Département de Physique Appliquée

Mémoire présenté à

LA FACULTE DES SCIENCES ET TECHNIQUES

Pour obtenir le diplôme de

MASTER ES SCIENCES ET TECHNIQUES

« GENIE ELECTRIQUE »

Option : **A**UTOMATIQUE et **I**NFORMATIQUE **I**NDUSTRIELLE

par Mr HAOUZAN Zakaria

Sous le thème :

Localisation par les amers et filtrage de Kalman en vue d'une planification de trajectoire pour un robot mobile

Réalisé au Laboratoire LSEEET

Soutenu le 20/07/2020

Devant le Jury composé de :

Président	:	Pr. ABDELJALIL EL KARI (FST Marrakech)
Examinateur	:	Pr. HASSAN BELAHRACH (FST Marrakech)
Encadrant	:	Pr. HASSAN AYAD (FST Marrakech)

N° 08/AII/2020 **Année Universitaire : 2019**

Remerciements

Le bon déroulement des travaux de recherche réalisés dans ce stage n'aurait pu avoir lieu sans le concours de nombreuses personnes que je tiens à remercier :

En tout premier lieu Je présente mes remerciements particuliers à monsieur HASSAN AYAD, pour son excellent encadrement pendant toute la durée de ce stage. Je ne peux que lui être reconnaissant et je le remercie pour le temps et l'aide qu'il m'a accordé, et pour les riches discussions qui m'ont poussé à donner le meilleur de moi-même.

Je tiens aussi à remercier AHMED OUTLGHIT pour les conseils et remarques judicieuses qui ont grandement aidé à améliorer la qualité de ce stage.

C'était un réel plaisir de travailler avec des personnes très motivées, intelligentes, enthousiastes et passionnées par leur travail.

Abstract

Dynamic state estimation is one of the problems of mobile robotics that has received a lot of research effort lately, especially on location-related applications. The aim of this project is to study a series of methods with a solid statistical basis, called Bayesian filters. Within this type of filters, we will study those known as Kalman filters. In particular, we will look at three types of filters, each designed to deal with particular situations. The algorithms evaluated in this project will be: the classical Kalman filter algorithm, the extended Kalman filter, the unscented Kalman filter .

Résumé

L'estimation dynamique de l'état est l'un des problèmes de la robotique mobile qui a reçu beaucoup d'efforts de recherche ces derniers temps, notamment sur les applications liées à la localisation. Le but de ce projet est d'étudier une série de méthodes ayant des bases solides en matière de statistiques, appelés filtres bayésiens. Dans ce type de filtres, nous étudierons ceux connus sous le nom de filtres de Kalman. Nous examinerons en particulier trois types de filtres, chacun d'eux étant conçu pour traiter avec des situations particulières. Les algorithmes évalués dans le cadre de ce projet seront les suivants : l'algorithme classique Le filtre de Kalman, le filtre de Kalman étendu, le filtre de Kalman non parfumé.

Liste des figures

Figure 1.1 Système de SLAM.....4
Figure 1.2 Principe du capteur ultrasons.....6
Figure 1.3 Principe de fonctionnement d'un télémètre laser. 6
Figure 1.4 Télémètres infrarouges. 7
Figure 1.5 Principe de positionnement en utilisant le GPS 7
Figure 1.6 Compas CMP03 et CPS9410.....8
Figure 1.7 codeur optique. 9
Figure 1.8 Repérage du robot dans un référentiel orthonormé direct R1.....10
Figure 1.9 Caractérisation du roulement sans glissement. 11
Figure 1.10 Fonction de densité de probabilité. 15
Figure 1.11 Fonction de densité de probabilité 2D. 16
Figure 2.1 la linéarisation de la fonction d'entrée.....22
Figure2.2-A les points sigma sont sélectionnés que leur moyenne u et covariance P sont les mêmes que la distribution de probabilité. 24
Figure 2.2-B La moyenne et la covariance P du point sigma sont utilisées pour calculer la nouvelle estimation de l'état. 24
Figure 3.1 Envoi de données à V-REP par appel de fonction non bloquant.....31
Figure 3.2 lecture des données de V-REP via appel de fonction bloquante. 31
Figure 3.3 la mesure de la vitesse des roues avec la fonction bob_getEncoders 33
Figure 3.4 la mesure de la position et l'orientation du robot avec la fonction bob_getPos 34
Figure 3.5 la map de l'environnement donne par la fonction bob_getLaser.....34
Figure 3.6 Le problème essentiel du SLAM, Une estimation simultanée du robot et du Il est nécessaire de disposer de points de repère. Les emplacements réels ne sont jamais connus ou mesurés directement. 35
Figure 3.7 Modification du vecteur de pose du robot. 37
Figure 3.8 Modèle cinématique d'entraînement différentiel. 38
Figure 3.9 Scène et trajectoire du robot dans V-REP 43
Figure 3.10 trajectoire du robot dans MATLAB utilisant l'algorithme EKF-SLAM. 44

Table des matières

Remerciements ..... **i**
Abstract ..... **ii**
Résumé ..... **iii**
Liste des figures ..... **iv**
Table des matières ..... **v**
Introduction générale **1**
Chapitre I : Problématique de localisation en robotique mobile. **2**
I-1. Introduction: 3
I-2. Le Principe du SLAM : 4
I-3. Les Capteurs en robotique mobile : 5
I-4. Capteurs de distances : 6
I-4.1 Captures ultrasons 6
I-4.2 Télémètres lasers 6
I-4.3 Capteurs infrarouges 7
I-5. Mesure d'orientation et de position : 7
I-5.1 Mesure de la position : GPS 7
I-5.2 Mesure d'orientation Géomètres gyromètres 8
I-5.3 Compas et boussoles 8
I-5.4 Les codeurs 8
I-6. Caméras : 9
I-6.1 Les amers 9
I.7- Odométrie : 9
I-8. Modélisation et outils probabilistes: 10
I-8.1 Modélisation Cinématique 10
I-9 Estimation et apprentissage : 14
I-10 Distribution gaussienne 1D : 15
I.11-Gaussiennes Multi-variables: 16
I-12 Conclusion 17
Chapitre II : Algorithmes de localisation **18**
II-1. Introduction 19
II-2. Modèles de l'espace d'État : 19
II-3. Filtre de KALMAN : 20
II-4. Filtre de KALMAN étendu (Extended Kalman Filter : EKF) 21

II-5. Filtre de Kalman non–Parfumé (Uncented Kalman Filter :UKF) 23
II-6. Conclusion 27
Chapitre III : La localisation par les amers 28
III-1. La Virtual Platform d'expérimentation V-REP 29
III-2. La communication entre MATLAB et V-REP 30
III-3. Le Framework de contrôle 32
III.4. Description de l'algorithme filtre de KALMAN avec les balises 35
III-4.1-Étape de prédiction 37
III-4.2 Étape de mise à jour 39
III.5. Simulation : 41
III-6. Conclusion 44
Conclusion générale 46
Références 47
Annexe : 50

Introduction générale

Les robots mobiles sont largement utilisés dans les environnements industriels pour le transport de produits par exemple. Le plus souvent ces tâches sont répétitives et suivent un chemin bien défini, parfois même bien matérialisé comme des lignes sur le sol ou des amers artificiels. Il y a actuellement une forte tendance à élargir les milieux où évoluent les robots à des environnements de bureaux ou à des environnements domestiques. Les types d'applications possibles sont innombrables. Cela peut être des tâches de nettoyage et d'entretien.

L'objectif de notre de projet est de doter notre robot d'une autonomie en se localisant par rapport aux amers. Pour ce faire, le système doit généralement accomplir trois tâches de base qui sont la localisation, la planification et la navigation. Parmi ces tâches, la localisation relativement à l'environnement occupe une place de choix puisqu'elle détermine le bon déroulement des deux autres. Pour effectuer une meilleure localisation exige de détecter des amers connus qui permettent au robot de corriger l'estimation odométrique de sa position. L'odométrie est en effet fragile comme méthode de localisation du robot pour plusieurs raisons : erreurs dues aux incertitudes de mesure et de calibration, etc. qui peuvent entraîner des erreurs cumulatives avec le temps. Pour cela nous avons utilisé les amers (constituant une carte). Il s'agit donc de maximiser la correspondance entre des cartes locales successivement construites avec l'évolution du mouvement du robot sur sa trajectoire. En lus nous avons utilisé les techniques d'estimation (filtrage de Kalman, Kalman Etendu, …etc.) qui sont à la base de combinaison de mesures extéroceptives et proprioceptives. Le télémètre laser, le sonar, sont utilisés pour acquérir les données extéroceptives.

Le présent rapport est scindé en trois chapitres. Le premier Chapitre présente une introduction à la problématique de localisation en robotique mobile. Le deuxième Chapitre regroupe les algorithmes de localisation. Le troisième Chapitre est consacré à la localisation par les amers.

Chapitre I : Problématique de localisation en robotique mobile.

I-1. Introduction:

Avec l'évolution de la technologie, il est devenu possible d'utiliser des robots intelligents dans de nombreux domaines, même ceux où il y a un risque élevé, comme alternatives aux êtres humains dans l'accomplissement de ces tâches. Lors de la réalisation de ces tâches, ces robots doivent naviguer en toute sécurité dans des milieux connus ou inconnus afin de découvrir leurs environnements. Ainsi, pour pouvoir interagir d'une manière cohérente avec son environnement, un robot a besoin de construire la carte de cet environnement et de connaître son emplacement dans cette carte en permanence. Cela cause un paradoxe, car la construction de la carte nécessite la connaissance de la positionne et aussi l'estimation de la position à son tour exige la carte de l'environnement. Par conséquent, les deux processus, l'estimation de la position et la création de la carte, doivent être réalisés simultanément. Ce problème est l'un des plus importants dans le domaine de la robotique ces dernières années, et est connu sous le nom de SLAM 'Simultaneous Localisation and Mapping'. C'est un processus d'auto-localisation d'un système de perception autonome, par rapport à une carte de l'environnement crée et mise à jour par le système lui-même. En particulier, pour de nombreuses applications SLAM, il est fondamental que ces tâches soient menées en temps réel. Plusieurs algorithmes SLAM ont été développés pour résoudre le problème de cartographie et localisation. Ces algorithmes sont catégorisés en fonction du type de capteurs et la nature de l'approche mathématique utilisée. Les premiers algorithmes SLAM étaient principalement basés sur l'utilisation des capteurs laser capable de fournir une information précise sur la profondeur des objets dans la scène.

La première formulation du SLAM fut établie par les travaux de Smith et Cheeseman [1] et de Durrant-Whyte [2]. Les bases statistiques de l'écriture mathématique reliant les amers et les incertitudes de mesures sont établies. Ils montrent qu'il doit y avoir un fort degré de corrélation entre les positions estimées des amers observés, ces corrélations doivent augmenter avec les observations successives. Des résultats de recherche sur la navigation de robots mobiles utilisant des algorithmes exploitant le filtre de Kalman. Il en résulte la parution d'un article clef du SLAM [3]. Il montre que lorsqu'un robot se déplace dans un environnement inconnu en effectuant des acquisitions successives d'observations d'amers, les estimations des positions des amers sont corrélées entre elles.

N'oublie pas qu'y a d'autres travaux qui traitent Le problème de la localisation des robots mobile sans introduire le mapping ,[4][5][6] base sur des algorithme comme Monte Carlo, Filtre à particule ou filtre de KALMAN ,etc… . L'article [6] introduit une approche pour la localisation indoor d'un robot mobile basé sur la technologie Ultra-Wideband en utilisant un filtre KALMAN étendu (EKF) comme technique possible pour améliorer la localisation. L'approche proposée permet d'utiliser une IMU (unité de mesure inertielle) à faible coût dont les performances sont améliorées par une procédure d'étalonnage. Les résultats obtenus montrent que le filtre permet d'obtenir un meilleur résultat en termes de localisation grâce à l'estimation du biais et du facteur d'échelle.

I-2. Le Principe du SLAM :

De nombreuses recherches tentent de résoudre le problème de la localisation et la cartographie simultanées (SLAM), les principales méthodes de SLAM sont basées sur des méthodes d'estimation statistiques. Il s'agit de filtrage statistique permettant d'estimer l'état d'un système dynamique à partir des données en provenance d'un ou plusieurs capteurs [7].

Le SLAM se décompose en plusieurs étapes :

— Une acquisition de données des divers capteurs (Perception).

— Une fusion des données provenant des différents capteurs (Filtrage et analyse).

— Une prédiction sur la nouvelle position et orientation du robot (Estimation de l'état).

— Une mise en correspondance de la carte en tenant compte des données capteurs et des prédictions.

— Une mise à jour de la carte globale et de la position/orientation du robot.

Cette structure du SLAM est représentée sur la figure 1.1.

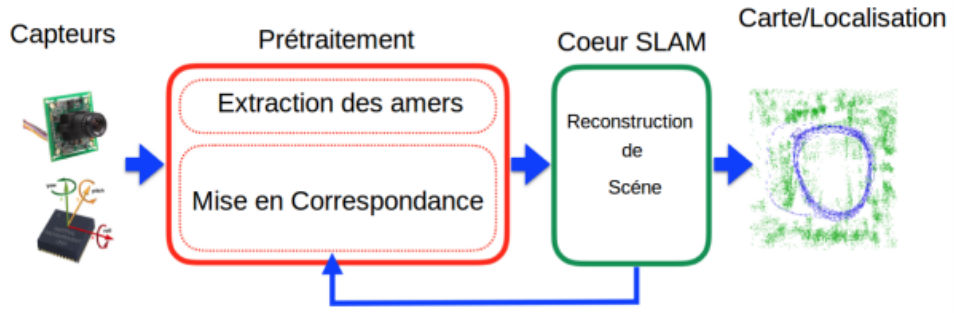

Figure 1.1 Système de SLAM

Les algorithmes développés dans ce domaine peuvent être classés selon plusieurs critères :

- Les types de capteurs utilisés.
- Les méthodes de calcul adoptées.
- Les types de cartes représentant l'environnement.
- Les types d'environnement... etc.

Le problème SLAM est-il résolu ?

Cette question est relativement souvent posée au sein de la communauté de robotique. La difficulté de répondre à cette question réside dans les cas d'applications. En particulier, la maturité d'un algorithme de SLAM peut être évaluée en tenant en compte les aspects suivants :

- Robot : les capteurs disponibles (résolution, taux d'échantillonnage), les ressources de calculs disponibles.
- Environnement : surface plane ou milieu en trois dimensions, présence des amers naturels ou artificiels, quantité d'éléments dynamiques.

- Performance requise : la précision souhaitée dans l'estimation de l'état du robot, la précision et le taux de réussite (pourcentage de tests dans lesquels les limites de précision sont remplies), temps d'exécution maximal, la taille maximale de la zone cartographiée.

Pour le moment, les algorithmes de SLAM qui sont considérés résolus sont décrits dans les cas suivants :

- ✓ La cartographie d'une zone intérieur 2D avec un robot équipé de codeurs de roue et un scanner laser, avec une précision suffisante (<10cm) et robustesse suffisante (taux d'échec bas), peut être considéré résolue (un exemple de système industriel effectuant ce type de SLAM est la Solution de Navigation implantée sur Kuka [8]).
- ✓ Un SLAM basé sur la vision dans le cas d'un robot se déplaçant lentement (Mars rovers [9], robots domestiques [10])
- ✓ Un SLAM basé sur l'odométrie visuelle et inertielle (NVI) [11] peut être considéré comme domaine de recherche mature.

Le filtre de kalman (KF), l'une des méthodes SLAM les plus populaires, peut être utilisé dans les processus linéaires et les systèmes de mesure, mais l'algorithme KF est limité au système de distribution gaussien [12]. Pour résoudre ce problème, les chercheurs ont proposé l'algorithme du filtre Kalman étendu (EKF) [13] et [14]. En raison de la complexité de calcul de la matrice jacobienne, l'EKF ne peut pas produire de grandes cartes et satisfaire aux exigences en temps réel. De plus, l'EKF est vulnérable aux problèmes liés aux données [15]. L'article [16,17] a montré que l'algorithme filtre de particule (PF) sans calcul de la matrice jacobéenne a été introduit pour traiter le problème SLAM [18] dans lequel, chaque particule porte sa propre carte qui peut être calculée en fonction de la trajectoire de cette particule. Chaque particule du PF correspond à un état estimé. [19,20] montrent que PF emploie plusieurs particules pour représenter la distribution des estimations. [21] a résolu le problème du SLAM en combinant la puissance de deux algorithmes de cartographie populaires, le filtre à particules Rao-Blackwellized (RBPF) et le filtre d'information étendu (EIF). Ces méthodes sont toutes basées sur l'approche de filtrage de Bayes qui intègre des informations de transition d'état de système interne et des résultats de mesure externes [22].

I-3. Les Capteurs en robotique mobile :

Afin de pouvoir se localiser un robot doit être capable de mesurer l'environnement qui l'entoure et/ou d'évaluer son déplacement. Pour se faire il doit être doté de capteurs. Il est possible de classer ces capteurs en deux grandes familles : les capteurs proprioceptifs et les capteurs extéroceptifs. Les capteurs proprioceptifs sont des capteurs qui vont mesurer l'état interne du robot (le nombre de tours des roues motrices par exemple). En général ces capteurs sont utilisés afin d'évaluer le déplacement du robot. Les capteurs extéroceptifs représentent l'ensemble des capteurs qui mesurent l'environnement dans lequel se déplace le robot. Il existe une multitude de capteurs extéroceptifs : sonars, lasers, caméras... Cette section présente différents capteurs classiquement utilisés pour les problèmes de localisation en robotique mobile.

I-4. Capteurs de distances :

Les capteurs de distances (télémètres) sont des capteurs très utilisés en robotique mobile. Ils permettent d'avoir une information de distance entre le robot et l'obstacle le plus proche (dans une direction donnée). On peut distinguer deux technologies de capteurs : les capteurs utilisant les ultrasons et les capteurs utilisant la lumière.

I-4.1 Captures ultrasons

Le capteur ultrasons (figure 1.2) est composé d'un émetteur et d'un récepteur. L'émetteur envoie des ultrasons qui sont réfléchis par l'obstacle puis récupérés par le récepteur.

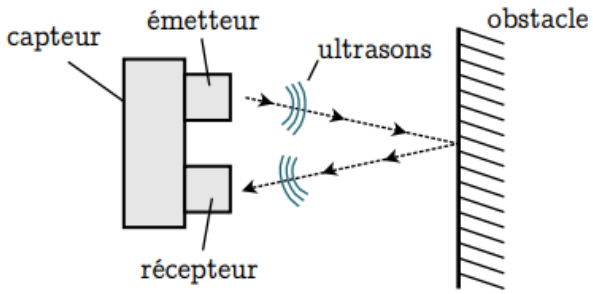

Figure 1.2 Principe du capteur ultrasons.

I-4.2 Télémètres lasers

Le principe d'un télémètre laser est sensiblement le même que celui d'un capteur ultrasons, sauf que cette fois le capteur émet un rayon laser (une onde lumineuse). Le télémètre laser est donc lui aussi composé d'un émetteur et d'un récepteur, et il évalue le déphasage entre l'émission et la réception. La Figure 1.3 illustre le principe de fonctionnement d'un télémètre laser.

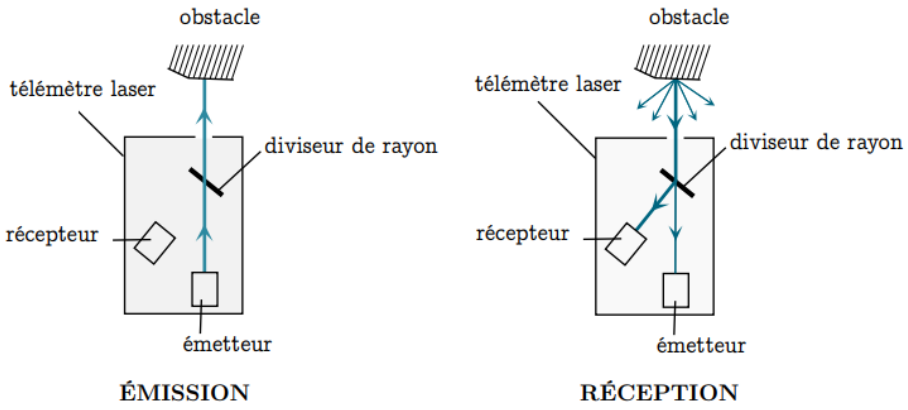

Figure 1.3 Principe de fonctionnement d'un télémètre laser.

On remarque qu'une partie du rayon laser réfléchi emprunte le même chemin que lors de l'émission. C'est cette partie qui est interceptée et redirigée vers le récepteur à l'aide d'un diviseur de rayon. La

mesure fournie alors par le télémètre laser est ponctuelle. Ce qui signifie que pour détecter un obstacle il faut que ce dernier soit exactement devant le capteur (sur la trajectoire du rayon laser).

I-4.3 Capteurs infrarouges

Les capteurs infrarouges sont constitués d'un ensemble émetteur/récepteur fonctionnant avec des radiations non visibles figure 1.4, dont la longueur d'onde est juste inférieure à celle du rouge visible. La mesure des radiations infrarouges étant limitée et, en tout état de cause, la qualité très dégradée au-delà d'un mètre, ces dispositifs ne servent que rarement de télémètres.

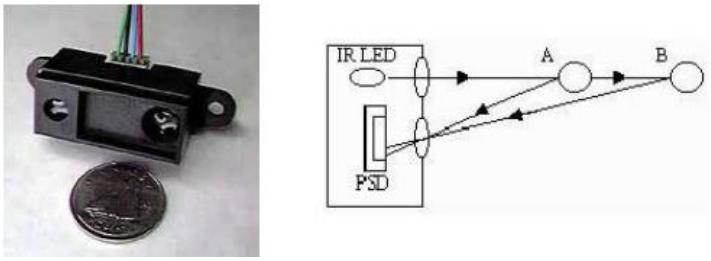

Figure 1.4 Télémètres infrarouges.

I-5. Mesure d'orientation et de position :

I-5.1 Mesure de la position : GPS

Le GPS (Global Positioning System) fonctionne avec un ensemble de satellites, qui effectuent des émissions synchronisées dans le temps. Par recoupement des instants d'arrivée des signaux et de la position des satellites émetteurs, les récepteurs peuvent calculer leur position.

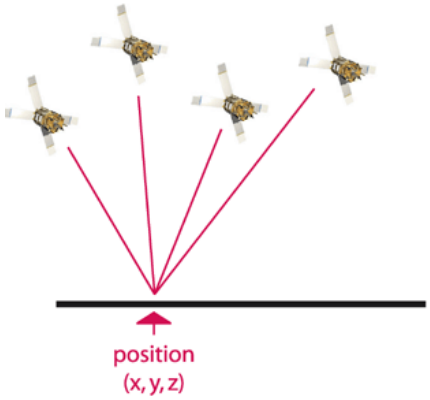

Figure 1.5 Principe de positionnement en utilisant le GPS.

Le principe de calcul de la position est basé sur une triangulation figure 1.5, à l'aide de quatre signaux reçus simultanément qui permet la localisation (du récepteur GPS) dans l'espace en 3 dimensions (longitude, latitude et altitude). En termes de précision, la localisation ainsi obtenue et entachée d'une erreur de l'ordre de la quinzaine de mètres, ce qui n'est bien évidemment pas suffisant [24].

I-5.2 Mesure d'orientation Géomètres gyromètres

Les gyromètres sont des capteurs proprioceptifs qui permettent de mesurer l'orientation du corps sur lequel ils sont places, ceci par rapport à un référentiel fixe et selon un ou deux axes. Montes sur un robot mobile plan, un gyromètre a un axe permet donc de mesurer son orientation. Il existe plusieurs sortes de gyromètres : mécaniques et optiques pour les plus connus, mais aussi a structures vibrantes, capacitifs, etc.

Les gyromètres optiques exploitent le fait que la vitesse de la lumière reste inchangée dans tout référentiel. Deux faisceaux lasers sont émis depuis une même source, pour parcourir des chemins identiques, l'un dans le sens des aiguilles d'une montre, l'autre en sens opposé. Lors de la mise en rotation du gyromètre il existe une différence de marche des deux rayons et des interférences apparaissent. On peut alors déduire la vitesse de rotation du système de cette mesure.

I-5.3 Compas et boussoles

Les compas et les boussoles fournissent une information d'orientation par rapport à une référence fixe (nord magnétique typiquement). Ils sont donc extéroceptifs. Les modules schématisés sur la figure 1.6 sont des compas électroniques capables de détecter le nord. Par déduction, un module comme le CMP03 indiquera son orientation avec une résolution de 3 à 4 degrés environ, ce qui reste acceptable en complément de l'odométrie.

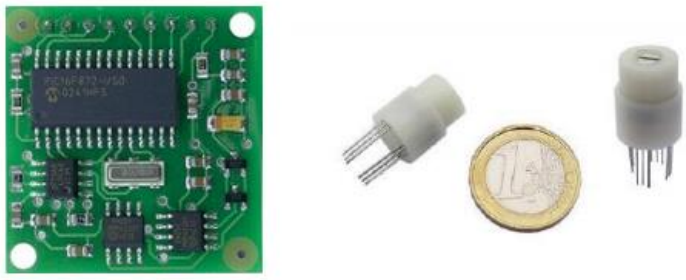

Figure 1.6 Compas CMP03 et CPS9410.

I-5.4 Les codeurs

Un codeur est un dispositif électromécanique qui peut mesurer le mouvement ou la position. La plupart des codeurs utilisent des capteurs optiques pour fournir des signaux électriques sous forme de trains d'impulsions, qui peuvent à leur tour être traduits en mouvement, direction ou position. Les codeurs rotatifs sont utilisés pour mesurer le mouvement de rotation d'un arbre. La figure 1.7 montre les composants fondamentaux d'un codeur rotatif, qui se compose d'une diode électroluminescente (LED), d'un disque et d'un détecteur de lumière sur le côté opposé du disque. Le disque, qui est monté sur l'arbre rotatif, présente des motifs de secteurs opaques et transparents codés dans le disque.

Lorsque le disque tourne, les segments opaques bloquent la lumière et, lorsque le verre est transparent, la lumière peut passer. Cela génère des impulsions d'ondes carrées, qui peuvent ensuite être interprétées en position ou en mouvement.

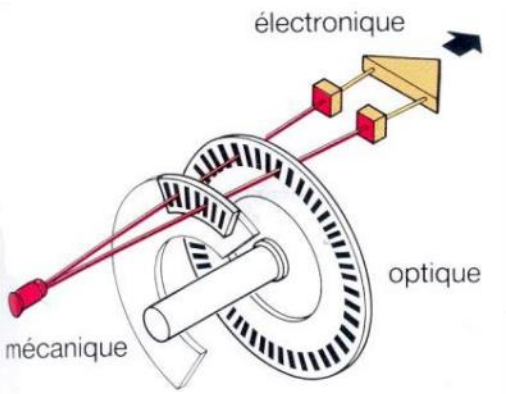

Figure 1.7 codeur optique.

I-6. Caméras :

La localisation visuelle (localisation à l'aide d'images) comprend une multitude de problématiques et de méthodes attachées à ces problématiques. Généralement la localisation visuelle consiste à identifier des amers et à se localiser en utilisant par exemple une carte de caractéristiques ou encore une carte topologique. Il est donc possible de diviser la localisation visuelle (dans un paradigme de localisation à l'aide d'amers) en deux tâches principales : l'identification d'amers et la localisation.

I-6.1 Les amers

Classiquement, deux types d'amers sont utilisés : les amers artificiels et les amers naturels. Les amers artificiels correspondent à des points d'intérêts ajoutés à l'environnement afin de faciliter la localisation du robot (Satellites GPS, phares...). Ils correspondent à des cibles facilement identifiables par le robot. L'avantage de ce type d'amer réside dans la simplicité d'utilisation. Cependant ces amers doivent être ajoutés à l'environnement ce qui représente une contrainte non négligeable.

Les amers naturels correspondent à des points d'intérêts naturellement présents dans l'environnement. Pour un environnement intérieur on pourra par exemple considérer les portes, les fenêtres, les coins de pièces... L'utilisation d'amers naturels permet de localiser un robot sans modifier l'environnement.

I.7- Odométrie :

L'odométrie correspond à l'utilisation des capteurs dans le but d'évaluer le déplacement d'un robot. L'odométrie permet une évaluation relative du déplacement du robot entre deux instants. C'est une technique très utilisée pour la localisation en robotique mobile, généralement couplée avec des capteurs extéroceptifs [27]. Bien que l'odométrie d'un robot soit généralement construite à l'aide de capteurs proprioceptifs (roues codeuses...), on notera qu'il existe aussi des techniques d'odométrie basées sur des capteurs extéroceptifs, comme des caméras [25, 26].

I-8. Modélisation et outils probabilistes:

La modélisation mathématique est une étape très importante pour la commande des robots. Deux types de modèles sont généralement utilisés lors de la commande, à savoir : le modèle cinématique et le modèle dynamique. D'après la littérature, on rencontre plusieurs types de robots à savoir : les robots de type uni-cycle, les robots de type tricycle, et les robots de type voiture. Dans le cadre de notre travail, nous utiliserons un robot de type uni-cycle à cause de sa simplicité de construction et de ses propriétés cinématiques intéressantes.

I-8.1 Modélisation Cinématique

Dans l'étude cinématique, seules les vitesses sont prises en compte. Le mouvement d'un robot mobile à conduite différentielle est caractérisé par deux contraintes cinématiques à savoir : Aucun glissement latéral, Roulement sans glissement.

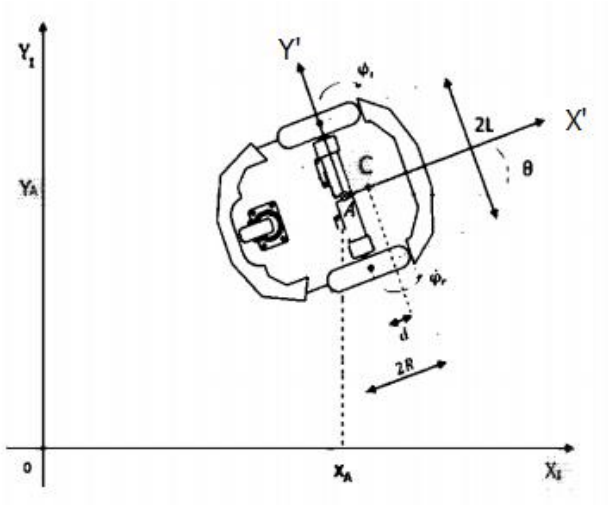

Figure 1.8 Repérage du robot dans un référentiel orthonormé direct R1.

Notons R1 = {X, Y} un repère fixe quelconque et R2 = {X', Y'} un repère mobile lié au robot.

Soient, $q_1 = [x, y, \theta]^T$ un point du repère R1 et $q_2 = [x', y', \theta']^T$ un point du repère R2. Les points q_1 et q_2 sont liés par la matrice orthogonale R(θ).

$$\dot{q}_1 = R(\theta)\dot{q}_2 \qquad (1.1)$$

Avec $R(\theta) = \begin{pmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$

La figure 1.8 présente le robot mobile de type uni-cycle dans les repères fixe et mobile.
A : est le point milieu de l'axe des roues.
2R : représente le diamètre de la roue du robot.
2L : représente la largeur du robot.
$\dot{\varphi}_d$ Et $\dot{\varphi}_g$: représentent respectivement la vitesse de rotation de la roue droite et de la roue gauche.
θ : est l'angle d'orientation du robot.

Le mouvement du robot est caractérisé par deux contraintes non-holonomes. Si le robot peut instantanément se déplacer en avant ou en arrière mais il ne peut pas se déplacer à droite et à gauche sans que les roues glissent, on dit qu'il possède une contrainte non holonome à savoir : Aucun glissement latéral, Roulement sans glissement. Par contre si chaque roue est capable de se déplacer en avant et de côté, on dit qu'il s'agit d'un comportement holonome du robot. Dans notre cas le robot est caractérise par deux contraintes non-holonomes d'où il doit respecter les deux hypothèses suivant :

Aucun glissement latéral: Cette contrainte signifie simplement que le robot peut se déplacer uniquement en avant et en arrière, mais pas latéralement. Cela signifie que la vitesse du robot associée au point A est nulle le long de l'axe latéral dans le repère mobile, soit $y_A{}' = 0$.

En utilisant la matrice de rotation R(θ), l'expression de la vitesse du robot associée au point A dans le repère fixe est :

$$q_2 = [x'_A, 0, \theta'_A]^T \quad \text{et} \quad \dot{q}_1 = R(\theta)\dot{q}_2$$

$$\begin{cases} \dot{x}_A = \dot{x'}_A . \cos(\theta) \\ \dot{y}_A = \dot{x'}_A . \sin(\theta) \end{cases}$$

$$\dot{x}_A \sin(\theta) = \dot{y}_A . \cos(\theta) \qquad (1.2)$$

Roulement sans glissement: La contrainte de roulement sans glissement représente le fait que chaque roue maintient un point en contact avec le sol comme indiqué dans la figure 1.9. Les hypothèses 1 et 2 ont été développées dans [28].

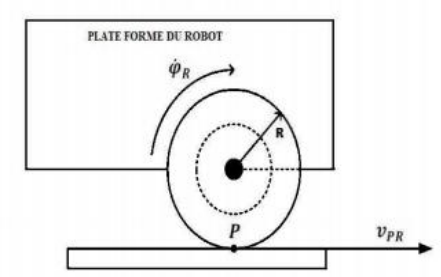

Figure 1.9 Caractérisation du roulement sans glissement.

Les expressions des positions généralisées et des vitesses généralisées dans le repère fixe en fonction des coordonnées du point A sont données par :

Roue droite

$$\begin{cases} x_p = x_A + L \sin(\theta) \\ y_p = y_A - L \cos(\theta) \end{cases} \qquad (1.3)$$

$$\begin{cases} \dot{x}_p = \dot{x}_A + L\dot{\theta} \cos(\theta) \\ \dot{y}_p = \dot{y}_A + L\dot{\theta} \sin(\theta) \end{cases} \qquad (1.4)$$

Roue gauche

$$\begin{cases} x_p = x_A - L\sin(\theta) \\ y_p = y_A + L\cos(\theta) \end{cases} \quad (1.5)$$

$$\begin{cases} \dot{x}_p = \dot{x}_A - L\dot{\theta}\cos(\theta) \\ \dot{y}_p = \dot{y}_A - L\dot{\theta}\sin(\theta) \end{cases} \quad (1.6)$$

En utilisant la matrice de rotation R(θ) et en l'appliquant à la roue droite on a:

$$\begin{pmatrix} \dot{x}_p \\ \dot{y}_p \\ \theta \end{pmatrix} = \begin{pmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} \dot{x}'_p \\ \dot{y}'_p \\ \theta \end{pmatrix} \quad (1.7)$$

Avec $\dot{y}'_p = 0$ signifie que la vitesse au point P de la roue droite est nulle (car aucun glissement latéral). Ainsi,

$$\begin{pmatrix} \dot{x}_p \\ \dot{y}_p \\ \theta \end{pmatrix} = \begin{pmatrix} \dot{x}'_p cos\theta \\ \dot{y}'_p \sin\theta \\ \theta \end{pmatrix} \quad (1.8)$$

Avec la vitesse de la roue droite est $v_d = \dot{x}_p = R\dot{\varphi}_d$

Pour la roue droite L'équation (1.8) dévient:

$$\begin{cases} \dot{x}_p\cos(\theta) = \dot{x}_A * \cos(\theta)^2 \\ \dot{y}_p\sin(\theta) = \dot{y}_A * \sin(\theta)^2 \end{cases} \quad (1.9)$$

En sommant les deux équations de 1.9

$$\dot{x}_p\cos(\theta) + \dot{y}_p\sin(\theta) = R\dot{\varphi}_d \quad (1.10)$$

Pour la roue gauche L'équation (1.8) dévient:

$$\begin{cases} \dot{x}_p\cos(\theta) = \dot{x}_A * \cos(\theta)^2 \\ \dot{y}_p\sin(\theta) = \dot{y}_A * \sin(\theta)^2 \end{cases} \quad (1.11)$$

En sommant les deux équations de 1.11

$$\dot{x}_p\cos(\theta) + \dot{y}_p\sin(\theta) = R\dot{\varphi}_g \quad (1.12)$$

On peut former le système d'équation des deux roues ou $\dot{x}_{pd}$ la position de la roue droite et $\dot{x}_{pg}$ la position de la roue gauche

$$\begin{cases} \dot{x}_{pd}\cos(\theta) + \dot{y}_{pd} * \sin(\theta) = R\dot{\varphi}_d \\ \dot{x}_{pg}\cos(\theta) + \dot{y}_{pd} * \sin(\theta) = R\dot{\varphi}_g \end{cases} \quad (1.13)$$

En introduisant les équations (1.4) et (1.6) dans (1.13) on obtient :

$$\begin{cases}(\dot{x}_A + L\dot{\theta}\cos\theta)\cos\theta + (\dot{y}_A + L\dot{\theta}\sin\theta)\sin\theta = R\dot{\varphi}_d \\ (\dot{x}_A - L\dot{\theta}\cos\theta)\cos\theta + (\dot{y}_A - L\dot{\theta}\sin\theta)\sin\theta = R\dot{\varphi}_g\end{cases} \quad (1.14)$$

Les deux hypothèses et les équations (1.14) et (1.2) produisent les contraintes suivantes :

$$\begin{cases}-\dot{x}_A\sin(\theta) + \dot{y}_A.\cos(\theta) = 0 \\ \dot{x}_{pd}\cos(\theta) + \dot{y}_{pd}\sin(\theta) - R\dot{\varphi}_d = 0 \\ \dot{x}_{pg}\cos(\theta) + \dot{y}_{pg}\sin(\theta) - R\dot{\varphi}_g = 0\end{cases} \quad (1.15)$$

De l'équation (1.13) on peut écrire: $A(q)\dot{q} = 0$ (1.16)

A(q) est la matrice de contraintes non-holonomes donnée par:

$$A(q) = \begin{pmatrix}-sin\theta & cos\theta & 0 & 0 & 0 \\ cos\theta & sin\theta & L & -R & 0 \\ cos\theta & sin\theta & -L & 0 & -R\end{pmatrix} \quad (1.17)$$

$\dot{q}$ Représente la dérivé de la coordonnée généralisée q, donnée par : $\dot{q} = [\dot{x_A}, \dot{y}_A, \dot{\theta}, \dot{\varphi_d}, \dot{\varphi_g}]^T$

L'équation (1.16) nous permet d'obtenir l'expression des vitesses linéaires des roues droite et gauche au point de contact (P).

$$\begin{cases}v_{pd} = v_A + L\dot{\theta} \\ v_{pg} = v_A - L\dot{\theta}\end{cases} \quad (1.18)$$

Avec v_A la vitesse du point A, v_{pg} est la vitesse de la roue droite au point P et v_{pd} est la vitesse de la roue gauche au point P.

En posant: $\begin{cases}v = v_A \\ \dot{\theta} = \omega\end{cases}$ (1.19)

$$\begin{cases}v_{pd} = v_d \\ v_{pg} = v_g\end{cases} \quad (1.20)$$

On obtient l'expression de la vitesse linéaire v et la vitesse angulaire ω du robot mobile en fonction des vitesses de rotation de la roue gauche $\dot{\varphi}_d$ et de la roue droite $\dot{\varphi}_g$.

$$\begin{cases}v = \frac{v_d + v_g}{2} = \frac{R(\dot{\varphi}_d - \dot{\varphi}_g)}{2} \\ \omega = \frac{v_d - v_g}{2L} = \frac{R(\dot{\varphi}_d - \dot{\varphi}_g)}{2L}\end{cases} \quad (1.21)$$

Dans le repère mobile les coordonnées du point A sont :

$$\begin{cases}\dot{x}'_A = v = \frac{R(\dot{\varphi}_d - \dot{\varphi}_g)}{2} \\ \dot{y}'_A = 0 \\ \dot{\theta}'_A = \omega = \frac{R(\dot{\varphi}_d - \dot{\varphi}_g)}{2L}\end{cases} \quad (1.22)$$

En se servant de l'équation (1.1), on peut écrire

$$\begin{pmatrix} \dot{x}_A \\ \dot{y}_A \\ \dot{\theta}_A \end{pmatrix} = \begin{pmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} \dot{x}'_A \\ \dot{y}'_A \\ \dot{\theta}'_A \end{pmatrix} \quad (1.23)$$

En remplaçant l'équation (1.22) dans (1.23), on obtient:

$$\begin{pmatrix} \dot{x}_A \\ \dot{y}_A \\ \dot{\theta}_A \end{pmatrix} = \begin{pmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} \frac{R(\dot{\varphi}_d - \dot{\varphi}_g)}{2} \\ 0 \\ \frac{R(\dot{\varphi}_d - \dot{\varphi}_g)}{2L} \end{pmatrix} \quad (1.24)$$

Ce qui donne

$$\begin{pmatrix} \dot{x}_A \\ \dot{y}_A \\ \dot{\theta}_A \end{pmatrix} = \begin{pmatrix} \frac{R}{2}cos\theta & \frac{R}{2}cos\theta \\ \frac{R}{2}sin\theta & \frac{R}{2}sin\theta \\ \frac{R}{2L} & -\frac{R}{2L} \end{pmatrix} * \begin{pmatrix} \dot{\varphi}_d \\ \dot{\varphi}_g \end{pmatrix} \quad (1.25)$$

En combinant les équations (1.21) et (1.25), soit encore à partir de l'équation (1.24), on obtient le modèle cinématique du robot mobile uni cycle :

$$\dot{q}_A = \begin{pmatrix} \dot{x}_A \\ \dot{y}_A \\ \dot{\theta}_A \end{pmatrix} = \begin{pmatrix} cos\theta & 0 \\ sin\theta & 0 \\ 0 & 1 \end{pmatrix} * \begin{pmatrix} v \\ \omega \end{pmatrix} \quad (1.26)$$

I-9 Estimation et apprentissage :

On va traiter diverses méthodes pour gérer le bruit et l'incertitude dans les robots réels et comment mettre en œuvre des algorithmes probabilistes pour tenir compte de cette incertitude. Estimation sa veux dire que nous entendons pour estimer certains aspects de l'état du monde de Données incomplètes et incertaines bruyantes et apprendre les robots utiliser l'expérience préalable pour améliorer leurs performances dans cette incertitude. Alors quelles sont les sources d'incertitude en robotique?

Parmi les sources d'incertitude en robotique les capteurs ils peuvent fournir des informations inexactes aussi il pourrait y avoir un manque de connaissance du monde les choses qui peuvent être cachées de vue ou que les robots ne peuvent pas percevoir. Il pourrait y avoir des changements dynamiques dans le mouvement et dans l'environnement et les choses peuvent se déplacer dans le temps et que les robots ne savent pas exactement où se trouvent les choses à l'instant présent Il y a donc plusieurs façons de traiter cette incertitude la solution pour faire face à cette incertitude la modélisation probabiliste qui utilise des distributions de probabilité pour tenir compte de cette incertitude.

I-10 Distribution gaussienne 1D :

En robotique, la distribution gaussienne [29] est une représentation probabiliste continue largement utilisée et elle fournit un moyen utile pour estimer l'incertitude dans l'enveniment et parmi les avantage de cette distribution d'abord, seuls deux paramètres sont nécessaires pour préciser le gaussien qu'ils sont appelé la moyenne et la variance et ils saisir le sens de la répartition ils sont également faciles à calculer et interpréter automatiquement la seconde la distribution a de belles propriétés par exemple produit de Gaussien forme une autre distribution gaussienne donc on ne rencontre pas d'autres formes de distributions lorsque on effectue des opérations sur un modèle gaussien.

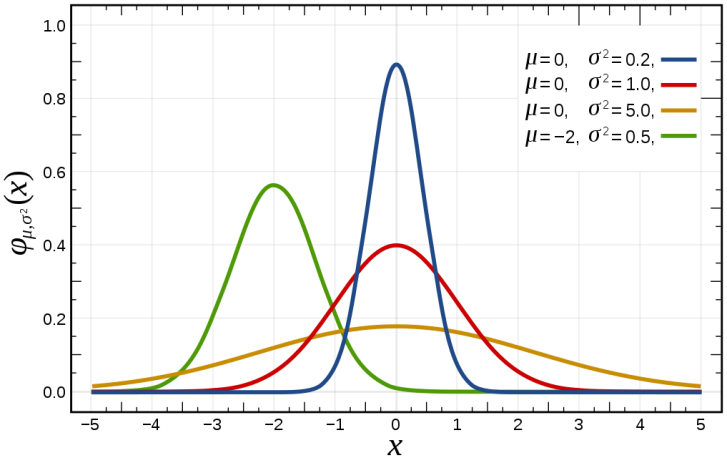

Figure 1.10 Fonction de densité de probabilité.

La distribution gaussienne figure 1.10 est exprimée comme un terme exponentiel multiplié par un scalaire nous voulons connaître la probabilité de x la variable liée à notre distribution gaussienne et nous appelons p(x) fonction de densité de probabilité.

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} exp\left\{\frac{(x - \mu)^2}{2\sigma^2}\right\}$$

x : variable aleatoire
μ : Moyenne
σ^2 : Variance

I.11-Gaussiennes Multi-variables:

Mathématiquement le Gaussien Multi-variables (figure1.11) est exprimé comme une exponentielle couplée avec un facteur scalaire c'est le même que le Gaussien 1D cela pourrait sembler très compliqué mais il a une structure similaire comme la fonction de densité Gaussienne 1D.

$$p(x) = \frac{1}{(2\pi)^{D/2}|\sum|^{1/2}} exp\left\{-\frac{1}{2}(x - \mu)^T \sum^{-1}(x - \mu)\right\}$$

D : la dimension de vecteur d'Etat

X : vecteurs de variables aléatoires

μ : Vecteurs de Moyennes

$\sum$: la matrice de Covariance

$|\sum|$: Déterminant de la matrice de covariance

Prenons un exemple D = 2 et x= [X1, X2] le vecteur de Moyenne $\mu = [0\ 0]^T$

$$\text{Alors } \sum = \begin{bmatrix} \sigma^2_{x1} & \sigma_{x1}\sigma_{x2} \\ \sigma_{x2}\sigma_{x1} & \sigma^2_{x2} \end{bmatrix}$$

Dans notre matrice de covariance il y a deux composantes clés les termes dans la diagonale et les termes hors diagonale l'exemple précèdent de 2D les termes diagonale de la matrice de covariance sont des variances indépendantes de chaque variable X 1 et X 2 les hors diagonale termes représentent les relations entre les deux variables .En générale $\sum$ représente la corrélation entre les deux variables X1 et X2. Pour bien éclaircir cette notion on pose pour cet exemple la matrice

$$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

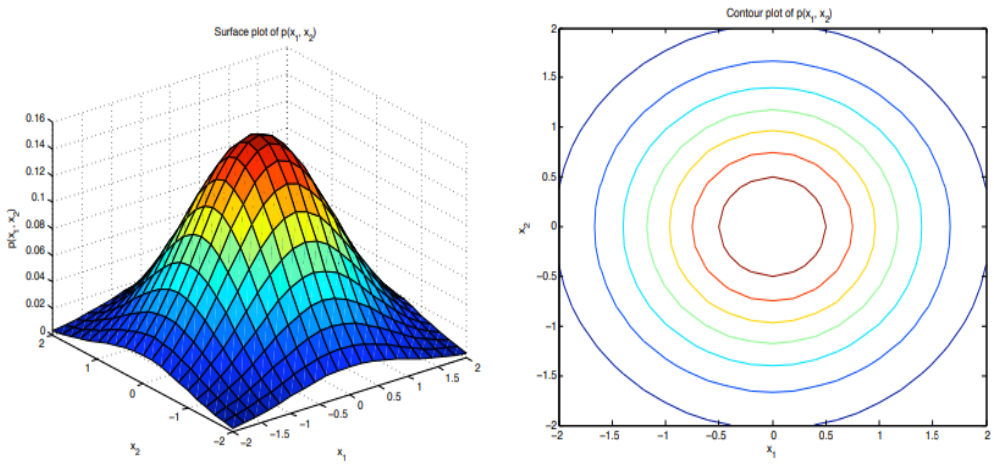

Figure 1.11 Fonction de densité de probabilité 2D.

Parlons brièvement à quoi ressemblent d'autres gaussiens ayant des moyens différents et des matrices de covariance différentes lorsque seule la moyenne est modifiée, la distribution est simplement décalée à

nouveau, comme dans le cas 1D. Si les termes de variance augmentent, la distribution s'étale les contours devient plus large.

Toutefois, la matrice de covariance du gaussien multi-variable a certaines propriétés que nous ne voyons pas dans le gaussien 1D, comme $\sum$ inclut des termes de corrélation dans les éléments hors diagonale si ils sont nul alors la forme de la Gaussienne semble asymétrique et apparemment cela ne peut pas se produire lorsque nous traitons une seul variable, il y a deux autres propriétés que je vais mentionner à propos de Sigma la premier la matrice de covariance doit être Symétrique et positive-définie ce qui signifie que les éléments de la matrice sont symétriques par rapport à sa diagonale les valeurs propres de Sigma doivent être positives. La deuxième propriété même lorsque la matrice de covariance a des termes de corrélation non nuls, nous pouvons toujours trouver une transformation de coordonnées qui fait apparaître la forme symétrique nous pouvons décomposer la matrice de covariance pour révéler la base de la transformation en utilisant des algorithmes de décomposition des valeurs propres.

I-12 Conclusion

Nous avons présenté dans ce chapitre une vue d'ensemble sur la Problématique de la localisation en robotique mobile. L'étape de modélisation qui vient d'être présentée, va maintenant servir à mettre en œuvre les différents algorithmes de localisation et de cartographie présentés dans les chapitres qui suivent. Pour cela, les modèles et les équations de ce chapitre seront réutilisés dans la réalisation des algorithmes.

Chapitre II : Algorithmes de localisation

II-1. Introduction

Pour modéliser la position du robot, nous souhaitons connaître ses coordonnées x et y et son orientation. Ces trois paramètres peuvent être combinés en un vecteur appelé vecteur de variable d'état. Le robot utilise les mesures de distance et d'angle des balises et les informations de locomotion sur la distance parcourue pour calculer sa position. Comme pour tout système réel, ces mesures comprennent une composante d'erreur ou de bruit. Si la trigonométrie est utilisée pour calculer la position du robot, celui-ci peut avoir une erreur importante et peut changer de manière significative d'une image à l'autre en fonction de la mesure du moment. Cela donne l'impression que le robot "saute" dans le champ. Le filtre de KALMAN est une façon plus intelligente d'intégrer les données de mesure dans une estimation en reconnaissant que les mesures sont bruyantes et que parfois elles doivent être ignorées ou n'ont qu'un faible effet sur l'estimation de l'état. Il lisse les effets du bruit dans la variable d'état estimée en incorporant plus d'informations provenant de données fiables que de données non fiables. L'utilisateur peut dire au filtre de Kalman combien de bruit il y a dans le système (y compris les mesures, les entrées et le modèle) et il calcule une estimation de la position en tenant compte du bruit. Il est moins probable qu'il intègre une mauvaise mesure si la confiance dans la position actuelle est élevée. L'algorithme du filtre de KALMAN permet également de combiner très facilement des mesures provenant de différentes sources (telles que des mesures de vision et des données de locomotion) et de différents moments (mises à jour au fur et à mesure de la marche d'un robot). En plus d'une estimation du vecteur de la variable d'état, l'algorithme fournit une estimation de l'incertitude du vecteur de la variable d'état, c'est-à-dire le degré de confiance de l'estimation, compte tenu de la valeur de l'erreur qu'elle contient.

II-2. Modèles de l'espace d'État :

Le vecteur de variable d'état mentionné ci-dessous est habituellement désigné X. Pour calculer ce vecteur à n'importe quel intervalle de temps, nous utilisons la formule :

$$X_{k+1} = AX_k + Bu_k$$

Le vecteur d'état résume les informations pertinentes du système à tout moment et décrit comment le système change en fonction du temps et des entrées. Les indices k et k+1 représentent le temps du vecteur. Dans ce cas, Bu_k est l'entrée que nous recevons en raison de la marche du robot (distance = vitesse x temps) et A est la matrice d'identité. En général, u est le vecteur d'entrée, A est une matrice reliant les variables d'état au pas de temps précédent aux variables d'état au pas de temps actuel et B est une matrice reliant l'entrée aux variables d'état.
Si nous avions une mesure des positions x et y et de l'orientation, nous pourrions écrire ces mesures dans la forme suivant :

$$Y_k = CX_k$$

C'est ce qu'on appelle le modèle de mesure et il décrit comment les données du capteur varient en fonction des variables d'état. Y est le vecteur de la variable de sortie (ou de mesure) et C est une matrice reliant les variables d'état aux variables de mesure.

Pour généraliser, tout système qui est fini dimensionnel, causal (la sortie ne dépend pas des valeurs d'entrée futures) et le temps invariant peut être décrit par un ensemble de variables qui constituent les aspects importants du comportement interne du système. Ces variables peuvent être combinées en un vecteur qui contient à tout moment les informations nécessaires pour caractériser le système. Ce type de

modèle est connu comme un modèle d'espace d'état et est couramment utilisé dans la modélisation du système de contrôle et le traitement du signal.

II-3. Filtre de KALMAN :

Le filtre de KALMAN [30] permet d'estimer l'état d'un système à partir d'une prédiction bruitée de son évolution et de mesures bruitées de cet état. C'est un filtre récursif optimal, qui suppose que le système considéré est linéaire et les bruits blancs (de moyenne nulle). Ce filtre est basé sur la caractérisation de variables aléatoires Gaussiennes par les moments de leurs densités de probabilités : à l'instant t la croyance est représentée par sa moyenne μ et et sa covariance$\sum$.

Le filtre de Kalman peut être utilisé si les deux conditions suivantes sont réunies.

-La densité de probabilité de l'état suivant $p(X_k|u_k, X_{k-1})$doit être une fonction linéaire. Ce qui se traduit par :

$$X_{k+1} = AX_k + Bu_k + \varepsilon_k$$

Avec ε_k Une variable aléatoire Gaussienne (de moyenne nulle et de covariance Rt) qui modélise le bruit sur le changement d'état.

-La densité de probabilité des mesures $p(y_k|X_k)$ doit aussi être une fonction linéaire

$$Y_k = CX_k + w$$

w Une variable aléatoire Gaussienne (de moyenne nulle et de covariance Qt) symbolisant le bruit des mesures.

Le fonctionnement du filtre se déroule en quatre étapes :

– Prédiction de l'état à l'instant courant X_k , ainsi que de sa covariance P_k à partir du modèle d'évolution, de l'estimation au pas de temps précédent et de la commande depuis cet instant :

$$\hat{X}_k = A\hat{X}_{k-1} + Bu_k$$

La covariance est également prédite par la formule :

$$P_k = AP_{k-1}A^T + BQB^T$$

– Prédiction de l'observation à partir du modèle d'observation et de l'estimation de l'état :

$$\hat{y}_k = C\hat{X}_k$$

– Observation de l'état : on obtient, grâce au système perceptif, une mesure Y_k dont on estime le bruit P_k grâce au modèle du processus de perception.

– Correction de l'état prédit par mise à jour proportionnellement à l'erreur entre l'observation prédite et l'observation réalisée :

$$\hat{X}_{k+1} = \hat{X}_k + K(y - \hat{y}_k)$$

$$P_{k+1} = (I - K_k C)P_k$$

Ou K est le gain de Kalman, calculé pour minimiser l'erreur d'estimation

$$K_k = \frac{P_k^{-} C^T}{C P_k^{-} C^T + R}$$

Ces quatre étapes sont utilisées à chaque nouvelle information de déplacement et de perception, afin de mettre à jour l'estimation de l'état du système.

Algorithme Filtre de KALMAN

Entrée $(\mu_{k-1}, P_{k-1}, u_k, Y_k)$

Sortie (μ_{k-1}, P_k)

1: procedure KF (μ_k, P_k, u_k, Y_k)

2: $\mu_{k+1} \leftarrow AX_k + Bu_k$

3: $P_{k+1} \leftarrow P_k = AP_{k-1}A^T + BQB^T$

4: K $\leftarrow \frac{P_k^{-} C^T}{C P_k^{-} C^T + R}$

5: $\mu_{k+1} \leftarrow \mu_k + K(y - \hat{y}_k)$

6: $P_{k+1} \leftarrow (I - K_k C)P_k$

7: return μ_k, P_k

II-4. Filtre de KALMAN étendu (Extended Kalman Filter : EKF)

Cet estimateur d'état est utilisé dans le cas où le modèle du système observable est non linéaire. Ce filtre permet de résoudre le problème de linéarisation des systèmes non linéaires autour d'un point de fonctionnement, en utilisant le développement de Tylor au premier ou au deuxième ordre [31].

$$\begin{cases} \dot{x_k} = f(x_{k-1}, u_k) + \varepsilon_k & (2.1) \\ y_k = h(x_k) + w & (2.2) \end{cases}$$

Le filtre étendu de Kalman prédit les états du processus aléatoire avec l'équation (2.1). Les états prévus sont mis à jour avec les mesures de l'équation (2.2). où $\dot{x_k}$ est le vecteur d'état de processus prévu, x_{k-1} est le vecteur d'état de processus estimé, $f(x_{k-1}, u_k)$ est la matrice de transition d'état discret, ε_k est le vecteur de bruit de processus, y_k est le vecteur de mesure, $h(x_k)$ est la matrice d'observation et w est le vecteur de mesure du bruit.

Ici, le bruit entre linéairement dans le système, mais il peut y avoir des systèmes où le bruit n'est pas additif Dans un système général, soit la fonction de transition d'état, soit la fonction de mesure ou les deux peuvent être non linéaires. Pour tous ces cas, nous devons utiliser un estimateur d'état non linéaire au lieu d'un filtre kalman .Cependant, si le model du système est non linéaire, alors la distribution d'état résultante ne peut pas être gaussienne et donc l'algorithme de filtre kalman peut ne pas converger Dans ce cas, vous pouvez implémenter un filtre kalman étendu (ekf) qui linéarise la fonction non linéaire autour de la moyenne de l'estimation de l'état actuel à chaque étape de temps la linéarisation est effectuée localement et les matrices jacobéennes résultantes sont ensuite utilisées dans les états de prédiction et de mise à jour de l'algorithme de filtre kalman

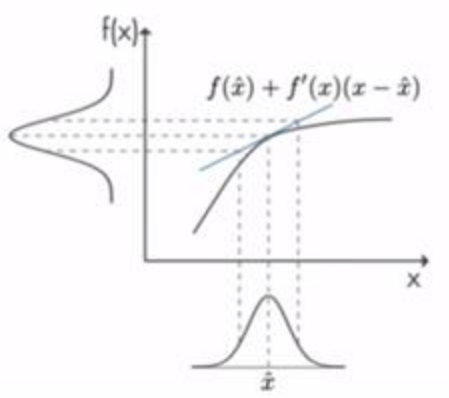

Figure 2.1 la linéarisation de la fonction d'entrée.

Le fonctionnement du filtre EKF se déroule avec les mêmes étapes du filtre de kalman avec l'utilisation des matrices Jacobéennes A et C définies par :

$$\begin{cases} A_{ij} = \dfrac{\partial f_i}{\partial x_j} \\ C_{ij} = \dfrac{\partial h_i}{\partial x_j} \end{cases}$$

Avec ces deux matrices Jacobéennes, le principe du filtre de Kalman reste exactement le même.
L'étape d'initialisation consiste à calculer le vecteur d'état initial à l'instant k = 0 et la matrice de covariance associée à l'état estimé initialementX_0 et P_0.
La phase de prédiction permet de calculer l'état du système et la matrice de covariance a priori :

$$\hat{X}_k = f(x_{k-1}, u_k)$$
$$P_k = AP_{k-1}A^T + BQB^T$$

Le gain de Kalman étendu est exprimé par l'équation suivante :

$$K_k = \frac{P_k^{-} C^T}{C P_k^{-} C^T + R}$$

La phase de mise à jour vise à estimer le vecteur d'état et la matrice de covariance à posteriori :

$$\hat{X}_{k+1} = \hat{X}_k + K(y - \hat{y}_k)$$
$$P_{k+1} = (I - K_k C) P_k$$

Algorithme filtre de KALMAN étendu (EKF)
Entrée $(\mu_{k-1}, P_{k-1}, u_k, Y_k)$
Sortie μ_k, P_k

Propagation

1 :$\mu_k = f(x_{k-1}, u_k)$

2 :$P_k = AP_{k-1}A^T + BQB^T$ ou $A_{ij} = \frac{\partial f_i}{\partial x_j}$

Mise à jour

3:$K_k = \frac{P_k^{-} C^T}{C P_k^{-} C^T + R}$ ou $C_{ij} = \frac{\partial h_i}{\partial x_j}$

4 :$\mu_{k+1} = \mu_k + K(y - \hat{y}_k)$

5:$P_{k+1} = (I - K_k C) P_k$

6: return μ_k, P_k

II-5. Filtre de Kalman non–Parfumé (Uncented Kalman Filter :UKF)

Le filtre de KALMAN non–Parfumé UKF est un estimateur qui a été proposé par Julier et Uhlmann. Il se différencie du filtre de KALMAN étendu par le fait que son algorithme n'exige pas l'étape de linéarisation des fonctions non-linéaires de transition f et de mesure h. Il autorise donc de se passer du problème de calcul fastidieux et délicat dans certains cas, des Jacobiennes du modèle. Son principe repose sur la transformation non parfumée (UT- Unscented Transformation), qui permet de calculer successivement la moyenne ainsi que la covariance à posteriori de l'état, en créant un nombre fini de points appelés « Sigma points », qui dépendent de la taille du vecteur d'état. Au lieu de se rapprocher d'une fonction non linéaire comme le font les filtres Kalman étendus, les filtres Kalman non- Parfumé se rapprochent de la distribution de probabilité.
Un filtre Kalman non–Parfumé sélectionne un ensemble minimal de points d'échantillonnage de sorte que leur moyenne et leur covariance soient les mêmes que cette distribution figure (2.2-A). Il s'agit de points sigma, répartis symétriquement autour de la moyenne. Chaque point sigma est ensuite propagé à travers le

modèle de système non linéaire. La moyenne et la covariance des points non linéairement transformés sont calculées et une distribution gaussienne empirique est calculée, qui sert ensuite à calculer la nouvelle estimation de l'état figure (2.2-B). Notez que dans l'algorithme linéaire du filtre Kalman, la covariance d'erreur P est calculée en utilisant la fonction de transition d'état dans l'étape de prévision, puis elle est mise à jour en utilisant la mesure. Cependant, dans le filtre KALMAN non parfumé, nous ne le calculons pas de la même façon, parce que nous l'obtenons empiriquement à la place. Un autre estimateur d'état non linéaire basé sur un principe très similaire est le filtre à particules (PF). Il utilise également des points d'échantillon appelés particules. La différence significative par rapport à un filtre Kalman non parfumé est qu'un filtre à particules se rapproche de toute distribution arbitraire, il n'est donc pas limité à une hypothèse gaussienne. Et pour représenter une distribution arbitraire qui n'est pas connue explicitement, le nombre de particules dont un filtre à particules a besoin est beaucoup plus grand qu'un filtre Kalman non parfumé.

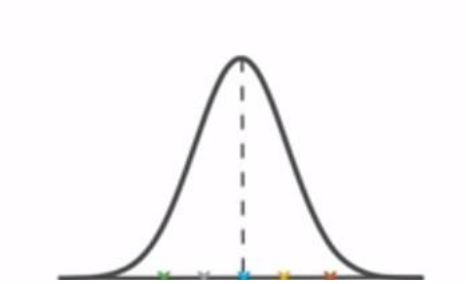

Figure2.2-A les points sigma sont sélectionnés que leur moyenne u et covariance P sont les mêmes que la distribution de probabilité.

Figure 2.2-B La moyenne et la covariance P du point sigma sont utilisées pour calculer la nouvelle estimation de l'état.

Transformation non–Parfumé(UT) :

Soit une variable aléatoire X de taille L, de moyenne $\bar{X}$ et sa matrice de covariance Px. Étant donné que Y = f(X), avec Y est aussi une variable aléatoire dont nous souhaitons déterminer sa moyenne et sa matrice de covariance et f est la fonction non-linéaire, la transformation non linéaire (UT) consiste à sélectionner un échantillon de points de taille finie égale à (2 L + 1), nommés « Sigma-points ». Ces points notés $\{\chi i\}^{2L+1}{}_i$ et pondérés par leurs poids $\{Wi\}^{2L+1}{}_i$, sont propagés par le modèle de prédiction non linéarisé et la moyenne et la covariance sont ensuite calculés par la moyenne pondérée. La phase suivante permet de calculer le vecteur de mesure prédit, la matrice de gain et la moyenne et la covariance à posteriori.

L'algorithme du filtre UKF est comme suit :

Étape 1. Initialisation : consiste à initialiser le vecteur d'état X_0 et la matrice de covariance d'erreurP_0. Leurs valeurs initiales sont données par $\hat{X}_0$ et P(0)

On a Q et R sont respectivement les matrices de covariance des bruits d'état et de mesure.

Étape 2 : Calcul des Sigma points

Cette étape permet de calculer les sigma points χi et leurs poids de pondération Wi correspondants. Les Sigma points sont calculés par les équations suivantes :

χ (0) = $\hat{x}(k-1)$

$\chi i = \hat{x}(k-1) + \sqrt{(L + \lambda) \cdot P_{k-1})_i}$, i = 1,….L

$\chi i = \hat{x}(k-1) - \sqrt{(L + \lambda) \cdot P_{k-1})_{i-L}}$, i = L + 1,….2L

Les poids de pondération des sigma points sont donnés par :

$$W^{(m)}_0 = \lambda/(L + \lambda)$$

$$W^{(c)}_0 = \frac{\lambda}{L + \lambda} + +(1 - \alpha^2 + \beta)$$

$W^{(m)}_i = W^{(c)}_i = \lambda/2(L + \lambda)$, i = L + 1….2L

Avec :

λ, β et α sont les paramètres de configuration, λ est exprimé ainsi : $\lambda = \alpha^2 \cdot (L + K) - L$

α détermine la distribution des sigma points autour de la moyenne. Ce paramètre vaut généralement un petit nombre positif (exemple $1 \cdot 10^{-3}$), β est le paramètre de répartition de l'état, il vaut 2 pour la répartition gaussienne et zéro pour une seule variable d'état, K est le paramètre d'échelonnage secondaire souvent égal à K=0.

$\sqrt{(L + \lambda) \cdot P_{k-1})_i}$ est la ième colonne de la racine carrée de la matrice $((L + \lambda) \cdot P_{k-1})$ par l'utilisation de la décomposition de Cholesky.

Étape 3 : Phase de prédiction

Lors de cette étape, le filtre prédit l'état du système $\hat{x}_{k/k-1}$ et la matrice de covariance d'erreur d'état$P_{k/k-1}$, en propageant les sigmas points $\chi k-1$ par la fonction de transition non-linéaire f. Les équations de la phase de prédiction sont présentées ci-dessous :

-- Propagation des Sigma-points :

$$\chi\,(i, k-1) = f(\chi i, W_i)$$

-- Estimation à priori de l'état du système :

$$\hat{x}_{k/k-1} = \sum_{i=0}^{2L} W^{(m)}_i * \chi(i, k/k-1)$$

- Calcul de la matrice de covariance d'erreur d'état à priori :

$$P_{k/k-1} = \sum_{i=0}^{2L} W^{(C)}_i * \left(\chi_{(i,k)} - \hat{\chi}_{(k|k+1)}\right).\left(\chi_{(i,k)} - \hat{\chi}_{(k|k+1)}\right)^T + Q_{k-1}$$

Étape 4 : Phase de Mise à jour

En vue de mettre à jour l'état du système $\hat{x}k$ ainsi que sa matrice de covariance de bruit Pk, les sigma points χi sont propagés selon la fonction non-linéaire d'observation h et donc de nouveaux sigma points sont obtenus :

$$Y(i,/k-1) = h(\chi\, i,k-1)$$

Le vecteur de mesures est écrit ainsi :

$$\hat{y}_{k/k-1} = \sum_{i=0}^{2L} W^{(m)}{}_i * \mathrm{Y(i,k/k-1)}$$

Le gain de Kalman UKF est donné par :

$$K_k = P_{\hat{x}_k.\hat{y}_k} * {P_{\hat{y}_k.\hat{y}_k}}^{-1}$$

Avec :

$P_{\hat{x}_k.\hat{y}_k}$ Est la matrice de covariance des états et des mesures, exprimée par :

$$P_{\hat{x}_k.\hat{y}_k} = \sum_{i=0}^{2L} W^{(c)}{}_i * \left(\chi_{(\mathrm{i,k})} - \hat{\chi}_{(\mathrm{k}|k+1)}\right).\left(\mathrm{Y}_{(\mathrm{i,k})} - \hat{y}_{(\mathrm{k}|k+1)}\right)^T$$

$P_{\hat{y}_k.\hat{y}_k}$Est la matrice de covariance d'erreur des mesures :

$$P_{\hat{y}_k.\hat{y}_k} = \sum_{i=0}^{2L} W^{(c)}{}_i * \left(\mathrm{Y}_{(\mathrm{i,k})} - \hat{y}_{(\mathrm{k}|k+1)}\right).\left(\mathrm{Y}_{(\mathrm{i,k})} - \hat{y}_{(\mathrm{k}|k+1)}\right)^T$$

La mise à jour des états du système est donnée par :

$$\hat{x}_{k/k} = \hat{x}_{k|k-1} + K_k(\mathrm{Y}_k - \hat{y}_{(\mathrm{k}|k+1)})$$

Et la mise à jour de la matrice de covariance des bruits de mesures est exprimée ainsi :

$$P_{k/k} = P_{k|k-1} + K_k P_{\hat{y}_k.\hat{y}_k} {K_k}^T$$

Tableaux : comparaison entre les algorithmes de localisation.

Estimateur d'état	**Modèle**	**Distribution Assumé**	**Coût de calcul**
Filtre de Kalman (KF)	Linéaire	gaussien	faible
Filtre de Kalman étendu (EKF)	Linéaire localement	gaussien	faible si les jacobiens doivent être calculés analytiquement moyen si les jacobiens doivent être calculés numériquement
Filtre de KALMAN non–Parfumé (UKF)	Non linéaire	gaussien	moyen

Les tableaux ci-dessus nous montre les diffèrent avantage et inconvénient des algorithmes de localisation. Le filtre de Kalman est optimal lorsqu'il est utilisé pour l'étude de systèmes linéaires

perturbés par des bruits gaussiens. Il est dit optimal dans le sens de la minimisation de l'erreur statistique. La palette de cas peut être élargie en utilisant filtre de kalman étendu (EKF) ou filtre de kalman non– Parfumé (UKF). Par exemple, dans le contexte de l'EKF, les équations d'état non linéaires sont transformées à l'aide d'un développement limité.

II-6. Conclusion

Comme nous l'avons vu précédemment, le bon fonctionnement du filtre de Kalman repose sur le respect d'un certain nombre de propriétés mathématiques qui ne sont pas toujours vérifiable sur un système embarqué en pratique. Nous avons déjà abordé deux méthodes (EKF et UKF) permettant de doubler la précision du filtre de Kalman, mais certains problèmes peuvent tout de même subsister. Par exemple, nous pouvons avoir une erreur d'estimation des matrices de covariance des bruits de mesure et du système dynamique, ou encore une erreur dans la modélisation du système lui-même, ce qui peut provoquer un comportement non désiré du filtre.

Chapitre III : La localisation par les amers

III-1. Introduction

Les concepts modernes de la robotique définissent un robot comme une unité électromécanique qui est dotée d'instructions et d'algorithmes qui lui permettent d'exécuter plus facilement les tâches assignées par les humains.

Cependant, l'étude des robots mobiles est embrouillée par de nombreux problèmes. Pour améliorer son efficacité et son intelligence, les entreprises technologiques doivent investir des milliards de dollars, au risque d'endommager de vrais robots. Dans ces conditions, la solution c'est logiciel de simulation robotique ou des Plateformes virtuelles.

Dans ce chapitre, nous introduirons tout d'abord la Plateforme virtuelle V-REP, et la communication entre MATLAB et V-REP. Ensuite, nous illustrerons l'algorithme EKF-SLAM.

III-2. La Virtual Platform d'expérimentation V-REP

La Virtual Platform d'expérimentation Robot (V-REP) est un cadre polyvalent et évolutif pour développer des simulations 3D dans un laps de temps relativement court. Ce simulateur a été créé en 2010 et s'est développé au cours des dernières années. Aujourd'hui, c'est l'un des simulateurs les plus utilisés pour l'éducation dans le domaine de la robotique (avec une licence gratuite à des fins éducatives).

V-REP dispose d'un environnement de développement intégré (IDE) basé sur une architecture de contrôle répartie : chaque objet/modèle peut être contrôlé individuellement via un script embarqué, un plug-in, un nœud Robot Operating System (ROS), un client API distant ou une solution personnalisée. Les contrôleurs peuvent être écrits en différentes langues : C/C++, Python, Java, Lua, MATLAB, Octave ou Urbi (Coppelia Robotics Gmbh). V-REP est livré avec un grand nombre d'exemples, modèles de robots, capteurs et actionneurs pour créer un monde virtuel et d'interagir avec elle dans le temps de fonctionnement.

V-REP offre divers moyens pour contrôler les simulations ou même pour personnaliser le simulateur lui-même Figure (4.3). V-REP est enveloppé dans une bibliothèque de fonctions et nécessite l'exécution d'une application client. L'application client par défaut V-REP est assez simple et prend soin de charger les modules d'extension, enregistre les callbacks d'événements (ou callbacks de messages), les relaye aux modules d'extension chargés, initialise le simulateur et gère la boucle d'application et de simulation. En outre, des fonctions de simulation personnalisées peuvent être ajoutées via :

1. Scripts dans le L'environnement MATLAB/Simulink

L'environnement MATLAB/Simulink est une possibilité très pratique pour développer, déboguer et tester des algorithmes de contrôle complexes de façon « rapide » de prototypage. On peut coder des algorithmes soit dans le langage de script MATLAB, soit en incluant du code C/C++ externe en utilisant, par exemple, des fonctions s. De plus, la fonction de génération automatique de code permet d'accélérer le temps d'exécution et même de déployer les binaires générés vers d'autres plateformes (par exemple, le robot lui-même). Enfin, l'utilisation des scopes, des affichages et des autres outils de visualisation et de post-traitement MATLAB représente une grande valeur ajoutée pendant la phase de débogage et de test des algorithmes.

2. Modules d'extension vers V-REP (plugins)

Les modules d'extension permettent d'enregistrer et de manipuler des commandes personnalisées. Une commande de script de haut niveau (par exemple robotMoveAndAvoidObstacles(duration)) peut alors être un module d'extension peut gérer cette commande de haut niveau en exécutant les appels de logique et de fonction API de bas niveau correspondants de manière rapide et cachée.

D'autre part, au meilleur de nos connaissances, MATLAB manque d'un moteur de simulation physique 3D facile à utiliser spécifiquement adapter aux applications robotiques. A cet égard, VREP représente une alternative valable : il se compose d'un moteur de simulation physique 3D de pointe (et disponible gratuitement pour l'usage académique) qui devient de plus en plus répandu dans la communauté robotique grâce à sa flexibilité (possibilité de simuler de nombreuses plateformes robotiques différentes), moteur dynamique, et enfin la personnalisation (il offre de nombreuses possibilités différentes d'inclure son propre code ou de l'interfacer avec le monde externe). Concernant ce dernier point, V-REP est livré avec un support natif de ROS qui facilite l'échange de données avec d'autres modules, par exemple pour envoyer l'état du robot et les données du capteur à un planificateur contrôleur externe et recevoir les commandes de mouvement.

III-3. La communication entre MATLAB et V-REP

V-REP peut être programmé/contrôlé par de nombreux moyens différents. L'un d'eux sont les scripts enfants (Child scripts) précédemment mentionnés. Ceux-ci sont utilisés dans la scène V-REP pour gérer tout ce qui est local à V-REP (à l'intérieur de V-REP). V-REP offre également plusieurs façons de contrôler une simulation à partir d'une application externe (c.-à-d. une application cliente), dont l'API à distance. Les principales caractéristiques de l'API distante sont :

- ✓ Multiplateforme
- ✓ Léger (également intégrable sur un microcontrôleur)
- ✓ Prise en charge directe de plusieurs langues (C/C++, Java, Python, MATLAB, Octave ou Urbi)
- ✓ Prise en charge des requêtes (c.-à-d. appels de fonction bloquants) ou streaming (c.-à-d. appels de fonction non bloquants)

Les scripts MATLAB ont la capacité de se connecter à V-REP via l'API à distance, et de contrôler la simulation. Par défaut, ils se connecteront au port local 19997. Du côté de V-REP, un service de serveur est normalement lancé automatiquement sur ce port au démarrage de V-REP. Ce comportement et certains autres paramètres peuvent être modifiés dans le dossier de V-REP, en modifiant le fichier remoteApi. Vous pouvez également modifier ce fichier afin de déboguer la connexion.

Pour les tests de configuration simulateur-dans-la-boucle V-REP offre également une méthode pour contrôler la simulation de l'extérieur du simulateur par l'algorithme externe de contrôleur implicite. Le contrôleur développé dans l'interface API à distance dans V-REP communique avec la scène de simulation en utilisant une communication socket. Il est composé de services de serveur API distant et de clients API distants.

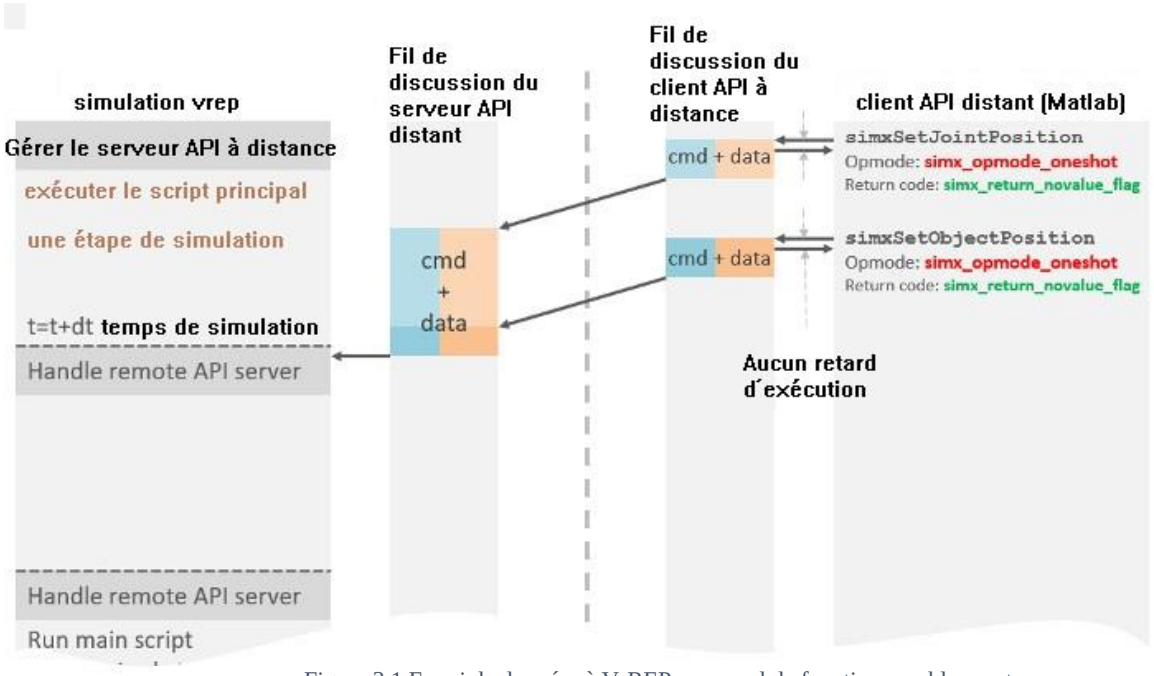

Figure 3.1 Envoi de données à V-REP par appel de fonction non bloquant

Le côté client peut être développé en langues C/C++, Python, Java, MATLAB ou Urbi, également intégré dans tout logiciel fonctionnant sur du matériel de contrôle à distance ou de vrais robots, et il permet l'appel de fonction à distance, ainsi que la diffusion rapide des données. Les fonctions prennent en charge deux méthodes d'appel pour s'adapter à n'importe quelle configuration : bloquer, attendre que le serveur réponde, ou non verrouiller, lire des commandes à partir d'un fichier. Le schéma de ce mode de communication est illustré à la figure 3.1 et la figure 3.2.

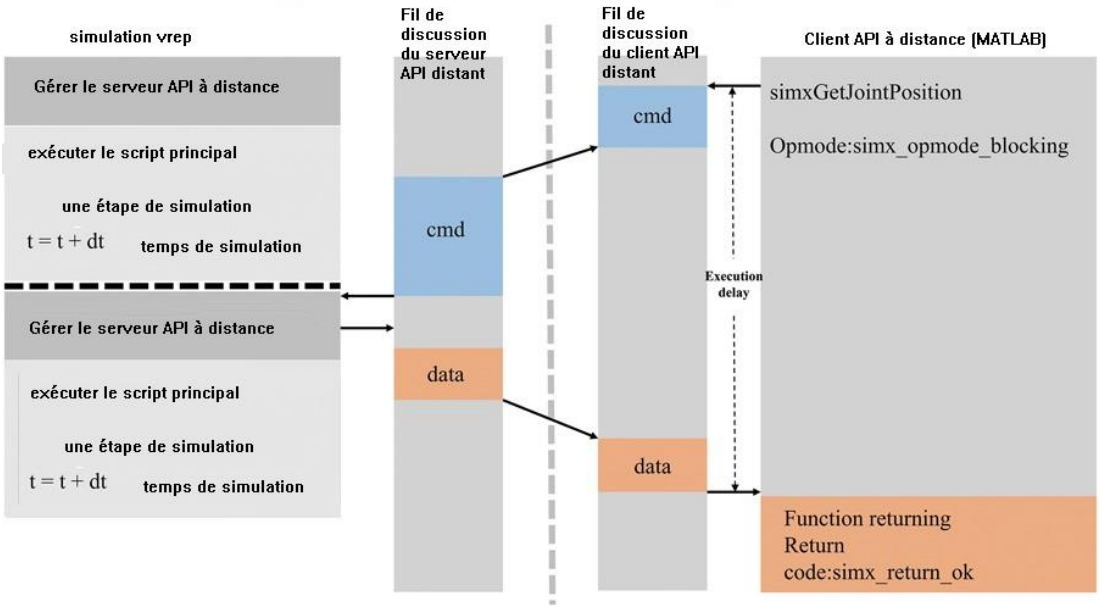

Figure 3.2 lecture des données de V-REP via appel de fonction bloquante.

Les

plugins implémentent le serveur API à l'intérieur de V-REP pour fournir un processus de simulation avec des commandes LUA standard. Ils sont donc généralement utilisés en combinaison avec des scripts. Par contre, s'il est nécessaire de faire des calculs rapides (les 35 langues compilées sont la plupart du temps plus rapides que les scripts) ou une interface vers un véritable appareil (par ex. un vrai robot), les plugins offrent des fonctionnalités spéciales. Le code qui configure un client API distant, nécessaire pour effectuer la communication MATLAB - V-REP, se trouve dans le dossier V-REP.

III-4. Le Framework de contrôle

Maintenant on peut contrôler Notre robot a partir l'environnement de MATLAB à l'aide de 'Remote API' fonctions MATLAB qui nous fournis des fonctions pour faciliter le contrôle mais le problème pour juste déplacer le robot il faut passe par quatre fonctions .parmi les solutions qui rendre le contrôle pratique et facile le Framework bob_robot qui contient des méthodes ou des fonctions de l'objet bob(robot) qui nous facilite la tâche. Notez les principales fonctions suivantes :

simulation_setup : ajoute les dossiers appropriés au chemin de recherche, afin que Matlab puisse localiser les bibliothèques d'API distantes spécifiques à votre plateforme/configuration (situées dans le dossier libs). La fonction chargera également la bibliothèque, si elle n'a pas déjà été chargée.

- ✓ simulation_openConnection : ouvre une ligne de communication vers V-REP. Assurez-vous que V-REP est lancé et que la scène correcte est chargée .
- ✓ bob_init : récupère les données statiques liées à Bob de la scène, et commence à transmettre les données de V-REP au client API distant (Matlab).
- ✓ simulation_start : demande à V-REP de démarrer la simulation.
- ✓ simulation_stop : demande à V-REP d'arrêter la simulation.
- ✓ simulation_closeConnection : ferme une ligne de communication vers V-REP précédemment ouverte via simulation_openConnection.

Le code de base qui nous permet dmImplémente certaines fonctionnalités de base de V-REP et le suivant.

```
connection = simulation_setup();
robotNb = 0;
connection = simulation_openConnection(connection, robotNb);
simulation_start(connection);
[bodyDiameter wheelDiameter interWheelDist scannerPose] = bob_init(connection);
[bodyDiameter] = bob_getBodyDiameter(connection);
[wheelDiameter] = bob_getWheelDiameter(connection);
[interWheelDist] = bob_getInterWheelDistance(connection);
[scannerPose] = bob_getScannerPose(connection);
% ------------------------------

for i=0:100

% -Notre Code

    simulation_triggerStep(connection);
    pause(0.1);
end
% ------------------------------
simulation_stop(connection);
```

```
simulation_closeConnection(connection);
```

La boucle for permet d'initialiser le robot et faire la communication à V-REP et le bloc après la boucle fermer la communication.les autre fonction lier au robot sont :

bob_setWheelSpeeds
Description : régler la vitesse du robot en deg/sec.
Synopsis MATLAB: `bob_setWheelSpeeds( clientID, motor droite, motor gauche);`

bob_getWheelSpeeds
Description : obtenir la Vitesse du robot en deg/sec.
Synopsis MATLAB: [`motor droite, motor gauche]=bob_getWheelSpeeds(clientID);`

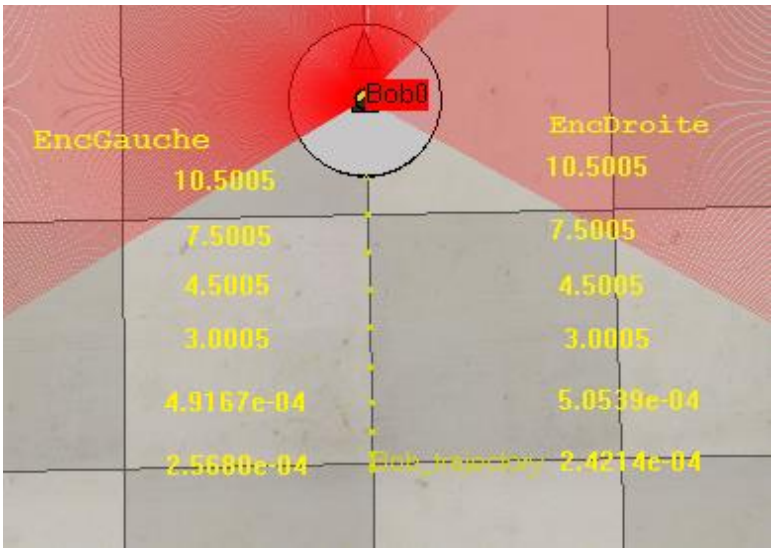

Figure 3.3 la mesure de la vitesse des roues avec la fonction bob_getEncoders

bob_getEncoders
Description : obtenir les encodeurs du moteur en degré figure 3.3.
Synopsis Matlab : `[EncGauche EncDroite] = bob_getEncoders(clientID);`
Exemple : on initialise la vitesse du robot par 10 rad/s dans 5 second on trouve les résultats de la fonction bob_getEncoders

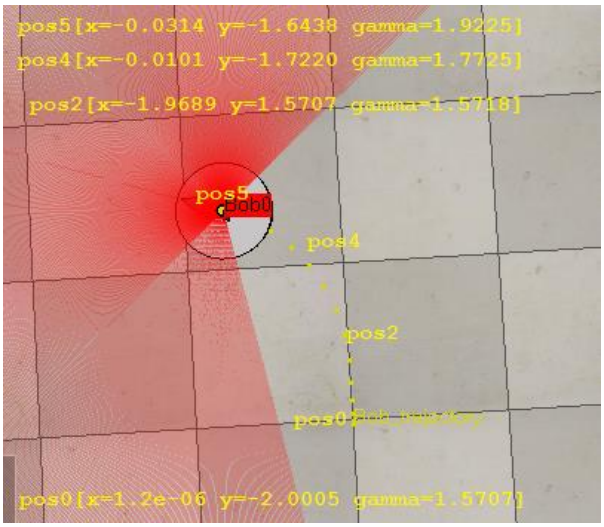

Figure 3.4 la mesure de la position et l'orientation du robot avec la fonction bob_getPos

bob_getPose

Description : obtenir la pose du robot (x,y,gamma), où les positions sont en mètres, orientation (gamma) en degrés figure 3.4.

Synopsis Matlab : `[x y gamma] = bob_getTargetGhostPose(clientID)`

Exemple : on initialise la vitesse du robot cette fois bob_setWheelSpeeds(connection,25,40) dans 5 second on trouve les résultats de la fonction bob_getPose.

bob_getLaserData

Description : obtenir des données laser coords X/Y. sont en mètres, et par rapport au cadre de référence du scanner laser figure 3.5.

Synopsis Matlab : `[laserDataX laserDataY] = bob_getLaserData(clientID)`

Exemple : on initialise la vitesse du robot cette fois bob_setWheelSpeeds(connection,0, 0) dans 5 second on trouve les resultas de la fonction bob_getLaserData et a laide de la commande scatter(laserDataX,laserDataY) en obtient les donnes de l'environnement capter par le laser.

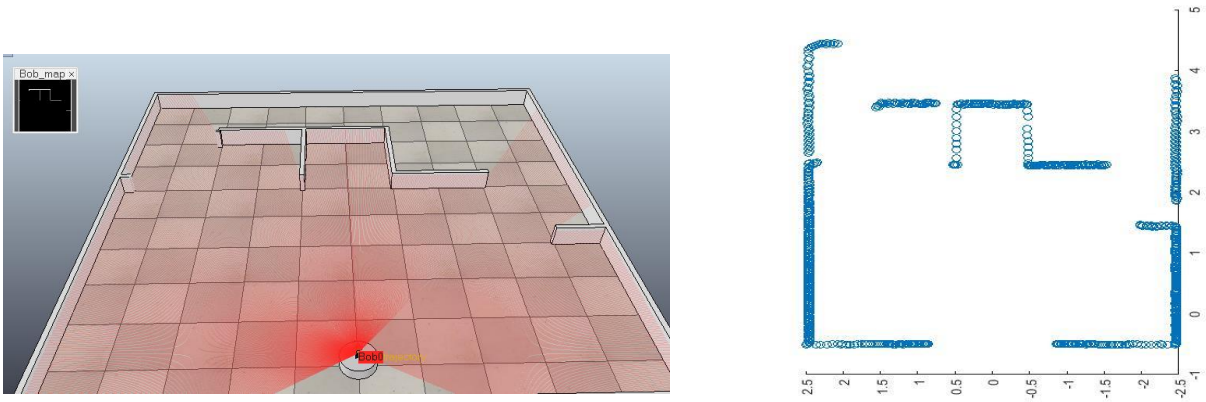

Figure 3.5 la map de l'environnement donne par la fonction bob_getLaser.

bob_addPathSegment
Description: Ajouter un segment dans la carte avec les cordonné(x0,y0,x1,y1).
Synopsis MATLAB : `bob_addPathSegment(clientID, x0,y0,x1,y1).`

bob_getMap
Description : Obtenir la carte globale.
Synopsis Matlab : `bob_getMap(connection)`

III.5. Description de l'algorithme filtre de KALMAN avec les balises

Le SLAM stochastique est réalisé en stockant la position du robot et les repères cartographiques (les balises) dans un vecteur à état unique, et en estimant les paramètres d'état via un processus récursif de prévision et d'observation. L'étape de la prédiction porte sur le mouvement du robot en fonction des estimations progressives des positions, et augmente l'incertitude de l'estimation de la position du robot. L'étape d'observation, ou de mise à jour, intervient après la coopération des mesures et la ré-observation des anciennes mesures, ce qui améliore l'estimation globale de l'état. Cependant, lorsqu'une caractéristique est observée pour la première fois, elle est ajoutée au vecteur d'état par un processus d'initialisation appelé augmentation de l'état

Lorsque un robot mobile se déplaçant dans un environnement et effectuant des observations relatives à un certain nombre de points de repère inconnus à l'aide d'un capteur situé sur le robot (IMU,GPS…), comme le montre la figure 3.6.

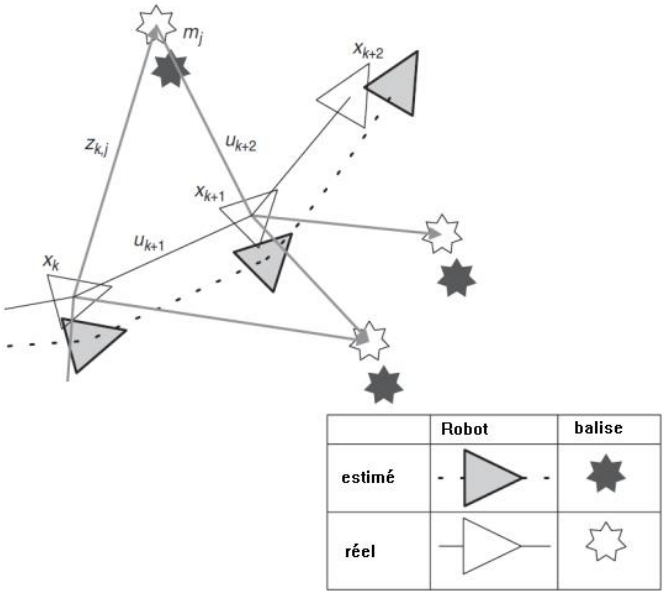

Figure 3.6 Le problème essentiel du SLAM, Une estimation simultanée du robot et du Il est nécessaire de disposer de points de repère. Les emplacements réels ne sont jamais connus ou mesurés directement.

L'état du robot est représenté, dans ce projet, par sa pose par rapport à un cadre de coordonnées cartésiennes de base .la moyenne et la covariance étant définies comme.

$$\hat{X}_r = [\hat{X}_r, \hat{Y}_r, \hat{\theta}_r]^T \tag{3.1}$$

$$P = \begin{pmatrix} \sigma^2{}_{xx} & \sigma^2{}_{xy} & \sigma^2{}_{x\theta} \\ \sigma^2{}_{xy} & \sigma^2{}_{xy} & \sigma^2{}_{y\theta} \\ \sigma^2{}_{x\theta} & \sigma^2{}_{y\theta} & \sigma^2{}_{\theta\theta} \end{pmatrix} \tag{3.2}$$

Les balises 2D observés par le robot forment une carte $\hat{\mathrm{X}}_{\mathrm{m}}$ dans le même système de coordonnées de base. La matrice de covariance P de cette carte comprend des informations de corrélation croisée entre les éléments (c'est-à-dire les termes hors diagonale) qui rendent compte de la dépendance de l'emplacement de chaque élément par rapport à la connaissance des autres éléments de la carte. Comme ces emplacements sont statiques, ces corrélations augmenteront à chaque nouvelle observation à mesure que la carte deviendra plus rigide.

$$\hat{X}_r = [\hat{X}_1 \hat{Y}_1 \dots \hat{X}_n \hat{Y}_n]^T \tag{3.3}$$

$$P = \begin{pmatrix} \sigma^2{}_{x_1x_1} \sigma^2{}_{x_1y_1} \dots \sigma^2{}_{x_1x_n} \sigma^2{}_{x_1y_n} \\ \sigma^2{}_{x_1y_1} \sigma^2{}_{y_1y_1} \dots \sigma^2{}_{x_1y_n} \sigma^2{}_{y_ny_n} \\ \vdots \quad \vdots \quad \ddots \quad \vdots \quad \vdots \\ \sigma^2{}_{x_1x_n} \sigma^2{}_{x_1y_n} \cdots \sigma^2{}_{x_nx_n} \sigma^2{}_{x_ny_n} \\ \sigma^2{}_{x_1y_n} \sigma^2{}_{y_ny_n} \cdots \sigma^2{}_{x_ny_n} \sigma^2{}_{y_ny_n} \end{pmatrix} \tag{3.4}$$

La carte SLAM est définie par un vecteur d'état augmenté formé par la concaténation de l'état du robot et de l'état de la carte des caractéristiques. Ceci est nécessaire car la cohérence du SLAM repose sur le maintien de corrélations P_{rm} entre le robot et la carte.

$$\hat{X}_a = \begin{bmatrix} \hat{X}_r \\ \hat{X}_m \end{bmatrix} \tag{3.5}$$

$$\mathrm{P}_a = \begin{bmatrix} \mathrm{P}_r & \mathrm{P}_{rm} \\ \mathrm{P}_{rm}{}^T & \mathrm{P}_m \end{bmatrix} \tag{3.6}$$

Notez que lorsque nous initialisons le vecteur d'état $\hat{\mathrm{X}}_{\mathrm{a}}$ nous fixons le lieu $\hat{\mathrm{X}}_{\mathrm{r}} = \hat{\mathrm{X}}_{\mathrm{m}} = 0$ et la covariance $\mathrm{P}_{\mathrm{r}} = \mathrm{P}_{\mathrm{m}} = 0$ cela signifie que je suis absolument sûr d'être là (c'est-à-dire que je pars de [0,0,0] et que je suis à 100%.bien sûr).

III-5.1-Étape de prédiction

Le modèle du processus SLAM précise que le robot se déplace par rapport à sa pose précédente selon une estimation du mouvement à l'estime et que les éléments de la carte restent stationnaires. Dead reckoning est le processus qui consiste à estimer la valeur actuelle d'une quantité variable en utilisant une valeur antérieure et en ajoutant les changements survenus entre-temps. La valeur antérieure et les changements peuvent être des quantités mesurées ou calculées (par exemple, un codeur à roue en quadrature).

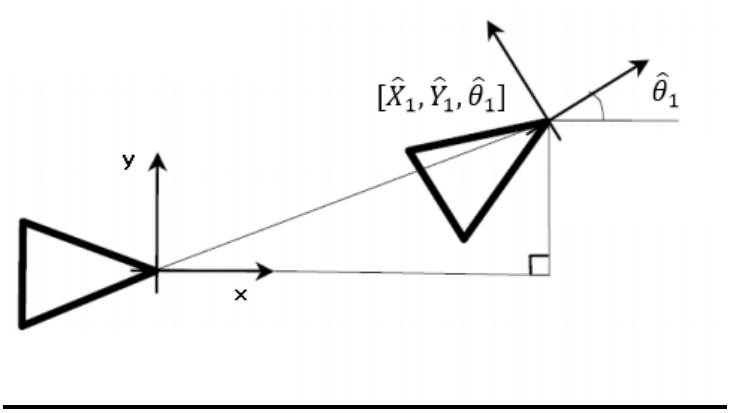

Figure 3.7 Modification du vecteur de pose du robot.

L'effet de ce modèle sur l'estimation de l'état est un changement dans la portion $\hat{X}_r$ du vecteur d'état, et dans les termes P_r et P_{rm} de la matrice de covariance d'état, alors que les portions $\hat{X}_m$ et P_m restent constantes.

Une estimation du changement de position du véhicule $\hat{X}_1 = [\hat{X}_1, \hat{Y}_1, \hat{\theta}_1]^T$ avec la covariance P_{r1} (figure 3.7) est généralement obtenue en utilisant l'odométrie du codeur de roue et un modèle cinématique du véhicule (figure 3.8).

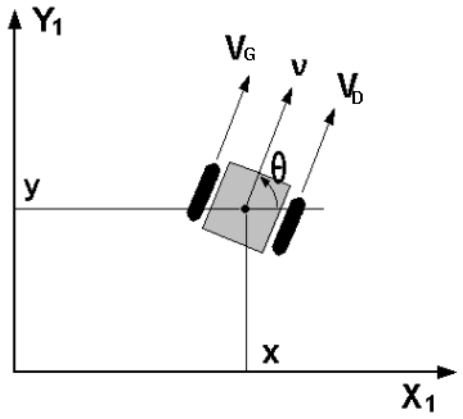

Figure 3.8 Modèle cinématique d'entraînement différentiel.

Le modèle cinématique d'entraînement différentiel est le suivant

$$\hat{X}_r = \begin{bmatrix} x_{k+1} \\ x_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + \frac{1}{2} dt\ \omega(v_{gk} + v_{dk})\cos(\theta_k) \\ y_k + \frac{1}{2} dt\ \omega(v_{gk} + v_{dk})\sin(\theta_k) \\ \theta_k + \frac{1}{D}(v_{gk} - v_{dk}) \end{bmatrix} \tag{3.7}$$

L'état et la covariance prévus sont donc donnés par

$$\hat{X}_a^{-} = f(\hat{X}_a, \hat{X}_c) = \begin{bmatrix} g(\hat{X}_r, \hat{X}_c) \\ \hat{X}_m \end{bmatrix} = \begin{bmatrix} x_k + \frac{1}{2} dt\ \omega(v_{gk} + v_{dk})\cos(\theta_k) \\ y_k + \frac{1}{2} dt\ \omega(v_{gk} + v_{dk})\sin(\theta_k) \\ \theta_k + \frac{1}{2}(v_{gk} - v_{dk}) \\ \hat{X}_m \end{bmatrix} \tag{3.8}$$

$$P_a^{-} = F_{xa} P_a {F_{xa}}^T + F_{ca} Q_c {F_{ca}}^T \tag{3.9}$$

Où les Jacobiennes F_{xa} et F_{ca} sont définis comme

$$F_{xa} = \frac{\partial f}{\partial x_a}\Big|_{(\hat{X}_a, \hat{X}_c)} = \begin{bmatrix} G_{x_r} & 0_r \\ {0^T}_r & I_m \end{bmatrix} \tag{3.10}$$

$$F_{ca} = \frac{\partial f}{\partial u_c}\Big|_{(\hat{X}_a, \hat{X}_c)} = \begin{bmatrix} G_{x_c} \\ {0^T}_r \end{bmatrix} \tag{3.11}$$

Les Jacobiennes G_{xa} et G_{ca} sont définis comme

$$G_{xa} = \frac{\partial g}{\partial x_a}\Big|_{(\mathrm{x\ y\ \theta})} = \begin{bmatrix} 1 & 0 & -\frac{1}{2}dt\ \omega(\mathrm{v}_{gk} + \mathrm{v}_{dk})\sin(\theta_k) \\ 0 & 1 & -\frac{1}{2}dt\ \omega(\mathrm{v}_{gk} + \mathrm{v}_{dk})\cos(\theta_k) \\ 0 & 0 & 1 \end{bmatrix} \tag{3.12}$$

$$G_{ca} = \frac{\partial g}{\partial u_c}\Big|_{(\mathrm{Vg\ Vd})} = \begin{bmatrix} \frac{1}{2}dt\ cos(\theta_k) & \frac{1}{2}dt\cos(\theta_k) \\ \frac{1}{2}dt\sin(\theta_k) & \frac{1}{2}dt\sin(\theta_k) \\ -\frac{dt}{\omega} & \frac{dt}{\omega} \end{bmatrix} \tag{3.13}$$

Comme ces Jacobines n'affectent que la partie véhicule de la matrice de covariance Pr et ses corrélations croisées Prm , l'équation 3.9 peut être mise en œuvre plus efficacement

$$P_a^{-} = \begin{bmatrix} G_{xr}P_r{G_{xr}}^T + G_{xc}P_c{G_{xc}}^T & G_{xr}P_{rm} \\ (G_{xr}G_{xr}P_{rm})^T & P_m \end{bmatrix} \tag{3.14}$$

III-5.2 Étape de mise à jour

La phase de mise à jour vient après les mesures des capteurs en coopération ; ces mesures extraient quelques caractéristiques intéressantes de l'environnement du robot (nous avons supposé un environnement statique).

Si une caractéristique déjà enregistrée sur la carte (précédemment observée), sa localisation $(\hat{\mathrm{x}}_\mathrm{i}, \hat{\mathrm{y}}_\mathrm{i})$ est mise à jour sur la carte actuelle, et si la caractéristique est observée pour la première fois, elle est observée par une représentation en distance de sa localisation comme suit

$$z = \begin{bmatrix} r \\ \theta \end{bmatrix} \tag{3.15}$$

$$R = \begin{pmatrix} {\sigma^2}_r & {\sigma^2}_{r\theta} \\ {\sigma^2}_{r\theta} & {\sigma^2}_{\theta} \end{pmatrix} \tag{3.16}$$

Où r est la distance (c'est-à-dire la distance entre l'observateur et l'amer) et θ est le relèvement (c'est-à-dire l'angle de l'amer) par rapport à l'observateur (c'est-à-dire le robot), et R est la covariance d'observation, les informations détectées sont alors reliées à la carte par l'équation suivante

$$\hat{z}_i = h_i(\hat{x}_a) = \begin{bmatrix} \sqrt{(\hat{x}_i - \hat{x}_r)^2 + (\hat{y}_i - \hat{y}_r)^2} \\ \arctan\left(\frac{\hat{y}_i - \hat{y}_r}{\hat{x}_i - \hat{x}_r}\right) - \widehat{\theta_r} \end{bmatrix} \tag{3.17}$$

En supposant que les données de l'observation z sont correctement associées (voir plus loin) à l'estimation des caractéristiques de la carte ($\hat{x}_i$, $\hat{y}_i$), le gain de KALMAN peut être déterminé comme suit :

$$v_i = z - h_i(\hat{x}^-_a) \tag{3.18}$$

$$K_i = \frac{P_a^- H_{xa}^T}{H_{xa} P_a^- H_{xa}^T + R} \tag{3.19}$$

Où le Jacobien, H_{xa} est donné par

$$H_{xa} = \frac{\partial h_i}{\partial x_a}\Big|_{(\hat{x}_a^-)} = \begin{bmatrix} -\frac{\Delta x}{d} & -\frac{\Delta y}{d} & 0 & 0 & \cdots & 0 & \frac{\Delta x}{d} & \frac{\Delta y}{d} & 0 & \cdots & 0 \\ \frac{\Delta y}{d^2} & -\frac{\Delta x}{d^2} & -1 & 0 & \cdots & 0 & \frac{\Delta y}{d^2} & \frac{\Delta x}{d^2} & 0 & \cdots & 0 \end{bmatrix} \tag{3.20}$$

$$\Delta x = \widehat{x_l} - \widehat{x_r}$$

$$\Delta y = \widehat{y_l} - \widehat{y_r}$$

$$d = \sqrt{(\hat{x}_i - \hat{x}_r)^2 + (\hat{y}_i - \hat{y}_r)^2}$$

Pour les cartes SLAM comportant un grand nombre de caractéristiques, le jacobien h est principalement constitué de termes zéros permettant de calculer efficacement les équations 3.18 et 3.19. Les termes non nuls s'alignent sur les positions des états des véhicules et les états des caractéristiques observées ($\hat{x}_i$, $\hat{y}_i$) dans le vecteur d'état augmenté. L'estimation a posteriori du SLAM est ensuite déterminée à partir des équations de mise à jour.

$$\hat{x}_a^+ = \hat{x}_a^- + K_i \, v_i$$

$$P_a^+ = P_a^- - K_i \, S_i \, K_i^T$$

$$S_i = H_{xa} P_a^- H_{xa}^T + R$$

Le modèle d'observation de l'équation 3.15 met en corrélation l'estimation des caractéristiques avec l'estimation de la pose du véhicule et permet de réduire l'incertitude des deux. Grâce à la corrélation avec l'estimation de la pose du véhicule, les caractéristiques de la carte sont corrélées entre elles et ces corrélations augmentent de façon monotone jusqu'à ce que leur emplacement (l'une par rapport à l'autre) soit parfaitement connu. Sur le plan pratique, si plusieurs observations indépendantes sont disponibles à la fois (une observation z = [r1 θ1 …… rn θn]), alors une mise à jour plus précise peut être possible si elles sont traitées par l'entreprise que si chaque observation est traitée individuellement. La raison de l'amélioration des performances est que l'EKF effectue une correction d'erreur linéarisée et la mise à jour tend à être mieux "tirée dans la bonne direction" si le vecteur d'innovation v est constitué de plusieurs erreurs d'observation à la fois. La différence de performance est plus marquée si la pose du véhicule est très incertaine avant la phase de mise à jour. Toutefois, un inconvénient du traitement de la mise à jour par lots est la nécessité d'inverser la matrice de covariance de l'innovation S qui nécessite un calcul de l'ordre $O(n^3)$ où n est le nombre de caractéristiques dans le lot. Un compromis simple consiste à traiter les observations en sous-lots gérables.

III.6. Simulation :

Dans EKF-SLAM, la carte est un grand vecteur empilant les capteurs et les états des points de repère (des amers), et elle est modélisée par une variable gaussienne. Cette carte, généralement appelée carte stochastique, est maintenue par l'EKF grâce aux processus de prédiction et de correction. Afin de réaliser une véritable exploration, l'EKF est enrichie par une étape supplémentaire d'initialisation des points de repère (des amers), où les points de repère nouvellement découverts sont ajoutés à la carte.

Les matrices d'état et de covariance

Comme nous ne savons pas où se trouvent les points de repère (les amers), nous devons estimer simultanément leur emplacement ainsi que la pose de notre robot. Cela signifie que les emplacements des points de repère font maintenant partie de l'état du système que nous essayons d'estimer. Les coordonnées des points de repère sont donc maintenant incluses dans le vecteur d'état. Lors de la localisation simple, le vecteur d'état avait une dimension de 3x1. Maintenant, en faisant SLAM le vecteur d'état à la dimension 2n+3x1 où n est le nombre de points de repère dans notre problème. Dans cette formulation, les coordonnées du point de repère $\boldsymbol{j}$ ($\boldsymbol{x_j}$ $\boldsymbol{et}$ $\boldsymbol{y_j}$) se trouvent aux indices $\boldsymbol{2i+2}$ et 2i+3 du vecteur d'état.

Rappelez-vous, notre estimation de l'état du système est quantifiée par le vecteur de la moyenne d'état $\boldsymbol{\mu}$ et la covariance$\boldsymbol{\Sigma}$. Donc, afin d'estimer un vecteur plus grand, nous devons également suivre une matrice Sigma plus grande. Plus précisément, nous allons étendre la matrice de covariance $\boldsymbol{\Sigma}$ de 3x3 à (2n+3) x (2n+3) où n est à nouveau le nombre de points de repère.

L'étape de la mise à jour

Pour s'adapter à la taille plus importante de notre vecteur d'état, nous devrons également apporter quelques modifications à l'algorithme EKF. Commençons par l'étape de mise à jour, où nous actualisons la position du robot en fonction des entrées de commande u. Nous devons encore mettre à jour les éléments du vecteur d'état et de la matrice de covariance qui correspondent à la position du robot. Il nous faut donc un moyen d'actualiser la position du robot sans affecter les estimations des points de repère. La réponse est ici une matrice spéciale que nous appelleronsF_k. F_k est simplement une matrice 3x2n+3 où le bloc le plus à droite est une matrice d'identité 3x3 et les autres entrées sont des zéros :

$$F_k = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \end{bmatrix}$$

F_k est utilisé pour cartographier les mises à jour de la moyenne de position du robot et la covariance, qui comprend les positions des points de repère. L'étape de mise à jour ressemble donc maintenant à ceci :

$$\bar{\mu}_k = \mu_{k-1} + {F^T}_k * u$$

$$G_k = I_{2n+3} + {F^T}_k * g_k * F_k$$

$$\bar{\Sigma}_k = G_k * \Sigma_{k-1} * {G^T}_k + {F^T}_k * R * F_k$$

Où u est l'entrée de contrôle, g_k est le jacobien de mouvement évalué au pas de temps k, I_{2n+3} est une matrice d'identité (2n+3) x (2n+3), et R est la covariance de bruit de processus constant.

L'étape de la mesure

Nous devons procéder à un changement similaire dans l'étape de mesure. Une mesure donnée correspond à un seul point de repère, de sorte que la mesure ne doit pas affecter les estimations des autres points de

repère. Cependant, chaque mesure affecte également l'estimation de la position du robot. Nous aurons donc besoin d'un moyen de cartographier nos corrections depuis les amers et l'état de position, tout en veillant à ce que seules les quantités de position dans le vecteur moyen, μ et la matrice de covariance, Σ soient affectées. Pour cela, nous avons besoin d'une autre matrice F. Cette fois, ce seraF_{ij}, l'indice ij signifiant qu'il relie la position i au point de repère j. F_{ij}est une matrice 5x2n+3. Le bloc 3x3 en haut à gauche est une matrice d'identité, correspondant aux termes de position dans la matrice d'état et de covariance. F_{ij} possède également un bloc d'identité 2x2 occupant les deux lignes inférieures et commençant à l'indice 2j+2 où j est l'indice du point de repère correspondant à la mesure actuelle. Toutes les autres entrées dans F_{ij}sont des zéros :

$$F_k = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 & \dots \\ 0 & 1 & 0 & & 0 & 0 & \\ 0 & 0 & 1 & \vdots\vdots & 0 & 0 & \vdots\vdots \\ 0 & 0 & 0 & \dots & 1 & 0 & \dots \\ 0 & 0 & 0 & & 0 & 1 & \end{bmatrix}$$

Cette matrice nous permet d'incorporer une mesure à la fois, en mettant seulement à jour les termes dans les matrices d'état et de covariance qui correspondent à la position du robot et au point de repère pertinent. Nous procédons de la manière suivante :

$$H_k = h_k * F_{ij}$$

$$K = \bar{\Sigma}_k * {H_k}^T (H_k * \bar{\Sigma}_k * {H_k}^T + Q)^{-1}$$

$$\mu_k = \overline{\mu_k} + K(z - \hat{z})$$

$$\Sigma_k = (I_{2n+3} - K * H_k) * \bar{\Sigma}_k$$

Où h_k est la mesure jacobienne.

Plusieurs tests ont été effectués dans le cadre du V-REP, principalement en ce qui concerne localisation et cartographie. Pour la réalisation des tests expérimentaux, a été créé le scénario présenté dans la Fig. 3.9, avec des murs entourant l'espace utile et des points de repère (des amers)

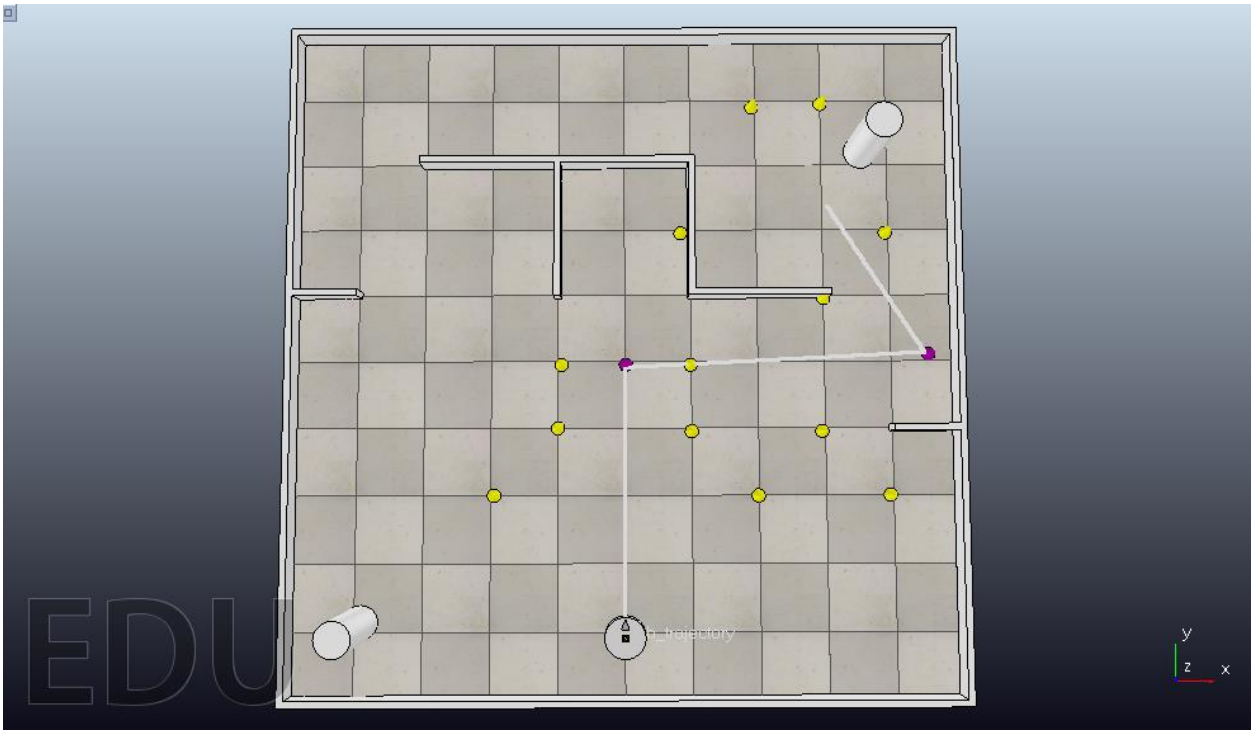

Figure 3.9 Scène et trajectoire du robot dans V-REP

La figure 3.10 montre le résultat de la simulation utilisant l'algorithme EKF-SLAM aussi l'emplacement des amers. À chaque instant le robot va estimer l'emplacement de chaque amer capté par les capteurs du robot ou par les amers. Après le robot estime leur position future.

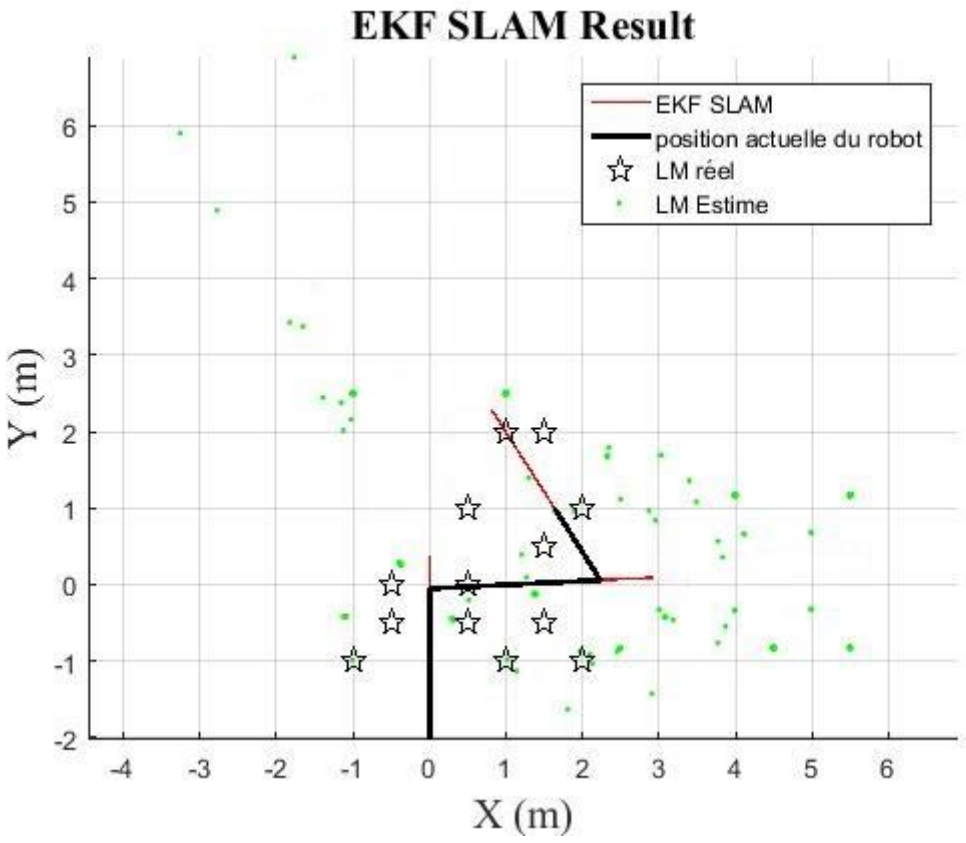

Figure 3.10 trajectoire du robot dans MATLAB utilisant l'algorithme EKF-SLAM.

L'EKF a été le mieux adapté à la trajectoire mais cela ne signifie pas qu'il soit le meilleur filtre, car nous voyons que à chaque fois le robot tourne l'algorithme nous donne une estimation fausse et n'oublient pas que la complexité de cette algorithme monte a O (n^3) à cause du nombre d'amers.
Le code de l'algorithme complet est indiqué dans l'annexe. Il est très similaire au code pour la localisation EKF avec une corrélation connue avec le vecteur d'état et les matrices de covariance qui incluent maintenant des points de repère et avec les étapes de mise à jour et de mesure augmentées comme discuté ci-dessus. Notez également qu'après l'estimation des états, le système calcule la somme des erreurs quadratiques pour les positions du robot et les positions des points de repère. Cela nous donne un moyen approximatif de juger de la qualité du réglage de notre filtre.

III-7. Conclusion

Dans ce chapitre, nous avons montré tous les processus associés au cycle de mise à jour du filtre de Kalman étendu (EKF), la localisation et la cartographie simultanées (SLAM) peuvent être réalisés de manière linéaire dans le temps avec le nombre d'éléments cartographiques. Nous décrivons l'algorithme SLAM de EKF, qui consiste à réduire la complexité de calcul ou le coût de calcul. Contrairement à de nombreuses techniques EKF SLAM actuelles à grande échelle, cet algorithme calcule une solution sans se baser sur des approximations ou des simplifications (autres que des linéarisations) pour réduire la complexité de calcul. De plus, des estimations et des covariances sont disponibles lorsque l'association de données le nécessite sans autre calcul. En outre, comme la méthode fonctionne la plupart du temps sur des

cartes locales, où les erreurs angulaires restent faibles, l'effet des erreurs de linéarisation est limité. Les estimations de robot et des cartes qui en résultent sont plus précises que celles obtenues avec la méthode standard EKF. Les erreurs par rapport à la valeur réelle sont plus faibles, et la covariance d'état calculée est cohérente avec l'erreur réelle de l'estimation. Les expériences simulées dans l'environnement MATLAB-VREP pour démontrer les avantages de cet algorithme.

Conclusion générale

Les algorithmes de localisation et de cartographie simultanée ont été largement étudiés pendant ces dernières années. Nous avons commencé par une présentation de la problématique générale de la localisation et de cartographie et nous avons défini l'architecture générale d'un système de SLAM.

Nous avons vu une variété de capteurs qui peuvent être utilisés dans les applications SLAM. Les capteurs proprioceptifs fournissent une estimation non-consistante de l'état future d'un système de SLAM. Les capteurs extéroceptifs, appelés aussi systèmes de perception, contribuent à l'amélioration de l'estimation de l'état future. Nous avons ensuite formulé Le modèle d'évolution ou bien cinématique interprète le mouvement du robot dans l'espace.

La deuxième partie, nous avons cité les approches algorithmiques principales utilisées pour résoudre le cœur du SLAM. Les algorithmes basés sur des approches probabilistes semblent être plus utilisés pour la résolution du problème SLAM, tandis que ceux bio-inspirés sont peu étudiés bien qu'il existe des recherches en cours de réalisation afin d'améliorer leur consistance algorithmique et de faciliter leur mise en œuvre.

Finalement, nous avons développé un cadre de prototypage et de simulation basé sur V-REP et MATLAB, qui a démontré être une combinaison appropriée d'un simulateur de robot dynamique avec un environnement expérimental de prototypage rapide.

Références

[1] Smith, R. C., and Cheeseman, P.. «On the representation and estimation of spatial uncertainty», in *The international journal of Robotics Research 5*,1986, *pp.56–68,*

[2] Durrant-Whyte, H. F. « Uncertain geometry in robotics. », *IEEE Journal on Robotics and Automation 4,* ,(Feb 1988) *pp.23–31,*

[3] Smith, R., Self, M., and Cheeseman, P. «Estimating uncertain spatial relationships in robotics. » *arXiv preprint arXiv :1304.3111, 2013.*

[4] Fengbing Luo, Bianjing Du and Zhen Fan, «Mobile robot localization based on particle filter.» *Proceeding of the 11th World Congress on Intelligent Control and Automation, Shenyang, 2014, pp. 3089-3093.*

[5]- A. C. Almeida, S. R. J. Neto and R. A. C. Bianchi, «Comparing Vision-Based Monte-Carlo Localization Methods.» *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE), Joao Pessoa, 2018, pp. 437-442.*

[6]- A. Benini , A. Mancini , A. Marinelli , S. Longhi «A Biased Extended Kalman Filter for Indoor Localization of a Mobile Agent using Low-Cost IMU and UWB Wireless Sensor Network. » *IFAC Proceedings Volumes 45, Issue 22, 2012, Pages 735-740.*

[7] El Hamzaoui, O. «Simultaneous Localization and Mapping for a mobile robot with a laser scanner : CoreSLAM. » *PhD thesis, Ecole Nationale Supérieure des Mines de Paris, Sept. 2012.*

[8] Kuka Robotics. Kuka Navigation Solution 2016. https://www.kuka.com/-/media/kuka downloads/imported/9cb8e311bfd744b4b0eab25ca883f6d3/kuka_navigation_solution_en.pdf [accessed *15-may-2020]*.

[9] Mark Maimone, Yang Cheng et Larry Matthies « Two years of visual odometry on the mars exploration rovers. » *Journal of Field Robotics, 24 pp169–186, 2007.*

[10] IEEE Spectrum. «AutomatonRoboticsHome Robots Dyson's Robot Vacuum Has 360-Degree Camera, Tank Treads, Cyclone Suction. » https://spectrum.ieee.org/automaton/robotics/+home-robots/dyson-the-360-eye-robot-vacuum.[Online ; accessed 01- *juin* -2020].

[11] Google. Project tango. https://developers.google.com/project-tango/ 2016. *[Online ; accessed 10-juillet 2020]*.

[12]. Ko N Y, Kim T G. «Comparison of Kalman filter and particle filter used for localization of an underwater vehicle. »*The Journal of China Universities of Posts and Telecommunications Volume 21, Issue 6, December 2014, Pages 78-86*

[13] Idris M Y I, Arof H, Noor N M, et al. « A co-processor design to accelerate sequential monocular SLAM EKF process. »*Measurement 45,8, October 2012, Pages 2141-2152*

[14] Ligorio G, Sabatini A M. «Extended Kalman filter-based methods for pose estimation using visual, inertial and magnetic sensors: Comparative analysis and performance evaluation. »*Sensors, 13(2) 2013, Pages 1919−1941*

[15] Dissanayake M W M G, Newman P, Clark S, et al. « A solution to the simultaneous localization and map building (SLAM) problem. » *IEEE Transactions on Robotics and Automation, 17(3) , 2001 Pages 229−241*

[16] Chen P, Li J H, Niu K, et al. « Particle filter based automatic frequency control scheme by combining the two-step structure. » *The Journal of China Universities of Posts and Telecommunications , 2012, 19(2): 9−14*

[17] Zhou F, He W J. «Tracking application about singer model based on marginalized particle filter. » *The Journal of China Universities of Posts and Telecommunications, 2010, 17(4): 47−51*

[18] Miao L F, Zhang J J, Chakrabarti C, et al. « Efficient Bayesian tracking of multiple sources of neural activity: Algorithms and real-time FPGA implementation. » *IEEE Transactions on Signal Processing, 2013, 61(3): 633−647, 2013*

[19] Georgy J, Noureldin A, Korenberg M J, et al. « Low-cost three-dimensional navigation solution for RISS/GPS integration using mixture particle filter. » *IEEE Transactions on Vehicular Technology,2010, 59(2): 599−615.*

[20] Li H W, Wang J. « Particle filter for manoeuvring target tracking via passive radar measurements with glint noise. IET Radar, » *Sonar & Navigation, 2012, 6(3): 180−189.*

[21] Ko N K, Kim T G, Moon Y S. « Particle filter approach for localization of an underwater robot using time difference of arrival. » *Proceedings of the OCEANS Conference (OCEANS'12), May 21−24, 2012, Yeosu, Republic of Korea. Piscataway, NJ, USA: IEEE, 2012: 7p*

[22] . He B, Zhang S J, Yan T H, et al. « A novel combined SLAM based on RBPF-SLAM and EIF-SLAM for mobile system sensing in a large scale environment. » *Sensors, 2011, 11(11): 10197−10219*

[23] . Fox D, Hightower J, Liao L, et al. « Bayesian filtering for location estimation. » *IEEE Pervasive Computing, 2003, 2(3): 24−33*

[24] J. Borenstein , H. R. Everett , and L. Feng Contributing authors: S. W. Lee and R. H. Byrne «Where am I? Sensors and Methods for Mobile Robot Positioning *» The university of Michigan, avril 1996.*

[25] M. Agrawal et K. Konolige. «Real-time Localization in Outdoor Environments using Stereo Vision and Inexpensive GPS. *» In Pattern Recognition, 2006. ICPR 2006. 18th International Conference on, volume 3, pages 1063–1068, 2006.*

[26] D. Nister, O. Naroditsky et J. Bergen. « Visual odometry. In Computer Vision and Pattern Recognition. *» 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on, volume 1, pages I–652–I–659, 2004.*

[27] A. Burguera, G. Oliver et J.D. Tardos. « Robust scan matching localization using ultrasonic range finders. » *In Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on, pages 1367–1372, 2005.*

[28] Dhaouadi, R., & Hatab, A. A. (2013). «Dynamic Modelling of Differential-Drive Mobile Robots using Lagrange and Newton-Euler Methodologies: A Unified Framework. *» Advance in Robotics & Automation 2013, pp2:2.*

[29] Bruno Siciliano, Oussama Khatib « springer-handbook-of-robotics (2nd Eds.) . » *Springer-Verlag Berlin Heidelberg 2016.*

[30] P. S. Maybeck. «Stochastic Models, Estimation and Control. »*Academic Press New York* 1979.

[31] Simon, D. (2006) «Optimal State Estimation: Kalman, H∞, and Nonlinear Approaches, »*17 January 2006 2006 John Wiley & Sons, doi: 10.1002/0470045345.*

Annexe :

Le code de l'algorithme EKF-SLAM

La fonction principale main.m

```
% de la simulation, ajoutera les chemins de matlab
global connection;
global bodyDiameter;
%global variables
global dt;
global PoseSize;     % Nombre d'états de posture du robot [x,y,yaw]
global LMSize;       % positions des amers [x,y] landmark
global Q;            % lerreur de mesure  produit par les capteurs
global Qsigma;       % Matrice de covariance pour l'observation
global Rsigma;       % Matrice de covariance pour la prediction


connection = simulation_setup();

% le robot avec lequel nous voulons interagir
robotNb = 0;

% ouvrir la connexion
connection = simulation_openConnection(connection, robotNb);

% lancer la simulation si elle n'a pas déjà commencé
simulation_start(connection);

% initialiser la connexion
[bodyDiameter wheelDiameter interWheelDist scannerPose] =
bob_init(connection);
[bodyDiameter] = bob_getBodyDiameter(connection);
[wheelDiameter] = bob_getWheelDiameter(connection);
[interWheelDist] = bob_getInterWheelDistance(connection);
[scannerPose] = bob_getScannerPose(connection);

% Variable pour stocker le résultat du calcul
result.time=[];    % temps de simulation
result.xTrue=[];   % position réel
result.xEst=[];    % position estime
result.z=[];       %l'observation du robot
result.PEst=[];    %la covariance estime
result.u=[];       %la commande


time = 0;
endtime = 6; % % Simulation end time [sec]
dt = 0.1; % % Temps de l'étape de simulation [sec]
deltaT= 0.1;
nSteps = ceil((endtime - time)/dt); % Nombre d'étapes de simulation
PEst = eye(3);
initP=eye(2)*1000;
```

```
% Vecteur d'état [x y gamma]'
% Affectation du vecteur aux lectures réelles du V-REP
[x y gamma] = bob_getPose(connection);
xEst=[x y gamma]';

PEst = eye(3);
PoseSize=length(xEst);
LMSize=2;
% État réel
xTrue=xEst;
% Matrice de covariance pour prédiction
R=diag([0.2 0.2 toRadian(1)]).^2;

% Matrice de covariance pour l'observation
Q=diag([10 toRadian(30)]).^2;%range[m], Angle[rad]

Qsigma=diag([0.1 toRadian(20)]).^2;
Rsigma=diag([0.1 toRadian(1)]).^2;
% Position des amers  [x, y] 3 amers [x1y1 x2y2 x3y3]
LM=[1 -1;-1 -1;0.5 -0.5;-0.5 -0.5;0.5 0; -0.5 0;0.5 1;2 -1;2 1;1 2; 1.5 2;1.5
-0.5 ;1.5 0.5];

MAX_RANGE=2;  %Distance d'observation maximale 2 m
for  i=1 : nSteps
% Courte pause

pause(dt);
time = time + dt;

tempo_atual = connection.vrep.simxGetLastCmdTime(connection.clientID)/1000;

% la commande
u=doControl(x,y,gamma);
% Observation
[z,xTrue,xd,u]=Observation(xTrue, xd, u, LM, MAX_RANGE);

% Première lecture
      if flag_first_read
         tempo_first = tempo_atual;
      end

% Enregistrement du temps de calcul actuel
     base_tempo(aux_i) = tempo_atual - tempo_first;
     disp(strcat('Tempo de simulacao: ',num2str(base_tempo(aux_i))));

[x ,y ,gamma] = bob_getPose(connection);
xTrue = [x ,y ,gamma]';
```

```
% Gestion des données pour les dossiers obj

     if flag_first_read
     else
            % Calcul du deltaT
 deltaT = base_tempo(length(base_tempo))  -base_tempo(length(base_tempo)-1);
          dt = deltaT;

     end



%%Traitement  pour le filtre Kalman EXTENDED============================
     % ------ EKF SLAM --------
    % Prediction

    xEst = f(xEst, u)
    [G,Fx]=jacobF(xEst, u);
    PEst= G'*PEst*G + Fx'*R*Fx;
     % Update


    for iz=1:length(z(:,1))%Pour chaque observation

        %Ajouter observations comme points de repère

        zl = CalcLMPosiFromZ(xEst,z(iz,:));%Calculer la position d amer à
partir de la valeur observée

        %Ajouter un vecteur d'état et une matrice de covariance

        xAug=[xEst;zl];
        PAug=[PEst zeros(length(xEst),LMSize);  zeros(LMSize,length(xEst))
initP];

        mdist=[];% Liste des distances de LM amers

        for il=1:GetnLM(xAug) % À propos de chaque point de repère
            if il==GetnLM(xAug)

                mdist=[mdist alpha];% Utilisez la valeur du paramètre pour la
distance du point nouvellement ajouté

            else
                lm=xAug(4+2*(il-1):5+2*(il-1));
                [y,S,H]=CalcInnovation(lm,xAug,PAug,z(iz,1:2),il);
                mdist=[mdist y'*inv(S)*y];% Calcul de la distance de LandMark
            end
        end

        %Correspond à la distance de lamer LM la plus proche

        [C,I]=min(mdist);
```

```
% Si la plus petite distance est ajoutée, la valeur observée est adoptée
comme repère
         if I==GetnLM(xAug)
             disp('New LM')
             xEst=xAug;
             PEst=PAug;
         end
        % Acquisition des données de repère associées
        lm=xEst(4+2*(I-1):5+2*(I-1));

        % Calcul de l'innovation

        [y,S,H]=CalcInnovation(lm,xEst,PEst,z(iz,1:2),I);
        K = PEst*H'*inv(S);
        xEst = xEst+k*y;
        PEst = (eye(size(xEst,1)) - K*H)*PEst;
    end

    xEst(3)=PI2PI(xEst(3));% Correction d'angle
    result.xEst=[result.xEst;xEst(1:3)'];
    result.xTrue=[result.xTrue; xTrue'];
%% ================================END ekf================================


     %% Mise à jour des graphiques

       if ~flag_first_read

         [~, num_leit] = size(x_n);
            % Switch qui décide du type de tracé à imprimer
            x=[ result.xTrue(:,1:2) result.xEst(:,1:2)];
                           % Traçage des trajectoires
                            hold on;
                            plot(x(:,3), x(:,4),'-r','linewidth', 1);
                            plot(x(:,1), x(:,2),'-k','linewidth', 2);
plot(LM(:,1),LM(:,2),'pk','MarkerSize',10);hold on;
for il=1:GetnLM(xEst);
    plot(xEst(4+2*(il-1)),xEst(5+2*(il-1)),'.g');hold on;
end
title('EKF SLAM Result', 'fontsize', 16, 'fontname', 'times');
xlabel('X (m)', 'fontsize', 16, 'fontname', 'times');
ylabel('Y (m)', 'fontsize', 16, 'fontname', 'times');
legend('EKF SLAM','position actuelle du robot','LM réel','LM Estime');
grid on;
axis equal;
end

% Indique que la première lecture est déjà passée
              if flag_first_read
                  flag_first_read = false;
              end
end
```

```
% arrêter la simulation
simulation_stop(connection);
% fermer la connexion
simulation_closeConnection(connection);
```

La fonction f.m décrit le modèle de mouvement

```
function x = f(x, u)
% Motion Model
global dt;
global PoseSize;
global LMSize;

F = horzcat(eye(PoseSize),zeros(PoseSize,LMSize*GetnLM(x)));

B = [dt*cos(x(3)) 0
     dt*sin(x(3)) 0
     0 dt];

x= x+F'*B*u;

x(3)=PI2PI(x(3));
```

la function JacobF.m
Fonction de calcul de la matrice jacobienne du modèle de mouvement

```
function [G,Fx]=jacobF(x, u)

global dt;
global PoseSize;
global LMSize;

Fx = horzcat(eye(PoseSize),zeros(PoseSize,LMSize*GetnLM(x)));

jF=[0 0 -dt*u(1)*sin(x(3))
    0 0 dt*u(1)*cos(x(3))
    0 0 0];
G=eye(length(x))+Fx'*jF*Fx;
```

Fonction pour calculer le jacobien du modèle d'observation

```
function H=jacobH(q,delta,x,i)

sq=sqrt(q);
G=[-sq*delta(1) -sq*delta(2) 0 sq*delta(1) sq*delta(2);
    delta(2)    -delta(1)   -1 -delta(2)    delta(1)];
G=G/q;

%astus pour les mesures
F=[eye(3) zeros(3,2*GetnLM(x));
   zeros(2,3) zeros(2,2*(i-1)) eye(2) zeros(2,2*GetnLM(x)-2*i)];
H=G*F;
```

Fonction qui calcule l'innovation à partir du résultat de l'association ou les résultats observe

```
function [y,S,H]=CalcInnovation(lm,xEst,PEst,z,LMId)
global Q;

delta=lm-xEst(1:2);
q=delta'*delta;
zangle=atan2(delta(2),delta(1))-xEst(3);
zp=[sqrt(q) PI2PI(zangle)];% Prédiction des observations

y=(z-zp)';
H=jacobH(q,delta,xEst,LMId);
S=H*PEst*H'+Q;
```