Véhicules autonomes et environnement semi-statiques

Iker Bellicot

Master recherche

Spécialité Imagerie Vision et Robotique

Véhicules autonomes et environnements semi-statiques

Iker Bellicot
iker.bellicot@inrialpes.fr

Encadrants :
Christopher Tay Meng Keat
Christian Laugier

26 septembre 2006

Projet préparé au sein de l'équipe eMotion

Laboratoire GRAVIR
INRIA Rhône-Alpes
ZIRST 655 Avenue de
l'Europe
38334 Saint Ismier Cedex
FRANCE

Table des matières

Résumé

Les robots autonomes utilisent des balises (murs, coins en intérieur ; arbres, bâtiments à l'extérieur) pour se localiser. Cependant cette localisation peut échouer si le robot confond les balises entre eux, ce qui peut arriver lorsque l'environnement change (dont l'exemple typique est le parking, pour lequel les voitures ne sont plus stationnées au même endroit) ; l'imprécision du capteur peut aussi laisser penser qu'un amer a bougé. La difficulté réside alors dans la comparaison des cartes d'amers : le nombre de points de comparaison est tel que des confusions sont possibles.

La majorité des algorithmes de SLAM existants ne prend pas en compte cet aspect semi-statique de l'environnement, la plupart se concentrant sur les problèmes des milieux ouverts (environnements comportant des objets mobiles). Toutefois parmi les approches proposées deux se détachent : la première tente de modéliser l'ensemble des configurations que peut prendre l'environnement. Cela fonctionne si la majeure partie de l'environnement reste statique. Son utilisation en milieu extérieur en devient par conséquent impossible. La seconde utilise le principe de l'oubli pour modéliser les changements de l'environnement. À chaque objet de l'environnement sont affectés des paramètres. L'évolution de ces paramètres manifeste l'évolution de la carte. L'emploi de tels paramètres limite le champ d'utilisation d'une telle méthode.

Nous proposons de représenter les objets par des ensembles de "‘patches"’ regroupés dans une structure hiérarchique. Cette structure permet de tempérer l'impact négatif dû à de mauvaises informations du capteur et de déterminer aisément les changements de position des repères. L'algorithme utilisé pour le SLAM est de type FastSLAM. Celui-ci utilise un filtre de particules qui représentent les positions possibles du robot auxquelles sont associées des cartes de l'environnement. En particulier, cela permet en cas d'indécision de travailler avec toutes les hypothèses qui correspondent à la situation courante.

Chapitre 1

Introduction

1.1 Motivations

La robotique a permis à l'homme de déléguer les tâches répétitives (par exemple dans les usines de montage de voitures) et dangereuses (robots utilisés lors de déminages, dans les zones à fortes radiations...). La robotique mobile est une partie importante de la robotique, puisqu'elle aborde le domaine de l'autonomie qui s'avère indispensable pour certaines applications comme l'exploration où le téléguidage est impossible. C'est le cas des robots envoyés sur Mars ou dans les fonds marins. Ces robots doivent pouvoir se déplacer de manière autonome.

Pour accomplir sa tâche, le robot effectue des actions qui dépendent de sa position et de celle de son but. Sa localisation est donc primordiale. Ainsi, un robot domestique doit à tout moment être en mesure de rejoindre sa base pour se recharger ; un robot guide doit être capable de se situer sur le trajet de la visite afin de donner les bonnes informations. Pour se localiser le robot peut avoir recours à des aides extérieures. Equipé d'un récepteur GPS (*Global Positing System*) par exemple, il peut connaître sa position absolue par triangulation de 4 satellites. Malheureusement ce système trouve ses limites dans les endroits où le signal est imprécis, pour lesquels la précision de l'ordre de quelques mètres n'est pas suffisante pour déterminer une trajectoire dans un milieu contenant des objets mobiles ; et les endroits où le signal est très faible, comme dans les environnements sous-marins.

Le robot peut aussi employer des capteurs (odométrie, laser, sonar...) pour récupérer des informations sur le milieu environnant. L'odométrie par exemple détermine la trajectoire du robot à partir du nombre de tours effectués par les roues. Il est ainsi possible de savoir si la trajectoire effectuée est rectiligne ou non et quelle est sa longueur. En complétant les informations odométriques avec des données acquises par un capteur "extéroceptif" (capteur renseignant le robot sur le monde environnant), il peut déterminer sa position relative dans l'environnement. Cependant ces mesures sont soumises à des imprécisions : si les roues viennent à glisser les calculs de la trajectoires seront faux. De façon plus générale, les capteurs ne peuvent être considérés parfaits. Leurs mesures sont toujours entachées d'incertitude.

Afin de limiter les erreurs dues à l'odométrie, une carte de l'environnement peut être proposée en complément. Les mesures sur l'environnement sont comparées aux mesures théoriques et permettent de rectifier la trajectoire. Le robot peut alors se localiser relativement plus précisément. Cependant ce système trouve rapidement ses limites : la connaissance précise du monde entourant le robot n'est pas toujours disponible (notam-

ment pour les robots explorateurs) et surtout ces cartes ne tiennent pas compte des objets mobiles. De plus, en cas de changement de configuration de l'environnement (disparition de repères depuis la création de la carte : bâtiments détruits / construits, voitures garées à des endroits différents ...), celles-ci deviennent totalement obsolètes. Pour une autonomie complète, le robot doit pouvoir se localiser sans utiliser de carte fournie préalablement.

Le SLAM (*Simultaneous Localization And Mapping*), introduit par Cheeseman et Smith [SC86], est la technique la plus utilisée actuellement pour la construction de robots autonomes. Le robot crée une carte de l'environnement et, simultanément, se localise grâce à celle-ci. La carte ainsi que la position du robot sont totalement inconnues au départ. En particulier le robot ne possède aucune information sur le type de l'environnement (statique ou dynamique), ni sur sa topologie. La carte de l'environnement est créée à partir de ses observations. Celle-ci est en réalité une collection d'amers dont il estime la position à partir de sa propre position et des données acquises par ses capteurs. Ces mêmes amers, utilisés conjointement avec l'odométrie, permettent d'estimer la position du robot. Déterminer la position du robot demande par conséquent une bonne connaissance de l'environnement, qui elle même nécessite une connaissance précise de la position du véhicule. Les estimations sont donc dépendantes les unes des autres, c'est un problème du type *l'œuf ou la poule*. Le SLAM permet d'estimer simultanément les positions du robot et des repères.

Les premiers algorithmes de SLAM [SC86] ne prennent en compte que les environnements statiques. Toutes les observations du robot sont des amers, il n'y a aucun objet mobile pouvant perturber ces observations. Cependant cette hypothèse très restrictive rend le robot inexploitable hors de quelques milieux précis (par exemple certains robots d'usine). La mobilité en milieu ouvert (milieu comportant des objets dynamiques) leur est impossible. Des techniques ultérieures ont permis la prise en compte des objets mobiles comme le DTMO (*Detection and Tracking of moving Objets*) [WT02, WTT03, MMM05]. Ces méthodes filtrent les données considérées mobiles pour ne modéliser que la partie statique de l'environnement. D'autres maintiennent deux cartes, une pour la partie statique et une pour la partie dynamique [WS].

Tous ces algorithmes supposent que les *balises* demeurent statiques. L'aspect temporel de la reconnaissance est négligé. Prenons par exemple le cas d'un robot faisant le tour d'un parking (figure 1.1) ; il observe lors de son premier tour quatre voitures A, B, C et D qui deviennent des amers. L'incertitude de l'odométrie ne lui permet pas de se localiser précisément ; sa position est quelque part dans l'ellipse rouge. Il observe A, B, C et D et obtient les distances le séparant de ces balises. Il peut alors en déduire que sa position est en (1).

Il effectue ensuite une seconde boucle pendant laquelle la voiture C disparaît (figure 1.2). Les distances entre les véhicules $d(A, B)$ et $d(B, D)$ sont les mêmes. Il ne peut donc savoir s'il observe les voitures A et B ou bien les voitures B et D. L'incertitude de l'odométrie ne lui permet pas de choisir entre les deux positions. S'il se trompe (dans cet exemple s'il pense être en (1)) il peut insérer dans la carte des amers qu'il a déjà observés. S'il effectue un troisième tour, D pourra être considéré comme un nouvel amer et sera donc rajouté une nouvelle fois dans la carte. Celle-ci devient complétement fausse et la localisation impossible.

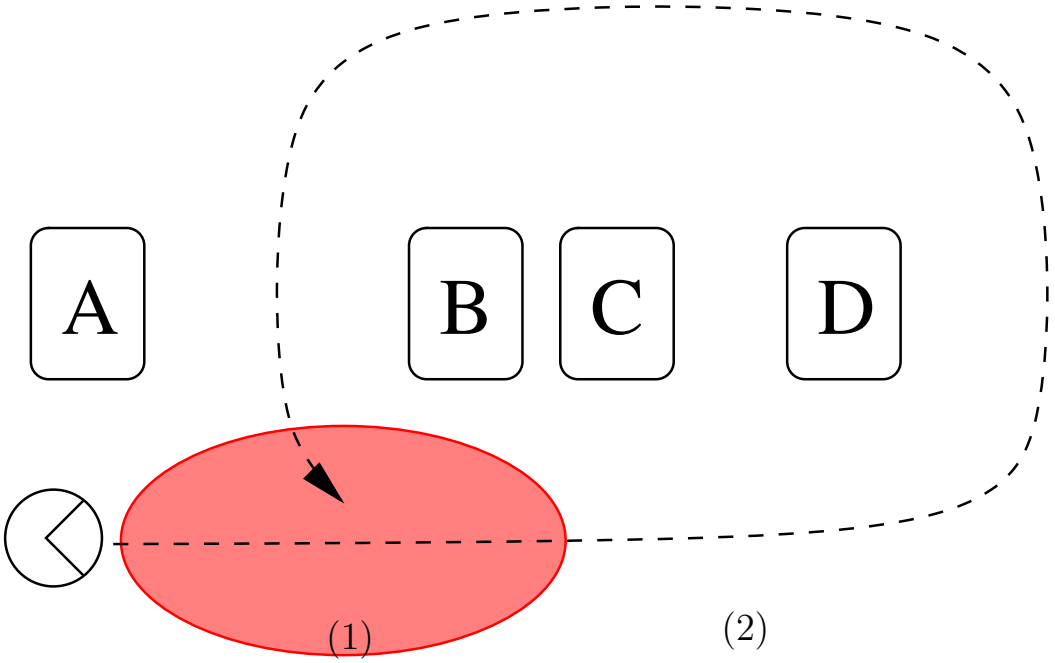


FIG. 1.1 – Problème des environnements semi-statiques : le robot effectue un premier tour du parking. Les voitures A, B, C et D deviennent des balises.

1.2 Objectifs

Les environnements statiques ne contiennent que des objets totalement immobiles au cours du temps. Les environnements dynamiques, quant à eux, comprennent des objets toujours en mouvement ; en particulier ils sont mobiles dans le champ du capteur, ce qui permet au robot de les détecter. Enfin, les environnements semi-statiques comprennent deux types d'objets :

1. ceux dont la vitesse de déplacement les rend immobiles pour les capteurs ;
2. ceux qui ne sont pas toujours mobiles au cours du temps.

Un robot autonome doit pouvoir appréhender de tels environnements ; il doit être en mesure de reconnaître les évolutions accomplies depuis sa dernière observation. Nous supposons que ces modifications n'interdisent pas toute correspondance avec les cartes précédentes (cas où tous les repères ont changé de position et où la comparaison devient impossible). Un tel système doit permettre au robot de lever un maximum d'ambigüités. Il doit aussi permettre une utilisation en milieu extérieur, où les amers sont peu nombreux en comparaison des milieux intérieurs. Cela augmente les risques de confusion puisque les comparaisons ne se font que sur un nombre réduit de repères.

1.3 Contribution

Le problème des environnements semi-statiques a été assez peu traité à l'heure actuelle. Stachniss et Burgard [SB05] ont proposé de créer un cluster de configurations possibles de l'environnement. Un tel apprentissage est impossible dans les milieux extérieurs. Biber et Duckett [PB05] utilisent le principe d'oubli : tout changement effectué dans une carte

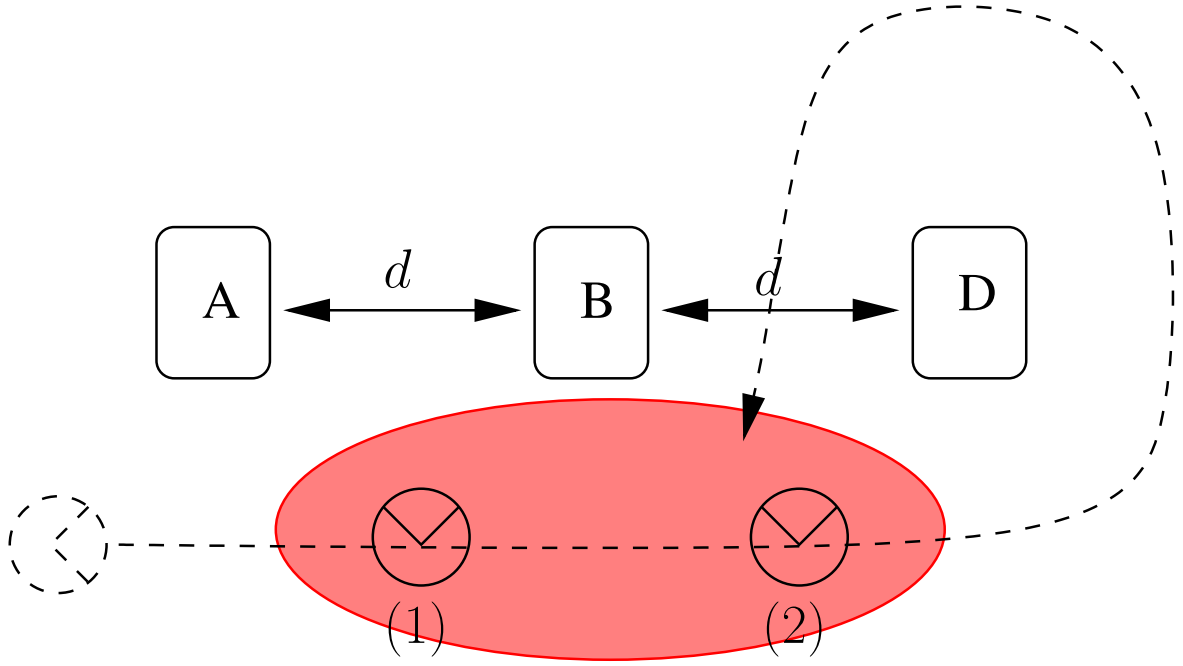


FIG. 1.2 – Problème des environnements semi-statiques : au cours de son deuxième tour la voiture C a changé de place. L'observation des voitures peut correspondre à la configuration (A, B) ou (B, D)

restée identique longtemps n'aura que peu d'importance au début. L'ancienne configuration est oubliée petit à petit. Si ce changement perdure, la nouvelle configuration finit par remplacer l'ancienne ; sinon le changement est considéré comme passager et la carte reste identique. L'inconvénient de cette méthode est l'absence de polyvalence causée par l'utilisation d'un paramètre différent pour chaque objet.

Nous avons traité le problème d'analyse des repères : pour une carte de l'environnement (c'est-à-dire une carte des amers) donnée, il faut pouvoir déterminer si elle correspond à une zone non encore visitée (les repères sont observés pour la première fois), ou à une configuration recontrée précédemment ou "presque" (les imprécisions des capteurs interdisant la certitude absolue). Nous avons représenté les amers par des groupes de "patches". Pour cela la carte prend la forme d'une grille d'occupation : chaque case de cette grille comporte la probabilité d'être occupée. Cela permet de prendre en compte les incertitudes liées au capteur. Les cases de probabilité élevée sont regroupées en patches. Ceux-ci sont eux-mêmes regroupés dans une structure hiérarchique représentant la distribution des positions des patches.

La structure permet une analyse à plusieurs niveaux de la carte. Imaginons un bâtiment rectangulaire. Le robot va percevoir les murs par fragments : ce sont les patches. Les patches sont regoupés selon un niveau hiérarchique (quatre, un pour chaque mur). Ces murs sont regroués en un nouveau niveau de hérarchie. Ce niveau (le plus élevé dans le hiérarchie) représente donc le bâtiment. Chacune des sous-hiérarchies représente un mur. La hiérarchie est donc un arbre dont les feuilles sont des groupes de patches.

Pour déterminer si les observations courantes du robot correspondent à une configuration déjà observée, on compare les hiérarchies courantes et passées. Pour cela on effectue un parcours des hiérarchies en profondeur. Les hiérarchies "modifiées" (ceux dont une

partie n'est pas détectée comme précédemment -typiquement une voiture garée dans un sens différent) ne sont pas pris en compte pour la localisation. La localisation ne se fera quà partir des niveaux de hiérarchie considérés statiques. Cela garantira une meilleure localisation puisque les comparaisons ne se font qu'entre hiérarchies *a priori* identiques.

1.4 Plan de lecture

Chapitre 2 : nous présenterons dans ce chapitre le fonctionnement général du SLAM et les principales implémentations existantes. Nous décriverons aussi les travaux réalisés sur les environnements semi-statiques.

Chapitre 3 : dans ce chapitre nous décrivons un algorithme particulier du SLAM appelé *FastSLAM*, pour lequel un filtre à particules est utilisé.

Chapitre 4 : nous expliquerons dans ce chapitre la représentation de la carte sous forme d'une grille d'occupation. Nous détaillerons aussi la modélisation des repères sous forme de patches et la structure hiérarchique dont ils dépendent.

Chapitre 5 : est le chapitre consacré aux analyses de nos résultats.

Chapitre 2

Vue d'ensemble du système

Notre système se décompose en trois parties principales :

1. l'algorithme de SLAM employé : nous avons utilisé l'algorithme FastSLAM de Georgio Grisetti, Cyrill Stachniss et Wolfram Burgard [1] présenté à la conférence ICRA [2] 2005 [3] ;
2. création des patches : parmi les impacts détectés par le capteur du robot ceux dont la probabilité est élevée vont former les embryons de patches. Les voisins de ces points sont examinés ; ceux de probabilité d'occupation élevée sont ajoutés au patch. On recommence l'opération jusqu'à ce que les points aient une probabilité trop faible pour être inclus dans le patch. Ces patches sont ensuite regoupés sous forme hiérarchique, ce qui permet dávoir une vision de la carte à plusieurs niveaux et donc de traiter les informations plus rapidement.
3. création de la structure hiérarchique, puis comparaison des informations reçues au temps t avec les informations précédentes. Il s'agit donc de comparer les cartes d'amers actuels avec les anciennes afin de se repérer. Cela consiste en fait à comparer les structure hiérarchique au temps t avec les anciennes structures. Cette partie porte le nom de *scanmatching* : elle détermine les correspondances de l'environnement actuel avec les environnements observés auparavant.

Le schéma 2.1 présente les relations existant entre les différentes parties de notre système. Les entrées sont les données odométriques et du capteur (un laser dans notre cas). Le scanmatching établit les correspondances entre les données actuelles du laser et les précédentes obtenues par le SLAM. La sortie du système est un couple (carte, liste de patches) : pour chaque carte créée on détermine la liste des patches correspondants.

[1] `http://www.informatik.uni-freiburg.de/~stachnis/research/rbpfmapper/`
[2] (*International Conference on Robotics and Automation*)
[3] `http://www.icra2005.org/frontal/Presentation.asp`

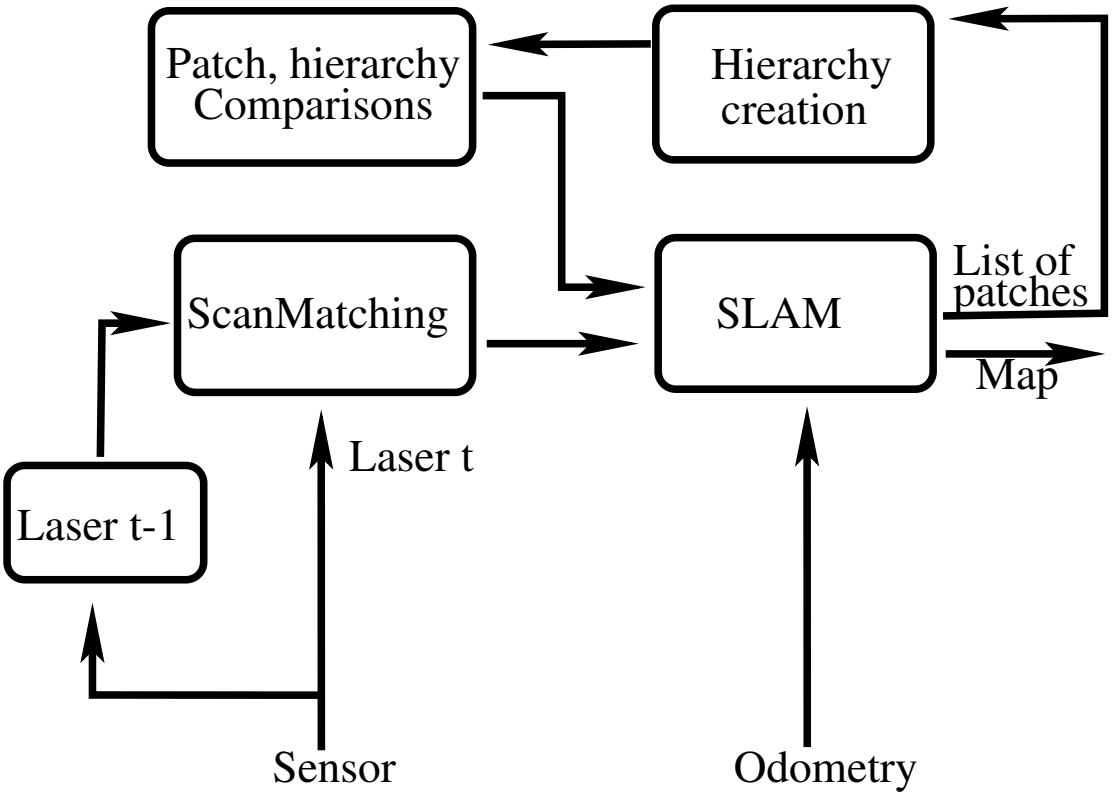


FIG. 2.1 – Vue d'ensemble du système

Chapitre 3

État de l'art

3.1 Utilisation de probabilités

Les mesures effectuées par les capteurs (odométrie, laser, sonar...) sont entâchées d'imprécisions liées à la technologie même du senseur ou de la manière avec laquelle elles sont prises. Il est ainsi impossible de déterminer la position exacte du robot, on ne peut que donner une zone plus ou moins grande où on pense que se situe le véhicule. On choisit de modéliser ces incertitudes *via* un modèle probabiliste.

3.2 Le problème du SLAM

Le problème du SLAM a été présenté pour la première fois dans [SC86] ; il consiste en la modélisation de l'environnement et la détermination de la pose du robot (c'est-à-dire sa position et orientation) à partir de données qu'il acquiert au fur et à mesure de son déplacement. Ces opérations sont effectuées en même temps. La carte est totalement inconnue au départ (en particulier, aucune information de type topologique n'est fournie). Le robot collecte des informations sur des balises environnantes et à partir de ces informations décide de sa trajectoire. Toutefois, les mesures concernant les repères et la pose du robot sont soumises à des imprécisions (imprécision des capteurs, roues qui glissent).

La figure 3.1 illustre le problème du SLAM graphiquement. Elle montre que l'incertitude de la trajectoire du robot augmente avec la distance parcourue, tout comme l'incertitude sur les caractéristiques de la carte. On peut remarquer que les incertitudes augmentent relativement lentement lors de déplacements rectilignes. Cependant des trajectoires courbes désorientent le robot de manière importante. La situation est analogue pour une personne qui traverserait les yeux bandés un couloir puis tournerait ensuite plusieurs fois sur sa droite ou sa gauche : en marchant droit devant elle elle sait à peu près où elle se situe. Une fois qu'elle a tourné, se localiser devient beaucoup plus difficile pour elle.

Une particularité du SLAM est montrée dans la figure 3.2 : le robot perçoit un repère observé auparavant, dont la position est relativement bien connue. Cette observation fournit au robot des informations sur sa pose. Cela augmente aussi ses connaissances concernant les autres repères de la carte, ce qui se traduit par une diminution de l'incertitude sur leurs positions et celle du robot (ellipses hachurées). Remarquons que cela pourrait permettre d'améliorer les estimations des poses précédentes ; toutefois le SLAM ne considère que la pose précédant immédiatement la pose actuelle afin de rendre la quantité de

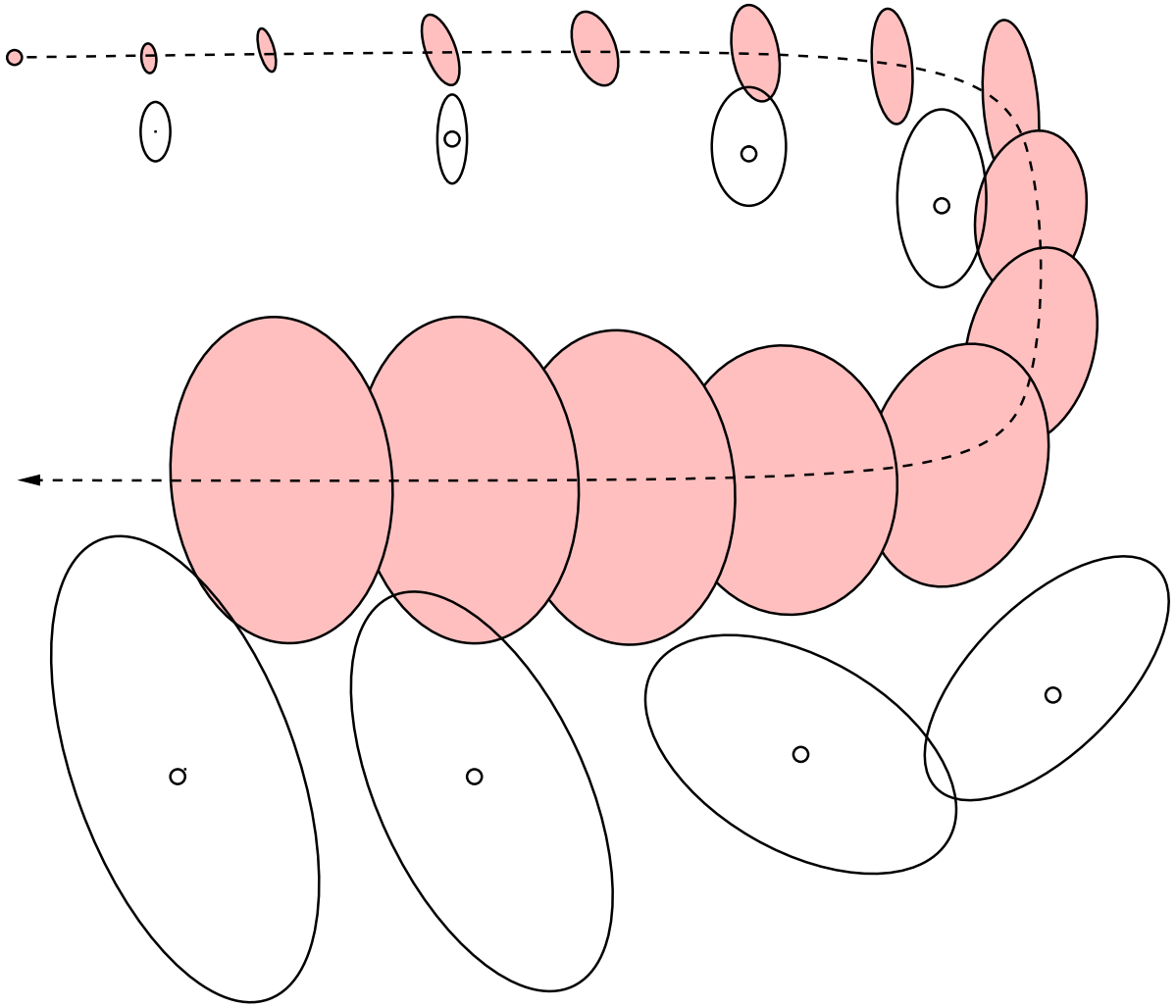


FIG. 3.1 – Le problème du SLAM : un robot se déplace dans un environnement inconnu ; plus il progresse, plus l'incertitude sur sa pose augmente, comme l'indiquent les ellipses colorées le long de sa trajectoire. L'incertitude concernant les maers (ellipses vides où le cercle représente le repère) se comporte de manière identique.

calculs indépendante de la longueur du chemin effectué.

3.2.1 Description formelle du SLAM

On note la carte de l'environnement Θ. Cette carte est en fait une collection d'maers - ou caractéristques - notés chacun θ_n. Le nombre total de caractéristiques est N. La pose du robot est définie par s_t, où t est l'index du temps discret. Les poses du robot se déplaçant dans le plan comprennent la position, exprimée en coordonnées cartésiennes, ainsi que l'orientation. Ce vecteur d'état est noté (x, y, θ). La séquence $s^t = s_1, s_2, \ldots, s_t$ est la séquence de poses prises par le robot ; cela correspond donc au trajet effectué du temps initial au temps t. D'une manière générale, les indices en exposant dénotent une séquence du temps $T = 1$ au temps $T = t$.

Pour construire la carte le robot utilise des capteurs (ultrason, laser, odomètre...), qui lui apportent des renseignements comme les distances ou les couleurs des balises, son

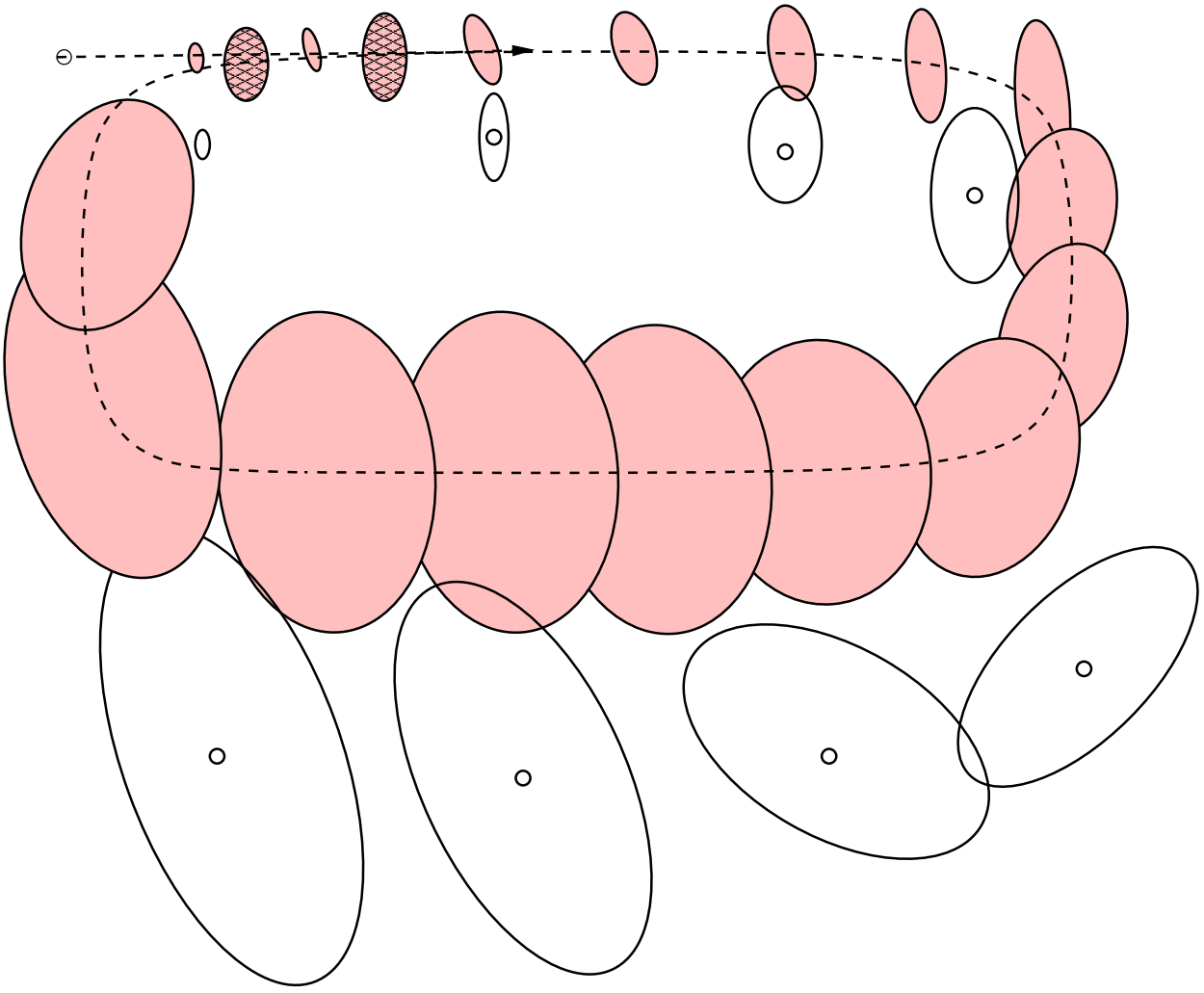


FIG. 3.2 – Une propriété du SLAM : l'observation d'amers perçus précédemment permet de diminuer l'imprécision sur la pose actuelle et sur les positions des amers.

orientation...

Nous supposons que le robot n'observe qu'un repère à la fois, ceci dans le but de simplifier les calculs. Cela n'engendre aucune perte de généralité. L'observation au temps t, notée z_t, est dans notre cas le couple (distance, direction par rapport au robot). Dans un premier temps nous supposerons que le robot est capable d'identifier le repère qu'il observe ; cette contrainte sera levée plus tard. Pour chaque observation z_t, n_t indique l'identité du repère observé ; n_t prend donc sa valeur dans l'ensemble $\{1, \ldots, N\}$.

Un modèle de mesures des capteurs, de type probabiliste, définit la façon dont les mesures sont générées. Ce modèle, nommé *modèle de mesure* est de la forme :

$$p(z_t \mid s_t, \theta_{n_t}, n_t) = g(\theta_{n_t}, s_t) + \varepsilon_t \tag{3.1}$$

L'observation z_t au temps t dépend de la position du robot s_t et de l'identité n_t du repère θ_{n_t}. Les mesures n'étant pas exactes la relation $p(z_t \mid s_t, \theta_{n_t}, n_t)$ est donc représentée par une fonction g bruitée de bruit ε. L'erreur de mesure est modélisée par une variable aléatoire ε_t ; c'est une Gaussienne de moyenne 0 et de covariance R_t.

La fonction g n'est en général pas linéaire. Par exemple si g est la mesure (distance,

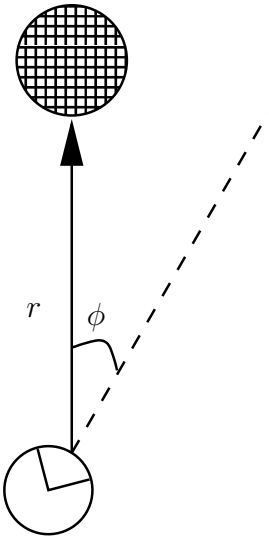


FIG. 3.3 – Robot observant un repère se trouvant à une distance r et un angle ϕ

direction) décrite plus haut, elle ne l'est pas, puisque le calcul de la direction fait intervenir des fonctions trigonométriques.

Une autre source d'information pour la résolution du SLAM est l'ensemble des actions effectuées par le robot. Ces actions sont notées u_t et correspondent à l'ensemble des commandes motrices exécutées durant l'intervalle $[t-1, t)$. La loi de probabilité gouvernant l'évolution des poses est appelé *modèle de mouvements cinématiques* ; elle est de la forme

$$p(s_t \mid u_t, s_{t-1}) = h(s_{t-1}, u_t) + \delta_t \tag{3.2}$$

La pose s_t dépend donc de la position précédente s_{t-1} et de la commande exécutée pendant l'intervalle $[t-1, t)$. C'est une fonction h de la pose du robot un pas de temps plus tôt. Le bruit δ_t est de nouveau modélisé par une Gaussienne de moyenne 0 et de covariance P_t. De même que pour le modèle de mesure, la fontion h n'est pas toujours linéaire.

L'un des buts du SLAM est de reconstruire la carte à partir des observations z_t et des contrôles u_t. La plupart des algorithmes de SLAM sont basés sur les filtres bayésiens ; la distribution de probabilité sur la carte Θ et la pose s_t est donnée par :

$$p(s_t, \Theta \mid z^t, u^t, n^t) \tag{3.3}$$

Si cette probabilité est calculée récursivement on dit que l'algorithme est un *filtre*. La plupart des algorithmes du SLAM sont des instances de filtre bayésien, qui calculent l'état suivant à partir de l'état un pas plus tôt :

$$p(s_t, \Theta \mid z^t, u^t, n^t) = \eta \times p(z_t \mid s_t, \theta_{n_t}, n_t) \int p(s_t \mid s_{t-1}, u_t)\; p(s_{t-1}, \Theta \mid z^{t-1}, u^{t-1}, n^{t-1}) ds_{t-1} \tag{3.4}$$

Ici η est une constante de normalisation (qui vaut $p(z_t \mid z^{t-1}, u^{t-1}, n^{t-1})$). Elle ne dépend d'aucune variable intervenant dans le calcul de l'état suivant s_t. Dans la suite,

nous écrirons de manière générale η comme constante de normalisation, sans chercher forcément à connaître sa valeur exacte.

Le filtre bayésien est au cœur de nombreux algorithmes contemporains. Dans le cas où g et h sont linéaires, l'équation est équivalente à un filtre de Kalman. Les filtres de Kalman étendus (EKF pour *Extended Kalman filters*) permettent de manipuler des fonctions g et h non linéaires, en les approximant par des fonctions linéaires obtenues par un développement de Taylor au premier ordre.

Au premier abord, on pourrait considérer que l'équation 3.4 donne toutes les informations nécessaires. Les premiers algorithmes de SLAM utilisaient cette relation : une matrice représente toute la carte (amers, pose du robot) ; les éléments hors diagonale indiquent les relations entre les erreurs de la pose du robot et les locations des amers. Cependant le coût de cette représentation est en $O(N^2)$, où N est le nombre de repères de la carte ; cela ne permet pas de manipuler de grandes quantité d'amers. Des recherches ont été faites pour diminuer cette complexité, par exemple en divisant la carte en sous-cartes [TKG+02, Bai01]. Le nombre de repères est alors assez faible pour que les calculs soient exécutés en un temps moindre. Un autre inconvénient de l'utilisation d'EKF est la possible divergence de l'algorithme si les associations de balises ne sont pas justes. Ces associations de données erronnées, connues sous le nom de *correspondance problem*, surviennent lorsque certains amers de l'environnement se ressemblent. Une mauvaise association peut aboutir à créer une carte totalement fausse. Pour remédier cela des techniques permettant d'annuler des associations précédentes ont été proposées, comme les algorithmes RANSAC [FB87], Expectation-Maximization [SK97]. Malheureusement ces techniques ne sont pas exécutables en temps réel.

Les algorithmes fondés sur les filtres de Kalman sont moins utilisés de nos jours mais restent les implémentations de référence auxquelles les nouveaux algorithmes sont comparés.

3.2.2 Approches par EKF

Utilisés dans les premières versions du SLAM [SC86, SC86], les algorithmes avec EKF représentent la carte par une Gaussienne multimodale de haute dimension : les moyennes représentent les positions des amers et du robot, les covariances les relations entre les positions de tous ces objets. Les EKF sont une extension du filtre de Kalman [Kal60] s'appliquant aux systèmes non linéaires. Ils remplacent les fonctions non linéaires par leur version linéarisée, grâce à une développement de Taylor.

Le principal inconvénient de ces méthodes est le coût quadratique dû à l'útilisation d'une matrice pour la Gaussienne. Tout changement concernant les informations de la carte nécessite une mise à jour complète de la matrice. Le nombre de balises que peuvent gérer ces algorithmes reste donc limité. De plus les EKF ne maintiennent qu'une hypothèse pour chaque position de repère. L'association balise-observation se fait par *maximum likelihood* ; si la probabilité d'une observation est trop faible le robot considère avoir observé un nouveau repère. Une mauvaise association remet en cause la convergence de l'algorithme, puisqu'il est impossible de revenir sur la décision.

3.2.3 Approches par EM

Une alternative aux filtres de Kalman est l'algorithme *expectation maximization (EM)* [Thr01, TFB98]. L'algorithme EM est de type statistique ; il a été présenté la première

fois par [DLR77].

L'algorithme comprend deux parties répétées alternativement :

1. **étape Expectation** calcule les estimées sur les poses du robot.
2. **étape Maximization** détermine par maximum likelihood la carte la plus vraissemblable d'après la pose définie par l'étape Expectation.

L'approche EM est l'une des meilleures techniques pour résoudre le problème du SLAM dans les environnements cycliques, même si les repères se ressemblent et ne peuvent être distingués que difficilement.

Lors de ses itérations, cet algorithme affine les cartes. En fait il effectue une recherche dans l'espace des cartes (qui sont donc traitées plusieurs fois). La création de cartes ne se fait pas incrémentalement ce qui rend l'utilisation de cet algorithme impossible pour les applications temps réel.

Plusieurs solutions alternatives ont tenté démuler l'approche EM en diminuant la complexité. Parmi elles, *incremental maximum likelihood method* qu'on trouve dans [Thr93, YB96]. Cela correspond à l'algorithme EM sans l'étape E. Cette méthode construit incrémentalement une carte grâce aux observations sans toutefois garder en mémoire l'incertitude résiduel. Cependant ce manque ne permet plus de gérer les environnements cycliques correctement.

3.2.4 Sous-cartes

Comme nous l'avons vu en 3.2.2, le principal inconvénient de l'approche par EKF est le coût quadratique de la mise à jour de la carte. Cela rend son utilisation impossible dans les milieux comprenant un nombre important d'amers, comme les milieux extérieurs. Partant de l'hypothèse que l'observation d'un amer n'influence en rien les observations des amers éloignés, certaines recherches ont porté sur la décomposition de la carte en sous-cartes, comme [KDR01]. Une sous-carte est une partie de la carte globale. Le robot ne met alors à jour la sous-carte dans laquelle il se trouve que lorsqu'il la quitte (la mise à jour est effectuée lorsqu'il rentre dans une autre partie de la carte). D'autres approches comme [BNL+03] divisent la carte de manière plus franche : les relations entre sous-cartes sont représentées par des graphes dont chaque nœud contient les informations de cette sous-carte. L'algorithm de Dijkstra permet de calculer les incertitudes entre deux sous-cartes (ces incertitudes sont contenues dans les arêtes du graphe).

3.2.5 Les algorithmes à base de filtres de particules

Nous détaillons le principe de ces algorithmes en 4.1.

3.3 Environnements semi-statiques

Les approches sur les environnements mobiles [MMM05] considèrent mobile tout ce qui bouge de manière continue. La plupart de ces approches filtrent ces objets mobiles : une carte statique de l'environnement est créée, dans laquelle les objets mobiles sont supprimés. D'autres utilisent deux cartes, une pour la partie statique et une pour la partie dynamique [WTT03, WT02]. Une analyse relativement simple des résultats obtenus par le capteur suffit pour déterminer quels sont les objets mobiles, puisque ces derniers bougent lors de l'acquisition des données : les mesures sur les distances robot-objets varient beaucoup

plus que pour les objets statiques. Notre intérêt se porte pour les objets dont on peut dire qu'ils sont mobiles, "mais pas tout le temps", et même "plutôt quand le capteur ne les observe pas".

Cette "branche" du SLAM n'est que peu développée ; les deux principales approches qu'on peut retenir sont celles de [PB05] et de [SB05]. La première se focalise sur la durée pendant laquelle les repères sont demeurés identiques ; la seconde utilise un cluster de sous-cartes pour déterminer toutes les configurations possibles que peut prendre l'environnement.

3.3.1 Apprentissage sur la durée

La première méthode part du principe que les changements ne sont pas permanents. Il est désirable que le robot "se souvienne" de l'état antérieur si ce changement n'est que temporaire. Ce problème concernant les systèmes apprenant sur un temps assez long est connu sous le nom de *stability-plasticity dilemna*. L'apprentissage "life-long" demande l'adaptation aux nouveaux états ainsi que la préservation des anciens.

Cette approche ne se préoccupe pas de la navigation autonome ni des changements topologiques mais de la localisation et de l'apprentissage de la carte lors d'un mouvement cyclique infini.

L'exemple qui sert de fil rouge à l'article [PB05] est la situation suivante : supposons un robot fixant un mur (figure 3.3.1). La "carte" à apprendre est simplement la distance robot-mur. Si l'environnement est statique l'apprentissage est simple (case de gauche uniquement) : toutes les valeurs autres que la véritable distance sont dues au bruit Gaussien des mesures ; la carte est parfaitement représentée par l'échantillon de la moyenne et de la variance.

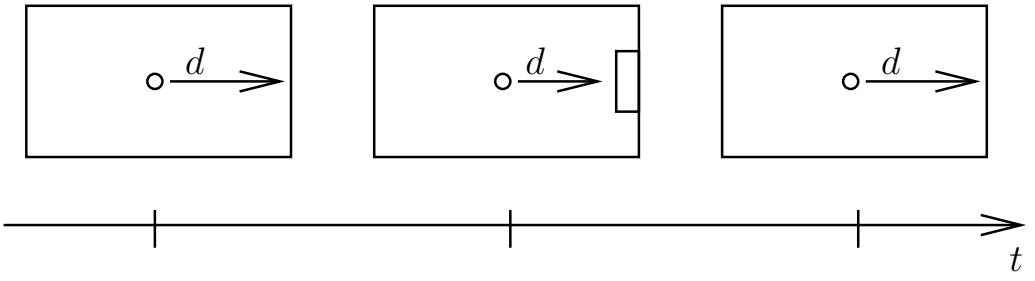


FIG. 3.4 – Le robot, placé n'importe où dans la pièce, fixe le mur. La donnée à apprendre est la distance robot-mur. La pièce est d'abord vide (cas statique), puis une armoire est placée devant le mur. Cette armoire est ensuite retirée (cas dynamique).

Ils ont établi trois contraintes :

1. le temps nécessaire pour adapter un changement à la carte ne doit pas dépendre du temps absolu passé. Si le robot a fixé le mur pendant des mois, l'ajout d'une armoire devant le mur ne doit pas être prise en compte au bout d'un temps égal. La loi de probabilité de changement dépend donc d'un paramètre temporel. Ce paramètre doit être choisi avec soin, ni trop grand ni trop petit : s'il est trop grand le changement n'aura jamais lieu. S'il est trop petit la moindre mesure différente des autres (par exemple une mauvaise mesure) sera considérée comme une preuve de changement.

2. la carte doit donc être robuste aux *outliers* (mesures erronnées) : une mauvaise mesure ne doit pas remettre en cause le processus d'apprentissage. C'est donc le paramètre temporel qui détermine si la mesure correspond ou non à un outlier.
3. la carte ne doit contenir que des valeurs observables réellement. Par exemple si d_1 et d_2 représentent respectivement les distances robot-mur et robot-armoire, moyenner l'ensemble des distances observées donnerait un résultat compris strictement entre d_1 et d_2. Or de telles valeurs ne sont pas possibles réellement.

Le principal problème de cette méthode est l'utilisation de paramètres temporels : il faut les déterminer de façon à ce qu'il ne provoquent pas la détection de chagements qui n'existent pas ou qu'ils crï¿½nt de trop nombreux outliers. Notre volonté étant de pouvoir construire un système se déplaçant aussi bien en milieu intérieur qu'en milieu extérieur, nous avons écarté cette technique, puisqu'elle demanderait de déterminer sans cesse de nouveaux paramètres (un pour tout nouvel amer). De plus l'apprentissage se fait sur un nombre important d'observations ; en milieu extérieur une telle contrainte n'est pas applicable.

3.3.2 Apprentissage de configurations de l'environnement

L'exécution du SLAM en environnement intérieur prend souvent pour balise les coins et les portes. Ces dernières peuvent prendre deux états (ouvert ou fermé) qui fournissent des informations précieuses pour la localisation du robot. Ces renseignements peuvent donc être intégrés dans la carte d'environnement.

Dans l'article *Mobile Robot Mapping and Localization in Non-Static Environments* [SB05] Stachniss et Burgard proposent d'utiliser les informations sur les changements subis par l'environnement, informations obtenues pendant la phase d'acquisition des données. Cela permet d'estimer les configurations spatiales que peut prendre l'environnement. Ces configurations sont stockées dans le modèle de la carte. Pour cela, la carte est divisée en sous-cartes, une pour chaque aire où des comportements dynamiques ont été observés. Des clusters de ces sous-cartes sont ensuite appris pour déterminer les configurations possibles de l'environnement.

Utiliser une carte pour l'environnement complet est impossible à cause de la complexité qui en découlerait : le nombre de cartes de configurations possibles serait exponentiel en le nombre d'objets dynamiques. Toutefois en utilisant l'indépendance de certains objets vis-à-vis des autres, on peut réduire cette complexité en divisant la carte en sous-cartes.

L'hypothèse est faite que le robot observe les mêmes aires à des périodes différentes afin d'apprendre les différents états possibles de l'environnement. Les observations permettent de créer des cartes locales qui sont groupées en clusters pour extraire de possibles configurations. Les données sont ainsi segmentées en séquences d'observations $\{\phi_1, \ldots, \phi_n\} = \Phi$. $\phi_i = \{z_{initial}, \ldots, z_{final}\}$ est un ensemble d'obervations. Lorsque le robot quitte une carte locale, la séquence en cours se termine et une nouvelle débute. De plus une nouvelle séquence est automatiquement mise en route après un certain temps passé dans la même aire.

Les grilles sont ensuite transformées en vecteur de probabilité. Ces vecteurs sont ensuite groupés en clusters grâce à la méthode des *k-means*.

À un instant t, on ne doit alors plus seulement évaluer la pose s_t mais aussi la carte m_t :

$$p(s_t, m_t \mid z^t, u^{t-1}) = \eta p(z_t \mid s_t, m_t, z^{t-1}, u^{t-1}) p(s_t, m_t \mid z_{t-1}, u^{t-1}) \tag{3.5}$$

Nous voyons deux défauts majeurs à cette technique : la première est qu'il est impossible d'apprendre autant de configurations possibles en environnement extérieur qu'en milieu intérieur. Comment en effet pouvoir observer toutes les positions possibles de véhicules garés ? Même en se restreignant à un nombre limité d'observations, cette méthode nécessite une certaine "base" pour pouvoir comparer les clusters. En d'autres termes, l'utilisation de repères comme les murs, les coins ... permet de n'avoir à comparer qu'une faible partie des informations données par le capteur. Cela rend cette approche inappropriée dans les milieux extérieurs, où les amers sont moins nombreux et souvent plus éloignés du robot, ce qui augmente les imprécisions odométriques et sensorielles.

Chapitre 4

Le SLAM par filtre à particules, FastSLAM

4.1 Principe du l'algorithme utilisé

Le FastSLAM [MTKW02, MT] est une instance du filtre de Rao-Blackwellized [DdFMR] proposé en 1999. Le FastSLAM utilise un filtre de particules pour déterminer la pose du robot et son environnement. Puisque les mesures des capteurs sont soumises à des imprécisions plus ou moins importantes, il est impossible de déterminer avec certitude la position exacte du robot. L'algorithme manipule donc un ensemble de particules représentant les poses possibles du véhicule. Cela permet de pouvoir toujours considérer plusieurs hypothèses quant à la position du robot. Chaque particule possède sa propre carte de l'environnement. Ces cartes sont comparées aux observations effectuées par les capteurs ; un poids est attribué aux particules selon le degré d'exactitude de la carte. Les particules les plus faibles sont supprimées de l'ensemble de particules, celles avec le poids le plus élevé sont échantillonnées afin de ne pas trop réduire l'ensemble des poses possibles.

4.2 Aspect mathématique

Reprenons l'équation 3.3 régissant le SLAM :

$$p(s_t, \Theta \mid z^t, u^t, n^t) \tag{4.1}$$

L'estimée porte sur la pose du robot *à l'instant* t. Les filtres à particules estiment quant à eux l'ensemble des chemins possibles ainsi que la carte Θ :

$$p(s^t, \Theta \mid z^t, u^t, n^t) \tag{4.2}$$

On peut se poser la question d'un tel choix : pourquoi estimer le chemin complet ? Plus le chemin grandit, plus l'espace sur lequel 4.2 est défini grandit. Une telle méthode semble contradictoire avec la volonté d'avoir un algorithme exécutable en temps réel. Cependant nous verrons plus loin qu'il existe des filtres capables de calculer les estimées d'un chemin aussi efficacement que s'il suffisait de calculer la pose suivante. Toutefois cet argument n'est pas le seul qui explique la multiplication des algorithmes fondés sur les filtres à particules. La véritable motivation est que l'équation 4.2 peut être décomposée en un produit de termes plus petits.

Déterminons le filtre de l'équation 4.2 :

$$p(s^t, \Theta \mid z^t, u^t, n^t) = \eta \times p(z_t \mid s_t, \theta_{n_t}, n_t) p(s_t \mid s_{t-1}, u_t)\ p(s^{t-1}, \Theta \mid z^{t-1}, u^{t-1}, n^{t-1}) \quad (4.3)$$

Cette équation de mise à jour diffère de l'équation 3.4 par l'absence d'intégrale ; en particulier, la pose au temps $t-1$, s_{t-1}, n'est pas intégrée. Les temps de calculs sont par conséquent grandement diminués.

Par la règle de Bayes on obtient :

$$p(s^t, \Theta \mid z^t, u^t, n^t) = \eta \times p(z_t \mid s^t, \Theta, n_t, z^{t-1}, u^t)\ p(s^t, \Theta \mid n^t, z^{t-1}, u^t) \quad (4.4)$$

Utilisons les relations d'indépendance pour simplifier cette relation : dans la première estimation, l'observation z_t ne dépend que de la pose s_t, l'identité n_t et la location θ_{n_t} du repère observé. On obtient alors :

$$p(z_t \mid s^t, \Theta, u^t, n^t, z^{t-1}) = p(z_t \mid s_t, \theta_{n_t}, n_t) \quad (4.5)$$

Dans le second membre, la probabilité $p(s^t, \Theta \mid z^{t-1}, u^t, n^t)$ peut-être factorisée de la manière suivante :

$$p(s^t, \Theta \mid n^t, z^{t-1}, u^t) = p(s_t \mid s^{t-1}, \Theta, n^t, z^{t-1}, u^t)\ p(s^{t-1}, \Theta \mid n^t, z^{t-1}, u^t) \quad (4.6)$$

Les deux termes sont simplifiés en supprimant les variables qui n'ont aucune influence dans les calculs : l'observation d'un repère ne dépend que du repère et de la pose du robot, et non des actions qui ont amené le robot à cette position. De même, la connaissance de s_{t-1} et de u_t est suffisante pour déterminer s_t ; toutes les autres variables dans le premier terme de droite de l'équation 4.6 n'apportent pas d'information et peuvent donc être omises. De manière analogue, n_t et u_t n'apportent pas d'information sur l'état suivant de s^{t-1} et Θ. On peut alors récrire 4.6 ainsi :

$$p(s^t, \Theta \mid n^t, z^{t-1}, u^t) = p(s_t \mid s^{t-1}, u_t)\ p(s^{t-1}, \Theta \mid n^{t-1}, z^{t-1}, u^{t-1}) \quad (4.7)$$

4.3 Factorisation de l'estimation du SLAM

La clé mathématique essentielle est la factorisation de 4.2 :

$$p(s^t, \Theta \mid n^t, z^t, u^t) = p(s^t \mid n^t, z^t, u^t) \prod_{n=1}^{N} p(\theta_n \mid s^t, n^t, z^t) \quad (4.8)$$

Cette caractéristique traduit le fait que l'observation d'un repère pour une pose donnée ne donne aucune information sur les positions des autres amers. Les estimées des repères sont donc indépendantes les unes des autres. Cette observation a été signalée pour la première fois dans [Mur] et exploitée par la suite dans [TMW02, ALTT02] et plusieurs papiers concernant le FastSLAM [MTKW02, WTT03, NGNT03]. Elle a aussi été utilisée dans des algorithmes précédents [TBF00] mais sans être explicite à ce moment.

Cette factorisation établit que le calcul sur l'estimée des chemins et de la carte peut être décomposé en $N+1$ estimations récursives : une estimation sur les chemins du robot $p(s^t, \mid n^t, s^t, u^t)$ et N estimations sur les locations des repères $p(\theta_n \mid s^t, n^t, z^t)$

(une pour chaque $n = 1, \ldots, N$). Ces dernières estimations dépendent de la trajectoire suivie. Le produit de ces probabilités représente l'estimée sous une forme factorisée. La représentation est exacte, ce n'est pas une approximation. C'est une propriété générique du problème du SLAM.

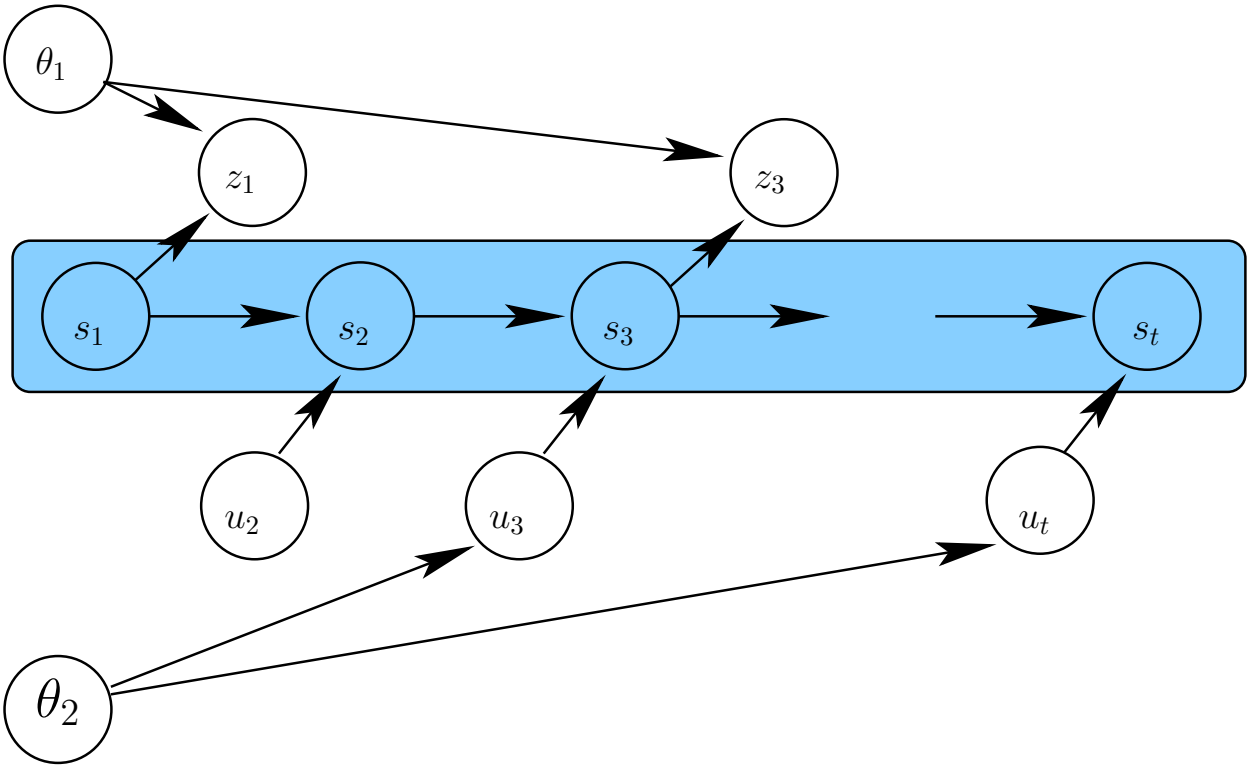


FIG. 4.1 – Représentation du problème du SLAM sous forme de réseau bayésien : le robot bouge de sa pose initiale s_1 grâce à des commandes $u_1, \ldots, u_t$. À $t = 1$, il observe θ_1 parmi deux repères $\{\theta_1, \theta_2\}$. L'observation est notée z_1. À $t = 2$, il observe θ_2 puis θ_1 au pas de temps suivant. Le SLAM estime les locations des repères et le chemin du robot à partir des observations z^t et des commandes u^t.

Pour illustrer la correction de cette factorisation, la figure 4.1 montre l'acquisition de données graphiquement, sous la forme d'un réseau Bayésien dynamique. Comme le suggère ce graphique, chaque mesure $z_1, \ldots, z_t$ est une fonction de la position du repère correspondant, avec la pose du robot aux temps où les mesures ont été effectuées. La connaissance du chemin que le robot emprunte "d-sépare" [Gha98] les estimations individuelles des repères et les rend indépendantes les unes des autres. La connaissance de la location exacte d'un repère ne nous apprendra donc rien sur le placement des autres repères.

Cette remarque peut être facilement dérivée mathématiquement. L'indépendance est donnée par le produit suivant :

$$p(\Theta \mid s^t, n^t, z^t) = \prod_{n=1}^{N} p(\theta_n \mid s^t, n^t, z^t) \tag{4.9}$$

Notez que toutes ces probabilités sont conditionnées par le chemin du robot s^t. La dérivation de 4.9 nécessite la distinction de deux cas possibles, selon que θ_n a été mesuré

en dernier ou non. En particulier, si $n_t \neq n$, la mesure la plus récente z_t n'a pas d'effet sur l'estimation, tout comme la pose du robot s_t n'a pas d'effet sur la correspondance n_t. On obtient donc :

$$p(\theta_n \mid s^t, n^t, z^t) = p(\theta_n \mid s^{t-1}, n^{t-1}, z^{t-1}) \tag{4.10}$$

Si $n_t = n$, c'est-à-dire si $\theta_n = \theta_{n_t}$ a été mesuré à la dernière mesure z_t, on peut utiliser la règle de Bayes et d'autres simplifications :

$$\begin{aligned} p(\theta_{n_t} \mid s^t, n^t, z^t) &= \frac{p(z_t \mid \theta_{n_t}, s^t, n^t, z^{t-1})\ p(\theta_{n_t} \mid s^t, n^t, z^{t-1})}{p(s_t \mid s^t, n^t, z^{t-1})} \\ &= \frac{p(z_t \mid \theta_{n_t}, s_t, n_t)\ p(\theta_{n_t} \mid s^{t-1}, n^{t-1}, z^{t-1})}{p(s_t \mid s^t, n^t, z^{t-1})} \end{aligned}$$

Cela nous donne l'expression suivante pour la probabilité $p(\theta_{n_t} \mid s^{t-1}, n^{t-1}, z^{t-1})$:

$$p(\theta_{n_t} \mid s^{t-1}, n^{t-1}, z^{t-1}) = \frac{p(\theta_{n_t} \mid s^t, n^t, z^t)\ p(z_t \mid s^t, n^t, z^{t-1})}{p(z_t \mid s^t, n^t, z^{t-1})} \tag{4.11}$$

La preuve de la correction de 4.9 se fait par induction. Supposons que l'estimée au temps $t-1$ a été factorisée :

$$p(\Theta \mid s^{t-1}, n^{t-1}, z^{t-1}) = \prod_{n=1}^{N} p(\theta_n \mid s^{t-1}, n^{t-1}, z^{t-1}) \tag{4.12}$$

Cette relation est vraie pour $t = 1$, puisqu'au départ le robot n'a aucune connaissance concernant les repères, par conséquent toutes les estimations sont indépendantes. Au temps t, l'estimée est de la forme :

$$\begin{aligned} p(\Theta \mid s^t, n^t, z^t) &= \frac{p(z_t \mid \Theta, s^t, n^t, z^{t-1}) p(\Theta \mid s^t, n^t, z^{t-1})}{p(z_t \mid s^t, n^t, z^{t-1})} \\ &= \frac{p(z_t \mid \theta_{n_t}, s_t, n_t) p(\Theta \mid s^{t-1}, n^{t-1}, z^{t-1})}{p(s_t \mid s^t, n^t, z^{t-1})} \end{aligned}$$

En insérant cette égalité dans l'hypothèse inductive 4.12 nous obtenons :

$$\begin{aligned} p(\Theta &\mid s^t, n^t, z^t) \\ &= \frac{p(z_t \mid \theta_{n_t}, s_t, n_t)}{p(z_t \mid s^t, n^t, z^{t-1})} \prod_{n=1}^{N} p(\theta_{n_t} \mid s^{t-1}, n^{t-1}, z^{t-1}) \\ &= \frac{p(z_t \mid \theta_{n_t}, s_t, n_t)}{p(z_t \mid s^t, n^t, z^{t-1})} \underbrace{p(\theta_{n_t} \mid s^{t-1}, n^{t-1}, z^{t-1})}_{eq.4.11} \prod_{n \neq n_t} \underbrace{p(\theta_{n_t} \mid s^{t-1}, n^{t-1}, z^{t-1})}_{eq.4.10} \\ &= p(\theta_{n_t} \mid s^t, n^t, z^t) \prod_{n \neq n_t} p(\theta_{n_t} \mid s^t, n^t, z^t) \\ &= \prod_{n=1}^{N} p(\theta_{n_t} \mid s^t, n^t, z^t) \end{aligned}$$

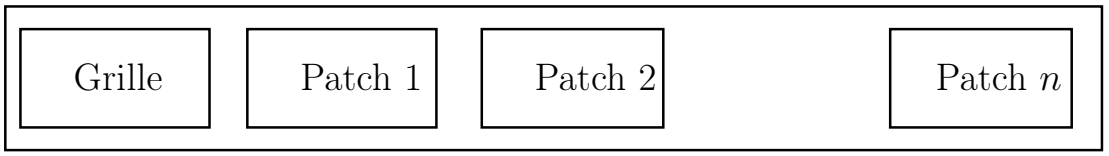


FIG. 4.2 – Structure d'une particule de notre algorithme

Notons que nous avons substitué les équations 4.10 et 4.11 comme indiqué. Cela montre la correction de l'équation 4.9. La correction de la forme principale 4.8 provient directement de ce résultat et de la transformation générique :

$$\begin{aligned} p(s^t, \Theta \mid n^t, z^t, u^t) &= p(s_t \mid n^t, z^t, u^t) p(\Theta \mid s^t, n^t, z^t, u^t) \\ &= p(s_t \mid n^t, z^t, u^t) p(\Theta \mid s^t, n^t, z^t) \\ &= p(s_t \mid n^t, z^t, u^t) \prod_{n=1}^{N} p(\theta_n \mid s^t, n^t, z^t) \end{aligned}$$

Il est donc nécessaire de conditionner selon le chemin global s^t pour obtenir ce résultat. La pose la plus récente s_t serait en effet insuffisante comme variable conditionnelle, puisque que les dépendances peuvent provenir de poses antérieures. Cette observation explique le choix de l'estimée sur la carte et les chemins 4.2, plutôt que sur des formes plus communes comme en 3.3. L'algorithme que nous avons choisi utilise le scanmatching pour bien placer les particules.

4.4 Modification de l'algorithme

L'algorithme que nous avons utilisé est celui de Giorgio Grisetti, Cyrill Stachniss et Zolfram Burgard [GSB05]. Il s'agit d'une amélioration de l'algorithme présenté par Hähnel, Burgard, Fox et Thrun [HFBT03]. Les particules possèdent une structure comme indiqué sur la figure 4.4 : chaque particule possède une carte de l'environnement ainsi qu'une liste de patches.

La figure 4.4 illustre la structure des M particules de l'algorithme de type FastSLAM que nous avons modifié. Chaque particule est de la forme :

$$S_t^{[m]} = \langle \text{Carte}, \text{Patch}_1, \text{Patch}_2, \ldots, \text{Patch}_n \rangle \tag{4.13}$$

La notation $[m]$ indique l'index de la particule : $S_t^{[m]}$ est la m^e particule de l'ensemble de particules à l'instant t. Le nombre de patches dépend de la particule. Nous détaillons la construction des patches en 5.2. La distribution de probabilité est de la forme :

$$p(s^t, \Theta, \mathcal{Q} \mid z^t, u^t) = p(s^t \mid n^t, z^t, u^t) p(\theta \mid s^t, z^t) \prod_n p(q_n \mid s^t, z^t) \tag{4.14}$$

où $\mathcal{Q}$ est la liste de patches de la carte. Nous faisons aussi l'hypothèse que les patches sont indépendants deux à deux, comme c'est le cas pour les amers de la carte avec le FastSLAM.

Mise à jour du filtre Filtrer, c'est-à-dire calculer l'estimée au temps t à partir de $t-1$ consiste à générer un nouvel ensemble de particules S_t à partir de S_{t-1}, ensemble de particules un pas de temps avant. Ce nouvel ensemble incorpore un nouveau contrôle u_t et une mesure z_t (de correspondance n_t). Cette mise à jour se fait à travers les étapes suivantes :

1. Échantillonnage
   La génération de particules S_t est obtenue à partir de S_{t-1} en échantillonnant selon une loi de distribution $\pi(s^t \mid z^t, u^t)$;
2. Calcul du poids des particules : chaque particule se voit assigner un poids $w^{[m]}$ selon la formule

$$w^{(m)} = \frac{p(s_t^m \mid z^t, u^t)}{\pi(s^t \mid z^t, u^t)} \tag{4.15}$$

   Le poids modélise le fait que la distribution proposée π n'est en général pas égale à la véritable distribution des états successeurs.
3. Rééchantillonnage
   Les particules avec un poids faibles sont supprimées de S_t. Elles sont remplacées par des particules de poids plus élevé. Cette étape est nécessaire puisque seul un nombre fini de particules est utilisé pour approximer une distribution continue. De plus rééchantillonner permet d'appliquer le filtre de particules dans des situations pour lesquelles la véritable distribution diffère de celle proposée.
4. Estimation de la carte et création des patches
   Pour chaque pose $s_t^{[m]}$, une estimation de la carte Θ_t^m est calculée selon la trajectoire et l'historique des observations $p(\Theta_t^m \mid s^{[m],t})$

Chapitre 5

Création des patches

5.1 Représentation de la carte

Le but de ce projet est de pouvoir se localiser alors que l'environnement a été modifié. Nous devons donc être en mesure de déterminer les repères qui ont disparu ou changé de position afin de ne prendre en compte que les repères demeurés statiques.

Nous avons donc représenté la carte de l'environnement sous la forme d'une grille d'occupation [Elf89]. Cette grille contient en chaque cellule la probabilité d'avoir un objet ou non. Plus la probabilité est élevée plus les chances que la case soit occupée sont importantes. De manière semblable, une faible probabilité indique que la case est certainement vide.

Les probabilités sont déterminées de la manière suivante :

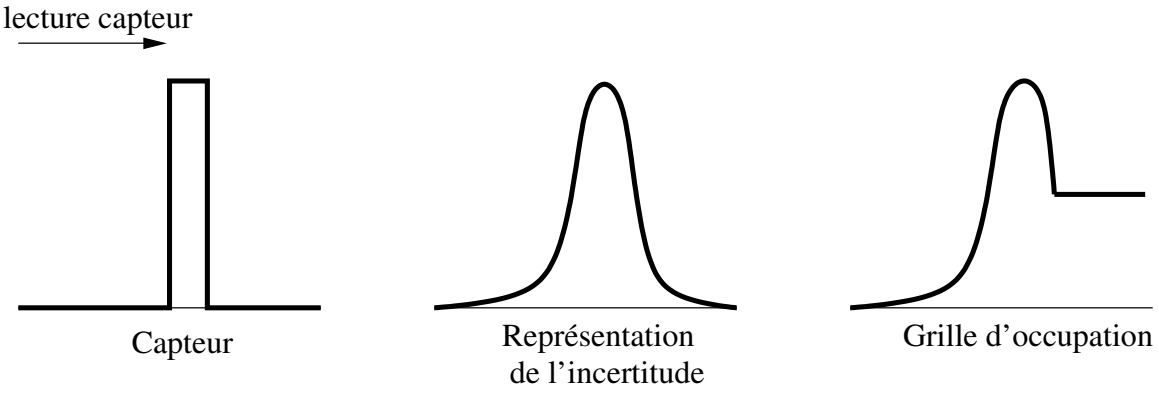


FIG. 5.1 – Probabilité d'occupation

Le capteur (un laser dans notre cas) indique la distance à laquelle se trouve l'objet. S'il est parfait on a le schéma de gauche : on passe d'une probabilité nulle à une probabilité de 1 de manière immédiate. Les capteurs souffrent toutefois d'imprécision. Représenter la position du repère sous forme de Gaussienne permet de modéliser ces incertitudes (schéma du milieu). La formule de la Gaussienne de moyenne d et de variance σ est :

$$P(d \mid z_t) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(d - z_t)^2}{2\sigma^2}\right) \tag{5.1}$$

d est la distance robot-objet renvoyée par le capteur,

La probabilité 0.5 représente l'inconnu. Les cellules de la grille sont donc initialisées avec cette valeur. Pendant le trajet du robot elle signifie que la zone n'a pas été traitée par le capteur. Typiquement ce sont les zones que le capteur n'a pas traitées (comme celles derrière un obstacle). C'est le schéma de droite.

Par conséquent, une cellule c_i peut avoir trois états : occupé, libre, inconnu. On note l'état d'une cellule $e(c_i)$. On a donc $e(c_i) \in \{\text{OCC, LIB, INC}\}$. La mise à jour des cellules se fait de manière incrémentale en utilisant Bayes. Supposons qu'on ait, pour une cellule c_i, la probabilité $P(e(c_i) = OCC \mid z^t)$ qu'elle soit occupée au temps t et une nouvelle observation z_{t+1} ; alors la mise à jour s'effectue grâce à la relation :

$$P(e(c_i) = \text{OCC} \mid z^{t+1}) = \frac{P(z_{t+1} \mid e(c_i) = \text{OCC})P(e(c_i) = \text{OCC} \mid z^t)}{\Sigma_{e(c_i)} P(z_{t+1} \mid e(c_i))P(e(c_i) \mid z^t)} \tag{5.2}$$

La grille d'occupation est modélisée par une chaîne de Markov d'ordre 0, c'est-à-dire que les états des cellules de la grille sont considérés comme des variables indépendantes. On peut utiliser des ordres plus grands mais le coût est exponentiel en le degré de la chaîne.

Nous avons représenté la carte sous forme d'une grille d'occupation comme ci-dessous :

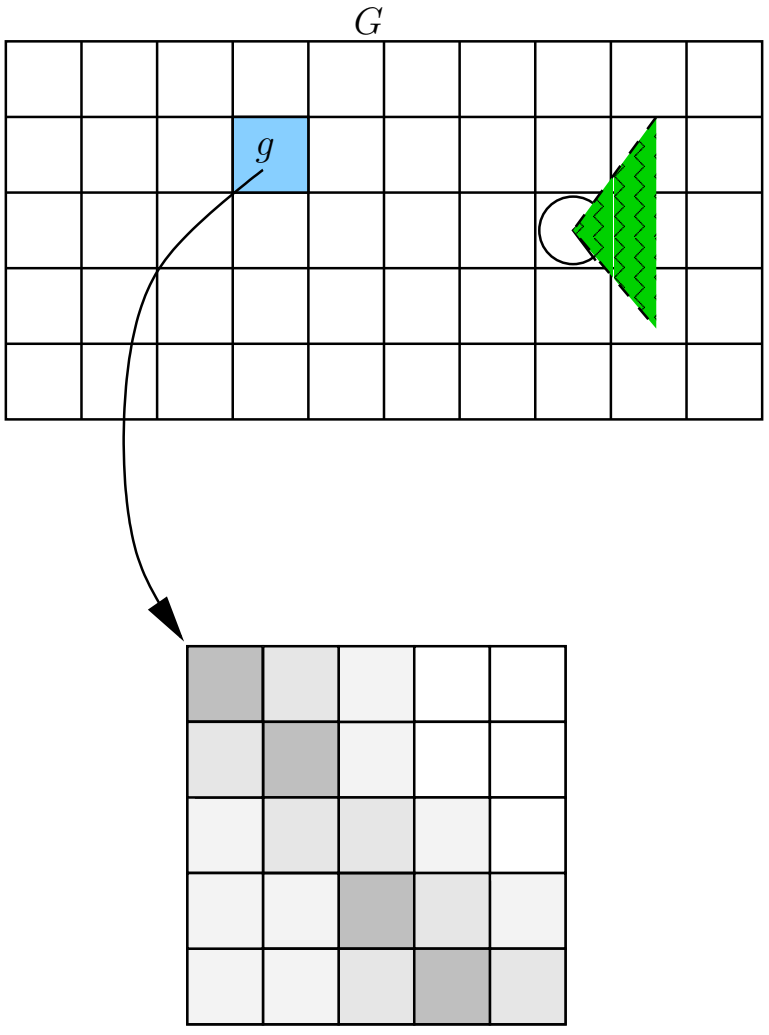


FIG. 5.2 – Représentation de la carte. La carte globale est découpée en sous-cartes.

La grille G est donc composée de grilles g plus petites ; ces dernières sont composées de cases (ou cellules) qui contiennent la probabilité P d'être occupées. Il est alors aisé d'émuler les Gaussiennes des positions des repères. Il suffit de mettre à jour les cellules voisines de la cellule impact et qui sont traversées par le rayon du laser.

Cette représentation permet également de ne pas mettre toute la carte en mémoire, il suffit de se resteindre à quelques sous-grilles selon la portée du capteur. Les grilles g sont construites de manière incrémentale au fur et à mesure de l'avancement du robot dans son environnement. Le robot commence dans une grille g de rayon la portée du laser. La teinte des cases représente la probabilité d'occupation. Plus la case est sombre et plus cette probabililté est élevée. Sur le dessin les rayons viennent d'en haut : on peut remarquer que les probabilités des cellules situées après une cellule d'impact ne sont pas nulles, selon le modèle décrit ci-dessus.

5.2 Patches

Les patches sont créés selon une version modifiée de l'algorithme de Shi, Song et Zhang [SSZ05]. Elle consiste à sélectionner les cellules de probabilité dont les probabilités sont les plus hautes, puis de chercher parmi les voisins ceux qui maximisent aussi la probabilité d'occupation. Cela revient à créer des amas de cellules dont le cœur (c'est-à-dire les cellules qui ont été traitées les premières) attire vers lui les cellules de plus faible probabilité.

Les patches sont une structure dont la partie la plus importante est la frontière. Nous avons d'abord choisi cette représentation qui permet des mises à jour relativement simples, les fusions entre patches sont également aisées.

Les figures 5.3, 5.4 et 5.5 ci-dessous montrent l'évolution d'un patch ; les cellules sont les cases du tableau. Les cellules en rouge sont les cellules qui ont été traitées. Les cellules en jaune forment la frontière. Enfin la cellule orange est celle possédant la probabilité d'occupation la plus élevée.

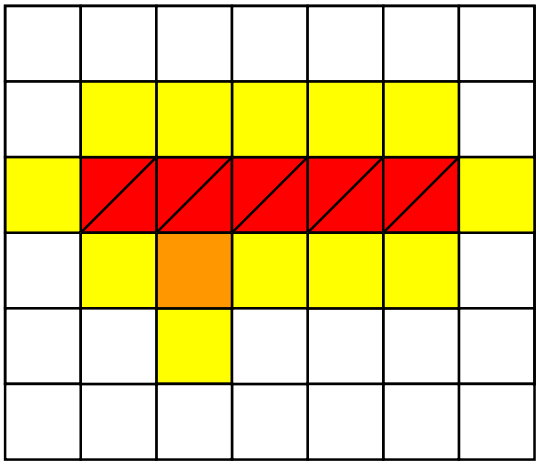


FIG. 5.3 – La cellule de la frontière possédant la plus forte probabilité d'occupation est repérée. Elle est sur le point d'être traitée.

Cette façon de procéder assure la "consistance" des patches, puisqu'on ne s'intéresse qu'aux probabilités atteignant un certain seuil. La frontière d'un point se compose de ses voisins nord, est, sud et ouest. Nous n'avons pas considéré les autres points car si un point

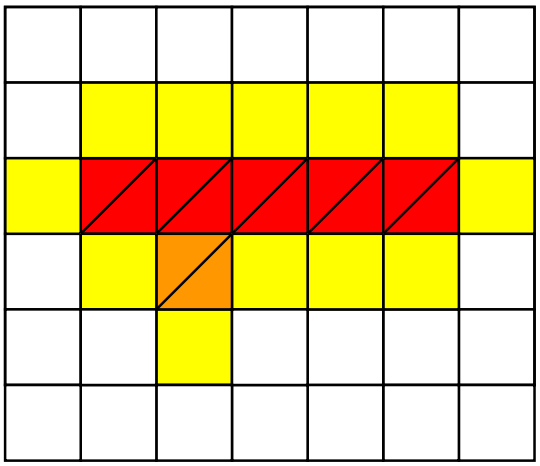


FIG. 5.4 – La cellule a été traitée, la frontière est mise à jour.

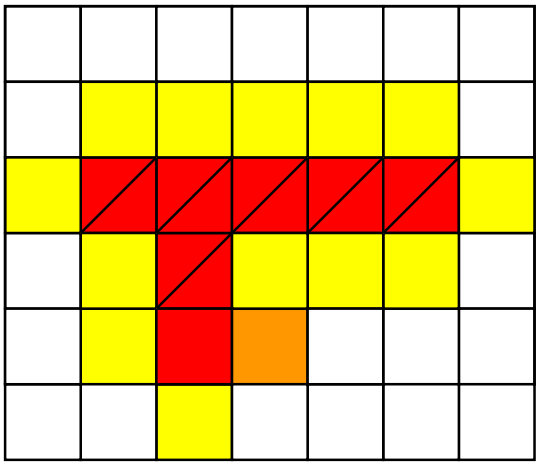


FIG. 5.5 – Les opérations de mise à jour se font tant que la fontière contient des cellules pouvant être considérées comme des repères (nous avons fixé le seuil à 0.5, ce qui correspond á la limite des états inconnus / occupés.

et son voisin nord-est devaient appartenir à un même patch, celui-ci incluerait forcément le voisin est ou le voisin nord : les patches doivent avoir un squelette continu.

5.3 Structure hiérarchique

5.3.1 Principe

Notre but est de pouvoir détecter les changements de l'environnement qui se sont opérés entre deux observations lointaines dans le temps. Ceci doit pouvoir se faire de manière à ce que le robot puisse se déplacer dans un milieu contenant des objets mobiles. Nous devons donc être en mesure de pouvoir reconnaître rapidement la carte dans son ensemble, et ne pas avoir à comparer les patches deux à deux.

Pour ce faire on introduit une structure hiérarchique qui prend la forme d'un arbre. Le niveau le plus élevé représente la carte dans sa totalité. Chaque niveau représente une vue

approchée de celle-ci, une sorte de "zoom". Chaque niveau comporte plusieurs nœuds, qui sont le regroupement des nœuds considérés proches du niveau inférieur. Le niveau le plus bas contient l'ensemble des patches.

Chaque nœud (ou feuille) est représenté par une Gaussienne, cela permet de connaître la distribution de position des balises tout en manipulant peu de données (centre, covariance).

Supposons que la carte contiennent deux bâtiments. La carte est totalement comprise dans le plus haut niveau de la hiérarchie (la racine). Chaque bâtiment est contenu dans un nœud du niveau inféieur. Chacun des murs des édifices se trouve dans une nœud. Les feuilles correspondent à des portions de mur 5.6.

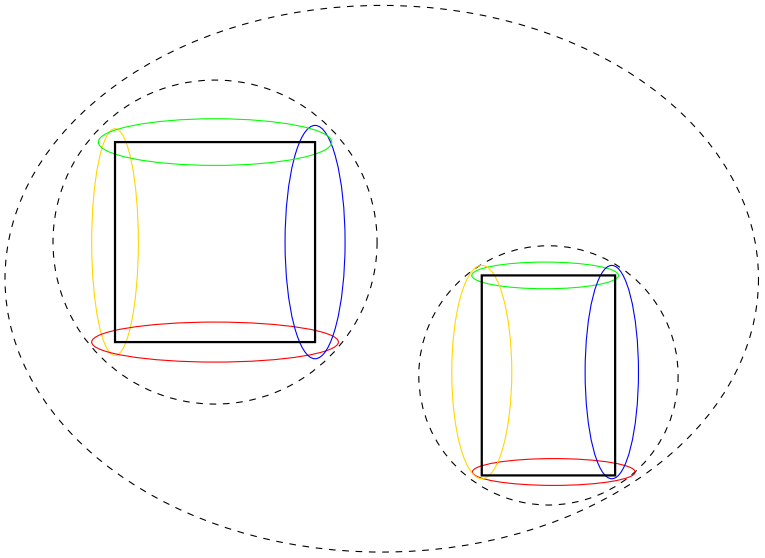


FIG. 5.6 – Structure hiérarchique sur une carte composée de deux bâtiments. Les différents niveaux de la structure forment une sorte de zoom sur la carte.

5.4 Fonction de score

La fonction de score détermine le degré de similitude entre deux cartes d'amers données. Elle calcule le ratio

$$\alpha \times \frac{\text{nombre de nœuds similaires} + \text{nombres de feuilles similaires}}{\text{nombre de nœuds} + \text{nombre de feuilles comparés}} \tag{5.3}$$

où α est un facteur dépendant du nombre de nœuds et de feuilles comparés (plus le nombre de comparaisons est élevé, plus *alpha* tend vers 1).

Si le score est élevé on procède à la comparaison des patches (*PatchMatching*). Les groupes de patches comparés ont la même distribution de position (ils appartiennent aux nœuds des deux cartes dont les centres et covariances sont semblables à une tolérance près). Le résultat de la comparaison des patches intervient dans le calcul du poids des particules :

$$p(s_t \mid \Theta_t, s_{t-1}, z_t, u_t, q_t) = \tag{5.4}$$

$$\frac{p(z_t \mid \Theta_{t-1}, s_t) \prod_t p(q_t \mid (Q^{t-1}), z^t, s^t) p(s_t \mid s_{t-1}, u_t)}{\int p(z_t \mid \Theta_{t-1}, s') \prod_t p(q_t \mid (Q^{t-1}), z^t, s') p(s' \mid s_{t-1}, u_t) ds'} \tag{5.5}$$

La comparaison proprement dite s'effectue en calculant une distance séparant les frontières des patches. Celles-ci sont inscrites dans un tableau $P_i, i \in \{1, 2\}$ qu'on parcourt, en opérant une différence absolue sur les cellules :

$$D(P_1, P_2) = \sum_{i=1}^{i=I} \sum_{j=1}^{j=J} |P_1(i, j) - P_2(i, j)| \tag{5.6}$$

Plus les patches "correspondent", plus la particule se voit affecter un poids élevé lui donnant de l'importance. Rappelons que les particules de poids élevés sont celles sur lesquelles le robot s'appuie pour déterminer sa pose et la carte de l'environnement.

$P_i(k, l)$ est la case du tableau i de ligne k et de colonne l. I et J sont choisis de manière à ce que les deux patches qu'on compare puissent ête inscrits dans P_i. On calcule ensuite la distance moyenne :

$$d_{moy} = \frac{D(P_1, P_2)}{I + J} \tag{5.7}$$

Plus le résultat est bas plus les patches sont similaires.

5.4.1 Construction et mise à jour de la hiérarchie

La robot doit déterminer si oui ou non l'environnement qu'il traverse a déjà été observé. Il doit être robuste aux changements importants de l'environnement ; on suppose toutefois que ces changements rendent encore possible la localisation, c'est-à-dire qu'il reste suffisamment de balises au même endroit depuis la dernière observation.

Les nœuds n_k de la hiérarchie sont caractérisés par deux paramètres : leur centre C_{n_k} et leur covariance Cov_{n_k}. Ceci permet de définir une Gaussienne, représentant la distribution de position des patches dont n_k est un ancêtre. Une fois les patches créés par la méthode décrite en 5.2, le robot détermine à quelles feuilles de la hiérarchie ils appartiennent. Si ces feuilles n'existent pas elles sont créées, sinon on met à jour ces feuilles et leurs parents (calcul des nouveaux centres et covariances).

Lorsque la comparaion avec une carte anérieure donne un score élevé (le robot estime que le nombre de balises est suffisant pour considérer que la configuration de l'environnement a déjà été observée dans son ensemble), le robot peut mettre à jour ou même supprimer les nœuds comme l'indiquent les schémas ci-dessous 5.4.1, 5.4.1. De manière générale la covariance est et le centre sont calculés à partir des covariances et centres des nœuds un niveau en dessous :

$$C^i_{n_k} = \begin{cases} \frac{1}{N} \sum_{j=1}^{j=N} C^{i-1}_{n_j} & \text{si } n_k \text{ est un nœud} \\ \frac{1}{N+M} \sum_{j=1}^{j=N} \sum_{k=1}^{k=M} a_{j,k} & \text{si } n_k \text{ est une feuille} \end{cases} \tag{5.8}$$

i est le niveau du nœud n_k ; $a_{j,k}, j \in \{1, \dots N\}, k \in \{1, \dots M\}$ est l'ensemble des cellules composant les patches.

La covariance est calculée comme suit :

$$Cov^i_{n_k} = \frac{1}{n}\begin{pmatrix} \alpha & \beta \\ \beta & \gamma \end{pmatrix} \tag{5.9}$$

où :
- $\alpha = \sum_x^i + 2\dot{(C^i_{n_k})}_x\dot{\sum}_x + (C_{n_k})^2$;
- $\beta = \sum_x y - (C^i_{n_k})_y \sum_x - (C^i_{n_k})_x \sum_y + n(C^i_{n_k})_x\dot{(C^i_{n_k})}_y$;
- $gamma = \sum_x^i + 2\dot{C}^i_{n_k}\dot{\sum}_x + (C_{n_k})^2$

$(C^i_{n_k})_x$ est l'abscisse du centre du nœud n_k du niveau i, $\sum_x$ (resp. $\sum_y$) est la somme des abscisses (resp. des ordonnées) des patches dont n_k est un ancêtre. n est le nombre de cellules totales des patches dont n_k est l'ancêtre.

Le robot observe dans un premier temps trois bâtiments 5.4.1 puis ensuite simplement deux 5.4.1 :

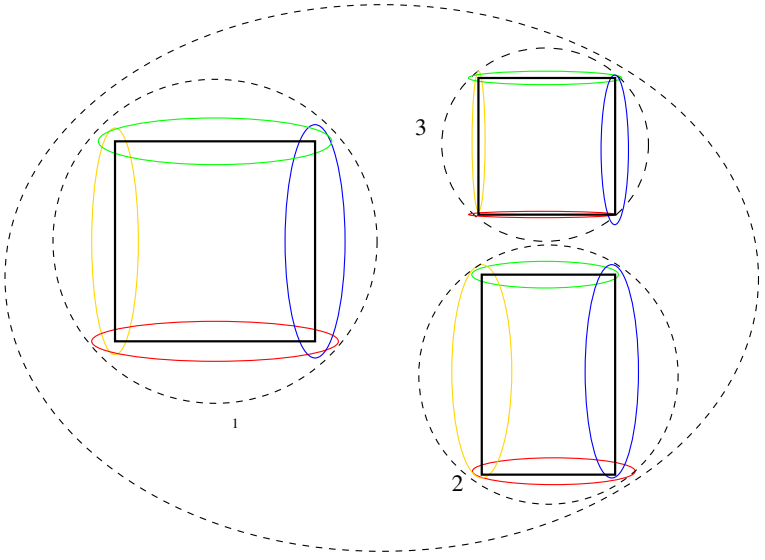


FIG. 5.7 – Carte créée à partir de l'observation de trois bâtiments. La hiérarchie est simple : un nœud par bâtiment ; les feuilles sont les murs de ces derniers.

La comparaison entre les nœuds 1 et 2 révèle qu'ils sont identiques ; toutefois la nouvelle carte ne contient plus le nœud 3. Celui-ci est supprimé, la racine est mise à jour (centre et covariance).

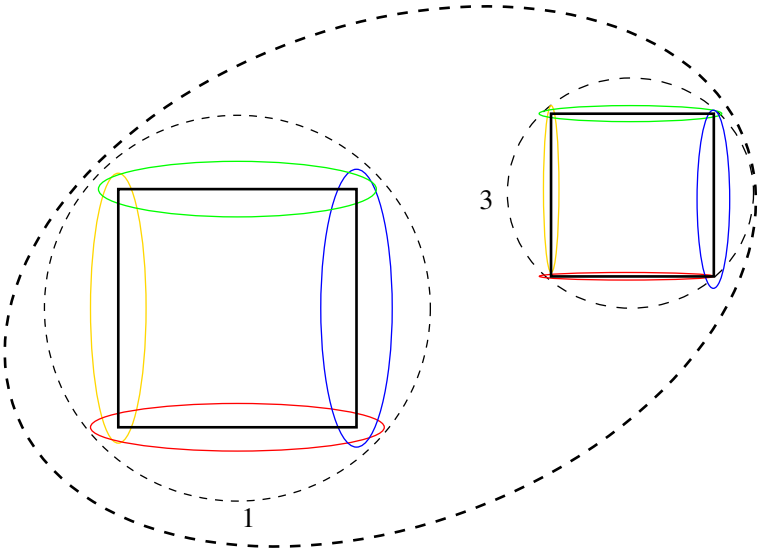


FIG. 5.8 – Plus tard, le robot n'en observe plus que deux. Il compare les deux hiérarchies obtenues à partir de ses différentes obervations ; les nœuds 1 et 3 sont identiques, le nœud 2 a disparu : la carte (donc les nœuds et patches) est mise à jour en conséquence

Chapitre 6

Résultats expérimentaux et analyse

6.1 Gestion des patches

Nous avons modifié l'algorithme de Georgio Grisetti, Cyrill Stachniss et Wolfram Burgard afin de modéliser les patches. C'est tout ce que nous avons pu tester par manque de temps. Voici les comparaisons, sans et avec la gestion des patches, de quelques images tirées d'une séquence offline de l'algorithme.

La première image montre la carte observée par le robot. On peut constater que les portions de mur ne sont pas parfaitement alignées ; malgré la précision du capteur (de l'ordre de quelques centimètres)

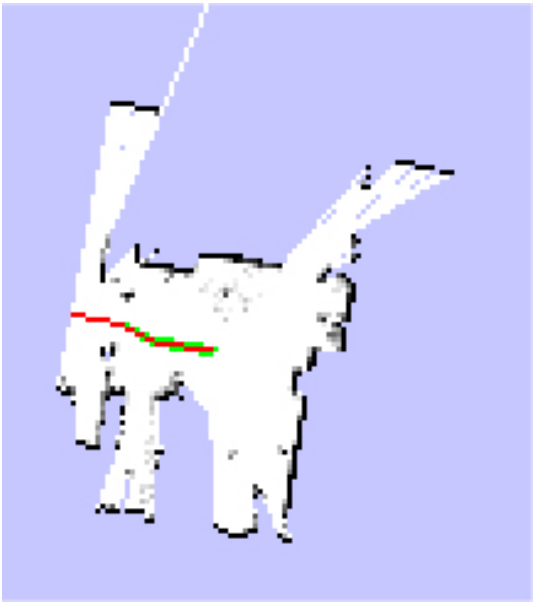


Image sans gestion des patches

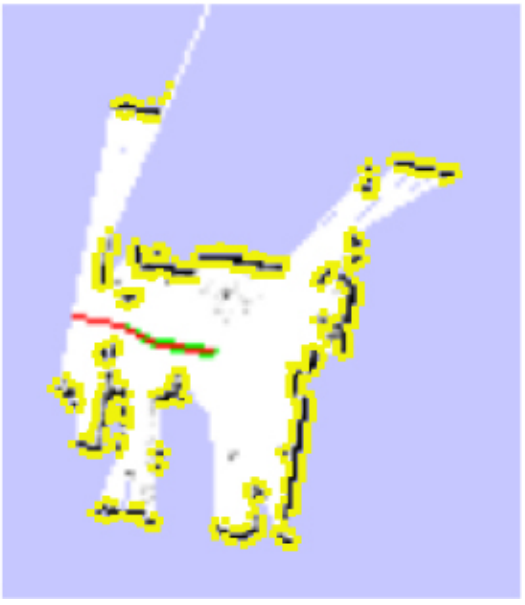


Image avec gestion des patches

FIG. 6.1 – La gestion des patches permet de mettre en évidence les relations entre les balises (*ie.* les points d'impacts du laser).

Une autre image avec gestion des patches...

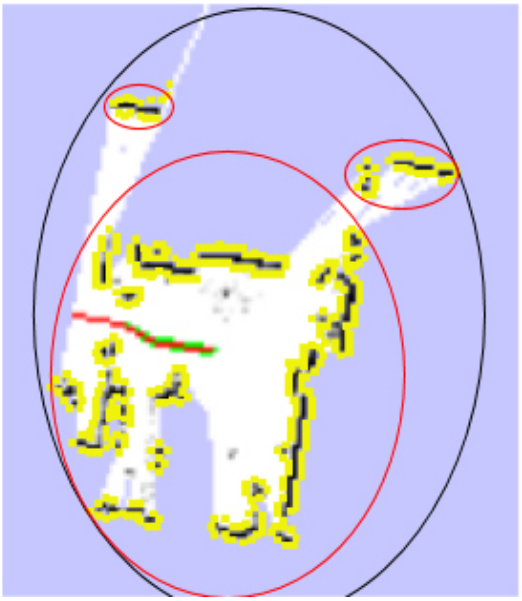
...et la même avec.

FIG. 6.2 – On peut s'apercevoir du nombre relativement important de patches que contient la carte.

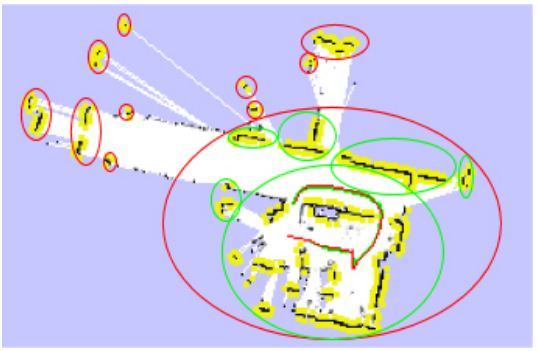
6.2 Gestion des hiérarchies

L'apport des patches et de la hiérarchie permet de structurer la carte et de s'affranchir de la contrainte de comparer toutes les données des capteurs entre elles. En effet, la hiérarchie décompose la carte en éléments principaux : les regroupements importants de balises. Ceux-ci permettent au robot de déterminer facilement si la carte a subi des changements importants. Les misesà jour de la hiérarchie se font en mettant à jour tous les ancêtres des nœuds modifiés depuis la dernière observation. La difficulté réside dans la détermination du facteur indiquant si deux patches appartiennent au même nœud. Dans nos exemples, un patch P appartient à un nœud n_k si le centre C_P de P est inclus dans la Gaussienne de n_k.

Nous n'avons pas pu tester la fonction de score complétement par manque de temps.



Gestion de la hiérarchie pour la première image



Gestion de la hiérarchie pour la seconde image

FIG. 6.3 – La structure hiérarchique rassemble les patches les plus proches ; cela traduit la volonté de traiter les informations sur les objets eux-mêmes plutôt que sur les balises brutes. Les couleurs des ellipses correspondent au niveau de la hiérarchie auquel elles appartiennent : noir pour la racine, rouge pour le second nveau, vert pour le niveau encore en-dessous. Pour plus de visibilité tous les niveaux n'ont pas été représentés.

Chapitre 7

Conclusion et perspectives

7.1 Conclusion sur le travail effectué

La prise en compte des environnements semi-statiques permet d'améliorer sensiblement les algorithmes de SLAM, puisqu'ils parviennent à déterminer les parties de l'environnement qui ont été modifiées (ajout, suppression ou déplacements de repères). Le robot est capable de détecter les parties de l'environnement les plus sûres pour calculer sa position. La carte vit son nombre d'erreurs (ajout de repères existants déjà dans la carte, localisation de repères erronnée) diminuer de façon significative. Elle devient alors plus précise et contient moins d'erreurs.

La carte de l'environnement a été modélisée sous la forme d'une grille d'occupation. Ce choix est motivé par des raisons de performance (création sous-cartes possibles) et pour sa capacité à représenter le modèle du capteur Nous avons pour cela représenté les amers par des patches. Ces patches correspondent à la présence d'un groupe de cellules adjacentes de probabilité d'être occupées élevées. La structure hiérarchique permet d'organiser la carte en parties d'objets puis en objets entiers. Nous n'avons pu malheureusement tester complétement le programme, la fonction de score n'étant pas totalement au point.

7.2 Perspectives

Il reste à finir d'implémenter la fonction de score qui permet de déterminer si deux cartes peuvent être considérées comme semblables. Il restera alors à créer des scénarios dans lequel l'envrionnement est modif'é à divers degrés.

Nous aimerions aussi modifier l'algorithme de FastSLAM afin de pouvoir, en cas d'indécision sur la pose du robot, travailler avec plusieurs ensembles de particules qui permettraient de briser l'ambiguité soulevée.

Bibliographie

[ALTT02] D. Avots, E. Lim, R. Thibaux, and S. Thrun. A probabilistic technique for simultaneous localization and door state estimation with mobile robots in dynamic environments, 2002.

[Bai01] T. Bailey. Mobile Robot Localisation and Mapping in Extensive Outdoor Environments, 2001.

[BNL+03] M. Bosse, P. Newman, J. Leonard, M. Soika, W. Feiten, and S. Teller. An atlas framework for scalable mapping. *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2003.

[DdFMR] A. Doucet, N. de Freitas, K. Murphy, and S. Russell. Rao-blackwellised particle filters for dynamic bayes net.

[DLR77] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of Royal Statistical Society B*, 39 :1–38, 1977.

[Elf89] A. Elfes. Using Occupancy Grids for Mobile Robot Perception and Navigation. *Computer*, 22(6) :46–57, 1989.

[FB87] Martin A. Fischler and Robert C. Bolles. Random sample consensus : a paradigm for model fitting with applications to image analysis and automated cartography. pages 726–740, 1987.

[Gha98] Zoubin Ghahramani. Learning Dynamic Bayesian Networks. In *Adaptive Processing of Sequences and Data Structures, International Summer School on Neural Networks, "E.R. Caianiello"-Tutorial Lectures*, pages 168–197, London, UK, 1998. Springer-Verlag.

[GSB05] G. Grisetti, C. Stachniss, and W. Burgard. Improving grid-based SLAM with RaoBlackwellized particle filters by adaptive proposals and selective resampling, 2005.

[HFBT03] D. Hähnel, D. Fox, W. Burgard, and S. Thrun. A highly efficient FastSLAM algorithm for generating cyclic maps of large-scale environments from raw laser range measurements. In *Proceedings of the Conference on Intelligent Robots and Systems (IROS)*, 2003.

[Kal60] R. E. Kalman. A new approach to linear filtering and prediction problems. In *Trans. ASME, Journal of Basic Engineering*, volume 82, pages 35–45, 1960.

[KDR01] J. Knight, A. Davison, and I. Reid. Towards constant time slam using postponement. *EEE/RSJ International Conference on Intelligent Robots and Systems*, pages 405–413, 2001.

[MMM05] L. Montesano, J. Minguez, and L. Montano. Modeling the Static and the Dynamic Parts of the Environment to Improve Sensor-based Navigation. In *IEEE Int. Conf. on Robotics and Automation*, Barcelona, Spain, 2005.

[MT] Michael Montemerlo and Sebastian Thrun. FastSLAM 2.0 : An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges.

[MTKW02] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM : A factored solution to the simultaneous localization and mapping problem, 2002.

[Mur] K. Murphy. Bayesian map learning in dynamic environments.

[NGNT03] J. Nieto, J. Guivant, E. Nebot, and S. Thrun. Real time data association for fastslam, 2003.

[PB05] Tom Duckett Peter Biber. Dynamic Maps for Long-Term Operation of Mobile Service Robots. In *Proceedings of Robotics : Science and Systems*, Cambridge, USA, June 2005.

[SB05] C. Stachniss and W. Burgard. Mobile Robot Mapping and Localization in Non-Static Environments. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, Pittsburgh, PA, USA, 2005. To appear.

[SC86] Randall C. Smith and Peter Cheeseman. On the representation and estimation of spatial uncertainly, 1986.

[SK97] Hagit Shatkay and Leslie Pack Kaelbling. Learning Topological Maps with Weak Local Odometric Information. In *IJCAI (2)*, pages 920–929, 1997.

[SSZ05] Yong Shi, Yuqing Song, and Aidong Zhang. A shrinking-based clustering approach for multidimensional data. *IEEE Transactions on Knowledge and Data Engineering*, 17(10) :1389–1403, 2005.

[TBF00] S. Thrun, W. Burgard, and D. Fox. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping. In *Proc. of the IEEE International Conference on Robotics & Automation*, 2000.

[TFB98] S. Thrun, D. Fox, and W. Burgard. A probabilistic approach to concurrent mapping and localization for mobile robots. *Machine Learning*, 31 :29–53, 1998. also appeared in Autonomous Robots 5, 253–271 (joint issue).

[Thr93] Sebastian Thrun. Exploration and model building in mobile robot domains. In *In Proceedings of the IEEE International Conference on Neural Networks*, 1993. IEEE Neural Network Council.

[Thr01] S. Thrun. A probabilistic online mapping algorithm for teams of mobile robots. *International Journal of Robotics Research*, 20(5) :335–363, 2001.

[TKG+02] S. Thrun, D. Koller, Z. Ghahramani, H. Durrant-Whyte, and A. Ng. Simultaneous mapping and localization with sparse extended information filters, 2002.

[TMW02] S. Thrun, M. Montemerlo, and W. Whittaker. Conditional particle filters for simultaneous mobile robot localization and people-tracking, 2002.

[WS] Denis Wolf and Gaurav S. Sukhatme. Online Simultaneous Localization and Mapping in Dynamic Environments.

[WT02] C. Wang and C. Thorpe. Simultaneous localization and mapping with detection and tracking of moving objects, 2002.

[WTT03] C. Wang, C. Thorpe, and S. Thrun. Online Simultaneous Localization and Mapping with Detection and Tracking of Moving Objects : Theory and Results from a Ground Vehicle in Crowded Urban Areas, 2003.

[YB96] B. Yamauchi and R. Beer. Spatial Learning for Navigation in Dynamic Environments, 1996.