une variante du simple coût entre deux noeuds d'un même graphe, afin de pouvoir les comparer. Les résultats en simulation sont à la hauteur des objectifs fixés. Il faut cependant être conscient qu'il reste encore de nombreux obstacles à franchir avant de pouvoir intégrer ce système réellement.

CONCLUSION

Le souhait d'explorer toujours plus loin des terres inconnues permet de faire avancer les recherches en robotique mobile au niveau de l'autonomie. L'exploration martienne est sans aucun doute une des principales motivations pour augmenter toujours plus l'autonomie des robots. La principale caractéristique que l'ont demande à un *rover* est d'être capable de parcourir des centaines de mètres sans aucune intervention humaine. Ce projet en collaboration avec l'Agence Spatiale Canadienne tente d'apporter des nouvelles solutions afin d'augmenter la distance quotidienne parcourue par un *rover*. Les recherches préalables de l'Agence Spatiale Canadienne se sont orientées vers l'utilisation d'un scanner de type LIDAR comme senseur pour mémoriser les informations de l'environnement.

La cartographie à partir du LIDAR nécessite plusieurs minutes, il est donc impossible de recréer en temps réel une carte générale du terrain. L'exploration doit donc se réaliser étape par étape, où le robot doit tenter de se rapprocher d'une destination. Il devra successivement scanner l'environnement, planifier son chemin, puis avancer avant de scanner à nouveau lorsqu'il se retrouve en zone inconnue. Le robot va ainsi construire un atlas de carte, au fur et à mesure, le long de son parcours. Cette notion d'atlas de cartes est encore peu traitée dans le domaine de la robotique mobile. Or, les avantages sont nombreux : aucune information n'est perdue, et il n'y a pas d'accumulation d'erreur lorsqu'on fusionne les cartes. De plus, le traitement d'un atlas de cartes ne demandera pas plus de ressources en s'agrandissant, chaque carte étant traitée indépendamment.

Nous nous sommes donc basés sur les cartes fournies par l'Agence Spatiale Canadienne afin de construire notre atlas et trouver notre chemin. Ces cartes sont initialement sous forme d'un nuage de points qu'il faut ensuite trianguler puis dé-

cimer afin d'obtenir une carte sous forme d'un Maillage Triangulaire Irrégulier. Un tel maillage est composé de triangles où chaque sommet appartient au nuage de point initial. Chaque triangle est alors considéré comme un noeud d'un graphe. Un graphe est constitué de noeuds et de liens entre ces noeuds. Notre graphe représentera donc chaque triangle et les connexions avec ses voisins ayant une arête commune. Chaque nouveau scan acquis au fur et à mesure de l'exploration va donc rajouter une nouvelle carte puis un nouveau graphe. On a constitué ainsi un atlas de graphes.

Afin de pouvoir planifier un chemin dans notre atlas, il faut tout d'abord un algorithme de recherche de plus court chemin puis un moyen d'évoluer d'un graphe à l'autre. L'algorithme A* a été préféré comme solution pour la planification. Celui-ci est en effet optimal et ses performances sont reconnues pour être les plus efficaces dans ce type d'application. Afin de pouvoir évoluer dans notre atlas, nous devons connaitre, à partir de n'importe quelle position, toutes les options possibles sur toutes les cartes disponibles à cet endroit. Il faut, pour cela, générer un quadrillage qui va permettre de faciliter cette recherche. Chaque case du quadrillage répertorie tous les noeuds disponibles dans ce carré de terrain. Ce quadrillage est structuré comme une matrice superposée sur le sol. A chaque endroit, il est donc possible de retrouver tous les noeuds disponibles pour poursuivre notre chemin. L'algorithme de planification A* a dû être modifié pour pouvoir être capable de planifier un chemin sur plusieurs graphes.

Nous avons voulu par la suite démontrer l'efficacité de notre système en simulant au mieux une exploration, de la même manière qu'elle se déroulerait dans un environnement inconnu. Nous avions à notre disposition deux cartes du terrain de simulation de l'Agence Spatiale Canadienne. La carte basse résolution est considérée comme totalement connue afin de planifier la trajectoire globale. La carte haute résolution permet de construire les scans que le robot obtiendrait s'il réali-

sait une acquisition réelle avec le LIDAR. Il a fallu concevoir des systèmes simulant le balayage du Laser en tenant compte des zones d'ombre et des erreurs de mesure. Puis les algorithmes de décimation permettent finalement d'obtenir des cartes sous forme de maillage triangulaire irrégulier suffisamment représentatives pour prouver le bon fonctionnement de la solution de planification dans un atlas de cartes.

Les résultats de cette simulation sont un premier pas pour augmenter l'autonomie des robots sur de longues distances. Il reste encore de nombreux tests à effectuer. Il faudrait bien entendu pouvoir vérifier le bon fonctionnement dans des conditions réelles de navigation. Certains paramètres ont pu être pris en compte, comme le bruit de mesure, l'erreur de localisation et les capacités de franchissement du robot. Il est impossible d'affirmer qu'ils sont suffisamment représentatifs des conditions réelles. De plus, il faudrait pouvoir vérifier si les trajectoires planifiées sont par la suite effectivement compatibles avec les autres nombreux systèmes que compose l'intelligence artificielle du robot mobile : système de prise de décision, de guidage, de déplacement, de localisation, de cartographie, etc... Les conditions réelles vont être également très différentes de celles de la simulation au niveau des capacités de calcul du robot et de la taille des cartes. L'ordinateur embarqué aura une puissance limitée et les cartes obtenues en réalité auront bien plus de triangles.

RÉFÉRENCES

AGUDELO A., *Planification de Chemin pour un Robot Mobile dans un Environnement Partiellement Connu.* Mémoire M.Sc.A, Ecole Polytechnique de Montréal,QC, 2008

ASTOLFI A., *Exponential stabilization of wheeled mobile robot via discontinuous control* Journal of Dynamic Systems Measurement and Control, March 1999, pp. 121-126

BAE K., *et al.*, *Automated Registration of unorganised point clouds from terrestrial laser scanners* Thèse Ph.D, Curtin University Of Technology, 2005

BAKAMBU J., *et al. 3D Reconstruction of Environments for Planetary Exploration*, Canadian Space Agency, Space Technologies, CRV.2005.3, Pages 594-601, 2005

BAKKER B., *et al. Hierarchical Dynamic Programming for Robot Path Planning*, Intelligent Systems Laboratory Amsterdam. Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, p. 3720-3725.

CIGNONI P., MONTANI C., SCOPIGNO R., *A comparison of mesh simplification algorithms.* In Computers & Graphics, volume 22. Pergamon Press. 1998

COHEN-OR D., SHAKED A., *Visibility and dead-zones in digital terrain maps.* Computer Graphics Forum, 14(3) :C/171–C/180, Sept. 1995

DUPUIS E., *et al. Towards autonomous long range navigation*, ISAIRAS 2005 CONFERENCE, Munich, 5-8 Septembre 2005.

HOPPE H., *et al.*, *Surface Reconstruction from Unorganized Points* Computer Graphics, 26,2, July 1992

HUNTSBERGER T., *et al.*, *Rover Autonomy for Long Range Navigation and Science Data Acquisition on Planetary Surfaces* Proceedings of the 2002 IEEE International Conference on Robotics & Automation, Washington,DC May 2002

KNAPP M., *Mesh Decimation Using VTK* Institute of Computer Graphics and Algorithms Vienna University of Technology, 2002

LATOMBE J.-C., *Robot Motion Planning.* Kluwer Academic Publishers, Boston, MA. Norwell, Mass, 1991

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

NAGATANI K., *et al.*, *Toward Robust Sensor Based Exploration by Constructing Reduced Generalized Voronoi Graph.*

PARRISH JC., *Long Range Rovers for Mars Exploration and Sample Return.*

REKLEITIS I., *et al.*, *Graph-Based Exploration using Multiple Robots* Centre for Intelligent Machines, McGill University.

REKLEITIS I., *et al.*, *The Hierarchical Atlas* IEEE Transactions on Robotics, Vol 21, No. 3, June 2005

RUSSEL S.J., NORVIG P., *Artificial Intelligence, a Modern Approach* Prentice Hall. 1995.

SCHROEDER W.J., *et al.*, *Decimation of Triangle Meshes* Computer Graphics, 26, 2, July 1992

VOLPE R., *Enhanced Mars Rover Navigation Techniques.* In Proc. IEEE International Conference on Robotics and Automation, San Francisco (USA), pages 926–931, 2000. 24

ZIVKOVIC Z., *et al.*, *Hierarchical Map Building and Planning base on Graph Partitioning* Intelligent Systems Laboratory Amsterdam.