Faculté
des Sciences
Département d'informatique

UMONS
Université de Mons

Algorithmes probabilistes de localisation implémentés sur une brique EV3

Mémoire réalisé par Benoît HOFBAUER
pour l'obtention du diplôme de Master en sciences informatiques

Année académique 2014–2015

Directeurs: Hadrien Mélot
Pierre Hauweele
Service: Algorithmique

Faculté des Sciences • Université de Mons • Place du Parc 20 • B-7000 Mons

Remerciements

Je tiens à remercier Hadrien Mélot le directeur de ce mémoire et Pierre Hauweele le codirecteur de ce mémoire de m'avoir apporté leur expertise dans le domaine et leur aide précieuse. Pierre Hauweele m'a également transmis sa passion de la robotique, ce qui m'a permis de trouver la motivation de faire ces recherches dans le domaine de la robotique qui est un domaine de taille importante et qui comporte de nombreuses difficultés.

Je tiens également à remercier ma mère qui m'a aidé à relire et structurer ce document ainsi que l'ensemble des membres de ma famille de m'avoir soutenu et aidé chacun à leur façon durant cette période.

Et finalement, je remercie Sylvain Kempen, étudiant en sciences mathématiques à l'UMons, de m'avoir aidé à comprendre les aspects mathématiques les plus complexes de ce mémoire. Sa gentillesse et ses capacités de vulgarisation des mathématiques m'ont été très utiles tout au long de la rédaction de ce mémoire.

Mots clés

— Localisation
— Extended Kalman filter
— Monte Carlo localization
— EV3
— Robotique

Résumé

L'objectif de ce mémoire est de développer des aspects théoriques de la robotique en y liant une validation pratique. La localisation d'un robot dans son environnement est le principal sujet abordé. Les algorithmes de localisation probabilistes, Extended Kalman filter (EKF) ainsi que Monte Carlo localization (MCL) seront étudiés en profondeur et comparés. Toutefois, une multitude de sous-sujets sont liés et découlent de la localisation d'un robot dans son environnement. Les principaux sous-sujets sont la construction de cartes de l'environnement du robot, la recherche de chemin entre un point de départ et un point final, les systèmes d'exploitation dans la robotique, la caractérisation des capteurs et des actuateurs, l'analyse d'images... Cet ensemble de sous-sujets est abordé dans ce mémoire pour permettre aux lecteurs de comprendre une partie de l'univers de la robotique et lui donner des pistes et des références pour entamer des recherches futures. La compréhension de l'univers de la robotique est primordiale pour intégrer les algorithmes de localisation en un tout cohérent.

Le kit EV3 de Lego est utilisé pour la validation pratique des concepts théoriques présentés. Ce kit est composé d'une brique intelligente, de capteurs, de moteurs ainsi que d'éléments de construction qui permettent de réaliser rapidement la structure d'un robot. Ce kit est combiné avec un smartphone tournant sous Android. Ce smartphone permet d'ajouter de la puissance de calcul et de se servir des capteurs présents sur le smartphone. En effet, les smartphones actuels possèdent un grand nombre de capteurs à couts réduits et leur capacité de calcul est importante. Le principal capteur utilisé est la caméra du smartphone.

Avant-propos

Les deux mots « voiture autonome » sont sur les lèvres de tous les constructeurs automobiles depuis quelques années. Plusieurs prototypes ont été réalisés depuis les années 1980 [24, 13]. Cependant, ces prototypes étaient limités soit par la vitesse du véhicule soit par les restrictions de l'environnement dans lequel ils évoluaient. L'évolution des technologies liées aux capteurs et l'augmentation de la puissance des processeurs ont permis une avancée considérable dans le domaine. Google, annonce en octobre 2010 [15] avoir conçu un système de pilotage automatique pour automobile. Ce prototype est capable de se déplacer dans la circulation automobile sans assistance humaine. À ce jour, de nombreux constructeurs automobiles travaillent sur des voitures autonomes. On peut citer Audi, Toyota [2], Nissan [1], Mercedes-Benz.

La multitude de capteurs qui équipent ces véhicules est évidemment primordiale. Toutefois, sans traitements et croisements de ces mesures, il est impossible de développer un véhicule autonome. Pour cette raison, des algorithmes ont été développés. Ces algorithmes permettent aux véhicules de se localiser dans leur environnement pour ensuite établir leur parcours. Ces algorithmes doivent prendre en compte que les valeurs des capteurs peuvent être entachées d'erreurs. Ils doivent également prendre en compte que les capteurs ne permettent de capter qu'une partie de l'environnement du véhicule. Des véhicules comme la voiture autonome de Google embarquent un grand nombre des capteurs de hautes précisions. Toutefois, équiper des voitures de ces capteurs se révèle encore très onéreux. Il est donc intéressant de se demander quelles sont les limites des algorithmes de localisation avec des capteurs à faibles couts. Ces algorithmes sont-ils perfectibles ? Les robots de nettoyage domestique [11] sont des exemples de robots qui ont des capteurs simples et peu onéreux, mais qui doivent se déplacer dans des environnements complexes que sont les habitations. En effet, les meubles, les escaliers, les humains et les animaux domestiques sont autant d'éléments qui évoluent dans le même environnement que le robot. Ils rendent donc la localisation et les déplacements plus complexes pour le robot.

Les voitures autonomes ainsi que les robots de nettoyage domestique peuvent être au même titre qualifiés de robots. On peut facilement assimiler les voitures autonomes et les robots de nettoyage à la définition suivante de ATILF : «Appareil effectuant, grâce à un système de commande automatique à base de microprocesseur, une tâche précise pour laquelle il a été conçu dans le domaine industriel, scientifique ou domestique». Dans ce mémoire le but est donc de faire découvrir l'univers de la robotique, avec la localisation d'un robot dans son environnement comme sujet principal. Il a été choisi de présenter ce mémoire de sorte que le lecteur puisse l'utiliser comme la

base principale pour implémenter ses propres algorithmes de localisation sur le robot de son choix. C'est pourquoi les algorithmes sont décrits, mais également comparés pour permettre au lecteur de choisir les algorithmes les plus adaptés aux spécificités et à l'environnement de son robot. De nombreuses citations permettent également aux lecteurs de continuer ses recherches selon la direction dans laquelle il souhaite approfondir ses recherches dans la robotique. On peut donc considérer ce mémoire comme un tutoriel qui aide à débroussailler l'univers énorme que représente la robotique. Il est découpé en deux parties distinctes. La première décrit la théorie. La seconde met en pratique la théorie présentée. Les personnes à l'aise avec cette théorie pourront donc directement passer à la partie pratique. Cependant pour les novices dans la robotique, il est recommandé de lire l'intégralité du mémoire, et ce dans l'ordre d'apparition des chapitres. En effet, des exemples concrets sont donnés pour aider le lecteur à se familiariser avec les différentes notions abordées.

Le livre de Sebastian Thrun « Probabilistic Robotics » [22] qui est la bible dans le domaine de la robotique a été une ressource importante pour la rédaction de ce mémoire. Rappelons que Sebastian Thrun est l'ingénieur principal qui a lancé le projet « Google driverless car ». Le mémoire de Pierre Hauweele [10] donne certaines démonstrations théoriques plus en profondeur alors que ce mémoire est plus centré sur les aspects pratiques.

Remarque 1 : ce mémoire est écrit suivant la nouvelle orthographe de la langue française approuvée par l'Académie française [7]. Le lecteur ne doit donc pas être surpris de voir, par exemple, le mot "cout" écrit sans accent circonflexe.

Remarque 2 : certains pieds de page présentent des URL qui renvoient à des sites web de références. Cependant, dans un souci de présentation de ce mémoire ces liens sont limités au nom de domaine du site. La version PDF du mémoire permet d'être redirigé vers le lien complet. Il est recommandé de m'envoyer un e-mail à l'adresse Hofbauer92@gmail.com pour vous procurer cette version PDF.

Remarque 3 : ce mémoire est destiné à être imprimé recto verso pour diminuer son impact environnemental. L'université de Mons prône fortement les valeurs de la réduction de l'impact environnemental des universités qui sont de grandes consommatrices de papier. L'UMons a même reçu un prix pour son engagement [1] . L'impression recto verso de ce mémoire est donc en accord avec ces valeurs.

1. UMons diminution de l'impact environnemental : portail.umons.ac.be

Table des matières

Table des figures

Première partie

Localisation d'un robot dans son environnement

Cette partie présente les notions théoriques qui sont utilisées dans la seconde partie. La seconde partie consiste à décrire l'implémentation de l'algorithme de localisation EKF sur un robot EV3. Cette implémentation est comparée à l'implémentation de l'algorithme de localisation MCL déjà présente dans la librairie Lejos qui est l'outil utilisé pour contrôler le robot EV3. Pour implémenter l'algorithme de localisation EKF, il est nécessaire de comprendre la logique des algorithmes de localisation. C'est pourquoi ces deux algorithmes de localisation qui sont les plus célèbres sont présentés dans cette partie. Ils sont ensuite comparés théoriquement. Ces algorithmes utilisent des cartes pour localiser le robot, la première section est donc destinée à présenter ces cartes. Pour le lecteur désirant en savoir plus, l'algorithme EKF SLAM et deux algorithmes de recherche de chemin sont présentés. La section SLAM décrit comment le robot peut simultanément construire et se localiser sur une carte. Les algorithmes de recherche de chemin permettent au robot d'aller d'un point de départ jusqu'à un point final sans percuter d'obstacles. Dans cette section l'algorithme A star et Dijkstra sont présentés, car les implémentations de ces deux algorithmes sont présentes dans la librairie Lejos et elles sont utilisées dans la partie pratique de ce mémoire.

Chapitre I

Les cartes

Les cartes de l'environnement du robot sont des éléments importants dans ce mémoire. Ces cartes peuvent être générées dynamiquement ou au préalable avant de les passer en paramètre au robot. Elles permettent de se localiser ou bien de trouver un plus court chemin entre un point de départ et un point d'arrivée. Il y a plusieurs types de cartes ayant chacune leurs caractéristiques. Les sections suivantes sont destinées à décrire les cartes les plus répandues qui sont les grilles d'occupation et les cartes composées de repères.

I.1 Carte composée de repères

Cette carte est définie à l'aide d'une liste ou d'une sous-liste d'objets composant l'environnement du robot. Cette liste comporte les objets importants de l'environnement, qui sont des repères pour le robot. Elle peut par exemple contenir la liste des murs de l'environnement et les meubles. Formellement, la carte m contient une liste de repères et est dite «feature-based». Elle est définie comme suit :

$$m = \{m_1, m_2, ..., m_N\}$$

où N représente le nombre de repères dans l'environnement. À chaque repère de la carte correspondent ses coordonnées dans l'environnement. Cette représentation permet facilement de modifier les données associées aux repères et d'y accéder. Il est donc facile de modifier la position des éléments dynamiques de l'environnement comme des personnes ou objets mobiles au cours du temps. La figure I.1 représente le parcours d'un drone qui se déplace dans un parc et où les objets cartographiés correspondent à la cime des arbres du parc. Il est clair dans cet exemple que la carte ne contient pas tous les objets du parc, mais bien un sous-ensemble des points de repère les plus importants. Ce technique permet donc de cartographier des environnements de grande taille en

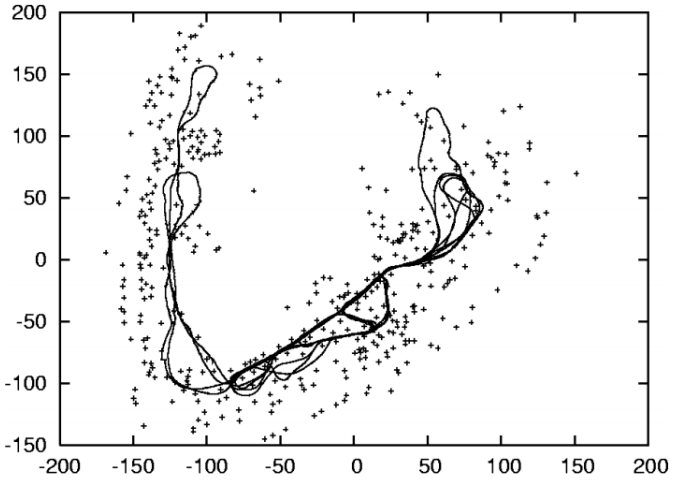

FIGURE I.1 – Carte composée de repères
Source: Probabilistic Robotics[22]

se préoccupant uniquement des éléments considérés comme importants. Une discussion sur l'extraction de repères à l'aide des capteurs et spécifiquement à l'aide d'une caméra est faite dans la section VIII.1.

I.2 La grille d'occupation

Les cartes de type grilles d'occupation découpent l'environnement du robot en un ensemble de parcelles de même taille. Cette carte m de type «location-based» est notée comme suit :

$$m = \{m_{1,1}, m_{1,2}, ..., m_{1,Y}, ..., m_{2,1}, ..., m_{X,Y}\}$$

où X, Y représentent respectivement la largeur et la hauteur maximales de l'environnement cartographié. À chaque parcelle une valeur est associée. Les valeurs peuvent être binaires et définir si une parcelle est occupée ou non, ou bien peuvent être une variable qui définit la probabilité d'avoir une parcelle vide ou occupée. Dans le second cas, un seuil qui définit si la parcelle est vide et un seuil qui définit si la parcelle est occupée doivent être définis pour pouvoir utiliser la carte. Ce type de carte à l'avantage de définir la présence d'un objet, mais également l'absence d'objet ce qui n'est pas le cas des cartes composées de repères. La figure I.2 correspond à la cartographie du campus de Stanford à l'aide d'une carte d'occupation. La carte est générée dynamiquement à l'aide d'un capteur de distance infrarouge qui permet de définir les parcelles

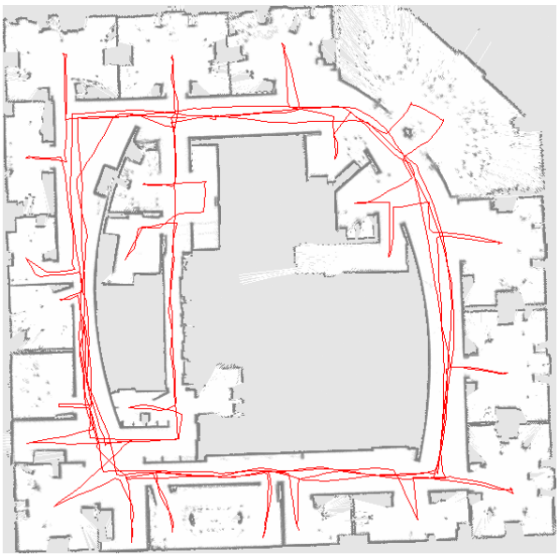

FIGURE I.2 – Grille d'occupation
Source: Probabilistic Robotics[22]

de la carte qui sont vides ou occupées. Le problème principal de ce type de cartes est que leur efficacité est fortement liée à la taille de la découpe. Si la découpe est importante, cette grille fournit des informations précises. À contrario, les temps de mises à jour et de lectures sont importants. Il est donc important de bien définir une découpe optimale. Cette découpe optimale est bien entendu fonction de la puissance de calcul, de la précision voulue et de la taille de l'environnement.

Chapitre II

Problème de localisation

II.1 Définition du problème

II.1.1 Explication du problème

Le problème de localisation d'un robot mobile consiste à déterminer sa position à un instant donné sur une carte donnée. Pour atteindre cet objectif, le robot a à sa disposition les mouvements qu'il a réalisés, des mesures provenant de ses capteurs ainsi qu'une carte de son environnement.

Cette situation peut facilement être comparée à un promeneur cherchant sa position dans la nature avec une carte topographique. Cette personne n'a à sa disposition que les observations qu'elle peut réaliser (sans l'aide d'instruments de localisation comme un GPS). Elle peut se localiser à l'aide des montagnes qui sont des points de repère intéressants. Elle peut également essayer de trouver des ressemblances avec le chemin qu'elle parcourt et ce qu'elle peut observer sur la carte. Cependant, il est difficile d'estimer exactement la distance qui sépare cette personne de la montagne. La distance parcourue par cette personne entre deux points est également difficile à estimer sans erreurs. Toutes ces informations sont donc approximatives. Malgré ces erreurs d'approximation, grâce à la quantité d'informations accumulées durant son parcours cette personne a de fortes chances d'être de plus en plus certaine de sa position. En effet, en début de parcours, cette personne peut supposer être à un ensemble d'endroits différents à la suite d'un manque d'informations en sa possession. Par la suite grâce aux nouvelles informations elle peut procéder par élimination pour déterminer sa position.

Il s'avère que les robots doivent faire face aux mêmes types de problèmes pour se localiser. Grâce à l'odométrie, il est possible de déterminer les mouvements du robot en fonction de la rotation de chacun des moteurs du robot. À l'aide de capteurs tels que les capteurs infrarouges, il est possible de déter-

miner la distance entre la position du robot et un objet. Cependant comme pour le promeneur ces informations sont entachées d'erreurs de précision. De plus, dans le cas des capteurs de distance infrarouges ou ultrasoniques, les ondes peuvent être réfléchies de façon inattendue selon la forme et la matière de la surface de l'objet réfléchissant l'onde. Il s'avère que chacun des capteurs possède des problèmes spécifiques aux technologies qu'ils utilisent. Une solution naïve serait de vouloir acheter des détecteurs et des moteurs toujours plus précis. Cependant, le cout des capteurs plus précis est plus important. De plus, des erreurs peuvent être impossibles à gérer à l'aide de matériel plus précis. En effet, il peut arriver que les roues du robot n'adhèrent pas parfaitement à la route. Ce qui entraine le glissement des roues et donc bien que le robot reste immobile, les moteurs enregistreront un mouvement. Si le robot évolue dans un monde dynamique il peut arriver qu'une personne, ou un autre objet passe devant un capteur or cet élément n'est pas représenté sur la carte. Ce qui pourrait entrainer l'assimilation de cet élément à un autre élément de la carte. Un autre élément d'erreur lié à la carte est sa précision. La carte fournie au robot n'est pas d'une précision infinie. Les éléments cartographiés peuvent se retrouver légèrement à côté de la position définie sur la carte. Les capteurs ont également des limitations physiques, il est par exemple impossible pour une caméra de voir à travers les murs, ce qui ne donne au robot qu'une vue partielle de l'environnement dans lequel il évolue. Finalement, une erreur spécifique aux robots est le kidnapping. Ce qui correspond à l'arrêt du robot, suivi d'un déplacement du robot. Une fois celui-ci remis en marche, il n'a pas conscience qu'il a été changé de place.

L'ensemble des problèmes et contraintes présentés met en évidence qu'il n'est pas possible d'utiliser les données des capteurs et moteurs sans une analyse et un traitement préalable. Il pourrait être catastrophique de vouloir intégrer une valeur fortement entachée d'erreurs. Cette seule valeur pourrait conduire à définir une localisation d'un robot complètement farfelue et ceci avec les risques qui en découlent. Les techniques de localisation probabiliste qui sont présentées dans ce mémoire permettent de pallier ces valeurs erronées. Les sections suivantes discutent des solutions apportées aux différents problèmes discutés.

II.1.2 Modèle de mouvement et d'observation

L'objectif de la localisation d'un robot est de définir sur une carte la position du robot à un instant donné. Cette position est dénotée par le vecteur x_t. Dans la suite de ce mémoire, la position du robot est définie par ses coordonnées dans un espace à deux dimensions ainsi que par son orientation. Les algorithmes présentés ne se limitent pas à cette situation et

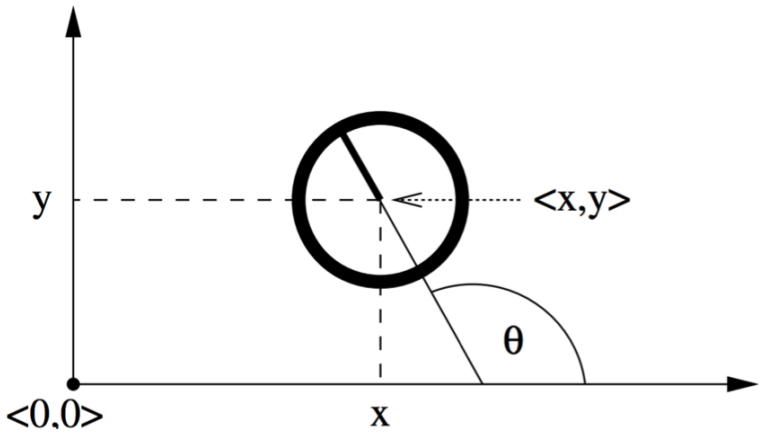

FIGURE II.1 – Position d'un robot dans un espace à deux dimensions
Source: Probabilistic Robotics[22]

peuvent être utilisés pour un espace à un nombre de dimensions supérieures. L'état des bras des robots industriels [18] peut également être représenté à l'aide de l'angle de chaque articulation du bras. Toutefois, les principes sont plus simples à comprendre dans cette situation et cette situation est souvent suffisante pour les robots mobiles. Formellement, la position d'un robot à l'instant t dans un espace à deux dimensions peut-être définie par le vecteur :

$$x_t = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$$

où x, y correspondent aux coordonnées dans l'espace à deux dimensions et θ correspond à l'orientation du robot (voir II.1).

Pour déterminer la position du robot dans le temps, il faut prendre en considération les contrôles du robot (notés : u_t) ainsi que les observations effectuées par le robot (notées : z_t). Les algorithmes qui seront présentés par la suite suivent l'hypothèse de Markov. C'est-à-dire que l'état x_t ne dépend que de l'état x_{t-1} ainsi que des contrôles u_t et des observations courantes z_t(voir figure II.2). Les noeuds grisés correspondent aux informations connues et les noeuds blancs aux informations inférées à l'aide des informations connues. Formellement, les contrôles peuvent être définis par le vecteur suivant :

$$u_t = \begin{pmatrix} d_t \\ \gamma_t \end{pmatrix}$$

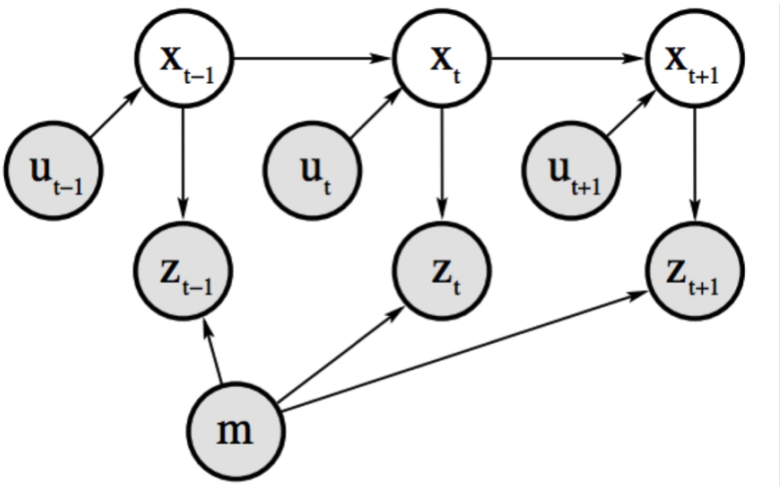

FIGURE II.2 – Hypothèse de Markov
Source: Probabilistic Robotics[22]

où d_t correspond à la distance parcourue et γ_t correspond à l'angle de rotation du robot dans l'intervalle de temps séparant deux contrôles. L'odométrie permet de déterminer les commandes u_t. Tandis que les valeurs de u_t permettent de définir itérativement les nouvelles positions à l'aide du modèle de mouvement suivant :

$$x_t = \begin{pmatrix} x_{t-1} + d_t \cos(\theta_{t-1} + \gamma_t) \\ y_{t-1} + d_t \sin(\theta_{t-1} + \gamma_t) \\ \theta_{t-1} + \gamma_t \end{pmatrix}$$

Les observations effectuées par le robot peuvent être définies par le vecteur suivant :

$$z_t = \begin{pmatrix} d_t^z \\ \rho_t^z \end{pmatrix}$$

où d_t^z correspond à la distance entre le robot et l'objet observé et ρ_t^z correspond à l'angle formé entre l'orientation du robot et la position de l'objet. Cet objet observé est noté j. Cette mesure de terrain doit être comparée aux informations sur l'objet j contenues dans la carte à la disposition du robot. La formule suivante donne à l'aide de la carte, la distance d^m et l'angle p^m entre la position du robot et l'objet j. (Remarque : Lorsque l'on considère qu'il n'y a pas d'erreurs de mesure, les valeurs de $\hat{z}_t$ sont égales à celles de z_t)

$$\hat{z}_t = \begin{pmatrix} d_t^m \\ \rho_t^m \end{pmatrix} = \begin{pmatrix} \sqrt{(m_{j,x} - x_t)^2 + (m_{j,y} - y_t)^2} \\ atan2(m_{j,y_t} - y, m_{j,x} - x_t) - \theta_t \end{pmatrix}$$

où $m_{j,x}, m_{j,y}$ correspondent aux coordonnées de l'objet scanné j. Ces coordonnées sont stockées dans la carte. x_t, y_t, θ_t correspondent aux coordonnées de la position et l'orientation du robot au temps t de l'observation.

Ces équations ne sont vraies que si les valeurs retournées par les moteurs et capteurs étaient justes à 100 %, ce qui n'est évidemment pas le cas. En effet, ces données sont sujettes à des erreurs de mesure. Il est intéressant de remarquer que si l'odométrie n'était pas sujette à des erreurs de mesure, il ne serait pas utile d'équiper ces robots de capteurs pour les localiser, l'odométrie serait suffisante. Afin de prendre en compte les erreurs, nous allons redéfinir notre modèle de mouvement et notre modèle d'observation. Une erreur de rotation initiale ainsi qu'une erreur sur la distance sont ajoutées à chaque contrôle u_t. Pour cela γ et d sont redéfinis par $\hat{\gamma}$ et $\hat{d}$ à l'aide des formules suivantes :

$$\hat{\gamma} = \gamma - \varepsilon_{\alpha_1\gamma^2+\alpha_2 d^2}$$

$$\hat{d} = d - \varepsilon_{\alpha_3 d^2+\alpha_4\gamma^2}$$

où $\alpha_1, \alpha_2, \alpha_3, \alpha_3$, sont des paramètres à déterminer et sont spécifiques à chaque robot et ε_{b^2} correspond à une gaussienne de moyenne nulle et de variance b^2. Cette formule montre que plus la distance d et l'angle γ sont importants et plus l'erreur risque d'être grande. Ce qui correspond bien à la réalité. $\alpha_2 d^2$ et $\alpha_4\gamma^2$ correspondent aux erreurs où une rotation est considérée par une translation et inversement. À l'aide de ces deux formules, le modèle de mouvement auquel on a ajouté les erreurs de mesure devient donc :

$$x_t = \begin{pmatrix} x_{t-1} + \hat{d}_t \cos(\theta_{t-1} + \hat{\gamma}_t) \\ y_{t-1} + \hat{d}_t \sin(\theta_{t-1} + \hat{\gamma}_t) \\ \theta_{t-1} + \hat{\gamma}_t \end{pmatrix}$$

Dans le modèle d'observation, une erreur gaussienne est ajoutée sur la mesure de la distance d^m et sur l'angle ρ^m entre le robot et l'objet j.

$$\hat{z}_t = \begin{pmatrix} d_t^m \\ \rho_t^m \end{pmatrix} = \begin{pmatrix} \sqrt{(m_{j,x} - x_t)^2 + (m_{j,y} - y_t)^2} \\ atan2(m_{j,y_t} - y, m_{j,x} - x_t) - \theta_t \end{pmatrix} + \begin{pmatrix} \varepsilon_{\sigma_d^2} \\ \varepsilon_{\sigma_\rho^2} \end{pmatrix}$$

où $\varepsilon_{\sigma_d^2}, \varepsilon_{\sigma_\rho^2}$ sont des gaussiennes de moyenne égale à zéro et respectivement de variance $\sigma_d^2, \sigma_\rho^2$. La section II.2 décrit comment les mesures de terrain z_t et les informations disponibles dans la carte sont utilisées pour permettre au robot d'ajuster sa position. Elle décrit également comment définir que l'objet scanné sur le terrain correspond à l'objet j de la carte.

Il est important de remarquer que le modèle de mouvement et le modèle d'observation présentés sont des exemples et doivent évidemment être adaptés aux différents robots.

II.1.3 Algorithme de localisation de Markov

Dans ce mémoire les algorithmes développés sont des algorithmes de localisation probabiliste. L'approche probabiliste permet d'intégrer dans l'algorithme les erreurs de précision des capteurs et des moteurs. Leur objectif est de déterminer la fonction de densité de probabilité du vecteur aléatoire X associé.

$$E \to [0;1] : x \mapsto p(X = x)$$

L'algorithme 1 qui est décrit en pseudocode correspond à l'algorithme de localisation de Markov qui est à la base de tous les algorithmes de localisation qui sont présentés dans ce mémoire. Il est dérivé du filtre bayésien [3] qui est lui-même basé sur la loi bayésienne qui est la suivante.

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

Le théorème de la probabilité totale est également important dans l'algorithme de Markov. Dans le cas discret, il est défini comme suit :

$$p(y) = \sum_{x'} p(y|x')p(x')$$

dans le cas continu, il devient :

$$p(y) = \int p(y|x')p(x')dx'$$

L'algorithme de Markov décrit donc la mise à jour de la position x_{t-1} vers la position x_t. Il prend en paramètre la croyance de la position précédente (c'est-à-dire la fonction de densité de probabilité du vecteur de position x au temps $t-1$), le contrôle courant, les observations courantes ainsi que la carte dans laquelle le robot évolue. Il est constitué d'une boucle principale, qui itère sur toutes les valeurs possibles pour la position x_t. Ce qui permet de déterminer la probabilité associée à chaque position que pourrait prendre le robot. Pour cela, cette boucle contient deux étapes importantes. La première étape se nomme «la prédiction» et consiste à calculer une croyance temporaire $\overline{bel}$ de la position du robot à l'aide de u_t et de la croyance de l'étape précédente $bel(x_{t-1})$. Elle se base sur le théorème de la probabilité totale. La seconde étape correspond à la mise à jour de la croyance $bel(x_t)$ à l'aide des mesures

z_t et de la croyance $\overline{bel}(x_t)$ calculée dans l'étape de prédiction. Cette étape se base sur la loi bayésienne dans laquelle $p(z_t)$ ne dépend pas de x_t et est toujours égale pour toutes valeurs de $p(x_t|z_t, x_t, m)$. Pour cette raison le η correspond à un normalisateur qui permet de garder une loi de probabilité et il est égal à $p(z_t)^{-1}$.

Algorithm 1 Localisation de Markov

1: **procedure** MARKOV($bel(x_{t-1}), u_t, z_t, m$)
2: **for all** x_t **do**
3: $\overline{bel}(x_t) \leftarrow \int p(x_t \mid u_t, x_{t-1}, m) bel(x_{t-1}) dx_{t-1}$ ▷ prédiction
4: $bel(x_t) \leftarrow \eta p(z_t \mid x_t, m) \overline{bel}(x_t)$ ▷ mise à jour
5: **end for**
6: **return** $bel(x_t)$
7: **end procedure**

La figure II.3 illustre une situation où l'algorithme de Markov est appliqué. Dans cette illustration, le robot se déplace dans un monde en 1 dimension. Le robot est capable de se déplacer vers la droite ou la gauche. Il peut déterminer avec une certaine probabilité s'il se trouve devant une porte ou non. Il peut aussi déterminer avec une certaine probabilité la position dans laquelle il se trouve à l'aide des déplacements qu'il a effectués. Dans l'image « a », le robot n'a encore effectué aucun déplacement ni observation et n'a aucune information initiale sur sa position. Il a donc une probabilité uniforme de se trouver sur n'importe quel point de la carte. Dans l'image « b », le robot observe qu'il se trouve devant une porte. Cette observation permet au robot de déduire qu'il est devant une des trois portes de la carte. La probabilité autour des portes augmente en conséquence. Dans l'image « c », le robot se déplace vers la droite. Ce qui implique de déplacer également la fonction de la croyance de sa position initiale. Ce déplacement implique une diminution de la croyance de sa position due aux erreurs d'estimation du déplacement. Cette diminution de la croyance est représentée par un aplatissement des différentes gaussiennes. Dans l'image « d », le robot découvre à nouveau une porte. Ce qui augmente encore sa croyance en sa position. Et finalement, l'image « e », démontre encore une fois que les déplacements diminuent la croyance de la position du robot.

La prédictibilité de l'environnement est un élément important dans le choix d'appliquer ou non des algorithmes probabilistes. Dans le cas d'un environnement bien structuré comme une chaine de montage, le degré d'imprédictibilité est bien moins important que lorsque le robot évolue en ville ou dans une maison. En effet, l'environnement d'une chaine de montage est beaucoup moins sujet à des éléments imprévus comme des personnes ou

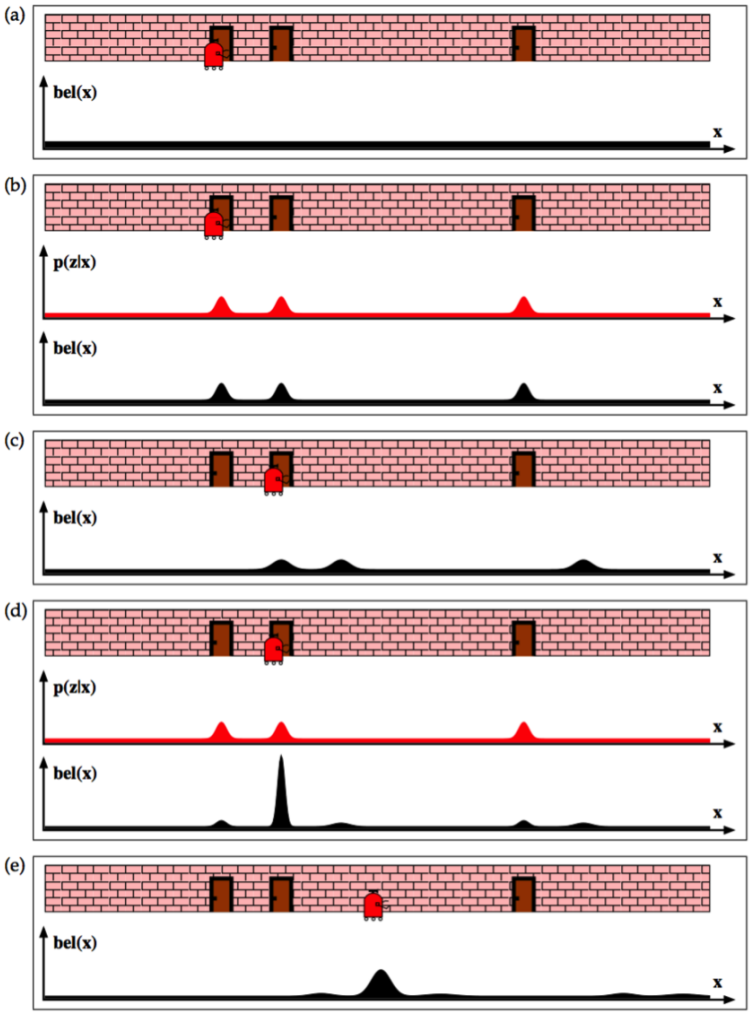

FIGURE II.3 – Idée générale de la localisation de Markov
Source: Probabilistic Robotics[22]

objets inconnus venant s'ajouter à son environnement de travail. Par la nature imprédictible des êtres vivants, l'imprédictibilité de l'environnement augmente fortement lorsque le robot évolue dans un environnement en présence d'êtres vivants. Les algorithmes probabilistes permettent de pallier à l'imprédictibilité d'un environnement. Lorsque l'environnement est fortement imprédictible, il est souvent plus prudent d'augmenter le nombre de capteurs. Des capteurs de proximité d'un robot de chaine de montage peuvent être ajoutés pour éviter un accident si une personne rentre dans le champ de mouvement du robot. Cependant, dans le cas des robots de chaines de montage, la vitesse d'exécution du robot est très importante pour la productivité de l'entreprise. Il est donc préférable de ne pas diminuer cette vitesse avec des vérifications de sécurité et plutôt d'interdire l'accès aux alentours de la zone de travail du robot.

Les algorithmes probabilistes souffrent de deux défauts importants. La première est la complexité en temps de calcul de l'algorithme qui augmente. En effet, dans les algorithmes probabilistes on considère toute la fonction de densité de probabilité lorsque les algorithmes classiques (non probabilistes) ne considèrent qu'un élément. La deuxième est le besoin d'utiliser des approximations de la densité de probabilité exacte. Considérer la fonction de densité exacte devient vite impossible à calculer et est donc indispensable d'utiliser des approximations comme des gaussiennes ou un nombre restreint des éléments de la fonction de densité. Dans certaines situations, ces représentations peuvent être éloignées de la réalité. Cependant, l'augmentation de la puissance de calcul des processeurs ainsi que les recherches d'algorithmes plus efficients permettent de grandes évolutions dans le domaine. Toutefois, ces deux points restent encore problématiques. Ces deux points sont donc discutés dans la présentation des algorithmes de localisation suivants.

II.2 Algorithmes de résolution du problème de localisation

L'algorithme de Markov permet de donner l'idée générale des algorithmes de localisation. Cependant pour pouvoir implémenter concrètement un algorithme de localisation un certain nombre de questions sont encore ouvertes. La plus importante correspond à la représentation de la fonction de probabilité. Deux grandes approches existent. La première définit une loi de probabilité à l'aide de ses paramètres. Et la seconde représente la probabilité à l'aide d'un certain nombre d'éléments discrets. Les sections suivantes discutent de ces deux approches. Elles présentent les algorithmes EKF et MCL qui font

partie des algorithmes de localisation les plus connus et qui présentent de bons résultats en pratique.

II.2.1 Algorithme paramétrique(EKF)

Les algorithmes paramétriques permettent de représenter les croyances de la position d'un robot à l'aide de lois de probabilité. Les concepts mathématiques de l'algorithme de Kalman ont été développés dans les années 60 [12]. L'algorithme de Kalman n'est pas uniquement utilisé dans la localisation en robotique. Il aussi par exemple utilisé dans la restauration d'image 2D [20], ou bien dans la gestion de la congestion du trafic routier [14] . Dans le cas de l'algorithme Kalman Filter(KF) 2 la loi de probabilité est une loi normale. Elle dépend donc de deux paramètres, son espérance μ et son écart type Σ. Cette loi normale est une loi normale multivariée lorsque le vecteur de position est composé de plusieurs éléments. Dans le cas de notre robot, on définit alors la moyenne de cette fonction multivariée comme suit

$$\mu = x_t = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$$

L'écart type est défini comme suit dans le cas de notre robot :

$$\Sigma = Var(\vec{x_t}) = \begin{pmatrix} Var(x) & Cov(x,y) & Cov(x,\theta) \\ Cov(y,x) & Var(y) & Cov(y,\theta) \\ Cov(\theta,x) & Cov(\theta,y) & Var(\theta) \end{pmatrix}$$

Cette matrice permet d'exprimer les erreurs sur la possition du robot. Les valeurs de la matrice augmentent durant le parcourt du robot et diminue lorsque le robot effectue une observation qui lui permet d'augmenter sa certitude en sa possition. Une explication détaillée de ce à quoi correspond cette matrice et comment il est possible de la représenté est donnée dans la section VI.2.4.

L'algorithme de Kalman est composé des deux mêmes étapes principales que l'algorithme de Markov, la prédiction et la mise à jour. La prédiction consiste à déplacer la moyenne de la gaussienne en fonction des mouvements du robot et d'adapter l'écart type en fonction de l'erreur associé au mouvement. L'étape de mise a jour permet de mettre a jour la moyenne et l'écart type à l'aide des observations et des erreurs de mesures liées à l'observations. L'importance associée à l'observation est formulée à l'aide du gain de Kalman exprimé à la ligne 4 de l'algorithme.

La matrice R_t et Q_t correspondent respectivement à la covariance de la fonction gaussienne centré des erreur de mouvement et des erreurs d'observations. R_t est de dimensions $n \times n$ où n est la taille du vecteur de position x_t. Q_t est de dimensions $k \times n$ où k est la dimension du vecteur d'obsevation z_t. La matrice A_t et B_t sont respectivement de dimensions $n \times n$ et $n \times m$ où m correspond à la dimension du vecteur de contrôle. La matrice B permet d'exprimer la transformation utile pour exprimer le vecteur de contrôle dans les mêmes variable que le vecteur de position μ. De même la matrice C de dimension $k \times n$ permet d'exprimer la transformation utile pour exprimer le vecteur de postion dans les mêmes variables que le vecteur d'observation.

Dans l'algorithme de Kalman l'ensemble des opérations sont effectuées sur des matrices. Les matrices manipulée devienne de grande taille lorsque la posiiton x_t et donc la moyenne μ_t devient de taille importante. Rappelons que x_t représente la position du robot. Dans le cas d'un robot avec plusieurs articulations, la position dans l'espace de chaque articulation est contenue dans ce vecteur. Ce qui montre que la taille de ce vecteur peut augmenter rapide. Deplus, la taille de la matrice de covariance est le carré de la taille de se vecteur de position. Il est donc important d'utiliser des librairie performante dans la manipulation de matrice pour évidé de rendre cette algorithme inutilisable. Finalement, certaine propriété de ces matrices permet de diminuer le temps de calcul. L'inversion de $(C_t\overline{\Sigma}_t C_t^T + Q_t)^{-1}$ qui est une matrice symétrique peut être réalisé à l'aide de la décompostion de Cholesky [25].

En appliquant les recommandations de manipulations de matrice, l'algorithme du filtre de Kalman a une bonne complexité. $O(n^{2.4} + k^{2.4})$ où k représente la taille du vecteur d'observation z_t et n la taille du vecteur de position x_t. Dans beaucoup d'application des algorithmes de localisation, la taille de n est tendance à être supérieure à celle de k. La complexité est donc dominé par $O(n^{2.4})$

Algorithm 2 Kalman filter

1: **procedure** KALMANFILTER($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)
2: $\overline{\mu_t} \leftarrow A_t\mu_{t-1} + B_t u_t$ ▷ prédiction
3: $\overline{\Sigma_t} \leftarrow A_t + \Sigma_{t-1}A_t^T + R_t$ ▷ prédiction
4: $K_t \leftarrow \overline{\Sigma}_t C_t^T (C_t\overline{\Sigma}_t C_t^T + Q_t)^{-1}$ ▷ Kalman Gain
5: $\mu_t \leftarrow \overline{\mu}_t + K_t(z_t - C_t\overline{\mu}_t)$ ▷ mise à jour
6: $\Sigma_t \leftarrow (I - K_t C_t)\overline{\Sigma}_t$ ▷ mise à jour
7: **return** μ_t, Σ_t
8: **end procedure**

L'algorithme Extended Kalman filter(EKF) correspond à la version non linéaraire de l'algorithme du filtre de Kalman. Cette variante a été développée

Algorithm 3 Extended Kalman filter

1: **procedure** ExtendedKalmanFilter $(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$
2: $\overline{\mu_t} \leftarrow g(u_t, \mu_{t-1})$ ▷ prédiction
3: $\overline{\Sigma_t} \leftarrow G_t \Sigma_{t-1} G_t^T + R_t$ ▷ prédiction
4: $K_t \leftarrow \overline{\Sigma}_t H_t^T (H_t \overline{\Sigma}_t H_t^T + Q_t)^{-1}$ ▷ Kalman Gain
5: $\mu_t \leftarrow \overline{\mu}_t + K_t(z_t - h(\overline{\mu}_t))$ ▷ mise à jour
6: $\Sigma_t \leftarrow (I - K_t H_t)\overline{\Sigma}_t$ ▷ mise à jour
7: **return** μ_t, Σ_t
8: **end procedure**

quelques années plus tard par la NASA[21] pour faire face au fait que la plupart des systèmes physiques ne sont pas linéaires. Pour ce faire les fonctions g et h ont été introduites. Ces fonctions ne doivent pas obligatoirement être linéaire mais doivent être dérivables. Contrairement ou filtre de Kalman classique où les fonctions sont obligatoirement linéaire pour préserver des fonctions de répartition gausienne.

L'algorithme 4 décrit l'algorithme de Kalman appliqué à la localisation de notre robot. Le robot utilise une carte composé de repère (comme défini dans le section I.1) pour se localisé. Cette carte est passé en paramètre en plus des paramètres classique de l'algorithme de Kalman.

II.2.2 Algorithme non paramétrique(MCL)

À l'inverse des algorithmes paramétriques, les algorithmes non paramétriques ne sont pas basés sur une loi de probabilité connue dont l'algorithme arrange les paramètres pour correspondre au mieux à la croyance de la position. Dans les algorithmes non paramétriques, la croyance est représentée par nombre déterminé de positions supposées. Une probabilité est associée à ces positions supposées. Plusieurs techniques existent pour représenter ses positions supposées.

La première technique consiste à découper la carte de l'environnement en une grille où chaque élément de la grille correspond à une position supposée[23]. Pour représenter l'orientation du robot, il faut multiplier le nombre de cases par le nombre d'angles d'orientation que peut prendre le robot (voir la figure II.5). Dans cette représentation, seules trois orientations sont possibles. Ces trois orientations correspondent aux trois plans de la représentation. Comme on peut s'en rendre compte, il est très important de définir la bonne granularité de la découpe. Une découpe trop importante augmente le temps de calcul, alors qu'une grille trop peu découpée rend la localisation trop peu précise. Dans ce type de découpe, plus la carte est grande et plus le temps de calcul

Algorithm 4 EKF localisation

procedure EKF LOCALISATION $(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t, m)$

$\theta_{t-1} \leftarrow \mu_{t-1,\theta}$

$G_t \leftarrow \begin{pmatrix} 1 & 0 & -d_t \sin(\theta_{t-1} + \gamma_t) \\ 0 & 1 & d_t \cos(\theta_{t-1} + \gamma_t) \\ 0 & 0 & 1 \end{pmatrix}$

$V_t \leftarrow \begin{pmatrix} \cos(\theta_{t-1} + \gamma_t) & -d \sin(\theta_{t-1} + \gamma_t) \\ \sin(\theta_{t-1} + \gamma_t) & d_t \cos(\theta_{t-1} + \gamma_t) \\ 0 & 1 \end{pmatrix}$

$M_t \leftarrow \begin{pmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{pmatrix}$

$\overline{\mu}_t \leftarrow \mu_{t-1} + \begin{pmatrix} d_t \cos(\theta_{t-1} + \gamma_t) \\ d_t \sin(\theta_{t-1} + \gamma_t) \\ \gamma_t \end{pmatrix}$ ▷ prédiction

$\overline{\Sigma}_t \leftarrow G_t \Sigma_{t-1} G_t^T + V_t M_t V_t^T$ ▷ prédiction

$Q \leftarrow \begin{pmatrix} \sigma_{d^z}^2 & 0 \\ 0 & \sigma_{\rho^z}^2 \end{pmatrix}$

for all observed features $z_t^i \leftarrow (d_t^i, \rho_t^i)^T$ **do**

$j \leftarrow$ landmark observed

$q \leftarrow (m_{j,x} - \overline{\mu}_{t,x})^2 + (m_{j,y} - \overline{\mu}_{t,y})^2$

$\hat{z}_t^i \leftarrow \begin{pmatrix} \sqrt{q} \\ atan2(m_{j,y} - \overline{\mu}_{t,y}, m_{j,x} - \overline{\mu}_{t,x}) - \overline{\mu_{t,\theta}} \end{pmatrix}$

$H_t^i \leftarrow \begin{pmatrix} -\frac{m_{j,x} - \overline{\mu}_{t,x}}{\sqrt{q}} & -\frac{m_{j,y} - \overline{\mu}_{t,y}}{\sqrt{q}} & 0 \\ \frac{m_{j,y} - \overline{\mu}_{t,y}}{q} & -\frac{m_{j,x} - \overline{\mu}_{t,x}}{q} & -1 \end{pmatrix}$

$S_t^i \leftarrow H_t^i \overline{\Sigma_t} [H_t^i]^T + Q_t$

$K_t^i \leftarrow \overline{\Sigma}_t [H_t^i]^T [S_t^i]^{-1}$ ▷ Kalman Gain

$\overline{\mu}_t \leftarrow \overline{\mu}_t + K_t^i (z_t^i - \hat{z}_t^i))$ ▷ mise à jour

$\overline{\Sigma}_t \leftarrow (I - K_t^i H_t^i) \overline{\Sigma}_t$ ▷ mise à jour

end for

$\mu_t \leftarrow \overline{\mu}_t$

$\Sigma_t \leftarrow \overline{\Sigma}_t$

return μ_t, Σ_t

end procedure

FIGURE II.4 – Illustration de MCL
Source: Wikipedia, Auteur : Daniel Lu

est important.

Dans le cas de l'algorithme de Monte Carlo localization (MCL) aussi appelé Particle filter localization[19] car il utilise un filtre à particule, une approche différente a été choisie. Dans cet algorithme (voir l'algorithme 5) la croyance de la position est représentée par M particules(voir la figure II.4). Chaque particule est considérée comme une hypothèse sur la position du robot. Les traits gris correspondent aux particules et le niveau de rouge représente la probabilité associée. Plus une région de la carte contient de particules avec une probabilité haute associée et plus la probabilité que le robot s'y trouve est grande. Contrairement aux algorithmes basés sur la découpe de la carte en une grille, MCL n'implique pas un temps de calcul supplémentaire lorsque la taille de la carte augmente. Cependant, il est possible de choisir d'augmenter le nombre de particules pour augmenter la précision.

MCL souffre d'un problème important, en particulier quand $M < 50$ et que l'environnement du robot est grand. Il peut arriver que l'ensemble des particules converge vers une position erronée. Une fois cette convergence atteinte il est difficile d'en sortir. Pour pallier à ce problème, à chaque itération un certain nombre de particules sont redistribuées aléatoirement dans la carte.

II.2.3 Comparaison de MCL et EKF

Le tableau II.1 donne et compare les principales caractéristiques de l'algorithme EKF et MCL. Ce tableau comparatif permet de mettre en valeur qu'aucun algorithme n'est globalement meilleur. Ils possèdent chacun leurs qualités et défauts. Les défauts de l'un se révèlent les qualités de l'autre. Par exemple, EKF est efficient en temps et mémoire contrairement au MCL dont l'efficience dépend fortement du nombre de particules.

Cependant, MCL est plus robuste que EKF. En effet, la représentation de la fonction de probabilité à l'aide d'une loi normale permet à EKF d'être efficient. Cependant, le revers de cette efficience est qu'il est moins robuste lorsque la fonction de probabilité est fortement différente d'une loi normale.

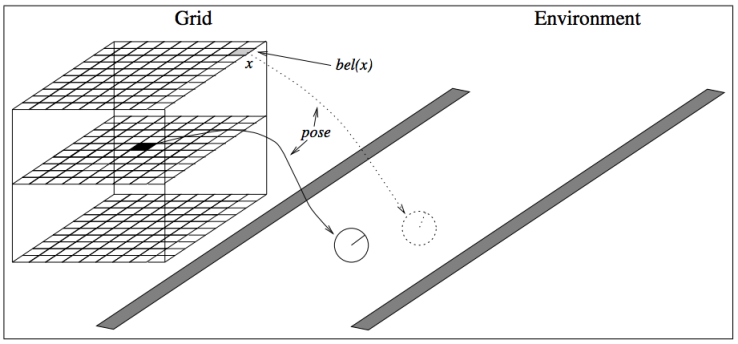

FIGURE II.5 – Illustration de la carte en grille
Source: Probabilistic Robotics[22]

Algorithm 5 MCL

procedure MCL $(\mathcal{X}_{t-1}, u_t, z_t, m)$
  $\overline{\mathcal{X}_t} \leftarrow \emptyset$
  $\mathcal{X}_t \leftarrow \emptyset$
  for $m = 1$ to M **do**
    $x_t^{[m]} \leftarrow sample\_motion\_model(u_t, x_{t-1}^{[m]})$
    $w_t^{[m]} \leftarrow measurement\_model(z_t, x_t^{[m]}, m)$
    $\overline{\mathcal{X}_t} \leftarrow \overline{\mathcal{X}_t} + \langle x_t^{[m]}, w_t^{[m]} \rangle$
  end for
  for $m = 1$ to M **do**
    draw i with probability $\propto w_t^{[i]}$
    add $x_t^{[i]}$ to $\mathcal{X}_t$
  end for
  return $\mathcal{X}_t$
end procedure

Prenons l'exemple d'un long couloir avec un grand nombre de portes et où le robot n'est pas capable de distinguer les portes. Dans cette situation MCL peut donner de meilleures performances. En effet, EKF assume que la croyance de la position est proche d'une distribution gaussienne et a donc de mauvaises performances lorsque la croyance correspond plutôt à distribution multimodale. Pour pallier à ce problème, l'algorithme classique EKF a été amélioré. Multi-hypothesis tracking (MHT)[4] filtre permet de représenter la croyance à l'aide d'un mixte de plusieurs gaussienne et donc d'avoir un algorithme plus robuste et efficient. Le même type de problème se présente pour la représentation de l'erreur de mesure. Dans le cas de MCL, cette erreur de mesure ne doit pas obligatoirement suivre une gaussienne contrairement à EKF. MCL permet de représenter l'erreur de mesure par une fonction quelconque ce qui permet de s'approcher plus facilement de la réalité physique.

La localisation globale correspond à un problème de localisation où la position initiale n'est pas connue et l'incertitude est donc grande à cet instant. Il s'avère qu' EKF est plus approprié pour suivre la position d'un robot dont on connait déjà la position initiale. En effet, une représentation unimodale est généralement une bonne représentation dans un problème où il s'agit de suivre une position, mais pas dans un problème de localisation globale. La linéarisation dans EKF ne fait qu'accroitre ce problème en risquant de converger vers une mauvaise position.

De plus, MCL permet de traiter directement dans l'algorithme des mesures brutes or EKF nécessite des repères. Il est donc possible à l'aide de MCL d'utiliser directement les valeurs de capteurs de distance entre le robot et des murs pour les comparer avec une carte représentant les murs. Ce qui n'est pas possible à l'aide de EKF qui nécessite une carte composée d'un nombre limité de repères qui permet de localiser le robot à l'aide des repères mesurés dans son environnement. Finalement, en pratique il s'avère que MCL est plus simple à implémenter qu'EKF.

	EKF	MCL
Mesures	Repères	Brute
Erreur de Mesure	Gaussienne	Toute
Posterior	Gaussienne	Particules
Efficience(mémoire)	++	+
Efficience(temps)	++	+
Facilité d'implémentation	+	++
Résolution	++	+
Robuste	-	++
Localisation Globale	non	oui

TABLE II.1 – Comparaison EKF et MCL

Chapitre III

Simultaneous Localization And Mapping

Les algorithmes de type Slam (Simultaneous Localization And Mapping) sont l'étape suivante de l'indépendance des robots. En effet, dans les algorithmes de simple localisation, la carte de l'environnement doit être construite avant de la passer en paramètre aux algorithmes de localisation. Comme son nom l'indique, dans un algorithme Slam la carte est construite en parallèle avec la localisation du robot. La section suivante présente l'EKF SLAM qui est l'extension de l'EKF localisation. Le FastSlam est un autre algorithme SLAM connu qui est l'extension du MCL. Les particles dans le FastSlam sont composée d'une estimation du chemin du robot et de l'ensemble des estimations des points de repères de la carte avec leur covariance associée. La description détaillée de cette algorithmes est disponibles dans le livre «Probabilistic robotics»[22]

III.1 EKF SLAM

EKF SLAM est un algorithme SLAM et comme sont nom le laisser penser est basé sur l'algorithme de localisation EKF. Il utilise également une gaussienne pour représenter son état. Cependant l'état contient à l'instant t en plus de la pose du robot, la carte qui est représentées comme l'ensemble des poses des repères découvert à l'instant t. Cet algorithme est aussi découpé en deux étapes important, l'étape de prédiction ainsi que l'état de correction. Dans l'état de prédiction la fonction de mouvement n'influence pas les valeurs de la moyenne des poses des repères. Ce qui est normal car les repères ne bougent pas même si le robot change de position.

Algorithm 6 EKF SLAM

procedure EKF SLAM $(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t, m)$

$\theta_{t-1} \leftarrow \mu_{t-1,\theta}$

$F_x \leftarrow \begin{pmatrix} 1 & 0 & 0 & \underbrace{0 \dots 0}_{2N} \\ 0 & 1 & 0 & \\ 0 & 0 & 1 & \end{pmatrix}$

$\overline{\mu}_t \leftarrow \mu_{t-1} + F_x^T \begin{pmatrix} d_t \cos(\theta_{t-1} + \gamma_t) \\ d_t \sin(\theta_{t-1} + \gamma_t) \\ \gamma_t \end{pmatrix}$ ▷ prédiction

$G_t \leftarrow I + F_x^T \begin{pmatrix} 0 & 0 & -d_t \sin(\theta_{t-1} + \gamma_t) \\ 0 & 0 & d_t \cos(\theta_{t-1} + \gamma_t) \\ 0 & 0 & 0 \end{pmatrix} F_x$

$\overline{\Sigma}_t \leftarrow G_t \Sigma_{t-1} G_t^T + F_x^T R_t F_x$ ▷ prédiction

$Q_t \leftarrow \begin{pmatrix} \sigma_{d^z}^2 & 0 \\ 0 & \sigma_{\rho^z}^2 \end{pmatrix}$

for all observed features $z_t^i \leftarrow (d_t^i, \rho_t^i)^T$ **do**

$j \leftarrow$ landmark observed

if landmark j never seen before **then**

$\begin{pmatrix} \overline{\mu}_{j,x} \\ \overline{\mu}_{j,y} \end{pmatrix} \leftarrow \begin{pmatrix} \overline{\mu}_{t,x} \\ \overline{\mu}_{t,y} \end{pmatrix} + \begin{pmatrix} d_t^i cos(\rho_t^i + \overline{\mu}_{t,\theta}) \\ d_t^i sin(\rho_t^i + \overline{\mu}_{t,\theta}) \end{pmatrix}$

end if

$F_{x,j} \leftarrow \begin{pmatrix} 1 & 0 & 0 & \underbrace{0 \dots 0}_{2j-2} & 0 & 0 & \underbrace{0 \dots 0}_{2N-2j} \\ 0 & 1 & 0 & & 0 & 0 & \\ 0 & 0 & 1 & & 0 & 0 & \\ 0 & 0 & 0 & & 1 & 0 & \\ 0 & 0 & 0 & & 0 & 1 & \end{pmatrix}$

$q \leftarrow (m_{j,x} - \overline{\mu}_{t,x})^2 + (m_{j,y} - \overline{\mu}_{t,y})^2$

$\hat{z}_t^i \leftarrow \begin{pmatrix} \sqrt{q} \\ atan2(m_{j,y} - \overline{\mu}_{t,y}, m_{j,x} - \overline{\mu}_{t,x}) - \overline{\mu_{t,\theta}} \end{pmatrix}$

$H_t^i \leftarrow \begin{pmatrix} -\frac{m_{j,x} - \overline{\mu}_{t,x}}{\sqrt{q}} & -\frac{m_{j,y} - \overline{\mu}_{t,y}}{\sqrt{q}} & 0 & \frac{m_{j,x} - \overline{\mu}_{t,x}}{\sqrt{q}} & \frac{m_{j,y} - \overline{\mu}_{t,y}}{\sqrt{q}} \\ \frac{m_{j,y} - \overline{\mu}_{t,y}}{q} & -\frac{m_{j,x} - \overline{\mu}_{t,x}}{q} & -1 & -\frac{m_{j,y} - \overline{\mu}_{t,y}}{q} & \frac{m_{j,x} - \overline{\mu}_{t,x}}{q} \end{pmatrix} F_{x,j}$

$S_t^i \leftarrow H_t^i \overline{\Sigma_t} [H_t^i]^T + Q_t$

$K_t^i \leftarrow \overline{\Sigma}_t [H_t^i]^T [S_t^i]^{-1}$ ▷ Kalman Gain

$\overline{\mu}_t \leftarrow \overline{\mu}_t + K_t^i (z_t^i - \hat{z}_t^i))$ ▷ mise à jour

$\overline{\Sigma}_t \leftarrow (I - K_t^i H_t^i) \overline{\Sigma}_t$ ▷ mise à jour

end for

$\mu_t \leftarrow \overline{\mu}_t$

$\Sigma_t \leftarrow \overline{\Sigma}_t$

return μ_t, Σ_t

end procedure

Chapitre IV

Algorithmes de recherche du meilleur chemin

Les algorithmes de recherche de meilleur chemin consistent à déterminer le chemin entre un point de départ et un point d'arrivée qui sont définis sur la carte. Bien entendu, il est souhaitable que ce chemin ne conduise pas littéralement le robot droit dans le mur ou ne le fasse pas tomber dans les escaliers dans le cas d'un robot aspirateur domestique. Ce chemin est défini par un ensemble de points que doit suivre le robot pour se déplacer du point de départ jusqu'au point d'arrivée. Dans la figure IV.1 les obstacles sont représentés par la couleur grisée et les positions libres qui permettent au robot de se déplacer sans percuter d'objet sont représentées en blanc. Il est donc évident que le chemin représenté n'est pas souhaitable, car celui-ci risque de faire percuter le robot avec un objet de son environnement. Les sections suivantes décrivent comment construire un graphe à l'aide d'une telle carte. Une fois ce graphe construit, il est possible d'appliquer les algorithmes classiques de recherche de chemin dans un graphe. Les plus connus sont les algorithmes A star et Dijkstra. Il est donc possible d'utiliser les connaissances disponibles dans le domaine de la théorie des graphes qui est un domaine relativement bien connu.

IV.1 Graphe

Pour pouvoir déterminer un chemin qui ne risque pas de percuter les objets de l'environnement, ces algorithmes nécessitent de posséder une ou plusieurs cartes de l'environnement du robot. Ces cartes peuvent avoir été générées dynamiquement par le robot ou bien construites au préalable. Elles serviront à produire un graphe où les noeuds représentent des positions possibles du

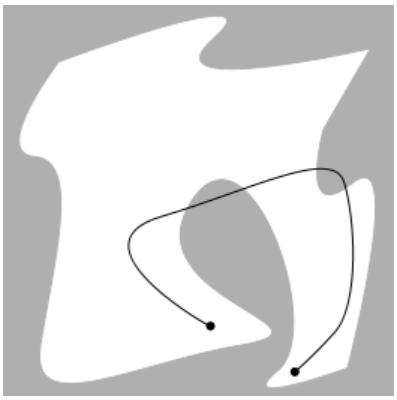

FIGURE IV.1 – Chemin non valide
Source: Wikipedia, Auteur : Simeon87

robot et les arrêtent les chemins qui permettent de rejoindre ces différentes positions. La figure IV.2 présente le graphe construit à l'aide de la carte de la figure IV.1 . Pour construire ce graphe à partir de cette carte, une technique est de découper l'environnement du robot en une carte grillagée où chaque élément de la grille prend une valeur occupée ou libre selon qu'un objet se trouve à l'intérieure ou non. Les arêtes du graphe représentent le lien entre deux cases adjacentes libres dans la grille. Un cout d'une unité est associé à ces arêtes. Le cout correspond à la distance entre chaque noeud. Si une case n'est pas libre, aucun lien ne la relit. Ce qui représente qu'il ne faut pas passer par cette case pour définir le chemin du robot. Il est ainsi possible de passer de case en case pour déterminer le chemin entre un point de départ et un point d'arrivée. Le cout total du chemin correspond à la somme des couts des arêtes empruntées pour aller du point de départ jusqu'au point d'arrivée. Il est également possible de déterminer si un point d'arrivée n'est pas joignable. Ce qui se produit lorsqu'il est impossible d'y accéder sans percuter des objets de l'environnement.

Plus la grille à un découpage important, et plus le chemin retourné est précis. Cependant, un découpage plus important augmente le temps de calcul de façon exponentielle. En plus d'un paramètre qui permet de déterminer la granularité de la carte, un paramètre qui permet de définir la distance à laquelle le robot peut être proche des murs doit être défini. Cette valeur permet de prendre en considération la largeur du robot ainsi que ses capacités de rotations. En effet, il est commun que la position du robot défini le centre du robot, cependant celui-ci possède une largeur qu'il faut prendre en considération pour éviter de percuter les murs avec les côtés du robot. Ce

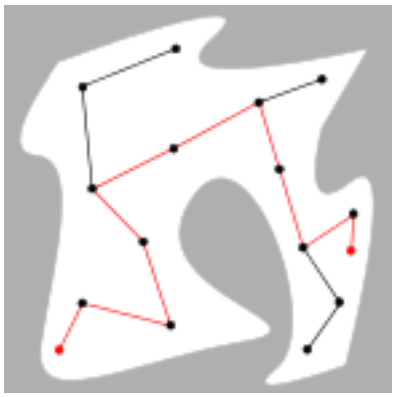

FIGURE IV.2 – Chemin valide construit à l'aide d'un graphe
Source: Wikipedia, Auteur : Simeon87

paramètre permet donc de construire les chemins avec des noeuds qui sont toujours à une distance minimale des murs.

IV.2 Dijkstra

L'algorithme de Dijkstra publié en 1959 [6] par son inventeur du même nom permet de résoudre le problème du plus court chemin en une complexité polynomiale en théorie des graphes. Dans le cas de la recherche d'un chemin valide pour un robot, cet algorithme prend en paramètre le graphe défini dans la section précédente et retourne le plus court chemin valide. Une condition supplémentaire sur le graphe est nécessaire pour appliquer Dijkstra. Le graphe doit être connexe pour trouver le plus court chemin, c'est-à-dire qu'entre chaque sommet du graphe il doit exister un chemin. Dans le cas contraire, il est évidemment impossible de déterminer un plus court chemin entre deux positions non joignables l'une à l'autre.

L'algorithme 7 correspond au pseudocode de l'algorithme de Dijkstra. Il est composé de deux parties principales. La première permet de définir la distance entre le noeud de départ et l'ensemble des noeuds du graphe. L'algorithme procède de façon itérative pour définir successivement les distances entre le noeud de départ et les noeuds les plus proches. La seconde partie permet de reconstruire le chemin entre les deux noeuds en partant du noeud de fin jusqu'à arriver au noeud de départ. L'algorithme de Dijksta est robuste et trouve toujours le meilleur chemin. Cependant, lorsque la taille du graphe devient importante et qu'on souhaite trouver rapidement le plus court chemin il est intéressant d'utiliser l'algorithme A star qui est en moyenne plus rapide

que Dijkstra. Cependant dans le pire des cas ils ont une complexité identique. La section suivante décrit donc l'algorithme A star.

Algorithm 7 Dijkstra

```
1: procedure DIJKSTRA (G = (S, A), S_deb, S_fin)
2:     Initialiser tous les sommets comme non marqué et donné un valeur
   +∞ à tous les labels L
3:     L(S_deb) ← 0
4:     while Il existe un sommet non marqué do
5:         choisir et marquer le sommet a non marqué de plus petit label L
6:         for all sommet b non marqué voisin de a do
7:             if L(b) > L(a) + v(a, b) then
8:                 L(b) ← L(a) + v(a, b)
9:                 b.prédédent ← a
10:            end if
11:        end for
12:    end while
13:    S_n ← S_fin
14:    while S_n ! = S_deb do
15:        chemin ← chemin + S_n
16:        S_n ← S_n.précédent
17:    end while
18:    return chemin
19: end procedure
```

IV.3 A star

L'algorithme A star a été publié en 1968 [9] et fonctionne de façon relativement semblable à l'algorithme de Dijkstra, mais celui-ci permet d'obtenir de meilleure performance que Dijksta grâce à l'utilisation d'une heuristique. Cette heuristique donne une estimation du cout jusqu'aux destinations recherchées. Pour appliquer A star son heuristique doit être admissible c'est-à-dire que la fonction heuristique ne doit jamais surestimer la valeur réelle du cout. Dans la recherche du plus court chemin valide de notre robot, l'heuristique représente la distance à vole d'oiseau qui est toujours plus courte ou égale à n'importe quel chemin. L'algorithme 8 correspond au pseudocode de l'algorithme A star. L'algorithme A star comme celui de Dijkstra est découpé en deux parties principales qui consistent dans la première partie à déterminer les plus courts chemins entre un sommet initial et les autres sommets, suivi

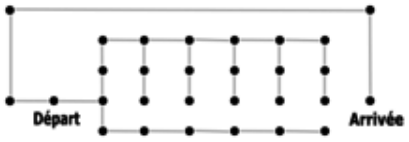

FIGURE IV.3 – A star dans un labyrinthe
Source: Wikipedia, Auteur : Bayo

de la reconstruction du chemin entre le sommet d'arrivée et le sommet de départ.

Intuitivement l'algorithme A star composé de l'heuristique estimant la distance à vole d'oiseau explore itérativement le noeud joignable et étant estimés par l'heuristique comme le plus proche du noeud cible. Lorsque ce noeud est choisi, ces voisins deviennent joignables et l'heuristique estime la distance entre ses voisins et le noeud cible. Et ce itérativement jusqu'au noeud cible. Dans un labyrinthe cet algorithme peut avoir des performances moins bonnes. La figure IV.3 illustre cette situation. Il est clair qu' à vol d'oiseau le noeud à gauche du départ est le chemin unique et le plus court pour atteindre l'arrivée. Cependant l'algorithme A star, va commencer par explorer les noeuds à droite du départ pour arriver jusqu'au cul-de-sac et finalement explorer le noeud à gauche du départ. Cependant, l'algorithme fini toujours par retourner le chemin le plus court.

Algorithm 8 A star

```
1: procedure A STAR (G = (S, A), S_start, S_goal)
2:     closedSet ← ∅                                   ▷ les noeuds déjà évalués
3:     openSet ← {S_start}
4:     came_from ← ∅
5:     g_score ← +∞                                    ▷ +∞ pour tous les éléments
6:     g_score[S_start] ← 0
7:     f_score ← +∞
8:     f_score[S_start] ← g_score[S_start] + h(S_start, S_goal)
9:     while openset !=∅ do
10:        S_current ←node with lowest f_score[]
11:        if S_current == S_goal then
12:            return reconstruct_path(came_from, S_goal)
13:        end if
14:        openSet ← openSet − S_current
15:        closedSet ← closedSet + S_current
16:        for all S_neighbor ∈ neighbor_nodes(S_current) & ∉ closedSet do
17:            t_g_score ← g[S_current] + dist(S_current, S_neighbor)
18:            if S_neighbor ∉ openSet||t_g_score < g_score[S_neighbor] then
19:                came_from[S_neighbor] ← S_current
20:                g_score[S_neighbor] ← t_g_score
21:                f_score[S_neighbor] ← g_score[S_neighbor] + h(S_neighbor, S_goal)
22:                if S_neighbor ∉ openSet then
23:                    openSet ← openSet + S_neighbor
24:                end if
25:            end if
26:        end for
27:    end while
28:    return failure
29: end procedure
30: procedure reconstruct_path(came_from, S_current)
31:    path ← {S_curentl}
32:    while S_current ∈ came_from do
33:        S_current ← came_from[S_current]
34:        path ← path + S_current
35:    end while
36:    return path
37: end procedure
```

Deuxième partie

Cas pratique

Cette partie décrit l'implémentation de l'algorithme EKF sur la brique EV3. En plus de cette description de l'algorithme, cette partie aide le lecteur qui souhaite développer un algorithme sur son propre robot. Une description de la brique EV3, de ses capteurs et des possibilités de programmation qu'offre la brique EV3 est donnée pour aider ce lecteur dans cette aventure. De plus des conseils sur la construction du robot sont également donnés pour permettre au développeur de construire un bon châssis pour son robot. Finalement, des alternatives à la brique EV3 sont données. En effet, cette brique est relativement couteuse et elle ne répond pas toujours au besoin du développeur. C'est pourquoi cette partie contient une présentation de quelques simulateurs en robotique ainsi que la présentation des microcontrôleurs Arduino et Raspberry Pi.

Chapitre V

Lego Mindstorms

Les Legos sont des jouets de construction fabriqués par le groupe danois «the Lego Group». La série Mindstorms correspond à la gamme « robotique programmable » de Legos. Cette série est vendue en kits qui permettent de réaliser certains modèles de robots prédéfinis et par la suite de laisser cours à son imagination. Les kits contiennent une brique intelligente programmable, un ensemble de capteurs et de moteurs ainsi qu'un ensemble d'éléments de constructions qui proviennent de la gamme Lego Technic. Ces kits permettent de réaliser rapidement et à couts modérés des robots simples. Ils se sont donc vite révélés comme des outils intéressants pour l'apprentissage de la robotique et de la programmation.

V.1 Description du hardware

Le kit Lego peut être comparé à des packs composés de microcontrôleurs Arduino [1] ou Raspberry Pi [2] et d'une série de capteurs et de moteurs. Le microcontrôleur Arduino et Raspberry Pi possède l'avantage de pouvoir utiliser une plus grande gamme des capteurs et moteurs à moindres couts. Cependant, la mise en place de ces capteurs se révèle de plus bas niveau et donc plus longue. Contrairement au kit Lego où les moteurs et capteurs sont de type «plug and play». Ce projet est plus basé sur les aspects de programmation en robotique que sur les aspects hardware, c'est pourquoi le kit Lego Mindstorms a été favorisé.

1. Arduino : www.arduino.cc
2. Raspberry Pi : www.raspberrypi.org

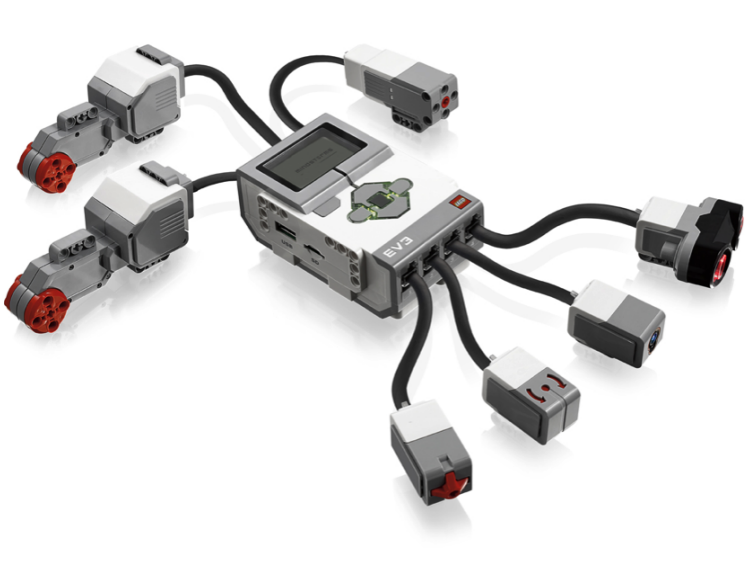

FIGURE V.1 – Brique EV3
Source: www.4erevolution.com

V.1.1 Brique intelligente

La brique intelligente programmable (voir Figure V.1.1) est le cerveau des robots EV3. Elle est dotée d'une interface à six boutons lumineux qui changent de couleur pour indiquer l'état d'activité de la brique, d'un affichage à haute résolution noir et blanc, d'un hautparleur intégré, d'un port USB, d'un lecteur de cartes mini SD, de quatre ports d'entrée et de quatre ports de sortie. La brique prend également en charge la communication USB, Bluetooth et Wifi avec un ordinateur. Son interface programmable permet à la fois de programmer et de journaliser des données directement sur la brique. Elle est compatible avec les appareils mobiles et est alimentée par des piles AA ou par la batterie CC rechargeable EV3. La brique EV3 fournit l'énergie pour les capteurs et les moteurs. Elle permet de récupérer et de traiter les informations des capteurs et d'envoyer des commandes aux moteurs par le biais des ports d'entrée et de sortie. Ce qui rend la connexion et l'utilisation des capteurs et des moteurs très facile.

Depuis son lancement la gamme de Lego Mindstorms a connu trois types de briques qui se sont succédé, la brique RXC, NXT et EV3. Elles ont progressivement augmenté de puissance de calcul. La vraie révolution de la brique EV3 est la possibilité de booter le système sur une carte SD. Ce qui augmente ainsi le potentiel de la brique (comme décrit dans la section V.2).

La brique EV3 embarque un processeur arm9, 16Mo de mémoire flash et 64Mo de RAM. La carte SD permet d'étendre la mémoire à 32Go. Il est vrai que la puissance a augmenté dans la brique EV3 en comparaison avec ses prédécesseurs. Toutefois, cette puissance de calcul reste réduite en comparaison aux ordinateurs et smartphones actuels.

V.1.2 Moteurs et capteurs

Le pack de base de la bique EV3 est composé de deux moteurs moyens, un petit moteur, un capteur de pression, un capteur de distance infrarouge et d'un capteur de couleurs. Il est possible d'acheter des capteurs Legos supplémentaires tels qu'un gyroscope, un capteur de distance ultrasonique et une boussole. Il est également possible d'acheter d'autre type de capteurs développés par des sociétés indépendantes de Lego. C'est possible grâce au support de protocole standard que doivent adopter les capteurs de données. On peut par exemple citer I^2C qui est un bus de données conçu par Philips et qui est très répandu dans le monde de l'électronique. On voit donc en vente des capteurs GPS, des capteurs RFID, des capteurs d'humidité qui sont compatible avec la brique EV3

La partie qui suit décrit plus en profondeur les capteurs et moteurs du kit de base qui sont ceux disponibles pour réaliser ce projet. Les descriptions sont basées sur les informations données par le constructeur. Les descriptions sont dirigées vers le type de données reçues ainsi que la précision des capteurs et des moteurs. Ces descriptions sont importantes dans ce mémoire, car elles permettent de mieux connaitre les capteurs et les moteurs à notre disposition ce qui permet ainsi de mieux définir les erreurs qui sont propres à ces capteurs.

Le capteur infrarouge EV3 détecte la proximité d'objets (jusqu'à 70 cm) et lit les signaux émis par la balise infrarouge EV3 (distance max de 2m). Les utilisateurs peuvent créer des robots télécommandés, faire des courses d'obstacles et se familiariser avec l'utilisation de la technologie infrarouge dont les télécommandes des TV, les systèmes de surveillance.

Le capteur de couleur numérique EV3 différencie huit couleurs. Il peut être utilisé aussi comme capteur photosensible en mesurant l'intensité lumineuse. Les utilisateurs peuvent construire des robots qui trient selon les couleurs ou suivent une ligne, expérimenter la réflexion de la lumière de différentes couleurs et se familiariser avec une technologie largement répandue dans les secteurs industriels du recyclage, de l'agriculture et de l'emballage.

Le capteur tactile analogique EV3 est un outil simple, mais extrêmement précis, capable de détecter toute pression ou relâchement de son bouton frontal. Les utilisateurs pourront construire des systèmes de commande marche/arrêt, créer des robots capables de s'extraire d'un labyrinthe et découvrir l'utilisation

de cette technologie dans des appareils tels que les instruments de musique numériques, les claviers d'ordinateur ou l'électroménager.

Le grand servomoteur EV3 est un puissant moteur avec retour tachymétrique pour un contrôle précis au degré près. Grâce à son capteur de rotation intégré, ce moteur intelligent peut être synchronisé avec les autres moteurs d'un robot pour rouler en ligne droite à la même vitesse. Il peut également être utilisé pour fournir une mesure précise pour des expériences. Par ailleurs, la forme du boitier facilite l'assemblage des trains d'engrenage.

Le servomoteur moyen EV3 est parfait pour des charges moins importantes, des applications à vitesse plus élevée, ainsi que des situations où une plus grande réactivité et un plus petit profil sont nécessaires lors de la conception du robot. Le moteur se sert d'un retour tachymétrique pour un contrôle précis au degré près et possède un capteur de rotation intégré.

V.2 Description du software

C'est dans la brique EV3 qu'il est possible d'injecter un programme contenant les instructions du robot. Lego fournit un langage de programmation graphique qui s'appelle RoboLab(voir V.2). Il est le fruit de la collaboration de Lego avec le M.I.T.. Il se révèle un bon langage de programmation pour l'apprentissage de la programmation. Cependant, il n'est pas approprié au développement d'algorithme complexe de localisation comme EKF ou bien MCL. Les briques Mindstorm ont vite été hackées pour permettre de développer des programmes à l'aide d'autre langage que RoboLab. Dans la version EV3, il est possible de booter sur une carte SD. Ce qui a permis que des OS tels que Linux soient adaptés pour fonctionner sur la brique. Le projet ev3Dev est un exemple d'adaptation de Linux pour la brique EV3. Il est donc maintenant potentiellement possible de développer à l'aide de tous les langages disponibles sur Linux. Lejos est une API écrite en Java comme le laisse supposer son nom. Lejos est l'outil principalement utilisé dans ce mémoire, une description plus complète est donc donnée dans une section dédiée.

V.3 Simulateurs

Pour les personnes qui ne souhaitent pas acheter de matériels ou pour des tests à plus grande échelle, il est possible d'utiliser des simulateurs. Un simulateur comme Gazebo [3] permet d'utiliser ou de construire des modèles de

3. Gazebo : gazebosim.org

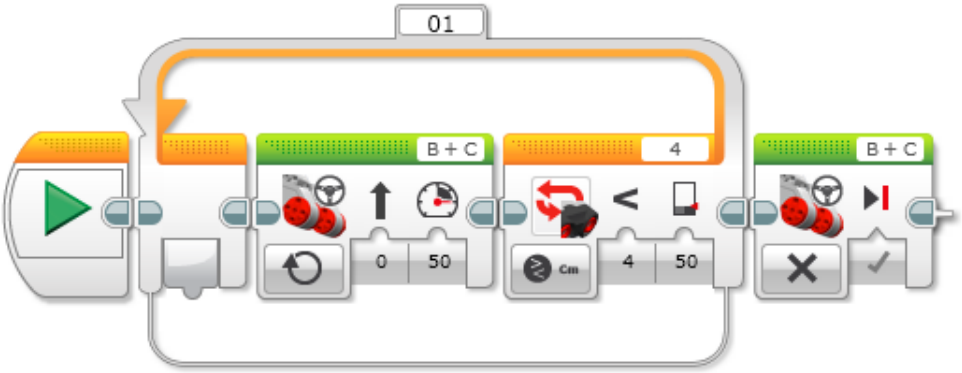

FIGURE V.2 – Robolab
Source: www.lego.com

robots qui évoluent dans un environnement prédéfini. Ce projet open source à reçu des financements de la DRC [4] pour l'adaptation à son robot ATLAS. Cependant, ce type de simulateur nécessite un temps d'apprentissage certain pour pouvoir commencer à l'utiliser. C'est pourquoi d'autres simulateurs plus simples de la brique de Lego existent. L'université de Berne a développé EV3JLIB [5] qui permet de lancer gratuitement un simulateur en ligne avec le comportement du robot codé en Java. Ce code à l'avantage d'être réutilisable sur la brique. D'autres logiciels destinés à l'apprentissage de la robotique à l'aide d'un simulateur de la brique EV3 et plus évolués graphiquement que EV3JLIB sont disponibles en ligne. Les deux principaux sont «virtual robotics toolkit» [6]et «Robot virtual worlds» [7] mais ils sont payants.

V.4 Lejos

Lejos est un microprogramme libre destiné à remplacer le microprogramme originalement installé sur la brique Lego Mindstorm. Il est disponible sur les briques RCX, NXT et EV3. Lejos inclut une machine virtuelle Java permettant de développer des robots dans le langage Java. Ce qui a donné le nom Lejos qui est le mot Legos où le « g » a été remplacé par un « j » pour Java. Pour la version EV3 de la brique Lego, cette machine virtuelle Java est

4. DRC : www.theroboticschallenge.org
5. EV3JLIB : www.aplu.ch
6. Virtual robotics toolkit : cogmation.com
7. Robot virtual worlds : www.robotvirtualworlds.com

mise à disposition par Oracle. Lejos permet un accès facile aux moteurs et aux capteurs. Ce qui est possible grâce aux routines écrites en c et mise dans le domaine public par le groupe Lego. En effet, Lego à décider de publier le code source de la brique EV3 en open source. Ces routines en c permettent en théorie d'utiliser n'importe quel langage de programmation pour développer des comportements de robots.

Depuis son lancement jusqu'à aujourd'hui Lejos dispose d'une communauté active et de taille assez importante. Le dépôt officiel du code source est sourceforge.net. Entre le 01-01-2015 et le 09-04-2015 la version 0.9.0-beta de la brique EV3 a été téléchargée 2700 fois. En effet, l'outil Lejos est couramment utilisé dans les universités et dans les hautes écoles pour l'apprentissage de la robotique et du langage Java. L'implémentation orientée objet permet aux étudiants d'étudier la robotique avec le niveau d'abstraction qu'ils désirent. Il est ainsi possible de développer des algorithmes de localisation sans se soucier des adresses hexadécimales des moteurs et des capteurs.

Lejos est également utilisé dans la recherche. De nombreux projets ont déjà vu le jour. Des chercheurs de l'université de Porto au Portugal ont déjà utilisé Lejos pour implémenter un algorithme qui cartographie l'environnement d'un robot [17]. Encore à l'université de Porto un simulateur simple et des exercices d'utilisation de l'algorithme EKF ont été développés [8] par les professeurs de l'université de Porto. Un étudiant suédois a développé un Slam [9] et des outils de visualisation de la carte et de la position du robot. Finalement, un projet qui permet de définir le comportement d'un robot à l'aide de diagramme d'état est disponible sur github [10]. L'ensemble de ses projets prouve que Lejos possède de bonnes bases pour développer des algorithmes complexes.

8. EKF : paginas.fe.up.pt/ robosoc
9. Slam : penemunxt.blogspot.be
10. Diagramme d'états : github.com/jabrena/liverobots

Chapitre VI

Implémentation d'EKF

Cette section contient la description de mon implémentation de l'algorithme EKF dans la librairie Lejos. Par la suite, l'implémentation de l'algorithme EKF est comparée à l'implémentation MCL déjà présente dans la librairie Lejos.

VI.1 Description du robot construit

La plateforme du robot est de type « differential wheeled robot »(voir VI.1). Ce qui consiste en deux moteurs indépendants positionnés de façons opposées sur le robot. Ce choix de plateforme est commun en robotique. Cette plateforme est simple à mettre en oeuvre et fournit une amplitude de mouvement importante. Les plateformes de type steering sont semblables aux voitures classiques. Elle demande un mécanisme plus complexe et leur amplitude de mouvement est moindre et donc moins adaptée à la robotique.

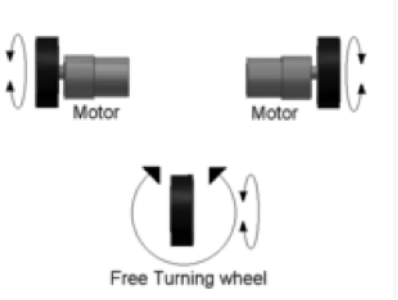

FIGURE VI.1 – Differential wheeled robot
Source: Wikipedia, Auteur : Patrik

FIGURE VI.2 – Robot

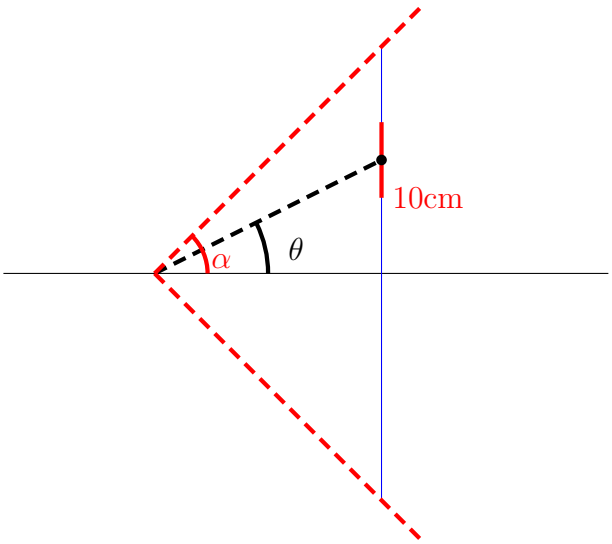

FIGURE VI.3 – Évaluation de la distance des codes QR

VI.2 Implémentation de l'algorithme EKF

VI.2.1 Détection de Feature avec la caméra du smartphone

Les codes QR sont des éléments faciles à identifier pour la caméra d'un smartphone. De nombreuses librairies de qualité ont déjà été développées pour détecter et décoder des codes QR. Zbar [1] est une de ces librairies open source. Elle est disponible sur Android et IOS. Pour ce mémoire, une application permettant d'estimer la distance du smarphone au code QR a été développée à l'aide de la librairie Zbar. Pour pouvoir utiliser les codes QR pour estimer la distance entre eux et le smartphone, les codes QR doivent être d'une dimension donnée (dans ce mémoire : un carré de 10cm de coté). La librairie Zbar renvoie la dimension du code QR en nombre de pixels captés par la caméra. À l'aide d'un étalonnage de la caméra qui consiste à déterminer l'ouverture de l'objectif et du calcul trigonométrique suivant, il est possible de déterminer la distance des codes QR de 10cm de coté(voir VI.3).

$$Distance = \frac{\frac{CapteurResolutionHorizontale}{MesureNombrePixelsHorinzontale} * LargeurCodeQR}{2 * \tan(\alpha)}$$

1. Zbar : zbar.sourceforge.net

Il est également possible de déterminer l'angle entre la direction du robot et le centre du code QR à l'aide de la formule suivante :

$$\theta = \alpha - \frac{\alpha * 2 * CentreCodeQRPixel}{CapteurResolutionHorizontale}$$

si le code QR se trouve à droite du robot la formule devient :

$$\theta = \frac{\alpha * 2 * CentreCodeQRPixel}{CapteurResolutionHorizontale} - \alpha$$

L'étalonnage consiste à déterminer α à l'aide de mesures faites à distance connue.

VI.2.2 Les cartes

La carte qui stocke les positions des codes QR et qui est utilisée par l'algorithme EKF est stockée dans une image au format SVG. Ce format consiste à définir des éléments graphiques simples dans un fichier XML. Les codes QR sont donc représentés par une ligne de 10cm de longueur. La figure VI.4 représente cette carte où les codes QR sont représentés en rouge et les murs en noir. Les murs sont également définis dans un fichier SVG différent. Cette décomposition des cartes est volontaire et elle permet de charger uniquement les sous-cartes utiles à l'algorithme. Il n'est par exemple pas utile pour l'algorithme MCL d'avoir la carte composée des codes QR.

En plus, de ces deux cartes qui ont été générées à la main au préalable. Une carte dynamique de type grille d'occupation est générée à l'aide du capteur infrarouge et du capteur de pression positionnée à l'avant du robot.

VI.2.3 Pseudo-code

L'implémentation EKF utilise la technique de détection de codes QR comme features présentées dans la section VI.2.1.

Cette implémentation d'EKF complète la librairie Commons Math [2] qui est une librairie mathématique open-source de Apache. Celle-ci contient une série de classe permettant de manipuler et d'appliquer des opérations sur des matrices. Ce qui se révèle très utile dans les algorithmes de Kalman. Elle contient également une implémentation du filtre de Kalman, mais ne contient pas d'implémentation du Extended Kalman Filter.

2. Commons Math : commons.apache.org/proper/commons-math

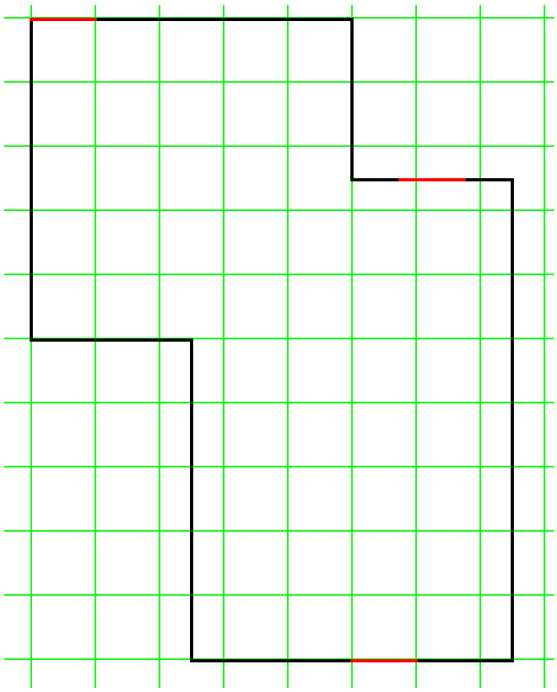

FIGURE VI.4 – carte EKF

VI.2.4 Représentation de la matrice de covariance

La matrice de covariance est comme son nom l'indique une matrice qui permet de représenter la covariance entre chaque variable de la position du robot. Les valeurs contenues dans cette matrice sont très utiles, mais restent fastidieuses à lire, car cette matrice change dynamiquement et régulièrement. Il a donc été important d'implémenter une représentation graphique des informations importantes de cette matrice de covariance. Cette représentation permet de se faire une idée rapide de ces valeurs, sans devoir les analyser une à une. La figure VI.5 présente la représentation choisie de la covariance. La moyenne donne la position estimée du robot. Cette position estimée est représentée par le point bleu central qui correspond à la position x, y estimée du robot et la droite bleue au milieu des deux autres correspond à la direction estimée du robot. L'ellipse autour de la position ainsi que les deux autres droites permettent de représenter la matrice de covariance. Voici la définition de la covariance pour mieux comprendre ce que représente la covariance et comprendre comment sont construites cette ellipse et ces droites :

$$Cov(X, Y) = E[(X - E[X])(Y - E[Y])]$$

où $E[]$ désigne l'espérance mathématique. La covariance caractérise la variation simultanée des deux variables aléatoires X, Y. Elle est positive lorsque la différence entre les variables aléatoire X, Y et leur moyenne ont tendance à être de même signe et négative dans le cas contraire. Soit le vecteur de position écrit :

$$\vec{X} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$$

La matrice de covariance pour le vecteur de position est la suivante :

$$Var(\vec{X}) = \begin{pmatrix} Var(x) & Cov(x,y) & Cov(x,\theta) \\ Cov(y,x) & Var(y) & Cov(y,\theta) \\ Cov(\theta,x) & Cov(\theta,y) & Var(\theta) \end{pmatrix}$$

La diagonale de la matrice de covariance est composée des variances des variables aléatoires de $\vec{X}$ ce qui est normal, car $Cov(X,X) = Var(X)$. La matrice de covariance est une matrice symétrique, car $Cov(X,Y) = Cov(Y,X)$. Pour revenir à la représentation de la matrice de covariance, l'angle d'écartement entre les deux droites de notre représentation est donné par $Var(\theta)$ ce qui caractérise donc la dispersion des valeurs de la direction du robot. Plus cette variance est petite et plus la direction estimée du robot est sure et inversement plus elle est grande et plus la direction est incertaine. L'ellipse est définie à l'aide de la sous-matrice suivante :

$$\begin{pmatrix} Var(x) & Cov(x,y) \\ Cov(y,x) & Var(y) \end{pmatrix}$$

Cette technique[3] qui permet de visualiser la covariance d'une matrice à l'aide d'une ellipse peut être appliqué à n'importe quelle matrice de covariance. $Var(x)$ et $Var(y)$ permettent de définir la largeur et la hauteur de l'ellipse à l'aide de l'équation de l'ellipse suivante :

$$\left(\frac{x}{Var(x)}\right)^2 + \left(\frac{y}{Var(y)}\right)^2 = 1$$

Il faut maintenant déterminer l'orientation de l'ellipse. Lorsque $Cov(x,y) = 0$ l'orientation de l'ellipse est inchangée. De façon générale l'angle d'orientation peut être défini par la formule suivante :

$$\alpha = arctan2(V_1.y, V_1.x)$$

3. Ellipse représentation : www.visiondummy.com

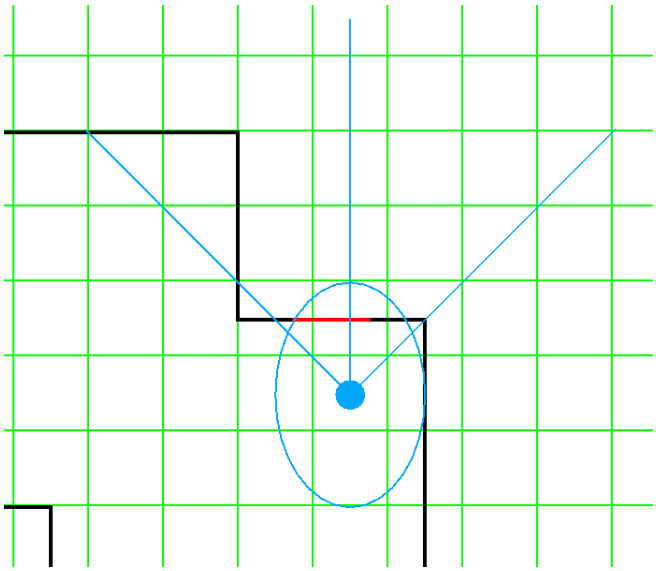

FIGURE VI.5 – Représentation de la covariance

où V_1 correspond au vecteur propre majeur et α correspond à l'angle entre V_1 et l'axe des x. Trouver le vecteur propre majeur consiste à résoudre l'équation suivante :

$$A\vec{v} = \lambda(\vec{v})$$

où A correspond à la matrice de covariance, v le vecteur propre et λ la valeur propre. Cette équation est résolue à l'aide de la libraire Commons Math déjà utilisée pour manipuler les matrices de l'algorithme EKF. Cette équation possède deux solutions. Le vecteur majeur correspond au vecteur qui possède la plus grande valeur propre.

VI.2.5 Tests et résultats de l'implémentation de EKF

VI.3 Comparaison avec le MCL

Troisième partie

Conclusion

Cette partie finale est composée d'une conclusion ainsi que des propostion de recherches futures pour continuer ce mémoire. La conclusion porte aussi bien sur les notions théorique des algorithme de localisation que sur les problèmes pratique rencontré au cours de mémoire. En effet, sans considéré la pratique il est impossible d'avoir des algorithmes de localisation de qualité.

Chapitre VII

Conclusion

Bien que les concepts globaux théoriques des algorithmes semblent à première vue simples, il s'avère qu'implémenter de tels algorithmes de localisation sur un robot réel demande de résoudre une quantité importante de sous problèmes. En effet, un algorithme de localisation n'est pas un élément isolé, il doit être incorporé dans un tout cohérent. Il est donc primordial d'avoir une compréhension d'ensemble ainsi qu'une compréhension détaillée de l'implémentation du robot. Il est également important de garder en tête l'objectif du robot et déterminer l'environnement dans lequel le robot évolue pour déterminer l'algorithme qui correspond le mieux à ces caractéristiques.

Chapitre VIII

Recherches futures

Ce mémoire n'a fait qu'effleurer la localisation d'un robot mobile dans son environnement, ce qui n'est qu'un sous domaine de l'intelligence artificielle dans la robotique. Il y a donc un grand nombre de possibilités pour continuer ce mémoire. Les sections suivantes présentes celle qui me semble les plus intéressantes.

VIII.1 Analyse d'images

Dans ce mémoire le choix d'utiliser des codes QR a permis de simplifier la détection de repère pour l'algorithme EKF. Cependant, il n'est pas possible d'utiliser des codes QR dans toutes les situations. Il est donc important de savoir extraire des repères des données brutes des capteurs. De nombreux algorithmes permettent d'extraire des repère à l'aide d'une caméra. Ce domaine de recherche discute des différents éléments qui sont considérés comme de bons repères. Par exemple, la détection de bord se base sur de fortes modifications de la lumière dans une image, l'ensemble des points où cette forte modification de lumière apparait est considéré comme un bord. L'algorithme de Canny est un exemple d'algorithme de détection de bords [5].De la même manière les algorithmes de détection de coins tentent de déterminer les caractéristiques d'un coin. L'algorithme de Moravec [16] et de Harris et Stephens [8] sont des exemples d'algorithmes de détection de coins.

VIII.2 Ajout d'IA dans Lejos

Dans un but pédagogique, il serait intéressant de continuer à développer la librairie Lejos. Rappelons qu'elle est utilisée par de nombreuses écoles

et universités. Elle fournit déjà un nombre important de classes pour manipuler un robot. Cet outil est donc parfait pour développer en pratique des aspects théoriques. Cependant, elle souffre du peu de classe permettant de développer une intelligence artificielle. Il serait donc intéressant d'implémenter des algorithmes de localisation utilisant d'autres types de capteurs, ou bien des algorithmes de prise de décision pour atteindre un objectif. Ces algorithmes pourraient être assemblés pour construire un tout cohérent. De plus cette librairie est écrite en Java et avec un bon niveau de décomposition des éléments, elle est donc facilement réutilisable dans d'autres projets. Elle pourrait ainsi devenir un outil semblable à ROS [1] ou MSRDS [2] qui sont des outils professionnels de haute qualité fonctionnant aussi bien sur des robots industriels que des robots de loisirs comme les drones.

1. ROS : www.ros.org
2. Microsoft Robotics Developer Studio 4 : www.microsoft.com

Bibliographie

[1] Nissan car drives and parks itself at ceatec. *BBC* (2013).

[2] Toyota sneak previews self-drive car ahead of tech show|publisher. *BBC* (2013).

[3] Bessière, P., Mazer, E., Ahuactzin-Larios, J. M., and Mekhnacha, K. *Bayesian Programming*. CRC Press, Dec. 2013.

[4] Blackman, S. Multiple hypothesis tracking for multiple target tracking. *Aerospace and Electronic Systems Magazine, IEEE 19*, 1 (Jan 2004), 5–18.

[5] Canny, J. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (1986).

[6] Dijkstra, E. W. A short introduction into the art of programming. 67–73. circulated privately.

[7] francaise, A. Rectifications de l'orthographe. *www.academie-francaise.fr* (1990).

[8] Harris, C., and Stephens, M. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference* (1988), pp. 147–151.

[9] Hart, P. E., Nilsson, N. J., and Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science, and Cybernetics SSC-4*, 2 (1968), 100–107.

[10] Hauweele, P. Slam : un problème de localisation et de cartographie simultanées.

[11] Ichbiah, D. Le robot domestique, du rêve à la réalité. *futura-sciences.com* (2009).

[12] Kalman, R. E., and Bucy, R. S. New results in linear filtering and prediction theory. *Trans. ASME, Ser. D, J. Basic Eng* (1961), 109.

[13] Kanade, T., Thorpe, C., and Whittaker, W. Autonomous land vehicle project at cmu. In *Proceedings of the 1986 ACM Fourteenth Annual Conference on Computer Science* (New York, NY, USA, 1986), CSC '86, ACM, pp. 71–80.

[14] Kotsialos, A., and Papageorgiou, M. The importance of traffic flow modeling for motorway traffic control. *Networks and Spatial Economics 1*, 1-2 (2001), 179–203.

[15] Markoff, J. Google cars drive themselves, in traffic. *The New York Times* (2010).

[16] Moravec, H. Obstacle avoidance and navigation in the real world by a seeing robot rover. In *tech. report CMU-RI-TR-80-03, Robotics Institute, Carnegie Mellon University doctoral dissertation, Stanford University*, no. CMU-RI-TR-80-03. September 1980.

[17] Oliveira, G., Silva, R., Lira, T., and Reis, L. P. Environment mapping using the lego mindstorms nxt and lejos nxj.

[18] osha. Industrial robots and robot system safety. *www.osha.gov*.

[19] Rekleitis, I. A particle filter tutorial for mobile robot localization. Tech. rep., Centre for Intelligent Machines, Montreal, Quebec, Canada, McGill University, 2004.

[20] Saint-Felix, D., Du, X., and Demoment, G. Image deconvolution using 2-d non-causal fast kalman filters. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '86.* (Apr 1986), vol. 11, pp. 2455–2458.

[21] Smith, G. Application of statistical filter theory to the optimal estimation of position and velocity on board a circumlunar vehicle. *National Aeronautics and Space Administration* (1962).

[22] Thrun, S., Burgard, W., and Fox, D. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents).* The MIT Press, 2005.

[23] Vu, T.-D., Burlet, J., and Aycard, O. Grid-based localization and online mapping with moving objects detection and tracking : new results. In *Intelligent Vehicles Symposium, 2008 IEEE* (June 2008), pp. 684–689.

[24] Wallace, R., Stentz, A. T., Thorpe, C., Moravec, H., Whittaker, W. R. L., and Kanade, T. First results in robot road-following. In *Proceedings of the International Joint Conference on Artificial Intelligence* (1985).

[25] Wikipedia. Cholesky decomposition — wikipedia, the free encyclopedia, 2015. [Online ; accessed 3-August-2015].