Localisation et cartographie simultanées pour un robot mobile équipé d'un laser à balayage : CoreSLAM

Oussama El Hamzaoui

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

MINES ParisTech

École doctorale n°432
Sciences des Métiers de l'Ingénieur

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

l'École nationale supérieure des mines de Paris

Spécialité « Informatique temps-réel, robotique et automatique »

présentée et soutenue publiquement par

Oussama El Hamzaoui

le 25/09/2012

Localisation et Cartographie Simultanées pour un robot mobile équipé d'un laser à balayage : CoreSLAM

~ ~ ~

Simultaneous Localization and Mapping for a mobile robot with a laser scanner : CoreSLAM

Directeur de thèse : **Claude Laurgeau**
Co-directeur de thèse : **Bruno Steux**

Jury

M. Laurent Trassoudaine, Professeur à l'Université Blaise Pascal de Clermont-Ferand	Rapporteur
M. David Filliat, Professeur à l'ENSTA ParisTech	Rapporteur
M. Philippe Bonnifait,, Professeur à l'Université de Technologie de Compiègne	Président du jury
M. Claude Laurgeau, Professeur à Mines ParisTech	Examinateur
M. Bruno Steux, Professeur à Mines ParisTech	Examinateur

T H È S E

MINES ParisTech
Centre de Robotique CAOR
60 Boulevard Saint Michel, 75272 Paris Cedex 06, France

Cette page est laissée vide intentionnellement

Abstract

Autonomous navigation is one of the main areas of research in the field of intelligent vehicles and mobile robots. In this field, we seek to create algorithms and methods that give robots the ability to move safely and autonomously in a complex and dynamic environment.

The algorithms that ensure the tasks of localization and mapping have an important place among the different algorithms used in this field. Indeed, without reliable information about the robot position (localization) and the nature of its environment (mapping), the other algorithms (trajectory generation, obstacle avoidance ...) cannot achieve their tasks properly.

Moreover, a robot needs to know its position to be able to create a map of the environment. On the other hand, it is imperative to have an already build map of the environment to locate itself. These two tasks are closely related. Without a map of the environment, or localization data, the robot should compute and estimate these values simultaneously. This is done through SLAM algorithms: Simultaneous Localization and Mapping.

Since the 1980s, several SLAM algorithms have been developed. The first and probably the most used is the EKF-SLAM (Extended Kalman Filter SLAM). Other methods based on particle filtering have been studied too. Each method has advantages and disadvantages, but the algorithms developed are generally complicated and have uncertain behavior.

In addition, several algorithms are based on a loop closure process to correct the position estimates and mapping. This kind of tasks requires the robot to do loops in the environment, to allow the SLAM algorithm to review its results and refine its estimates. This loop closure is not always a good idea, mainly in some cases where we cannot waste time to revisit some areas already explored

Under these constraints, we focused our work in the thesis on a specific problem: to develop a simple, fast and light SLAM algorithm that can **minimize localization errors** without loop closing.

At the center of our approach, there is an IML algorithm: Incremental Maximum Likelihood. This kind of algorithms is based on an iterative estimation of the localization and the mapping. It contains thus naturally a growing error in the localization process. The choice of IML is justified mainly by its simplicity and lightness.

The main idea of the work done during this thesis is built around the different tools and algorithms used to minimize the localization error of IML, while keeping its advantages.

The major contribution of this thesis is to develop a new algorithm for SLAM, called CoreSLAM. It is a **fast** and **lightweight** algorithm. It can work correctly on an embedded system, even with limited resources. CoreSLAM gives nearly the same SLAM quality than methods which use a loop closing algorithm. The use of IML keeps it simple and clear, easily understood and offers many opportunities for improvement.

Résumé

La thématique de la navigation autonome constitue l'un des principaux axes de recherche dans le domaine des véhicules intelligents et des robots mobiles. Dans ce contexte, on cherche à doter le robot d'algorithmes et de méthodes lui permettant d'évoluer dans un environnement complexe et dynamique, en toute sécurité et en parfaite autonomie.

Les algorithmes assurant les tâches de localisation et de cartographie occupent une place importante parmi les différents algorithmes utilisés dans le domaine de la navigation autonome. En effet, sans informations suffisantes sur la position du robot (localisation) et sur la nature de son environnement (cartographie), les autres algorithmes (génération de trajectoire, évitement d'obstacles ...) ne peuvent pas fonctionner correctement.

Par ailleurs, un robot a besoin de connaitre sa position pour pouvoir cartographier un environnement. D'un autre côté, il doit absolument disposer d'une carte préétablie de son environnement pour pouvoir s'y localiser. Ces deux tâches sont liées. Lorsqu'on ne dispose pas d'une carte de l'environnement, ni de données de localisation, le robot doit trouver ces informations simultanément. Cette opération est effectuée grâce aux d'algorithmes de SLAM : Simultaneous Localization and Mapping.

Depuis les années 1980, plusieurs méthodes de SLAM ont vu le jour. La plus importante, et sans doute la plus utilisée, reste le SLAM par filtre de Kalman étendu (EKF : Extended Kalman Filter). On peut aussi trouver des méthodes basées sur l'utilisation de filtres particulaires. Chaque méthode a des avantages et des inconvénients, mais les algorithmes développés restent en général assez compliqués, et leur comportement incertains.

En plus, plusieurs algorithmes se basent sur un processus de fermeture de boucle afin de corriger les estimations de position et de cartographie. Ce type d'opérations nécessite un retour du robot à des endroits déjà visités, permettant ainsi à l'algorithme de SLAM de revoir ses calculs et raffiner ses estimations. Ce bouclage n'est pas pratique dans certains cas, où on ne peut pas se permettre de perdre du temps à revisiter certains endroits déjà explorés

Sous ces contraintes, nous avons centré notre travail de thèse sur une problématique précise : développer un algorithme de SLAM simple, rapide, léger et **limitant les erreurs** au maximum. Au cœur de notre approche, on trouve un algorithme d'IML : Incremental Maximum Likelihood. Ce type d'algorithmes se base sur une estimation itérative de la localisation et de la cartographie. Il est ainsi naturellement divergent. Le choix de l'IML est justifié essentiellement par sa simplicité et sa légèreté.

La particularité des travaux réalisés durant cette thèse réside dans les différents outils et algorithmes utilisés afin de limiter la divergence de l'IML au maximum, tout en conservant ses avantages.

La contribution majeure de cette thèse est le développement d'un nouvel algorithme de SLAM, *CoreSLAM*. Il s'agit d'un algorithme **rapide** et **léger**. Il peut fonctionner sans aucun problème sur un système embarqué, même avec des ressources limitées. L'algorithme assure une qualité de SLAM comparable à celle obtenue avec des algorithmes utilisant une fermeture de boucle. L'utilisation de l'IML permet de garder une structure simple et claire,

facile à comprendre et offrant plusieurs possibilités d'amélioration.

Table des matières

Table des figures

Liste des tableaux

Liste des algorithmes

Liste des symboles

ALV	Autonomous Land Vehicles
CAROTTE	CArtographie par RObоT d'un TErritoire
CEKF	Compressed Extended Kalman Filter
EKF	Extended Kalman Filter
IML	Incrmental Maximum Likelihood
RBPF	Rao-Blackwellized Particle Filter
SIMD	Single Instruction Multiple Data
SISD	Single Instruction Single Data
SLAM	simultaneous Localization and Mapping
SLAMMOT	SLAM with Moving Object Tracking
SSE	Streaming SIMD Extensions
UKF	Unscented Kalman Filter

L'utilisation de l'information sensorielle pour localiser un robot dans son environnement et un problème fondamentale pour offrir à un robot mobile des capacités d'autonomie

Ingemar J. Cox

1 Introduction

Figure 1.1 – *Quelques applications de la robotique intelligente*

1.1 L'autonomie des véhicules

Avec un parc de robots personnels de plus de 5 millions d'unités, et un cumul des ventes de robots professionnels s'élevant à 13,2 milliards de dollars en 2009, la robotique de service va dépasser les $22 milliards en 2013 et subir sa plus rapide accélération dans les 3 prochaines années.

La robotique de service désigne les robots qui rendent service à l'humain en se substituant à lui. La variété des tâches accomplies par ces robots est immense : on retrouve des robots de service dans presque tous les milieux hostiles : spatial, sous-marins, nucléaire, déminage, défense, sécurité, etc. En les retrouve également en remplacement des hommes dans les situations pénibles, dans le bâtiment ou l'agriculture. Ces robots sont dits de type professionnels : ils sont en général très cher unitairement et produits en nombre d'unités restreint par modèle. On retrouve ensuite des robots de service jusque dans nos maisons sous la forme d'aspirateur ou tondeuse automatique, ou encore sous forme de jouets, ou kit pour l'éducation. Leur prix est très faible pour entrer en masse chez les particuliers. Le tableau de la figure 1.1 représente quelques exemples d'utilisations de la robotique intelligente.

Dans le domaine de la robotique de service, l'un des éléments les plus stratégiques est la question de la mobilité ou navigation, qui représente la capacité des robots de s'orienter, de se déplacer et de reconnaitre en temps réel leur environnement.

Pour que ces robots puissent accomplir leurs tâches efficacement, ils ont besoin d'un minimum d'intelligence artificielle. Cette intelligence leur offre généralement des capacités de cognition, de prise de décision et de manipulation de différents objets.

L'un des challenges majeurs reste la conception d'un système intelligent capable de se déplacer de manière autonome et sûre dans un environnement inconnu, dynamique et large, en se basant seulement sur les données de ses capteurs.

Un tel robot doit avoir des capacités de perception lui permettant d'obtenir des infor-

mations exactes et précises concernant sa propre position et les caractéristiques des objets qui l'entourent. Il doit donc accomplir deux tâches différentes, mais complémentaires : la localisation et la cartographie.

- La localisation : le robot utilise les données fournies par ses capteurs embarqués pour pouvoir trouver sa position dans une carte locale ou globale de son environnement.
- La cartographie : le robot crée une carte de son environnement en utilisant les données des capteurs. Pour cela, il a besoin de connaître sa position.

Ces deux tâches sont bien connues et il existe des solutions à chacune des deux. Mais lorsque le robot ne dispose ni d'une carte de l'environnement, ni de sa position, il doit effectuer ces deux tâches simultanément. Ce problème est appelé SLAM.

Le SLAM est l'acronyme de *Simultaneous Localization and Mapping*, ce qui veut dire : Localisation et Cartographie Simultanées. Ce nom désigne l'ensemble des algorithmes offrant à un véhicule mobile robotisé la possibilité de cartographier un terrain inconnu tout en assurant sa propre localisation simultanément.

1.2 Contexte de la thèse

L'idée principale du SLAM consiste à estimer l'état général du système robotisé. Cet état inclut la position du véhicule ainsi que celles des différents points de l'environnement.

Beaucoup de travaux de recherche ont tenté, et tentent toujours de résoudre ce problème. Il a d'abord été traité dans les années 1980 par Smith et Cheesman [6]. Leur approche se base sur l'utilisation d'un EKF : Extended Kalman Filter. Après l'extraction des points particuliers de l'environnement (les amers), le filtre de Kalman est utilisé pour trouver l'état du robot en estimant simultanément la position du robot ainsi que celles des amers.

Les méthodes basées sur l'EKF sont encore largement utilisées pour la résolution du SLAM. Ce type de méthodes permet d'avoir une estimation a postériori sur la carte des amers et les positions du robot. Néanmoins, l'EKF-SLAM souffre de quelques problèmes. Sa grande faiblesse réside dans le fait qu'il s'appuie sur **de fortes hypothèses** concernant le modèle du robot et le bruit des capteurs utilisés. En plus, l'EKF-SLAM utilise uniquement des **cartes d'amers**, ce qui nécessite un prétraitement des données des capteurs afin d'en extraire les amers. Cette opération d'extraction n'est pas toujours facilement faisable dans des **environnements non structurés**.

Un autre type de méthodes largement a été utilisée pour résoudre ce problème. Il se base sur le filtrage particulaire. Dans ce genre de méthodes, on utilise un ensemble de particules afin d'estimer l'état du robot. Cette technique permet d'éviter certains inconvénients du filtre de Kalman, en offrant plus de flexibilité. Mais le filtrage particulaire présente également quelques problèmes, notamment la **complexité** des algorithmes, la **diversité** des particules et les difficultés de **paramétrage**.

Nous avons ainsi cherché à explorer d'autres pistes pour résoudre ce problème. Notre principal objectif est d'avoir un algorithme **simple**, **rapide**, **efficace** et **robuste**.

Afin d'assurer la simplicité, nous nous sommes tournés vers un algorithme de maximisation de vraisemblance. Ce genre d'algorithmes estime l'état du robot le plus probable à un instant donné, en se basant sur l'ensemble des données des capteurs. Pour permettre à l'algorithme d'être rapide, nous avons opté pour une version **incrémentale** de l'estimation : l'algorithme n'utilise plus toutes les données des capteurs, mais seulement les informations de l'étape précédente des calculs. On travaille dans le cadre d'un algorithme d'estimation incrémentale du maximum de vraisemblance (*IML* : Incrmental Maximum Likelihood).

L'utilisation de l'IML dans le cadre de la résolution du SLAM a largement été délaissée par le passé. En effet, l'IML est **divergent par construction**. Ce qui fournit des résultats de SLAM biaisés. On peut éventuellement ajouter un algorithme de fermeture de boucle pour corriger cette divergence. Ce genre d'algorithme permet au robot de corriger l'erreur d'estimation de sa position lorsqu'il **revient** à un point qu'il a déjà visité. Ceci impose donc au véhicule de faire des boucles dans l'environnement, ce qui n'est pas toujours pratique dans des situations réelles [1].

L'objectif de la thèse est d'améliorer au maximum le fonctionnement de l'IML sans fermeture de boucle, afin d'obtenir une qualité de SLAM comparable aux algorithmes conventionnels, tout en gardant la simplicité et la rapidité des calculs.

1.3 Les principales contributions de la thèse

Durant cette thèse, nous avons travaillé sur le développement et l'amélioration d'un algorithme de SLAM basé sur l'estimation incrémentale du maximum de vraisemblance. L'utilisation d'une telle méthode induit un risque majeur : notre algorithme est divergent par construction. Ceci affecte la qualité du SLAM.

Nous avons intégré plusieurs techniques innovantes à l'algorithme afin de minimiser cette divergence au maximum.

- **Carte du SLAM** : la carte utilisée dans notre algorithme est spécialement conçue afin de rendre l'estimation de la position du robot plus rapide et plus efficace.
- **Robustesse** : l'algorithme intègre des techniques de **filtrage** et de **latence** lui permettant de s'assurer de la qualité de ses estimations avant de les inscrire dans la liste des résultats à produire.
- **Vitesse** : en plus de méthodes logicielles utilisées pour améliorer la vitesse de calcul, l'algorithme profite de l'accélération matérielle de certains processeurs dans les parties de calcul les plus critiques.

1. Pour un robot de sauvetage par exemple, on ne peut pas se permettre de faire des boucles dans l'environnement afin de déterminer avec précision les positions des personnes à secourir

1.4 Organisation du manuscrit

Dans ce document, nous allons d'abord présenter le problème du SLAM, sa formulation, ses difficultés, les étapes de sa résolution ... Nous nous intéresserons ensuite aux solutions les plus connues pour ce problème. Dans le troisième chapitre, nous allons commencer par une présentation des capteurs utilisés durant les différentes phases de développement et de test qui ont accompagnées l'évolution de notre solution au problème du SLAM que nous avons d'abord appelé *tinySLAM* puis *CoreSLAM*. Nous expliquerons ensuite en détails les bases de fonctionnement de l'algorithme et les principales idées lui permettant d'accomplir sa tâche.

L'idée de base du sujet de thèse est de ramener la qualité du SLAM à **son maximum en limitant la dérive**. Ainsi, nous présenterons dans le quatrième chapitre les différents outils d'optimisation qui nous ont permis d'atteindre cet objectif. Nous détaillerons chacune des techniques et des méthodes utilisées.

Dans le chapitre 5, nous allons présenter le cadre expérimental des opérations de validation et de vérification des solutions proposées. On finit dans le sixième chapitre par une présentation et une analyse des résultats obtenus durant la thèse.

Dis-moi et j'oublierai, montre-moi et je me souviendrai, implique-moi et je comprendrai

Confucius

2

SLAM : présentation générale

Dans ce chapitre, nous allons présenter le problème du SLAM de manière générale, afin de mieux situer le contexte du travail de la thèse. Nous commençons d'abord par les origines et la formulation initiale du problème, ce qui nous ramène à la description probabiliste du SLAM faite en 1986 [6]. Nous présentons ensuite le cadre mathématique de traitement du problème, avec quelques formules clés qui lui sont liées. Dans les sections 2.2.1 et 2.2.2, nous allons présenter différentes méthodes de résolution du SLAM ainsi que des représentations de la carte de l'environnement d'évolution du robot. Nous concluons chaque section pour une brève comparaison entre les éléments présentés.

2.1 Définitions

2.1.1 Les origines

Le problème de localisation et cartographie simultanées (SLAM) traite deux questions importantes dans la robotique mobile. La première question est : *"Où suis-je ?"*. La réponse à cette question définit la localisation du robot. La deuxième question concerne les caractéristiques de l'environnement du robot : *"À quoi ressemble l'environnement où je me trouve ?"*.

Dans un système de SLAM, un véhicule robotisé placé dans un environnement inconnu, dans une position inconnue, doit construire la carte de l'environnement tout en essayant de se localiser par rapport à cette carte. Le robot dispose de plusieurs capteurs qui l'aident à récupérer les informations dont il a besoin. La réalisation de cette tâche peut paraître impossible dans la mesure où le robot a besoin d'une carte pour se localiser, mais en même temps il doit connaître sa position (se localiser) pour pouvoir construire la carte. Afin de faciliter le traitement de cette relation "poule-œuf [1]", les scientifiques ont unifié les deux questions précédentes en une seule question : *"Où suis-je susceptible d'être dans la carte la plus probable du monde que j'ai observé jusqu'à maintenant ?"*.

La formulation du problème de SLAM ainsi a permis de définir un cadre de résolution probabiliste. L'émergence du SLAM probabiliste a certainement été durant la conférence *IEEE Robotics and Automation Conference* en *1986* à San Francisco, California. Plusieurs chercheurs tentaient d'appliquer des méthodes théoriques d'estimation au problème de localisation et cartographie.

Les travaux de Smith et Cheeseman [6] et de Durant-Whyte [7] constituent une base des méthodes statistiques de description des relations entre les positions d'amers dans un environnement et l'estimation de l'incertitude géométrique de la carte. L'un des éléments clés de ces travaux traite du degré de corrélation entre les estimations des positions des amers dans une carte.

L'intérêt porté aux problématiques du SLAM a augmenté de manière exponentielle pendant les dix dernières années, notamment grâce aux conférences internationales (ICRA, IROS...) qui attirent de plus en plus la communauté scientifique.

2.1.2 Formulation du problème de SLAM

Le SLAM est composé d'un ensemble de méthodes permettant à un robot de construire une carte d'un environnement et en même temps de se localiser en utilisant cette carte. La trajectoire du véhicule et la position des amers dans la carte sont estimées au fur et à mesure, sans avoir besoin de connaissances *a priori* [2].

Considérons un robot se déplaçant dans un environnement inconnu, en observant un certain nombre d'amers grâce à un capteur embarqué sur le robot. La figure 2.1 montre

1. Qu'est-ce qui est apparu en premier : l'œuf ou la poule ?
2. Une grande partie de cette section est une adaptation de l'excellent tutorial [1]

une illustration du problème.

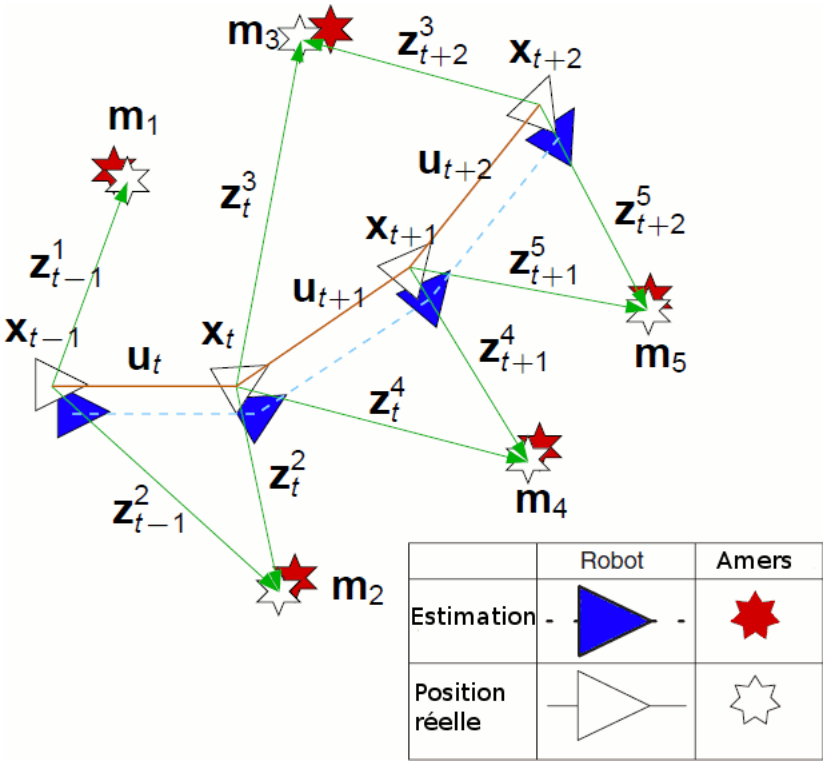


Figure 2.1 – *L'idée de base du SLAM [1]*

A l'instant k on définit les quantités suivantes :

- x_k : le vecteur d'état. Il contient la position du robot
- u_k : le vecteur de contrôle. L'application de u_k à l'instant $k-1$ mène le robot de l'état x_{k-1} à l'état x_k
- m_i : vecteur contenant la position de l'amer i.
- z_k : l'observation à l'instant k

On définit aussi les ensembles suivants :

- $X_{0:k} = \{x_0, x_1, ..., x_k\} = \{X_{0:k-1}, x_k\}$: l'ensemble des vecteurs d'état jusqu'à l'instant k
- $U_{0:k} = \{u_0, u_1, ..., u_k\} = \{U_{0:k-1}, u_k\}$: l'ensemble des vecteurs de commande jusqu'à l'instant k
- $Z_{0:k} = \{z_0, z_1, ..., z_k\} = \{Z_{0:k-1}, z_k\}$: l'ensemble des observations jusqu'à l'instant k
- $m = \{m_1, m_2, ..., m_n\}$: la carte de l'environnement contenant une liste d'objets statiques

2.1.2.1 La localisation

Le problème de localisation du robot consiste à estimer sa position dans un environnement donné, en utilisant l'historique de ses observations, l'historique des commandes et la connaissance de l'environnement. La figure 2.2 schématise ce principe.

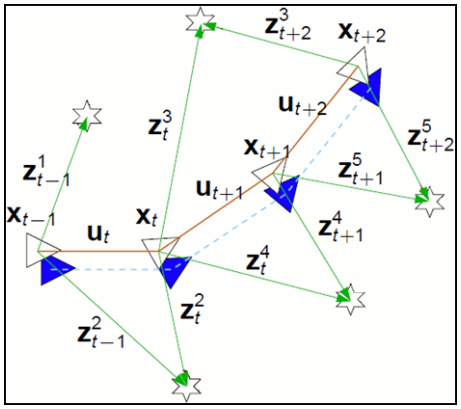


FIGURE 2.2 – *La localisation : le système cherche à estimer sa position en utilisant les informations sur l'environnement dont il dispose*

On peut analytiquement représenter cette opération par l'estimation de la probabilité de distribution :

$$P(x_k|Z_{0:k}, U_{0:k}, m)$$

L'estimation d'une telle quantité définit la localisation globale, dans la mesure où on utilise toutes les données de l'historique des observations et des commandes pour estimer la position. On obtient ainsi une estimation robuste de la position a postériori, mais on augmente largement la complexité des calculs.

Afin de simplifier l'algorithme, on peut définir une localisation locale, où on utilise uniquement les données de l'instant $(k-1)$ pour estimer la position à l'instant k. On représente analytiquement cette opération par l'estimation de la distribution de probabilité :

$$P(x_k|z_{k-1}, u_{k-1}, x_{k-1}, m)$$

En utilisant cette méthode, on simplifie largement la complexité de l'algorithme, mais on risque de dévier de la position correcte du robot, sans pouvoir corriger cela.

2.1.2.2 La cartographie

Le problème de cartographie consiste à déterminer la carte d'un environnement, en utilisant les données des capteurs et l'historique des positions réelles du robot. Sur le schéma de la figure 2.3, le système connait sa position exacte et estime la carte de l'environnement en utilisant les données de ses capteurs.

On peut exprimer cela analytiquement ainsi :

$$P(m_k|Z_{0:k}, X_{0:k})$$

Les positions réelles du robot peuvent être obtenues en utilisant des balises dans un environnement interne ou un récepteur GPS en externe. Ces positions doivent être précises et correctes afin d'obtenir une bonne cartographie.

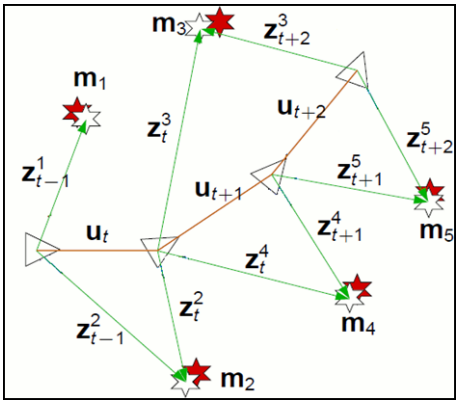


FIGURE 2.3 – *La cartographie : le système crée la carte de l'environnement en se basant sur sa position connue et les informations de ses capteurs*

2.1.2.3 Le SLAM

La formulation probabiliste du problème de SLAM nécessite le calcul, à chaque instant k, de la quantité de probabilité :

$$P(x_k, m|Z_{0:k}, U_{0:k}, x_0)$$

Ce calcul est généralement effectué récursivement. On commence par la probabilité $P(x_{k-1}, m|Z_{0:k-1}, U_{0:k-1})$, puis on utilise le théorème de Bayes pour déduire la quantité $P(x_k, m|Z_{0:k}, U_{0:k})$ à partir de z_k et u_k. Afin d'effectuer cette déduction, nous avons besoin de connaître $P(x_k|x_{k-1}, u_k)$ et $P(z_k|x_k, m)$. Le terme $P(z_k|x_k, m)$ désigne le **modèle d'observation**. Il définit la probabilité d'avoir une mesure z_k connaissant l'état du véhicule x_k et une carte de l'environnement m. Le terme $P(x_k|x_{k-1}, u_k)$ définit le **modèle de transition** (modèle de mouvement du véhicule robotisé). Il permet de prévoir l'état x_k du système, qui ne dépend que de l'état précédent x_{k-1} et de la commande de contrôle appliquée. Dans ce cas, le processus de transition entre les états du système x_k est dit Markovien.

On définit donc le problème du SLAM en deux parties par les équations 2.1 et 2.2 :

– Une partie de mise-à-jour de la position :

$$P(x_k, m|Z_{0:k-1}, U_{0:k}, x_0) = \begin{array}{l} \int [P(x_k|x_{k-1}, u_k) \\ *P(x_{k-1}, m|Z_{0:k-1}, U_{0:k-1}, x_0)]dx_{k-1} \end{array} \quad (2.1)$$

– Une partie de mise-à-jour de l'observation :

$$P(x_k, m|Z_{0:k}, U_{0:k}, x_0) = \frac{P(z_k|x_k, m) * P(x_k, m|Z_{0:k-1}, U_{0:k}, x_0)}{P(z_k|Z_{0:k-1}, U_{0:k})} \quad (2.2)$$

En utilisant ainsi l'estimation *a posteriori* à l'instant $(k-1)$ donnée par le terme

$P(x_{k-1}, m|Z_{0:k-1}, U_{0:k-1}, x_0)$, on peut calculer la prédiction (équation 2.1) et déduire ensuite l'estimation *a posteriori* à l'instant k. On a donc :

$$\begin{aligned} P(x_k, m|Z_{0:k}, U_{0:k}, x_0) = \quad & \eta * P(z_k|x_k, m) \\ & * \int [P(x_k|x_{k-1}, u_k) \qquad * \\ & * P(x_{k-1}, m|Z_{0:k-1}, U_{0:k-1}, x_0)] dx_{k-1} \end{aligned} \tag{2.3}$$

Sachant que $\eta = \frac{1}{P(z_k|Z_{0:k-1}, U_{0:k})}$ est une constante de normalisation dépendant du modèle d'observation et du modèle de transition.

Cette structure du SLAM est représentée sur le schéma de la figure 2.4. Sur ce schéma, les cercles gris représentent les données connues, tandis que les cercles blancs désignent les quantités à estimer.

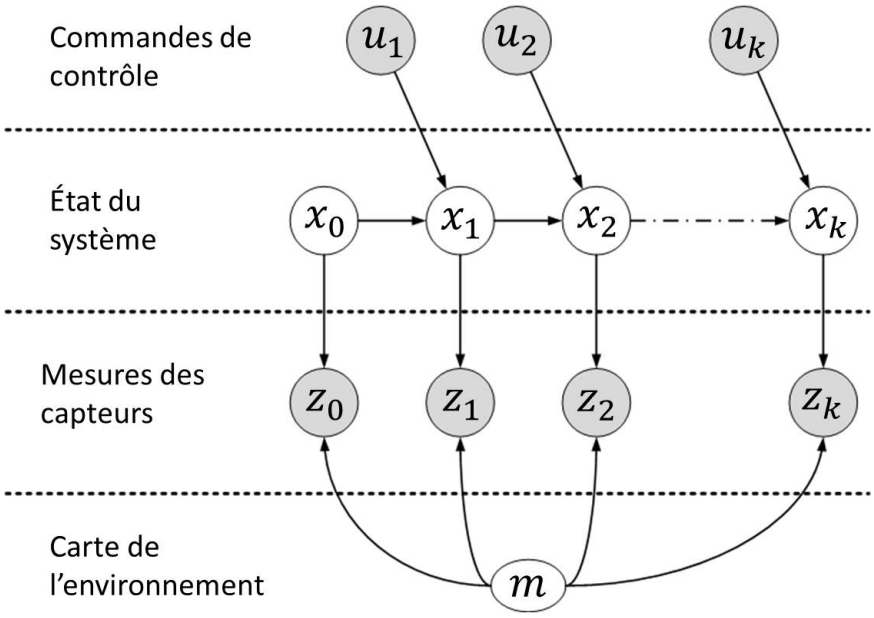


Figure 2.4 – *Représentation graphique du problème de SLAM*

2.2 Résolution du SLAM

De nombreuses recherches tentent de résoudre le problème de la localisation et la cartographie simultanées, communément appelé SLAM. Les principales méthodes de SLAM sont basées sur des méthodes d'estimation statistiques. Il s'agit de filtrage statistique permettant d'estimer l'état d'un système dynamique à partir des données en provenance d'un ou plusieurs capteurs. On cherche à connaître l'état courant qui correspond le mieux aux données récupérées et, éventuellement, aux informations a priori dont on dispose. Les algorithmes développés peuvent être classés selon plusieurs critères : les types de capteurs utilisés, les méthodes de calcul adoptées, les types de cartes représentant l'environnement ... On trouve ainsi des algorithmes basés sur la vision par ordinateur en utilisant une caméra [8] ou plusieurs caméras [9], et des algorithmes qui utilisent un (ou plusieurs) capteur

laser ou sonar [10]. Concernant les méthodes de calcul, deux grandes familles existent. D'une part, les algorithmes basés sur l'utilisation du filtrage de Kalman [11], et d'autre part, on trouve des algorithmes utilisant des filtres à particules [12], ou des filtres particulaires Rao-Blackwellizés - un mélange de filtrage particulaire et de filtrage de Kalman - comme FastSLAM [13].

On peut également classifier les systèmes de SLAM suivant l'environnement de travail. On trouve ainsi le SLAM terrestre à l'intérieur et à l'extérieur, le SLAM aérien ainsi que le SLAM sous-marin. La majorité des travaux de recherche se concentre sur le SLAM par des systèmes robotisés terrestres dans des environnements internes ([14], [15] et [16]), mais de plus en plus de travaux traite le cas de robots aériens ([17] et [18]) ou sous-marins ([19], [20] et [21]).

Certaines recherches ont porté sur la comparaison des différents algorithmes au niveau des performances et de la vitesse de calcul [22]. Les résultats de ces comparaisons montrent que le problème **n'est toujours pas universellement résolu**.

2.2.1 Représentation de la carte

Le choix de la représentation de la carte de l'environnement est une étape importante dans le SLAM. Dans [3] l'auteur analyse trois approches fondamentales de représentation de l'environnement :

- L'approche directe [23]
- L'approche basée sur les caractéristiques géométriques (*feature-based*) [24]
- L'approche basée sur une grille d'occupation (*grid-based*) [25]

La carte de l'environnement peut aussi être représentée par une approche topologique. Mais cette méthode n'est pas analysée, dans la mesure où elle est basée sur un partitionnement des cartes de types feature-based ou grid-based en régions cohérentes.

2.2.1.1 L'approche directe

La méthode de représentation directe de la carte de l'environnement est généralement adaptée à **l'utilisation des capteurs Laser**. Cette méthode utilise les données brutes des mesures du capteur pour représenter l'environnement sans aucune extraction d'amers ou de caractéristiques prédéfinies (lignes, coins ...).

Dans le cas d'un capteur laser, chaque mesure est constituée d'un ensemble de points d'impact du faisceau laser sur les objets de l'environnement. On peut ainsi construire une carte simplement en superposant les différents points de mesure. On obtient ainsi une représentation en nuage de points. La figure 2.5 montre un exemple d'une carte en nuage de points.

2.2.1.2 L'approche *feature-based*

Cette méthode réduit les données des mesures en caractéristiques prédéfinies. On utilise généralement des primitives géométriques, comme des lignes, des cercles ou des coins. La

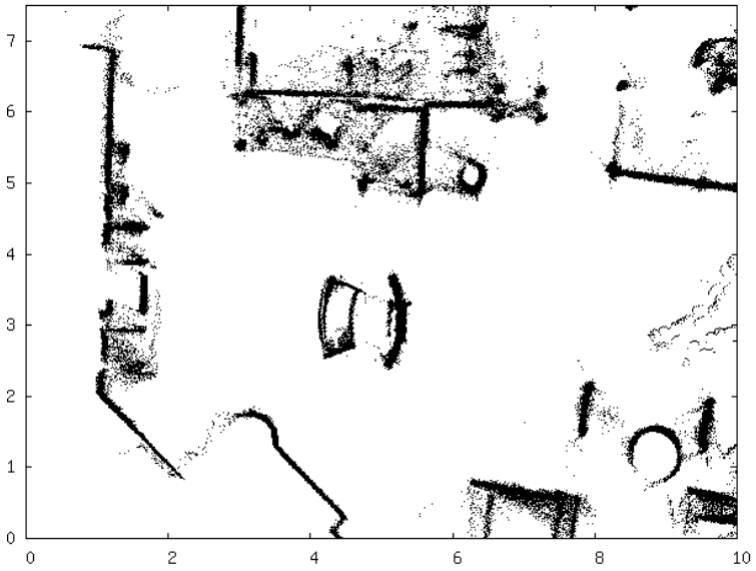


FIGURE 2.5 – *Carte en nuage de points*

création de la carte consiste ensuite à estimer les paramètres des primitives afin qu'ils correspondent au mieux aux observations.

Pour détecter les caractéristiques géométriques, plusieurs méthodes existent. Les plus connues sont :

- La méthode split-and-merge [26] pour la détection des segments de lignes
- La transformation de Hough [27] pour la détection des lignes ou des cercles
- RANSAC [28] pour la détection des lignes ou des cercles aussi

La figure 2.6 montre un exemple d'une carte feature-based. Ce type de cartes est limité aux objets et formes modélisés et prédéfinis. Il est donc incompatible avec les environnements **trop complexes** et **non structurés**.

Contrairement à la méthode directe de représentation, où on reproduit fidèlement la structure de l'environnement, les approches feature-based sont des approximations seulement.

2.2.1.3 L'approche *grid-based*

Les grilles d'occupation ont été introduites par [25]. Dans cette représentation, l'environnement est divisé en cellules rectangulaires. La résolution de la représentation de l'environnement dépend directement de la taille des cellules. En plus de cette discrétisation de l'espace, une mesure de probabilité d'occupation est estimée pour chaque cellule indiquant si celle-ci est occupée ou non.

La mise à jour de l'état de chaque cellule se fait à la réception de nouvelles données. On trouve dans la littérature plusieurs méthodes pour réaliser cette opération :

- Le filtrage bayésien ([29] et [30]) : cette méthode a notamment été utilisée par Thrun dans [31] afin de modéliser la connaissance sur l'état de la cellule. Dans cette approche, on attribue à chaque cellule une probabilité entre 0 et 1. Une probabilité

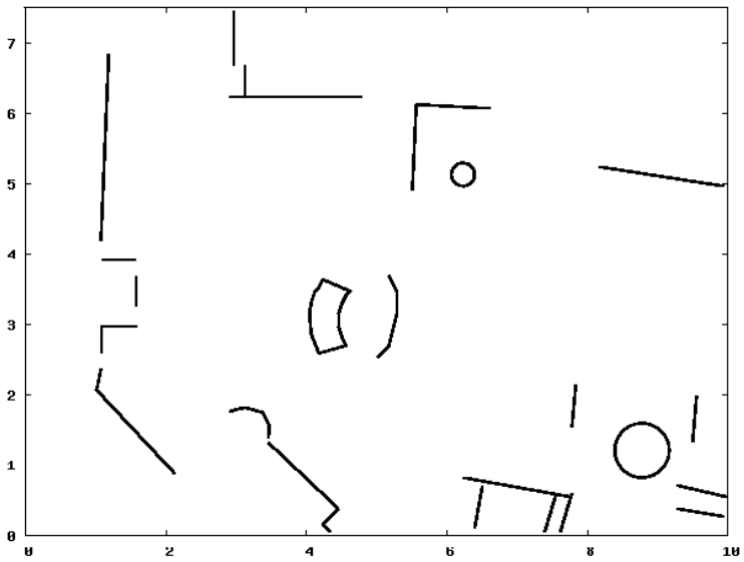


FIGURE 2.6 – *Carte basée sur l'extraction de caractéristiques géométriques de l'environnement*

de 0 signifie qu'on a la certitude que la cellule est libre. La probabilité de 1 signifie qu'on a la certitude qu'elle est occupée.

- La théorie de Dempster-Shafer [32] : comme indiqué dans [33], dans cette approche on associe à chaque cellule deux poids probabiliste, P_f et P_e. P_f est une mesure de l'importance des informations fournies par les capteurs extéroceptifs qui vont dans le sens de l'hypothèse « la cellule est occupée ». Pe mesure l'importance des informations contraires. Pf et Pe varient entre 0 et 1, et leur somme est toujours inférieur ou égale à 1. Ainsi, un couple $(Pf, Pe) = (0, 0)$ indique l'absence d'information sur la cellule (c'est la valeur d'initialisation de la carte), alors que le couple $(Pf, Pe) = (1, 0)$ par exemple indique que la cellule est occupée avec certitude.
- La logique floue [34] : l'état d'occupation de la cellule est modélisé par un ensemble flou [35]. Chaque cellule peut exprimer à la fois deux états partiels ($E = \textit{vide}$ et $O = \textit{occupée}$) et le degré d'appartenance entre eux se détermine en utilisant la théorie des possibilités. Dans cette approche, chaque cellule peut avoir des données conflictuelles ($E \cap O$) fournies par le capteur, elle sera considérée comme une cellule ni libre ni occupée. Afin d'éliminer l'ambiguïté sur l'état de ce type de cellule, on a besoin de plus de données en provenance des capteurs.

La figure 2.7 présente un exemple d'une carte basée sur une grille d'occupation.

Les cellules en blanc représentent l'espace libre, tandis que les cellules en noir indiquent la présence d'obstacles. L'approche basée sur une grille d'occupation est efficace lorsqu'il s'agit de modéliser l'incertitude ou fusionner les mesures de différents capteurs.

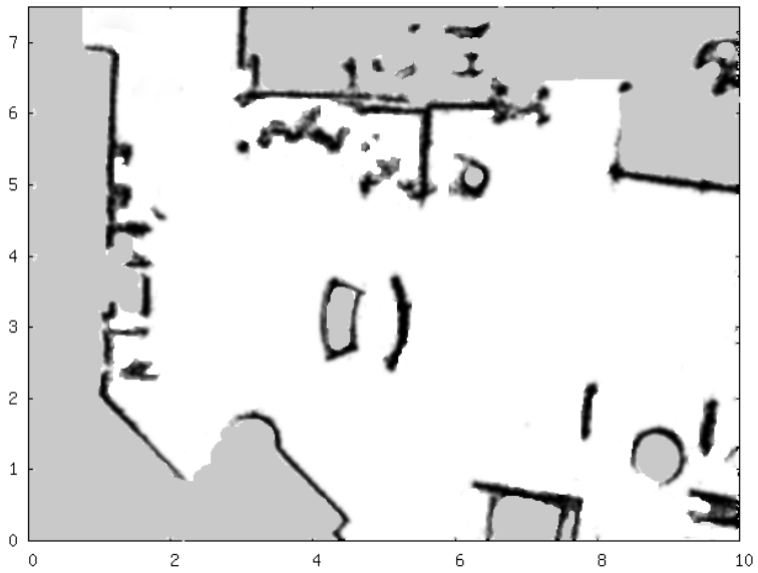


Figure 2.7 – *Carte à base d'une grille d'occupation*

2.2.1.4 Comparaison entre les types de cartes

Malgré sa simplicité, l'approche directe peut représenter tous les types des environnements. Mais elle présente l'inconvénient d'une grande consommation de mémoire et d'un manque de précision concernant la représentation de l'incertitude dans les mesures des capteurs.

Les cartes feature-based constituent une représentation compacte de l'environnement. Elles sont néanmoins basées sur l'extraction de caractéristiques connues et prédéfinies, ce qui limite leur utilisation aux environnements structurés et internes.

Les grilles d'occupation utilisent aussi une grande quantité de mémoire, mais elles offrent la possibilité de représenter tous les types d'environnements avec une prise en charge des caractéristiques des capteurs. Ce type d'approches est le mieux adaptés aux **capteurs de profondeur** comme les **lasers** ou les sonars.

Afin de réduire la consommation mémoire des méthodes basées sur les grilles d'occupation, on peut suivre une approche similaire à celle présentée dans [36]. L'idée principale consiste à construire la carte localement dans la zone de capture du laser. On concatène ensuite les cartes locales afin de construire la carte globale.

Le tableau 2.1, extrait de [3], résume la comparaison entre les différents types de représentations de l'environnement.

2.2.2 Algorithmes

Nous allons présenter dans cette partie trois méthodes largement utilisées pour la résolution du problème de SLAM. La majorité des autres méthodes et algorithmes en dérivent. Le premier exemple est le SLAM par Filtre de Kalman Etendu (EKF). C'est la plus ancienne méthode, encore largement utilisée. Le deuxième exemple utilise des techniques

	Direct	*Feature-based*	*Grid-based*
Compression des données	Ø	✓	Ø
Représentabilité de l'environnement	✓	Ø	✓
Gestion de l'incertitude	Ø	✓	✓
Caractéristiques du capteur	Ø	Ø	✓

Table 2.1 – *Comparaison entre les différents types de représentation de l'environnement [3]*

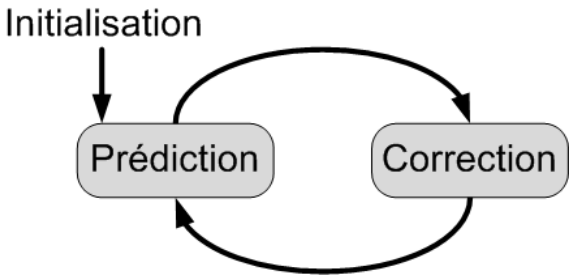


Figure 2.8 – *Le cycle du filtre de Kalman basé sur les deux étapes récursives :* Prédiction *et* Correction

de filtrage statistique qu'on nomme généralement des filtres particulaires. La troisième méthode présentée se base sur la recherche du maximum de vraisemblance (Maximum Likelihood).

2.2.2.1 Filtre de Kalman

Dans [37], Kalman propose une solution récursive au filtrage des données linéaires. Cette méthode, améliorée ensuite par Kalman lui-même et Bucy, ouvre de nouvelles pistes de recherche dans le domaine de la navigation autonome des robots mobiles. L'approche de base du filtre de Kalman est basée sur un cycle récursif nécessitant trois hypothèses pour assurer un fonctionnement, prouvé théoriquement, optimal (voir figure 2.8) :

- Un modèle d'évolution linéaire du système
- Une relation linéaire entre l'état et les mesures
- Un bruit blanc gaussien

Le cycle du filtre de Kalman est constitué de deux étapes fondamentales :

- Étape de prédiction : durant laquelle on estime l'état du système à l'instant t en utilisant l'estimation corrigée de l'instant (t-1)
- Étape de correction : durant laquelle on corrige l'estimation de l'état du système à l'instant t en utilisant les informations sensorielles reçu à l'instant t

Le filtre de Kalman Étendu est une amélioration du filtre de Kalman basique qui consiste à linéariser les modèles non linéaires afin que les conditions requises pour appliquer le filtre de Kalman soient satisfaites. La première implémentation de l'EKF a été faite par Stanley Schmidt dans le cadre du programme spatial Apollo. D'ailleurs, l'ancien nom du filtre était *le filtre de Kalman-Schmidt.*

Dans le cadre du SLAM, cette méthode a été introduite dans [6] et [38]. Dans ces articles, on trouve pour la première fois l'estimation en un seul processus de la position du robot et de la localisation des amers dans l'environnement. Les erreurs dans l'estimation sont représentées par une matrice de covariance mise à jour régulièrement en utilisant un filtre de Kalman Étendu [39] à chaque fois que le robot change de position. La taille de cette matrice grandit quadratiquement au fil des observations du robot.

Dans un algorithme de SLAM par EKF, on décrit le modèle de transition sous la forme :

$$P(x_k|x_{k-1}, u_k) \Longleftrightarrow f(x_{k-1}, u_k) + w_k \tag{2.4}$$

Dans l'équation 2.4, f représente le modèle du véhicule robotisé et $w_k \sim \aleph(0, Q_k)$ est un bruit gaussien de moyenne nulle et de variance Q_k.

Le modèle d'observation se présente sous la forme :

$$P(z_k|x_k, m) \Longleftrightarrow h(x_k, m) + v_k \tag{2.5}$$

où h décrit le modèle d'observation et $v_k \sim \aleph(0, R_k)$ un bruit gaussien de moyenne nulle et de variance R_k.

L'annexe B décrit plus en détails les formulations mathématiques du KF et de l'EKF. Le travail de thèse de Csorba [40] est consacré au SLAM pour EKF, et permet d'avoir plus de détails.

La convergence d'un filtre de Kalman est prouvée analytiquement dans le cadre linéaire [41] Mais elle n'est pas toujours réalisable dans des cas réel où les données sont fortement non linéaires. Ce problème apparaît également avec un EKF ou un UKF [3] (Unscented Kalman Filter [42]).

Une implémentation d'un filtre de Kalman évolue généralement quadratiquement en $O(n^2)$ (où n est le nombre d'amers de la carte) dans le temps et l'utilisation des ressources mémoire du système. Ainsi, avec l'évolution du système, l'algorithme atteindra un point où il ne pourra pas mettre à jour sa carte en temps réel. Ce problème vient du fait que chaque amer de l'environnement est corrélé à tous les autres. Cette corrélation se justifie par le fait que l'observation de chaque nouvel amer est faite par les capteurs du robot, l'erreur de localisation de l'amer est ainsi liée à l'erreur de localisation du robot lui-même et aux erreurs des autres amers dans la carte.

Afin de réduire ces exigences en puissance du matériel, le filtre de Kalman étendu compressé (CEKF) [43] a été introduit. Il traite et maintient les informations liées à un espace local avec un coût de calcul proportionnel au carré du nombre d'amer de la carte locale. Ces informations sont ensuite transférées à la carte globale d'une manière similaire à l'algorithme ordinaire, mais en une seule itération. Le CEKF réduit l'utilisation de mémoire, mais nécessite la détection d'amers robustes et souffre du problème d'association

3. L'UKF n'est applicable que dans le cas de bruits gaussiens, et nécessite plus de puissance de calcul que l'EKF

des données. Ce problème est renforcé par l'inconsistance due aux différentes approximations de linéarisation introduites dans le filtrage de Kalman. Plusieurs recherches ont ainsi tenté de proposer des extensions de cette méthode, permettant d'améliorer l'association des données. On trouve notamment quelques travaux de Burgard, Fox et Thrun (voir [44] et [45]) utilisant des techniques de statistiques avancées comme l'algorithme d'Espérance-Maximisation de Dempster [46]. Mais cela engendre encore plus de calculs et augmente la complexité de l'algorithme.

2.2.2.2 Filtre particulaire

Un filtre particulaire est un filtre récursif qui permet d'estimer l'état a postériori en utilisant un ensemble de particules. Les origines de ce type de filtres remontent à [47], mais le rapprochement entre le filtrage particulaire et le monde du SLAM est apparu avec les articles [48] et [49]

Contrairement aux filtres paramétriques comme le filtre de Kalman, un filtre particulaire représente une distribution par un ensemble d'échantillons créés à partir de cette distribution. Un filtre particulaire est ainsi capable de traiter les systèmes fortement non linéaires avec un bruit non gaussien.

La complexité des calculs du filtrage particulaire augmente exponentiellement avec le nombre d'amers de l'environnement, ce qui constitue un problème majeur dans le cadre d'une application temps-réel. Afin de résoudre ce genre de problèmes, certains travaux de recherche combinent le filtrage particulaire avec d'autres méthodes. C'est le cas des travaux de Montemerlo dans FastSLAM [13].

L'algorithme FastSLAM décompose le problème du SLAM en deux parties : un problème de localisation du robot et une collection de problèmes d'estimation d'amers liés à l'estimation de la position du robot. Dans cette configuration, chaque particule se charge de l'association des données locales qui lui sont liées. Par comparaison, un filtre EKF traite une seule hypothèse d'association de données pour tout le filtre. FastSLAM nécessite ainsi moins de mémoire et de temps de calcul que l'EKF.

L'utilisation du filtrage particulaire souffre également des difficultés rencontrées lors de la définition du nombre de particules. En effet, la qualité de l'estimation est fortement corrélée à la discrétisation de l'espace de recherche. Mais il est difficile de trouver un nombre optimal de particules.

2.2.2.3 Maximum de Vraisemblance

Alors que qu'un filtre particulaire ou un filtre de Kalman constituent des solutions probabilistes du problème de SLAM, la recherche incrémentale du maximum de vraisemblance (notée IML pour Incremental Maximum Likelihood) est une **approche d'optimisation**, où on teste plusieurs hypothèses à la recherche de celle qui maximise la vraisemblance.

Contrairement aux différentes déclinaisons de filtre de Kalman ou des approches à maximisation globale de vraisemblance (Expectation Maximization [46]), qui essaient d'établir une estimation a posteriori sur les positions du robot et sur la carte, l'idée de l'IML est de

construire une seule carte incrémentalement à chaque réception des données des capteurs sans garder un suivi de l'incertitude. Ce principe assure la simplicité de l'IML, qui reste son grand avantage comparé aux autres méthodes de SLAM.

Mathématiquement, l'idée de base de l'IML est de maintenir une série de cartes ($\hat{m}_1, \hat{m}_2, \ldots$)[4]et une série de positions du robot ($\hat{x}_1, \hat{x}_2, \ldots$) maximisant la vraisemblance. La position du robot à l'instant k est calculée en utilisant l'estimation à l'instant $k-1$ en maximisant la vraisemblance :

$$\langle \hat{x}_k, \hat{m}_k \rangle = \underset{x_k, m_k}{argmax} \{ P(z_k | x_k, m_k) * P(x_k, m_k | u_k, \hat{x}_{k-1}, \hat{m}_{k-1}) \} \tag{2.6}$$

La carte de l'environnement m_k est construite lorsque la position x_k est connue. Ainsi, en pratique, il suffit de chercher dans l'espace des positions du robot. Afin de déterminer la position $\hat{x}_k$ maximisant la vraisemblance, l'IML nécessite simplement une recherche dans l'espace de toutes les positions x_k lorsque l'algorithme reçoit de nouvelles données des capteurs :

$$\hat{x}_k = \underset{x_k}{argmax} \{ P(z_k | x_k, \hat{m}_{k-1}) * P(x_k | u_k, \hat{x}_{k-1}) \} \tag{2.7}$$

Dans l'équation 2.7, le terme $P(z_k | x_k, \hat{m}_{k-1})$ décrit la probabilité d'observer les dernières mesures z_k des capteurs en utilisant la carte $\hat{m}_{k-1}$ construite à l'étape $k-1$ et la position du robot x_k. Le terme $P(x_k | u_k, \hat{x}_{k-1})$ représente la probabilité que le système soit à l'état x_k en supposant connu l'état $\hat{x}_{k-1}$ et la commande u_k. Le résultat $\hat{x}_k$ trouvé est utilisée afin de mettre à jour la carte en utilisant les données correspondantes z_k :

$$\hat{m}_k = \hat{m}_{k-1} \cup \langle \hat{x}_k, z_k \rangle \tag{2.8}$$

La maximisation de l'équation 2.7 revient à trouver la position x_k du robot qui permet de satisfaire le modèle de mouvement du véhicule assurant une meilleure concordance entre les données des capteurs z_k et la carte m_{k-1}. Dans les différents travaux de recherche portant sur le SLAM par recherche du maximum de vraisemblance, on a souvent recourt aux méthodes de mise en correspondance des scans Laser (scan-matching).

Ces méthodes diffèrent selon la représentation de la carte choisie (section 2.2.1). On peut ainsi utiliser un scan-matching direct [23], se baser sur les caractéristiques géométriques de l'environnement [50] ou profiter de la grille probabiliste de la carte [51] (plus de détails dans la section 3.4). L'une des techniques les plus utilisées dans le cadre du scan-matching est l'ICP (pour Iterative Closest Point) [10].

4. Attention : la notation m_k utilisée ici est différente de la définition donnée dans 2.1.2. Dans la section 2.1.2, m_k désigne l'amer k de la carte, alors que dans cette section, elle désigne la carte construite à l'étape k.

La simplicité et la rapidité du SLAM par recherche du maximum de vraisemblance permet de construire des cartes de l'environnement en temps réel, mais cette approche ne peut pas garder une notion d'incertitude dans les estimations. En plus, la nature incrémentale de l'algorithme limite les traitements sur une seule étape de calcul et néglige l'ensemble des données capturées dans les autres étapes (contrairement au filtre de Kalman par exemple). Lorsqu'une position x_k du robot et une carte m_k ont été déterminées, elles sont fixées et ne peuvent plus être changées ou corrigées en utilisant les prochaines données (cas de fermeture de boucle -Section7.2.2- par exemple). L'erreur d'estimation de la position x_k peut ainsi grandir sans limite. Afin de corriger de problème, Hähnel [52] propose d'utiliser un arbre d'associations pour suivre plusieurs hypothèses de la carte de l'environnement. Si cette idée peut améliorer la qualité des résultats de l'algorithme, elle risque de nécessiter plus de charges de calcul, ce qui peut freiner l'utilisation temps-réel de l'IML dans le SLAM.

2.2.2.4 Comparaison entre les algorithmes

Le tableau 2.2 présente une liste d'avantages et d'inconvénients de certaines méthodes et algorithmes de SLAM ([3] et [53]).

	Avantages	***Inconvénients***
Filtre de Kalman	- Prise en compte des incertitudes - Convergence prouvée - Fermeture de boucle	- Hypothèses fortes - Complexité - Problème d'association des données - Utilisation possible sur un seul type de cartes
Filtre particulaire	- Élimination des hypothèses du KF - Fermeture de boucle	- Complexité - Paramétrage difficile
IML	- Concept simple et rapide - Tous les types de cartes	- Divergent par construction - Pas de fermeture de boucle

Table 2.2 – *Tableau comparant les avantages et les inconvénients de quelques méthodes de SLAM*

L'intérêt majeur des approches basées sur le filtrage de Kalman est la possibilité d'estimer l'état du système a posteriori en se basant sur les amers de l'environnement et les positions du robot. Leur grande faiblesse vient des contraintes et des hypothèses fortes qu'on doit appliquer aux différents modèles utilisés. En plus, il n'est pas toujours facile d'extraire des amers corrects et intéressants dans un environnement non-structuré ou externe.

L'utilisation d'un filtre particulaire Rao-Blackwellisé permet de maintenir une estimation a posteriori de l'état du système d'une manière plus rapide que le KF. Le filtrage particulaire peut également être utilisé avec des cartes grid-based ou feature-based. Cette

méthode souffre néanmoins de plusieurs problèmes à cause de sa complexité grandissantes et ses difficultés de paramétrage.

L'IML reste une solution intéressante grâce à sa simplicité et son efficacité en temps de calcul. En plus, il peut être appliqué à tous les types de cartes. Malheureusement, on ne peut estimer que les meilleures hypothèses en position à chaque étape de calcul, ce qui freine les possibilités de l'IML lors de la fermeture d'une boucle dans un environnement cyclique. Afin de résoudre ce type de problèmes, Wang [36] a proposé une méthode intéressante. Son idée est de se concentrer sur la construction de cartes locales, et les considérer ensuite comme des amers de l'environnement. Un EKF-SLAM se charge ensuite de parcourir les différents amers et d'estimer leurs positions relatives afin de construire une carte globale.

2.3 Conclusion

Dans ce chapitre, nous avons étudié une partie de l'état de l'art du SLAM. Nous avons ainsi commencé par une présentation de la problématique générale de la localisation et la cartographie simultanées. Nous avons ensuite formulé le problème mathématiquement, afin de pouvoir l'intégrer dans le cadre probabiliste de sa résolution.

La suite du chapitre est consacrée à la résolution du problème de SLAM. Nous avons dons commencé par la section 2.2.1 qui décrit les représentations de l'environnement les plus utilisées. Nous avons ainsi vu des cartes basée sur une concaténation des données (cartes en nuage de points). Nous somme passé ensuite à un autre type de cartes, basé sur l'extraction des caractéristiques géométriques de l'environnement et puis aux cartes en grille d'occupation. Nous avons conclu cette section par une comparaison entre les différents types de cartes présentés.

La section 2.2.2 est consacrée à la présentation de trois des algorithmes les plus utilisés pour la résolution du SLAM. Nous avons commencé par le filtrage de Kalman, et ses dérivés. Nous sommes passé ensuite au filtrage particulaire, et principalement le cas Rao-Blackwellisé proposé dans FastSLAM. La suite a été consacrée à l'algorithme d'IML, que nous avons choisi d'utiliser dans *CoreSLAM*. Nous finissons la section par une comparaison entre ces algorithmes.

D'après ce chapitre sur l'état de l'art, il parait clairement qu'aucune approche n'arrive à réaliser des cartes de SLAM exactes pour de grands espaces, principalement à cause du coût de calcul élevé et des incertitudes. Il s'agit donc d'une problématique qui nécessite de l'amélioration. Certaines approches cherchent à résoudre cette problématiques en utilisant plusieurs cartes de l'environnement, ou des cartes locales qu'on regroupe ensuite pour créer une carte globale ([15], [54] et [55]). Ce genre de méthodes se base essentiellement sur une bonne association des données. *CoreSLAM* innove justement à ce niveau, en proposant une méthode innovante et simple pour réaliser cette étape.

Le plus important dans la science n'est pas tellement d'obtenir de nouveaux faits, mais de découvrir de nouvelles manières d'y penser

William Lawrence Bragg

3

Le SLAM par Laser : de tinySLAM à CoreSLAM

Ce chapitre explique les principales bases de notre algorithme de SLAM. Appelé *tinySLAM* au début, car son implémentation ne dépassait pas 200 lignes de codes en langage C, l'algorithme a changé de nom pour devenir *CoreSLAM*. Plusieurs changements y ont été introduits, dans le but de gagner plus de stabilité et de fiabilité, tout en respectant un cadre d'exécution en temps réel.

Sommaire

	tinySLAM	*CoreSLAM*
Latence	Appliquée sur toutes les données	Pas de latence pour les données des nouvelles zones observées
Carte des distances	Carte carrée	Carte hexagonale
Algorithme d'optimisation	Monte Carlo et Algorithme Génétique	Algorithme Génétique et Algorithme Multi-échelle

Table 3.1 – *Tableau présentant les principales différences entre* tinySLAM *et* CoreSLAM

3.1 Introduction

Dans ce chapitre, nous allons parcourir les principales étapes marquant le passage de l'algorithme *tinySLAM* à *CoreSLAM*. Les grandes différences entre ces deux algorithmes de SLAM sont résumées dans le tableau 3.1.

Nous allons d'abord présenter les capteurs utilisés durant les différents tests de *tinySLAM* et de *CoreSLAM*. Ensuite, nous présentons les idées de base caractérisant chacun de ces algorithmes.

3.2 Les capteurs utilisés

Le développement et l'amélioration de *tinySLAM* a commencé sur une plate-forme robotique appelé le Mines-Rover (la section 5.1 présente le Mines Rover en détail). Elle est équipée d'un capteur Laser Hokuyo URG 04LX d'une portée de 4 mètres. Cette plate-forme présente l'avantage d'offrir une excellente odométrie. Ainsi *tinySLAM* pouvait fonctionner en utilisant le capteur Laser seulement ou avec l'aide des données d'odométrie.

Dans le cadre du concours de robotique CAROTTE (voir la section 5.2.1 pour plus de détails sur ce concours), nous avons continué les développements sur un autre type de plates-formes, basées sur des Wifibots [1] et équipées d'un capteur Laser Hokuyo UTM 30LX, qui permet d'atteindre une portée de 30 mètres.

Nous avons ensuite profité du lancement du capteur Kinect de Microsoft en 2010, afin d'étudier la possibilité d'ajouter des capacités de vision et de fonctionnement en 3D à l'algorithme de SLAM.

URG 04LX

Le capteur Laser URG 04LX permet d'atteindre une portée maximale de 4 mètres dans un arc de 240° avec une résolution angulaire de 0.36° et une précision de 10 mm. Le nombre de points de balayage (un scan laser) peut atteindre 682 points.

Les premiers travaux de développement de *tinySLAM* ont été effectués en utilisant ce capteur, car il offre un excellent rapport entre le coût d'achat et la qualité des données.

1. http ://www.wifibot.com/

FIGURE 3.1 – *Le capteur URG 04LX d'Hokuyo est le premier capteur utilisé pour le développement de l'algorithme de SLAM* tinySLAM

Ce capteur laser souffre néanmoins de quelques inconvénients :

- Sa portée maximale est limitée à 4 mètres. Durant les premiers tests, nous utilisions un robot qui peut atteindre une vitesse maximale de 3 m/s (voir la section 5.1 pour plus de détails*)*, ce qui signifie que la carte de l'environnement peut changer complétement en une seconde. Il s'agit d'un vrai challenge pour l'algorithme de SLAM, qui risque de perdre ses références de localisation facilement.
- La fréquence des mesures de 10 Hz est aussi un facteur limitant pour les performances du SLAM. Lorsque le robot se dirige vers à un mur par exemple à 3 m/s, le mur apparait incliné. En effet, le temps que le laser parcourt ses 240° depuis le point de la première mesure, le robot aura avancé de $240 * 360 * 3/10 = 20$ centimètres. Cette erreur a néanmoins été corrigée dans l'algorithme de SLAM.

UTM 30LX

Le télémètre à balayage Laser UTM 30LX permet d'atteindre une portée maximale de 30 mètres avec une résolution de 1 mm et une précision variant de 30 à 50 mm. Il offre un arc de vision plus grand de 30° que le capteur URG 04LX, avec une résolution de 0.25° et 1081 points de balayage. Ce capteur a été installé sur la plateforme Wifibot Lab V2 (cf. 5.2.2) dans le cadre de la participation au concours de robotique CAROTTE (5.2.1).

La Kinect

Le capteur Kinect contient une caméra RGB et une caméra infra-rouge, en plus d'un projecteur infra-rouge. Ce dernier permet de projeter un motif précis de points dans la scène. La détection de la disparité sur ce motif permet au capteur de fournir l'information de profondeur.

La Kinect offre un champ de vision horizontal de 57° et un champ vertical de 43°. Sa portée varie entre 0.6 mètres et 7 mètres, mais les données des objets situés au-delà de 4 mètres sont peu fiables.

Des études sur la qualité des mesures de la Kinect (cf . [56] et l'annexe D)montrent

Figure 3.2 – *Le capteur UTM 30LX est plus performant. Son utilisation dans le cadre du concours CAROTTE a permi d'améliorer* tinySLAM *vers* CoreSLAM

Figure 3.3 – *La Kinect nous permet de relier la profondeur à la couleur. L'algorithme de SLAM profite ainsi de la fusion de données et améliore les résultats de localisation et de cartographie*

une précision de l'ordre du millimètre pour les objets se situant à moins d'un mètre du capteur.

3.3 L'algorithme de base : *Incremental Maximum Likelihood*

L'Incremental Maximum Likelihood (IML) (Section 2.2.2.3) consiste à rechercher à chaque instant la meilleure correspondance entre l'observation courante (les données provenant du laser) et la carte courante (combinant la connaissance de l'environnement), et à remettre à jour la carte conformément à cette mise en correspondance. L'idée générale de cet algorithme est présentée dans le schéma de la figure 3.4.

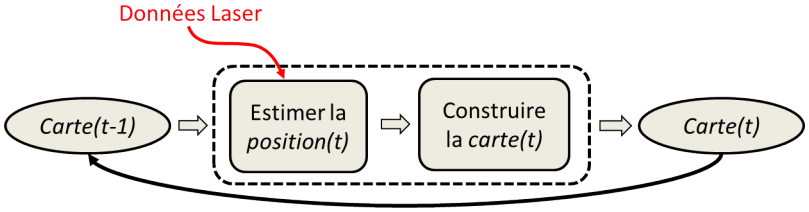


Figure 3.4 – *Principe général de l'IML implémenté dans tinySLAM/CoreSLAM*

L'IML est un processus évidemment divergent, puisqu'on utilise la localisation du robot pour mettre à jour la carte, et la carte mise à jour pour trouver la localisation du robot.

C'est pourquoi l'IML est une méthode très peu utilisée en SLAM, du fait de son caractère peu "Simultaneous". De fait, on lui préfère des techniques d'estimation mélangeant carte de position du robot dans l'état du système, tels que les SLAM à base de Filtre de Kalman (Section 2.2.2.1), ou les SLAM à base de RBPF (Section 2.2.2.2), ou encore des approches par optimisation off-line (Maximum-Likelihood estimation) [57]. Les algorithmes à base de filtre de Kalman ou de filtres particulaires ne sont en général pas convergents non plus en pratique - même s'ils le sont en théorie -, et sont donc associés à des dispositifs de détection de fermeture de boucle [58].
Si les filtres stochastiques ne sont convergents qu'en théorie et non en pratique, **pourquoi dans ce cas ne pas essayer de pousser le concept d'IML jusqu'au bout** ? C'est le pari que nous avons fait en développant *tinySLAM/CoreSLAM*, et l'approche se résume à essayer de limiter la dérive (ou divergence) au maximum en tirant profit de la simplicité initiale du concept d'IML.
Afin de limiter la dérive inhérente à l'IML, nous avons travaillé sur deux fronts :

- La **mise en correspondance** entre les scans du laser et la carte courante, qui doit être précise et rapide.
- La **mise à jour de la carte**, qui doit être intelligente pour maîtriser la dérive.

L'idée principale de la localisation par IML dans *tinySLAM* (et par la suite *CoreSLAM*) est basée sur le principe de mise en correspondance (**scan-matching**) des données du capteur Laser avec la carte la plus récente obtenue (Cf. figure 3.5).

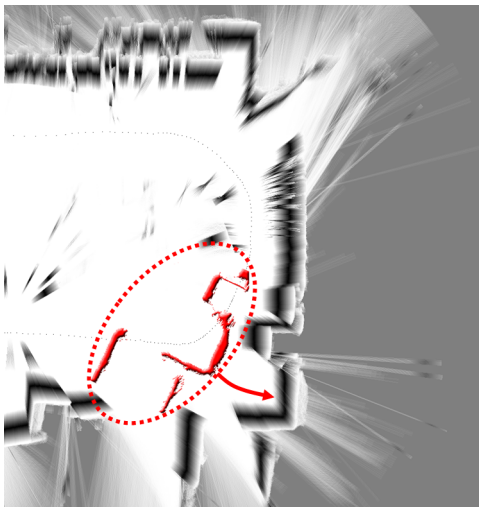


Figure 3.5 – *L'algorithme de scan-matching essai de trouver la bonne combinaison translation/rotation entre les données d'un scan du laser et la carte établie*

Cela revient à estimer la meilleure combinaison translation/rotation permettant aux données Laser de s'ajuster avec la forme des données de la partie de la carte correspondante.

3.4 L'estimation de la position : le scan-matching

Afin d'estimer la position du robot, l'algorithme *CoreSLAM* passe par le scan-matching (la mise en correspondance).

Le scan-matching a largement été traité dans plusieurs travaux. On va commencer par une brève description de certaines approches connues afin de mettre les explications données dans cette partie dans le bon contexte. Les algorithmes de scan-matching peuvent être divisés en trois parties:

- Lorsqu'on cherche des correspondances entre des amers, on parle d'une approche "amer-amer". On extrait des lignes [59] ou des coins [60] à partir des données du laser afin de les comparer. Dans de telles méthodes, l'algorithme a besoin d'un environnement structuré afin d'en extraire plus facilement les caractéristiques et les amers les plus connus.
- Dans les approches "scan-amer", on cherche une mise en correspondance entre des points d'un scan laser et des amers, tel des lignes [61] par exemple. Cet amer peut être une partie d'une carte que l'algorithme de SLAM vient de construire. Il n'est pas toujours nécessaire d'avoir des environnements structurés, par exemple Biber a considéré des distributions gaussiennes avec des moyennes et des variances calculées à partir du point d'impact du laser dans les cellules de la grille de la carte [62]. Ces approches nécessitent la présence des amers choisis dans l'environnement.
- Dans les approches qui se basent sur une comparaison "scan-scan", l'algorithme ne dépend pas de l'existence d'amers dans l'environnement. La mise en correspondance est faite directement sur les données brutes du laser, en utilisant une distance que l'algorithme cherche à minimiser [63].

Plusieurs travaux de recherche se sont intéressés à l'amélioration de l'étape du scan-matching dans les algorithmes de SLAM. Certains visent à accélérer la mise en correspondance en changeant la façon d'interpréter les données du laser [64], tandis que d'autres méthodes choisissent de changer l'espace de travail pour améliorer la qualité de l'appariement [65].

Le scan-matching implémenté dans *CoreSLAM* se base sur une **comparaison** entre les **données du scan laser** et la **carte établie**, sans pour autant en extraire des amers. Cette opération se divise en plusieurs étapes :

- L'algorithme récupère les données Laser correspondant à la nouvelle position du robot.
- Il génère plusieurs hypothèses sur la position et leur attribut **un score** en utilisant la carte établie durant la dernière opération de cartographie.
- Il garde l'hypothèse ayant obtenu le meilleur score comme la plus proche de la position réelle.

3.4.1 La localisation

Pour mieux expliquer le processus de localisation, on va considérer un cas simple. Dans la figure 3.6, on a représenté les 4 étapes clés du processus de localisation. La forme des données du laser est mise dans le cadre rouge et la carte actuelle est représentée dans le cadre gris.

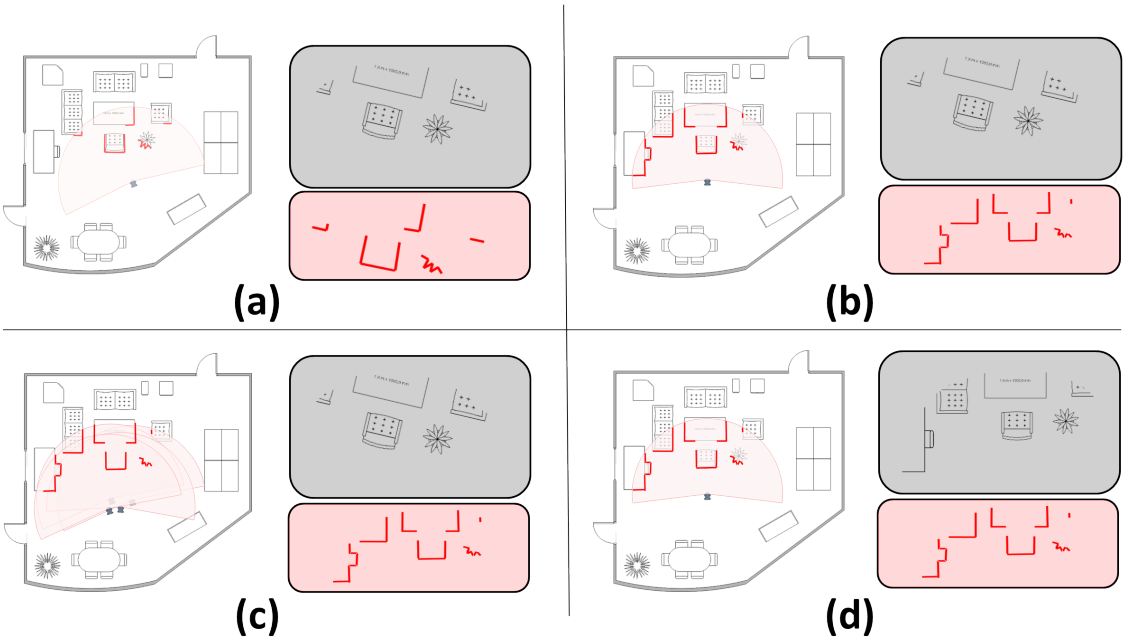


Figure 3.6 – *Les étapes clés de la localisation dans* tinySLAM/CoreSLAM

Voici une explication des étapes représentées dans la figure.

- (a) : le robot commence sa cartographie. Il enregistre les données du laser dans la carte.
- (b) : le robot se déplace. Il récupère de nouvelles données du capteur laser. La carte reste inchangée.
- (c) : l'algorithme de SLAM génère plusieurs hypothèses concernant la position du robot. Il calcul les différents scores en utilisant la méthode expliquée dans la section 3.4.2. Il utilise pour cela les nouvelles données du laser et la carte actuelle. On peut voir que les données du laser ont une partie commune avec la carte, l'algorithme profite de cette partie pour estimer le déplacement du robot.
- (d) : lorsque l'algorithme trouve la meilleure estimation de la position du robot, il met à jour sa carte en y ajoutant les nouvelles données du laser.

3.4.2 Le calcul du score d'une hypothèse

Calculer le score d'une hypothèse revient à définir une distance entre le scan laser correspondant à la position de l'hypothèse et la carte actuelle.

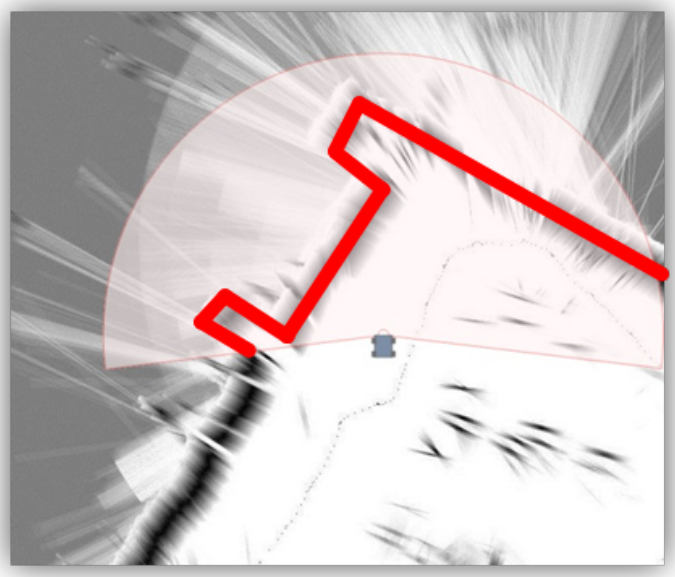


Figure 3.7 – *Le robot enregistre les données Laser correspondant à sa nouvelle position*

Le calcul du score doit être une opération simple, dans la mesure où elle est exécutée plusieurs dizaines de milliers de fois durant une phase de localisation. Elle est basée sur une addition simple. L'algorithme contient trois types de valeurs additionnées.

La première formule implique directement les valeurs des pixels dans la carte correspondant aux impacts des points Laser.

$$score = \sum_i valeur\_position_i$$

C'est la formule la plus simple et la plus rapide en termes de temps de calcul. Elle donne la même importance aux données correspondant à des objets proches du robot que celles des objets situés plus loin. Ceci peut causer quelques problèmes si la qualité des mesures du capteur Laser varie grandement en fonction de la position des objets.

La deuxième formule applique une fonction "logarithme" sur les valeurs avant d'effectuer la somme :

$$score = \sum_i \log(valeur\_position_i)$$

Cette formule donne moins d'importance aux objets situés loin du robot. À l'intérieur des bâtiments, cette formule respecte les variations de la qualité des mesures du capteur laser en fonction de la position des objets.

La troisième formule est basée sur une somme des carrées des valeurs de mesures fournies par le capteur Laser.

$$score = \sum_i (valeur\_position_i)^2$$

L'utilisation de cette formule peut être intéressante dans un contexte *outdoor*. En effet, dans un environnement externe, les objets les plus loin sont généralement les plus fiables.

Nous allons prendre un exemple utilisant la première formule pour mieux illustrer l'opération de calcul du score d'une position.

Dans la figure 3.7, le robot vient de se déplacer vers une nouvelle position. Il enregistre les données en provenance du capteur laser. L'algorithme de SLAM génère plusieurs hypothèses sur la position. Pour simplifier, on va considérer deux hypothèses seulement. La première hypothèse est située loin de la position réelle et la deuxième hypothèse est plus proche.

La figure 3.8 représente la première hypothèse. On voit que les impacts des points du Laser correspondent à des pixels qui se situent, pour la majorité, dans des endroits plutôt clairs. Ce qui signifie que la somme des valeurs de ces pixels, en niveau de gris, va être grande.

Le calcul du score donne : $score_1 = p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8$ donc $score_1 \simeq 50 + 255 + 100 + 120 + 200 + 255 + 10 + 255$. Après les calculs, on obtient : $score_1 \simeq 1245$.

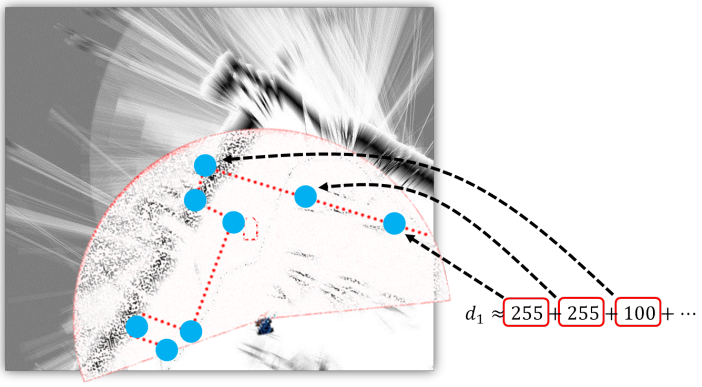


Figure 3.8 – *L'algorithme calcul le score de la première hypothèse*

Sur la figure 3.9, l'hypothèse générée est plus proche de la position réelle que la première hypothèse. Les impacts des points Laser sont situés dans des zones plus sombres. La somme des valeurs des pixels correspondant est ainsi plus petite qu'avant.

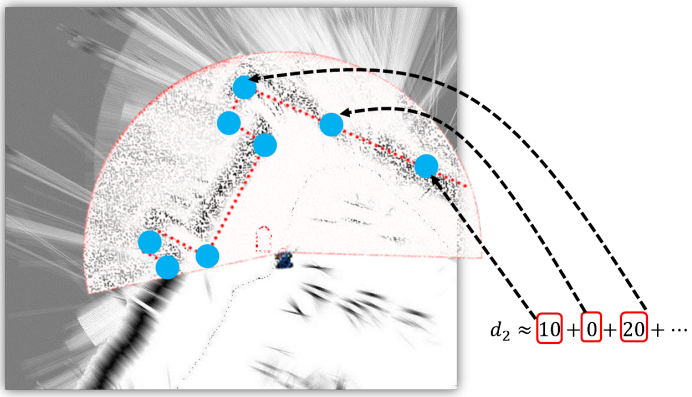


FIGURE 3.9 – *L'algorithme calcul le score de la deuxième hypothèse*

Le calcul du score donne : $score_2 = p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8$ donc $score_2 \simeq 200+50+100+100+10+150+10+150$. On obtient après calcul : $score_2 \simeq 770$.

L'hypothèse la plus proche de la position réelle du robot est celle ayant le score le plus faible.

3.5 La mise en correspondance

La phase de mise en correspondance (scan-matching) doit être la plus **précise** possible, tout en restant **rapide** afin de soutenir un fonctionnement en **temps réel**.
La mise en correspondance doit s'appuyer sur un algorithme d'optimisation cherchant la meilleure position du robot telle que l'observation (le dernier scan laser) corresponde au mieux à la carte courante. Pour qu'il puisse être rapide et précis, cet algorithme de recherche doit s'appuyer sur une représentation de la carte de SLAM facilitant cette recherche d'un minimum de distance entre les données du scan laser et la carte.
La représentation de carte que nous avons choisie a évolué entre *tinySLAM* et *CoreSLAM*. En effet, dans *tinySLAM*, on utilisait une carte carrée traditionnelle, qui a été remplacée par une carte hexagonale dans *CoreSLAM*. Néanmoins, dans les deux cas, **la construction de la carte a été pensée de manière à rendre la convergence de l'algorithme d'optimisation plus rapide et plus facile**.

3.5.1 La carte de SLAM dans *tinySLAM*

3.5.1.1 Définition

L'idée de *tinySLAM* est d'intégrer des informations du capteur laser dans un sous-système de localisation basé sur un filtrage particulaire, nous avons ainsi écrit deux fonctions principales :

– La fonction de calcul de la distance entre un scan et la carte. Cette fonction correspond au calcul du score expliqué dans 3.4.2. Elle agit comme une fonction de vraisemblance utilisée pour tester chaque hypothèse (particules) de position dans le

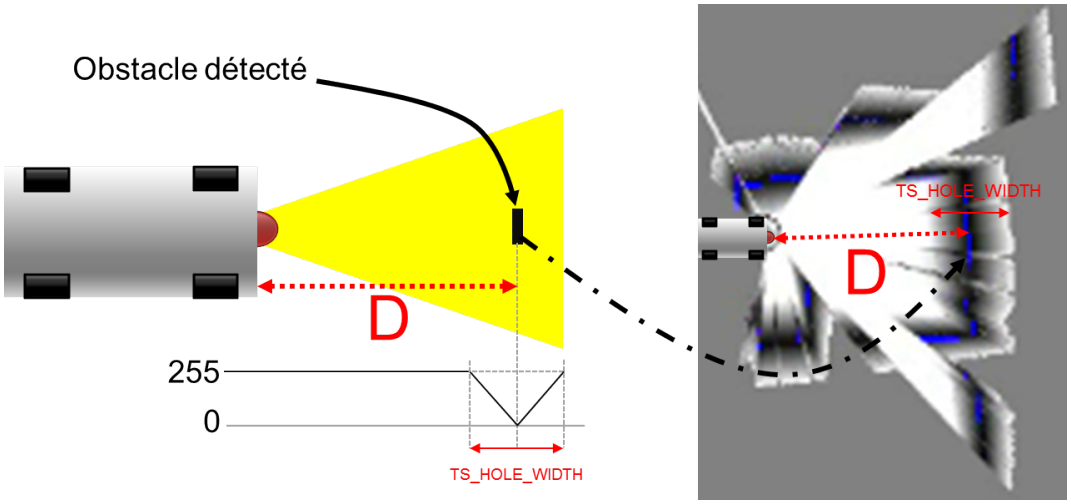


Figure 3.10 – *À chaque détection d'un obstacle, l'algorithme de SLAM insère un profil contenant un trou dans la carte. La figure 3.11 représente la carte de l'environnement*

filtre. Le code source de cette fonction est fourni en annexe[2]. Notez qu'il consiste simplement en une somme de toutes les valeurs de la carte à tous les points d'impact du scan (par rapport à la position de la particule). C'est une procédure très rapide, utilisant uniquement des opérations sur des entiers, ce qui offre plus de souplesse dans le choix du nombre de particules lors de l'exécution en temps réel. Toutefois, cette manière d'estimer la vraisemblance nécessite une carte (d'environnement) construite spécifiquement.

– La fonction de mise à jour la carte. Elle est utilisée pour construire la carte en même temps que le robot avance. Cette fonction va être traitée en détail ci-dessous.

Construire une carte compatible avec un filtre à particules n'est pas simple : le pic de la fonction de vraisemblance est très important pour l'efficacité du filtre et, par conséquent, doit être **facilement contrôlable**. Nous avons réussi cette tâche en utilisant des cartes en niveaux de gris. La mise à jour de la carte consiste à tracer des trous dont la largeur est directement liée au pic de notre fonction de calcul de vraisemblance. Pour chaque obstacle détecté, l'algorithme ne trace pas un seul point, mais une fonction avec un trou dont le minimum correspond à la position de l'obstacle. La figure 3.10 illustre cette idée. En conséquence, la carte construite ne ressemble pas aux cartes traditionnelles. Elle est constituée d'une surface contenant des trous de guidage de l'algorithme de mise en correspondance.

L'intégration dans la carte se fait par l'intermédiaire d'un filtre $\alpha\beta$ (ligne 74 de l'algorithme A.4), entraînant une convergence de la carte vers les nouveaux profils. La mise-à-jour de la carte est exécutée à la dernière position du filtre à particules (soit la moyenne pondérée de toutes les particules, après évaluation de la vraisemblance). La figure 3.12 montre un agrandissement d'une zone de la carte.

2. Voir ts_distance_scan_to_map dans l'algorithme A.2

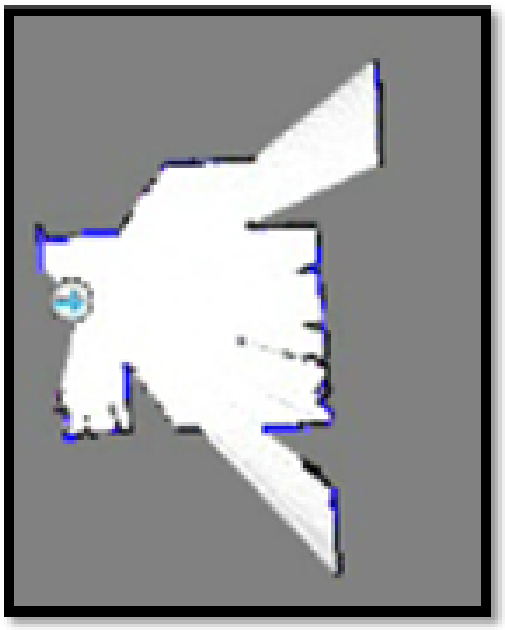


Figure 3.11 – *La carte de l'environnement correspondant à la carte de distance de la figure 3.10*

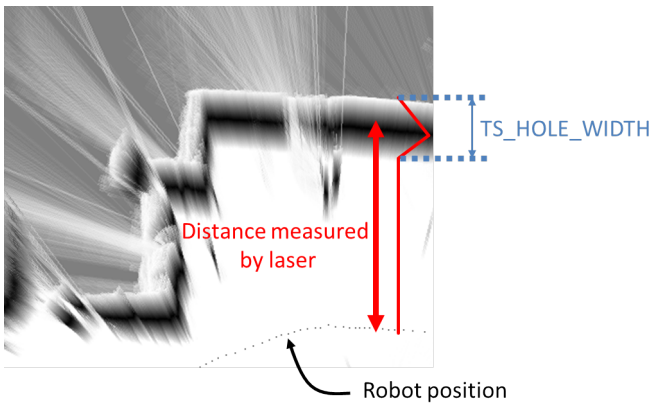


Figure 3.12 – *La carte de distance dans tinySLAM est construite d'une manière spéciale, qui aide l'algorithme à converger plus rapidement*

3.5.1.2 Construction

Le code source de la fonction ts_map_update est fourni en annexe (Algorithme A.3). Elle utilise la fonction ts_map_laser_ray (Algorithme A.4) qui est la partie la plus difficile. Cette fonction utilise un algorithme de Bresenham [66] pour tracer les rayons laser dans la carte, couplé avec un autre algorithme amélioré de Bresenham pour calculer le bon profil. Nous n'utilisons que des calculs sur entiers et pas de divisions dans les parties critiques de l'algorithme. Même si la mise à jour de la carte est exécutée une seule fois par étape (à 10 Hz pour *tinySLAM*), il est essentiel que cette procédure soit rapide, car une grande partie de la carte est touchée par le balayage du capteur laser, puisque la carte peut avoir une haute résolution (qui peut aller jusqu'à 1 cm par pixel).

3.5.2 La carte de SLAM dans *CoreSLAM*

3.5.2.1 Définition

Les différentes expériences menées avec *tinySLAM* ont révélé quelques problèmes avec la manière dont on construisait notre carte de SLAM. En effet, lorsque le robot se trouve en parallèle avec un mur, dans une petite zone, la zone dégradée en niveaux de gris (le trou), permettant la convergence rapide de l'algorithme, se trouve inévitablement réduite. La figure 3.13 montre un exemple de ce phénomène. Ceci peut causer de graves problèmes pour *tinySLAM*, dans la phase de localisation (et affecter par la suite tout le processus de SLAM, basé sur l'IML).
Nous avons ainsi changé la forme de la carte pour une **carte hexagonale ternaire** (obstacle - non obstacle - non exploré) à partir de laquelle est calculée incrémentalement une **carte de distance** aux obstacles, débordant autour des obstacles sur les zones non explorées (cf. 3.14). De fait, dans cette carte, les obstacles apparaissent comme des trous noirs qui vont attirer les points de scan laser via l'algorithme d'optimisation.

La carte étant hexagonale, l'attirance est quasi-isotrope : on profite de la similitude de la connexité hexagonale avec une distance euclidienne.

En suivant le même principe que la carte en niveau de gris de *tinySLAM*, si chaque point d'un scan laser tombe dans un trou (obstacle), on obtient un score de 0, qui correspond à un matching parfait.

L'avantage principal d'une carte hexagonale comparée une carte carrée traditionnelle est que la distance entre le centre de chaque cellule hexagonale et le centre de l'ensemble des six cases adjacentes est constante (voir la figure 3.15). Par comparaison, dans une carte carrée, la distance entre le centre de chaque cellule carrée et le centre des quatre cellules adjacentes en diagonale, avec lesquelles elle partage un coin, est supérieure à la distance au centre des quatre cellules adjacentes avec lesquelles elle partage une arête (voir la figure 3.16). Le caractère isotrope d'une carte hexagonale est important dans le domaine de la robotique dans lequel la mesure du mouvement est un facteur important. La figure 3.17 illustre une meilleure cartographie après l'implémentation de la carte hexagonale.

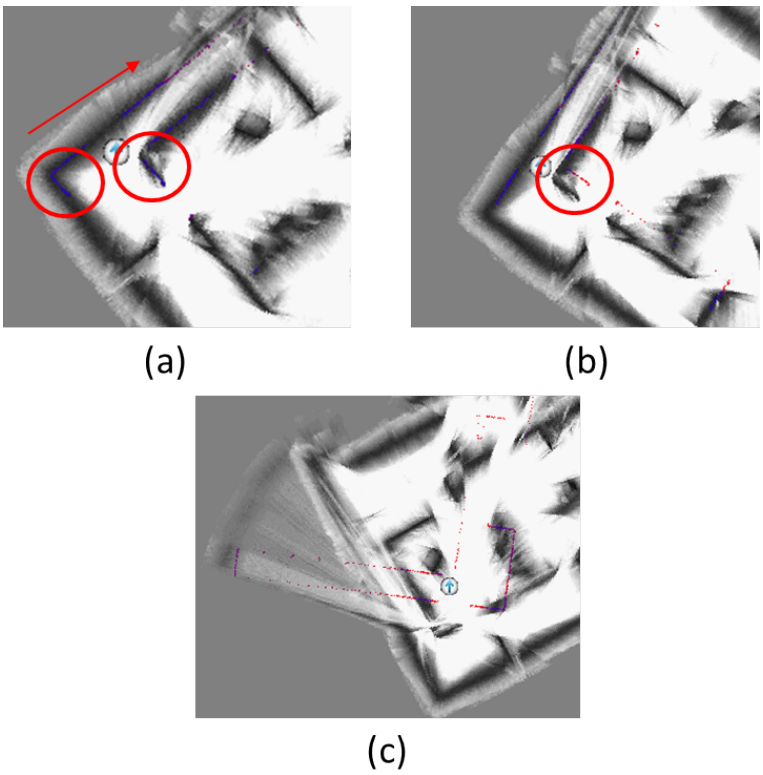


FIGURE 3.13 – *Dans certains cas (ici un couloir), la largeur de la zone permettant d'accélérer la convergence de l'algorithme de recherche est petite, ce qui cause des erreurs de localisation*

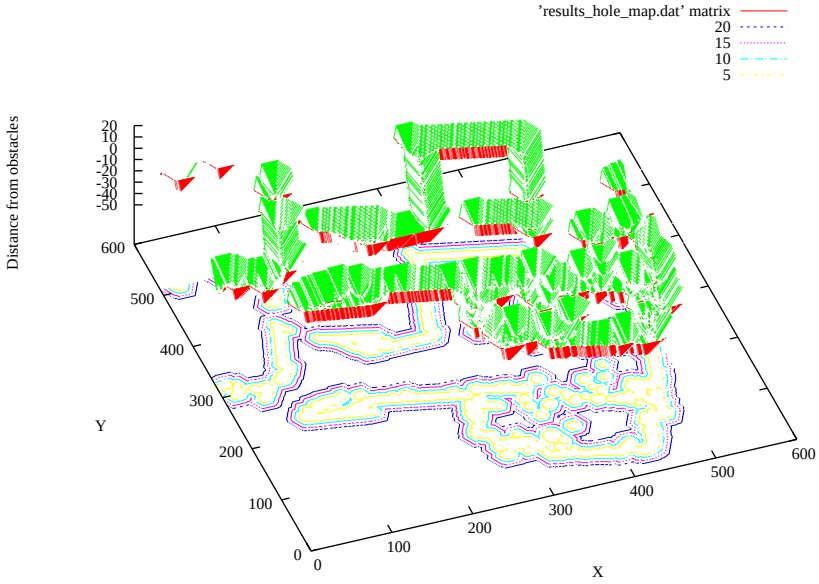


Figure 3.14 – *Carte de distance aux obstacles utilisée pour la mise en correspondance*

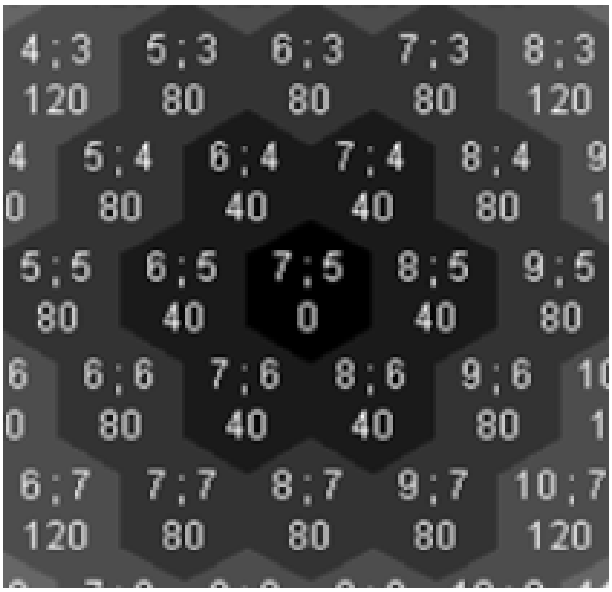


Figure 3.15 – *Carte de distances hexagonale. Ce type de carte permet de se rapprocher de la distance euclidienne. Chaque cellule contient ses coordonnées et un entier représentant sa distance à la cellule du centre (ici la cellule aux coordonnées 7 :5)*

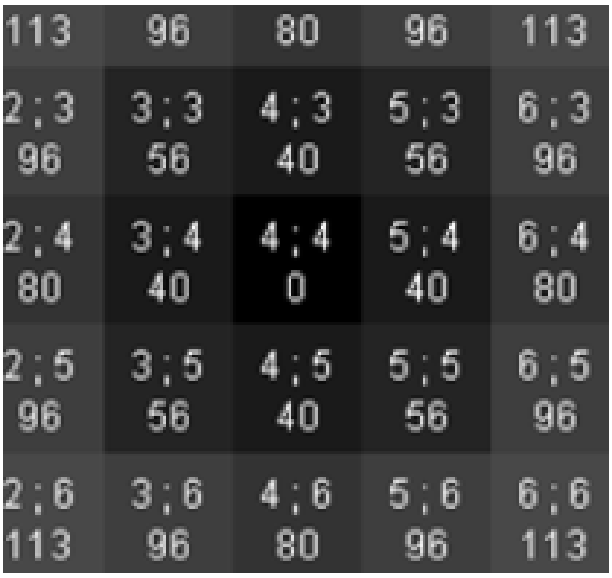


Figure 3.16 – *Carte de distances carrée. On constate que les distances des points ne sont pas homogènes autour du point de coordonnées (4,4)*

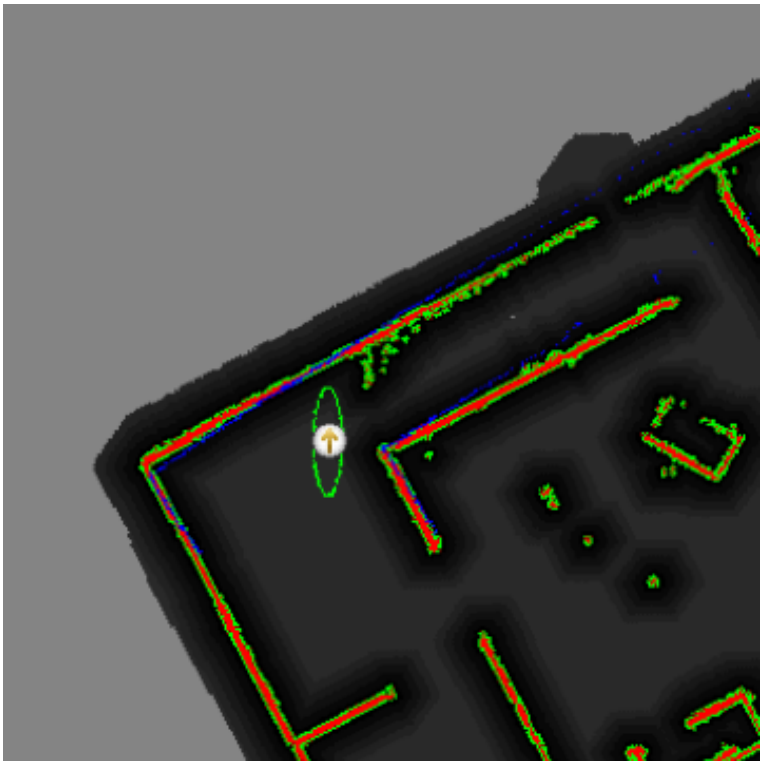


Figure 3.17 – *L'utilisation d'une carte de distances hexagonale permet de corriger le problème identifié dans la figure 3.13*

La carte hexagonale présente un autre avantage. En effet, les cellules voisines partagent toujours une arête entre elles, il n'y a jamais deux cellules voisines ayant un seul point de contact.

3.5.2.2 Construction

Format de la carte La carte de SLAM utilisée dans *CoreSLAM* est une **carte de distances hexagonale**. Il s'agit d'une transformée de distance appliquée à la carte des obstacles. Afin de travailler dans une carte hexagonale, on utilise la connexité 6. Rappelons que la connexité dérive de la notion de voisinage. Deux points M et P sont connexes s'ils sont mutuellement voisins par le système de voisinage défini. Il existe principalement deux ordres de connexité : 4 et 8 (voir la figure 3.18). Ce nombre correspond à la taille du plus petit voisinage non vide d'un pixel.

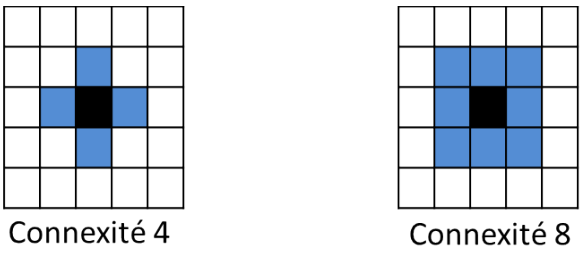


Figure 3.18 – *Les voisinages 4 et 8*

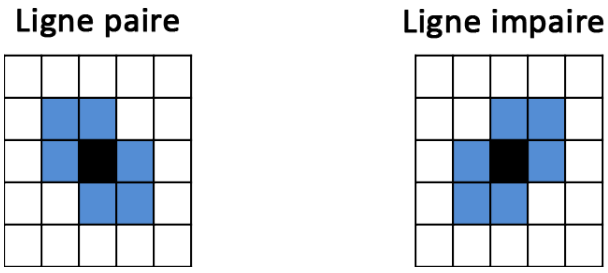


Figure 3.19 – *Définition du voisinage 6. On distingue les cas des lignes pairs et des lignes impairs*

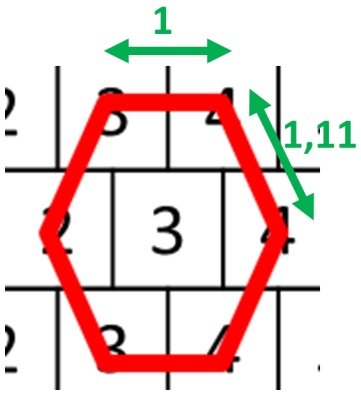


Figure 3.20 – *Les tailles des côtés de l'hexagone*

La méthode suivie pour la construction de la carte hexagonale a la particularité de **changer entre les pixels de lignes paires et les pixels de lignes impairs**. La figure 3.19 montre la manière dont on procède pour construire la carte.

L'hexagone construit n'est pas parfait, comme on peut le voir sur la figure 3.20. Cette propriété de l'hexagone de la carte permet de garder les avantages traditionnels d'une carte hexagonale tout en évitant ses inconvénients, car les déplacements horizontaux et verticaux restent équivalents à ceux d'une carte carrée. On ne se déplace pas en zigzag, ce qui permet d'avoir une distance proche de la distance euclidienne (le cas idéal).

Afin d'illustrer cette approximation de la distance euclidienne, on va prendre un exemple. On va calculer la distance séparant des points de coordonnées *(0, 0)* et *(4, 4)*. Dans le cas idéal d'une distance euclidienne, on obtient :

$$d_e = \sqrt{4^2 + 4^2} \approx 5.65$$

On va utiliser la figure 3.21 pour calculer cette même distance en utilisant une carte carrée puis une carte hexagonale.

– Pour la carte carrée en connexité 4, le calcul est simple. On a besoin de "4 déplacements" en horizontal et puis en vertical pour aller du point *(0, 0)* au point *(4, 4)*. Ce qui nous donne une distance :

$$d_c^4 = 4 + 4 = 8$$

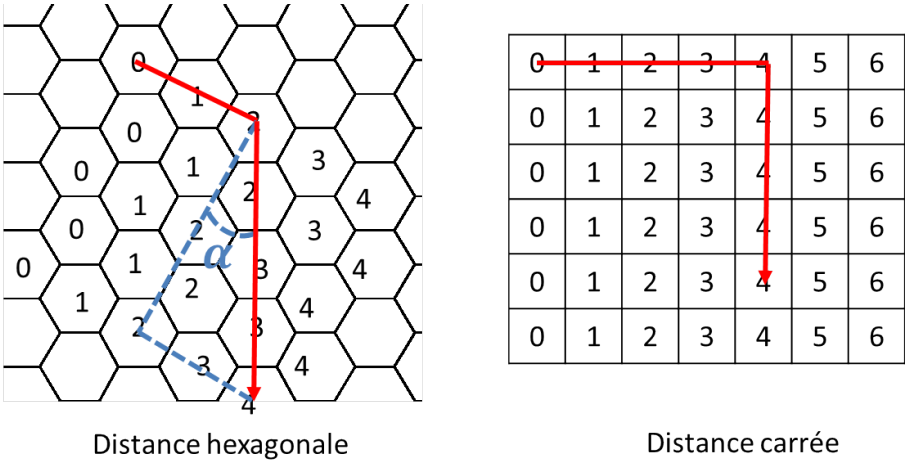


FIGURE 3.21 – *Calcul de distance*

– Pour la carte carrée en connexité 8, l'algorithme réalise des déplacements directs entre les deux points, en diagonale.

$$d_c^8 = 1 + 1 + 1 + 1 = 4$$

– Pour la carte hexagonale (connexité 6), on a :

$$d_h = 2 + \frac{2}{sin(\alpha)} = 2 + \frac{2}{sin(30°)} = 6$$

On voit que d_h est plus proche de d_e que d_c^4 et d_c^8, ce qui confirme l'avantage d'utilisation d'une carte hexagonale.

Génération de la carte La carte de l'environnement dans *CoreSLAM* est organisée en trois parties :

– Obstacle : on fixe les valeurs des pixels des obstacles à 0
– Zone navigable : la valeur des pixels des zones libres est fixée à 255
– Non explorée : les zones non explorée ont une valeur de 128

Lorsque l'algorithme de SLAM reçoit les données du capteur Laser, il ajoute les obstacles dans la carte de distances. Une fois cette tâche terminée, l'algorithme va **propager l'information** sur la carte de distance.

Cette opération s'appuie sur un principe simple et puissant qu'on a appelé : Enforce The Law (forcer l'application de la loi). En effet, pour chaque point de la carte, on calcul sa valeur en utilisant les valeurs de ses voisins. On applique la loi de calcul suivante :

$$val = min(neighbours) + 1$$

L'application de la loi est effectuée grâce à une fonction récursive parcourant les points de la carte. La figure 3.22 montre les points obstacles détectés par le capteur Laser et servant d'entrée à l'algorithme de construction de la carte de distance.

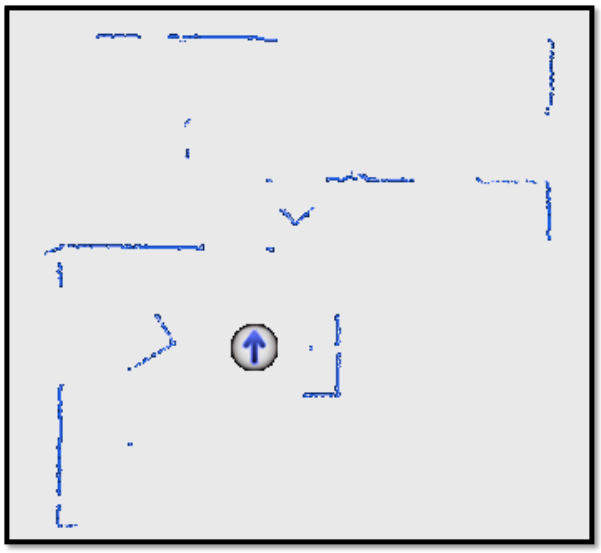


FIGURE 3.22 – *La carte des obstacles constitue l'entrée de l'algorithme de construction de la carte de distance*

La figure 3.23 illustre la méthode suivie pour construire la carte de distance avec un exemple. On a représenté les obstacles en gris, les valeurs qui respectent la loi en vert et les valeurs à changer en rouge. Par souci de simplification, on travaille avec une carte carrée dans l'exemple, mais la carte utilisée réellement dans l'algorithme de SLAM est bien hexagonale.

A la détection d'un obstacle (étape a), on met sa valeur à 0 et on commence à s'intéresser à ses voisins. Chaque point du premier groupe parcouru a comme voisin le point de valeur 0, on donne à tous ces points la valeur $val = min(neighbours) + 1 = 1$.

On passe au deuxième groupe (étape c). Les points marqués en vert ont tous une valeur de 2. Sachant que la valeur minimale de leurs voisins est 1, ils respectent bien la loi. Ainsi, on ne va pas modifier leur valeur.

Les valeurs des autres points, marqués en rouge, doivent être modifiées.

L'algorithme de création et de mise à jour de la carte de distance est représenté dans 3.1.

L'algorithme continue ainsi de se propager à chaque obstacle découvert, créant ainsi la carte des obstacles hexagonale, qui permet au robot de se localiser. La figure 3.24 illustre le type de résultats qu'on peut avoir après insertion d'un nouvel obstacle dans une carte de distance. Dans cet exemple, le robot détecte un mur au milieu de la zone de navigation. *CoreSLAM* insère l'obstacle détecté (les valeurs 0) et mets à jour la carte de distance.

3.6 La mise à jour de la carte et la latence

3.6.1 L'importance de la latence

Le concept de latence - le temps entre la réception d'un scan du laser et son intégration à la carte - est nécessaire pour mesurer les petits déplacements par rapport à la résolution de

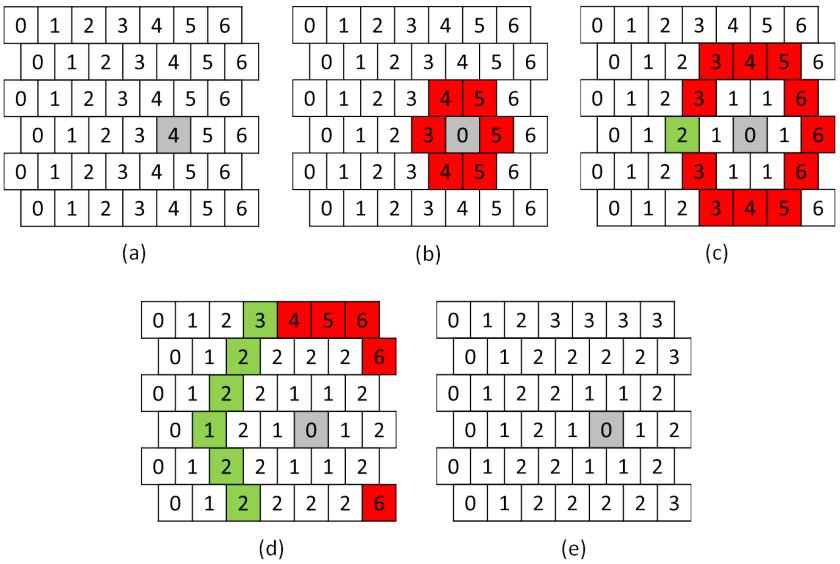


FIGURE 3.23 – *Le principe suivi dans la création de la carte hexagonale. (a) : détection d'un obstacle dans la cellule en gris. (b) : les cellules autour de l'obstacle ne respectent pas la loi. (c) : après une première itération, un autre ensemble de cellules ne respecte pas la loi. On doit changer leurs valeurs. (d) : l'ensemble des cellules en vert respecte la loi, les changements de valeurs seront donc faits sur les cellules en rouge seulement. (e) : l'algorithme d'application de la loi a fini ses itérations.*

Algorithme 3.1 Application de la loi de voisinage

Require: carte m, file q
  while (! fin de q) **do**
    if $l(x,y)! = 0$ **then**
      for $i = 1, 6$ **do**
        chercher le plus petit voisin
      end for
    end if
    if $l(x,y)! = minimum + 1$ **then**
      $l(x,y) = minimum$
    end if
    voir les voisins pour s'assurer de l'application de la loi
    for $i = 1, 6$ **do**
      ajouter à la file q
    end for
  end while

a

5	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
0	1	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
0	1	2	255	255	255	255	255	255	255	255	255	255	255	255	255	255
0	1	2	3	255	255	255	255	255	255	255	255	255	255	255	255	255
0	1	2	3	4	255	255	255	255	255	255	255	255	255	255	255	255
0	1	2	3	4	5	255	255	255	255	255	255	255	255	255	255	255
0	1	2	3	4	5	6	255	255	255	255	255	255	255	255	255	255
0	1	2	3	4	5	6	7	255	255	255	255	255	255	255	255	255
0	1	2	3	4	5	6	7	8	255	255	255	255	255	255	255	255
0	1	2	3	4	5	6	7	8	9	255	255	255	255	255	255	255
0	1	2	3	4	5	6	7	8	9	10	255	255	255	255	255	255
0	1	2	3	4	5	6	7	8	9	10	11	255	255	255	255	255
0	1	2	3	4	5	6	7	8	9	10	11	12	255	255	255	255
0	1	2	3	4	5	6	7	8	9	10	11	12	13	255	255	255
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	255	255
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	255
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	255
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	255	255
0	1	2	3	4	5	6	7	8	9	10	11	12	13	255	255	255
0	1	2	3	4	5	6	7	8	9	10	11	12	255	255	255	255
0	1	2	3	4	5	6	7	8	9	10	11	255	255	255	255	255
0	1	2	3	4	5	6	7	8	9	10	255	255	255	255	255	255
0	1	2	3	4	5	6	7	8	9	255	255	255	255	255	255	255
0	1	2	3	4	5	6	7	8	255	255	255	255	255	255	255	255
0	1	2	3	4	5	6	7	255	255	255	255	255	255	255	255	255
0	1	2	3	4	5	6	255	255	255	255	255	255	255	255	255	255
0	1	2	3	4	5	255	255	255	255	255	255	255	255	255	255	255
0	1	2	3	4	255	255	255	255	255	255	255	255	255	255	255	255
0	1	2	3	255	255	255	255	255	255	255	255	255	255	255	255	255
0	1	2	255	255	255	255	255	255	255	255	255	255	255	255	255	255
0	1	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
0	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255

b

0	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
0	1	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
0	1	2	255	255	255	255	255	255	255	255	255	255	255	255	255	255
0	1	2	3	255	255	255	255	255	255	255	255	255	255	255	255	255
0	1	2	3	4	255	255	255	255	255	255	255	255	255	255	255	255
0	1	2	3	4	5	255	255	255	255	255	255	255	255	255	255	255
0	1	2	3	4	5	6	255	255	255	255	255	255	255	255	255	255
0	1	2	3	4	5	6	7	255	255	255	255	255	255	255	255	255
0	1	2	3	4	5	6	7	8	255	255	255	255	255	255	255	255
0	1	2	3	4	5	6	7	8	9	255	255	255	255	255	255	255
0	1	2	3	4	5	6	7	8	9	10	255	255	255	255	255	255
0	1	2	3	4	5	6	7	8	8	9	9	255	255	255	255	255
0	1	2	3	4	5	6	6	7	7	8	8	9	255	255	255	255
0	1	2	3	4	4	5	5	6	6	7	7	8	8	255	255	255
0	1	2	3	3	3	4	4	5	5	6	6	7	8	9	255	255
0	1	2	3	2	2	3	3	4	4	5	6	7	8	9	10	255
0	1	2	2	1	1	2	2	3	4	5	6	7	8	9	10	255
0	1	2	2	1	0	1	2	3	4	5	6	7	8	9	255	255
0	1	2	2	1	0	1	2	3	4	5	6	7	8	255	255	255
0	1	2	2	1	0	1	2	3	4	5	6	7	255	255	255	255
0	1	2	2	1	0	1	2	3	4	5	6	255	255	255	255	255
0	1	2	2	2	1	1	2	3	4	5	255	255	255	255	255	255
0	1	2	3	3	2	2	3	3	4	255	255	255	255	255	255	255
0	1	2	3	4	3	3	4	4	255	255	255	255	255	255	255	255
0	1	2	3	4	4	4	5	255	255	255	255	255	255	255	255	255
0	1	2	3	4	5	5	255	255	255	255	255	255	255	255	255	255
0	1	2	3	4	5	255	255	255	255	255	255	255	255	255	255	255
0	1	2	3	4	255	255	255	255	255	255	255	255	255	255	255	255
0	1	2	3	255	255	255	255	255	255	255	255	255	255	255	255	255
0	1	2	255	255	255	255	255	255	255	255	255	255	255	255	255	255
0	1	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
0	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255

Figure 3.24 – *(a) : une carte de distance. (b) : la même carte de distance après l'ajout d'un obstacle (un mur) et la mise à jour de la carte*

la carte. Par exemple, afin de mesurer correctement le déplacement d'un objet se déplaçant à 1cm/s avec une carte de résolution de 1 cm et une fréquence de 10 Hz, il est nécessaire d'attendre 10 mesures, donc le temps de latence devrait être de 10. En effet, la formule théorique de la latence est :

$$\lambda = \frac{ResolutionDeLaCarte * FréquenceDeMesure}{VitesseDuRobot}$$

Dans le cas du robot Mines Rover par exemple (voir section 5.1 pour plus de détails sur la plateforme) , et compte tenu de la résolution de la carte de *tinySLAM*, la formule est : $\lambda = \frac{0.01*10}{VitesseDuRobot}$. Le Mines Rover fonctionne à une vitesse qui peut atteindre des valeurs élevées (3 m/s). Sa vitesse est rarement inférieure à 0,01 m/s, et même à cette vitesse, une latence égale à 1 est suffisante pour effectuer les calculs.
L'application d'une latence λ est très importante pour la mise à jour de la carte. Son importance est démontrable mathématiquement.
Soit $error(i)$ l'erreur de localisation à l'instant i. Cette erreur est l'addition de l'erreur produite par l'algorithme de mise en correspondance, nommée ε, supposée dans la suite systématique et constante, et de l'erreur accumulée dans la carte $\xi(i - \lambda)$, en prenant en compte la latence de mise à jour de la carte.

$$error(i) = \varepsilon + \xi(i - \lambda)$$

On peut considérer de manière simple que la dérive de localisation est la même que la dérive ou erreur accumulée dans la carte, et on peut donc affirmer que

$$\xi(i) \approx error(i)$$

et donc

$$error(i) = \varepsilon + error(i - \lambda)$$

et en partant d'une erreur nulle au départ, on en déduit une croissance linéaire de l'erreur de localisation telle que

$$error(i) \approx \frac{i\varepsilon}{\lambda}$$

La dérive est donc inversement proportionnelle au paramètre de latence de mise à jour de la carte, ce qui rend le concept indispensable. L'algorithme IML de base (à savoir mise en correspondance - mise à jour de la carte) suppose une latence λ égale à 1, ce qui est le cas le pire. L'idéal serait d'avoir une latence infinie, c'est à dire pas de mise à jour de la carte : dans ce cas on n'observerait aucune dérive. Cela justifie pourquoi nous ne mettons à jour la carte **que rarement**, et seulement avec une **latence relativement importante**. Le choix d'une latence d'une seconde permet de limiter la dérive tout en garantissant que les obstacles mobiles courants (humains, autres robots, portes) sont bien suivis dans la carte. De même, si le robot avance très lentement, son mouvement ne pourra être détecté que si l'erreur de localisation grandit sensiblement plus faiblement que la vitesse du robot.

Le seul paramètre sur lequel il est possible de jouer est λ, la latence, qui doit donc être inversement proportionnelle à la vitesse du robot en cas de faible vitesse.

3.6.2 Dans *tinySLAM*

La mise à jour de la carte de l'environnement est réalisée en utilisant un algorithme de Bresenham modifié. En effet, les lignes correspondant aux scans du capteur laser sont ajoutées à la carte d'une **manière particulière** (détaillée dans la section 3.5.1), afin de faciliter la convergence de l'algorithme de mise en correspondance. Pour chaque détection d'obstacle, l'algorithme ne trace pas un seul point, mais une fonction avec un trou dont la pointe est à la position de l'obstacle. L'algorithme de mise en correspondance convergent plus facilement vers l'obstacle, car le **trou de la fonction sert de guide**.

La précision "SubPixel"

Notons ici une particularité de la carte de SLAM dans *tinySLAM*.

La fonction *ts_distance_scan_to_map* permet de mesurer des déplacements de moins de 1 cm, même si la carte a une résolution de 1cm dans *tinySLAM*. En effet, elle prend en compte plusieurs points pour faire les calculs. Même un mouvement de 1mm peut être mesuré, car certains points du scan laser se trouveront dans un autre point de la carte. C'est ce qu'on appelle la précision "SubPixel".

3.6.3 Dans *CoreSLAM*

La première règle appliquée dans le cadre de la mise à jour de la carte dans *CoreSLAM* consiste à effectuer cette opération seulement **lorsqu'on est sûr que c'est utile**. Par exemple, si le score de matching est de 0, c'est qu'on a un matching parfait. Il ne faut surtout pas mettre à jour la carte dans ce cas, car cela correspond à un robot qui bouge beaucoup trop lentement pour que le mouvement soit détecté. Ensuite, si on détecte une rotation importante du robot ou que le score de matching est mauvais par rapport à la moyenne, on en profite pour ne PAS mettre à jour la carte. En fait, on choisit de ne mettre à jour la carte que rarement, en des moments choisis qu'on appelle **points de cohérence**. Et cette mise à jour se fait à travers un filtre passe-bas à hystérésis, et en appliquant une latence de mise à jour. Il est à noter qu'il reste nécessaire de mettre à jour la carte de temps en temps, afin d'en suivre les modifications dynamiques : les obstacles bougent, les portes s'ouvrent et se ferment. On ne peut se passer complètement d'une mise à jour de carte.

L'algorithme de mise à jour de la carte s'appuie sur un concept très simple : essayer d'utiliser les données les plus anciennes possibles. Plus les données inscrites dans la carte sont récentes, plus elles contiennent de dérive. Afin que les données dans la carte soient les plus vieilles possible, il est impératif d'inscrire dans la carte les données le plus tôt possible. Ainsi, la **deuxième règle** appliquée est d'**inscrire directement** dans la carte

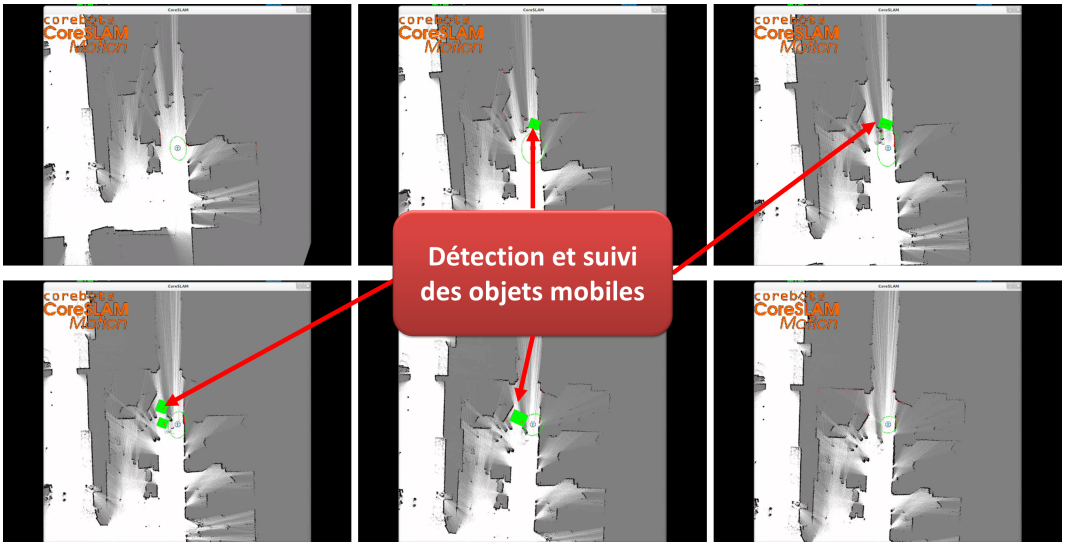


FIGURE 3.25 – *Le module* Motion *permet de détecter les humains qui se déplacent à côté du robot pendant l'acquisition*

les nouveaux obstacles découverts dans une zone non explorée, sans aucun délai et aucun filtre.

3.7 Motion : *CoreSLAM* et la détection des objets mobiles

Lorsqu'un algorithme de SLAM détecte et suit les objets mobiles, on parle de SLAM-MOT. Le SLAMMOT signifie : SLAM with Moving Object Tracking.

La dernière version de *CoreSLAM* intègre un module de détection d'objets mobiles : *Motion*. L'avantage d'un tel module réside dans l'amélioration qu'il peut apporter à l'algorithme de SLAM. En effet, en détectant les objets mobiles dans l'environnement, *CoreSLAM* les exclut de sa carte de SLAM (la carte permettant d'estimer la position), ce qui lui permet de réaliser une meilleure localisation.

Le fonctionnement du module *Motion* se base sur le calcul d'un score lié aux objets détectés par le laser dans une carte de distances hexagonale. Lorsque le robot découvre une nouvelle zone, l'algorithme enregistre le premier scan du laser comme un scan de référence. A partir du scan laser suivant, il détecte les pixels ayant un mauvais score (valeur dans la carte de distance). Si ces pixels sont groupés, on crée une zone qui les entoure et on les marque comme "objets mobiles". **Ces objets sont ignorés pour le processus de localisation**.

Le module *Motion* n'utilise **aucun filtrage**. Il détecte seulement des zones de la carte à ignorer. Ces zones sont traquées naturellement.

La figure 3.25 présente un exemple d'une détection d'objet mobile. Il s'agit ici de personnes se déplaçant à côté du robot.

L'algorithme marque les zones détectées comme objets mobiles en vert. Lorsqu'on ignore ces zones, on permet au processus de localisation (utilisant le scan-matching) d'avoir une carte ne contenant que les objets fixes, pour estimer la position. Nous présentons dans

le chapitre 6 les résultats de l'intégration de ce module.

3.8 Conclusion

Dans ce chapitre, nous avons présenté les bases de l'algorithme de SLAM proposé. Nous avons commencé par une présentation des capteurs utilisés pour les tests d'expérimentation et de validation. Le premier capteur est un Hokuyo URG-04LX. Avec ses caractéristiques techniques qui le placent en entrée de gamme, il nous a permis d'améliorer l'algorithme de SLAM et d'optimiser ses paramètres. Au fur et à mesure que l'algorithme évolue, d'autres capteurs ont été utilisés, comme le capteur laser à 30 mètre Hokuyo UTM-30LX ou le capteur de profondeur de Microsoft, la Kinect.

Nous avons ensuite présenté le fonctionnement de base de l'algorithme de SLAM. Il s'agit du principe de la recherche incrémentale du maximum de vraisemblance, que nous désignons par IML.

L'utilisation de l'IML permet d'implémenter un algorithme de SLAM simple et rapide. La première version a d'ailleurs été codée **en moins de 200 lignes de code en langage C**. Nous l'avons appelé *tinySLAM*. L'idée générale de l'algorithme consiste à estimer la position du robot à chaque instant t en se basant sur la carte construite à l'instant (t-1) et les données des capteurs de l'instant t. Une fois cette estimation faite, on l'utilise pour mettre à jour la carte de l'instant t, et passer à l'étape de localisation (t+1).

Après la présentation du fonctionnement globale de l'IML, nous avons détaillé la méthode utilisée pour localiser le robot. Cette localisation se base sur le scan-matching (la mise en correspondance) entre la carte de l'environnement et les données des capteurs. Cette étape est critique dans le processus de SLAM, nous avons ainsi cherché à la rendre la plus rapide et la plus correcte possible.

Afin d'atteindre ce but, nous avons développé une **représentation spéciale** de la carte de mise en correspondance (appelée carte de distance). Cette représentation a été modifiée entre la première version de l'algorithme, *tinySLAM*, et la dernière version, appelée *CoreSLAM*. Mais le but principale de cette représentation est le même : faciliter la convergence de l'algorithme de scan-matching vers la position la plus probable du robot. Nous avons détaillé l'algorithme de construction de la carte distance.

A la fin du chapitre, nous avons présenté un module de détection des objets mobiles que nous avons intégré à l'algorithme de SLAM. Ce module utilise la carte de distance générée préalablement, et il présente l'avantage d'être simple et efficace. Il n'utilise aucun filtrage et permet le suivi des objets en mouvement dans l'environnement. L'intérêt principal d'un tel module est qu'il permet d'exclure ces objets mobiles de la carte de mise en correspondance, ce qui améliore la qualité de la localisation.

Après avoir développé les bases du fonctionnement de notre algorithme de SLAM, nous avons travaillé à l'amélioration de sa qualité, en testant plusieurs méthodes d'optimisation des différentes étapes de l'algorithme. C'est l'objet du prochain chapitre.

Bien que cela puisse paraître paradoxal, toute science exacte est dominée par la notion d'approximation

Bertrand Russell

4

Minimisation et optimisation

Afin de minimiser la dérive naturellement présente dans l'algorithme d'IML, nous avons testé plusieurs méthodes d'optimisation, à plusieurs niveaux de l'algorithme de SLAM. Recherche de position, création de la carte, améliorations des opérations mathématiques ... On parlera ici des méthodes et algorithmes de minimisation et d'optimisation implémentés.

Sommaire

4.1 Algorithmes de recherche

4.1.1 L'algorithme de Monte Carlo

Au cœur de l'algorithme *tinySLAM*, la fonction de recherche de la position optimale est un bloc important. Dans cette fonction, l'étape la plus critique concerne la manière de générer les hypothèses testées. En effet, lorsque les hypothèses sont générées de manière optimale, l'algorithme peut avoir un gain intéressant au niveau de la vitesse d'exécution et de la qualité de l'estimation.

Dans un algorithme de Monte Carlo pour la localisation, les hypothèses sont générées aléatoirement, autour de la meilleure estimation trouvée à l'étape précédente. Voici un algorithme (voir listing 4.1) qui présente la méthode de Monte Carlo pour la localisation d'une manière basique, inspiré de l'article [5].

Algorithme 4.1 Monte Carlo basique, inspiré de l'article [5]

for $i \leftarrow 1, n$ **do**
  $x_k = new\_pos(x_k, u_k)$
  $w_k^{(l)} = prob(z_k | x_k^{(l)})$
end for
$S_{k+1} = null$
for $i \leftarrow 1, n$ **do**
  Sample an index j from the distribution given by the weights in S_k
  Add $(x_k^{(j)}, w_k^{(j)})$ to S_{k+1}
end for
return S_{k+1}

S_k est le set de samples à l'étape k. Un sample est représenté par le couple (x_k, w_k), sachant que x_k est une hypothèse sur l'état du robot, et w_k le poids associé à cette hypothèse. u_k représente les entrées de l'algorithme (odométrie ou le résultat du modèle d'évolution du système en général) et z_k les données du capteur (un capteur laser dans notre cas). Le nombre de samples est n.

L'implémentation que nous avons utilisée pour l'algorithme de Monte Carlo correspond à celle présentée ci-dessus, avec quelques changements pour optimiser ses résultats. La figure 4.1 permet de mieux comprendre le fonctionnement de l'algorithme utilisé.

Nous nous basons sur une génération aléatoire de positions/hypothèses. Une fonction score (détaillée dans la section 3.4.2) permet de donner un poids à chaque hypothèse. Elle prend en entrée les données d'un balayage laser, une carte préétablie de l'environnement ainsi que l'hypothèse qu'on souhaite tester.

L'algorithme que nous avons utilisé se base sur deux paramètres pour arrêter la recherche.

Le paramètre *stop* permet d'arrêter la recherche après un certain nombre d'itération. Une fois ce nombre d'itérations atteint, l'algorithme nous retourne la meilleure position qu'il a trouvée.

Le paramètre *threshold* nous offre plus de chances d'éviter de tomber dans des maxi-

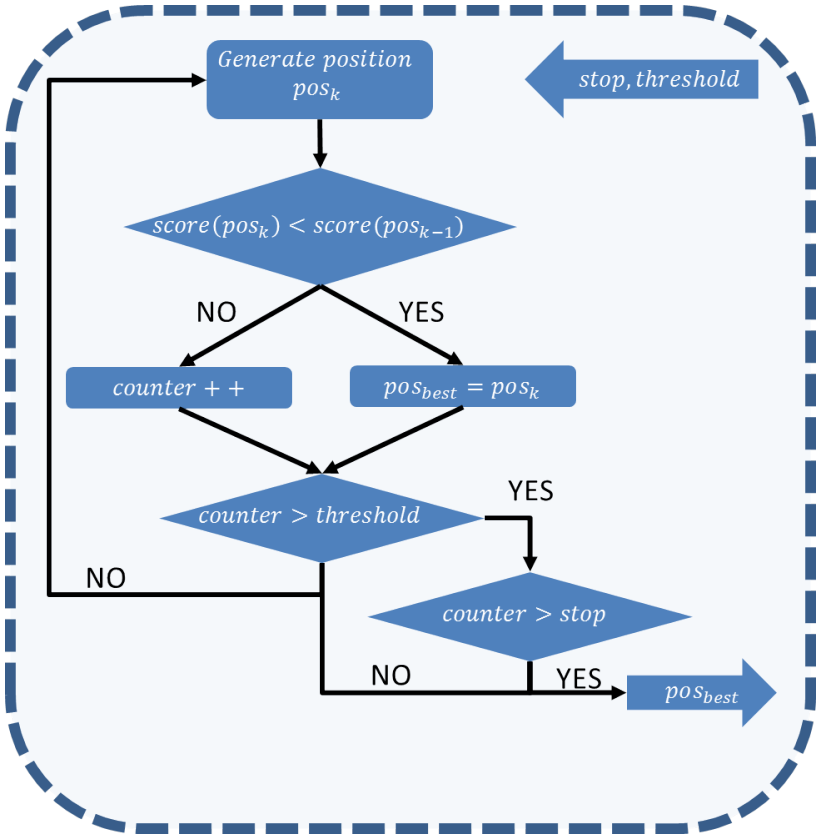


Figure 4.1 – *Digramme représentant le fonctionnement de l'algorithme de Monte Carlo utilisé dans cette étude. On fournit les deux paramètres* stop *et* threshold *à l'entrée de l'algorithme, qui fonctionne en boucle jusqu'à ce que le nombre de test autorisé soit atteint* (stop)*. Le paramètre* threshold *permet d'éviter les maximas locaux*

mas locaux. En effet, on commence à chercher l'estimation de la position du robot en générant des hypothèses dans une zone entourant la position de départ (soit la dernière position estimée du robot ou la position générée suivant le modèle d'évolution en utilisant les odomètres). Après sélection d'une meilleur hypothèse, l'algorithme essai de centrer la recherche sur cette hypothèse en diminuant la zone de génération des hypothèses.

En centrant la recherche autour d'un maxima local, on risque de ne pas bien estimer la position. On a ainsi introduit le paramètre *threshold* qui permet de valider l'hypothèse sélectionnée, avant de passer à la prochaine étape de recherche.

La figure 4.2 illustre un processus de recherche se basant sur la méthode de Monte Carlo.

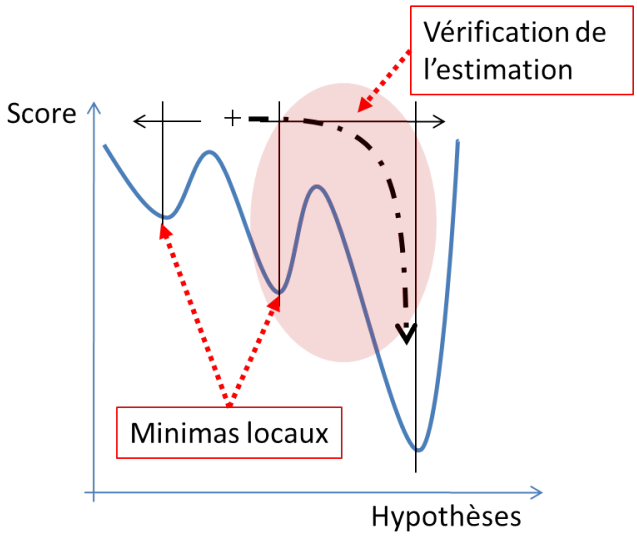


FIGURE 4.2 – *Recherche de maximum de vraisemblance par Monte Carlo. A partir d'une position de départ, on génère plusieurs hypothèses. En attribuant un poids à chaque hypothèse, on peut centraliser la recherche sur des hypothèses de plus en plus pertinentes, en convergeant vers la position optimale.*

4.1.2 L'algorithme Génétique

4.1.2.1 Présentation

Les algorithmes génétiques sont des méthodes d'optimisation inspirées par la nature. Ils utilisent essentiellement le processus de sélection naturel dont le principal effet est la survie des plus aptes. Dans notre travail, nous avons utilisé une implémentation générique des algorithmes génétiques, en nous basant sur le travail de Mitchel [67].

Les algorithmes génétiques ont été étudiés depuis les années 70 [68]. Ils ont été utilisés dans plusieurs projets liés aux problèmes d'optimisation dans la robotique.

L'un des premiers articles à avoir traité le problème du SLAM en utilisant les algorithmes génétiques est [69]. Dans cet article, le problème du SLAM est défini comme un problème d'optimisation globale et l'algorithme génétique est utilisé pour estimer la meilleure carte possible de l'environnement dans l'espace des hypothèses. Dans [70], l'ap-

proche présentée est similaire à celle utilisée dans ce travail de thèse. Dans cet article, on essaie d'estimer la position d'un robot mobile en cherchant la correspondance entre les données de deux séquences d'acquisition à partir du capteur laser.

D'autres études utilisent l'algorithme génétique fusionnée avec d'autres méthodes pour améliorer le processus de localisation, généralement dans un contexte de SLAM. On trouve ainsi un algorithme génétique couplé à un filtre Rao-Blackwellisé dans [71]. Dans [72], Begum présente une nouvelle méthode afin d'intégrer la logique floue et un algorithme génétique dans le SLAM. La logique floue fournit les conditions de démarrage de l'algorithme génétique, qui se charge d'améliorer l'information de localisation.

L'utilisation des algorithmes génétiques dans la robotique suscite de plus en plus d'intérêt, spécialement dans le domaine du SLAM, dans la mesure où ce genre d'algorithmes offre plusieurs avantages au niveau de la vitesse des calculs et de la consommation de mémoire, comparé à d'autres algorithmes d'optimisation.

4.1.2.2 Implémentation

Dans l'algorithme génétique utilisé dans *tinySLAM/CoreSLAM*, les hypothèses ne sont pas générées totalement d'une manière aléatoire. Certains processus, inspirés de la biologie, permettent de créer des hypothèses à partir de celles déjà existantes. Ces processus sont la **mutation** et le **croisement**. La figure 4.3 montre le schéma général d'un algorithme génétique.

- La mutation est une procédure qui consiste à générer une hypothèse à partir d'une autre hypothèse, en changeant l'un des paramètres (coordonnée x, y ou θ dans notre cas).
- Le croisement permet d'obtenir une hypothèse enfant à partir du croisement de deux hypothèses parents.

La figure 4.4 illustre les principes de mutation et de croisement tels qu'ils ont été utilisés dans notre étude.

L'algorithme génétique utilisé durant cette thèse suit le même schéma de fonctionnement représenté sur la figure 4.3. Cet algorithme se base sur la boucle représentée, et arrête les tests lorsqu'il atteint un nombre *stop*, en suivant le même principe que l'algorithme de Monte Carlo, expliqué dans la section 4.1.1.

La figure 4.5 illustre la manière dont l'algorithme génétique permet de trouver une valeur optimale rapidement. A partir d'une population d'hypothèses, on génère des hypothèses enfants qui sont généralement beaucoup mieux adaptées au problème grâce aux processus de mutation et croisement.

4.1.3 L'algorithme de recherche multi-échelle

Un des algorithmes d'optimisation implémentés dans *CoreSLAM* est une recherche multi-échelle. On part d'un cube de $6x6x6$ hypothèses (dans l'espace x, y, θ de position du robot) autour de la position supposée du robot. Si la meilleure hypothèse se trouve sur les bords du cube, le cube est translaté et l'échelle inchangée. Si la meilleure hypothèse se trouve à

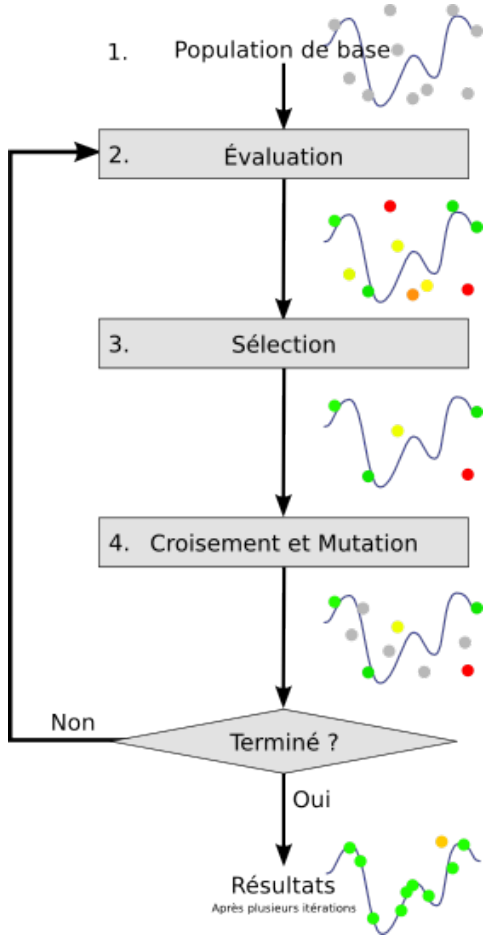


FIGURE 4.3 – *Principe général d'un algorithme génétique. L'étape de l'évaluation est effectuée avec la même fonction qu'on a utilisée pour attribuer des poids aux hypothèses générées par l'algorithme de Monte Carlo*

l'intérieur du cube, on passe à l'échelle inférieure en générant un nouveau cube de $6x6x6$ centré autour de la meilleure hypothèse (voir figure 4.6).

Les cubes successifs explorés sont illustrés sur la figure 4.7, un cas de recherche simple, et la figure 4.8, représentant un cas de recherche de minimum complexe éloigné du point de départ de la recherche.
On remarquera sur la figure 4.9 que le choix d'un cube de $6x6x6$ hypothèses n'est pas innocent : il permet lorsqu'on passe à une résolution 2 fois inférieure de ne pas tester deux fois un point déjà testé. La meilleure hypothèse se retrouve au centre du nouveau cube, qui lui-même n'est pas évalué.

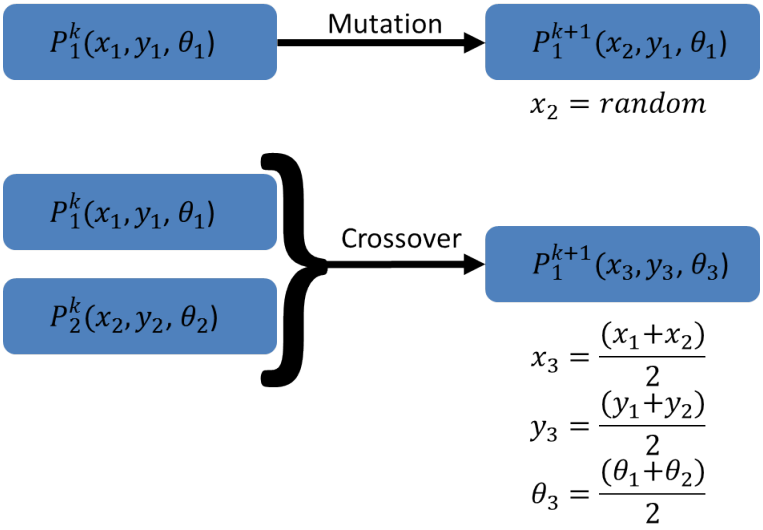


FIGURE 4.4 – *Les processus de mutation et de croisement dans l'algorithme génétique utilisé dans le cadre de cette étude*

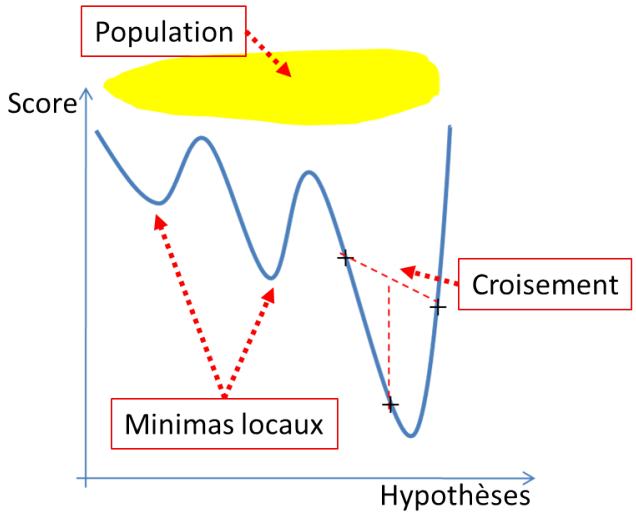


FIGURE 4.5 – *Recherche du maximum de vraisemblance par algorithme génétique. Les processus de mutation et de croisement permettent de générer des éléments de populations de plus en plus proches de la solution du problème.*

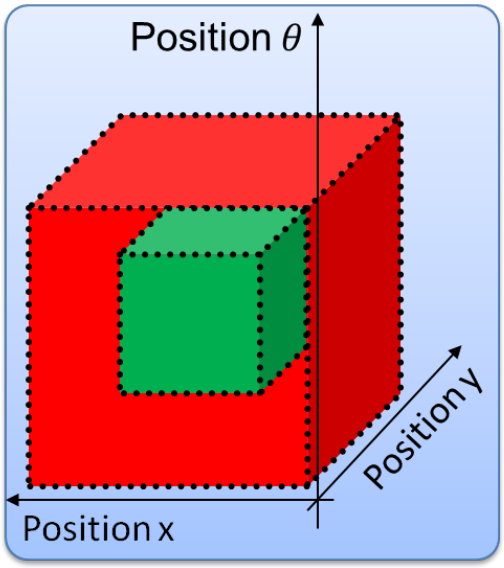


Figure 4.6 – *Réduction du cube des hypothèses durant une recherche multi-échelle*

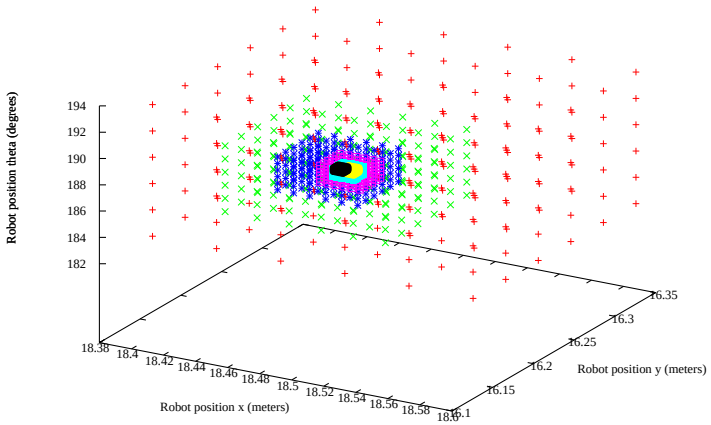


Figure 4.7 – *Algorithme de recherche multi-échelle : cas simple*

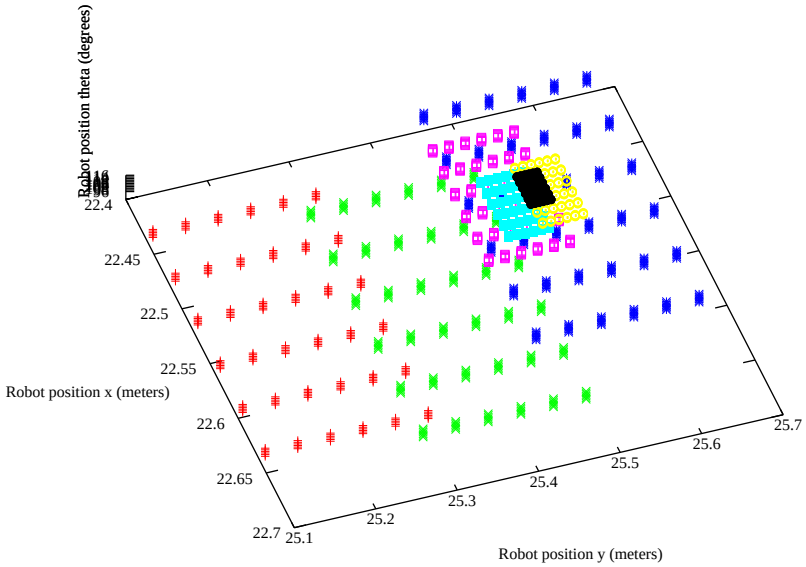


FIGURE 4.8 – *Algorithme de recherche multi-échelle : cas complexe où le déplacement du robot est important*

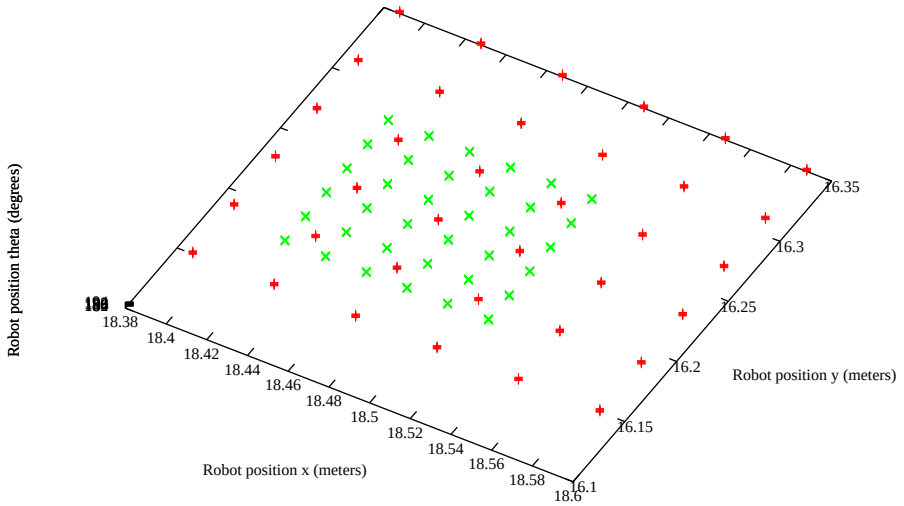


FIGURE 4.9 – *Algorithme de recherche multi-échelle : zoom sur deux échelles successives*

On se donne la possibilité d'explorer jusqu'à 7 cubes de $6x6x6$, ce qui permet d'atteindre une résolution de localisation de l'ordre du millimètre en position et de l'ordre du $10^{ème}$de degré en orientation. A la plus basse résolution et avec les paramètres courants, la résolution du cube en (x, y) est de $200/(6-1) = 40\,mm$. Elle est divisée par 2 à chaque itération si le minimum local se situe à l'intérieur du cube, ce qui aboutit à une résolution de $40/2^6 = 0.625\,mm$. Concernant la résolution angulaire, on part d'une résolution de $10/(6-1) = 2°$ pour aboutir à une possible résolution la plus fine de $2/2^6 = 0.03°$.
Ceci évidemment dans le meilleur des cas, i.e. lorsque le mouvement du robot d'un scan au suivant est d'une amplitude inférieure à $10\,cm$ en position et 5° en rotation, et que la fonction est parfaitement régulière. A titre d'exemple, ce n'est pas du tout le cas sur 4.7 et 4.8. Sur 4.7, la dernière résolution n'est pas atteinte car le bruit sur la fonction est trop important : on atteint dans ce cas la limite de résolution de l'algorithme de recherche par rapport à la régularité de la fonction à minimiser. Sur 4.8, seules quatre résolutions sont explorées car le mouvement du robot est très important ($35\,cm$ de déplacement d'un scan à l'autre. Le robot est très rapide). La résolution spatiale est dans ce cas de $40/2^3 = 0.5\,mm$ et la résolution angulaire de $2/2^3 = 0.25°$.

4.2 La file de priorité

4.2.1 Fonctionnement de la file de priorité dans *CoreSLAM*

Durant le processus d'évaluation des hypothèses, l'algorithme calcul les différents scores en utilisant l'ensemble des valeurs correspondantes aux impacts des points du Laser dans la carte. Ce processus implique, dans certains cas, de faire des millions de calculs pour chaque étape de localisation.

Afin d'améliorer cette étape, et réduire de nombre d'opérations nécessaires, nous avons implémenté une file de priorité. L'algorithme de la file de priorité permet d'éviter l'évaluation des hypothèses ayant de faibles chances d'être proches de la position réelle (avoir les meilleurs scores).

- *Une file de priorité est un type abstrait de données opérant sur un ensemble ordonné, et muni des opérations suivantes :*
  - trouver le maximum
  - insérer un élément
  - retirer le maximum

L'algorithme commence par calculer le score de l'ensemble des hypothèses, pour une partie des points d'impact laser [1]. Ces points sont choisis de manière uniforme. Les hy-

1. pour un capteur laser de type Hokuyo UTM 30LX, ayant 1080 points de données, on prend le $1/5^{ème}$ de l'ensemble des données, ce qui correspond à 216 points

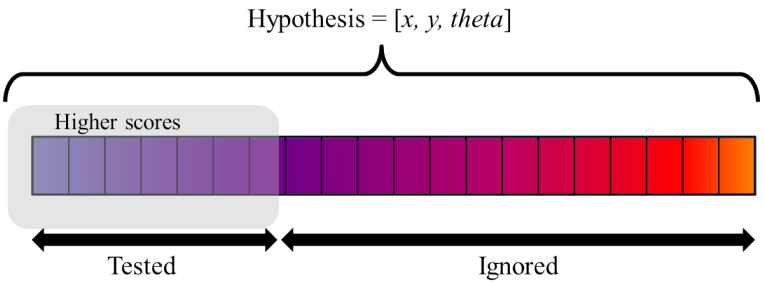


FIGURE 4.10 – *Le principe de la file de priorité. Les hypothèses ayant un score plus intéressant sont évaluées avec un grand nombre de points du scan Laser*

Algorithme 4.2 La file de priorité

Require: l'ensemble des hypothèses générées
  var $max_{eval} \leftarrow 2000$;
  évaluer toutes les hypothèses ;
  classer les hypothèses ;
  while $nb_{eval} \leq max_{eval}$ **do**
    var $hyp_{best} \leftarrow meilleure\ hypothèse$;
    évaluer hyp_{best} ;
    insérer hyp_{best} dans la file ;
    $nb_{eval} \leftarrow nb_{eval+1}$;
  end while
Ensure: le premier élément de la file de priorité est la meilleure hypothèse

pothèses sont ensuite triées globalement (avec un algorithme *QuickSort*) en fonction de cette première évaluation, et on recalcule les scores des meilleures hypothèses seulement, en utilisant une autre partie des points du laser, ce qui raffine le classement des hypothèses. Le nombre d'évaluations est limité à un certain seuil à ne pas dépasser. En fin de processus, seules les hypothèses les plus valides sont complètement évaluées.

Prenons un exemple pour mieux comprendre. Considérons un système équipé du capteur laser Hokuyo UTM 30LX (offrant 1080 points de données à chaque scan et fonctionnant à 40 Hz). Lorsque *CoreSLAM* utilise l'algorithme génétique pour l'estimation de la position réelle du robot, on génère 400 individus pour chaque population d'hypothèses.

On divise l'ensemble des données d'un scan laser en 5 parties (chacune contenant 216 points). On appelle "une évaluation" l'opération qui consiste à calculer le score d'une hypothèse en utilisant le $1/5^{ème}$ des données du laser. On peut clairement voir qu'il faut faire 5 évaluations pour calculer le score d'une hypothèse pour l'ensemble des points du laser. Afin de déterminer la meilleure des 400 hypothèses générées, on a donc besoin de 2000 évaluations. L'utilisation d'une file de priorité permet de diviser ce nombre par 2, tout en gardant un bon résultat de localisation, car on limite le nombre d'évaluations maximal à 1000 au lieu de 2000. Les évaluations manquantes sont liées aux hypothèses les moins proches de la position réelle.

Les étapes de fonctionnement de l'algorithme de la file de priorité sont représentées dans l'algorithme 4.2.

Pour chaque étape d'évaluation, l'algorithme extrait la meilleure hypothèse, l'évalue, et la réinsère ensuite dans la file de priorité. L'algorithme permettant d'extraire et insérer les hypothèses de cette manière est un algorithme de **tri par tas**. Ce type d'algorithmes simples nous permet de réaliser les opérations nécessaires à la file de priorité rapidement.

4.2.2 Algorithme de tri de la file de priorité : le tri par tas

Les traitements effectués sur les éléments de la file de priorité doivent être fait de manière **rapide** et **simple**. On cherche à avoir le meilleur élément de la file à chaque évaluation.

Afin de trier les éléments de la file de priorité, on utilise un algorithme de tri par tas [73].

- *Un tas (en anglais « heap ») est un arbre binaire tassé tel que le contenu de chaque nœud soit supérieur ou égal à celui de ses fils.*
- *Un arbre binaire est tassé si son code est un segment initial pour l'ordre des mots croisés.*

Dans un arbre binaire tassé, tous les niveaux sont entièrement remplis à l'exception peut-être du dernier niveau, et ce dernier niveau est rempli "à gauche". La figure 4.11 montre un exemple d'un arbre binaire tassé.

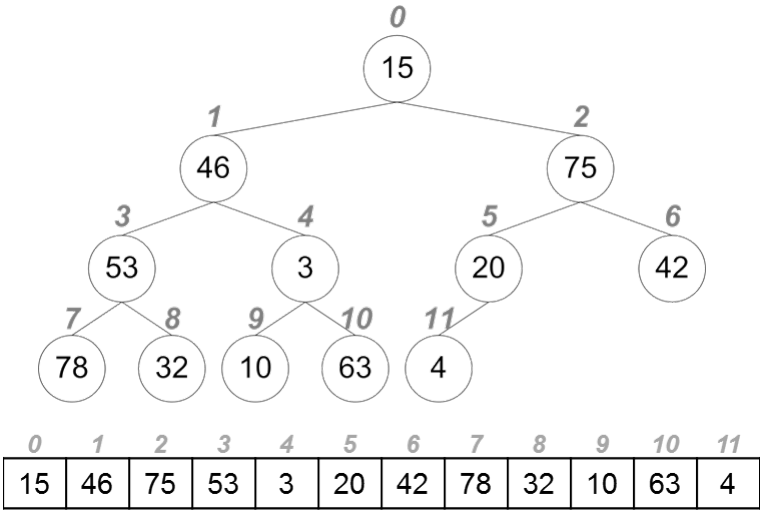


Figure 4.11 – *L'utilisation d'un arbre binaire tassé dans un tableau*

Les opérations effectuées sur un arbre binaire tassé respectent les principes des tas :
- Un élément a un contenu toujours supérieur ou égal à celui des fils
- L'insertion d'un élément commence par la gauche

L'utilisation d'un algorithme de tri par tas permet d'avoir une faible complexité pour les opérations de base de la file de priorité. On peut ainsi trouver le meilleur élément en *O(1)*, en insérer un en *O(log n)* et retirer le maximum en *O(log n)*

La figure 4.12 montre une opération d'extraction du meilleur élément d'un tas. Après avoir retiré le meilleur élément (étape 2), on le remplace par l'élément le plus à droite du dernier niveau de l'arbre (étape 3). On réorganise ensuite l'arbre de manière à respecter les principes du tas. 'M' étant inférieur à 'S' et 'R', on permute ces éléments (étapes 4 et 5). A l'étape 5, on s'aperçoit que l'élément 'M' a un contenu supérieur à ses deux fils, on garde la forme ainsi obtenue de l'arbre.

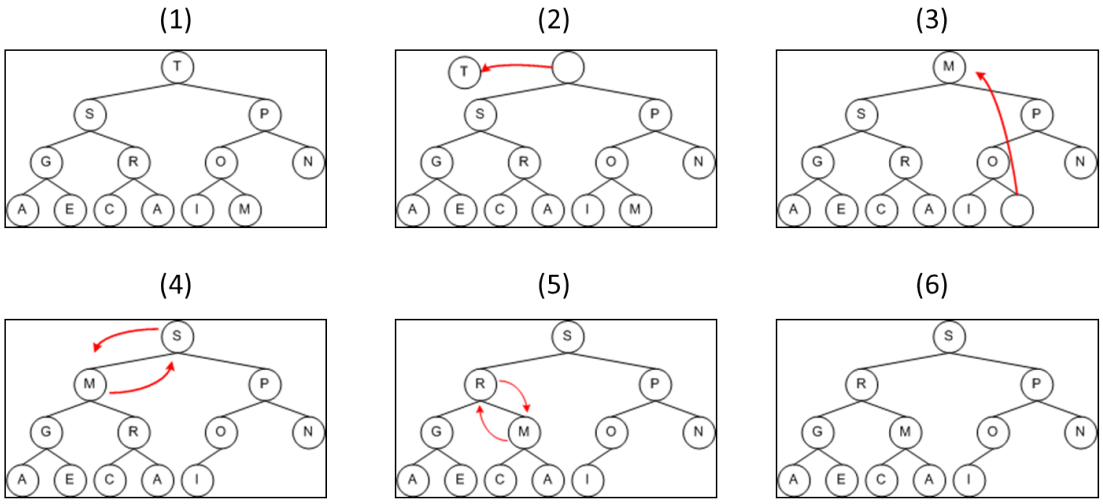


FIGURE 4.12 – *Les étapes suivies par un algorithme de tri par tas pour retirer l'élément maximal*

L'opération d'insertion d'un élément respecte le même schéma. Après avoir inséré l'élément souhaité au nœud le plus à droite dans le dernier niveau de l'arbre, on commence les opérations de permutation nécessaires afin de respecter le principe des tas.

Pour passer d'un arbre binaire tassé à une file de priorité, on numérote les nœuds de l'arbre en largeur de gauche à droite. On utilise ensuite ces numéros comme indices dans un tableau. La figure 4.11 illustre cette opération.

Les positions des différents nœuds dans le tableau sont définies ainsi :

- La racine est le nœud d'indice 0
- Pour chaque nœud d'indice i, on définit :
  - L'indice du nœud parent : $^{(i-1)}/_{2}$
  - L'indice du fils gauche : $2i + 1$
  - L'indice du fils droit : $2i + 2$

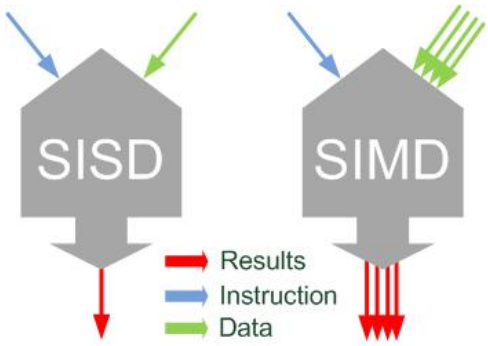


Figure 4.13 – *SISD et SIMD. SIMD exécute la même instruction sur plusieurs données et donne les résultats simultanément*

4.3 L'optimisation par Streaming SIMD Extensions

4.3.1 SIMD

SIMD est l'acronyme pour *Single Instruction Multiple Data*. Il s'agit d'une approche permettant d'améliorer les performances dans les applications qui contiennent un grand nombre d'opérations répétitives. En d'autres termes, *SIMD* offre des techniques pour appliquer la même opération sur un ensemble de plusieurs données simultanément. Cette technique a suscité beaucoup d'intérêt dans les domaines de traitement de l'image et du son [74].

Traditionnellement, lorsque les développeurs ont besoin d'effectuer une opération sur un large ensemble de données, ils utilisent une boucle effectuant la procédure requise pour chaque élément de l'ensemble. Avec chaque itération, un seul élément est traité et une seule opération est exécutée. Ce genre de programmes est appelé *Single Instruction Single Data (SISD)*. La figure 4.13 illustre la différence entre l'approche *SISD* et l'approche *SIMD*.

SISD est facile à implémenter. Cependant, ses boucles sont généralement inefficaces en termes de temps de calcul, car elles doivent parcourir des milliers, voire des millions de fois la base de données. Idéalement, pour augmenter les performances, le nombre d'itérations d'une boucle doit être réduit, et c'est exactement ce que *SIMD* permet de faire.

4.3.2 SSE

Les SSE sont un ensemble d'instructions SIMD développés par Intel. Les origines des SSE remontent aux instructions MMX, la première famille d'instructions SIMD qu'on pouvait trouver dans les premiers ordinateurs équipés de processeurs de la famille Pentium. Les SSE ont été publiées en 6 versions. Chaque version a ajouté de nouvelles fonctionnalités (instructions) et en a amélioré d'autres, aussi bien au niveau matériel que logiciel. Les SSE ont été introduites avec le Pentium III en 1999, mais leur utilisation est restée limitée dans de nombreux domaines, alors qu'elles apportaient des améliorations significatives par rapport à MMX. L'ajout des opérations sur des nombres à virgule flottante avec SSE3 a réveillé l'intérêt pour cet ensemble d'instructions, particulièrement grâce aux instructions horizontales.

L'utilisation d'instructions SSE a de nombreux avantages. Tout d'abord, elles ne nécessitent aucun équipement spécial (à l'exception d'un processeur compatible [2]). En outre, elles permettent de réaliser une grande amélioration des performances des programmes. Mais la mise en œuvre d'algorithmes basés sur SSE reste plus difficile que celle des algorithmes traditionnels. Les SSE doivent dont être utilisées dans de petites parties critiques de l'algorithme.

4.3.3 Analyse du problème

Après cette courte présentation des SSE et des méthodes SIMD, on va analyser le problème du temps de calcul nécessaire à l'étape du scan-matching dans notre algorithme de SLAM. On présente ensuite la manière dont on règle ces problèmes avec les SSE.

Dans cette partie, on ne modifie pas l'algorithme, mais on utilise les possibilités offertes par les processeurs pour exécuter des opérations plus rapides. Cette approche permet d'améliorer la vitesse d'exécution de l'algorithme de mise en correspondance quelle que soit la technique de calcul utilisée.

La préparation du scan-matching

Ayant deux ensembles de données (scans) issues d'un capteur laser, le but du scan-matching est de trouver une rotation R et une translation T qui permettent de mettre en évidence les parties communes des deux scans. Pour effectuer cette tâche, on génère différentes hypothèses sur la position de capture du deuxième scan, en partant de la position de capture du premier scan du laser. Ces positions sont ensuite évaluées en fonction d'une distance à minimiser. Ceci implique l'application d'une rotation et d'une translation à chaque point du nouveau scan, pour chaque hypothèse générée. La figure 4.14 illustre cette opération.

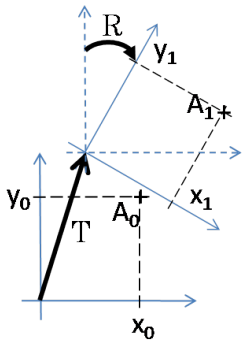


Figure 4.14 – *Les opérations de rotation R et de translation T sont appliquées sur chaque point des données du scan laser*

Ces opérations correspondent à l'équation suivante :

2. Il y a un grand nombre de processeurs compatibles

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} \cos(\theta_R) & -\sin(\theta_R) \\ \sin(\theta_R) & \cos(\theta_R) \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} x_T \\ y_T \end{bmatrix}$$

Le calcul de cette équation inclut 4 additions et 4 multiplications. On convertit ensuite les nombres flottants du résultat $\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$ en entiers. Cette conversion permet de comparer ces données à celles contenues dans la carte créée ou à celles d'un ancien scan laser. Les opérations de scan-matching sont équivalentes à des projections des données du nouveau scan laser en suivant les hypothèses de position. C'est une petite opération simple exécutée des millions de fois dans le processus de localisation de l'algorithme de SLAM. L'amélioration de cette étape permet des gains considérables en temps de calcul.

La conversion d'un flottant en un entier

Le langage C n'étant pas adapté à la conversion des nombres à virgule flottante en entiers, la communauté scientifique utilise plusieurs méthodes et astuces pour effectuer ce type de conversions. La première méthode (et la plus utilisée) consiste à tronquer le flottant en entier directement par l'ajout de la directive (int) avant le nombre à convertir. Cette opération est dangereuse et peut produire des résultats biaisés [3], car les valeurs obtenues ne correspondent pas toujours à l'entier le plus proche. Les valeurs -0.99 and 0.99 par exemple sont converties à 0.

La méthode correcte la plus couramment utilisée pour la conversion d'un flottant vers un entier est basée sur l'utilisation de la fonction $floor()$, en ajoutant 0.5 au nombre qu'on souhaite convertir. Cette solution offre l'entier le plus proche, mais elle a l'inconvénient de se baser sur une opération d'appel de fonction, qui est une lourde tâche au niveau de la consommation du temps de calcul pour le processeur.

Dans une implémentation basée sur les SSE, on peut utiliser l'instruction :

$$\_mm\_cvtps\_pi32()$$

Cette instruction permet de convertir deux flottants en entiers en un seul coup d'horloge du processeur, tout en garantissant un résultat correct et rapide.

4.3.4 Utilisation des SSE

L'utilisation des SSE se base sur des registres à 128 bits. Chacun des registres contient 4 valeurs de 32 bits qu'on va pouvoir traiter simultanément. La figure 4.15 montre le principe suivi dans cette opération.

L'idée derrière l'utilisation des SSE se base sur la parallélisation des opérations de calcul lourdes du scan-matching, notamment les multiplications et les conversions des flottants en entiers. La méthode suivie est représentée dans la figure 4.16.

3. Rappelez-vous l'accident d'ARIAN V, dû à une erreur de conversion [?, 75]

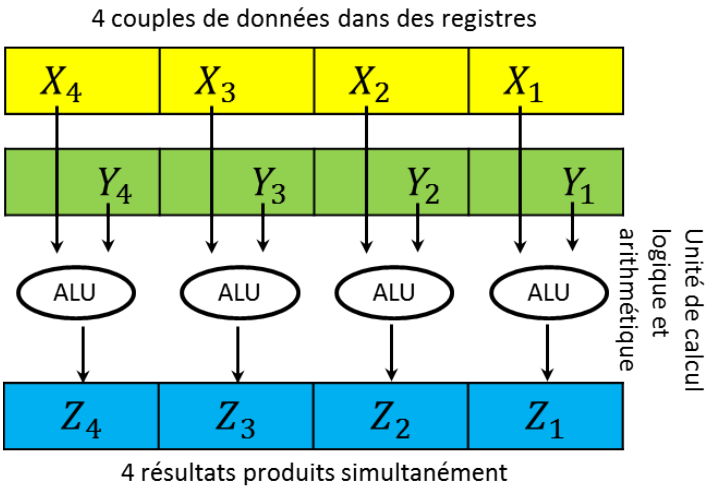


Figure 4.15 – *Le principe de base des calculs utilisant les SSE*

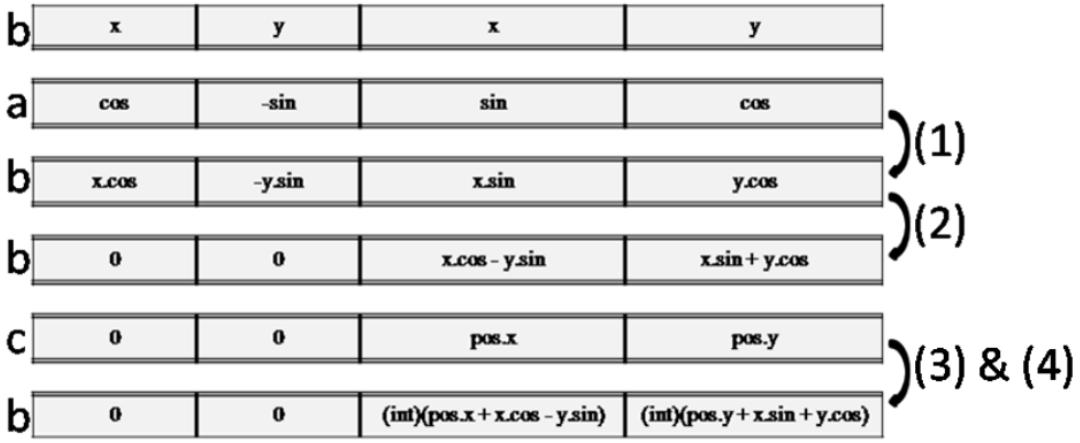


Figure 4.16 – *Les différentes étapes du scan-matching utilisant les SSE*

On utilise trois registres a, b et c. Chacun d'eux est à 128 bits. Afin d'obtenir le résultat final, on applique 4 instructions à ces registres.

1. *_mm_mul_ps(a, b)*. Elle permet la multiplication, terme par terme, de chacun des sous-registres à 32 bits de a et b. Le résultat est ensuite stocké dans le registre b.
2. *_mm_hadd_ps(b, b)*. Elle effectue l'addition de chacun des deux sous-registres 32-bits du registre b et met le résultat dans les deux plus hauts sous-registres 32-bits de b. Il s'agit d'une instruction à fonctionnement horizontal. Le fonctionnement horizontal permet une implémentation simple et rapide du scan-matching basé sur SSE.
3. *_mm_add_ps(b, c)*. Cette instruction effectue l'addition terme à terme entre les deux plus hauts sous-registres 32-bits de b et c. Les résultats sont placés dans b.
4. *_mm_cvtps_pi32(b)*. Durant cette étape, on convertit les éléments du registre b à des entiers.

Le code source présenté dans 4.3 permet d'avoir plus de détails concernant l'implémentation de cette méthode en langage C.

Algorithme 4.3 Implémentation basé sur SSE

```
typedef union {
    struct { int y; int x; } pos;
    __m64 mmx;
} cs_pos_mmx_t;
//Définition des registres
//Registre a contenant les opérateurs de sinus et cosinus
a = _mm_set_ps(part->c, -part->s,
               part->s, part->c);
//Registre c contenant les coordonnées de la position
c = _mm_set_ps(part->map_pos_x,
               part->map_pos_y,
               part->map_pos_x,
               part->map_pos_y);
cs_pos_mmx_t pos;
//Registre b contenant les coordonnées des points du Laser
b = _mm_set_ps(scan->x[j],
               scan->y[j],
               scan->x[j],
               scan->y[j]);
//Calculs
b = _mm_mul_ps(a, b);
b = _mm_hadd_ps(b, b);
b = _mm_add_ps(b, c);
//Conversion
pos.mmx = _mm_cvtps_pi32(b);
//Récupération des résultats
x = pos.pos.x;
y = pos.pos.y;
```

Les résultats de l'utilisation des SSE dans notre algorithme de SLAM seront présentés dans le chapitre 6.

4.4 Le SLAM parallèle

La qualité de la localisation dépend fortement de certains paramètres de l'algorithme de SLAM, ainsi que de l'environnement du robot. Dans *tinySLAM* par exemple, le paramètre TS_HOLE_WIDTH, qui représente la taille du trou dans la carte (figure 3.12 et section 3.5.1), permet d'avoir une localisation plus ou moins précise, selon les valeurs de ce paramètre, et les déplacements du robot. La différence au niveau de la qualité de la localisation n'est pas toujours facile à distinguer visuellement, mais les logs de l'algorithme permettent facilement de la voir. La figure 4.17 illustre ces propos. Les deux courbes représentent les scores des meilleures positions estimées en utilisant deux valeurs différentes du

paramètre TS_HOLE_WIDTH. Ces deux courbes ont été réalisées pour les données de la carte représentée dans la figure 6.5. En comparant les cartes obtenues visuellement, on ne peut pas distinguer de différences, mais les courbes montrent clairement les changements au niveau de la qualité de la localisation entre les deux cartes. Par exemple, dans la zone 1, la première carte (Map 1) offre une bonne qualité de localisation. Dans la zone 2, c'est la deuxième carte (Map 2) qui permet une bonne localisation.

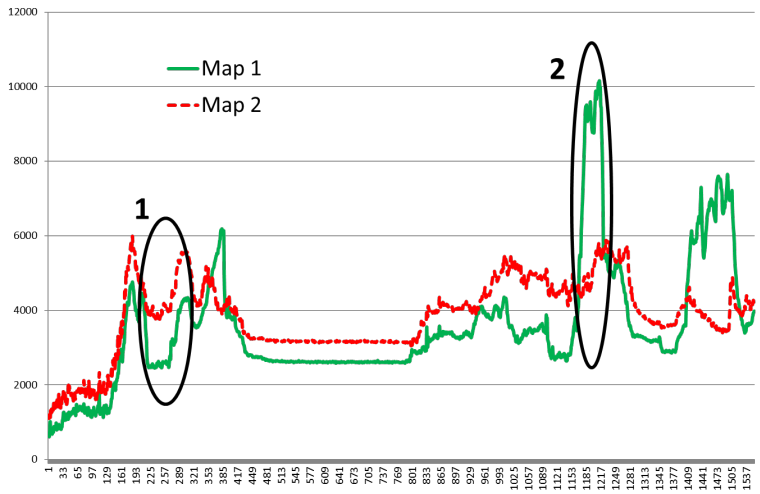


FIGURE 4.17 – *Comparaison des scores des meilleures estimations de position pour deux valeurs différentes du paramètre TS_HOLE_WIDTH*

On a ainsi implémenté un processus de localisation parallèle, qui permet d'adapter *tinySLAM* à plusieurs situations dans l'environnement à cartographier. Cette capacité d'adaptation est due principalement à l'utilisation de deux valeurs différentes du paramètre TS_HOLE_WIDTH, ce qui donne deux cartes différentes.

La précision des cartes créées change selon les déplacements du robot. Certaines positions du robot offrent un ensemble de données laser permettant une localisation facile, tandis que les données issues d'autres positions sont plus difficiles à exploiter. Durant la localisation parallèle, l'algorithme estime la position du robot en utilisant les deux cartes dont il dispose. Chacune des deux cartes offre une position avec une estimation de la qualité de la position trouvée. On garde la meilleure des deux positions, et on met à jour les deux cartes en utilisant cette meilleure position. On corrige ainsi les deux cartes, ce qui permet de continuer à utiliser ce parallélisme de la localisation. La figure 4.18 illustre cette opération.

4.5 Conclusion

CoreSLAM est basé sur le principe d'IML, il est ainsi divergent par construction. Ce chapitre présente les méthodes d'optimisation permettant de limiter cette divergence au maximum. Durant le processus de scan-matching, l'algorithme génère plusieurs hypothèses concernant la position du robot, et choisit la meilleure hypothèse en se basant sur son

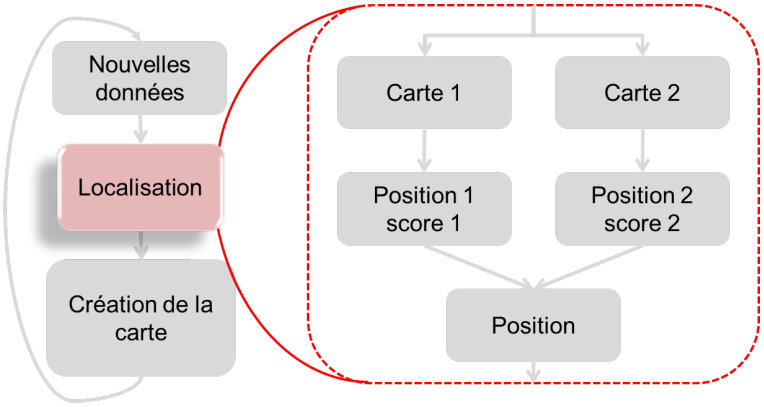


FIGURE 4.18 – *Le processus de localisation utilise deux paramètres différents pour créer deux cartes. Il met à jour la position ensuite en utilisant le meilleur résultat*

score. La méthode de génération des hypothèses peut avoir un grand impact sur la vitesse et l'efficacité de l'algorithme. Nous détaillons trois algorithmes de génération d'hypothèses : l'algorithme de Monte Carlo, l'algorithme génétique et l'algorithme multi-échelle. Nous présentons ensuite la file de priorité, l'un des éléments clés ayant permis d'accélérer grandement le fonctionnement de *CoreSLAM* (voir chapitre 6 pour les résultats). Cette technique permet de cibler plus efficacement les meilleures hypothèses. Dans la continuité de l'idée d'accélération du fonctionnement de *CoreSLAM*, nous avons profité des possibilités offertes par le matériel. Nous avons utilisé les méthodes basées sur les Streaming SIMD Extensions. Il s'agit de directives utilisées dans les parties critiques de l'algorithme afin d'effectuer des calculs parallèles. Nous finissons le chapitre par la présentation d'une technique permettant d'améliorer la qualité. En effet, la partie de localisation de *CoreSLAM* est devenu tellement rapide qu'on s'est autorisé à l'exécuter deux fois simultanément, avec des paramètres différents. *CoreSLAM* garde ensuite le meilleur des résultats. Cette localisation parallèle permet d'adapter *CoreSLAM* à plusieurs situations différentes (en utilisant les meilleurs paramètres pour chaque situation).

Les vérités sont plongées dans des incertitudes [et les autorités scientifiques ne sont] pas à l'abri de l'erreur

Alhazen Ibn Al-Haytham

5 Expérimentations

Afin de valider la qualité de localisation et de cartographie de *CoreSLAM*, nous avons procédé à plusieurs tests et expérimentations. Ce chapitre présente les conditions des expérimentations ainsi que les principales plates-formes utilisées.

Sommaire

5.1 Le Mines Rover

Le Mines Rover est un robot à 6 roues, dont 4 sont motrices et directrices (voir figure 5.1), conçu autour d'une architecture en Rocker-Bogie (figure 5.2) similaire à celle du Sojourner de la NASA (mission Mars Pathfinder) qu'on voit sur la figure 5.3.

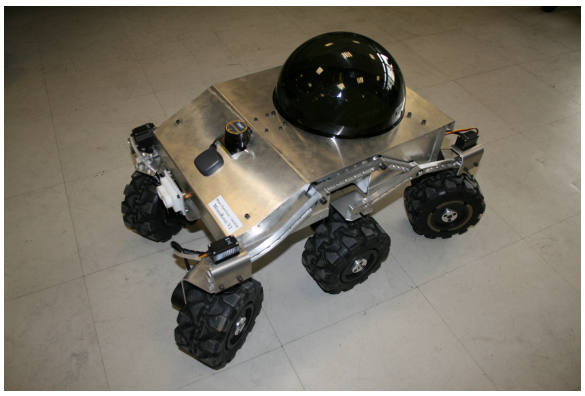


Figure 5.1 – *La plate-forme Mines Rover. On peut apercevoir la coupole de la caméra, les servomoteurs de rotation, le laser HOKUYO URG-04LX, le récepteur GPS (le carré gris) et les capteurs à ultrasons d'arrêt d'urgence à l'avant du robot*

Les deux roues non motrices centrales du Mines Rover sont ainsi maintenues au sol y compris sur terrain accidenté. Ces deux roues sont dans notre cas équipées d'odomètres à 2000 points par tour qui procurent à notre plateforme une très bonne odométrie et l'aide à éviter le problème de glissement.

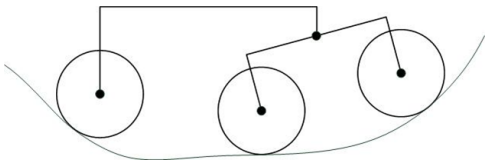


Figure 5.2 – *L'architecture mécanique du Mines Rover est basée sur une articulation du type Rocker Bogie*

Le Mines Rover est équipé d'un télémètre laser Hokuyo URG-04LX, d'une caméra IP couleur Pan-Titl-Zoom AXIS 213, d'une centrale inertielle 5-axes (3 accéléromètres, 2 gyromètres), d'un compas électronique et de proximètres à ultrasons (utilisés uniquement pour l'arrêt d'urgence). Il est également équipé d'un GPS assurant une bonne localisation à l'extérieur.

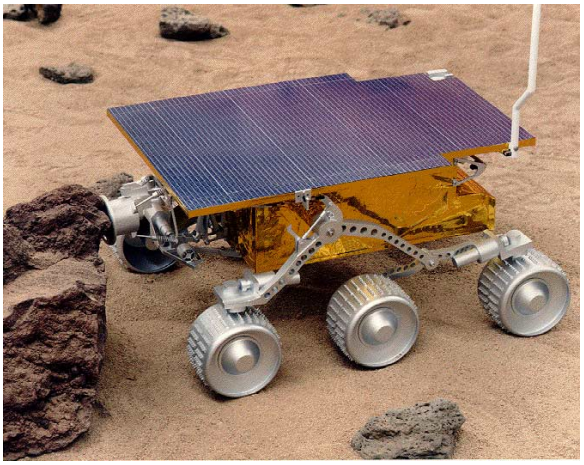


FIGURE 5.3 – *Le robot Sejourner de la mission Mars Pathfinder*

Au niveau électronique, seuls des composants sur étagère sont utilisés : des ponts en H SaberTooth sont utilisés pour l'alimentation en puissance des moteurs à courant continu et des servomoteurs de modélisme Hitec sont utilisés pour le pilotage de la direction des roues. Les 4 moteurs CC à 45W assurent une vitesse maximale de 3 m/s au Mines Rover. On note que le robot a atteint une vitesse linéaire de 2.5 m/s et une vitesse angulaire de 150 °/s. Cette grande vitesse est assurée par les 4 roues directrices.

Au niveau de l'informatique embarquée, la figure 5.4 montre que le Rover est conçu autour d'une Qwerk (liée au projet Terk de Carnegie Mellon University [76]), qui intègre un processeur ARM9 et un FPGA Xilinx, seuls éléments programmables du robot. Le FPGA est utilisé pour l'acquisition des capteurs (bus I^2C, encodeurs optiques) et le pilotage des actionneurs (sorties servomoteurs). Le télémètre laser et le GPS sont connectés au processeur ARM par USB. L'alimentation est assurée par une batterie Lithium-Polymer de 4Ah assurant une autonomie de 8 heures en veille et de 20 minutes en fonctionnement actif [1].

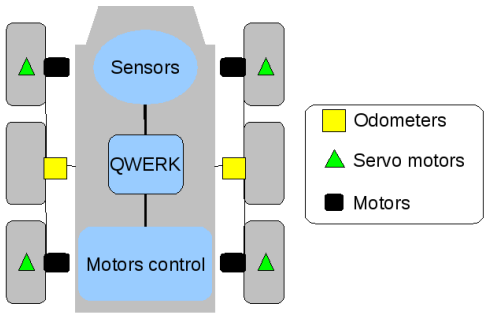


FIGURE 5.4 – *Schéma de l'architecture du Mines Rover. C'est un robot 6 roues, avec 4 roues motrices et directrices, et deux roues libres équipées d'odomètres à 2000 points. Le module Qwerk est au centre du robot. Son microprocesseur cadencé à 200 Mhz peut assurer la gestion de tous les actionneurs et les capteurs avec fiabilité*

1. Un fonctionnement actif du robot implique une vitesse moyenne de 2 m/s avec une gestion de tous les capteurs embarqués

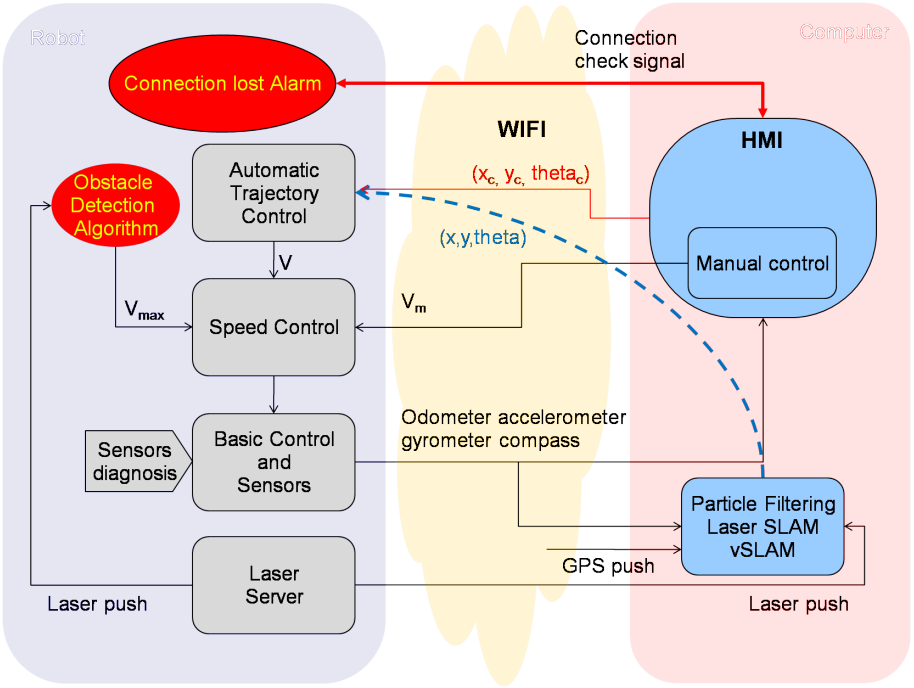


Figure 5.5 – *Diagramme représentant l'architecture logicielle du Mines Rover.*

Au niveau logiciel, l'intégralité du code embarqué dans le robot a été écrite au laboratoire de robotique CAOR à Mines ParisTech. L'architecture de service utilisée est nommée *Cables* et elle est issue d'un projet ANR appelé *AROS*. Elle facilite énormément l'utilisation du robot en rendant accessibles les données capteurs et les actionneurs à tout ordinateur connecté en réseau. De plus, Cables étant compatible Zeroconf, tout robot utilisant Cables est directement reconnu et accessible par son nom (pas de numéro IP ni de port à configurer). Le système d'exploitation utilisé est une distribution Linux spécialement compilé pour la Qwerk, afin d'en exploiter toute la puissance. Le code applicatif embarqué sur le robot est schématisé sur la figure 5.5.

Le logiciel du robot intègre un asservissement de vitesse fin, prenant en compte la tension d'alimentation de la batterie et le modèle du robot. L'asservissement en position utilise à l'heure actuelle un algorithme classique de contrôle par retour d'état [77]. Le code intègre par ailleurs toutes les alarmes nécessaires à l'utilisation en toute sécurité du robot : la détection de perte de transmission entraîne l'arrêt du robot, et la détection d'obstacle limite sa vitesse voire l'empêche d'avancer, et ce au plus bas niveau de contrôle. Le reste de la partie applicative (SLAM, navigation etc.) est déportée hors du robot (seulement pour le Mines Rover ayant une faible puissance de calcul), ce qui facilite le développement des applications dans la mesure où on n'a pas besoin de toucher au code du robot pour pouvoir l'exploiter.

Sur la figure 5.5, on a un diagramme simplifié du système gérant le Mines Rover. À gauche, on trouve le système embarqué et à droite le logiciel tournant sur PC. La communication est assurée par liaison WIFI. Les calculs liés à l'intelligence artificielle de haut niveau (SLAM, filtrage, traitement d'images) sont exécutés sur le PC distant, car la

carte Qwerk n'a pas assez de puissance de calcul pour les effectuer. On y exécute par contre toute la partie liée à l'intelligence de bas niveau (contrôle de la vitesse, gestion des alarmes, des situations d'urgence classiques...). Afin d'assurer une bonne qualité de connexion, on utilise un routeur MIMO sur le robot.

5.2 Le défi CAROTTE et l'utilisation des robots Wifibot

5.2.1 Le défi CAROTTE

5.2.1.1 Introduction

Le Centre de Robotique Mines ParisTech-CAOR, laboratoire au sein duquel s'est déroulé ma thèse a développé un partenariat pour participer au concours CAROTTE (CArtographie par RObot d'un Territoire) de l'Agence National de la Recherche (ANR) et la Délégation Générale de l'Armement (DGA). Le partenariat a dérivé dans la création d'une équipe de travail nommé CoreBots. Cette équipe est constituée de partenaires académiques (écoles et laboratoires de recherche) et de petites entreprises spécialisées dans la robotique. Les partenariats sont les suivants :

- Mines-Paristech (ENSMP)
- Intempora
- INRIA / IMARA
- EPITECH

5.2.1.2 Le consortium

Mines ParisTech L'École Nationale Supérieure des Mines de Paris ENSMP est chargée originellement de la formation des ingénieurs civils des mines et du Corps Techniques de l'Etat. L'École a développé depuis les années soixante des activités de recherche et d'enseignement de troisième cycle, en liaison avec l'industrie et avec l'aide de l'association ARMINES. L'École regroupe aujourd'hui 263 enseignants chercheurs, 444 doctorants et environ 750 étudiants qui sont regroupés non seulement sur le site de Paris mais aussi sur Fontainebleau, Evry et Sophia-Antipolis.

Armines Créée en 1967 à l'initiative de l'Ecole des Mines de Paris, ARMINES est une association de recherche contractuelle, partenaire de grandes Ecoles d'Ingénieurs. Elle a pour objet la recherche "orientée vers l'industrie" et apporte à ses centres de recherche communs aux écoles, des moyens en personnel, équipement et fonctionnement à hauteur de son volume d'activité contractuelle. Près de 600 personnes salariées d'ARMINES (enseignants chercheurs, ingénieurs de recherche, doctorants, post-docs, techniciens, personnels administratifs) animent la recherche aux côtés des collègues des écoles, au sein de centres de recherche « communs » ARMINES/écoles.

Figure 5.6 – *Le robot CoreBot 1 au point de démarrage de sa mission*

Intempora Intempora est une spin-off de l'Ecole des Mines de Paris. Intempora édite le logiciel RTMaps, un environnement de développement rapide pour les applications multi-capteurs temps-réel notamment dans les domaines de la robotique mobile, de l'automobile, etc. RTMaps permet le développement modulaire d'applications mettant en œuvre des capteurs asynchrones et hétérogènes (caméras, têtes de stéréovision, bus CAN, GPS, télémètres laser, radars, audio, etc.). RTMaps est utilisé aujourd'hui par des industriels et laboratoire de recherche tels que THALES, RENAULTS, PSA, le SwRI, l'INRIA, le LCPC, l'INRETS...

INRIA L'INRIA, Institut National de Recherche en Informatique et Automatique, est un centre national de recherche regroupant les compétences scientifiques publiques dans les domaines des sciences informatiques et du contrôle.

Epitech L'Ecole pour l'informatique et les nouvelles technologies (EPITECH) aussi appelée European Institue of Technology, est un établissement privée français d'enseignement supérieur en informatique et nouvelles technologies.

Pour la première année du défi (2010), l'équipe a fait concourir un robot Wifibot 4 roues équipé d'une carte Core 2 Duo, de deux télémètres laser Hokuyo (un laser 30m et un laser 4m), d'une caméra FireWire DCAM, et d'une centrale inertielle.

Les développements ont été menés sur le middleware modulaire et distribué *Cables* . L'architecture logicielle développée est constituée de plusieurs modules distribués autour d'un module central en charge de la gestion de la carte.

D'autres outils logiciels ont été utilisés comme le simulateur de robots Marilou de la société AnyKode, l'outil RTMaps de Intempora. La société Wifibot est également impliquée par l'intermédiaire de Laurent Bouraoui, dirigeant de Wifibot et ingénieur de recherche à ARMINES (Mines ParisTech).

Pour l'édition 2011 du Défi Carotte, l'équipe CoreBots a présenté un nouveau robot profondément modifié avec :

– une nouvelle plate-forme matérielle, le CoreBot M (voir Figure 5.7). Comme le robot 2010, il est capable d'évoluer en extérieur mais, doté de 6 roues, il possède de plus

Figure 5.7 – *Le robot CoreBot M*

grandes capacités de franchissement et se rapproche encore davantage d'un produit fini. Il est équipé d'une carte Core I7 embarquée sous linux, d'un télémètre laser Hokuyo 30m, d'une caméra 3D Kinect, d'une centrale inertielle VectorNav et d'un ultrason directif.

- un nouveau capteur : la Kinect. Ce capteur bas coût présente de très bonnes performances et nous permet de reconnaître les objets grâce à des acquisitions 3D couleur, mais aussi d'éviter les obstacles proches et de reconstruire l'environnement 3D du robot.
- Une nouvelle version des logiciels embarqués, pour le SLAM (CoreSLAM2), la planification de trajectoire (CoreControl2), la reconnaissance des objets (Core3DLearner et Core3DAnalyzer), dans une architecture toujours modulaire et fondée sur le middleware Cables.
- Une nouvelle stratégie d'exploration de l'environnement plus exhaustive.

5.2.1.3 Le défi

Que ce soit en milieu naturel ou urbain, il est fréquent que l'environnement dans lequel évoluent les robots soit mal connu ou/et évolutif. Cette incertitude est préjudiciable à la réalisation des missions confiées aux robots. Un champ particulièrement important concerne l'exploration de zones dangereuses.

Dans ce contexte, de petits engins terrestres non habités peuvent être utilisés pour suppléer l'homme grâce à leurs capacités de reconnaissance. Une des facultés clé de ces

Figure 5.8 – *Logo du défi CAROTTE.*

systèmes robotisés est leur capacité à collecter de l'information sur leur environnement, et de l'analyser afin de fournir des informations sur la configuration des lieux (cartographie) et la reconnaissance et localisation d'objets d'intérêt. L'autonomie maximale des robots doit aller de pair avec la robustesse du système vis-à-vis par exemple des interruptions de communication. Pour améliorer les capacités de localisation, de cartographie de bâtiments et d'analyse de terrain en milieu urbain, la DGA et l'ANR ont initié un défi intitulé CAROTTE. Plus précisément, les objectifs de ce défi sont de :

- Faire progresser l'état de l'art en robotique dans le domaine perception – cognition.
- Susciter des rapprochements entre des roboticiens et des chercheurs/industriels issus de domaines connexes.

Il s'agit de rassembler des équipes qui s'affrontent autour du défi de réaliser un système robotisé autonome, capable de s'orienter dans un espace clos et de reconnaître des objets présents dans ce local, ce qui lui permettra de réaliser une cartographie accompagnée d'annotations sémantiques d'un espace inconnu.

Le défi s'étend sur trois ans. La première phase, d'une durée d'un an, correspond à la réalisation de ces systèmes. Elle se termine par une compétition courant juin 2010 suivie de l'analyse des résultats. La deuxième phase consiste en des développements supplémentaires en vue de la participation à la deuxième compétition (juin 2011) plus complexe. La troisième phase est similaire à la précédente avec une troisième compétition (juin 2012). Corebot remporta les deux premières éditions du challenge (2010 et 2011).

Description de la mission Chaque équipe dispose de 30 minutes pour effectuer la mission définie dont les principaux éléments sont :

- Le système robotisé devra reconnaître l'ensemble des pièces et des objets présents dans l'arène : il devra ainsi cartographier les pièces, les dénombrer ainsi que détecter, identifier et localiser les objets.
- Un objet défini (ex. ballon rouge) devra être trouvé dans l'arène et touché par le robot. Une clé devra également être localisée.
- Le temps mis pour effectuer la mission sera chronométré.
- A l'issue de sa mission, le système robotisé doit être en mesure de produire une cartographie avec des annotations sémantiques des lieux visités, localisant les différents objets et les obstacles, ainsi que des fichiers de log. Durant la mission le système est autonome : aucune intervention sur le système robotisé n'est autorisée.

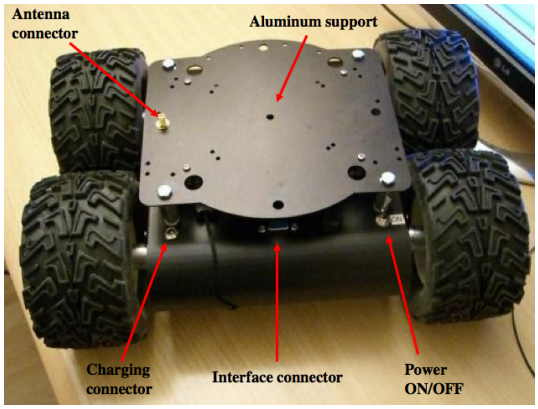


FIGURE 5.9 – *La plateforme Wifibot*

5.2.2 Wifibot Lab V2

Le système robotisé CoreBot 1 est basé sur une plateforme ouverte du commerce le WIFIBOT LAB V2 [2]. C'est un robot 4 roues motrices de petite taille qui peut facilement être transportable par un seul homme (voir la figure 5.9).

Ce genre de plates-formes offre un rapport coût/technologie intéressant et dispose d'outils de programmation simples, complets et efficaces. Le robot Wifibot utilisé est un robot modulaire, on a ainsi pu lui ajouter plusieurs actionneurs et capteurs afin d'atteindre les objectifs fixés dans le défi CAROTTE.

Nous l'avons équipé de plusieurs capteurs de perception et d'une batterie NIMH 12V :

- Capteur Laser UTM-30LX (voir la section 3.2) : grâce à sa portée de 30 mètres et sa position horizontale, ce capteur permet au robot d'effectuer les tâches de localisation et de cartographie de l'environnement.
- Capteur Laser URG-04LX (voir la section 3.2) : ce capteur Laser est placé sur un support incliné (voir la figure 5.10), afin de permettre au robot de détecter les obstacles non détectable par le capteur Laser horizontal.
- Caméra IEEE1394 : Cette caméra est équipé du FireWire, elle est très compacte et dotée de capteurs à haute sensibilité (CMOS, CCD). Elle possède des modèles monochromes et couleurs. Elle offre une excellente qualité d'images. Ce capteur est utilisé pour l'identification d'objets dans l'arène
- IMU : il s'agit d'une centrale inertielle VectorNav VN-100. Elle combine un accéléromètre, un gyromètre et un magnétomètre 3 axes.

Le robot CoreBot 1 (voir la figure 5.11) utilise un PC embarqué industriel basé sur un processeur Intel Duo Core sous Linux. Mais une version basse consommation Atom N270 a aussi été testée pour valider le portage des algorithmes sur des cibles à faible coûts et faible consommation électrique. Le système de communication est aussi une carte sous Linux, utilisant la technologie MIMO.

2. www.wifibot.com

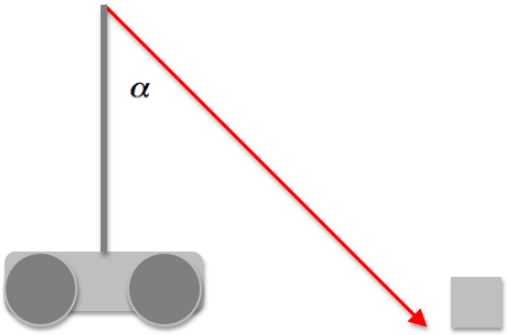


Figure 5.10 – *Le capteur Laser est placé sur un plan incliné, afin de détecter les petits obstacles*

Figure 5.11 – *Le robot CoreBot 1 est basé sur l'architecture Wifibot Lab V2*

Le robot bénéficie du middleware modulaire *Cables*[3]. Il est compatible avec plusieurs systèmes d'exploitation (Windows, Linux et Mac).

L'architecture logicielle développée pour le robot est constituée de plusieurs modules distribués autour d'un module central en charge de la gestion de la carte de l'environnement. Les modules sont les suivants :

- *CoreSLAM1* : Localisation et cartographie par Laser
- *CoreControl1* : Génération et contrôle de la trajectoire pour un robot différentiel à 4 roues
- *CoreCarotte1* : Gestion des missions et contrôle de la stratégie du robot
- *Core3D* : Création du nuage de points 3D et reconnaissance d'objets
- *CoreHMI* : Interfaces graphiques de contrôle et de visualisation
- *Wifibot* : Communication avec la plateforme Wifibot

Control externe/IHM Grâce à l'interface *Cables,* tous les services lancés sur le robot peuvent être contrôlés ou exploités à distance, en s'inscrivant aux services proposés en « Zeroconf ». Le système robotisé est contrôlé par une série de services, tous interconnectés. Une interface web permet de diagnostiquer et de manipuler simplement ces différents services.

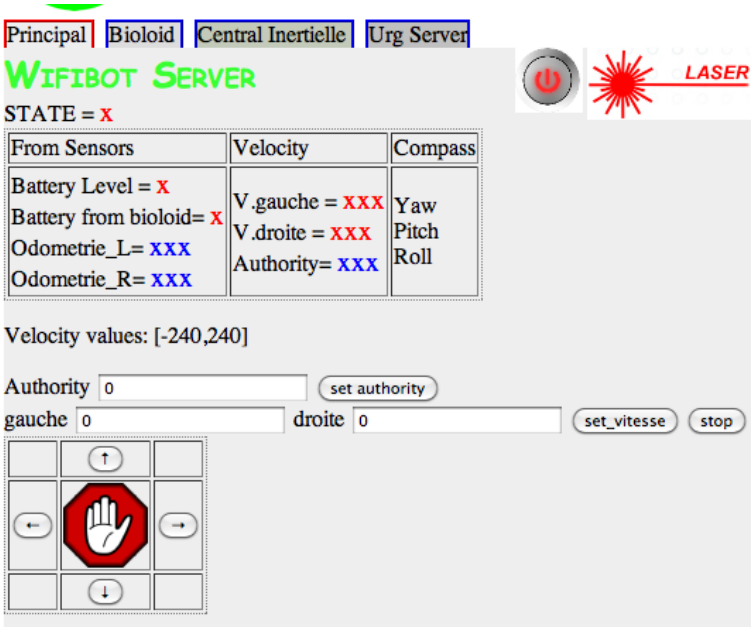


FIGURE 5.12 – *Serveur Web pour le contrôle du Wifibot*

Service Wifibot : Il assure le contrôle de la plateforme

Service Bioloid : Le serveur « Bioloid » propose un ensemble de services pour le contrôle des servomoteurs

Service URG : Ce serveur assure l'acquisition des données des télémètres laser Hokuyo

3. *Cables* est une interface de programmation réseau, services et découverte automatique (Zeroconf). Ce framework est développé dans le cadre du projet ANR *AROS* : Automotive Robust Operating Services

Service IMU : Ce serveur fournit les informations issues de la centrale inertielle

Service CAMERA : Le service Caméra propose les fonctions de récupération de l'image pour les traitements vidéo.

Service SLAM : Le service SLAM gère tout l'aspect localisation, découverte et navigation par « Waypoints » (points de passage)

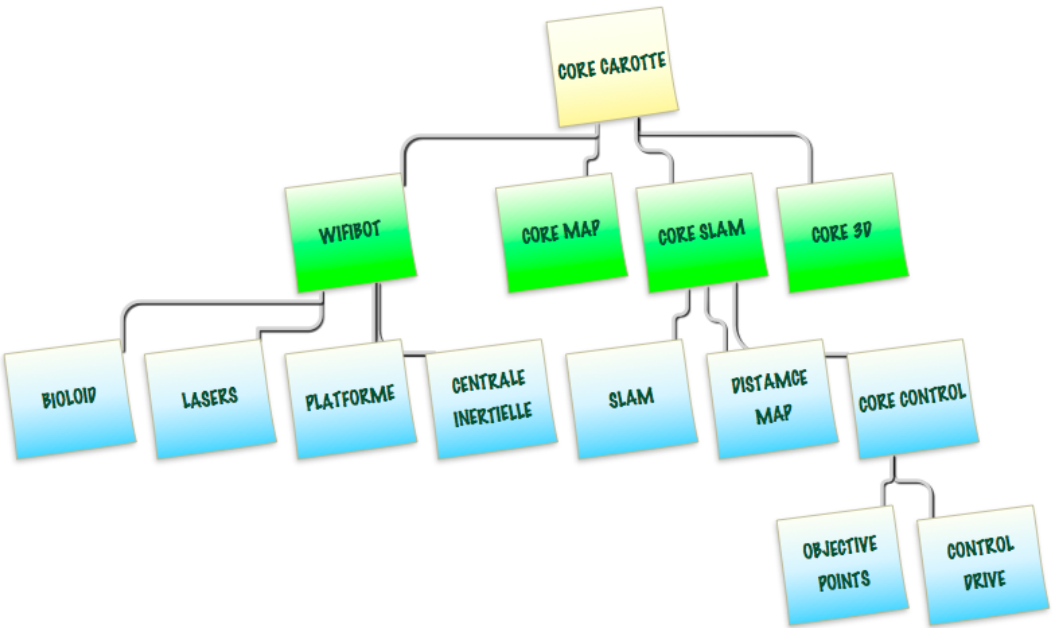


Figure 5.13 – *Arbre présentant la hiérarchie des services fonctionnant sur le robot*

5.2.3 Wifibot Lab M

Description de la plateforme CoreBot M Le système robotisé est basé sur une plateforme ouverte du commerce, le WIFIBOT LAB M. C'est un robot à 6 roues motrices de petite taille facilement transportable par un seul homme. Il est équipé de plusieurs capteurs de perception et d'une batterie LIFE 12.8V 13Ah :

- Capteur Laser 30m
- Kinect
- IMU (centrale inertielle) VectorNav
- Capteur à ultrasons

Figure 5.14 – *Robot Corebot M.*

Le robot utilise un PC embarqué industriel basé sur un processeur Core i7 sous Linux. Le système de communication est aussi une carte sous Linux, utilisant la technologie MIMO.

Voici plus de détails sur les nouveaux capteurs intégrés au CoreBot M pour l'année 2011 comparé à l'édition 2010 du concours CAROTTE : centrale inertielle, capteur à ultrasons et caméra 3D Kinect.

Centrale Inertielle La centrale inertielle du fabriquant VectorNav permet au robot de gérer les pentes et les changements de niveaux. Elle intègre un filtrage de Kalman (voir figure 5.16) qui permet d'obtenir l'attitude du robot à 200Hz. Ses caractéristiques complètes sont données dans le tableau 5.1.

Capteur Ultrason Pour détecter la présence de miroirs, le système robotique utilise un capteur ultrason directif (figure 5.17) à l'avant d'une portée de 7m. Il permet de rajouter sur la carte des obstacles que le système Laser/Kinect ne peut pas distinguer, comme les vitres ou les miroirs.

Ce capteur est dit directif compte tenu de la forme de son faisceau, représentée sur la figure 5.18. Sur cette figure, on a représenté le faisceau du capteur à ultrasons dans une grille de 60 cm.

Capteur 3D Kinect Ce capteur est utilisé pour l'évitement d'obstacle et la reconnaissance d'objets. Avec un angle de vue de 60°, il nous transmet sur le même bus USB une image RGB (ou YUV) et une image de profondeur (cf. figure 5.19).

La section 3.2 présente plus de détails sur ce capteur.

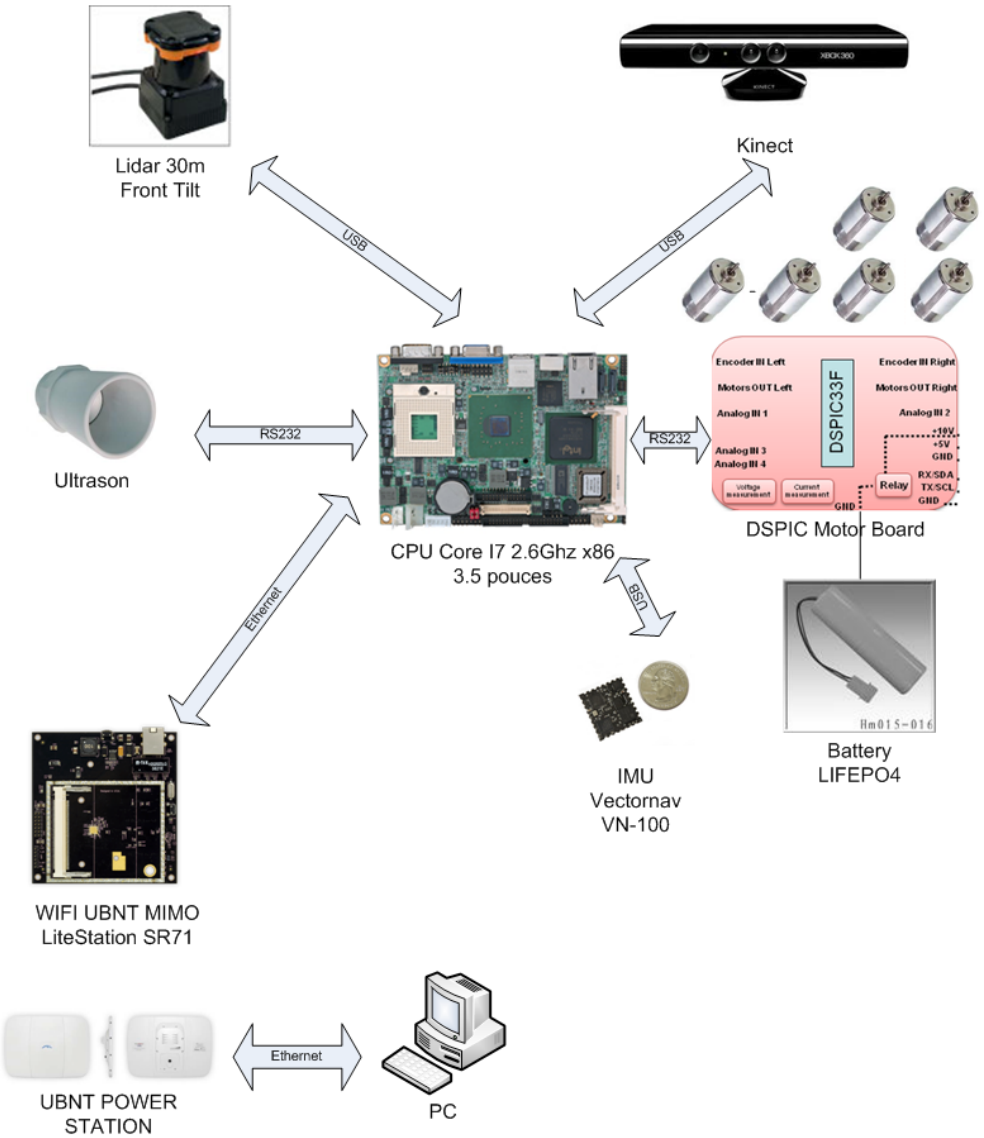


FIGURE 5.15 – *Architecture du système.*

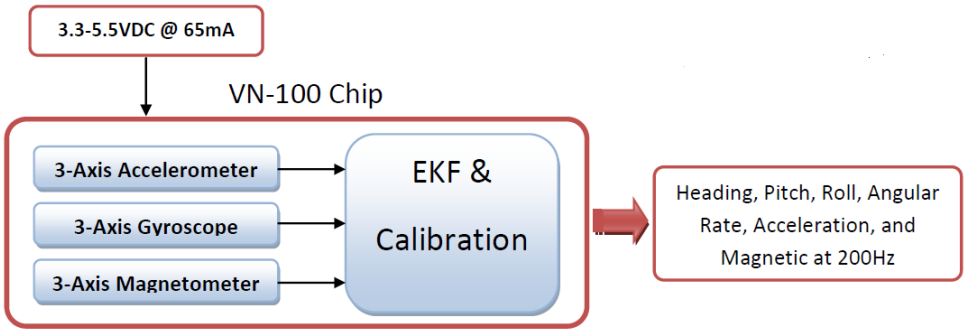


Figure 5.16 – *Principe de fonctionnement de la centrale inertielle VectorNav basé sur un filtrage de Kalman*

Heading	
Range	±180 °
Accuracy (RMS) @ 25°C	< 0.5 °
Accuracy: (3 Sigma) @ 25°C	< 2.0 °
Temp Sensitivity: VN-100	< 0.06 ° / °C
Temp Sensitivity: VN-100T	< 0.005 ° / °C
Resolution:	< 0.05 °

Attitude	
Range: Pitch, Roll	±180 °, ±90 °
Accuracy: (RMS) @ 25°C	< 0.2 °
Accuracy: (3 Sigma) @ 25°C	< 0.5 °
Temp Sensitivity: VN-100	< 0.06 ° / °C
Temp Sensitivity: VN-100T	< 0.005 ° / °C
Resolution:	< 0.05 °

Angular Rate	
Range: Yaw, Pitch, Roll	±500 °/sec
Zero Rate Bias Stability: @ 25°C	< 100 °/hr
Resolution: Heading	< 0.01 °/sec
Resolution: Pitch, Roll	< 0.01 °/sec
Bandwidth:	140 Hz

Acceleration	
Input Range: X/Y/Z	±2 g, ±6 g
Resolution: X/Y	< 0.4 mg
Resolution: Z	< 1 mg
Bandwidth:	50 Hz

Table 5.1 – *Caractéristiques de la centrale inertielle VectorNav.*

Architecture logicielle du CoreBot M La figure 5.20 présente les différents composants logiciels intégrés dans le CoreBot M. Chaque composant est un processus système communiquant avec ses voisins par sockets TCP/IP (grâce au middleware *Cables*), éventuellement à travers de la mémoire partagée (c'est en particulier le cas pour les données issues de la Kinect, qui seraient trop lourdes pour être transmises par sockets). Chaque composant est un serveur *Cables*, interrogeable par HTTP (et donc par un navigateur), capable de renvoyer des informations de diagnostic (mémoire utilisée, clients connectés, trafic échangé). Les composants plus importants sont décrits plus loin dans cette section.

Intégration de la Kinect L'intégration de la Kinect a fait l'objet d'un soin particulier pour la deuxième année du défi CAROTTE. Elle s'appuie sur la réalisation de 3 composants *Cables* dédiés :

– Le *kinect_ server*, qui délivre les images de profondeur et les images RGB ou YUV422 acquises par la Kinect.

Figure 5.17 – *Capteur ultrason.*

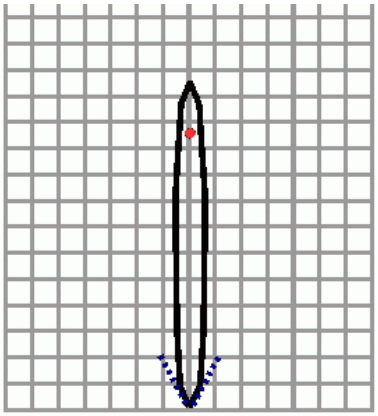


Figure 5.18 – *Le faisceau du capteur à ultrasons dans une grille de 60 cm*

- Le *kinect_ 3d_ server*, qui associe l'image couleurs avec l'image de profondeur et produit un nuage de points texturé (cf. figure 5.21). Ce composant s'appuie sur les techniques de calibration développées par Nicolas Burrus et Konolige pour ROS [78]. Ce composant est configurable en termes de vitesse d'acquisition et de résolution. Il permet aussi à travers le service *"kinect.save"* d'enregistrer des nuages de points.
- Le *kinect_ voxel_ server*, qui effectue la voxelisation du nuage de points. Il repositionne ce nuage de points dans un repère global, utilisant à la fois la position extrinsèque de la Kinect sur le robot, et la position du robot fournie par *CoreSLAM*. Le nuage de points trié et indexé fourni en sortie est prêt à être exploité pour la segmentation des objets, du sol et des murs. Il est également directement utilisé pour la reconstruction 3D de l'ensemble de l'environnement, par simple accumulation. Une somme verticale des voxels est également produite. C'est cette somme, identifiant les objets proches du robot, qui est envoyée à *CoreControl2* pour leur prise en compte dans le calcul de trajectoire et donc l'évitement d'obstacles. Ces différents composants communiquent ensemble grâce à la fonctionnalité de mémoire partagée de *Cables*, c'est à dire à coût quasi nul bien que chaque composant tourne dans son propre processus. Le positionnement précis des données de la Kinect par rapport aux données issues du SLAM est assurée par une synchronisation offline entre les capteurs laser Hokuyo (utilisé pour le SLAM) et la Kinect. Toutes les données sont datées et synchronisées au sein du système par *Cables*.

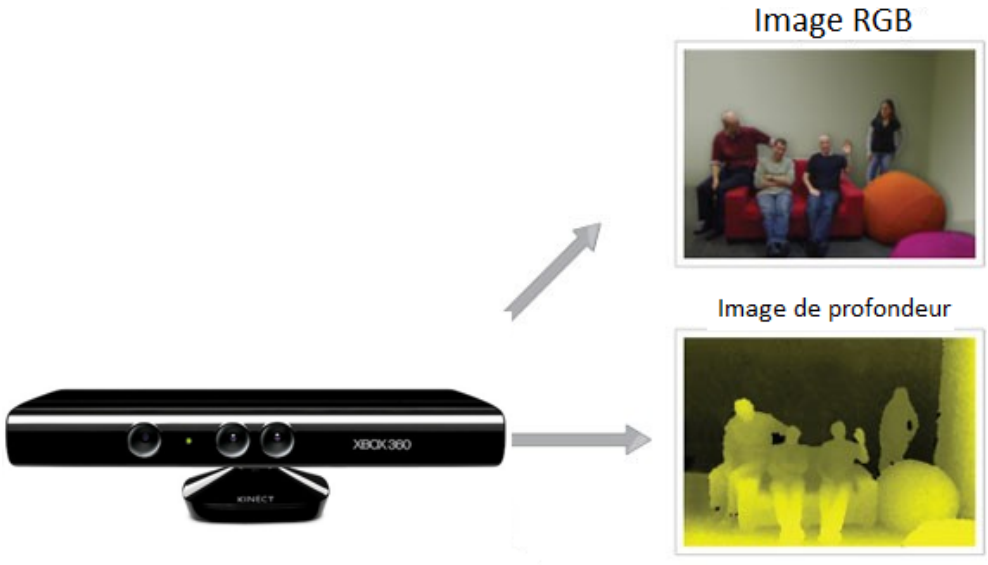


Figure 5.19 – *Données émises par la Kinect.*

CoreControl2 Le générateur de trajectoire et contrôleur de robots *CoreControl2* a été entièrement revu en 2011. La version *CoreControl*1 était conçue pour un robot de forme ronde, alors que le robot CoreBot M a une forme allongée : il ne passe pas latéralement dans une ouverture de porte, et ce point doit être pris en compte par le contrôleur. Il a donc été réalisé un tout nouveau générateur de trajectoire, qui calcule la trajectoire d'un point A à un point B, en considérant l'angle d'arrivée. La génération de trajectoire exploite un algorithme A* [79] qui explore l'espace (x, y, θ), cherchant la trajectoire qui minimise la distance parcourue, tout en assurant le minimum de rotations et conservant une bonne distance aux obstacles. Un tel algorithme est très coûteux, notamment en mémoire, car l'espace de recherche est immense. Afin de le faire tourner en temps-réel sur le robot, quatre optimisations ont été appliquées :

- L'espace des angles est discrétisé suivant 12 directions. Les trajectoires sont donc des segments de droite connectés par des angles de 30° (cf. figure 5.22). Ceci nous permet d'exploiter directement et efficacement notre carte de distance hexagonale (présentée dans 3.5.2.1). Les directions qui ne sont pas alignées sur des hexagones sont obtenues par alternance de deux directions hexagonales (cf. figure 5.23)
- Le test de validité d'une position (x, y, θ) est très rapide. Le robot est modélisé par deux cercles, et donc le test de seulement deux points dans la carte de distance est nécessaire pour estimer la distance du robot aux obstacles (cf. figure 5.24). De fait, une fois optimisé, notre algorithme est capable de tester plus d'un million d'hypothèses par seconde pour sa génération de trajectoire (sur un seul cœur).
- En opération normale et pour l'évitement d'obstacles proches, le robot recalcule toutes les secondes les 2 mètres devant lui et recolle cette trajectoire à la trajectoire globale (local path planning).

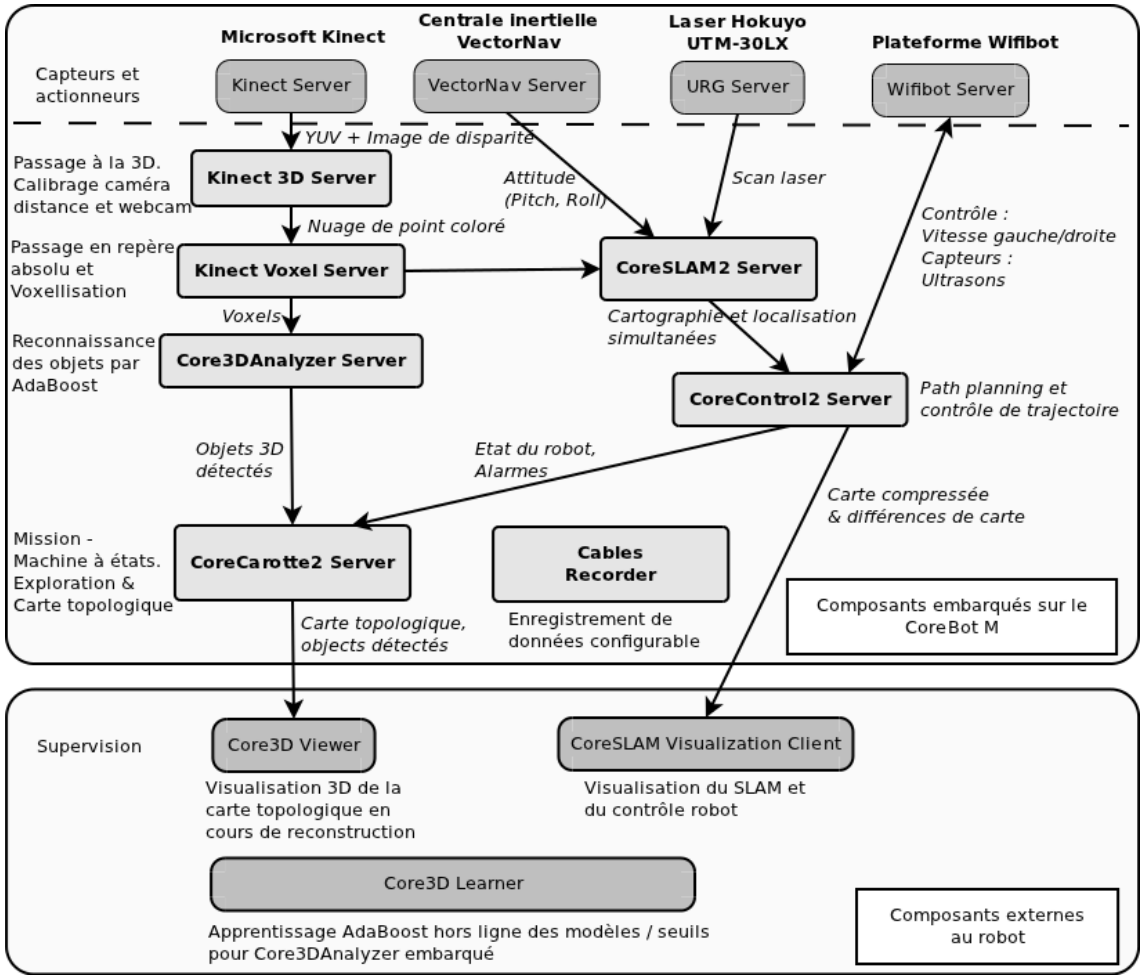


FIGURE 5.20 – *Composants logiciels du Corebot M.*

CoreCarotte2 Le composant *CoreCarotte2* a été conçu de manière à mieux explorer l'ensemble des éléments à identifier pour le défi CAROTTE : les objets à trouver, mais aussi les types de sols et de murs. La stratégie consiste essentiellement à trouver un point d'observation à fort potentiel d'observation (suivant un compromis potentiel d'observation - proximité), de se diriger vers ce point, et d'opérer une rotation autour de ce point pour observer le maximum d'éléments inconnus à partir de ce point. Un plan topologique de cellules 1m x 1m est reconstruit, permettant une identification de haut-niveau des pièces. A noter que le problème des miroirs est résolu au niveau de *CoreCarotte2* par un "locking" : le robot, une fois sorti d'une pièce, verrouille le plan intérieur de cette dernière de manière à ce que cette partie du plan ne puisse être détruite par un mauvais mapping lié à un miroir dans une pièce adjacente. Le robot est ainsi certain de retrouver son chemin vers l'entrée/sortie.

Core3DViewer Le composant Core3DViewer est un outil de visualisation des nuages de points 3D réalisés par le robot. Il permet notamment :

- Une visualisation en temps réel des acquisitions 3D et des objets reconnus au cours de la mission (voir la figure 5.25)

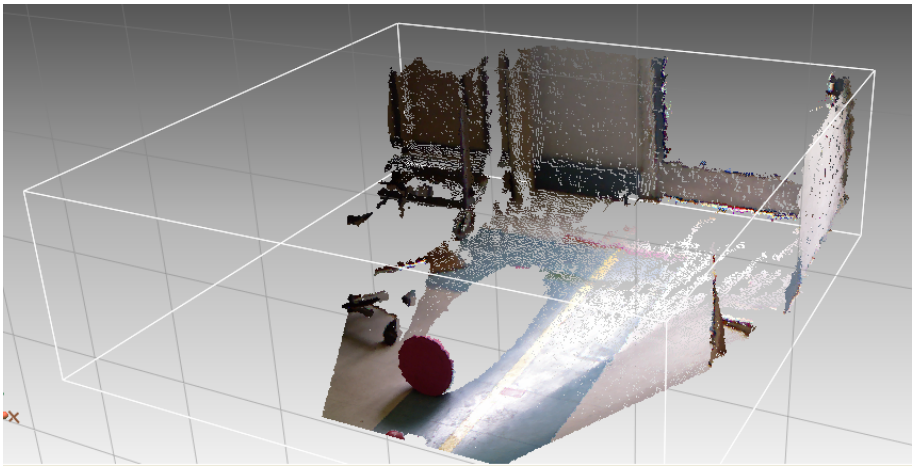


Figure 5.21 – *Nuage de points acquis avec la Kinect embarquée sur le robot.*

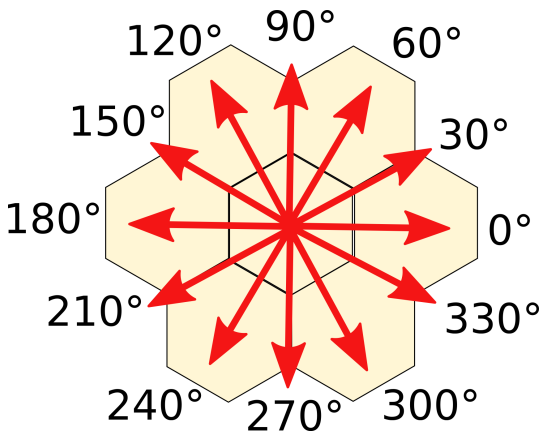


Figure 5.22 – *Les 12 directions discrètes du générateur de trajectoires.*

– Une architecture modulaire et orientée multiplateforme, qui simplifie le portage sur différents systèmes de bureau (Linux, Windows) et systèmes embarqués (Android ou iOS pour iPhone et iPad). On peut ainsi suivre l'évolution d'une mission depuis des postes variés, y compris des systèmes très légers et très mobiles.

5.3 CoreSLAM dans CAROTTE

L'algorithme *CoreSLAM* a été modifié pour le contexte du concours CAROTTE, afin d'intégrer les données de plusieurs capteurs et produire ensuite une carte de l'environnement où évolue le robot ainsi que ses déplacements. La figure 5.26 montre une partie d'une carte produite par l'algorithme. On peut voir les données liées aux différents capteurs représentées dans des couleurs différentes.

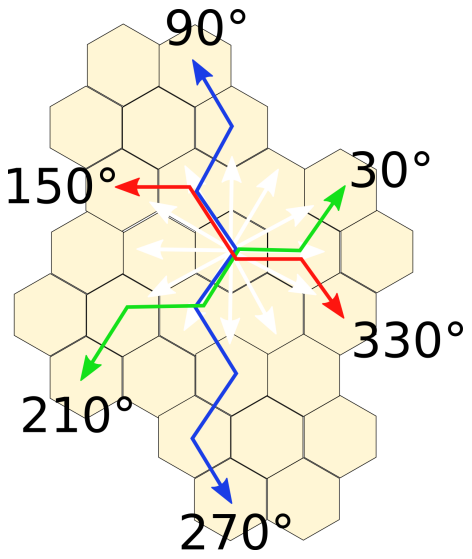


Figure 5.23 – *Utilisation des hexagones pour les directions non alignées sur les hexagones.*

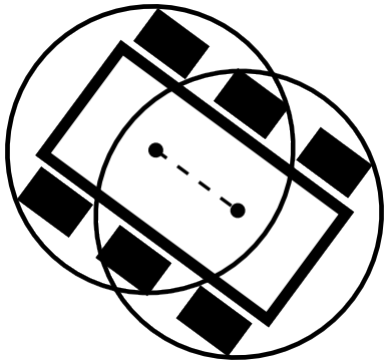


Figure 5.24 – *Modélisation du robot par deux cercles pour le calcul de la distance des obstacles au robot.*

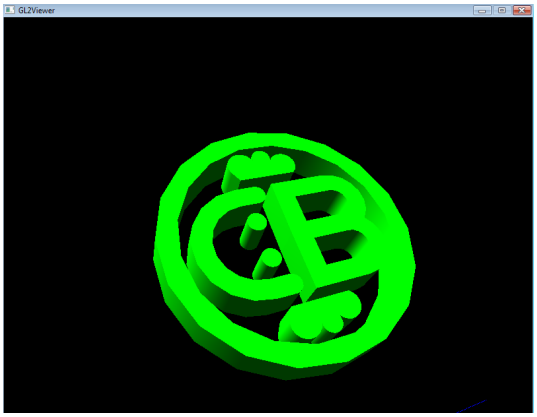


Figure 5.25 – *Visualisation d'un objet 3D modélisé avec un outil tiers.*

Figure 5.27 – *Le CoreBot M en plein évitement des chaises roulantes. Les roulettes ne sont vues que par la Kinect (voir la figure 5.26)*

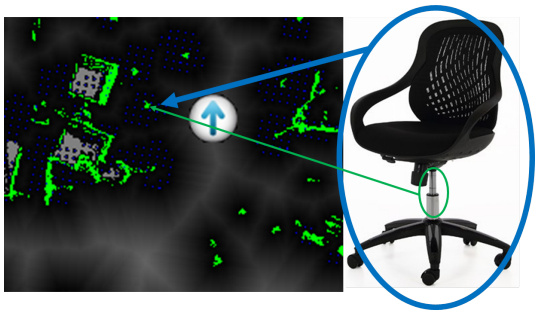


Figure 5.26 – *Exemple de carte produite par* CoreSLAM. *Dans cette carte, on voit que la fusion de capteur permet d'obtenir une meilleure détection de certains objets*

Dans la mesure où *CoreSLAM* est un algorithme léger, on a pu l'exécuter sur la carte embarquée dans le robot avec un minimum de ressources système.

5.3.1 CoreSLAM1

Durant la première année du concours, CoreSLAM intégrait deux capteurs Laser pour la création de la carte. Le premier capteur Laser est un Hokuyo UTM 30LX (voir 3.2), qui offre une portée de 30 m. Ce capteur est placé de manière à scanner l'environnement horizontalement, il permet ainsi à l'algorithme d'effectuer les tâches de localisation et de cartographie.

Le deuxième capteur atteint une portée maximale de 4 m, il s'agit d'un Hokuyo URG 04LX (voir 3.2). Sa position inclinée lui permettait de détecter les obstacles au sol non perçus par le premier laser. La Figure 5.28 montre la disposition de ces éléments sur la plateforme robotique CoreBot 1.

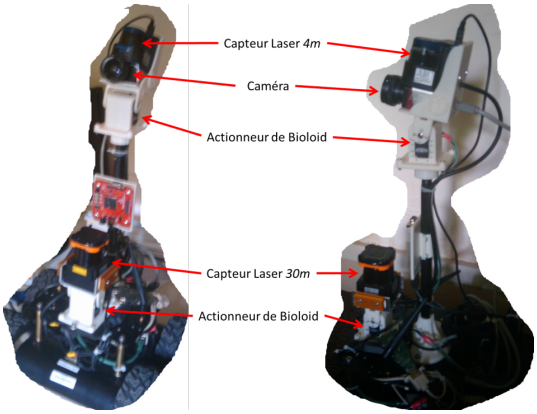


Figure 5.28 – *La disposition de certains capteurs sur le robot*

Le capteur Laser de 4 m peut être remplacé facilement par d'autres types de détecteurs d'obstacles, comme des capteurs à infrarouge ou une caméra RGB-D. Les plateformes des deux capteurs Laser contiennent un servomoteur « Bioloid ».

Figure 5.29 – *Utilisation du bioloid pour réaliser des captures de nuages de point 3D*

Ce montage nous permet ainsi de faire des captures des pièces explorées par le robot en un nuage de points 3D (voir la figure 5.29), exploitable ensuite pour la détection d'objets par exemple.

5.3.2 CoreSLAM2

CoreSLAM2 exploite principalement les données laser fournies par l'Hokuyo UTM 30LX. Ce composant a été intégralement réécrit pour l'édition de 2011 du concours CAROTTE. *CoreSLAM2* a été conçu de manière beaucoup plus évolutive que *CoreSLAM1,*

notamment du fait de son architecture client / serveur. Elle permet de réaliser des clients distants récupérant la carte en temps-réel, par transmission de différences de cartes, ou de cartes complètes compressées en RLE (run-length encoding).

CoreSLAM2 est beaucoup plus précis et performant que son prédécesseur, exploitant notamment mieux les différents cœurs d'un processeur.

Afin de prendre en compte l'évolution du règlement du défi CAROTTE, *CoreSLAM2* exploite les données de la centrale inertielle du robot pour évoluer correctement sur sol non plat (voir la figure 5.30).

Figure 5.30 – *Test de* CoreSLAM2 *sur une rampe.*

5.4 Conclusion

Ce chapitre présente le contexte expérimental des tests effectués afin d'évaluer la qualité de CoreSLAM. L'algorithme CoreSLAM a été testé sur plusieurs plateformes et dans des conditions diverses et variées. Les premiers tests sur le Mines Rover ont permis d'avoir la première version, nommée tinySLAM. Néanmoins, les grandes améliorations de l'algorithme ont pu être testées et validées grâce au défi de robotique CAROTTE. En effet, durant ce concours, nous avons pu avoir accès à des environnements structurés contenant plusieurs objets, et construits à des fins de tests et de validation (cf. Section 6.7 sur la qualité de la cartographie). Nous avons aussi utilisé des données de benchmark téléchargées sur internet, ce qui nous a permis de comparer CoreSLAM à d'autres algorithmes et valider son bon fonctionnement avec différents capteurs et plateformes. Les résultats de ces comparaisons seront présentés dans le chapitre 6 des résultats.

Que la stratégie soit belle est un fait, mais n'oubliez pas de regarder le résultat

Winston Churchill

6

Analyse des résultats

Ce chapitre est consacré à la présentation et l'analyse des différents résultats obtenus durant nos expérimentations. Nous allons d'abord présenter les résultats liés à chacune des améliorations majeures apportées à *tinySLAM/CoreSLAM*. Ensuite, nous allons présenter et analyser quelques résultats de test et de benchmarks.

Sommaire

Figure 6.1 – *Une photo du laboratoire où les expériences ont eu lieu*

6.1 Premiers résultats : *tinySLAM*

Les premières expériences et tests de *tinySLAM* se sont déroulés au laboratoire d'électronique de l'école Mines ParisTech. Cet environnement est vraiment difficile pour l'algorithme (voir la figure 6.1. Cela correspond à la partie supérieure gauche de la carte figurant sur la page 113.). C'est un environnement très encombré avec des caisses et des ordinateurs au sol, beaucoup de tables et d'étagères qui sont très difficiles à détecter et à tracer à l'aide d'un capteur laser. Ces premières expériences ont été réalisées avec le robot Mines Rover et son télémètre Laser à 4 mètres. La portée limitée du laser comparée à la taille du laboratoire d'électronique causait parfois de sérieux problèmes à *tinySLAM*, car l'algorithme se trouve avec des données Laser contenant quelques points d'impact seulement (moins de 20 point sur les 682 théoriques).

Le robot Mines Rover dispose d'une bonne odométrie, grâce notamment à son architecture mécanique en Rocker-Bogie (voir figure 5.2). Ce type d'architecture, permet au robot de garder les roues du milieu en contact permanent avec le sol, même sur un terrain accidenté.

Nous avons ainsi profité de la qualité de cette odométrie afin d'effectuer les premières évaluations de la qualité de la localisation dans *tinySLAM*.

Les figures 6.2, 6.3 et 6.5 affichent des résultats avec le même échantillon de données. Sur les deux premières figures, on voit la comparaison entre l'odométrie et l'estimation de mouvement par le laser. La figure 6.2 montre les résultats des mesures de la vitesse du robot par les odomètres et par *tinySLAM* (en utilisant le Laser seulement). La similitude quasi parfaite entre les deux courbes nous donne une première idée sur la qualité de localisation de *tinySLAM*.

Sur la figure 6.3, on présente une autre comparaison entre les mesures par odométrie et les mesures par *tinySLAM*. L'élément de comparaison étant la vitesse angulaire. Nous pouvons faire la même remarque que pour la première comparaison, sauf pour une petite

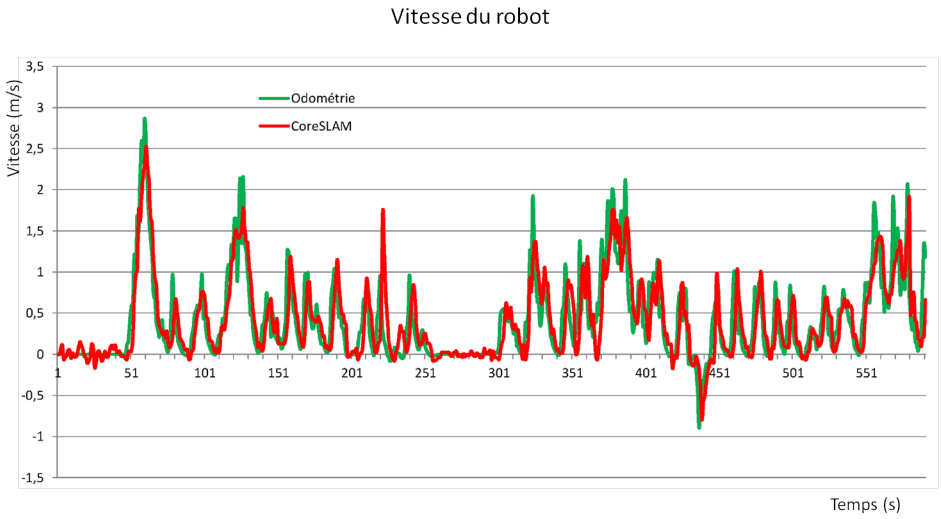


FIGURE 6.2 – *Vitesse mesurée avec odométrie (courbe verte) et avec* tinySLAM *Laser (courbe rouge)*

période (notée par l'ellipse bleue), où le robot a eu des problèmes d'odométrie (le robot a touché une chaise, ce qui a soulevé les roues de la partie droite, voir figure 6.4).

La carte de la figure 6.5 a été construite en combinant les données du laser et de l'odométrie, cette figure montre une bonne précision de la reconstruction, on voit que la boucle est presque fermée. Sur cette figure, nous avons représenté en rouge, les "trous" de la carte construite par le robot. En gris, nous avons superposé tous les scans pris par le laser. En bleu, la reconstitution de la trajectoire du robot.

On peut faire une remarque intéressante ici, les dérapages observés dans cette expérience (voir la figure 6.7) ont été très bien corrigés par les données du laser. Notez que la vitesse du robot dans nos expériences a atteint 2,5 m/s et la vitesse de rotation 150 °/s (mesurée par odométrie, et confirmé par la mesure du laser). Cette grande vitesse angulaire du robot peut être atteinte grâce aux quatre roues directrices.

6.2 Localisation parallèle

L'algorithme *CoreSLAM* est devenu de plus en plus rapide au fur et à mesure des améliorations. Nous avons ainsi profité de ce gain en vitesse de calcul pour améliorer la qualité de la localisation. Rappelons que le processus de localisation parallèle consiste à lancer plusieurs opérations d'estimation de la position du robot simultanément, avec des paramètres différents. Ceci permet d'améliorer la localisation dans certains cas particuliers, car l'algorithme garde la meilleure estimation parmi toutes celles lancées.

La figure 6.8 présente un exemple de résultats obtenu en activant la localisation parallèle. Les trois courbes présentent le meilleur score obtenu pour chaque étape de localisation.

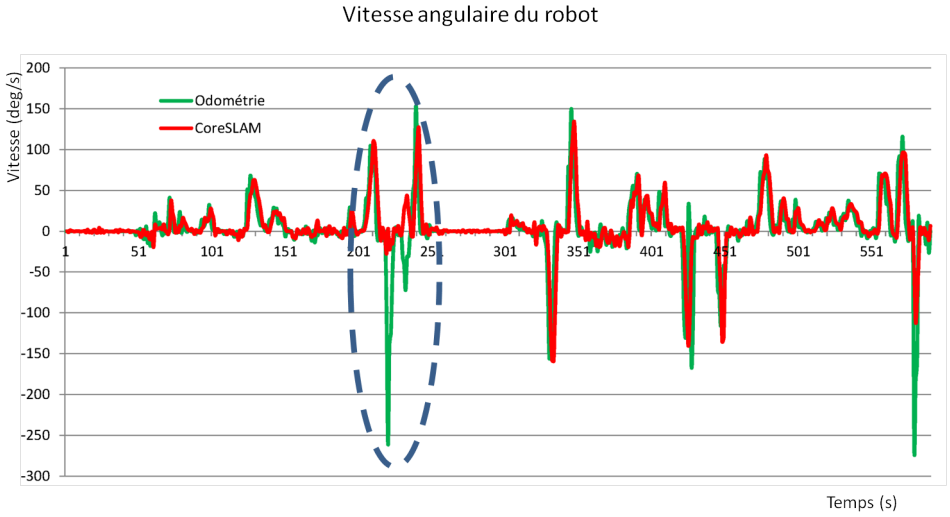


Figure 6.3 – *Vitesse angulaire mesurée avec odométrie (courbe verte) et avec* tinySLAM *Laser (courbe rouge)*

Figure 6.4 – *Le robot touche une chaise, ce qui soulève une partie des roues, en particulier celles de l'odométrie, causant une erreur au niveau des estimations de vitesse angulaire*

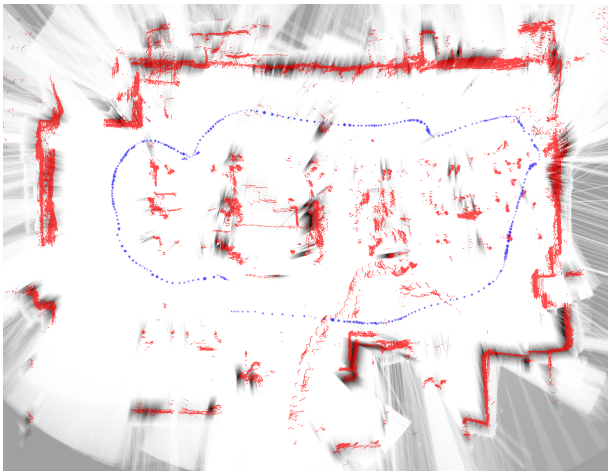


Figure 6.5 – *Une carte de laboratoire obtenue au cours de nos expériences.*

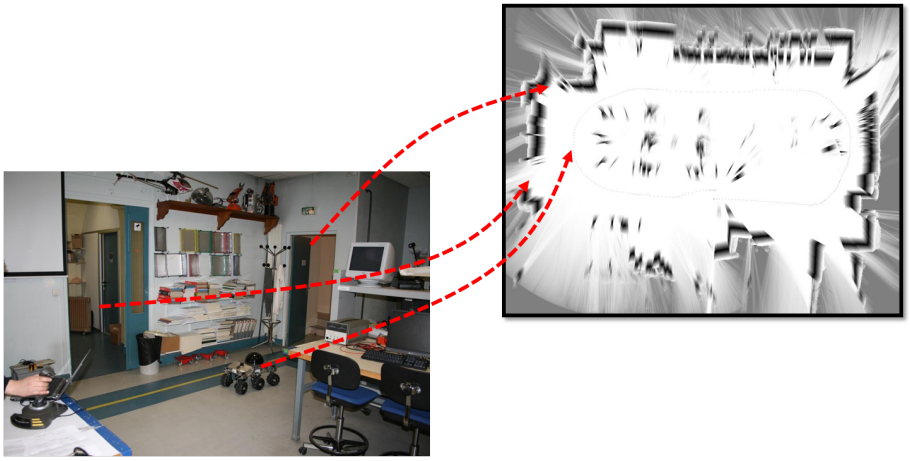


Figure 6.6 – *Comparaison entre les éléments de l'environnement et leur représentation dans la carte de distances*

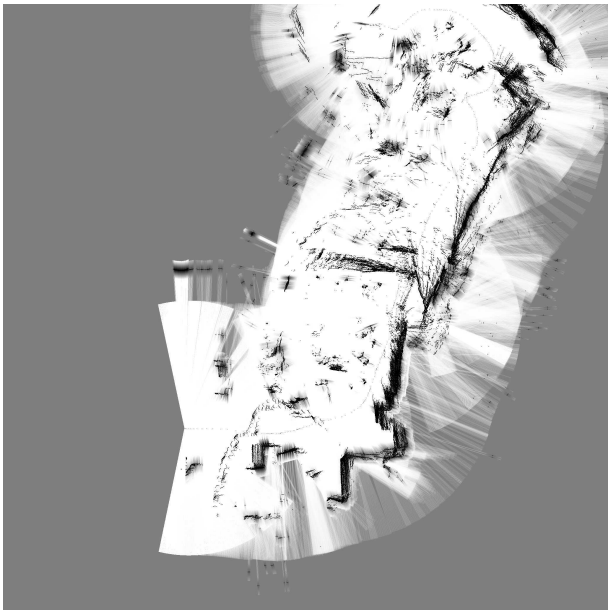


Figure 6.7 – *La carte construite en utilisant l'odométrie seulement.*

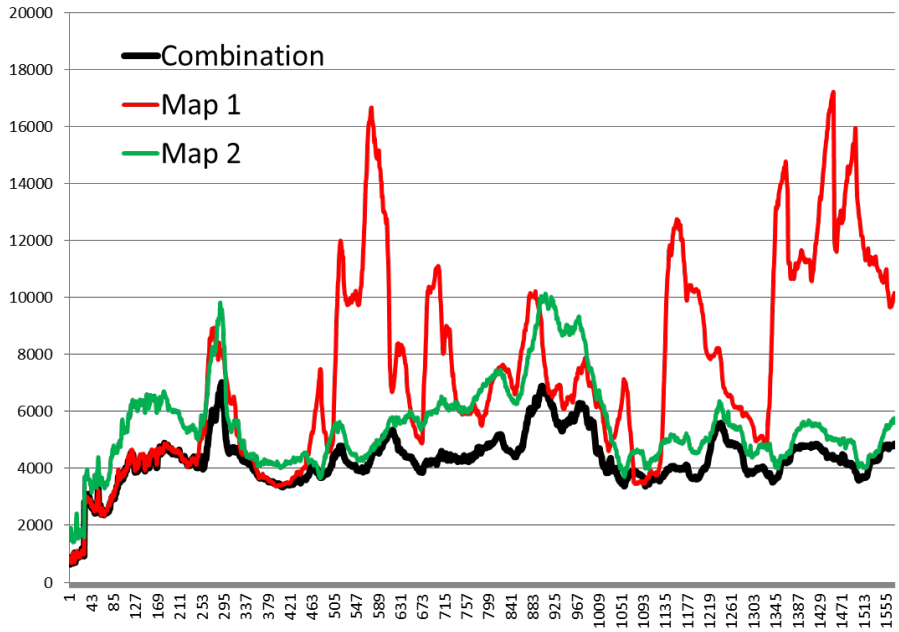


Figure 6.8 – *La combinaison des processus de localisation permet de garder le meilleur score dans différentes situations et environnements*

Nous avons lancé deux processus d'estimation simultanés, en modifiant le paramètre lié à la taille du trou dans la fonction de création de la carte. Nous avons choisi ce paramètre suite à plusieurs expériences qui ont montré son impact important sur la qualité de la localisation.

- La courbe rouge correspond aux meilleurs scores de localisation pour la première estimation.
- La courbe verte correspond aux meilleurs scores pour la deuxième estimation.
- La courbe noire est le résultat de la combinaison des deux.

En fonction des situations, la qualité des deux estimations varie fortement. Mais la combinaison garde toujours un meilleur résultat, ce qui donne forcément une meilleure localisation.

6.3 Algorithmes de recherche

Afin d'améliorer la vitesse de convergence de l'algorithme d'estimation de position, nous avons implémenté et testé différents algorithmes d'optimisation. L'algorithme *tinySLAM* se base sur une méthode de Monte Carlo. Durant le passage vers *CoreSLAM*, nous avons implémenté un algorithme génétique. Nous avons ainsi procéder à quelques tests de comparaison entre les deux algorithmes.

Afin d'effectuer ces tests, nous avons intégré les deux algorithmes dans la partie de localisation de l'algorithme *tinySLAM*. Comme nous pouvons le constater sur la figure 6.9, nous avons utilisé une partie de la carte de l'environnement à laquelle on applique une translation et une rotation, puis nous avons demandé à l'algorithme de retrouver la

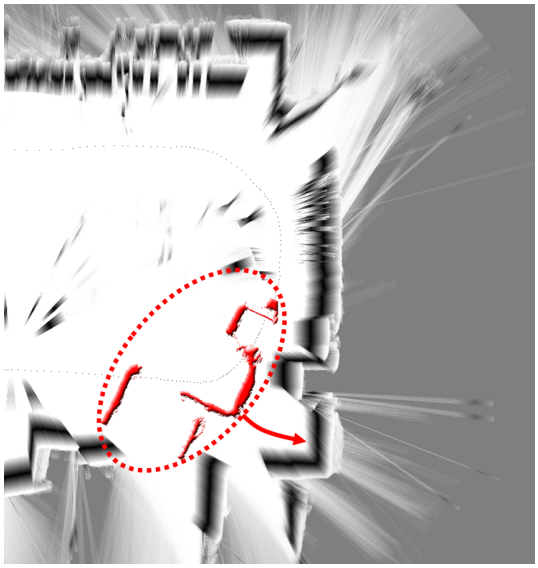


Figure 6.9 – *Nous décalons une partie de la carte et nous demandons à l'algorithme de localisation de retrouver la position d'origine.*

position d'origine de la carte, ce qui est équivalent à la position du robot lorsque le laser a fourni les données associés à la partie déplacée.

La figure 6.10 représente le score calculé pendant une seule opération d'estimation de position. L'algorithme d'optimisation (génétique ou Monte Carlo) tente de minimiser le score afin de trouver la meilleure hypothèse de position.

On remarque que l'algorithme génétique atteint le minimum souhaité dans un nombre d'itérations moins élevé que l'algorithme de Monte Carlo. Ce résultat a été remarqué pour plusieurs tests et plusieurs expériences différentes. Nous pouvons ainsi diminuer le nombre d'itérations nécessaire pour la localisation, lorsqu'on utilise un algorithme génétique, tout en gardant une bonne estimation de la position du robot. Cette diminution offre un gain considérable en vitesse d'exécution de l'algorithme de SLAM.

6.4 Streaming SIMD Extensions

Dans le même esprit d'optimisation de l'algorithme, nous avons cherché à profiter des possibilités offertes par les processeurs Intel.

La figure 6.11 montre une comparaison entre les résultats de conversion d'un flottant en entier, avec trois méthodes différentes.

Ce graphe montre clairement l'erreur qu'on peut avoir en utilisant l'opération de troncature. La fonction *floor()* et la directive *_mm_cvtps_pi32()* donnent les mêmes résultats, sauf pour les cas autour de v.5. [1].

Les opérations de conversion consomment beaucoup de ressources systèmes, dans le

1. Ceci est probablement dû à une incompatibilité entre l'implémentation de l'instruction _mm_cvtps_pi32 et les standards IEEE 754

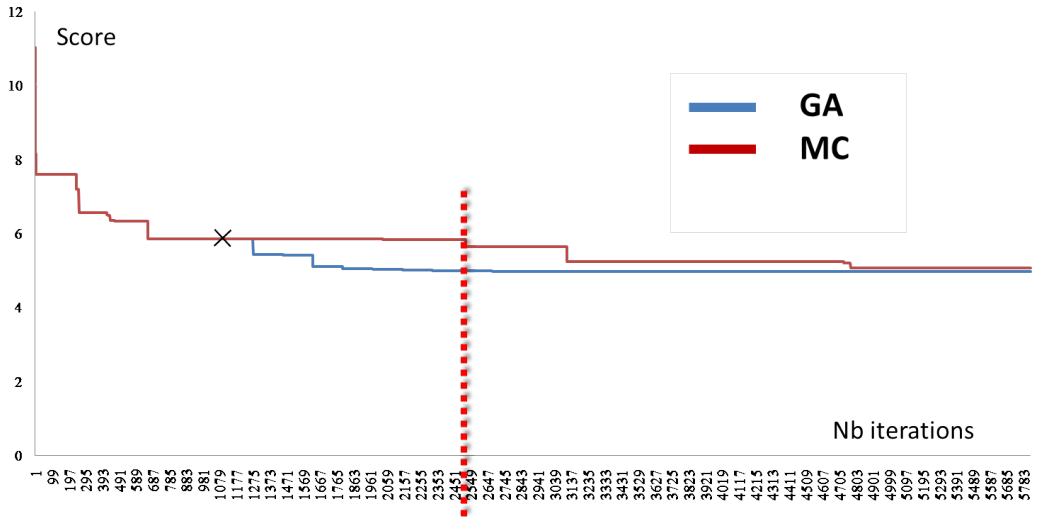


Figure 6.10 – *Le score obtenu en utilisant un algorithme génétique et un algorithme de Monte Carlo. L'algorithme génétique obtient le score minimum plus rapidement*

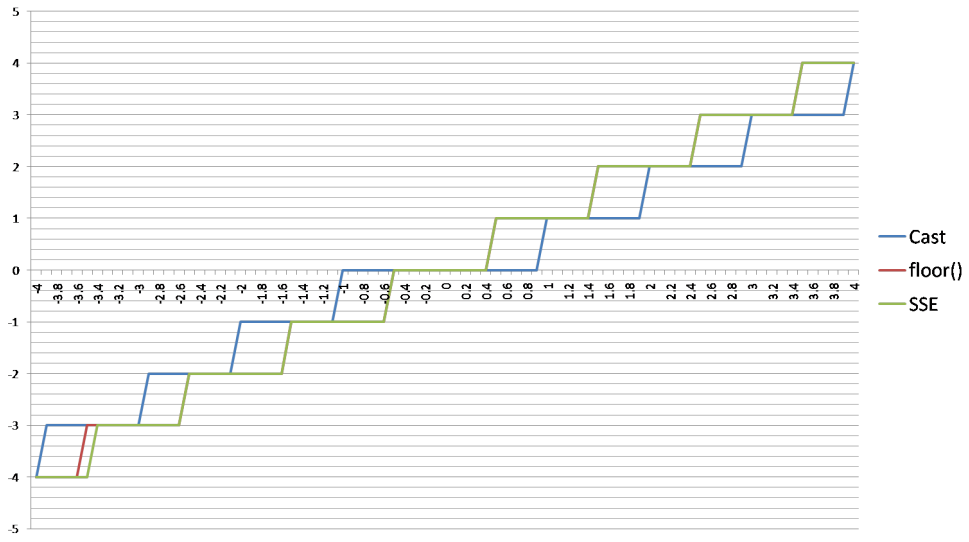


Figure 6.11 – *Comparaison entre les résultats de conversion en utilisant trois méthodes différentes*

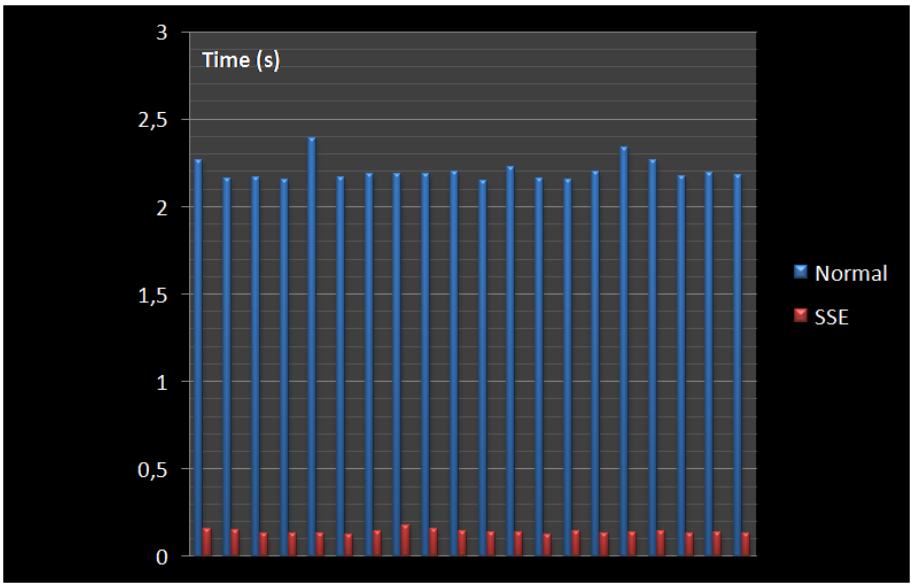


Figure 6.12 – *Comparaison entre les durées de traitement pour l'opération de conversion en utilisant deux méthodes*

mesure où elles sont exécutées plusieurs fois dans un processus de mise en correspondance de données Laser dans un algorithme de SLAM. Dans notre programme de test, cette opération peut être exécutée 4 millions de fois pour chaque itération (682 données laser et 6000 hypothèses pour cette expérience).

La figure 6.12 montre une comparaison du temps nécessaire pour faire des opérations de conversion de flottant en entier. La méthode dite normale est basée sur la fonction *floor()* et la méthode SSE utilise l'instruction *_mm_cvtps_pi32()*. Le programme avait pour tâche d'exécuter 100 tests de 50 millions de conversions chacun. En moyenne, la méthode basée sur SSE et 16 fois plus rapide que la méthode normale.

Nous avons ensuite mesuré l'amélioration de la vitesse globale de l'algorithme en utilisant SSE. La figure 6.13 montre des exemples de résultats trouvés. Durant les tests, nous avons alourdit la tâche de localisation, en cherchant dans un espace d'hypothèses plus grand, et en fixant le nombre maximal d'itérations à une valeur très élevée. Ceci nous garantissait l'obtention de la meilleure hypothèse possible, pour chacune des deux méthodes testées.

Les expériences montrent que notre méthode basée sur SSE est 4 fois plus rapide que les méthodes conventionnelles en moyenne.

6.5 Filtrage

La figure 6.14 montre deux cartes d'une des arènes du concours CAROTTE. La carte de gauche a été construite sans filtrage ni latence, contrairement à la carte de droite, où on les utilise. La différence au niveau de la qualité de la carte est clairement visible. Rappelons que la latence et le filtrage permettent d'éviter la mise à jour de la carte lorsque les résultats de la phase de localisation ne sont pas satisfaisants.

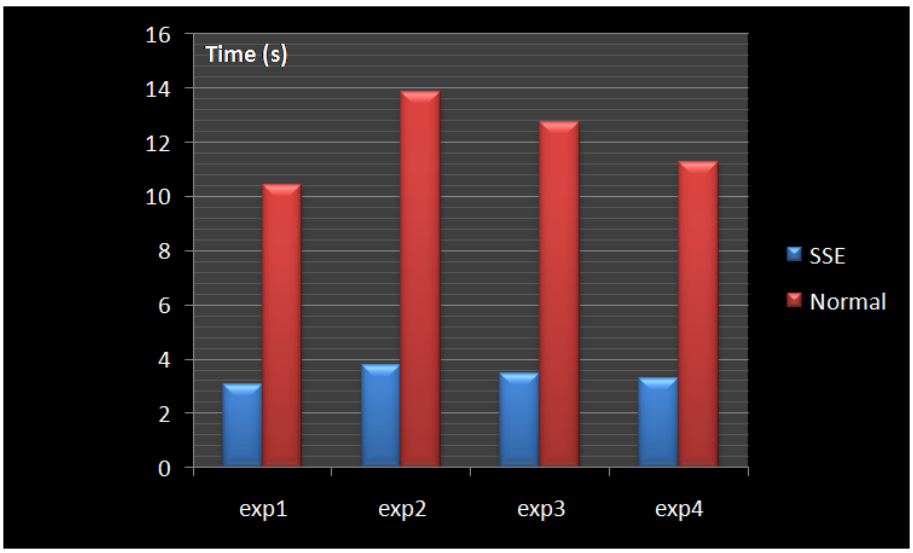


Figure 6.13 – *Comparaison entre les durées de la phase de localisation. La méthode basée sur SSE permet de réduire le temps nécessaire pour cette tâche critique*

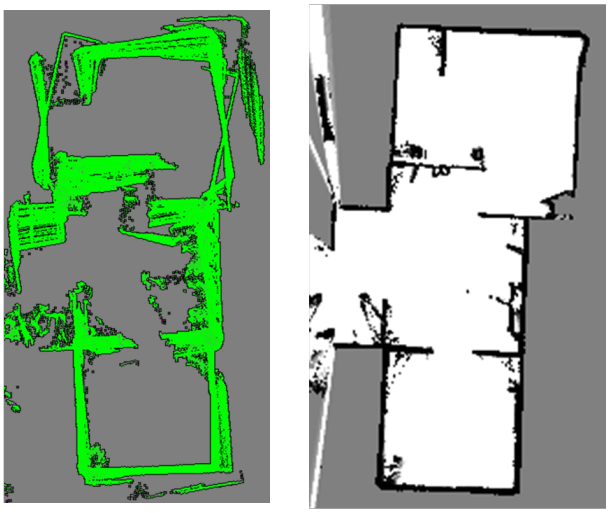


Figure 6.14 – *Lorsqu'on utilise le filtrage, la carte est plus précise et l'algorithme de SLAM évite quelques problèmes de localisation*

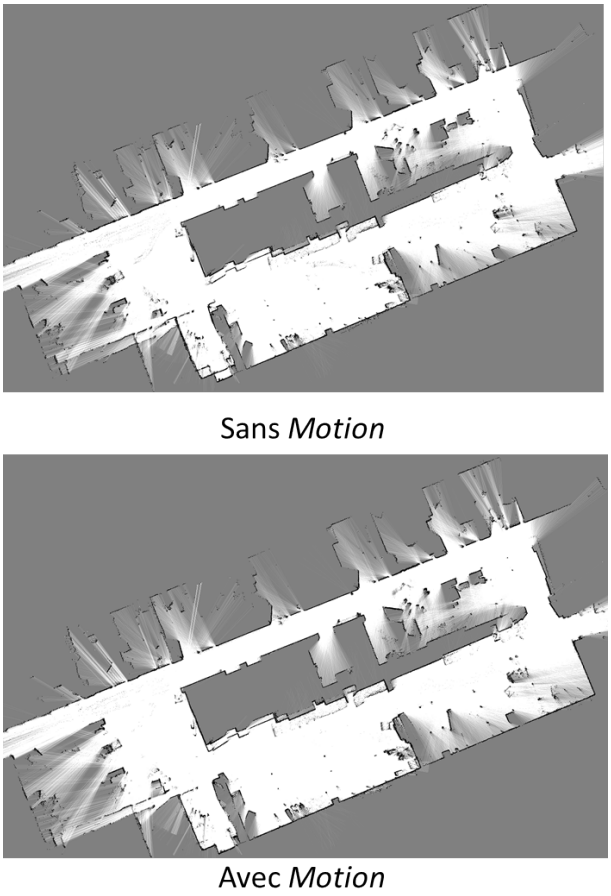


Figure 6.15 – *L'effet de l'utilisation du module* Motion *n'est pas clairement visible sur les cartes*

En filtrant la position estimée lors de l'étape de localisation, l'algorithme de SLAM a pu éviter les mauvaises estimations, et ainsi construire une carte beaucoup plus correcte..

6.6 Le module *Motion*

Afin d'analyser l'effet de l'utilisation du module *Motion* dans le SLAM, nous avons créé deux cartes en utilisant le même paramétrage, et le même ensemble de données. La figure 6.15 montre le résultat global de l'exécution de l'algorithme *CoreSLAM*. Il est difficile de détecter la différence entre les deux cartes visuellement. Nous avons ainsi agrandit une partie de la carte et nous avons mesuré l'erreur de la cartographie.

La figure 6.16 présente le résultat des mesures de l'erreur de cartographie. L'utilisation du module *Motion* permet à l'algorithme de SLAM d'améliorer sa localisation, et ainsi diminuer la dérive naturelle de la méthode IML utilisée dans *CoreSLAM*.

Le module *Motion* est simple et rapide. Il ne contient aucune forme de filtrage. Il a été facilement intégré à *CoreSLAM* et permet d'améliorer la qualité des résultats et nous offre un pas de plus vers notre objectif de diminuer la dérive de l'IML au maximum.

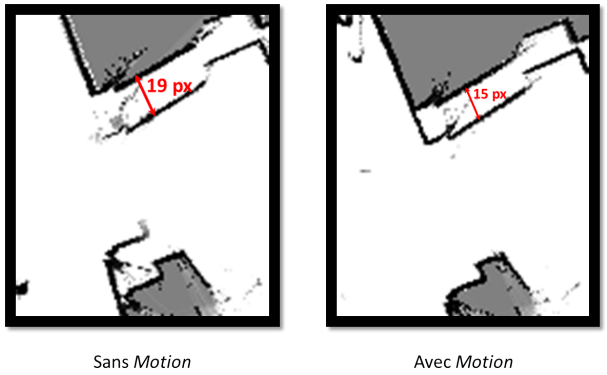


Figure 6.16 – *En agrandissant certaines parties de la carte, on peut voir l'amélioration apportée par le module* Motion

Figure 6.17 – *Exemple d'une arène explorée lors du concours CAROTTE. Deux mesures sont indiquées*

6.7 Qualité de la cartographie

Pour évaluer la qualité de la cartographie dans *CoreSLAM*, nous avons utilisé les données enregistrées lors de notre participation au concours CAROTTE. Durant ce concours, les robots participants doivent effectuer des missions d'exploration autonome d'environnements inconnus. Les arènes d'évolution des robots sont construites à base de cloisons de 1 mètre. La figure 6.17 montre un exemple d'arène.

La taille fixe des cloisons nous offre une vérité terrain qui nous a aidés à évaluer la qualité des cartes produites avec *CoreSLAM*. La figure 6.18 montre l'idée de base de cette évaluation. Dans cette carte, chaque pixel représente 1 centimètre dans la réalité. Pour les deux distances marquées, on a une erreur de $abs(3*100-303)+abs(5*100-498)=5\,cm$ pour $800\,cm$.

En utilisant cette méthode pour un ensemble de 20 enregistrements de données, nous

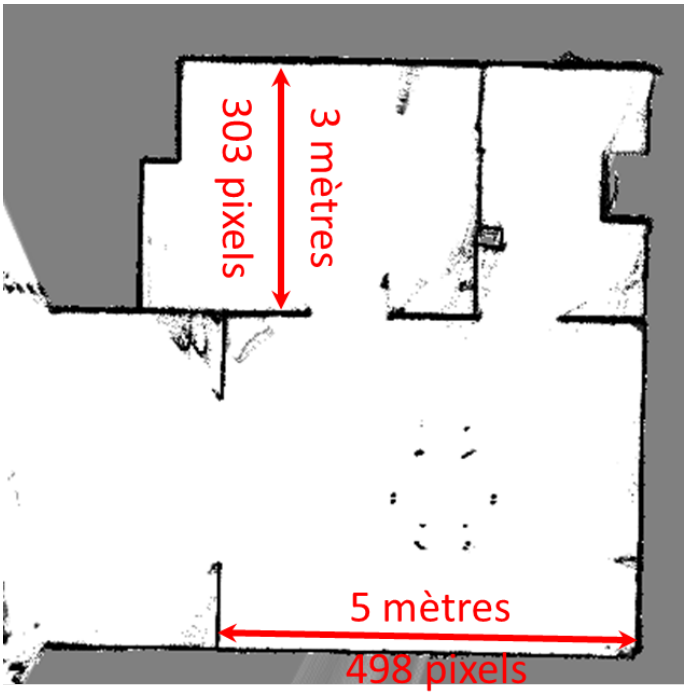


Figure 6.18 – *Mesure de la qualité de la cartographie dans CoreSLAM en utilisant la carte d'une arène du concours CAROTTE*

avons obtenu une erreur moyenne de $1,15\%$.

Pour évaluer la qualité de la cartographie visuellement, nous avons utilisé des données disponibles en téléchargement sur internet. Il s'agit d'un enregistrement fait du bâtiment de Willow Garage,

La figure 6.19 présente une comparaison entre la carte produite par notre algorithme et celle du logiciel GMapping. La partie agrandie permet de voir une erreur de cartographie du logiciel GMapping, alors que *CoreSLAM* fournit une meilleure carte.

Nous avons utilisé le même ensemble de données afin de comparer *CoreSLAM* au logiciel commercial Karto. Visuellement, les deux cartes sont semblables. On note ici que le logiciel Karto utilise l'odométrie, ce qui n'est pas le cas de *CoreSLAM*. L'odométrie permet d'éviter certains problèmes de localisation qui peuvent être causés par le Laser (comme le couloir de la partie agrandie).

6.8 Qualité de la localisation

Afin d'évaluer la qualité de la localisation du SLAM, nous avons utilisé la méthode expliquée dans l'article [4]. Cette méthode se base sur le calcul de l'erreur de localisation pour chaque déplacement. On utilise uniquement les relations relatives entre les positions du robot. L'équation permettant le calcul de l'erreur se présente ainsi :

$$\varepsilon(\delta) = \sum_{ij} (\delta_{ij} \ominus \delta^*_{ij})^2$$

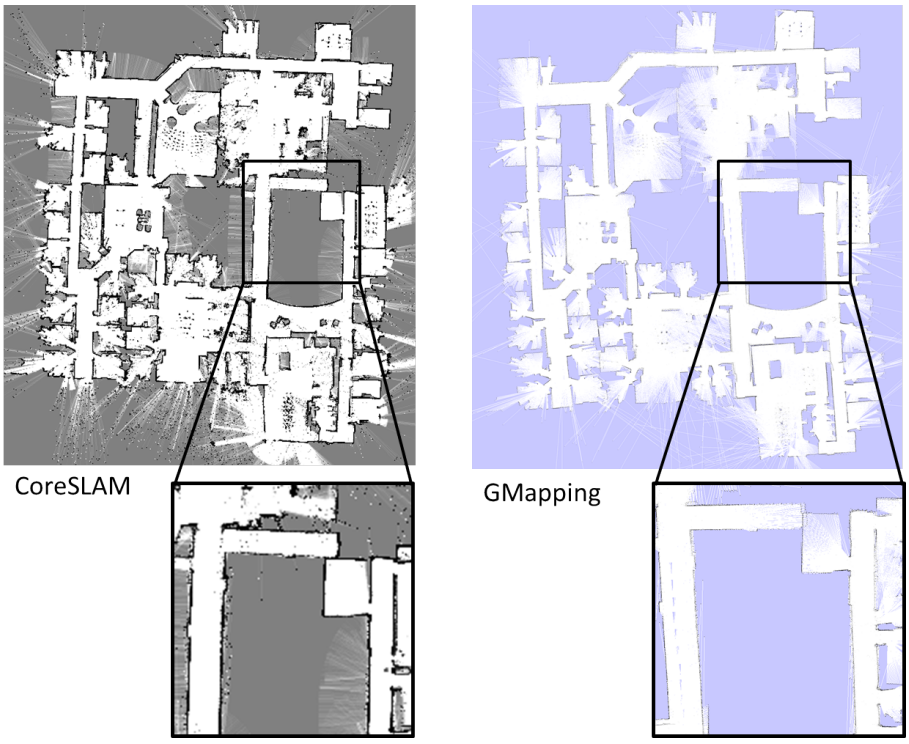


Figure 6.19 – *Comparaison entre* CoreSLAM *et l'algorithme GMapping*

Figure 6.20 – *Comparaison entre* CoreSLAM *et le logiciel Karto*

Fichier de relations

timestamp1 timestamp2 x y z roll pitch yaw

Fichier de SLAM

FLASER num_readings [range_readings] x y theta odom_x odom_y odom_theta timestamp hostname logger_timestamp

Figure 6.21 – *Le fichier des relations et le fichier de SLAM*

	Mean	Std	Min	Max
a	**0.526167**	0.469187	0.000040	7.182943

	Mean	Std	Min	Max
b	**0.938020**	1.867734	0.000061	10.991265

	Mean	Std	Min	Max
c	**4.292477**	12.737558	0.004069	54.791952

Table 6.1 – *Les résultats d'évaluation des trois versions de la carte du MIT CSAIL*

où δ_{ij} est le déplacement relatif entre deux positions et δ^*_{ij} est la réalité terrain correspondante. Les relations représentant la vérité terrain peuvent être obtenue en utilisant des scans laser alignés manuellement (du moment où les humains connaissent la structure de l'environnement) ou par d'autres types de capteurs (GPS pour les environnements à l'extérieur par exemple).

L'algorithme prend en entrée deux fichiers. Le premier est un fichier de relations. Il est construit manuellement et contient la vérité terrain. Le deuxième fichier est un fichier produit par l'algorithme de SLAM. Il contient les positions du robot au format Carmen [80]. La figure 6.21 montre la structure des deux fichiers.

Afin de vérifier la validité des évaluations de cette méthode, nous avons lancé l'algorithme *CoreSLAM* sur un ensemble de données (le bâtiment MIT CSAIL) en modifiant quelques paramètres, pour avoir des cartes différentes. La figure 6.22 montre les résultats obtenus. La carte (a) correspond à un fonctionnement normal de *CoreSLAM*. Pour obtenir la carte (b), **nous avons limité la valeur maximale des données laser**, ce qui fournit de fausses données, et ainsi une fausse carte. Une modification moins importante permet d'avoir la carte (c).

Nous avons ensuite calculé l'erreur de localisation en utilisant la méthode décrite plus haut. Le tableau 6.1 présente les résultats. La colonne *Mean* correspond à l'erreur moyenne de localisation. On peut clairement voir que cette erreur augmente à chaque fois qu'on fournit une mauvaise carte à l'algorithme d'évaluation. La méthode choisie pour évaluer *CoreSLAM* fournit ainsi de bons résultats.

Nous avons appliqué cette méthode d'évaluation sur l'ensemble de données du bâtiment

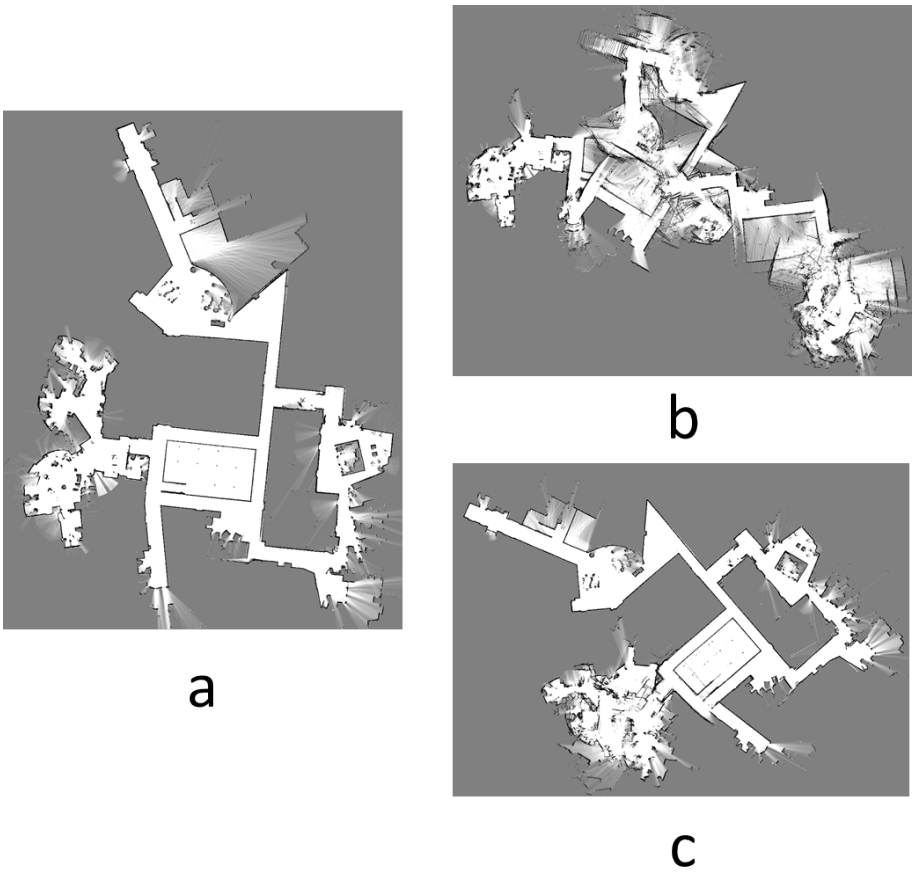


Figure 6.22 – *La carte du bâtiment MIT CSAIL construite par* CoreSLAM *avec trois configurations différentes*

Figure 6.23 – *Carte* CoreSLAM *du bâtiment 079 de l'université de Freiburg*

	Scan Matching [*]	RBPF [*]	Graph Mapping [*]	**CoreSLAM**
Absolute	0.258 (std 0.427)	0.061 (std 0.044)	0.056 (std 0.042)	**0.040 (std 0.028)**
Squared	0.249 (std 0.687)	0.006 (std 0.020)	0.005 (std 0.011)	**0.002 (std 0.003)**

Table 6.2 – *Comparaison des erreurs de localisation pour la carte du bâtiment 079 de l'université de Freiburg (Les résultats marqués d'un * sont extraits de [4])*

079 de l'université de Freiburg (figure 6.23) et du bâtiment MIT CSAIL (figure 6.24).

La qualité de la localisation a ensuite été évaluée et comparée à la localisation par 3 algorithmes : Scan Matching, RBPF et Graph Mapping.

- Scan Matching [81] : estimation de la position du robot par mise en correspondance des données du laser entre chaque deux scans consécutifs
- RBPF [82] : utilisation d'un filtre particulaire Rao-Blackwellisé. L'implémentation utilisée est disponible sur internet [83].
- Graph Mapping [84] : estimation de la position par optimisation de graphes.

Les résultats obtenus pour le bâtiment 079 sont représentés dans le tableau 6.2. L'algorithme *CoreSLAM* offre de bons résultats, comparé aux autres algorithmes.

Pour les données du MIT CSAIL, le Graph Mapping donne de très bons résultats. *CoreSLAM* reste comparable au RBPF (voir tableau 6.3). Nous signalons ici que nous n'avons pas pu vérifier les résultats du Graph Mapping, nous avons utilisés ceux inclus dans l'article [4].

	Scan Matching [*]	RBPF [*]	Graph Mapping [*]	**CoreSLAM**
Absolute	0.106 (std 0.325)	0.049 (std 0.049)	0.004 (std 0.009)	**0.040 (std 0.032)**
Squared	0.117 (std 0.728)	0.005 (std 0.013)	0.0001 (std 0.0005)	**0.002 (std 0.008)**

Table 6.3 – *Comparaison des erreurs de localisation pour la carte du bâtiment MIT CSAIL (Les résultats marqués d'un * sont extraits de [4])*

Figure 6.24 – *Carte* CoreSLAM *du bâtiment MIT CSAIL*

6.9 Conclusion

Nous avons présenté et analysé dans ce chapitre les différents résultats obtenus lors des tests expérimentaux de *CoreSLAM*. Nous avons commencé par les résultats de *tinySLAM*, la première version de l'algorithme, codée en moins de 200 lignes de code en langage C. Pour cette version, notre principal objectif était d'avoir un résultat de localisation correcte. Nous avons ainsi profité de l'excellente odométrie du robot Mines Rover afin d'évaluer la qualité de la localisation fourni par *tinySLAM*. Nous présentons ensuite les résultats d'améliorations obtenues en utilisant le SLAM parallèle. Les tests ont été effectués pour deux processus de localisation simultanés, et on peut clairement voir l'impact positif de la localisation parallèle au niveau de l'amélioration de la qualité. Nous passons ensuite aux différents résultats liés aux algorithmes de recherche et d'optimisation présentés dans la section 4.1. Dans la même logique des améliorations de la vitesse d'exécution de l'algorithme, nous présentons ensuite les résultats de l'implémentation du SSE dans les étapes critiques de l'algorithme. Les sections qui suivent concernent les améliorations faites pour la qualité des résultats, et non la vitesse d'exécution. En effet, nous discutons des effets observés à l'ajout du filtrage et l'intégration de la détection des objets mobiles. Les deux dernières sections présentent un résumé des résultats d'évaluation de la qualité globale de l'algorithme *CoreSLAM*. Nous avons évalué la qualité de sa cartographie en utilisant les caractéristiques des arènes structurées du concours CAROTTE. Pour évaluer la qualité de la localisation, nous avons utilisé la méthode décrite dans [4], avec des données de benchmark téléchargées sur Internet.

7

Conclusions

Nous résumons dans ce chapitre les principaux éléments de cette thèse. Après un rappel du contenu des différentes parties de ce manuscrit et une analyse des principales contributions apportées, nous présentons quelques perspectives d'amélioration du travail effectué durant cette thèse. Ces perspectives d'améliorations visent à compléter ce travail et fournir de meilleurs résultats.

Sommaire

7.1 Conclusions générales

Les tâches de localisation et de cartographie, regroupées sous le terme de SLAM, constituent un pilier fondamental du fonctionnement autonome d'un véhicule mobile robotisé. Bien que plusieurs algorithmes aient été développés dans le but de résoudre le problème du SLAM, ils nécessitent généralement de grandes ressources système pour atteindre ce but correctement et efficacement. Pourtant, un système embarqué dispose habituellement de ressources limitées, en matière d'espace de stockage, d'énergie et de puissance de calcul. Ceci impose de fortes contraintes aux algorithmes. Dans ce contexte, nous avons travaillé durant cette thèse sur le développement et l'amélioration d'un algorithme de SLAM simple, rapide, léger et efficace : *CoreSLAM*. Notre approche est basée sur l'utilisation de la maximisation de vraisemblance incrémentale. Ce type d'approches a largement été délaissé à cause de la divergence naturelle qui le caractérise. Notre principal objectif été de minimiser au maximum cette divergence, tout en gardant les avantages de l'IML. L'idée de base est donc de limiter la dérive incluse naturellement dans les méthodes d'IML. Nous avons pu atteindre notre objectif en introduisant plusieurs idées innovantes au fonctionnement de base de *CoreSLAM*, tout en gardant sa simplicité, sa vitesse d'exécution et sa légèreté. Nous avons commencé par une présentation générale du problème de SLAM. Ce qui permet de comprendre certaines bases du contexte de notre travail de thèse. Après quelques rappels nécessaires à la bonne compréhension de la problématique, nous avons discuté de l'état de l'art du filtrage de Kalman et du filtrage particulaire, dans la mesure où ces deux méthodes constituent la base de la majorité des algorithmes de SLAM laser actuels. Nous avons aussi présenté les problèmes de ces deux techniques, et les raisons pour lesquelles nous avons choisi une implémentation incrémentale de notre algorithme de SLAM. Le troisième chapitre est consacré à la présentation des idées fondamentales constituant la base de notre approche. Après une brève introduction des capteurs ayant servi aux expérimentations, nous avons expliqué la manière dont nous avons implémenté un processus de SLAM incrémental basé sur l'utilisation de l'IML. Nous avons ensuite détaillé les deux grandes étapes de l'algorithme :

- La localisation : *CoreSLAM* utilise la mise en correspondance des données de scans lasers pour estimer la position du robot. Notre méthode cherche un couple (translation, rotation) tel que les données de la dernière acquisition du capteur Laser correspondent au mieux à la carte courante. Cette recherche passe par un algorithme d'optimisation qui teste plusieurs hypothèses et nous en fournis la meilleure (section 3.4 sur l'estimation de la position). L'algorithme utilise une carte de SLAM particulière, construite de manière à garantir une localisation la plus rapide et la plus efficace possible (section 3.5 sur la mise en correspondance).
- La cartographie : la carte de *CoreSLAM* est mise à jour en suivant plusieurs règles (filtrage, latence ...). Cette méthode permet de s'assurer de la qualité des résultats de l'étape de localisation avant de modifier la carte de l'environnement (section 3.6 sur la mise à jour de la carte).

Nous concluons ce chapitre par le module Motion, permettant à l'algorithme *CoreSLAM* de détecter et suivre les éléments mobile de la carte. Ce module, simple et efficace, améliore la qualité des résultats de *CoreSLAM*, dans la mesure où les objets détectés mobiles sont exclus de la carte de SLAM, ce qui offre plus de robustesse à la localisation.

Après avoir présenté le squelette de notre algorithme de SLAM, nous détaillons dans le chapitre 4 les techniques d'optimisation utilisées afin de limiter la dérive de l'IML est obtenir les meilleurs résultats de SLAM.

Nous avons commencé par les algorithmes de recherche testés. Ces algorithmes interviennent au cœur de l'étape de localisation, et génèrent l'ensemble des hypothèses concernant la position du robot. La manière dont un algorithme choisit les hypothèses à générer est importante pour la qualité et la vitesse d'exécution de la localisation. Nous présentons dans cette section trois algorithmes :

- Monte Carlo : nous avons implémenté une version modifiée de l'algorithme de Monte Carlo, dans le but d'éviter les minimas locaux.
- Algorithme Génétique : l'algorithme génétique effectue une optimisation inspirée des processus de sélection naturelle.
- Algorithme multi-échelle : cet algorithme innovant génère des réseaux cubiques d'hypothèses, qu'il raffine ensuite en fonction des scores obtenus.

Afin d'accélérer les traitements, nous avons ajouté une file de priorité à *CoreSLAM*. Cette file définit les hypothèses ayant une grande probabilité d'être choisies comme meilleure estimation de position. Ceci permet de restreindre les évaluations de scores à ces hypothèses et ainsi réduire la charge de calcul nécessaire.

La section 4.3 de ce chapitre présente une technique d'optimisation basée sur l'utilisation des caractéristiques matérielles du robot. En effet, certains processeurs embarquent un ensemble de d'instructions dites SIMD : Single Instruction Multiple Data. Elles permettent de traiter simultanément un ensemble de données. L'utilisation de ces instructions dans les tâches répétitives de l'algorithme a permis de réduire encore plus le temps d'exécution de *CoreSLAM*.

Après avoir optimisé le temps d'exécution de *CoreSLAM* au maximum, nous présentons dans la section 4.4 notre approche de SLAM parallèle, qui profite du gain de temps pour améliorer la qualité de des résultats de l'algorithme. En effet, durant un SLAM parallèle, nous lançons deux processus de localisation simultanément, avec des paramètres différents. On se base ensuite sur la meilleure estimation de position pour continuer la construction de la carte.

Plusieurs expérimentations et tests ont été effectués pour observer le fonctionnement de *CoreSLAM*. Leurs contextes font l'objet du chapitre 5.

Nous présentons dans ce chapitre les différentes plateformes utilisées, ainsi que leurs configurations matérielle et logicielle. Nous avons commencé les développements de notre algorithme de SLAM avec la version tinySLAM, sur la plateforme Mines Rover. Ensuite, dans le cadre de la participation du laboratoire de robotique CAOR au concours CAROTTE, nous avons utilisé des plateformes de commerce de la société Wifibot.

Le concours CAROTTE nous a offert une excellente opportunité pour améliorer différents aspect de notre algorithme de SLAM.

Le sixième et dernier chapitre est consacré à la présentation et l'analyse des résultats. Nous avons choisi de présenter des résultats expérimentaux liés à chacune des techniques fondamentales utilisées dans *CoreSLAM*. Nous finissons le chapitre par une évaluation générale de la qualité de cartographie (en profitant des environnements structurés du concours CAROTTE) et de la qualité de la localisation (en utilisant la méthode présentée dans [4]).

7.2 Perspectives d'amélioration

7.2.1 Intégrer la vision

Le SLAM par laser a atteint un bon niveau de maturité. Il reste néanmoins des cas qui posent quelques problèmes.

On pense notamment au classique cas de cartographie d'un couloir. En effet, l'absence d'éléments distinctifs sur les murs perturbe le processus de mise en correspondance, qui fournit ainsi une fausse localisation du robot. La figure 7.1 illustre ces propos.

Figure 7.1 – *Illustration du problème du corridor pour un capteur laser*

Un autre problème est apparu lors d'expériences avec des meubles en métal. En effet, nous avons remarqués des déformations de la forme des objets scannés par le capteur Laser. La figure 7.2 montre un exemple d'environnement scanné, et on peut observer la forme des données en provenance du capteur Laser sur la figure 7.3. Ces déformations peuvent perturber la localisation, dans la mesure où l'algorithme de mise en correspondance peut les prendre comme repères, ce qui donne encore une fois une estimation erronée de la position du robot.

Afin de résoudre ce genre de problème, nous avons pensé à un système de localisation par vision basé sur un filtrage de Kalman et intégré à *CoreSLAM*. L'idée de base consiste à fusionner les estimations du SLAM par Laser et du SLAM par vision au moment de la localisation, et garder la meilleure estimation en corrigeant l'autre. La figure FIG présente un schéma de fonctionnement possible de cette idée.

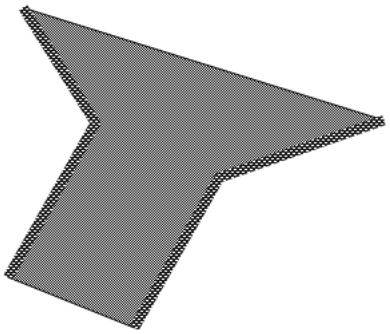


Figure 7.2 – *Schéma représentant la forme réelle de l'objet scanné*

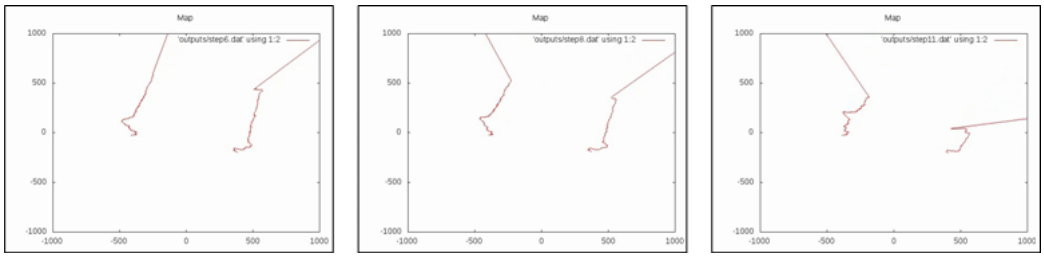


Figure 7.3 – *Données du Laser correspondant à l'objet de la figure 7.2, enregistrées pendant le déplacement du robot*

Cette stratégie de fusion permet de corriger intelligemment les erreurs et les incohérences de chaque méthode de SLAM. Ainsi, lorsque le processus de mise en correspondance du SLAM par laser n'arrive pas à estimer la localisation, le filtre de Kalman l'aide à prendre la bonne décision. D'un autre côté, à chaque fois que la localisation par Laser présente une bonne estimation, l'algorithme améliore les positions des amers (keypoints de l'image) utilisés par le filtre de Kalman, ce qui corrige le biais du filtre dû aux fortes hypothèses nécessaires au fonctionnement du SLAM par filtrage de Kalman. Par ailleurs, l'aide de la vision ne doit être appliquée que sur des sous-cartes de l'environnement. Ceci permet de limiter la charge de calcul nécessaire pour le SLAM visuel.

7.2.2 La fermeture de boucle

L'utilisation d'une approche incrémentale basée sur l'IML dans *CoreSLAM* ajoute inévitablement une petite erreur de localisation à chaque étape. Nous avons optimisé *CoreSLAM* en utilisant plusieurs techniques et méthodes afin de réduire cette erreur au maximum. Cependant, cette erreur s'accumule et peut devenir observable lorsqu'on exécute l'algorithme pour une longue période et un environnement d'une grande superficie. Ainsi, lorsque le robot fait une boucle dans l'environnement, sa position réelle risquerait d'être éloignée de la position estimée, comme illustré sur l'image de la figure 7.5.

La résolution de ce problème de divergence se base sur l'utilisation d'un algorithme de fermeture de boucle. Lorsque le robot reconnait un endroit qu'il a déjà visité, il peut estimer l'erreur accumulée et ainsi corriger la divergence. Ce problème de fermeture de

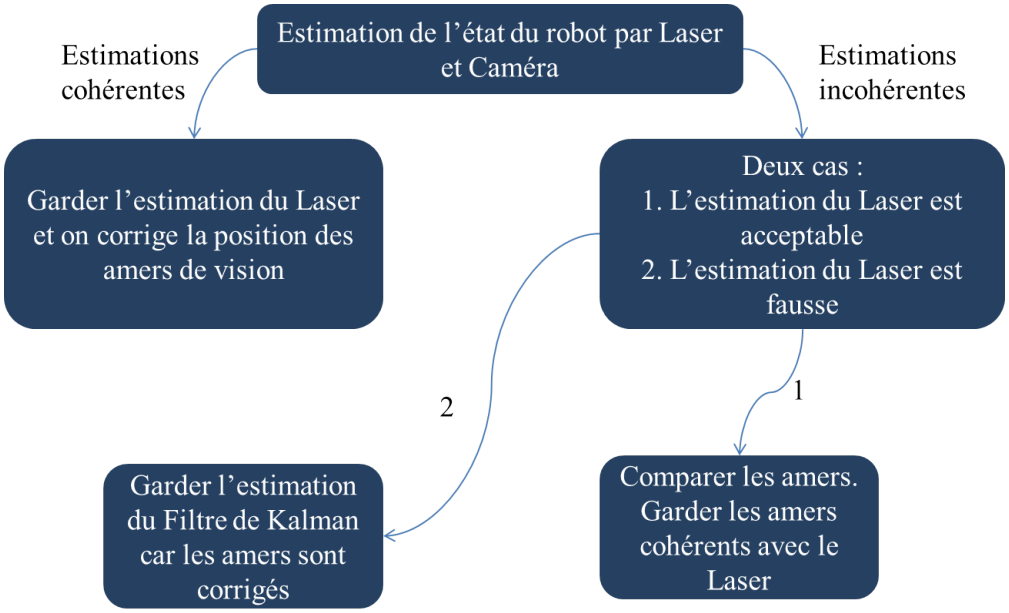


Figure 7.4 – *Schéma de fonctionnement de la stratégie de fusion proposée*

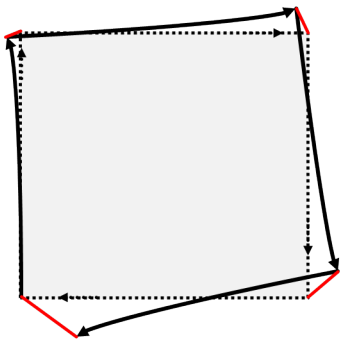


Figure 7.5 – *Illustration du problème de fermeture de boucle. La position réelle du robot est différente de la position estimée, à cause de l'accumulation des erreurs*

boucle est largement étudié dans le SLAM, et plusieurs travaux de recherche ont été menés afin de le résoudre.

Dans le cadre de l'utilisation de la vision, on peut citer les travaux de Newman and Ho [85], où les images servent à détecter la fermeture de boucle et le capteur laser permet d'effectuer la cartographie. Leur système permet au robot de détecter une fermeture de boucle et corriger ainsi sa position. Ils se sont basé sur des keypoints de type SIFT pour le traitement visuel de l'information.

Callmer and Granström ont présenté dans [86] un système similaire pour la détection de boucle. Mais l'utilisation d'un arbre de vocabulaire, introduit par Nister et Stewenius dans [87], leur a permis d'accélérer la recherche dans la base de données des caractéristiques des endroits déjà visités par le robot.

Si on souhaite se baser sur les données du capteur Laser pour la fermeture de boucle, on peut utiliser les résultats des travaux de Konolige et Gutmann [2]. Leur idée, illustrée sur l'image de la figure 7.6, se résume à l'utilisation d'une mesure de la corrélation entre

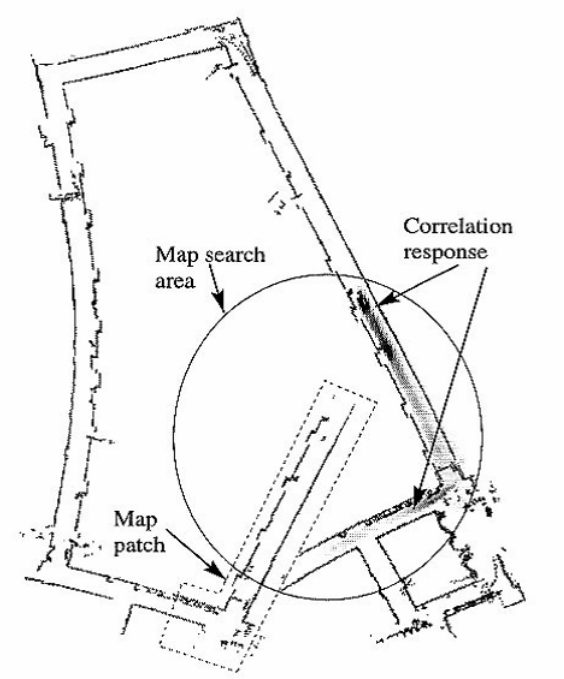


FIGURE 7.6 – *La méthode de fermeture de boucle présentée dans [2] compare une carte locale avec la carte globale de l'environnement*

une carte locale et la carte globale afin de détecter la présence du robot dans une partie de l'environnement déjà cartographiée.

Cette méthode nécessite une bonne estimation initiale pour la localisation, ce qui est déjà offert par *CoreSLAM*. Elle est rapide et permet une exécution temps réel de l'algorithme. La correction de l'erreur peut ensuite être effectuée en gardant plusieurs sous-cartes de l'environnement.

Bibliographie

[1] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping : part i," *IEEE Robotics and Automation Magazine*, pp. 99–108, 2006.

[2] J.-S. Gutmann and K. Konolige, "Incremental mapping of large cyclic environments," *IEEE International Symposium on Computational Intelligence in Robotics and Automation*, 2000.

[3] T.-D. Vu, "Localisation, mapping avec detection, classification et suivi des objets mobiles," Ph.D. dissertation, Institut National Polytechnique de Grenoble, 2009.

[4] W. Burgard, C. Stachniss, G. Grisetti, B. Steder, R. Kummerle, C. Dornhege, M. Ruhnke, A. Kleiner, and J. D. Tardos, "A comparison of slam algorithms based on a graph of relations," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2089 – 2095, 2009.

[5] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte carlo localization : Efficient position estimation for mobile robots," *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, 1999.

[6] R. C.Smith and P. Cheeseman, "On the representation and estimation of spatial uncertainty," *International Journal of Robotics Research*, vol. 5, pp. 56–68, 1986.

[7] H. Durrant-Whyte, "Uncertain geometry in robotics," *IEEE Journal of Robotics and Automation*, vol. 4, pp. 23–31, 1988.

[8] A. J. Davison, "Real-time simultaneous localisation and mapping with a single camera," *Ninth IEEE International Conference on Computer Vision (ICCV'03) - Volume 2*, 2003.

[9] J. Sola, A. e Monin, M. Devy, and T. Vidal-Calleja, "Fusing monocular information in multicamera slam," *IEEE Transactions on Robotics*, vol. 24, 2008.

[10] F. Lu and E. Milios, "Robot pose estimation in unknown environments bu matching 2d range scans," *Journal of Intelligent and Robotic Systems*, 1997.

[11] S. Riisgaard and M. R. Blas, *SLAM for Dummies. A Tutorial Approach to Simultaneous Localization and Mapping.*

[12] A. Eliazar and R. Parr, "Dp-slam : Fast, robust simultaneous localization and mapping without predetermined landmarks," *Proceedings of the International Joint Conference on Artificial Intelligence*, 2003.

[13] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "Fastslam : A factored solution to the simultaneous localization and mapping problem," *In Proceedings of the AAAI National Conference on Artificial Intelligence*, pp. 593–598, 2002.

[14] A. Davison and D. Murray, "Simultaneous localization and map-building using active vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, pp. 865 – 880, 2002.

[15] C. Estrada, J. Neira, and J. Tardos, "Hierarchical slam : real-time accurate mapping of large environments," *IEEE Transactions on Robotics*, vol. 21, pp. 588–596, 2005.

[16] J. Leonard and P. Newman, "Consistent, convergent, and constant-time slam," *Proceedings of the 18th international joint conference on Artificial intelligence*, pp. 1143–1150, 2003.

[17] J. Kim and S. Sukkarieh, "Airborne simultaneous localisation and map building," *IEEE International Conference on Robotics and Automation*, vol. 1, pp. 406 – 411, 2003.

[18] ——, "Autonomous airborne navigation in unknown terrain environments," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 40, pp. 1031 – 1045, 2004.

[19] S. Williams, P. Newman, G. Dissanayake, and H. Durrant-Whyte, "Autonomous underwater simultaneous localisation and map building," *IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1793 – 1798, 2000.

[20] J. Saez, A. Hogue, F. Escolano, and M. Jenkin, "Underwater 3d slam through entropy minimization," *IEEE International Conference on Robotics and Automation*, pp. 3562 – 3567, 2006.

[21] R. Eustice, "Large-area visually augmented navigation for autonomous underwater vehicles," Ph.D. dissertation, Massachusetts Institute of Technology, 2005.

[22] R. Ouellette and K. Hirasawa, "A comparison of slam implementations for indoor mobile robots," *Proceedings of the 2007 IEEURSJ, Int. Conference on Intelligent Robots and Systems*, 2007.

[23] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous Robots*, vol. 4, pp. 333 – 349, 1997.

[24] J. Leonard and H. Durrant-Whyte, "Simultaneous map building and localization for an autonomous mobile robot," *IEEE/RSJ International Conference on Intelligent Robots and System*, 1991.

[25] A. Elfes, "Occupancy grids : a probabilistic framework for robot perception and navigation," Ph.D. dissertation, Carnegie Mellon University, 1989.

[26] T. Einsele and G. Farber, "Real-time self-localization in unknown indoor environments using a panorama laser range finder," *IEEE/RSJ International Workshop on Robots ans Systems*, pp. 697–703, 1997.

[27] S. Pfister, S. Roumeliotis, and J. Burdick, "Weighted line fitting algorithms for mobile robot map building and efficient data representation," *IEEE International Conference on Robotics and Automation*, vol. 1, pp. 1304 – 1311, 2003.

[28] D. A. Forsyth and J. Ponce, *Computer Vision : A Modern Approach.* Prentice Hall, 2011.

[29] A. Elfes, "Multi-source spatial data fusion using bayesian reasoning integration," *Data Fusion in Robotics and Machine Intelligence*, pp. 137–163, 1992.

[30] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics.* The MIT Press, 2005.

[31] S. Thrun, "Learning maps for indoor mobile robot navigation," Carnegie Mellon University, Tech. Rep., 1997.

[32] D. Pagac, E. Nebot, and H. Durrant-Whyte, "An evidential approach to map-building for autonomous vehicles," *IEEE Transactions on Robotics and Automation*, vol. 14, pp. 623 – 629, 1998.

[33] F. Vincent, "Modelisation de l'environnement et localisation pour un véhicule," Master's thesis, L'Institut National Polytechnique de Grenoble, 1997.

[34] G. Oriolo, M. Vendittelli, and G. Ulivi, "Fuzzy maps : A new tool for mobile robot perception and planning," *Journal of Robotic Systems*, vol. 14, pp. 179–197, 1997.

[35] B. Abdellatif, "Cartographie de l'environnement et suivi simultané de cibles dynamiques par un robot mobile," Ph.D. dissertation, Université Paul SABATIER, 2007.

[36] C.-C. Wang, "Simultaneous localization, mapping and moving object tracking," Ph.D. dissertation, Carnegie Mellon University, 2004.

[37] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, pp. 35–45, 1960.

[38] R. Smith, M. Self, and P. Cheeseman, "Estimating uncertain spatial relationships in robotics," *Autonomous Robot Vehicles*, pp. 167–193, 1990.

[39] P. Maybeck, "The kalman filter : An introduction to concepts," *Autonomous Robot Vehicles*, pp. 194 – 204, 1990.

[40] M. Csorba, "Simultaneous localisation and map building," Ph.D. dissertation, University of Oxford, 1997.

[41] T. Bailey, J. Nieto, J. Guivant, M. Stevens, and E. Nebot, "Consistency of the ekf-slam algorithm," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006.

[42] E. Wan and R. van der Merwe, *Kalman Filtering and Neural Networks.* John Wiley & Sons Inc, 2001, ch. The Unscented Kalman Filter.

[43] J. Guivant and E. Nebot, "Optimization of the simultaneous localization and map building algorithm for real time implementation," *IEEE Transactions on Robotics and Automation*, vol. 17, pp. 242–257, 2001.

[44] W. Burgard, D. Fox, H. Jans, C. Matenar, and S. Thrun, "Sonar-based mapping of large-scale mobile robot environments using em," *Proceedings of the Sixteenth International Conference on Machine Learning*, pp. 67 – 76, 1999.

[45] S. Thrun, W. Burgard, D. Fox, H. Hexmoor, and M. Mataric, "A probabilistic approach to concurrent mapping and localization for mobile robots," *Machine Learning*, 1998.

[46] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society*, vol. 39, pp. 1–38, 1977.

[47] N. Metropolis and S. Ulam, "The monte carlo method," *Journal of the American Statistical Association*, vol. 44, pp. 335–341, 1949.

[48] D. Blackwell, "Conditional expectation and unbiased sequential estimation," *The Annals of Mathematical Statistics*, vol. 18, pp. 105–110, 1947.

[49] C. Rao, "Information and accuracy obtainable in estimation of statistical parameters," *Bulletin of the Calcutta Mathematical Society*, 1945.

[50] F. Ramos, D. Fox, and H. Durrant-Whyte, "Crf-matching : Conditional random fields for feature-based scan matching," *Proc. of Robotics : Science & Systems*, 2007.

[51] S. Thrun, D. Fox, and W. Burgard, "A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping," *IEEE International Conference on Robotics and Automation*, 2000.

[52] D. Hähnel, S. Thrun, B. Wegbreit, and W. Burgard, "Towards lazy data association in slam," *International Symposium on Robotics Research*, 2003.

[53] J. Aulinas, Y. Petillot, J. Salvi, and X. Llado, "The slam problem : a survey," *Conference on Artificial Intelligence Research and Development : Proceedings of the 11th International Conference of the Catalan Association for Artificial Intelligence*, pp. 363–371, 2008.

[54] S. Se, D. Lowe, and J. Little, "Vision-based global localization and mapping for mobile robots," *IEEE Transactions on Robotics*, vol. 21, pp. 364 – 375, 2005.

[55] L. M. Paz, J. D. Tardós, and J. Neira, "Divide and conquer : Ekf slam in 0(n)," *IEEE Transactions on Robotics*, pp. 1107 – 1120, 2008.

[56] T. Chretien, M. Houdeau, O. Liandrat, F. Rixain, and L. Reveret, "http://www.terk.ri.cmu.edu," 2011.

[57] S. L. Corff, G. Fort, and E. Moulines, "Online expectation maximization algorithm to solve the slam problem," *IEEE Statistical Signal Processing Workshop*, pp. 225 – 228, 2011.

[58] S. Thrun and M. Montemerlo, "The graphslam algorithm with applications to large-scale mapping of urban structures," *International Journal on Robotics Research*, vol. 25, pp. 403–430, 2005.

[59] J.-S. Gutmann, "Robuste navigation autonomer mobiler systeme," Ph.D. dissertation, Albert-Ludwigs-Universitat Freiburg, 2000.

[60] K. L. andH. Surmann, A. Nuchter, and J. Hertzberg, "Indor and outdoor localization for fast mobile robots," *IEEE International Conference on Intelligent Robot and Systems*, 2004.

[61] I. J. Co, "Blanche-an experiment in guidance and navigation of an autonomous robot vehicle," *IEEE Transactions on Robotics and Automation*, 1991.

[62] P. Biber, "The normal distributions transform : A new approach to laser scan matching," *IEEE/RSJ Int. Conference on Intelligent Robots and Systems*, 2003.

[63] P. J. Besl and N. D. McKay, "A method for registration of 3d shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1992.

[64] A. Diosi and L. Kleeman, "Laser scan matching in polar coordinates with application to slam," *IEEE International Conference on Intelligent Robot and Systems*, 2005.

[65] A. Censi, L. Iocchi, and G. Grisetti, "Scan matching in the hough domain," *IEEE International Conference on Robotics and Automation*, 2005.

[66] J. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems Journal*, vol. 4, pp. 25–30, 1965.

[67] M. Mitchell, *An Introduction to Genetic Algorithms*. The MIT Press, 1998.

[68] J. H. Holland, *Adaptation in Natural and Artificial Systems*. MIT Press, 1975.

[69] T. Duckett, "A genetic algorithm for simultaneous localization and mapping," *International Conference on Robotics and Automation. ICRA'03*, 2003.

[70] J. M. A. M. a. A. J. G.-C. Jorge L. Martinez, Javier Gonzalez, "Mobile robot motion estimation by 2d scan matching with genetic and iterative closest point algorithms," *Journal of Field Robotics*, 2006.

[71] A. P. S. J. F. Dong, W. Sardha Wijesoma, "An efficient rao-blackwellized genetic algorithmic filter for slam," *International Conference on Robotics and Automation*, 2007.

[72] M. Begum, G. K. I. Mann, and R. G. Gosine, "Integrated fuzzy logic and genetic algorithmic approach for simultaneous localization and mapping of mobile robots," *Applied Soft Computing*, vol. 8, 2008.

[73] R. Schaffer and R. Sedgewick, "The analysis of heapsort," *Journal of Algorithms*, vol. 15, pp. 76–100, 1993.

[74] S. Kyo, S. Okazaki, and I. Kuroda, "An extended c language and a simd compiler for efficient implementation of image filters on media extended micro-processors," *Proceedings of Advanced Concepts for Intelligent Vision Systems*, 2003.

[75] A. . F. . Failure, "sunnyday.mit.edu," 1996.

[76] iRobot Vacuum Cleaning Robot, "www.irobot.com."

[77] C. SAMSON and K. AIT-ABDERRAHIM, "Mobile robot control part 1 : Feedback control of a nonholonomic wheeled cart in cartesian space," *Rapport de recherche INRIA N°1288*, 1990.

[78] N. Burrus, "Kinect calibration. http://nicolas.burrus.name/index.php/Research/KinectCalibration.

[79] D. Rina and P. Judea, "Generalized best-first search strategies and the optimality of a*," *Journal of the ACM*, vol. 32, pp. 505–536, 1985.

[80] C. . R. N. Toolkit, "carmen.sourceforge.net."

[81] A. Censi, "Scan matching in a probabilistic framework," *International Conference on Robotics and Automation*, pp. 2291–2296, 2006.

[82] G. Grisetti, C. Stachniss, and W. Burgard, "Improved tehniques for grid mapping with rao-blackwellized partile filtres," *IEEE transactions on Robotics*, vol. 23, pp. 34–46, 2007.

[83] OpenSLAM.org, "www.openslam.org."

[84] E. Olson, "Robust and efficient robotic mapping," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, USA, 2008.

[85] P. Newman and K. Ho, "Slam - loop closing with visually salient features," *IEEE International Conference on Robotics and Automation*, 2005.

[86] J. Callmer and K. Granstrom, "Large scale slam in an urban environment," Ph.D. dissertation, Linkoping University, 2008.

[87] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 2161–2168, 2006.

[88] G. G. Garrido, "Développement d'un capteur composite vision/laser à couplace serré pour le slam d'intérieur," Ph.D. dissertation, Ecole Nationale Supérieur des Mines de Paris, 2011.

[89] S. J. Hennessy and R. H. King, "Future mining technology spinoffs from the alv program," *IEEE Transactions on Industry Applications*, vol. 25, pp. 377–384, 1989.

Annexes

A

Code source de *tinySLAM*

Algorithme A.1 *tinySLAM* : entête et structures de données

```
#ifndef _tinySLAM_H_
#define _tinySLAM_H_

#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif

#define TS_SCAN_SIZE 8192
#define TS_MAP_SIZE 2048
#define TS_MAP_SCALE 0.1
#define TS_DISTANCE_NO_DETECTION 4000
#define TS_NO_OBSTACLE 65500
#define TS_OBSTACLE 0
#define TS_HOLE_WIDTH 600

typedef unsigned short ts_map_pixel_t;

typedef struct {
    ts_map_pixel_t map[TS_MAP_SIZE * TS_MAP_SIZE];
} ts_map_t;

typedef struct {
    double x[TS_SCAN_SIZE], y[TS_SCAN_SIZE];
    int value[TS_SCAN_SIZE];
    int nb_points;
} ts_scan_t;

typedef struct {
    double x, y;     // in mm
    double theta;    // in degrees
} ts_position_t;

void ts_map_init(ts_map_t *map);
int ts_distance_scan_to_map(ts_scan_t *scan, ts_map_t *map, ts_position_t *pos);
void ts_map_update(ts_scan_t *scan, ts_map_t *map, ts_position_t *position,
                   int quality);

#endif // _tinySLAM_H_
```

Algorithme A.2 *tinySLAM* : La mesure de la distance d'un scan à la carte

```
int ts_distance_scan_to_map(ts_scan_t *scan, ts_map_t *map, ts_position_t *pos)
{
    double c, s;
    int i, x, y, nb_points = 0;
    int64_t sum;

    c = cos(pos->theta * M_PI / 180);
    s = sin(pos->theta * M_PI / 180);
    // Translate and rotate scan to robot position
    // and compute the distance
    for (i = 0, sum = 0; i != scan->nb_points; i++) {
        if (scan->value[i] != TS_NO_OBSTACLE) {
            x = (int)floor((pos->x + c * scan->x[i]
                - s * scan->y[i]) * TS_MAP_SCALE + 0.5);
            y = (int)floor((pos->y + s * scan->x[i]
                + c * scan->y[i]) * TS_MAP_SCALE + 0.5);
            // Check boundaries
            if (x >= 0 && x < TS_MAP_SIZE && y >= 0 && y < TS_MAP_SIZE) {
                sum += map->map[y * TS_MAP_SIZE + x];
                nb_points++;
            }
        }
    }
    if (nb_points) sum = sum * 1024 / nb_points;
    else sum = 2000000000;
    return (int)sum;
}

void ts_map_init(ts_map_t *map)
{
    int x, y, initval;
    ts_map_pixel_t *ptr;
    initval = (TS_OBSTACLE + TS_NO_OBSTACLE) / 2;
    for (ptr = map->map, y = 0; y < TS_MAP_SIZE; y++) {
            for (x = 0; x < TS_MAP_SIZE; x++, ptr++) {
                *ptr = initval;
            }
    }
}
```

Algorithme A.3 *tinySLAM* : la fonction de mise-à-jour de la carte

```
 1 void ts_map_update(ts_scan_t *scan, ts_map_t *map, ts_position_t *pos, int quality)
   {
       double c, s, q;
       double x2p, y2p;
       int i, x1, y1, x2, y2, xp, yp, value;
 6     double add, dist;

       c = cos(pos->theta * M_PI / 180);
       s = sin(pos->theta * M_PI / 180);
       x1 = (int)floor(pos->x * TS_MAP_SCALE + 0.5);
11     y1 = (int)floor(pos->y * TS_MAP_SCALE + 0.5);
       // Translate and rotate scan to robot position
       for (i = 0; i != scan->nb_points; i++) {
           x2p = c * scan->x[i] - s * scan->y[i];
           y2p = s * scan->x[i] + c * scan->y[i];
16         xp = (int)floor((pos->x + x2p) * TS_MAP_SCALE + 0.5);
           yp = (int)floor((pos->y + y2p) * TS_MAP_SCALE + 0.5);
           dist = sqrt(x2p * x2p + y2p * y2p);
           add = TS_HOLE_WIDTH / 2 / dist;
           x2p *= TS_MAP_SCALE * (1 + add);
21         y2p *= TS_MAP_SCALE * (1 + add);
           x2 = (int)floor(pos->x * TS_MAP_SCALE + x2p + 0.5);
           y2 = (int)floor(pos->y * TS_MAP_SCALE + y2p + 0.5);
           if (scan->value[i] == TS_NO_OBSTACLE) {
               q = quality / 2;
26             value = TS_NO_OBSTACLE;
           } else {
               q = quality;
               value = TS_OBSTACLE;
           }
31         ts_map_laser_ray(map, x1, y1, x2, y2, xp, yp, value, q);
       }
   }
```

Algorithme A.4 *tinySLAM* : fonction de tracé du rayon laser

```
#define SWAP(x, y) (x ^= y ^= x ^= y)

void ts_map_laser_ray(ts_map_t *map, int x1, int y1, int x2, int y2,
                      int xp, int yp, int value, int alpha)
{
    int x2c, y2c, dx, dy, dxc, dyc, error, errorv, derrorv, x;
    int incv, sincv, incerrorv, incptrx, incptry, pixval, horiz, diago;
    ts_map_pixel_t *ptr;

    if (x1 < 0 || x1 >= TS_MAP_SIZE || y1 < 0 || y1 >= TS_MAP_SIZE)
        return; // Robot is out of map

    x2c = x2; y2c = y2;
    // Clipping
    if (x2c < 0) {
        if (x2c == x1) return;
        y2c += (y2c - y1) * (-x2c) / (x2c - x1);
        x2c = 0;
    }
    if (x2c >= TS_MAP_SIZE) {
        if (x1 == x2c) return;
        y2c += (y2c - y1) * (TS_MAP_SIZE - 1 - x2c) / (x2c - x1);
        x2c = TS_MAP_SIZE - 1;
    }
    if (y2c < 0) {
        if (y1 == y2c) return;
        x2c += (x1 - x2c) * (-y2c) / (y1 - y2c);
        y2c = 0;
    }
    if (y2c >= TS_MAP_SIZE) {
        if (y1 == y2c) return;
        x2c += (x1 - x2c) * (TS_MAP_SIZE - 1 - y2c) / (y1 - y2c);
        y2c = TS_MAP_SIZE - 1;
    }

    dx = abs(x2 - x1); dy = abs(y2 - y1);
    dxc = abs(x2c - x1); dyc = abs(y2c - y1);
    incptrx = (x2 > x1) ? 1 : -1;
    incptry = (y2 > y1) ? TS_MAP_SIZE : -TS_MAP_SIZE;
    sincv = (value > TS_NO_OBSTACLE) ? 1 : -1;
    if (dx > dy) {
        derrorv = abs(xp - x2);
    } else {
        SWAP(dx, dy); SWAP(dxc, dyc); SWAP(incptrx, incptry);
        derrorv = abs(yp - y2);
    }
    error = 2 * dyc - dxc;
    horiz = 2 * dyc;
    diago = 2 * (dyc - dxc);
    errorv = derrorv / 2;
    incv = (value - TS_NO_OBSTACLE) / derrorv;
    incerrorv = value - TS_NO_OBSTACLE - derrorv * incv;
    ptr = map->map + y1 * TS_MAP_SIZE + x1;
    pixval = TS_NO_OBSTACLE;
    for (x = 0; x <= dxc; x++, ptr += incptrx) {
        if (x > dx - 2 * derrorv) {
            if (x <= dx - derrorv) {
                pixval += incv;
                errorv += incerrorv;
                if (errorv > derrorv) {
                    pixval += sincv;
                    errorv -= derrorv;
                }
            } else {
                pixval -= incv;
                errorv -= incerrorv;
                if (errorv < 0) {
                    pixval -= sincv;
                    errorv += derrorv;
                }
            }
        }
        // Integration into the map
        *ptr = ((256 - alpha) * (*ptr) + alpha * pixval) >> 8;
        if (error > 0) {
            ptr += incptry;
            error += diago;
        } else error += horiz;
    }
}
```

B

Le filtre de Kalman

B.1 Le filtre de Kalman

Ce chapître est adaptée de [88].

Le filtre de Kalman traite le problème d'estimation de l'état x d'un processus discret déterminé par l'équation suivante :

$$x_t = Ax_{t-1} + Bu_t + w_{t-1}$$

avec des mesures z ayant un bruit gaussien v et qu'on peut exprimer ainsi :

$$z_t = Hx_t + v_t$$

où :

- v_t représente le bruit des mesures. $v \sim N(0, \mathbf{R})$
- w_t représente le bruit du processus. $w \sim N(0, \mathbf{Q})$

Equations de prédiction et de mise à jour

L'état du système au moment de la réception de la prochaine mesure peut être prédit :

$$x_{t+1/t} = Ax_{t/t} + Bu_t$$

La covariance de la prédiction de l'état :

$$P_{t+1/t} = AP_{t/t}A^T + Q_t$$

Equation de mise à jour des mesures

L'innovation pondérée par le gain du filtre, plus l'état prédit, à partir de l'estimation de l'état mise à jour :

$$x_{t+1/t+1} = x_{t+1/t} + W_{t+1}v_{t+1}$$

La covariance de l'état mise à jour :

$$P_{t+1/t+1} = P_{t+1/t} - W_{t+1}S_{t+1}W_{t+1}^T$$

où :

- L'innovation v est la différence entre la mesure réelle et la mesure prédite :

$$v_{t+1} = z_{t+1} - Hx_{t+1/t}$$

- Le gain de Kalman W est :

$$W_{t+1} = P_{t+1/t}H_{t+1/t}^T S_{t+1}^{-1}$$

– La covariance de l'innovation S est :

$$S_{t+1} = H_{t+1/t} P_{t+1/t} H_{t+1/t}^T + R_{t+1}$$

où R est la covariance du bruit de mesure.

B.2 Le filtre de Kalman étendu : EKF

Les équations de transition et de mesure ne sont pas toujours linéaires. Le filtre EKF est une extension du filtre de Kalman permettant de traiter ces problèmes de non linéarité. Les simplifications mathématiques introduites ont toutefois un inconvénient : les distributions de probabilités ne sont plus modélisées correctement et la linéarisation cause des inconsistances. Cependant, en pratique, les résultats sont souvent satisfaisants.

Equations de prédiction et de mise à jour

$$x_{t+1/t} = f(x_{t/t}, u_t)$$

$$P_{t+1/t} = (\nabla_x f)_{t/t} P_{t/t} (\nabla_x f)_{t/t}^T + Q_t$$

où :
– f est l'équation de mise à jour de l'état
– $x_{t/t}$est l'estimation de l'état à l'instant t en se basant sur l'information à l'instant t
– $x_{t+1/t}$est l'estimation de l'état à l'instant $t+1$ en se basant sur le modèle de transition (sans intégrer l'information de mesure)
– P correspond à la matrice de covariance
– Q représente la matrice de covariance du bruit du processus

Equation de mise à jour des mesures

$$x_{t+1/t+1} = x_{t+1/t} + W_{t+1} v_{t+1}$$

$$P_{t+1/t+1} = P_{t+1/t} - W_{t+1} S_{t+1} W_{t+1}^T$$

Les équations de mise à jour des mesures ajoutent des informations à partir des nouvelles mesures afin de corriger les estimations faites à partir du modèle de transition. v est appelée l'innovation et correspond à l'ensemble de l'information non prédite ayant été obtenue à partir des mesures. W est le gain de Kalman, et il exprime le degré de confiance qu'on a dans les mesures.

$$v_{t+1} = z_{t+1} - h x_{t+1/t}$$

$$W_{t+1} = P_{t+1/t}(\nabla_x h)^T_{t+1/t} S^{-1}_{t+1}$$

$$S_{t+1} = (\nabla_x h)_{t+1/t} P_{t+1/t} (\nabla_x h)^T_{t+1/t} + R_{t+1}$$

R est la covariance du bruit de mesure.

C

Les expériences du DARPA sur les ALV

Year	Distance	Speed	Capability
May 1985 (preliminary road-following demonstration)	1 km	5 km/h top, 3 km/h average	The vehicle traversed a straight uniform-surfaced portion of the test area and used a prestored map of the test track to navigate when the vision system failed to provide navigable scene models. 3 of 1200 created scene models were unusable.
June 1986 (road-following demonstrations)	4.2 km	10 km/h top, 6 km/h average	The vehicle traveled the length of the 2.1-km test track, counter-rotated and returned. The track included nonuniform road surfaces and a sharp curve. Vehicle speed varied as a function of an algorithm-defined confidence in the quality of scene models. No map input was used to compensate for poor scene models.
October 1986 (fast road-following and obstacle-avoidance technology status review)	0.4 km	17.5 km/h top for fast road following, 3 km/h for top obstacle avoidance	Two separate prototype software experiments: in the first, the vehicle used video data to travel a short section of test track at fast speeds; in a second experiment, actively scanned laser range data was used to avoid obstacles within video-generated road boundaries at slower speeds.
1986-1987 (long-range system and architecture study)			Aggressive long-range goals for the program motivated a comprehensive study and redesign of the vehicle's mobility, communication, and computing systems. All system upgrades were completed toward the end of summer 1987 .
November 1987 (road-following and obstacle-avoidance demonstration)	4.2 km	22 km/h top, 12 km/h average, 5 km/h for obstacle avoidance	A single integrated demonstration in which the vehicle navigated through obstacles in a map-cued portion of the test track and throughout the rest of the run achieved top speeds governed by heuristics measuring the "goodness" of the scene models produced by the vision system.
December 1987 (demonstration of off-road navigation)	1 km	1 km/h	A team from the Hughes Research Center implemented and demonstrated an off-road navigation system that used hybrid lab/vehicle processing, connected via a digital RF link .
January 1988 and continuing			The ALV program is in the process of supporting experiments by both ALV researchers at participating universities and research centers and the Martin Marietta ALV team. Planned experiments include off-road navigation, hybrid road/off-road navigation, landmark recognition, new sensor integration, intersection recognition for traversal of networks of roads, and sensor fusion for navigation.

Table C.1 – *Les démonstrations de DARPA dans le cadre des ALV*

Le tableau présenté ici est extrait de [89]

D

La précision de la Kinect

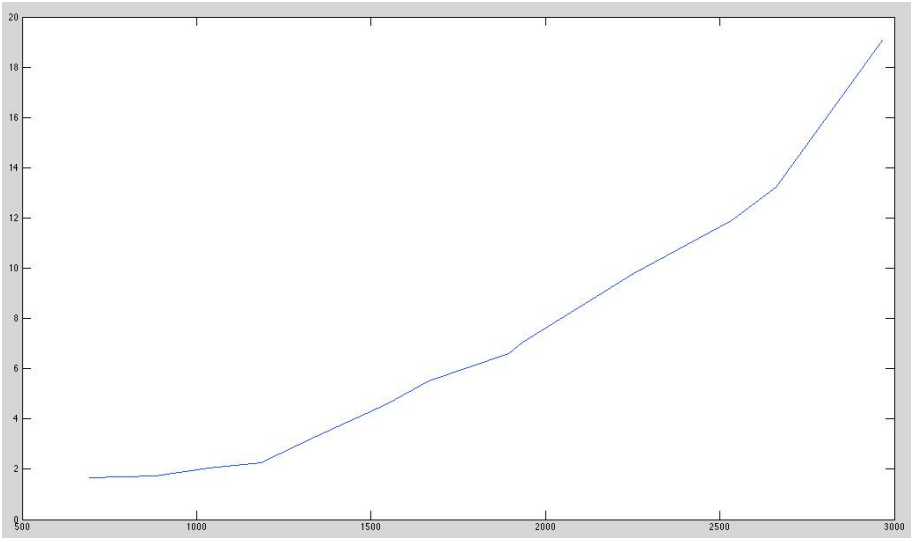


Figure D.1 – *Erreur moyenne (en mm) en fonction de la distance (en mm)*

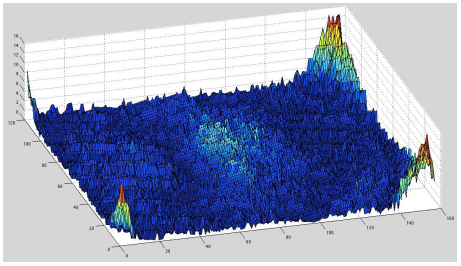
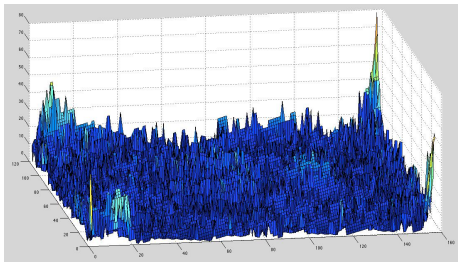
(a)  (b) 

Figure D.2 – *Graphe des erreurs (en valeur absolue) à (a) d=694mm et (b) d=1936mm*

Pour étudier la précision de la Kinect, le travail présenté dans [56] est intéressant. En effet, après avoir récupéré la carte de profondeur d'un mur, on construit un nuage de points 3D. Une méthode d'approximation permet ensuite de calculer l'équation du plan le plus proche de ce nuage de points. On déduit ainsi l'erreur en profondeur en calculons l'erreur de positionnement de chaque point, par rapport à ce plan.

Cette méthode donne le graphe de la figure D.1.

L'erreur en profondeur varie en fonction de la distance de la Kinect aux obstacles. La figure D.2 montre une comparaison entre l'erreur de profondeur sur le nuage de points 3D du mur, pour une distance de 694 mm entre le capteur Kinect et le mur, et puis pour une distance de 1936 mm.

D'autres expériences (du même document [56]) ont montré que la précision des mesures en profondeur est de l'ordre du millimètre pour les objets se situant à moins d'un mètre de la Kinect.

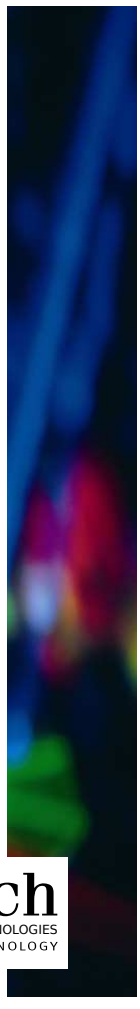
Localisation et Cartographie Simultanées pour un robot mobile équipé d'un laser à balayage : CoreSLAM

Résumé : La thématique de la **navigation autonome** constitue l'un des principaux axes de recherche dans le domaine des véhicules intelligents et des robots mobiles. Dans ce contexte, on cherche à doter le robot d'algorithmes et de méthodes lui permettant d'évoluer dans un environnement complexe et dynamique, en toute **sécurité** et en parfaite **autonomie**. Dans ce contexte, les algorithmes de localisation et de cartographie occupent une place importante. En effet, sans informations suffisantes sur la position du robot (localisation) et sur la nature de son environnement (cartographie), les autres algorithmes (génération de trajectoire, évitement d'obstacles ...) ne peuvent pas fonctionner correctement.
Nous avons centré notre travail de thèse sur une problématique précise : développer un algorithme de SLAM **simple**, **rapide**, **léger** et **limitant les erreurs** de localisation et de cartographie au maximum sans fermeture de boucle. Au cœur de notre approche, on trouve un algorithme d'IML : Incremental Maximum Likelihood. Ce type d'algorithmes se base sur une estimation itérative de la localisation et de la cartographie. Il est ainsi naturellement divergent. Le choix de l'IML est justifié essentiellement par sa simplicité et sa légèreté.
La particularité des travaux réalisés durant cette thèse réside dans les différents outils et algorithmes utilisés afin de limiter la divergence de l'IML au maximum, tout en conservant ses avantages.

Mots clés : SLAM, Localisation, Cartographie, Robotique mobile, Conduite automatique

Simultaneous Localization and Mapping for a mobile robot with a Laser : CoreSLAM

Abstract: One of the main areas of research in the field of intelligent vehicles and mobile robots is **Autonomous navigation**. In this field, we seek to create algorithms and methods that give robots the ability to move **safely** and **autonomously** in a complex and dynamic environment. In this field, localization and mapping algorithms have an important place. Indeed, without reliable information about the robot position (localization) and the nature of its environment (mapping), the other algorithms (trajectory generation, obstacle avoidance ...) cannot achieve their tasks properly.
We focused our work during this thesis on a specific problem: to develop a **simple**, **fast** and **lightweight** SLAM algorithm that can **minimize localization errors** without loop closing. At the center of our approach, there is an IML algorithm: Incremental Maximum Likelihood. This kind of algorithms is based on an iterative estimation of the localization and the mapping. It contains thus naturally a growing error in the localization process. The choice of IML is justified mainly by its simplicity and lightness. The main idea of our work is built around the different tools and algorithms used to minimize the localization error of IML, while keeping its advantages.
Keywords: SLAM, Localization, Mapping, Mobile Robots, Automatic driving



MINES ParisTech

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY