

# 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





## 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. Je remercie particulièrement ma soeur qui m'a aidé à réaliser le logo ci-dessus qui symbolise ce mémoire.

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 Localization
- 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 à l'aide de composants à faible coût. La limite de ces composants y est discutée. 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 actionneurs, 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. L'ajout du smartphone permet de se servir de son puissant processeur et de ses capteurs. En effet, les smartphones actuels possèdent un grand nombre de capteurs à coûts 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], ou encore 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 haute précision. 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 coûts. 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<sup>1</sup> : « 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

---

1. ATILF (Analyse et traitement informatique de la langue française) : [www.atilf.fr](http://www.atilf.fr)

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 une 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, « 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'UMons prône fortement les valeurs de la réduction de l'impact environnemental des universités qui sont de grandes consommatrices de papier. Elle a même reçu un prix pour son engagement<sup>2</sup>. L'impression recto verso de ce mémoire est donc en accord avec ces valeurs.

---

2. UMonS diminution de l'impact environnemental : [portail.umons.ac.be](http://portail.umons.ac.be)

# Table des matières

<b>I</b>	<b>Localisation d'un robot dans son environnement</b>	<b>1</b>
<b>I</b>	<b>Les cartes</b>	<b>5</b>
I.1	Carte composée de repères . . . . .	5
I.2	La grille d'occupation . . . . .	6
<b>II</b>	<b>Problème de localisation</b>	<b>9</b>
II.1	Définition du problème . . . . .	9
II.1.1	Explication du problème . . . . .	9
II.1.2	Modèle de mouvement et d'observation . . . . .	10
II.1.3	Algorithme de localisation de Markov . . . . .	14
II.2	Algorithmes de résolution du problème de localisation . . . . .	18
II.2.1	Algorithme paramétrique(EKF) . . . . .	18
II.2.2	Algorithme non paramétrique(MCL) . . . . .	23
II.2.3	Comparaison de MCL et EKF . . . . .	27
<b>III</b>	<b>Simultaneous Localization And Mapping</b>	<b>29</b>
III.1	EKF SLAM . . . . .	29
<b>IV</b>	<b>Algorithmes de recherche du meilleur chemin</b>	<b>33</b>
IV.1	Graphe . . . . .	33
IV.2	Dijkstra . . . . .	35
IV.3	$A^*$ . . . . .	36
<b>II</b>	<b>Implémentations</b>	<b>39</b>
<b>V</b>	<b>Lego Mindstorms</b>	<b>43</b>
V.1	Description du hardware . . . . .	43
V.1.1	Brique intelligente . . . . .	44
V.1.2	Moteurs et capteurs . . . . .	45
V.1.3	Simulateurs . . . . .	46

V.2	Description du software . . . . .	47
V.2.1	RoboLab . . . . .	47
V.2.2	Lejos . . . . .	47
V.3	Description du robot construit . . . . .	49
<b>VI</b>	<b>Algorithme de localisation</b>	<b>53</b>
VI.1	Algorithme EKF . . . . .	53
VI.1.1	Odométrie . . . . .	53
VI.1.2	Détection de repères . . . . .	54
VI.1.3	Les cartes . . . . .	55
VI.1.4	Implémentation EKF . . . . .	56
VI.1.5	Matrice de covariance . . . . .	57
VI.2	Tests et Résultats . . . . .	59
VI.2.1	Tests et résultats de l'implémentation de EKF . . . . .	59
VI.2.2	Comparaison avec le MCL . . . . .	60
<b>VII</b>	<b>Grille d'occupation et recherche de chemin</b>	<b>63</b>
VII.1	Construction de la grille d'occupation . . . . .	63
VII.1.1	Capteur de pression . . . . .	63
VII.1.2	Capteur infrarouge . . . . .	64
VII.1.3	Trajet . . . . .	65
VII.2	Recherche de chemin . . . . .	65
<b>III</b>	<b>Conclusions</b>	<b>69</b>
<b>VIII</b>	<b>Conclusions</b>	<b>73</b>
VIII.1	Implémentation . . . . .	73
VIII.2	Composants à faible coût . . . . .	73
VIII.3	Recherche de chemin . . . . .	74
<b>IX</b>	<b>Recherches futures</b>	<b>75</b>
IX.1	Analyse d'images . . . . .	75
IX.2	Ajout d'IA dans Lejos . . . . .	75
IX.3	Prise de décisions . . . . .	76
<b>X</b>	<b>Enrichissements personnels</b>	<b>77</b>
X.1	Projets open source . . . . .	77
X.2	Domaine complet . . . . .	78
X.3	Travail de taille importante . . . . .	78



# Table des figures

I.1	Carte composée de repères . . . . .	6
I.2	Grille d'occupation . . . . .	7
II.1	Pose d'un robot dans un espace à deux dimensions . . . . .	11
II.2	Hypothèse de Markov . . . . .	12
II.3	Mouvement $u_t$ du robot . . . . .	13
II.4	Idée générale de la localisation de Markov . . . . .	17
II.5	EKF linéarisation . . . . .	22
II.6	Illustration de MCL . . . . .	26
II.7	Illustration de la carte en grille . . . . .	26
III.1	EKF SLAM . . . . .	30
IV.1	Chemin non valide . . . . .	34
IV.2	Chemin valide construit à l'aide d'un graphe . . . . .	35
IV.3	$A^*$ dans un labyrinthe . . . . .	37
V.1	Brique EV3 . . . . .	44
V.2	Robolab . . . . .	48
V.3	Differential wheeled robot . . . . .	49
V.4	Robot . . . . .	51
VI.1	Évaluation de la distance des codes QR . . . . .	55
VI.2	Carte EKF . . . . .	56
VI.3	Représentation de la covariance . . . . .	59
VI.4	Test MCL . . . . .	61
VII.1	Capteur de pression . . . . .	64
VII.2	Capteur infrarouge . . . . .	64
VII.3	Trajet . . . . .	65
VII.4	Recherche de chemin . . . . .	66
VII.5	Recherche de chemin dans la grille d'occupation . . . . .	67

# 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 de Dijkstra et  $A^*$  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 :

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

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. Cette technique permet donc de cartographier des environnements de

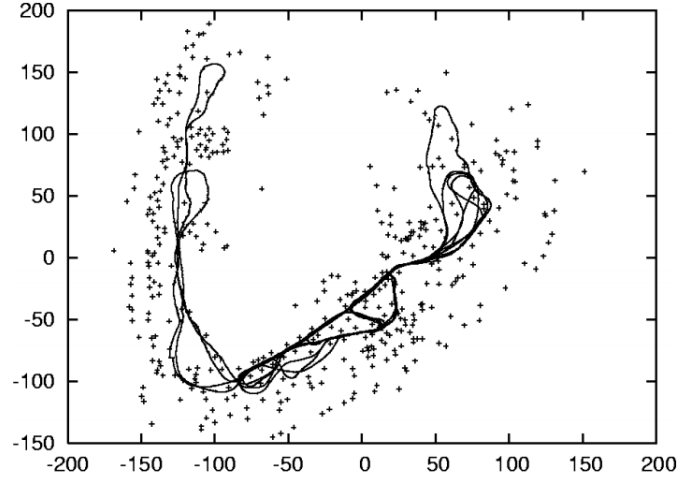


FIGURE I.1 – Carte composée de repères  
**Source:** Probabilistic Robotics [22]

grande taille en 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 [IX.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 :

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

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 a 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

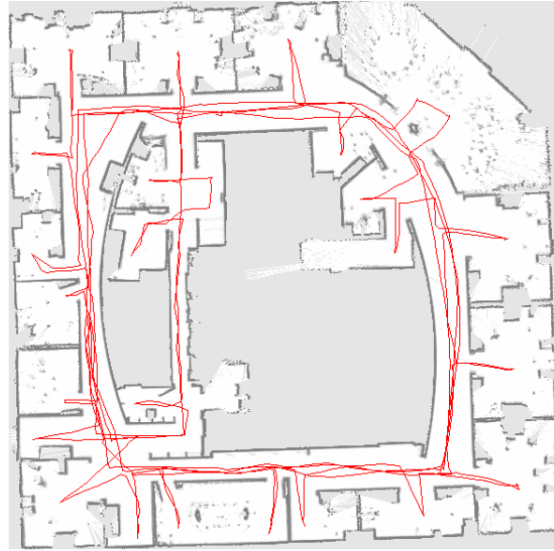


FIGURE I.2 – Grille d'occupation  
**Source:** Probabilistic Robotics [22]

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, elle 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. Ceci entraîne 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. Ceci pourrait entraîner 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. Ceci 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 pose du robot à un instant donné. Cette pose est dénotée par le vecteur  $x_t$ . Dans la suite de ce mémoire, la pose 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 peuvent être utilisés pour un espace à un

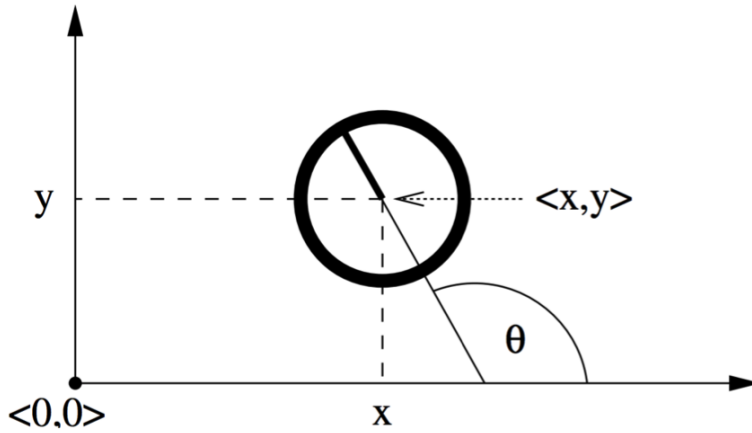


FIGURE II.1 – Pose d'un robot dans un espace à deux dimensions

**Source:** Probabilistic Robotics [22]

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 elle est souvent suffisante pour les robots mobiles. Formellement, la pose 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  $x$  et  $y$  correspondent aux coordonnées dans l'espace à deux dimensions et  $\theta$  correspond à l'orientation du robot (voir II.1). Remarque : lorsque « position » est utilisé cela se réfère plutôt aux coordonnées  $x$  et  $y$ . La dénomination « pose » est utilisée lorsque d'autres paramètres sont ajoutés à ses coordonnées.

Pour déterminer la pose 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  $z_t$ . La figure II.2 représente cette hypothèse de Markov où les noeuds grisés correspondent aux informations connues et les noeuds blancs aux informations inférées à l'aide des informations connues. Formellement, les contrôles peuvent

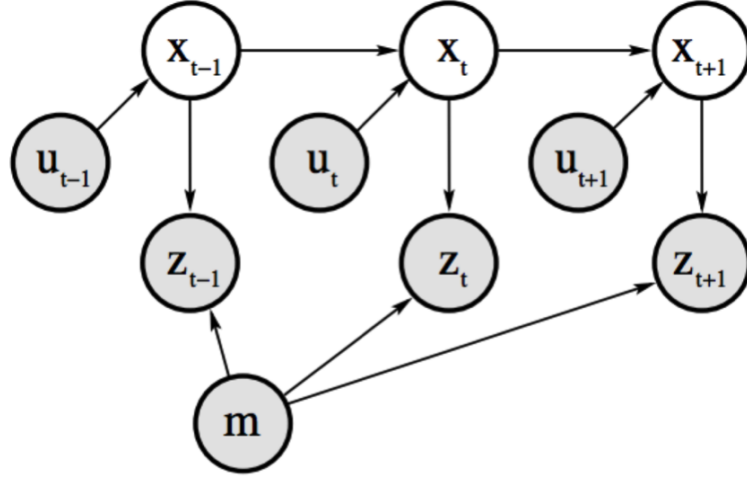


FIGURE II.2 – Hypothèse de Markov  
**Source:** Probabilistic Robotics [22]

être définis par le vecteur suivant :

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

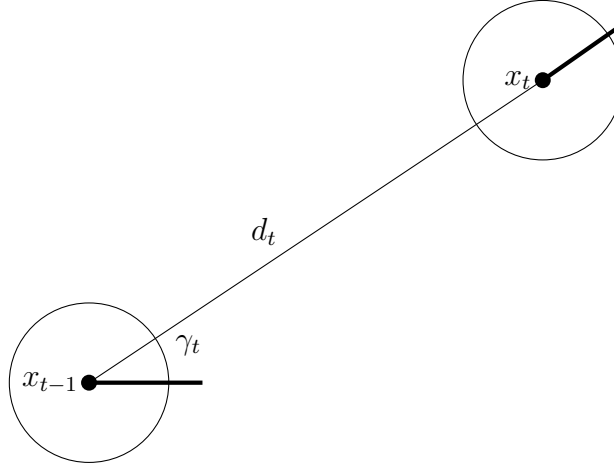
où  $d_t$  correspond à la distance parcourue et  $\gamma_t$  correspond à l'angle de rotation du robot dans l'intervalle de temps séparant deux contrôles. L'odométrie permet de déterminer les commandes  $u_t$ . Tandis que les valeurs de  $u_t$  permettent de définir itérativement les nouvelles poses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

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

La figure II.3 illustre la mise à jour de la pose à l'aide de la commande  $u_t$ . 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  $\rho_t^z$  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

FIGURE II.3 – Mouvement  $u_t$  du robot

pose 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  $z_t$ .

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

où  $m_{j,x}, m_{j,y}$  correspondent aux coordonnées de l'objet scanné  $j$ . Ces coordonnées sont stockées dans la carte. Les variables  $x_t, y_t, \theta_t$  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 sont exactes 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  $\gamma$  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  $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ , sont des paramètres à déterminer et sont spécifiques à chaque robot et  $\varepsilon_{b^2}$  correspond à une gaussienne de moyenne nulle et de variance  $b^2$ . Cette formule montre que plus la distance  $d$  et l'angle  $\gamma$  sont importants et plus l'erreur risque d'être grande. Ceci correspond bien à la réalité. Les valeurs de  $\alpha_2 d^2$  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  $\rho^m$  entre le robot et l'objet  $j$ .

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

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 pose. 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.

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

où  $E$  est l'espace d'état de la pose du robot et  $x$  une pose spécifique du vecteur aléatoire  $X$ . 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 :

$$p(x) = \sum_y p(x|y)p(y)$$

Dans le cas continu, il devient :

$$p(x) = \int p(x|y)p(y)dy$$

L'algorithme de Markov décrit donc la mise à jour de la pose  $x_{t-1}$  vers la pose  $x_t$ . Il prend en paramètre la croyance sur la pose précédente (c'est-à-dire la fonction de densité de probabilité du vecteur aléatoire  $X$  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 pose  $x_t$ . Ceci permet de déterminer la densité de probabilité associée à chaque pose 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 pose 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 d'inférence 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  $\eta$  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:      $\overline{bel}(x_t) \leftarrow \int p(x_t | u_t, x_{t-1}, m) bel(x_{t-1}) dx_{t-1}$            ▷ prédiction
4:      $bel(x_t) \leftarrow \eta p(z_t | x_t, m) \overline{bel}(x_t)$                        ▷ mise à jour
5:   end for
6:   return  $bel(x_t)$ 
7: end procedure

```

---



La figure II.4 illustre une situation où l'algorithme de Markov est appliqué. Dans cette illustration, le robot se déplace dans un monde en une 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 pose 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 pose. 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. Ceci implique de déplacer la fonction de la croyance. Le déplacement du robot implique une diminution de la confiance en la croyance de sa pose due aux erreurs d'estimation du déplacement. Cette diminution est représentée par un aplatissement des différentes gaussiennes. Dans l'image (d), le robot découvre à nouveau une porte. Ceci augmente la confiance en la croyance de sa pose. Et finalement, l'image (e), démontre encore une fois que les déplacements diminuent la confiance en la croyance de la pose 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 chaîne 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 chaîne de montage est beaucoup moins sujet à des éléments imprévus comme des personnes ou 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 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 chaîne 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 chaînes 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. Le premier est la complexité en temps de calcul de l'algorithme qui augmente. En effet, dans les algorithmes probabilistes on considère toute la fonction de

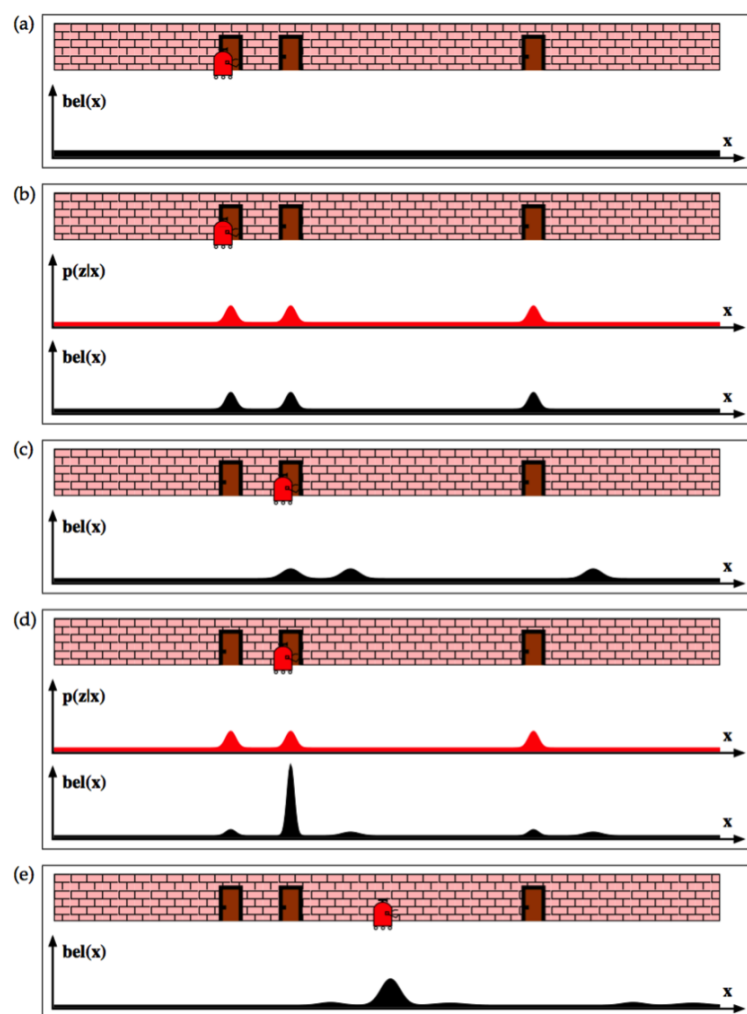


FIGURE II.4 — Idée générale de la localisation de Markov  
**Source:** Probabilistic Robotics [22]

densité de probabilité lorsque les algorithmes classiques (non probabilistes) ne considèrent qu'un élément. Le 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 il 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 efficaces permettent de grandes évolutions dans le domaine. Toutefois, ces deux points restent encore problématiques. Ils 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, il reste un certain nombre de questions auxquelles il faut encore répondre. 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 de localisation permettent de représenter la croyance de la pose d'un robot à l'aide d'une loi de probabilité. Les concepts mathématiques de l'algorithme de Kalman, qui est un algorithme paramétrique, ont été développés dans les années 60 [12]. L'algorithme de Kalman n'est pas uniquement utilisé dans la localisation en robotique. Il est 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) (voir algorithme 2) la loi de probabilité est une loi normale. Dans le cas unidimensionnel, elle dépend donc de deux paramètres, son espérance  $\mu$  et son écart type  $\sigma$  ou  $\sigma^2$  qui correspond à la variance. Cette loi normale est une loi normale multivariée lorsque le vecteur de pose est composé de plusieurs éléments. Dans le cas de notre robot, on définit alors la

moyenne de cette fonction multivariée comme suit :

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

Dans ce cas, la matrice de covariance est définie comme suit :

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

Cette matrice permet d'exprimer les erreurs sur la pose du robot. Les valeurs de la diagonale de la matrice augmentent durant le parcours du robot. Elles diminuent lorsque le robot effectue une observation qui lui permet d'augmenter sa certitude en sa pose. Une explication détaillée de la matrice de covariance est donnée dans la section VI.1.5. On y trouve également une technique pour représenter visuellement la matrice de covariance.

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. Elle adapte la matrice de covariance en fonction de l'erreur associée au mouvement. L'étape de mise à jour permet de mettre à jour la moyenne et la matrice de covariance à l'aide des observations et des erreurs de mesure liées à l'observation. L'importance associée à l'observation au temps  $t$  est formulée à l'aide du gain de Kalman exprimé à la ligne 4 de l'algorithme. Cette importance est fonction de la certitude en la pose du robot et de la certitude de l'observation.

La matrice  $R_t$  et la matrice  $Q_t$  correspondent respectivement à la covariance des fonctions gaussiennes centrées en zéro associées aux erreurs de mouvement et aux erreurs d'observation. La matrice  $R_t$  est de dimensions  $n \times n$  où  $n$  est la taille du vecteur de pose  $x_t$ . La matrice  $Q_t$  est de dimensions  $k \times n$  où  $k$  est la dimension du vecteur d'observation  $z_t$ . Les matrices  $A_t$  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  $u_t$  dans les mêmes variables que le vecteur de pose  $\mu$ . De même la matrice  $C$  de dimensions  $k \times n$  permet d'exprimer la transformation utile pour exprimer le vecteur de pose dans les mêmes variables que le vecteur d'observation.

Dans l'algorithme de Kalman, l'ensemble des opérations est effectué sur des matrices. Les matrices manipulées deviennent de grande taille lorsque la pose  $x_t$  et donc la moyenne  $\mu_t$  deviennent de taille importante. Rappelons que  $x_t$

représente la pose du robot. Dans le cas d'un robot avec plusieurs articulations, la position dans l'espace de chaque articulation est contenue dans ce vecteur. Ceci montre que la taille de ce vecteur peut augmenter rapidement. De plus, la taille de la matrice de covariance est le carré de la taille de ce vecteur de pose. Il est donc important d'utiliser des bibliothèques performantes pour la manipulation de matrice. Cela évite de rendre cet algorithme inutilisable. Finalement, certaines propriétés de ces matrices permettent de diminuer le temps de calcul. L'inversion de  $(C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$  qui est une matrice symétrique peut être réalisée à l'aide de la décomposition de Cholesky [25].

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

---

**Algorithm 2** Kalman Filter

---

```

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

```

---



---

**Algorithm 3** Extended Kalman Filter

---

```

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

```

---

L'algorithme Extended Kalman Filter(EKF) correspond à la version non linéaire de l'algorithme du filtre de Kalman. Cette variante a été développée 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 dans l'algorithme EKF. Ces fonctions sont des fonctions linéaires ou non linéaires. Elles correspondent au modèle de mouvement et au modèle d'observation du robot. Les matrices  $G$  et  $H$  correspondent à la décomposition de Taylor du premier ordre des fonctions  $g$  et  $h$ . Cette décomposition de Taylor permet de rendre linéaires les fonctions  $g$  et  $h$ . EKF applique ces fonctions linéaires sur le calcul de la matrice de variance. Il est donc maintenant possible de passer en paramètres un modèle de mouvement non linéaire et un modèle d'observation non linéaire en gardant une représentation gaussienne de la pose du robot. Contrairement au filtre de Kalman classique où les modèles sont obligatoirement linéaires pour préserver des fonctions de répartition gaussienne.

Cette décomposition de Taylor n'ajoute pas de complexité à l'algorithme. En effet, elle est faite analytiquement au préalable. Cependant, la linéarisation implique que l'algorithme EKF retourne une approximation de la fonction de densité de probabilité associée au modèle de mouvement et d'observation non linéaires. La figure II.5 représente la linéarisation de la fonction  $g$ . On voit clairement que si la fonction  $g$  est appliquée sans linéarisation sur l'ensemble de la croyance, la croyance n'est plus une gaussienne. On peut également voir que la gaussienne construite à l'aide de la linéarisation correspond à une approximation de la vraie gaussienne. Dans le cas d'un modèle de mouvement linéaire et un modèle d'observation linéaire, l'algorithme EKF fournit les mêmes résultats que l'algorithme KF.

L'algorithme 4 décrit l'algorithme EKF appliqué à la localisation de notre robot. Le robot utilise une carte composée de repères (comme défini dans la section I.1) pour se localiser. Cette carte est passée en paramètre en plus des paramètres classiques de l'algorithme EKF. Cette version de l'EKF localisation permet d'intégrer plusieurs observations au temps  $t$  grâce à la boucle qui itère sur chaque observation. Le modèle de mouvement et le modèle d'observation correspondent à ceux définis dans la section II.1.2. La fonction  $g$ , qui correspond au modèle de mouvement, est définie comme suit :

$$g = \begin{pmatrix} x_t \\ y_t \\ \theta_t \end{pmatrix} = \mu_{t-1} + \begin{pmatrix} d_t \cos(\theta_{t-1} + \gamma_t) \\ d_t \sin(\theta_{t-1} + \gamma_t) \\ \gamma_t \end{pmatrix}$$

Le développement de Taylor du premier ordre de la fonction  $g$  en  $\mu_{t-1}$  est donné par la matrice  $G$  suivante :

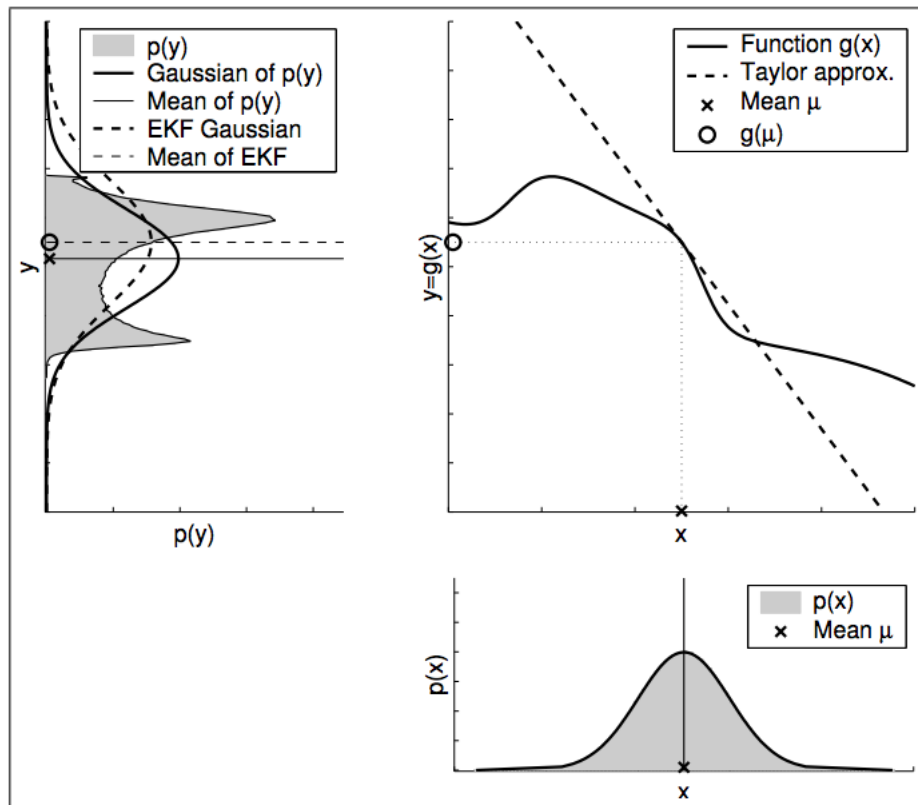


FIGURE II.5 – EKF linéarisation  
**Source:** Probabilistic Robotics [22]

$$G = \begin{pmatrix} \frac{\partial x_t}{\partial \mu_{t-1,x}} & \frac{\partial x_t}{\partial \mu_{t-1,y}} & \frac{\partial x_t}{\partial \mu_{t-1,\theta}} \\ \frac{\partial y_t}{\partial \mu_{t-1,x}} & \frac{\partial y_t}{\partial \mu_{t-1,y}} & \frac{\partial y_t}{\partial \mu_{t-1,\theta}} \\ \frac{\partial \theta_t}{\partial \mu_{t-1,x}} & \frac{\partial \theta_t}{\partial \mu_{t-1,y}} & \frac{\partial \theta_t}{\partial \mu_{t-1,\theta}} \end{pmatrix} = \begin{pmatrix} 1 & 0 & -d_t \sin(\theta_{t-1} + \gamma_t) \\ 0 & 1 & d_t \cos(\theta_{t-1} + \gamma_t) \\ 0 & 0 & 1 \end{pmatrix}$$

La fonction  $h$  qui correspond au modèle d'observation est définie comme suit :

$$h = \begin{pmatrix} \sqrt{(m_{j,x} - \bar{\mu}_{t,x})^2 + (m_{j,y} - \bar{\mu}_{t,y})^2} \\ \text{atan2}(m_{j,y} - \bar{\mu}_{t,y}, m_{j,x} - \bar{\mu}_{t,x}) - \bar{\mu}_{t,\theta} \end{pmatrix}$$

où  $m_{j,x}, m_{j,y}$  correspondent aux coordonnées du repère enregistrées dans la carte. Le développement de Taylor du premier ordre de la fonction  $h$  en  $\bar{\mu}_t$  est donné par la matrice  $H$  qui est définie comme suit :

$$H = \begin{pmatrix} \frac{\partial d_t^i}{\partial \bar{\mu}_{t,x}} & \frac{\partial d_t^i}{\partial \bar{\mu}_{t,y}} & \frac{\partial d_t^i}{\partial \bar{\mu}_{t,\theta}} \\ \frac{\partial \rho_t^i}{\partial \bar{\mu}_{t,x}} & \frac{\partial \rho_t^i}{\partial \bar{\mu}_{t,y}} & \frac{\partial \rho_t^i}{\partial \bar{\mu}_{t,\theta}} \end{pmatrix} = \begin{pmatrix} -\frac{m_{j,x} - \bar{\mu}_{t,x}}{\sqrt{q}} & -\frac{m_{j,y} - \bar{\mu}_{t,y}}{\sqrt{q}} & 0 \\ \frac{m_{j,y} - \bar{\mu}_{t,y}}{q} & -\frac{m_{j,x} - \bar{\mu}_{t,x}}{q} & -1 \end{pmatrix}$$

Pour pouvoir intégrer l'erreur de mouvement dans l'espace de pose du robot, il est nécessaire de l'exprimer dans les mêmes variables que la pose du robot. En effet, le vecteur de pose du robot est exprimé en  $(x, y, \theta)$  or l'erreur de contrôle est exprimée sur les variables  $(d, \gamma)$ . La matrice  $V_t$  correspond au développement de Taylor du premier ordre en  $\mu_{t-1}$ . Elle est définie comme suit :

$$V_t = \begin{pmatrix} \frac{\partial x_t}{\partial d_t} & \frac{\partial x_t}{\partial \gamma_t} \\ \frac{\partial y_t}{\partial d_t} & \frac{\partial y_t}{\partial \gamma_t} \\ \frac{\partial \theta_t}{\partial d_t} & \frac{\partial \theta_t}{\partial \gamma_t} \end{pmatrix} = \begin{pmatrix} \cos(\theta_{t-1} + \gamma_t) & -d \sin(\theta_{t-1} + \gamma_t) \\ \sin(\theta_{t-1} + \gamma_t) & d \cos(\theta_{t-1} + \gamma_t) \\ 0 & 1 \end{pmatrix}$$

La formule suivante permet d'exprimer l'approximation de l'erreur des commandes dans les mêmes variables que le vecteur de pose du robot.

$$R_t = V_t M_t V_t^T$$

## 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 pose. Dans les algorithmes non paramétriques, la croyance est représentée par un nombre déterminé de poses supposées. Une probabilité est associée à ces poses supposées. Plusieurs techniques existent pour les représenter.



**Algorithm 4** EKF localisation

---

```

1: procedure EKF LOCALISATION ( $\mu_{t-1}, \Sigma_{t-1}, u_t, z_t, m$ )
2:    $\theta_{t-1} \leftarrow \mu_{t-1, \theta}$ 
3:    $G_t \leftarrow \begin{pmatrix} 1 & 0 & -d_t \sin(\theta_{t-1} + \gamma_t) \\ 0 & 1 & d_t \cos(\theta_{t-1} + \gamma_t) \\ 0 & 0 & 1 \end{pmatrix}$ 
4:    $V_t \leftarrow \begin{pmatrix} \cos(\theta_{t-1} + \gamma_t) & -d_t \sin(\theta_{t-1} + \gamma_t) \\ \sin(\theta_{t-1} + \gamma_t) & d_t \cos(\theta_{t-1} + \gamma_t) \\ 0 & 1 \end{pmatrix}$ 
5:    $M_t \leftarrow \begin{pmatrix} \sigma_{d_t}^2 & 0 \\ 0 & \sigma_{\gamma_t}^2 \end{pmatrix}$ 
6:    $\bar{\mu}_t \leftarrow \mu_{t-1} + \begin{pmatrix} d_t \cos(\theta_{t-1} + \gamma_t) \\ d_t \sin(\theta_{t-1} + \gamma_t) \\ \gamma_t \end{pmatrix}$  ▷ prédiction
7:    $\bar{\Sigma}_t \leftarrow G_t \Sigma_{t-1} G_t^T + V_t M_t V_t^T$  ▷ prédiction
8:    $Q \leftarrow \begin{pmatrix} \sigma_{d^z}^2 & 0 \\ 0 & \sigma_{\rho^z}^2 \end{pmatrix}$ 
9:   for all observed features  $z_t^i \leftarrow (d_t^i, \rho_t^i)^T$  do
10:      $j \leftarrow$  landmark observed
11:      $q \leftarrow (m_{j,x} - \bar{\mu}_{t,x})^2 + (m_{j,y} - \bar{\mu}_{t,y})^2$  ▷  $m_{j,x} m_{j,y}$  coordonnées du
repère
12:      $\hat{z}_t^i \leftarrow \begin{pmatrix} \sqrt{q} \\ \text{atan2}(m_{j,y} - \bar{\mu}_{t,y}, m_{j,x} - \bar{\mu}_{t,x}) - \bar{\mu}_{t,\theta} \end{pmatrix}$ 
13:      $H_t^i \leftarrow \begin{pmatrix} -\frac{m_{j,x} - \bar{\mu}_{t,x}}{\sqrt{q}} & -\frac{m_{j,y} - \bar{\mu}_{t,y}}{\sqrt{q}} & 0 \\ \frac{m_{j,y} - \bar{\mu}_{t,y}}{q} & -\frac{m_{j,x} - \bar{\mu}_{t,x}}{q} & -1 \end{pmatrix}$ 
14:      $S_t^i \leftarrow H_t^i \bar{\Sigma}_t [H_t^i]^T + Q_t$ 
15:      $K_t^i \leftarrow \bar{\Sigma}_t [H_t^i]^T [S_t^i]^{-1}$  ▷ Kalman Gain
16:      $\bar{\mu}_t \leftarrow \bar{\mu}_t + K_t^i (z_t^i - \hat{z}_t^i)$  ▷ mise à jour
17:      $\bar{\Sigma}_t \leftarrow (I - K_t^i H_t^i) \bar{\Sigma}_t$  ▷ mise à jour
18:   end for
19:    $\mu_t \leftarrow \bar{\mu}_t$ 
20:    $\Sigma_t \leftarrow \bar{\Sigma}_t$ 
21:   return  $\mu_t, \Sigma_t$ 
22: end procedure

```

---

La première technique consiste à découper la carte de l'environnement en une grille où chaque élément de la grille correspond aux coordonnées supposées du robot [23]. Pour ajouter l'orientation du robot dans cette représentation sous forme de grille, il faut multiplier le nombre de cases par le nombre d'angles d'orientation que peut prendre le robot. Dans la représentation II.7, 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 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 particules, une approche différente a été choisie. Dans cet algorithme (voir l'algorithme 5) la croyance de la pose est représentée par  $M$  particules (voir la figure II.6). Chaque particule est considérée comme une hypothèse sur la pose 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 et plus la précision de cette zone de la croyance est importante. Plus le poids associé à la particule est important et plus la probabilité que le robot s'y trouve est grande. L'algorithme est composé de l'étape de prédiction et de l'étape de mise à jour comme l'algorithme de Markov. Pour chaque particule, ces deux étapes sont appliquées. L'étape de prédiction applique le mouvement à chaque particule. L'étape de mise à jour détermine une valeur en fonction de la probabilité d'avoir une telle observation en étant à la pose de la particule. Une fois ces deux étapes appliquées à chaque particule, la partie finale consiste à redéfinir l'ensemble des particules. Le nouvel ensemble est construit avec les particules qui ont la plus grande probabilité associée à l'étape précédente. 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 pose 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.



FIGURE II.6 – Illustration de MCL  
**Source:** Wikipedia, Auteur : Daniel Lu

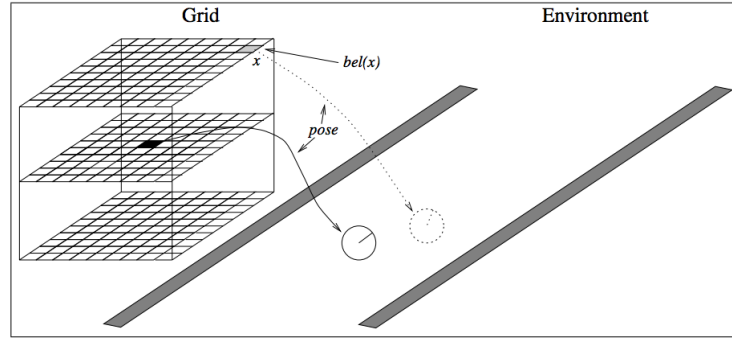


FIGURE II.7 – Illustration de la carte en grille  
**Source:** Probabilistic Robotics [22]

---

**Algorithm 5** MCL

---

```

1: procedure MCL ( $\mathcal{X}_{t-1}, u_t, z_t, m$ )
2:    $\overline{\mathcal{X}}_t \leftarrow \emptyset$ 
3:    $\mathcal{X}_t \leftarrow \emptyset$ 
4:   for  $m = 1$  to  $M$  do
5:      $x_t^{[m]} \leftarrow \text{sample\_motion\_model}(u_t, x_{t-1}^{[m]})$  ▷ prédiction
6:      $w_t^{[m]} \leftarrow \text{measurement\_model}(z_t, x_t^{[m]}, m)$  ▷ mise à jour
7:      $\overline{\mathcal{X}}_t \leftarrow \overline{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$ 
8:   end for
9:   for  $m = 1$  to  $M$  do
10:    draw  $i$  with probability proportional to  $w_t^{[i]}$ 
11:    add  $x_t^{[i]}$  to  $\mathcal{X}_t$ 
12:   end for
13:   return  $\mathcal{X}_t$ 
14: end procedure

```

---

### 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 être 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 qu'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. 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 pose est proche d'une distribution gaussienne et a donc de mauvaises performances lorsque la croyance correspond plutôt à une distribution multimodale. Pour pallier à ce problème, l'algorithme classique EKF a été amélioré. Le filtre Multi-hypothesis Tracking (MHT) [4] permet de représenter la croyance à l'aide d'un mixte de plusieurs gaussiennes 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 pose initiale n'est pas connue. L'incertitude est donc grande à cet instant. Il s'avère qu'EKF est plus approprié pour suivre la pose d'un robot dont on connaît déjà la pose initiale. En effet, une représentation unimodale est généralement une bonne représentation dans un problème où il s'agit de suivre une pose, mais pas dans un problème de localisation globale. La linéarisation dans EKF ne fait qu'accroître ce problème en risquant de converger vers une mauvaise pose.

De plus, MCL permet de traiter directement dans l'algorithme des mesures brutes or EKF nécessite des repères. Déterminer un repère nécessite un prétraitement des données brutes. 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. Ceci n'est pas possible à l'aide d'EKF qui nécessite une carte composée d'un nombre limité de repères qui permet de localiser le robot à l'aide des repères mesurés

	EKF	MCL
Mesures	Repères	Brutes
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 d'EKF et de MCL

dans son environnement. Finalement, en pratique il s'avère que MCL est plus simple à implémenter qu'EKF.

## Chapitre III

# Simultaneous Localization And Mapping

Les algorithmes de type SLAM (Simultaneous Localization And Mapping) sont l'étape suivante vers 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 particules dans le FastSLAM sont composées d'une estimation du chemin du robot et de l'ensemble des estimations des points de repère de la carte avec leur covariance associée. La description détaillée de cet algorithme est disponible dans le livre « Probabilistic robotics » [\[22\]](#).

### III.1 EKF SLAM

EKF SLAM est un algorithme SLAM (voir algorithme 6) et comme son nom le laisse penser il 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 représentée comme l'ensemble des poses des repères découverts à l'instant  $t$ . Cet algorithme est aussi découpé en deux étapes importantes, 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. Ceci est normal, car les repères ne bougent pas même si le robot change de pose. Dans l'étape de mise à jour, lorsqu'un repère est découvert pour la première

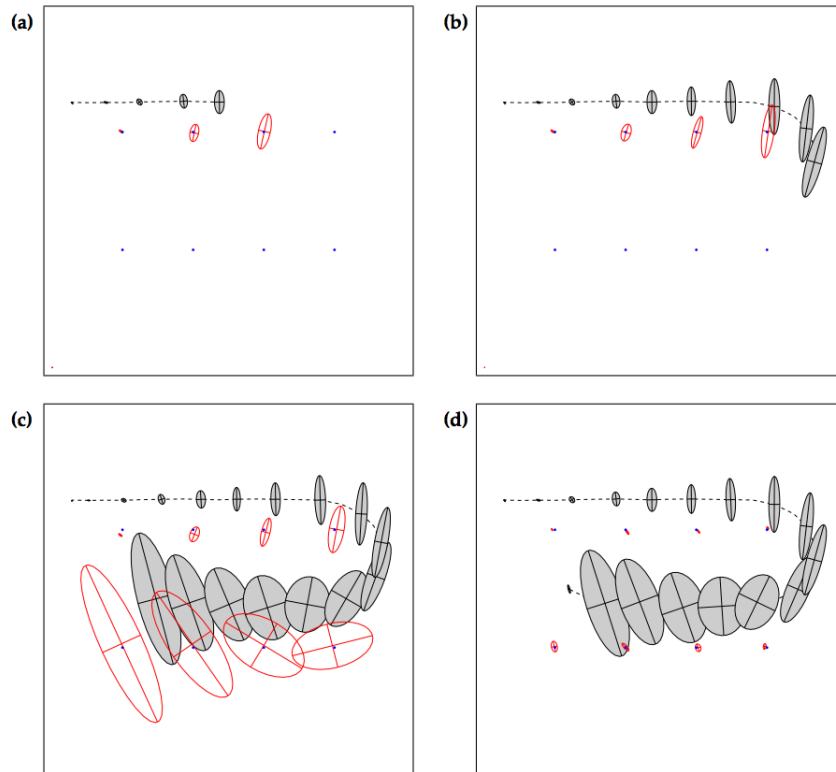


FIGURE III.1 – EKF SLAM

**Source:** Probabilistic Robotics [22]

fois il est initialisé à l'aide de la pose du robot, de la distance et de l'angle auxquels il a été observé. La figure III.1 présente le parcours d'un robot qui applique l'EKF SLAM. Les petits points représentent les repères qui sont dans l'environnement du robot. Lorsque le robot observe un point de repère, l'ellipse représentant la covariance associée à ce repère est représentée sur le dessin. Plus la pose du robot est incertaine et plus la position des repères qu'il va découvrir est également incertaine. Finalement, lorsque le robot scanne un repère qu'il a déjà analysé, l'ensemble des covariances des repères déjà observés sont mises à jour.

**Algorithm 6** EKF SLAM

---

```

1: procedure EKF SLAM ( $\mu_{t-1}, \Sigma_{t-1}, u_t, z_t, m$ )
2:    $\theta_{t-1} \leftarrow \mu_{t-1, \theta}$ 
3:    $F_x \leftarrow \begin{pmatrix} 1 & 0 & 0 & 0 \dots 0 \\ 0 & 1 & 0 & 0 \dots 0 \\ 0 & 0 & 1 & \underbrace{0 \dots 0}_{2N} \end{pmatrix}$  ▷ N = nombre de repères
4:    $\bar{\mu}_t \leftarrow \mu_{t-1} + F_x^T \begin{pmatrix} d_t \cos(\theta_{t-1} + \gamma_t) \\ d_t \sin(\theta_{t-1} + \gamma_t) \\ \gamma_t \end{pmatrix}$  ▷ prédiction
5:    $G_t \leftarrow I + F_x^T \begin{pmatrix} 0 & 0 & -d_t \sin(\theta_{t-1} + \gamma_t) \\ 0 & 0 & d_t \cos(\theta_{t-1} + \gamma_t) \\ 0 & 0 & 0 \end{pmatrix} F_x$ 
6:    $\bar{\Sigma}_t \leftarrow G_t \Sigma_{t-1} G_t^T + F_x^T R_t F_x$  ▷ prédiction
7:    $Q_t \leftarrow \begin{pmatrix} \sigma_{d^z}^2 & 0 \\ 0 & \sigma_{\rho^z}^2 \end{pmatrix}$ 
8:   for all observed features  $z_t^i \leftarrow (d_t^i, \rho_t^i)^T$  do
9:      $j \leftarrow$  landmark observed
10:    if landmark j never seen before then
11:       $\begin{pmatrix} \bar{\mu}_{j,x} \\ \bar{\mu}_{j,y} \end{pmatrix} \leftarrow \begin{pmatrix} \bar{\mu}_{t,x} \\ \bar{\mu}_{t,y} \end{pmatrix} + \begin{pmatrix} d_t^i \cos(\rho_t^i + \bar{\mu}_{t,\theta}) \\ d_t^i \sin(\rho_t^i + \bar{\mu}_{t,\theta}) \end{pmatrix}$ 
12:    end if
13:     $F_{x,j} \leftarrow \begin{pmatrix} 1 & 0 & 0 & 0 \dots 0 & 0 & 0 & 0 \dots 0 \\ 0 & 1 & 0 & 0 \dots 0 & 0 & 0 & 0 \dots 0 \\ 0 & 0 & 1 & 0 \dots 0 & 0 & 0 & 0 \dots 0 \\ 0 & 0 & 0 & 0 \dots 0 & 1 & 0 & 0 \dots 0 \\ 0 & 0 & 0 & \underbrace{0 \dots 0}_{2j-2} & 0 & 1 & \underbrace{0 \dots 0}_{2N-2j} \end{pmatrix}$ 
14:     $q \leftarrow (m_{j,x} - \bar{\mu}_{t,x})^2 + (m_{j,y} - \bar{\mu}_{t,y})^2$ 
15:     $\hat{z}_t^i \leftarrow \begin{pmatrix} \sqrt{q} \\ \text{atan2}(m_{j,y} - \bar{\mu}_{t,y}, m_{j,x} - \bar{\mu}_{t,x}) - \bar{\mu}_{t,\theta} \end{pmatrix}$ 
16:     $H_t^i \leftarrow \begin{pmatrix} -\frac{m_{j,x} - \bar{\mu}_{t,x}}{\sqrt{q}} & -\frac{m_{j,y} - \bar{\mu}_{t,y}}{\sqrt{q}} & 0 & \frac{m_{j,x} - \bar{\mu}_{t,x}}{\sqrt{q}} & \frac{m_{j,y} - \bar{\mu}_{t,y}}{\sqrt{q}} \\ \frac{m_{j,y} - \bar{\mu}_{t,y}}{q} & -\frac{m_{j,x} - \bar{\mu}_{t,x}}{q} & -1 & -\frac{m_{j,y} - \bar{\mu}_{t,y}}{q} & \frac{m_{j,x} - \bar{\mu}_{t,x}}{q} \end{pmatrix} F_{x,j}$ 
17:     $S_t^i \leftarrow H_t^i \bar{\Sigma}_t [H_t^i]^T + Q_t$ 
18:     $K_t^i \leftarrow \bar{\Sigma}_t [H_t^i]^T [S_t^i]^{-1}$  ▷ Kalman Gain
19:     $\bar{\mu}_t \leftarrow \bar{\mu}_t + K_t^i (z_t^i - \hat{z}_t^i)$  ▷ mise à jour
20:     $\bar{\Sigma}_t \leftarrow (I - K_t^i H_t^i) \bar{\Sigma}_t$  ▷ mise à jour
21:  end for
22:   $\mu_t \leftarrow \bar{\mu}_t$ 
23:   $\Sigma_t \leftarrow \bar{\Sigma}_t$ 
24:  return  $\mu_t, \Sigma_t$ 
25: end procedure

```

---





# Chapitre IV

## Algorithmes de recherche du meilleur chemin

Les algorithmes de recherche du 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  $A^*$  et de 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

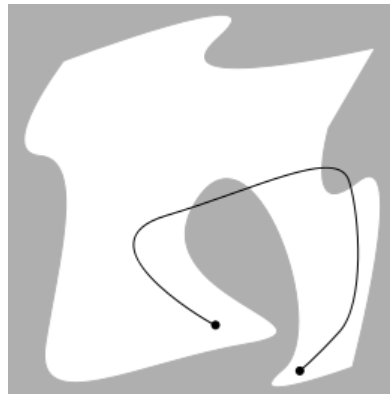


FIGURE IV.1 – Chemin non valide

**Source:** Wikipedia, Auteur : Simeon87

robot et les arêtes représentent 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'intérieur 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 relie. Ceci signifie 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. Ceci se produit lorsqu'il est impossible d'y accéder sans percuter des objets de l'environnement.

Plus la grille a un découpage important, et plus le chemin qui est 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 rotation. En effet, il est commun que la position du robot définisse 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

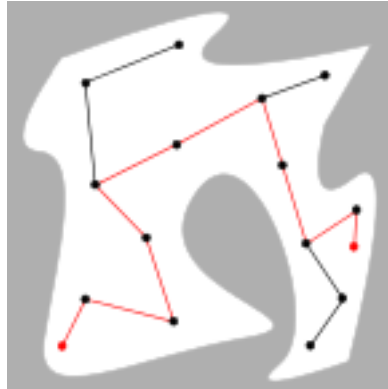


FIGURE IV.2 – Chemin valide construit à l’aide d’un graphe

**Source:** Wikipedia, Auteur : Simeon87

robot. Ce paramètre permet donc de construire les chemins avec des positions 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 Dijkstra 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  $A^*$  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  $A^*$ .

---

**Algorithm 7** Dijkstra
 

---

```

1: procedure DIJKSTRA ( $G = (S, A), S_{deb}, S_{fin}$ )
2:   Initialiser tous les sommets comme non marqués et donner un valeur
    $+\infty$  à tous les labels L
3:    $L(S_{deb}) \leftarrow 0$ 
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:          $L(b) \leftarrow L(a) + v(a, b)$ 
9:          $b.prédédent \leftarrow a$ 
10:      end if
11:    end for
12:  end while
13:   $S_n \leftarrow S_{fin}$ 
14:  while  $S_n \neq S_{deb}$  do
15:     $chemin \leftarrow chemin + S_n$ 
16:     $S_n \leftarrow S_n.précédent$ 
17:  end while
18:  return  $chemin$ 
19: end procedure

```

---

### IV.3 $A^*$

L'algorithme  $A^*$  a été publié en 1968 [9] et fonctionne de façon relativement semblable à l'algorithme de Dijkstra, mais celui-ci permet d'obtenir de meilleures performances que Dijkstra grâce à l'utilisation d'une heuristique. Cette heuristique donne une estimation du cout jusqu'aux destinations recherchées. Pour l'appliquer 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 vol d'oiseau qui est toujours plus courte ou égale à n'importe quel chemin. L'algorithme 8 correspond au pseudocode de l'algorithme  $A^*$ . L'algorithme  $A^*$ , comme celui de Dijkstra, est découpé en deux parties principales. La première partie consiste à déterminer les plus courts

FIGURE IV.3 —  $A^*$  dans un labyrinthe**Source:** Wikipedia, Auteur : Bayo

chemins entre un sommet initial et les autres sommets. Elle est suivie de la reconstruction du chemin entre le sommet d'arrivée et le sommet de départ.

Intuitivement, l'algorithme  $A^*$ , composé de l'heuristique estimant la distance à vol d'oiseau, explore itérativement le noeud joignable et étant estimé par l'heuristique comme le plus proche du noeud cible. Lorsque ce noeud est choisi, ses voisins deviennent joignables. Ensuite, 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  $A^*$  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 finit toujours par retourner le chemin le plus court.

**Algorithm 8**  $A^*$ 


---

```

1: procedure  $A^*$  ( $G = (S, A), S_{start}, S_{goal}$ )
2:    $closedSet \leftarrow \emptyset$  ▷ les noeuds déjà évalués
3:    $openSet \leftarrow \{S_{start}\}$ 
4:    $came\_from \leftarrow \emptyset$ 
5:    $g\_score \leftarrow +\infty$  ▷  $+\infty$  pour tous les éléments
6:    $g\_score[S_{start}] \leftarrow 0$ 
7:    $f\_score \leftarrow +\infty$ 
8:    $f\_score[S_{start}] \leftarrow g\_score[S_{start}] + h(S_{start}, S_{goal})$ 
9:   while  $openSet \neq \emptyset$  do
10:     $S_{current} \leftarrow \text{node with lowest } f\_score[]$ 
11:    if  $S_{current} == S_{goal}$  then
12:      return  $reconstruct\_path(came\_from, S_{goal})$ 
13:    end if
14:     $openSet \leftarrow openSet - S_{current}$ 
15:     $closedSet \leftarrow closedSet + S_{current}$ 
16:    for all  $S_{neighbor} \in \text{neighbor\_nodes}(S_{current}) \ \& \notin closedSet$  do
17:       $t\_g\_score \leftarrow g[S_{current}] + dist(S_{current}, S_{neighbor})$ 
18:      if  $S_{neighbor} \notin openSet \ || \ t\_g\_score < g\_score[S_{neighbor}]$  then
19:         $came\_from[S_{neighbor}] \leftarrow S_{current}$ 
20:         $g\_score[S_{neighbor}] \leftarrow t\_g\_score$ 
21:         $f\_score[S_{neighbor}] \leftarrow g\_score[S_{neighbor}] + h(S_{neighbor}, S_{goal})$ 
22:        if  $S_{neighbor} \notin openSet$  then
23:           $openSet \leftarrow openSet + S_{neighbor}$ 
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:   $path \leftarrow \{S_{current}\}$ 
32:  while  $S_{current} \in came\_from$  do
33:     $S_{current} \leftarrow came\_from[S_{current}]$ 
34:     $path \leftarrow path + S_{current}$ 
35:  end while
36:  return  $path$ 
37: end procedure

```

---

# Deuxième partie

## Implémentations





Cette partie décrit l'implémentation de l'algorithme EKF sur la brique EV3. Il y est également présenté un algorithme qui utilise la position du robot pour construire une grille d'occupation. Cette grille d'occupation sert à construire un graphe qui permet de trouver un chemin entre un point de départ et un point d'arrivée de la carte. En plus de ces descriptions d'algorithmes, 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 coûteuse 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 construction 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<sup>1</sup> ou Raspberry Pi<sup>2</sup> et d'une série de capteurs et de moteurs. Les microcontrôleurs Arduino et Raspberry Pi possèdent 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](http://www.arduino.cc)

2. Raspberry Pi : [www.raspberrypi.org](http://www.raspberrypi.org)

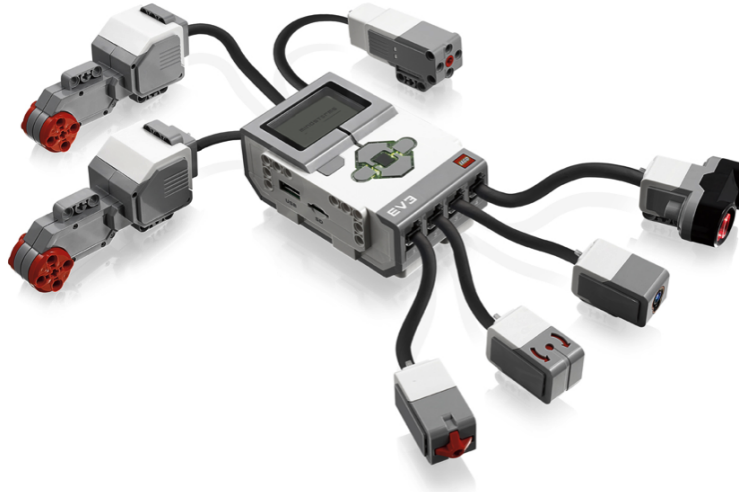


FIGURE V.1 – Brique EV3  
Source: [www.4erevolution.com](http://www.4erevolution.com)

### V.1.1 Brique intelligente

La brique intelligente programmable (voir Figure V.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 l'intermédiaire des ports d'entrée et de sortie. Ceci 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 succédés : 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 depuis la carte SD. Ceci augmente ainsi le potentiel de la brique (comme décrit dans la section V.2.1).

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 brique 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'autres types de capteurs développés par des sociétés indépendantes de Lego. C'est possible grâce au support de protocoles standards 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 ou encore des capteurs d'humidité qui sont compatibles 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. Elles sont dirigées vers le type de données reçues ainsi que la précision des capteurs et des moteurs. Elles sont importantes dans ce mémoire, car elles permettent de mieux connaître les capteurs et les moteurs à notre disposition. Il est ainsi plus facile de 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. Cette technologie est présente dans les télécommandes des TV et 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 toutes les pressions ou tous les relâchements 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 boîtier 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 il possède un capteur de rotation intégré.

### V.1.3 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<sup>3</sup> permet d'utiliser ou de construire des modèles de robots qui évoluent dans un environnement prédéfini. Ce projet open source a reçu des financements de la DRC<sup>4</sup> pour l'adaptation du simulateur à son robot ATLAS. Cependant, ce type de simulateur nécessite un temps d'apprentissage certain pour pouvoir commencer à être utilisé. C'est pourquoi d'autres simulateurs plus simples de la brique de Lego existent. L'université de Berne a développé EV3JLIB<sup>5</sup> qui permet de lancer gratuitement un simulateur en ligne. Le comportement du robot est codé en Java. Ce code a 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 »<sup>6</sup> et « Robot virtual worlds »<sup>7</sup> mais ils sont payants.

---

3. Gazebo : [gazebo.sim.org](http://gazebo.sim.org)

4. DRC : [www.theroboticschallenge.org](http://www.theroboticschallenge.org)

5. EV3JLIB : [www.aplu.ch](http://www.aplu.ch)

6. Virtual robotics toolkit : [cogmation.com](http://cogmation.com)

7. Robot virtual worlds : [www.robotvirtualworlds.com](http://www.robotvirtualworlds.com)

## V.2 Description du software

### V.2.1 RoboLab

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](#)). Ce langage de programmation est constitué d'un ensemble de briques. Il est possible de les assembler pour définir le mini programme qui définit le comportement du robot. Les briques permettent de définir des actions sur les moteurs, afficher des messages sur l'écran de la brique EV3, jouer des sons. Il est également possible de définir des conditions à l'aide des valeurs obtenues par les capteurs et même de lancer des boucles. 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'algorithmes complexes de localisation comme EKF ou bien MCL. Les briques Mindstorm ont vite été hackées pour permettre de développer des programmes à l'aide d'autres langages que RoboLab. Dans la version EV3, il est possible de booter sur une carte SD. Ceci a permis que des OS tels que Linux soient adaptés pour fonctionner sur la brique. Le projet ev3Dev est un exemple d'adaptation de Debian GNU/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 en est donc donnée dans une section dédiée.

### V.2.2 Lejos

Lejos est un microprogramme libre destiné à remplacer le microprogramme originellement 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. La machine virtuelle Java 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 mise à disposition par Oracle. Lejos permet un accès facile aux moteurs et aux capteurs. Ceci est possible grâce aux routines écrites en C et mises dans le domaine public par le groupe Lego. En effet, Lego a décidé 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.



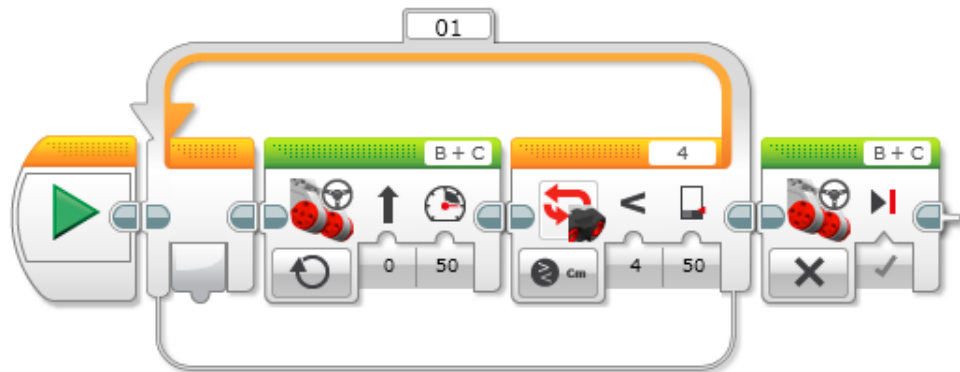


FIGURE V.2 – Robolab  
Source: [www.lego.com](http://www.lego.com)

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](http://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<sup>8</sup> par les professeurs de l'université de Porto. Un étudiant suédois a développé un SLAM<sup>9</sup> 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<sup>10</sup>. L'ensemble de ces projets prouve que Lejos possède de bonnes bases pour développer des algorithmes complexes.

8. EKF : [paginas.fe.up.pt/~robosoc](http://paginas.fe.up.pt/~robosoc)

9. SLAM : [penemunxt.blogspot.be](http://penemunxt.blogspot.be)

10. Diagramme d'états : [github.com/jabrena/liverobots](https://github.com/jabrena/liverobots)

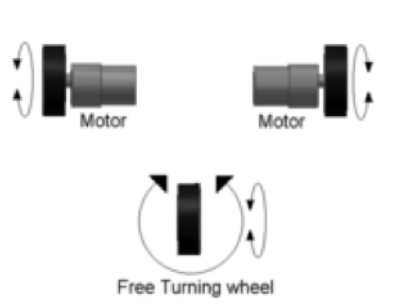


FIGURE V.3 – Differential wheeled robot

Source: Wikipedia, Auteur : Patrik

## V.3 Description du robot construit

La plateforme du robot construit est de type « differential wheeled robot » (voir V.3). Ceci consiste en deux moteurs indépendants positionnés de façon opposée sur le robot. En plus de ses deux roues, une roue libre est placée à l'arrière. Il est conseillé de ne pas placer cette roue libre au centre de l'axe de rotation vertical de celle-ci pour créer un levier qui garantit une meilleure rotation de cette roue libre. Ce choix de plateforme est commun en robotique. Cette plateforme est simple à mettre en oeuvre et fournit une amplitude de mouvement importante. A l'inverse, les plateformes « steering » sont semblables aux voitures classiques. Elles demandent un mécanisme plus complexe et leur amplitude de mouvement est moindre et donc moins adaptée à la robotique.

La figure V.4 présente le robot qui a été construit pour réaliser l'implémentation et tester les algorithmes présentés dans ce mémoire. Le robot est composé d'un capteur de pression. Il n'est pas possible de déterminer la localisation d'une pression sur le pare-chocs avant. La pression est identifiée indifféremment que ce soit le côté gauche ou le côté droit du pare-chocs qui a été touché. Il est également équipé d'un capteur de distance infrarouge sur sa face avant. Un support est placé sur le haut du robot pour accueillir le smartphone qui s'occupe de scanner les codes QR qui sont les points de repère du robot dans l'algorithme EKF. Pour permettre à la brique EV3 de communiquer de façon efficace avec le smartphone et l'ordinateur, un stick USB wifi a été ajouté à la brique EV3. La communication avec l'ordinateur doit être efficace pour permettre que les données captées par les capteurs de la brique EV3 soient rapidement envoyées à l'ordinateur et inversement pour que les commandes envoyées par l'ordinateur arrivent rapidement sur la brique EV3.

La construction de la plateforme de déplacement et l'ajout des capteurs sur le robot sont des étapes à ne pas négliger. Elle est relativement longue pour avoir un robot utilisable. En effet, il est important de positionner les capteurs et les moteurs à des endroits réfléchis. Il est par exemple primordial de ne pas placer des éléments de la plateforme devant le capteur de distance. Le pare-chocs muni du capteur de pression doit être construit de façon à bien réagir à tous les types de chocs. L'ensemble des câbles doit être introduit dans les ports appropriés de la brique EV3 sans entraver le fonctionnement des moteurs ou des capteurs. Finalement, certaines propriétés du robot permettent de simplifier leur manipulation et les algorithmes qui sont implémentés pour les manipuler. Il est par exemple plus simple de manipuler un robot avec un châssis arrondi et capable de faire un demi-tour sur lui-même. Il est possible à l'aide d'un châssis arrondi de faire facilement un demi-tour sur soi-même dans un angle d'un mur sans le percuter. Alors qu'un châssis de type rectangulaire a tendance à se bloquer lors de la rotation pour faire le demi-tour. En effet, les coins du robot ont tendance à percuter le mur. Dans le cas d'un robot rectangulaire, les algorithmes de mouvement doivent prendre en compte les particularités de la forme du robot pour éviter de percuter des objets. L'ensemble des contraintes de construction des robots rend compliquée la construction d'un robot. Il est donc possible de trouver sur le web des exemples de plateformes qui ont fait leurs preuves. Le site [nxtprograms](http://nxtprograms.com)<sup>11</sup> propose une série d'exemples de plateformes évolutives.

---

11. [nxtprograms](http://nxtprograms.com) : [www.nxtprograms.com](http://www.nxtprograms.com)

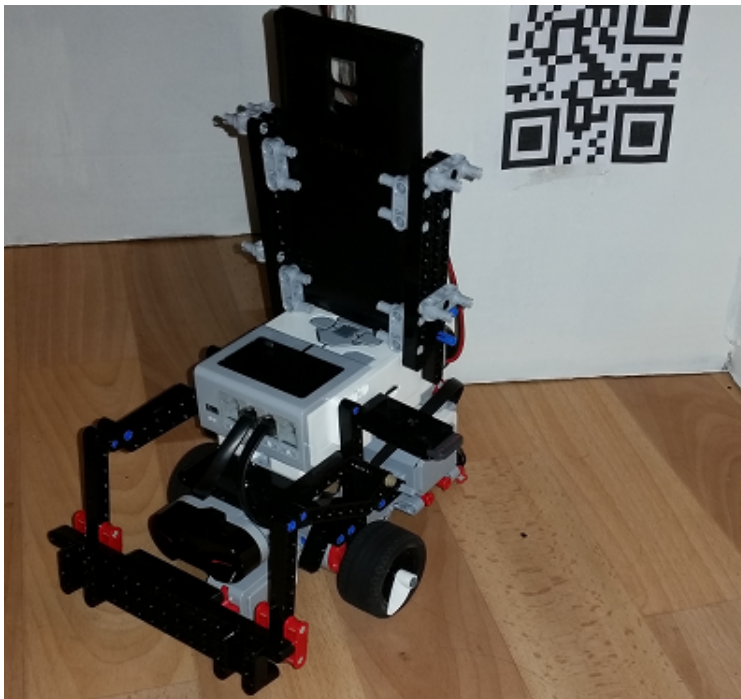


FIGURE V.4 – Robot



# Chapitre VI

## Algorithme de localisation

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

### VI.1 Algorithme EKF

#### VI.1.1 Odométrie

L'odométrie permet de déterminer les mouvements  $u_t$  nécessaires à l'étape de prédiction de chaque itération de l'algorithme EKF. Pour déterminer le mouvement  $u_t$ , l'odométrie consiste à compter le nombre de rotations des moteurs. Elle y associe le mouvement correspondant en fonction du châssis du robot. Il est nécessaire de connaître la taille des roues ainsi que leur écartement pour déterminer le mouvement en fonction du nombre de tours des moteurs. Dans le cas d'un robot « Differential wheeled » les formules sont les suivantes :

$$d = \frac{d_d + d_g}{2}$$

$$\gamma = \frac{d_d - d_g}{b}$$

où  $d$  représente la distance parcourue par le robot et  $\gamma$  l'angle de rotation.  $d_d$  et  $d_g$  correspondent respectivement à la distance parcourue par la roue droite et par la roue gauche. Finalement,  $b$  correspond à l'écart entre les deux roues. Les distances  $d_d$  et  $d_g$  sont calculées en multipliant le nombre de tours des moteurs par la circonférence de la roue. La formule pour calculer  $d_g$  est

la suivante et celle pour calculer  $d_d$  est similaire en adaptant les variables :

$$d_g = 2 * \pi * R_g * RotationMoteur_g$$

où  $R_g$  est le rayon de la roue gauche. Les mouvements  $u_t$  doivent être soit des lignes droites soit des rotations sur soi-même. Cela permet d'appliquer le modèle de mouvement. Pour rappel, le modèle de mouvement est construit avec une première rotation suivie d'un mouvement en ligne droite. Cette représentation n'est pas restrictive. En effet, il est possible de découper la trajectoire en un ensemble de sous mouvements qui sont des lignes droites ou des rotations.

Ces formules sont vraies pour les robots « Differential wheeled ». Si des robots ont un plus grand nombre de roues ou si elles ne sont pas placées de façon opposée, il est nécessaire d'adapter ces formules. Dans le cas d'une plateforme de type « steering », qui rappelons-le, est semblable aux voitures classiques, il y a un moteur unique. Il est donc nécessaire de connaître l'orientation des roues pour déterminer les rotations.

### VI.1.2 Détection de repères

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<sup>1</sup> 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 smartphone 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 10 cm de côté). 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 10 cm de côté (voir VI.1). La formule est la suivante :

$$Distance = \frac{\frac{ResolutionHorizontale}{TailleCodeQRPixelsHorizontale} * LargeurCodeQR}{2 * \tan(\alpha)}$$

Il est également possible de déterminer l'angle entre la direction du robot et le centre du code QR. Si le code QR se trouve à gauche du robot, la formule est la suivante et  $\rho$  est positif :

---

1. Zbar : [zbar.sourceforge.net](http://zbar.sourceforge.net)

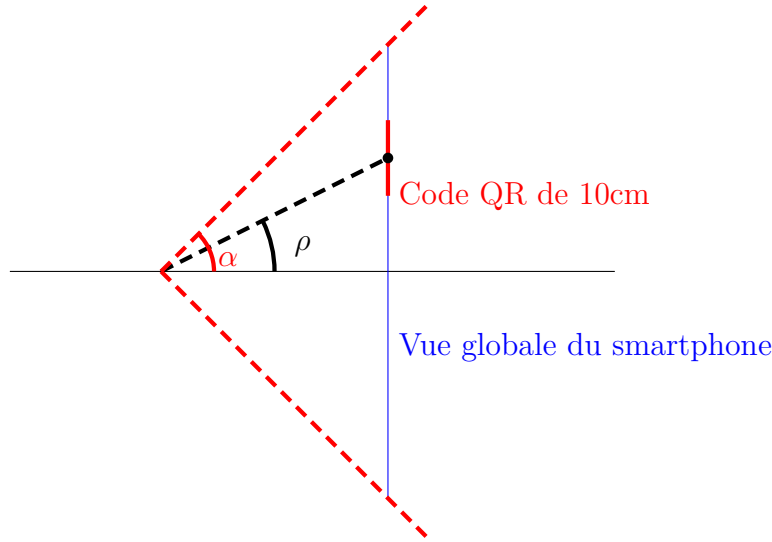


FIGURE VI.1 – Évaluation de la distance des codes QR

$$\rho = \alpha - \frac{\alpha * 2 * CentreCodeQRPixels}{ResolutionHorizontale}$$

si le code QR se trouve à droite du robot,  $\rho$  est négatif et la formule devient :

$$\rho = \frac{\alpha * 2 * CentreCodeQRPixels}{ResolutionHorizontale} - \alpha$$

L'étalonnage consiste à déterminer  $\alpha$  à l'aide de mesures faites à distance connue. Cet angle  $\alpha$  a de fortes chances d'être différent d'un capteur d'image à l'autre. Il est donc primordial de faire ces mesures d'étalonnage lors de la première utilisation du nouveau capteur d'images.

### VI.1.3 Les cartes

La carte qui stocke les positions des codes QR et qui est utilisée par l'algorithme EKF est enregistrée 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 10 cm de longueur. La figure VI.2 représente cette carte où les codes QR sont représentés en rouge et les murs sur lesquels les codes QR sont collés sont représentés en noir. Les murs sont 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



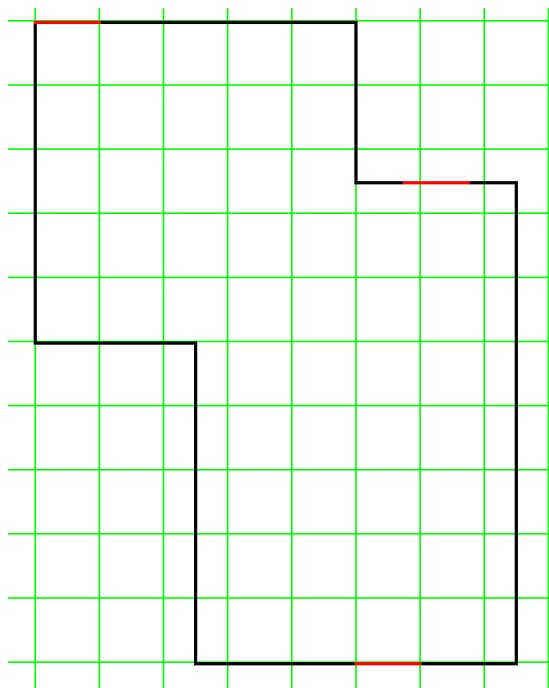


FIGURE VI.2 – Carte EKF

d'avoir la carte composée des codes QR. Inversement, il n'est pas utile pour l'algorithme EKF d'avoir la carte contenant les murs, la carte contenant la position des codes QR est suffisante.

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. Cette carte est décrite plus en détail dans le chapitre VII.

### VI.1.4 Implémentation EKF

Les sections VI.1.1, VI.1.2, VI.1.3 ont permis de définir l'ensemble des paramètres nécessaires à l'utilisation de l'implémentation de l'algorithme EKF de ce mémoire. Cette implémentation suit le pseudocode de l'algorithme EKF présenté dans la section II.2.1. L'implémentation d'EKF complète la librairie Commons Math<sup>2</sup> qui est une librairie mathématique open-source d'Apache. Celle-ci contient une série de classes qui permettent de manipuler et d'appliquer des opérations sur des matrices. Celles-ci se révèlent très utiles dans l'implémentation de l'algorithme de Kalman. Elle contient également une

---

2. Commons Math : [commons.apache.org/proper/commons-math](https://commons.apache.org/proper/commons-math)

implémentation du filtre de Kalman, mais ne contient pas d'implémentation de l'Extended Kalman Filter. C'est donc cette librairie qui a été utilisée pour implémenter l'algorithme EKF. L'ensemble du code implémentés et des adaptations des différentes librairies sont versionnés à l'aide de Git. Il est donc facile de déterminer les ajouts et les modifications utiles à ce mémoire.

### VI.1.5 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 pose 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.3 présente la représentation choisie de la covariance. La moyenne  $\mu_t$  donne la pose estimée du robot. Cette pose 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 pose 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 :

$$\text{Cov}(X, Y) = E[(X - E[X])(Y - E[Y])]$$

où  $E[\cdot]$  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 aléatoires et leur moyenne ont tendance à être de même signe et négative dans le cas contraire. Soit le vecteur de pose écrit :

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

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

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

La diagonale de la matrice de covariance est composée des variances des variables aléatoires de  $\vec{X}$ , ce qui est normal, car  $\text{Cov}(X, X) = \text{Var}(X)$ .

La matrice de covariance est une matrice symétrique, car  $Cov(X, Y) = Cov(Y, X)$ . Pour en revenir à la représentation de la matrice de covariance, l'angle d'écartement entre les deux droites de notre représentation est donné par  $Var(\theta)$ .  $Var(\theta)$  caractérise donc la dispersion des valeurs possibles de la direction du robot. Plus cette variance est petite et plus la direction estimée du robot est sûre 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<sup>3</sup>, qui permet de visualiser la covariance d'une matrice à l'aide d'une ellipse, peut être appliquée à n'importe quelle matrice de covariance.  $Var(x) = \sigma_x^2$  et  $Var(y) = \sigma_y^2$  permettent de définir la largeur et la hauteur de l'ellipse à l'aide de l'équation de l'ellipse suivante :

$$\left(\frac{x}{\sigma_x}\right)^2 + \left(\frac{y}{\sigma_y}\right)^2 = S$$

où  $S$  est obtenu grâce à la table de probabilité de la distribution du  $\chi^2$  à deux degrés de liberté. Pour un intervalle de confiance de 95 %  $S = 5.991$ . 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 :

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

où  $V_1$  correspond au vecteur propre majeur et  $\alpha$  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 la valeur propre  $\lambda$ . Cette équation est résolue à l'aide de la librairie Commons Math déjà utilisée pour manipuler les matrices de l'algorithme EKF. Cette équation possède deux solutions. Le vecteur majeur correspond au vecteur de solution qui possède la plus grande valeur propre.

---

3. Ellipse représentation : [www.visiondumy.com](http://www.visiondumy.com)

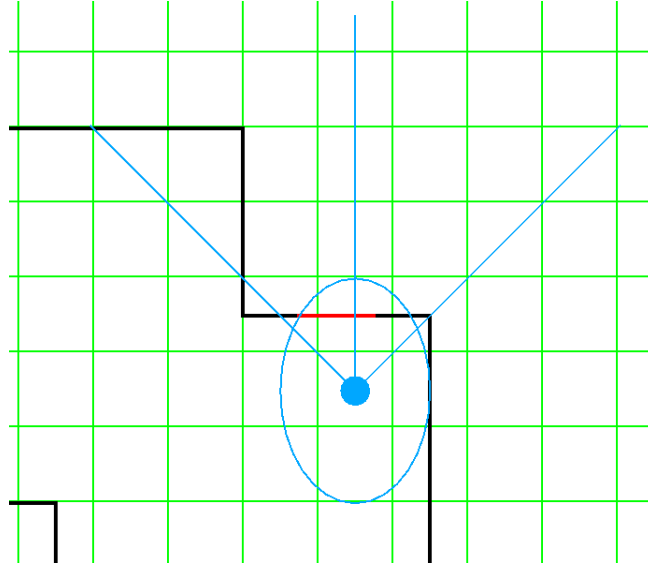


FIGURE VI.3 – Représentation de la covariance

## VI.2 Tests et Résultats

### VI.2.1 Tests et résultats de l'implémentation de EKF

L'environnement de tests construit est représenté dans la figure VI.2. Il est composé de trois codes QR représentés en rouge. Ils sont placés sur les murs qui sont représentés en noir. Les valeurs initiales de la matrice de covariance sont les suivantes :

$$\Sigma = \begin{pmatrix} 20 & 0 & 0 \\ 0 & 20 & 0 \\ 0 & 0 & \pi/2 \end{pmatrix}$$

La pose initiale est la suivante :

$$\mu = \begin{pmatrix} 110 \\ 80 \\ \pi \end{pmatrix}$$

Les valeurs des paramètres des erreurs de mouvement et d'observation ont été déterminées après avoir effectué des mesures à la main de la pression moyenne des capteurs et des moteurs. Voici les valeurs de ces paramètres :

$$\sigma_{d^z}^2 = 10$$

$$\sigma_{\rho^z}^2 = 0.01$$

$$\alpha_1 = 0.01$$

$$\alpha_2 = 0.0001$$

$$\alpha_3 = 0.01$$

$$\alpha_4 = 0.001$$

Le parcours du robot est aléatoire. Il consiste à avancer tout droit jusqu'à ce qu'une collision avec un mur soit détectée à l'aide du capteur de pression. Après la collision, le robot recule légèrement et puis tourne de quelques degrés sur sa gauche puis recommence à avancer jusqu'à la prochaine collision. À la collision suivante, le même procédé est répété. Pour que le robot parcoure tout son environnement, lorsqu'il a parcouru une longue ligne droite sans percuter de mur, il tourne légèrement sur sa droite .

Durant le parcours du robot, la matrice de covariance augmente progressivement comme prévu. Elle diminue fortement lors de la détection d'un code QR. Cette forte diminution est due à la précision importante de la caméra du smartphone. La détection du code QR ajuste également la pose du robot de façon importante. Il s'avère que l'algorithme EKF est efficace pour suivre une pose du robot. Cependant, lorsque la pose estimée est fort différente de pose réelle, cet algorithme est moins efficace.

## VI.2.2 Comparaison avec le MCL

Le même environnement est utilisé pour les tests de l'algorithme MCL. Le capteur infrarouge relève les mesures entre la distance du robot et des murs. L'algorithme est initialisé avec cent particules qui sont réparties uniformément à l'intérieur des murs. À chaque fin de mouvement, le robot balaye l'environnement avec son capteur de distance sur 45 degrés à droite et 45 degrés à gauche. Après un tour de l'environnement, le robot est déjà très sûr de sa position. La figure VI.4 montre la pose des particules après ce tour de l'environnement. La particule en bleu correspond à la pose estimée à l'aide de l'ensemble des particules. MCL permet donc d'avoir très vite une bonne estimation de la pose du robot avec le capteur infrarouge qui est de précision et de fiabilité pourtant très faibles. L'algorithme MCL est également simple à comprendre et à utiliser comparé à EKF. Cependant, il s'avère que l'algorithme MCL a son efficacité qui diminue fortement lorsque des éléments non cartographiés sont ajoutés à l'environnement. En effet, le capteur infrarouge ne fait aucune différence entre un mur cartographié et un élément non cartographié ajouté à l'environnement. Des poses erronées proches des murs sont donc favorisées lorsque le robot se trouve proche de cet élément non cartographié.

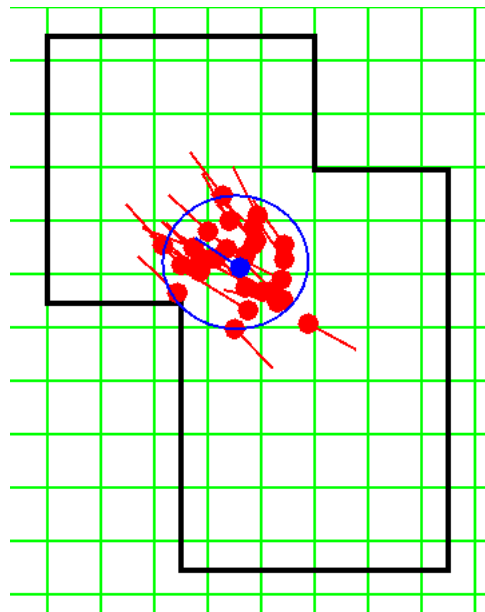


FIGURE VI.4 – Test MCL



# Chapitre VII

## Grille d'occupation et recherche de chemin

Ce chapitre est destiné à expliquer comment une grille d'occupation est complétée dynamiquement à l'aide de l'algorithme EKF pour, par la suite, construire un graphe qui permet de faire une recherche de chemin. Trois éléments permettent de mettre la carte à jour : le capteur de pression, le capteur infrarouge et le trajet du robot. Les sections suivantes décrivent comment ces éléments mettent la grille d'occupation à jour.

### VII.1 Construction de la grille d'occupation

Les valeurs de mise à jour sont proportionnelles à la covariance. Lorsque la covariance est moins importante, le robot est plus certain de sa pose. Il est donc également plus certain de la position des obstacles qu'il découvre durant son parcours. Il peut donc compléter la grille d'occupation avec un facteur de pertinence plus important.

#### VII.1.1 Capteur de pression

Le capteur de pression se déclenche lorsqu'un objet percute le pare-chocs avant. Il est cependant impossible de déterminer d'où provient la pression. Une pression sur le bord droit ou sur le bord gauche du pare-chocs est identifiée indifféremment. La figure [VII.1](#) montre que la pression du pare-chocs augmente fortement la probabilité (la nuance de rouge est forte) des cases devant toute la largeur du pare-chocs et non seulement devant le centre du robot.



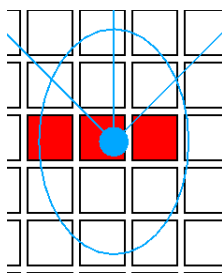


FIGURE VII.1 – Capteur de pression

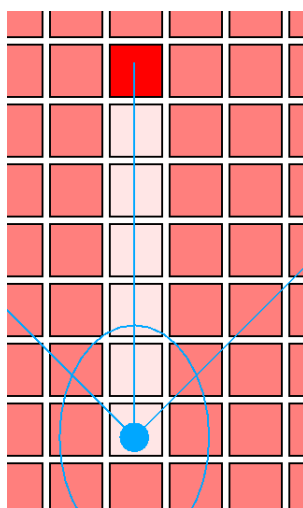


FIGURE VII.2 – Capteur infrarouge

### VII.1.2 Capteur infrarouge

Le capteur infrarouge permet de détecter la présence d'objets placés devant le robot. Il est donc possible d'augmenter la probabilité de la case où le capteur a détecté un objet. Pour les cases entre l'objet et le robot, si le capteur n'a pas détecté une distance plus courte, c'est que ces cases ne contiennent pas d'objet. Les cases entre le robot et l'objet vont donc diminuer leur probabilité d'être occupées (voir figure VII.2). Il faut cependant prendre en considération la largeur de la case et la largeur balayée par le capteur. En effet, si la case est très grande et la largeur balayée petite, il est possible que le faisceau laser ne détecte pas l'objet, car il passe à côté. Cette case est donc considérée comme vide à tort.

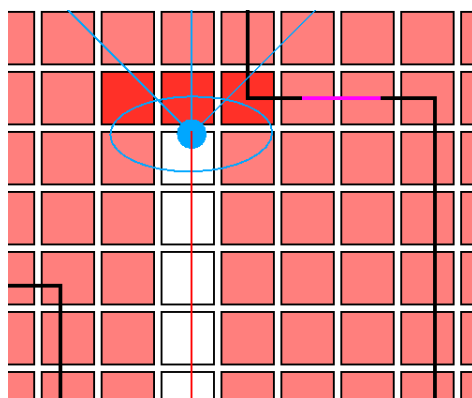


FIGURE VII.3 – Trajet

### VII.1.3 Trajet

Lors d'un trajet d'un point A à un point B, si aucun objet n'est détecté à l'aide du capteur de pression avant, la probabilité d'avoir des objets dans les cases empruntées durant le trajet est diminuée. La figure VII.3 représente cette situation où le trajet du robot est dessiné en rouge. Après avoir avancé d'une certaine distance, le robot touche un mur, la probabilité d'être dans une case occupée augmente donc près du mur. Cependant, la probabilité que des cases soient occupées diminue sur le trajet où aucun obstacle n'a été détecté.

## VII.2 Recherche de chemin

Le chapitre IV décrit en détail la manière de construire un graphe à l'aide des cartes de l'environnement. Il décrit également la manière de l'utiliser pour construire un chemin entre un point de départ et un point d'arrivée. Dans la figure VII.4, les lignes noires correspondent aux murs de l'environnement qui ont été cartographiés au préalable et enregistrés dans un fichier SVG. La position de départ du robot est représentée en bleu et la position d'arrivée en rose. Le chemin est représenté en bleu. Il est calculé à l'aide d'un algorithme de recherche de chemin dans un graphe. Le chemin est calculé en évitant de percuter les murs. Pour éviter de percuter les murs, le graphe est généré de façon à ne pas relier deux noeuds qui coupent un mur. Finalement, un espace minimum doit être respecté pour éviter de percuter les murs durant les mouvements. C'est pour cette raison que certains noeuds trop proches des murs ne sont pas reliés aux autres noeuds.

De la même façon, il est possible de diminuer l'espace praticable pour le

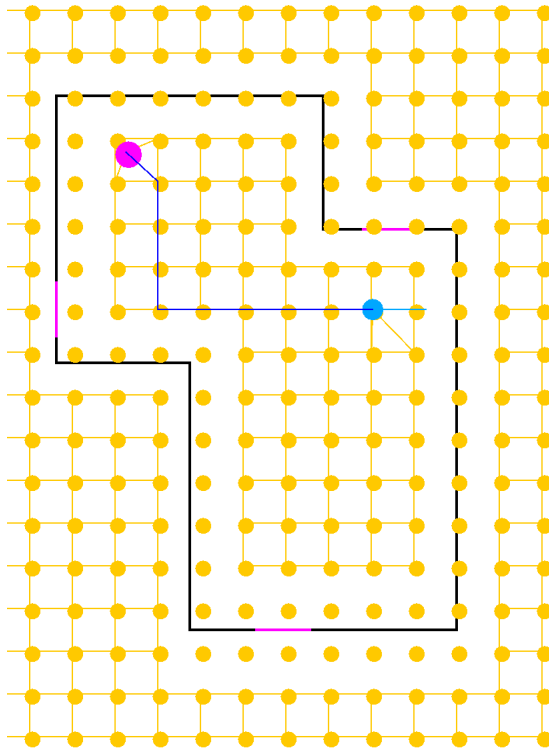


FIGURE VII.4 – Recherche de chemin

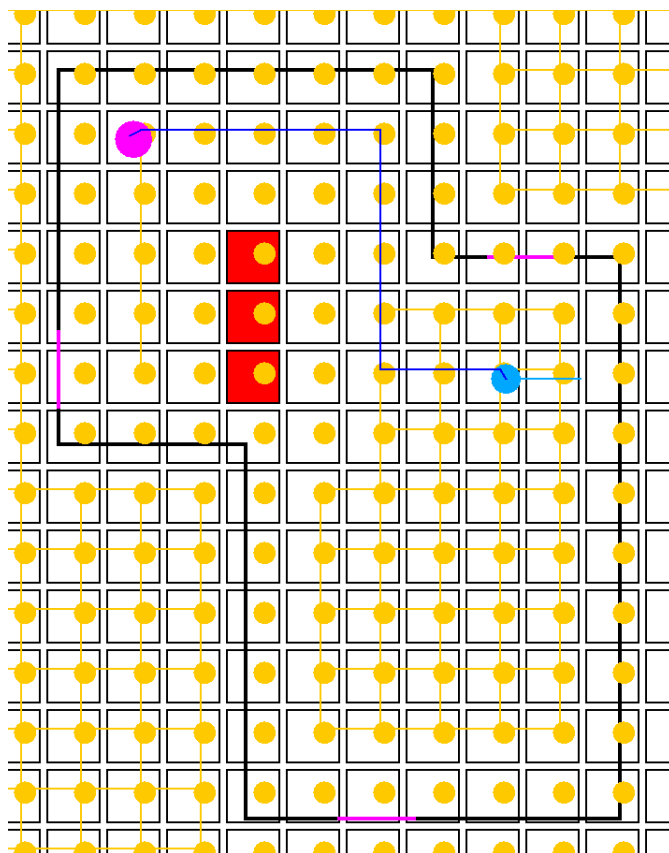


FIGURE VII.5 – Recherche de chemin dans la grille d’occupation

robot à l’aide de la grille d’occupation générée dynamiquement. Il est ainsi possible de construire dynamiquement des chemins en prenant en considération des objets non cartographiés au préalable. La figure VII.5 représente le parcours du robot entre un point A et B. Il est différent du parcours de la figure VII.4. Il est adapté pour prendre en considération les objets détectés dynamiquement et enregistrés dans la grille d’occupation. Un noeud du graphe qui est dans une case considérée comme occupée n’est pas joignable. Il n’y a donc pas d’arêtes entre un noeud dans une case occupée et un autre noeud dans une case occupée ou non. De la même façon que pour les murs, un noeud est également non joignable s’il se trouve trop proche d’une case occupée.



## Troisième partie

### Conclusions



Cette partie finale est composée d'une section « conclusions » ainsi que de propositions de recherches futures pour continuer ce mémoire. Finalement, une description de ce que ce mémoire m'a apporté est donnée. Les conclusions portent aussi bien sur les notions théoriques des algorithmes de localisation que sur les problèmes pratiques rencontrés au cours du mémoire. En effet, sans considérer la pratique, il est impossible d'avoir des algorithmes de localisation de qualité.





# Chapitre VIII

## Conclusions

### VIII.1 Implémentation

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 de déterminer l'environnement dans lequel le robot évolue pour déterminer les modèles et les algorithmes qui correspondent le mieux à ses caractéristiques. Il s'avère que la construction du robot a un impact sur les modèles. Le type de plateforme, la position des capteurs, la vitesse d'utilisation des moteurs sont autant d'éléments qui doivent être pris en considération lors de la construction du modèle de mouvement et de mesure.

### VIII.2 Composants à faible coût

Ce mémoire a démontré qu'il est possible de localiser un robot dans un environnement simple à l'aide de capteurs, de moteurs et de processeurs à faible coût. L'algorithme MCL démontre qu'il est possible d'utiliser des capteurs très peu coûteux comme des capteurs infrarouges pour se localiser dans un environnement statique. Cependant, lorsque l'environnement possède des éléments non cartographiés, l'algorithme MCL est moins efficace. L'algorithme EKF nécessite des capteurs capables de collecter des données qui sont analysées pour localiser un point de repère. Ils sont donc plus coûteux, mais restent abordables. Il s'avère également que l'extraction des points de repère

des données brutes du capteur nécessite une quantité de traitements non négligeable. Extraire des points de repère d'une image haute définition nécessite un processeur ou une carte graphique puissante. Cependant, l'algorithme EKF compense ce besoin de puissance par la faible complexité de l'étape de mise à jour de la croyance de la pose du robot. Finalement, ces algorithmes sont utilisables sans modification pour des robots différents et des capteurs différents. Seuls les modèles de mouvement et de mesure sont à adapter. Le même algorithme de localisation peut donc être utilisé avec un simple capteur infrarouge ou même avec un Lidar 3D. Cependant, l'augmentation de la taille des vecteurs d'observation et du vecteur de pose implique un surplus de consommation CPU.

### VIII.3 Recherche de chemin

L'implémentation des algorithmes de recherche de chemin démontre qu'il est possible d'utiliser les algorithmes de localisation pour construire des cartes dynamiques qui sont ensuite utilisées par les algorithmes de recherche de chemin. Les cartes dynamiques sont générées à l'aide de capteurs simples comme le détecteur de pression et le capteur de distance. Une fois le graphe construit à l'aide des cartes disponibles, il est possible d'utiliser l'ensemble des algorithmes de recherche de chemin dans un graphe. Les algorithmes de Dijkstra et  $A^*$  se sont révélés sans surprise très efficaces pour trouver des chemins dans ces graphes.

# Chapitre IX

## 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 présentent les pistes qui me semblent les plus intéressantes.

### IX.1 Analyse d'images

Dans ce mémoire, le choix d'utiliser des codes QR a permis de simplifier la détection de repères 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 repères 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 apparaît 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.

### IX.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 classes 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<sup>1</sup> ou MSRDS<sup>2</sup> qui sont des outils professionnels de haute qualité fonctionnant aussi bien sur des robots industriels que des robots de loisirs comme les drones.

### IX.3 Prise de décisions

Les algorithmes présentés dans ce mémoire sont des algorithmes qui ne prennent aucune décision. Le parcours donné au robot pour lui permettre de se localiser rapidement est soit donné étape par étape par le développeur ou par des étapes générées aléatoirement. Il est cependant intéressant que le robot puisse prendre lui-même la décision d'effectuer un certain parcours. Par exemple, un véhicule sous-marin qui doit relier un port canadien jusqu'à un port de la mer Caspienne doit-il prendre le chemin le plus court sous la glace et risquer de perdre le signal GPS ou prendre le chemin le plus long ? Il n'est plus question de simplement trouver un chemin possible comme présenté dans le chapitre IV, mais plutôt de trouver le meilleur chemin. Ce type de problème peut être résolu en pondérant les arêtes d'un graphe selon les risques d'emprunter un chemin donné. Cependant, ces couts doivent être liés à une probabilité et non à un cout fixe. Dans notre exemple, ce cout est fonction de la probabilité de perdre le signal GPS. Ce type de problème peut être résolu à l'aide d'un modèle de processus de décision markovien qui est décrit dans le livre Probabilistic Robotics [22]. Ces algorithmes probabilistes de prise de décisions sont donc dans la continuité de ceux présentés dans ce mémoire. Ils sont donc de bonnes pistes de recherche futures.

---

1. ROS : [www.ros.org](http://www.ros.org)

2. Microsoft Robotics Developer Studio 4 : [www.microsoft.com](http://www.microsoft.com)

# Chapitre X

## Enrichissements personnels

Ce chapitre est destiné à expliquer ce que la réalisation de ce mémoire m’a apporté. Elle m’a apporté de nouvelles connaissances, m’a permis d’en approfondir d’autres, et m’a également appris sur moi-même. La première et la deuxième partie de ce chapitre se concentrent plus sur les compétences et les connaissances qui ont été améliorées. Tandis que la dernière partie est plus centrée sur mes capacités de gestion de projet et mes connaissances sur moi-même. En effet, il est important de définir des objectifs réalistes en connaissant ses points forts, ses points faibles et sa tolérance à la charge de travail.

### X.1 Projets open source

La librairie Lejos ainsi que la librairie Commons Math sont deux librairies open source de taille moyenne. Bien que l’apprentissage de ces librairies m’a pris un temps certain, le fait de pouvoir réutiliser des briques déjà implémentées permet d’obtenir un travail de qualité supérieure en comparaison à un projet commencé à partir de zéro. Je pense qu’utiliser des librairies open source est la meilleure façon d’apprendre et de progresser en informatique. Le code source de ce genre de librairie est de taille importante et il est réalisé par des personnes différentes. Ceci permet lors de l’analyse de ce code source d’apprendre de chacun des programmeurs ayant ajouté leur brique à l’édifice. En effet, chacun a son style de programmation. Il est donc possible de retirer le meilleur de chacun des styles de programmation présents dans la librairie. De plus, cette technique permet d’apprendre à coder en équipe. Bien que chaque partie de code ne correspond pas toujours à nos attentes, il est primordial de l’adapter aux fonctionnalités qu’on souhaite ajouter. Ceci sans pour autant modifier le comportement déjà implémenté. De plus, les fonctionnalités et le

code créés par vos soins vont être commentés et ajustés. Ceci permet d’avoir un bon feedback sur nos erreurs de programmation. Finalement, la popularité des projets open source n’est plus à démontrer. Linux, Firefox, Eclipse, sont autant de projets de qualité et ils sont indispensables dans leur domaine.

## X.2 Domaine complet

Par sa nature complète et complexe, la robotique aborde un grand nombre de sujets et de sous-sujets, aussi bien liés à l’informatique, qu’aux mathématiques en passant par la physique et la mécanique. Ce mémoire m’a donc permis de revoir des algorithmes importants, de l’analyse de graphes, des formules physiques, des concepts probabilistes, des notions d’électronique, de l’informatique en temps réel. Je pense qu’en fin de master, il est intéressant et gratifiant de mettre en pratique une partie de l’ensemble des connaissances accumulées durant cinq années d’études. L’objectif de mettre en pratique une grande partie de mes connaissances était un souhait dès le début de ce mémoire. J’estime que cet objectif a été atteint. En plus d’avoir revu et mis en pratique des concepts connus, j’ai appris de nouvelles choses dans chacun des domaines abordés. Finalement, ce mémoire a suscité ma curiosité dans chacun des sous-domaines présentés. Bien que ce mémoire soit terminé, je vais continuer à approfondir mes connaissances pour donner des réponses aux nouvelles questions qui se sont ouvertes. En effet, un travail scientifique lève au moins autant de nouvelles questions qu’il essaye d’y répondre.

## X.3 Travail de taille importante

S’informer, définir les objectifs, faire des recherches de référence, implémenter et rédiger ce mémoire sont autant d’étapes qui ont rythmé cette dernière année de master. Autant d’étapes importantes qui doivent être réparties et incrustées dans l’ensemble des autres cours et activités personnelles. Ce mémoire m’a donc appris énormément de choses sur mes capacités de gestion du stress, sur l’évaluation de mes capacités de travail et sur l’évaluation de la charge de travail. Ces compétences sont primordiales pour mener à bien un projet. Il a donc été important de me remettre constamment en question, de réévaluer régulièrement l’état d’avancement du travail, de jauger si le travail effectué correspond bien aux attentes des directeurs, et au besoin d’adapter les objectifs. Il est possible de les adapter soit en les modifiant soit en les revoyant à la baisse ou au contraire en les revoyant à la hausse. Je pense que bien que cet exercice soit difficile et les débuts laborieux, ce mémoire m’a

permis de vraiment m'améliorer sur les différents points abordés. J'ai par exemple appris à diviser chaque partie d'un travail en sous-parties. Ceci permet d'appréhender le travail plus facilement qu'en voulant l'attaquer sur tous les fronts. Cette technique permet également d'évaluer plus facilement son état d'avancement. Il est plus facile de définir l'avancement global lorsqu'un certain nombre de sous-tâches sont terminées plutôt qu'avoir un ensemble des sous-tâches en cours d'avancement.





# 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 MEKH-NACHA, 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].