Faculté
des Sciences
Département d'informatique

UMONS
Université de Mons

Algorithmes de localisation probabilistes 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

Rapporteurs: Rapporteur 1
Rapporteur 2

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

Remerciements

Je tiens à remercier Hadrien Mélot le directeur de ce mémoire et Pierre Hauweele le codirecteur de ce mémoire pour m'avoir apporté leur expertise dans le domaine et leur aide précieuse.

Je tiens également à remercier ma mère qui m'a aidé à relire et structurer ce document.

Résumé

L'objectif de ce mémoire est de développer des aspects théoriques de la robotique en y liant une validation pratique. La localisation d'un robot dans son environnement est le principal sujet abordé. Les algorithmes de localisation probabilistes, Extended Kalman filter (EKF) ainsi que Monte Carlo localization (MCL) seront étudiés en profondeur. Toutefois, une multitude de sous-sujets en découlent, tels que la caractérisation des capteurs et des actuateurs, les systèmes d'exploitation dans la robotique, l'analyse d'images...

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

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[18][9]. 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 [10] avoir conçu un système de pilotage automatique pour automobile. Ce prototype est capable de se déplacer dans la circulation automobile sans assistance humaine. À ce jour, de nombreux constructeurs automobiles travaillent sur des voitures autonomes. On peut citer Audi, Toyota[2], Nissan[1], Mercedes-Benz.

La multitude de capteurs qui équipent ces véhicules est évidemment primordiale. Toutefois, sans traitement et croisement de ces mesures, il est impossible de développer un véhicule autonome. Pour cela, des algorithmes ont été développés. Ces algorithmes permettent aux véhicules de se localiser dans leur environnement pour ensuite établir leur parcours. Ces algorithmes doivent prendre en compte que les valeurs des capteurs peuvent être entachées d'erreurs. Ils doivent également prendre en compte que les capteurs ne permettent de capter qu'une partie de l'environnement du véhicule. Des véhicules comme la voiture autonome de Google embarquent un grand nombre des capteurs de hautes précisions. Toutefois, équiper des voitures de ces capteurs se révèle encore très onéreux. Il est donc intéressant de se demander quelles sont les limites des algorithmes de localisation avec des capteurs à faibles couts. Ces algorithmes sont-ils perfectibles ? Les robots de nettoyage domestique[7] sont des exemples de robots qui ont des capteurs simples et peu onéreux, mais qui doivent se déplacer dans un environnement complexe.

Le livre de Sebastian Thrun « Probabilistic Robotics » [16] qui est la bible dans le domaine de la robotique a été une ressource importante pour la rédaction de ce mémoire. Rappelons que Sebastian Thrun est l'ingénieur principal qui a lancé le projet « Google driverless car ». Le mémoire de Pierre Hauweele [6] donne certaines démonstrations théoriques plus en profondeur alors que ce mémoire est plus centré sur les aspects pratiques.

Table des matières

Table des figures

Chapitre I

Problème de localisation

I.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'instrument 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 erreur. Toutes ces informations sont donc approximatives. Malgré ces erreurs d'approximations, grâce à la quantité d'informations accumulées durant son parcours cette personne a de fortes chances d'être de plus en plus certaine de sa position. En effet, en début de parcours, cette personne peut supposer être à un ensemble d'endroits différents à la suite d'un manque d'informations en sa possession. Par la suite grâce aux nouvelles informations elle peut procéder par élimination pour déterminer sa position.

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

pour le promeneur ces informations sont entachées d'erreurs de précisions. 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. Une solution naïve serait de vouloir acheter des détecteurs et des moteurs toujours plus précis. Cependant, le cout des capteurs plus précis est plus important. De plus, des erreurs peuvent être impossibles à gérer à l'aide de matériel plus précis. En effet, il peut arriver que les roues du robot n'adhèrent pas parfaitement à la route. Ce qui entraine le glissement des roues et donc bien que le robot reste immobile, les moteurs enregistreront un mouvement. Finalement, les capteurs ont également des limitations physiques, il est par exemple impossible pour une caméra de voir à travers les murs.

La prédictibilité de l'environnement est un élément important dans le choix d'appliquer ou non des algorithmes probabilistes. Dans le cas d'un environnement bien structuré comme une chaine de montage, le degré de prédictibilité est bien plus important que lorsque le robot évolue en ville ou dans une maison. L'imprédictibilité augmente fortement lorsque le robot évolue dans un environnement en présence de personnes. Les algorithmes probabilistes permettent de pallier à l'imprédictibilité d'un environnement.

I.2 Définition formelle du problème

I.2.1 Modèle de mouvement et d'observation

L'objectif de la localisation d'un robot est de définir sur une carte la position du robot à un instant donné. Cette position est dénotée par le vecteur x_t. Dans la suite de ce mémoire, la position du robot est définie par ses coordonnées dans un espace à deux dimensions ainsi que par son orientation. Les algorithmes présentés ne se limitent pas à cette situation et peuvent être utilisés pour un espace à un nombre de dimensions supérieures. L'état des bras des robots industriels[13] peut également être représenté à l'aide de l'angle de chaque articulation du bras. Toutefois, les principes sont plus simples à comprendre dans cette situation et cette situation est souvent suffisante pour les robots mobiles. Formellement, la position d'un robot dans un espace à deux dimensions peut-être définit par le vecteur :

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

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

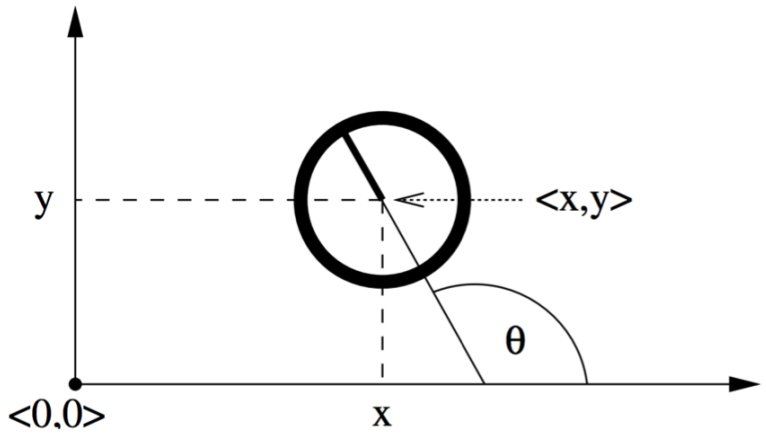

FIGURE I.1 – Position d'un robot dans un espace à deux dimensions

Pour déterminer la position du robot dans le temps, il faut prendre en considération les contrôles du robot (notés : u_t) ainsi que les observations effectuées par le robot (notées : z_t). Les algorithmes qui seront présentés par suite suivent l'hypothèse de Markov. C'est-à-dire que l'état x_t ne dépend que de l'état x_{t-1} ainsi que des contrôles u_t et des observations courantes z_t(voir figure I.2).

Formellement, les contrôles peuvent 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 γ_t correspond à l'angle de rotation du robot. L'odométrie permet de déterminer les commandes u_t. Les valeurs de u_t permettent de définir itérativement les nouvelles positions à l'aide de l'équation suivante :

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

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

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

où d_t^z correspond à la distance entre le robot et l'élément détecté et ρ_t correspond à l'angle formé entre l'orientation du robot et la position de l'élément.

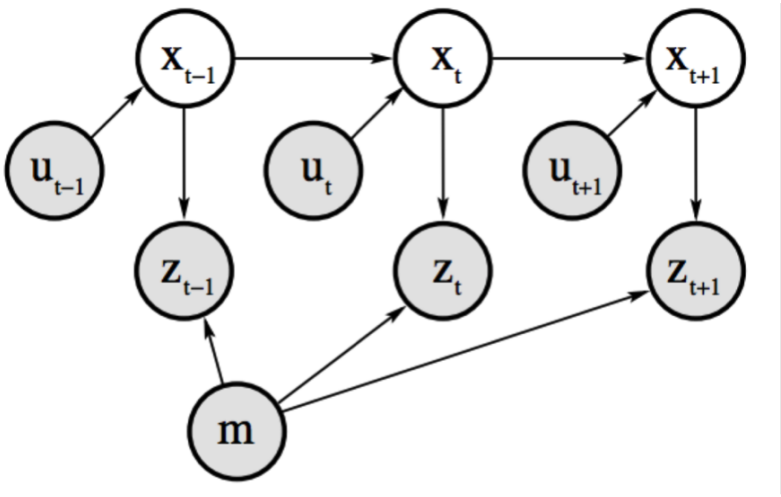

FIGURE I.2 – Hypothèse de Markov permettant de déterminer la position actuelle du robot

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.

Ces équations ne sont vraies que si les valeurs retournées par les moteurs et capteurs étaient 100 % juste, ce qui n'est évidemment pas le cas. En effet, ces données sont sujettes à des erreurs de mesures. Il est intéressant de remarquer que si l'odométrie n'était pas sujette à des erreurs de mesures, il ne serait pas utile d'équiper ses robots de capteurs pour les localiser, l'odométrie serait suffisante.

I.2.2 Algorithme de localisation de Markov

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

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

L'algorithme 1 qui est décrit en pseudocode correspond à l'algorithme de localisation de Markov qui est à la base de tout les algorithmes de localisation qui sont présentés dans ce mémoire. Il décrit la mise à jour de la position x_{t-1} vers l'état x_t. Il prend en paramètre la croyance de la position précédente (c'est-à-dire la fonction de densité de probabilité du vecteur de position x),

le contrôle courant, les observations courantes ainsi que la carte dans laquelle le robot évolue. Il est constitué d'une boucle principale, qui itère sur toutes les valeurs possibles pour la position x_t. Cette boucle contient deux étapes importantes. La première étape se nomme «la prédiction» et consiste à calculer une croyance temporaire $\overline{bel}$ de la position du robot à l'aide de u_t et de la croyance de l'étape précédente $bel(x_{t-1})$. 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.

Algorithm 1 Localisation de Markov

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

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

Les algorithmes probabilistes souffrent de deux défauts importants en comparaison aux techniques traditionnelles de programmation en robotique. La première est la complexité en temps de calcul de l'algorithme qui aug-

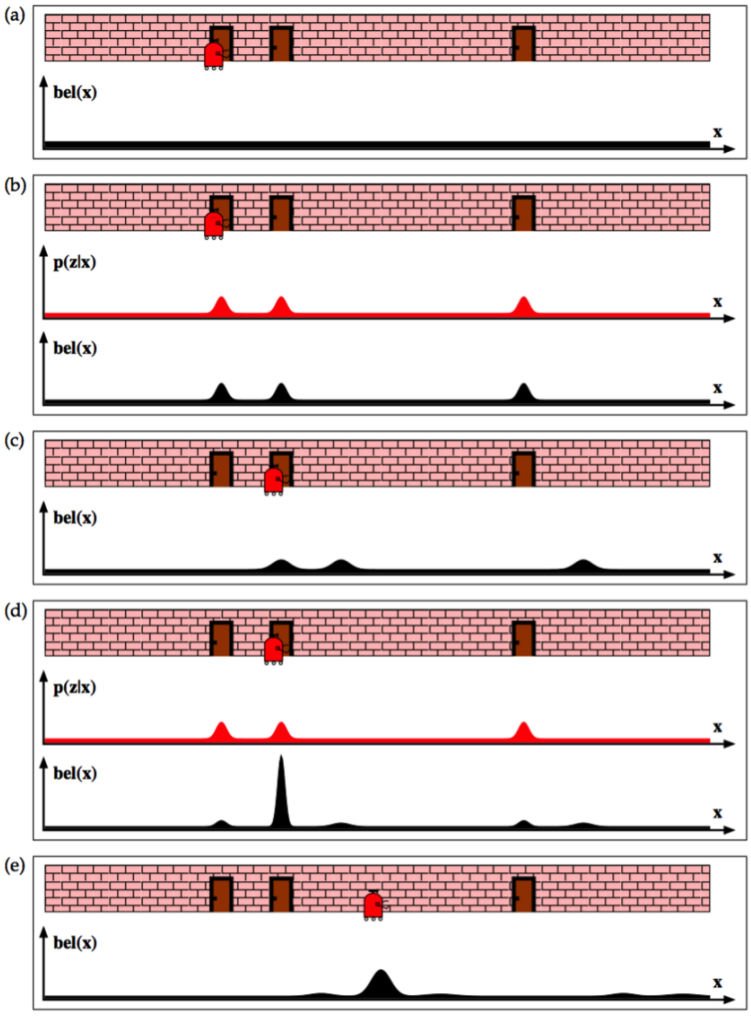

FIGURE I.3 – Idée générale de la localisation de Markov

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

I.3 Algorithmes de résolution du problème de localisation

L'algorithme de Markov permet de donner l'idée générale des algorithmes de localisation. Cependant pour pouvoir implémenter concrètement un algorithme de localisation un certain nombre de questions sont encore ouvertes. La plus importante correspond à la représentation de la fonction de probabilité. Deux grandes approches existent. La première définit une fonction 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.

I.3.1 Algorithme paramétrique(EKF)

Les algorithmes paramétriques permettent de représenter les croyances de la position d'un robot à l'aide de lois de probabilité. Les concepts mathématiques de l'algorithme de Kalman ont été développés dans les années 60 [8]. Dans le cas de l'algorithme Kalman Filter(KF) la loi de probabilité est une loi normale. Elle dépend donc de deux paramètres son espérance μ et son écart type σ. Cette loi normale est une loi normale multivariée lorsque le vecteur de position est composé de plusieurs éléments. On définit alors la

moyenne de cette fonction multivariée comme suit

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

Algorithm 2 Kalman filter

procedure KalmanFilter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)
  $\overline{\mu_t} \leftarrow A_t\mu_{t-1} + B_tu_t$ ▷ prédiction
  $\overline{\Sigma_t} \leftarrow A_t + \Sigma_{t-1}A_t^T + R_t$ ▷ prédiction
  $K_t \leftarrow \overline{\Sigma}_tC_t^T(C_t\overline{\Sigma}_tC_t^T + Q_t)^{-1}$ ▷ Kalman Gain
  $\mu_t \leftarrow \overline{\mu}_t + K_t(z_t - C_t\overline{\mu}_t)$ ▷ mise à jour
  $\Sigma_t \leftarrow (I - K_tC_t)\overline{\Sigma}_t$ ▷ mise à jour
  return μ_t, Σ_t
end procedure

Algorithm 3 Extended Kalman filter

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

L'algorithme Extended Kalman filter(EKF) correspond à la version non linearaire de l'algorithme du filtre de Kalman. Cette variante a été développée quelques années plus tard par la NASA[15] pour faire face au fait que la plupart des systèmes physiques ne sont pas linéaires. Pour ce faire les fonctions g et h ont été introduites. Ces fonctions ne doivent pas obligatoirement être linéaire mais doivent être dérivables. Contrairement ou filtre de Kalman classique où les fonctions sont obligatoirement linéaire pour préserver des fonctions de répartition gausienne.

I.3.2 Algorithme non paramétrique(MCL)

À l'inverse des algorithmes paramétriques, les algorithmes non paramétriques ne sont pas basés sur une loi de probabilité connue dont l'algorithme

arrange les paramètres pour correspondre au mieux à la croyance de la position. Dans les algorithmes non paramétriques, la croyance est représentée par nombre déterminé de positions supposées. Une probabilité est associée à ces positions supposées. Plusieurs techniques existent pour représenter ses positions supposées.

La première technique consiste à découper la carte de l'environnement en une grille où chaque élément de la grille correspond à une position supposée[17]. Pour représenter l'orientation du robot, il faut multiplier le nombre de cases par le nombre d'angles d'orientation que peut prendre le robot (voir I.5).Dans cette représentation, seules trois orientations sont possibles. Ces trois orientations correspondent aux trois plans de la représentation. Comme on peut s'en rendre compte, il est très important de définir la bonne granularité de la découpe. Une découpe trop importante augmente le temps de calcul, alors qu'une grille trop peu découpée rend la localisation trop peu précise. Dans ce type de découpe, plus la carte est grande et plus le temps de calcul est important.

Dans le cas de l'algorithme de Monte Carlo localization (MCL) aussi appelé Particle filter localization [14] car il utilise un filtre à particule, une approche différente a été choisie. Dans cet algorithme (voir l'algorithme 4) la croyance de la position est représentée par M particules(voir I.4). Chaque particule est considérée comme une hypothèse sur la position du robot. Plus une région de la carte contient de particules et plus la probabilité que le robot s'y trouve est grande. Contrairement aux algorithmes basés sur la découpe de la carte en une grille, MCL n'implique pas un temps de calcul supplémentaire lorsque la taille de la carte augmente. Cependant, il est possible de choisir d'augmenter le nombre de particules pour augmenter la précision.

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

I.3.3 Comparaison de MCL et EKF

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

FIGURE I.4 – Illustration de MCL, les traits gris correspondent aux particules et le niveau de rouge représente la probabilité associée

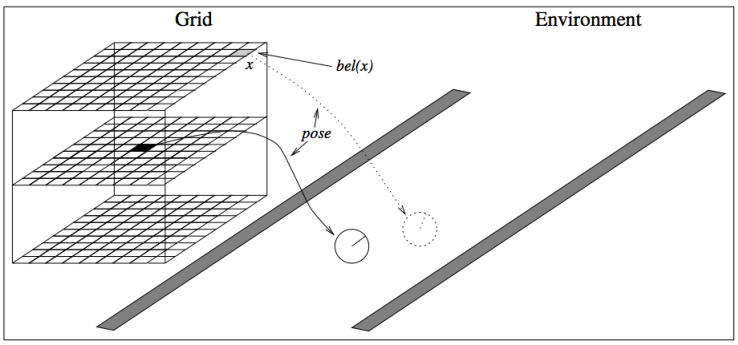

FIGURE I.5 – Illustration de la carte en grille

Algorithm 4 MCL

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

Cependant, MCL est plus robuste à 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 position est proche d'une distribution gaussienne et a donc de mauvaises performances lorsque la croyance correspond plutôt à distribution multimodale. Pour pallier à ce problème, l'algorithme classique EKF a été amélioré. Multi-hypothesis tracking (MHT)[3] filtre permet de représenter la croyance à l'aide d'un mixte de plusieurs gaussienne et donc d'avoir un algorithme plus robuste et efficient.

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

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

I.4 Simultaneous Localization And Mapping

Les algorithmes Slam (Simultaneous Localization And Mapping) sont l'étape suivante de l'indépendance des robots. En effet, dans les algorithmes de simple localisation, la carte de l'environnement doit être construite avant de la passer en paramètre aux algorithmes de localisation. Comme son nom l'indique, dans un algorithme Slam la carte est construite en parallèle avec la localisation du robot.

Algorithm 5 EKFSLAM

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

$F_x \leftarrow \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

$\overline{\mu}_t \leftarrow \mu_{t-1} + F_x^T \begin{pmatrix} d\cos \\ d\sin \\ \gamma \end{pmatrix}$

$G_t \leftarrow I + F_x^T \begin{pmatrix} 0,0 \\ 0,0 \\ 0,0 \end{pmatrix}$

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

$Q_t \leftarrow \begin{pmatrix} \sigma_r^2 & 0 & 0 \\ 0 & \sigma_r^2 & 0 \\ 0 & 0 & \sigma_r^2 \end{pmatrix}$

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

$j \leftarrow c_t^i$

if landmark j never seen before **then**

$\begin{pmatrix} \overline{\mu}_{j,x} \\ \overline{\mu}_{j,y} \end{pmatrix} \leftarrow$

end if

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

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

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

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

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

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

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

end for

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

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

return μ_t, Σ_t

end procedure

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

TABLE I.1 – Comparaison EKF et MCL

Chapitre II

Lego Mindstorms

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

II.1 Description du hardware

II.1.1 Brique intelligente

La brique intelligente programmable (voir Figure II.1) est le cerveau des robots EV3. Elle est dotée d'une interface à six boutons lumineux qui changent de couleur pour indiquer l'état d'activité de la brique, d'un affichage à haute résolution noir et blanc, d'un hautparleur intégré, d'un port USB, d'un lecteur de cartes mini SD, de quatre ports d'entrée et de quatre ports de sortie. La brique prend également en charge la communication USB, Bluetooth et Wifi avec un ordinateur. Son interface programmable permet à la fois de programmer et de journaliser des données directement sur la brique. Elle est compatible avec les appareils mobiles et est alimentée par des piles AA ou par la batterie CC rechargeable EV3. La brique EV3 fournit l'énergie pour les capteurs et les moteurs. Elle permet de récupérer et de traiter les informations des capteurs et d'envoyer des commandes aux moteurs par le

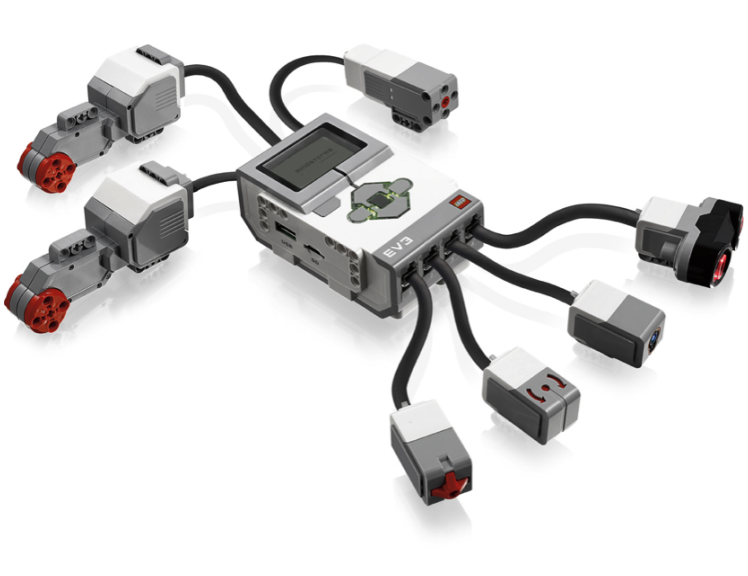

FIGURE II.1 – Brique EV3

biais des ports d'entrée et de sortie. Ce qui rend la connexion et l'utilisation des capteurs et des moteurs très facile.

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

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

II.1.2 Moteurs et capteurs

Le pack de base de la bique EV3 est composé de deux moteurs moyens, un petit moteur, un capteur de pression, un capteur de distance infrarouge et d'un capteur de couleurs. Il est possible d'acheter des capteurs Legos supplémentaires tels qu'un gyroscope, un capteur de distance ultrasonique et une boussole. Il est également possible d'acheter d'autre type de capteurs développés par des sociétés indépendantes de Lego. C'est possible grâce au support de protocole standard que doivent adopter les capteurs de données.

On peut par exemple citer I^2C qui est un bus de données conçu par Philips et qui est très répandu dans le monde de l'électronique. On voit donc en vente des capteurs GPS, des capteurs RFID, des capteurs d'humidité qui sont compatible avec la brique EV3

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

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

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

Le capteur tactile analogique EV3 est un outil simple, mais extrêmement précis, capable de détecter toute pression ou relâchement de son bouton frontal. Les utilisateurs pourront construire des systèmes de commande marche/arrêt, créer des robots capables de s'extraire d'un labyrinthe et découvrir l'utilisation de cette technologie dans des appareils tels que les instruments de musique numériques, les claviers d'ordinateur ou l'électroménager.

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

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

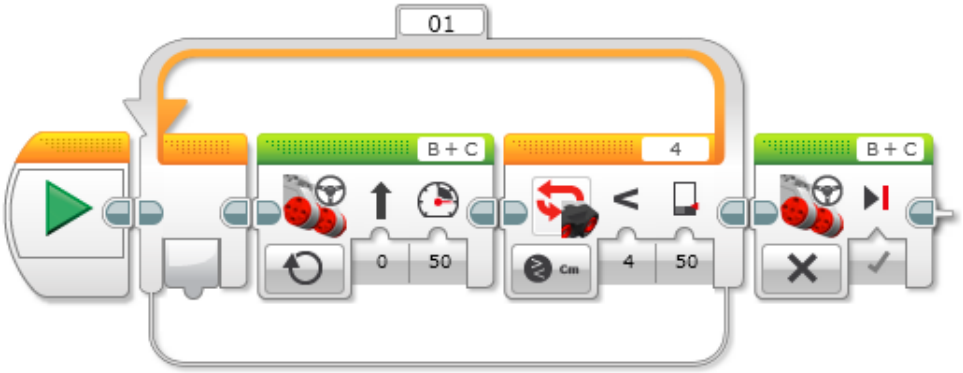

FIGURE II.2 – Robolab

II.2 Description du software

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

II.3 Lejos

Lejos est un microprogramme libre destiné à remplacer le microprogramme originalement installé sur la brique Lego Mindstorm. Lejos inclut une machine virtuelle Java permettant de développer des robots dans le langage Java. Ce qui a donné le nom Lejos qui est le mot Legos où le « g » a été

remplacé par un « j » pour Java. Pour la version EV3 de la brique Lego, cette machine virtuelle Java est mise à disposition par Oracle. Lejos est disponible sur la brique RCX, NXT et EV3. Depuis son lancement jusqu'à aujourd'hui Lejos dispose d'une communauté active et de taille assez importante. Le dépôt officiel du code source est sourceforge.net. Entre le 01-01-2015 et le 09-04-2015 la version 0.9.0-beta de la brique EV3 a été téléchargée 2700 fois. En effet, l'outil Lejos est couramment utilisé dans les universités et dans les hautes écoles pour l'apprentissage de la robotique et du langage Java. L'implémentation orientée objet permet aux étudiants d'étudier la robotique avec le niveau d'abstraction qu'ils désirent. Il est ainsi possible de développer des algorithmes de localisation sans se soucier des adresses hexadécimales des moteurs et des capteurs. Des chercheurs de l'université de Porto au Portugal ont déjà utilisé Lejos pour implémenter un algorithme qui cartographie d'un son environnement [12].

II.3.1 MCL

Lejos fournit dans sa librairie une implémentation de l'algorithme MCL. Dans cette implémentation les données de mesures proviennent d'un capteur de distance.

Chapitre III

Implémentation personnelle

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

III.1 Description du robot construit

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

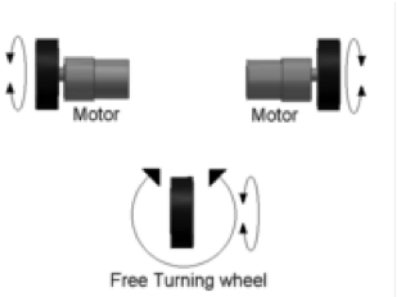

FIGURE III.1 – Differential wheeled robot

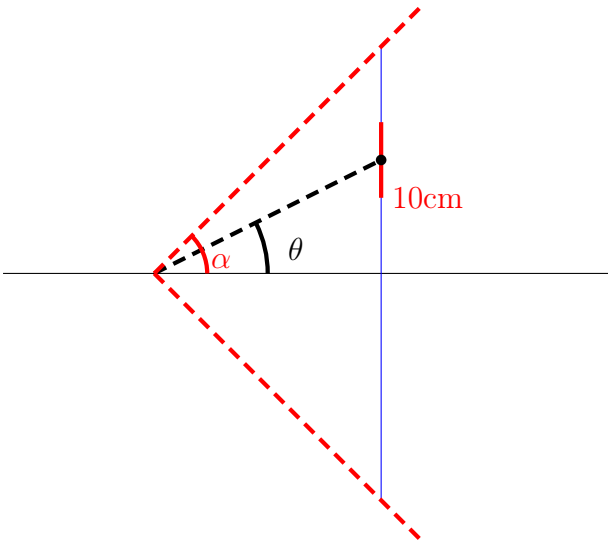

FIGURE III.2 – Évaluation de la distance des codes QR

III.2 Détection de Feature avec la caméra du smartphone

Les codes QR sont des éléments faciles à identifier pour la caméra d'un smartphone. De nombreuses librairies de qualité ont déjà été développées pour détecter et décoder des codes QR. Zbar [1] est une de ces librairies open source. Elle est disponible sur Android et IOS. Pour ce mémoire, une application permettant d'estimer la distance du smarphone au code QR a été développée à l'aide de la librairie Zbar. Pour pouvoir utiliser les codes QR pour estimer la distance entre eux et le smartphone, les codes QR doivent être d'une dimension donnée (dans ce mémoire : un carré de 10cm de coté). La librairie Zbar renvoie la dimension du code QR en nombre de pixels captés par la caméra. À l'aide d'un étalonnage de la caméra qui consiste à déterminer l'ouverture de l'objectif et du calcul trigonométrique suivant, il est possible de déterminer la distance des codes QR de 10cm de coté(voir III.2).

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

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

1. Zbar : http ://zbar.sourceforge.net/

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

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

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

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

III.3 Implémentation de l'algorithme EKF

L'implémentation EKF utilise la technique de détection de codes QR comme features présentées dans la section III.2.

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

III.4 Tests et résultats de l'implémentation de EKF

III.5 Comparaison avec le MCL

III.6 EKF Slam

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

Algorithm 6 EKFCodeQR

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

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

$G_t \leftarrow \begin{pmatrix} 1 & 0 & -d_t \sin(\theta) \\ 0 & 1 & d_t \cos(\theta) \\ 0 & 0 & 1 \end{pmatrix}$

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

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

$\overline{\mu}_t \leftarrow \mu t - 1 + \begin{pmatrix} d \cos \\ d \sin \\ \gamma \end{pmatrix}$

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

$Q \leftarrow \begin{pmatrix} \sigma_r^2 & 0 & 0 \\ 0 & \sigma_r^2 & 0 \\ 0 & 0 & \sigma_r^2 \end{pmatrix}$

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

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

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

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

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

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

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

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

end for

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

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

return μ_t, Σ_t

end procedure

Chapitre IV

Conclusion

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

IV.1 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.

Dans ce mémoire le choix d'utiliser des codes QR a permis de simplifier la détection de features 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 features des données brutes des capteurs. De nombreux algorithmes permettent d'extraire des features à l'aide d'une caméra. Ce domaine de recherche discute des différents éléments qui sont considérés comme de bons features. Par exemple, la détection de bord se base sur de fortes modifications de la lumière dans une image, l'ensemble des points où cette forte modification de lumière apparait est considéré comme un bord. L'algorithme de Canny est un exemple d'algorithme de détection de bords[4].De la même manière les algorithmes de détection de coins tentent de déterminer les carac-

téristiques d'un coin. L'algorithme de Moravec [11] et de Harris et Stephens [5] sont des exemples d'algorithmes de détection de coins.

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

1. ROS : http ://www.ros.org/
2. Microsoft Robotics Developer Studio 4 : https ://www.microsoft.com/en-us/download/details.aspx ?id=29081

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

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

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

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

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

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

[9] 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.

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

[11] 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.

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

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

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

[15] 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).

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

[17] 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.

[18] 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).