RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Ecole Nationale Polytechnique

Département d'Automatique

Laboratoire de Commande des Processus

Mémoire pour l'obtention du diplôme
de master en Automatique

Etude Comparative entre les Estimateurs EKF et SASMO dans un contexte de SLAM

Anes BENMERZOUG

Sous la direction de
Mr. M. TADJINE Professeur
Mr. M. CHAKIR Enseignant Chercheur

Présenté et soutenu publiquement le 26/06/2016

Composition du Jury :

Président	Mr. M.S. BOUCHERIT	Professeur	Ecole Nationale Polytechnique
Promoteurs	Mr. M. TADJINE	Professeur	Ecole Nationale Polytechnique
	Mr. M. CHAKIR	Enseignant Chercheur	Ecole Nationale Polytechnique
Examinateur	Mr. B. HEMICI	Professeur	Ecole Nationale Polytechnique

ENP 2016

ملخص

في هذه المذكرة, سوف نقوم بدراسة مقارنة بين فلتر كالمان الممدد EKF و مراقب انزلاقي تكيفي ستوكاستيكي SASMO و ذلك في سياق اشكالية التحديث و الاستحداث الآني البصري الأحادي(SLAM monoculaire)ولنوع الروبوت quadrotor . نبدأ بتقديم اشكالية التحديث و الاستحداث الآني SLAM بشكل عام والبصري الأحادي بشكل خاص ونقدم حالته من التطور ونقترح نهجين لحله، واحد باستعمال EKF والأخر باستعمال SASMO. ثم نقدم ROS و GAZEBO المفعل من أجل تنفيذ و محاكاة النهجين المقترحين. وأخيرا، فإننا سوف نقوم بمقارنة نوعية وكمية بين النهجين.

الكلمات المفتاحية : EKF , SASMO , الأحادي التحديث و الاستحداث الآني (SLAM monoculaire) , الأحادي التحديث و الاستحداث الآني SLAM , GAZEBO , ROS

Abstract

In this thesis, we will make a comparative study between the extended Kalman filter (EKF) and a stochastic adaptative sliding observer (SASMO) in the context of monocular visual SLAM and for a quadrotor. We begin by presenting the SLAM problem in general and the monocular SLAM in particular and we present its state of the art and propose two approaches to solve it, the first using the EKF and the second using the SASMO. Then, we introduce both ROS and Gazebo and we will use them to implement and simulate realistically the two proposed approaches. Finally, we will make a qualitative and quantitative comparison between the two approaches.

Keywords: EKF, SASMO, Sliding Observer, SLAM, Monocular SLAM, ROS, Gazebo, UAV, quadrirotor, RMSE, MAE, Standard-deviation.

Résumé

Dans ce mémoire, nous allons faire une étude comparative entre le filtre de Kalman étendu (EKF) et un observateur glissant stochastique et adaptatif (SASMO) et ce dans un contexte de SLAM visuel monoculaire et pour un robot mobile du type quadrirotor. Nous commencerons par présenter le problème du SLAM en général et du SLAM visuel monocualire en particulier et nous presénterons son état de l'art et nous proposerons deux approches pour le résoudre, l'une avec l'EKF et l'autre avec SASMO. Puis, nous introduirons ROS et Gazebo que nous utiliserons pour implémenter et simuler de façon réaliste les deux approches proposées. Enfin, nous ferons une comparaison qualitative et quantitative entre les deux.

Mots clés: EKF, SASMO, Observateur Glissant, SLAM, SLAM Visuel Monoculaire, ROS, Gazebo, drone, quadrirotor, RMSE, MAE, Ecart-type.

Dédicace

Je dédie ce travail à mes très chers parents, dont le sacrifice, la tendresse, l'amour, la patience, le soutien, l'aide et l'encouragement sont l'essentiel de ma réussite. Sans eux je ne serai pas à ce stade aujourd'hui.

A mon frère et ma sœur pour leur soutien continue durant mon parcours.

A ma grande famille.

Et à tous mes amis.

Anes

Remerciements

Je remercie Dieu le tout puissant de nous avoir donné la force et le courage pour réaliser ce travail.

Les travaux présentés dans ce mémoire ont été effectués au sein du Laboratoire de Commande des Processus (LCP) de l'Ecole Nationale Polytechnique.

Ce travail que je présente a été effectué sous la direction de Monsieur M. Tadjine, professeur à l'Ecole Nationale Polytechnique, et Monsieur M. Chakir, enseignant chercheur à l'Ecole Nationale Polytechnique, qui ont suivie de très près ce travail et que je remercie pour leur orientation pédagogique dans l'élaboration de ce mémoire.

Je tiens à remercier monsieur M.S. Boucherit, professeur à l'Ecole Nationale Polytechnique, pour l'honneur qu'il nous fait de présider le jury de notre soutenance.

Que Monsieur Achour, enseignant chercheur à l'Ecole Nationale Polytechnique, soit convaincu de ma sincère reconnaissance pour avoir accepté d'examiner et de critiquer ce mémoire.

Enfin, je tiens à remercier tous les gens qui ont contribué à notre réussite tout au long de notre parcours d'étude.

Table des matières

Liste des tableaux

Liste des figures

Introduction Générale

Au cours des dix dernières années, les avancées technologiques et les nombreuses applications potentielles ont suscité un intérêt croissant pour la robotique mobile. Ces applications vont des tâches de surveillance et de maintenance jusqu'aux missions de sauvetages.

Il s'agit d'un domaine multidisciplinaire qui regroupe des aspects concernant la mécanique, l'informatique ou encore l'électronique.

L'une des tâches les plus importantes pour un robot mobile autonome est la navigation. Cette dernière lui permet de se mouveoire dans des environnements inconnus.

La localisation et la cartographie simultanées (SLAM) est une étape primordiale de la navigation, elle permet de découvrir l'environnement qui entoure le robot mobile et de le positionner par rapport à son entourage[1].

Dans ce mémoire, nous allons nous intéresser au problème du SLAM visuel monoculaire et plus particulièrement aux filtres utilisés pour faire la fusion des données et l'estimation. Nous nous intéresserons aussi à son implémentation sous ROS et à sa simulation sous Gazebo pour un robot mobile du type quadrirotor.

Le quadrirotor est l'une des plateformes aériennes mobiles les plus flexibles et adaptables dans le domaine de la recherche. C'est un drone à voilure tournante de type VTOL (Vertical Take Off and Landing).

Dans le premier chapitre, nous commencerons par la définition du SLAM et particulièrement du SLAM visuel monoculaire. Puis, nous enchainerons par une présentation de l'état de l'art en ce sujet, ses inconvénients ainsi que les algorithmes de fusion de données et d'estimation utilisés pour combler ses lacunes. Nous finirons par expliquer deux approches de SLAM visuel monoculaire que nous implémenterons au deuxième chapitre.

Dans le deuxième chapitre, nous présenterons ROS et Gazebo. Puis, nous implémenterons les deux approches de SLAM visuel monoculaire présenté au chapitre précédent. Ensuite, nous les simulerons sous Gazebo pour un robot mobile du type quadrirotor. Nous finirons par une étude comparative entre les deux approches.

Chapitre 1

SLAM Visuel Monoculaire

1.1 Introduction

Un robot mobile autonome doit pouvoir naviguer dans des environnements inconnus. Mais, avant de pouvoir commencer à se déplacer, il doit être capable de se localiser dans l'environnement dans lequel il évolue. De plus, il doit être capable de le cartographier afin de disposer de points de repères pour améliorer sa localisation. Lorsque se deux étapes se font en même temps cela devient un problème de SLAM.

Plusieurs types de capteurs ont été utilisés pour récolter des informations sur l'environnement du robot mobile et résoudre ainsi le problème du SLAM. Parmi ces capteurs, la caméra est celle qui offre le plus de possibilité de par son rapprochement à la solution biologique qu'offre les yeux des êtres humains et des animaux à ce problème.

Dans ce chapitre, nous allons définir ce qu'est le SLAM et en particulier le SLAM visuel monoculaire. Puis, nous allons présenter l'état de l'art de ce dernier, les inconvénients des approches utilisées et quelques uns des algorithmes de fusion de données utilisés pour les combler. Enfin, nous expliquerons les deux approches que nous proposons pour résoudre le problème du SLAM visuel monoculaire et que nous implémenterons et simulerons au chapitre suivant.

1.2 Définitions

Simulataneous Localization And Mapping (SLAM) ou Localisation et Cartographie Simultanées est un processus par lequel un robot mobile peut construire une carte de son environnement (une carte composées d'amers, qui sont des points de repères supposés statiques) et en même temps utiliser cette carte pour déduire son emplacement[2].

Parmi les variantes du problème de SLAM se trouve le SLAM visuel monoculaire. Le principe de la localisation par vision monoculaire d'un véhicule est identique pour la grande majorité des méthodes. La première position de la caméra est inconnue et est donc fixée arbitrairement. Par la suite, le déplacement en trois dimensions de la caméra dans l'environnement qui l'entoure va se traduire visuellement dans l'image par un déplacement en deux dimensions des éléments d'intérêt de l'image. Ces éléments d'intérêt sont généralement des zones ou plus couramment des points de l'image qu'il est possible de suivre au fil du flux vidéo, c'est à dire entre les différentes images. A partir de l'observation du déplacement de ces éléments d'intérêt, il est possible d'inférer le mouvement de la caméra.

1.2.1 Etat de l'art

Différentes questions sont impliqués dans le problème du SLAM visuel et de nombreuses approches différentes existent afin de les résoudre. Nous allons présenter quelque uns des travaux majeurs qui ont été réalisés.

MonoSLAM (2003)

MonoSLAM[3], est le premier algorithme de SLAM Monoculaire à être capable de fonctionner en temps-réel. Le concept clé de MonoSLAM est l'utilisant d'une carte probabiliste des caractéristiques de l'environnement, qui représente à un instant donné la position et l'orientation estimée de la caméra et de tout les points d'intérêts, qui sont regroupées dans un vecteur d'état, ainsi que l'incertitude de ces estimées, qui sont regroupées dans une matrice de covariance, figure 1.2. Le vecteur d'état et la matrice de covariance augmentent à chaque nouvelle observation et les estimées sont misent à jour grâce à un filtre de Kalman étendu (EKF). C'est là que réside son inconvénient majeure. Le temps de calcul est proportionnel au carré du nombre de points de d'intérêts, ce qui veut dire que l'algorithme devient rapidement inutilisable pour de grands environnements.

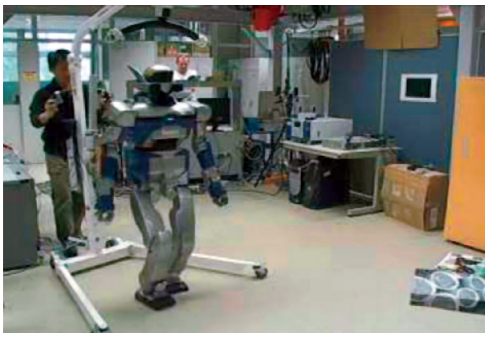


FIGURE 1.1: Robot HRP-2 qui marche d'une manière autonome en utilisant MonoSLAM

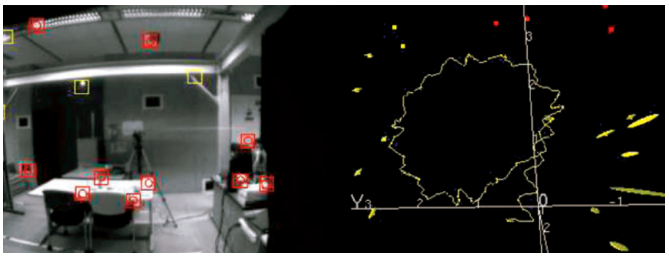


FIGURE 1.2: Extrait de MonoSLAM, en jaune la trajectoire estimée du robot HRP-2

Parallel Tracking and Mapping - PTAM (2007)

PTAM[4] est un algorithme d'estimation de la position et de l'orientation de la caméra dans un environnement inconnue. Bien que cela ai déjà été tentée par l'adaptation des algorithmes de SLAM développés pour l'exploration robotique. PTAM propose une solution simple, efficace, et l'entrée de bas niveau à ce problème de SLAM. Plutôt que de compter sur des descripteurs de couleur, des marqueurs placés à des endroits spécifiques dans l'environnement, et les modèles de CAO de l'environnement existant, PTAM utilise l'algorithme Fast[5] (Features from Accelerated Segment Test) qui est un algorithme de détection de caractéristiques qui sont rapides à détecter et relativement peu chéres à traiter.

PTAM exécute la détection de la position et de l'orientation de la caméra et la cartographie sur l'environnement en parallèle réduisant ainsi les coûts de calcul et lui permettant de fonctionner en temps-réel. Les points forts de PTAM sont aussi ses limites : C'est un système relativement simple, de sorte qu'il n'extrait aucun sens structurel de l'environnement reconstruit autre que les points détectés par l'algorithme Fast. Il est également limité par la nécessité d'effectuer l'initialisation stéréo d'une manière très spécifique qui nécessite l'intervention de l'utilisateur.

Cette algorithme peut être résumé en ces cinq points :

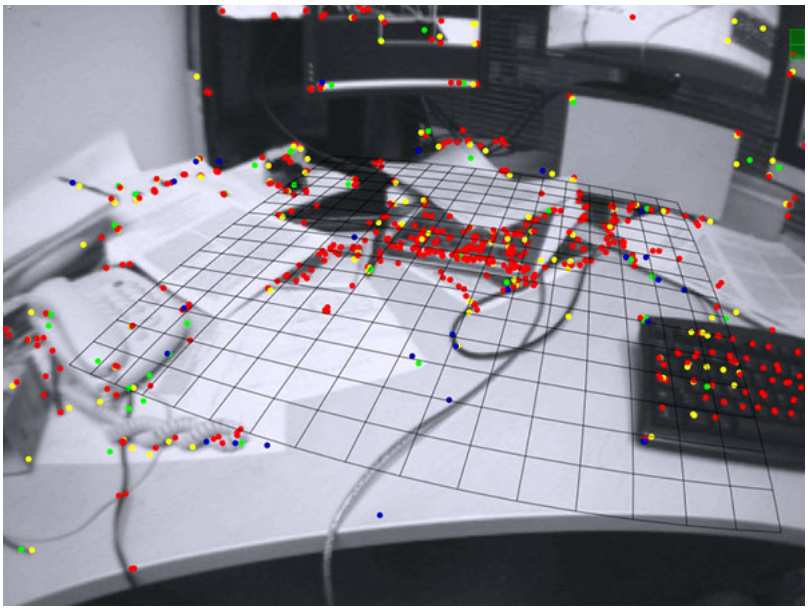


FIGURE 1.3: Exemple d'usage de PTAM

- Le suivi et la cartographie sont séparés et fonctionnement en parallèle dans deux fils d'exécution
- La cartographie est basée sur des images clés, qui sont traités par lots en utilisant l'ajustement de faisceaux (ou Bundle Adjustment[6])
- La map est initialisée en utilisant une paire d'images grâce à l'algorithme des 5 points (5-point Algorithm)
- De nouvelles amers sont initialisées à l'aide d'une fouille epipolaire
- Un grand nombre d'amers sont cartographiés

Dense Tracking And Mapping - DTAM (2011)

DTAM[7] est un algorithme qui permet le suivi de la position et de l'orientation de la caméra et la reconstruction de l'environnement en temps réel. Il ne repose pas sur l'extraction de caractéristiques à partir de l'image, mais sur des méthodes denses qui utilisent tout les pixels de l'image. Comme une seule caméra RGB main vole au-dessus d'une scène statique, nous estimons cartes de profondeur texturés détaillées à des images clés sélectionnés pour produire une mosaïque de surface avec des millions de sommets. Nous utilisons les centaines d'images disponibles dans un flux vidéo pour améliorer la qualité d'un terme de données photométriques simple, et de minimiser une énergie globale spatialement régularisée fonctionnelle dans un cadre novateur d'optimisation non-convexe. Entrelacé, nous suivons 6DOF le mouvement de la caméra précisément par frame-rate l'alignement d'image entier contre l'ensemble du modèle dense. Nos algorithmes sont très parallelisable partout et DTAM réalise des performances en temps réel en utilisant le courant matériel produit GPU

Large-Scale Direct monocular SLAM - LSD-SLAM (2014)

LSD-SLAM[8] est une technique directe monoculaire SLAM : Au lieu d'utiliser des points clés, il opère directement sur les intensités d'image à la fois pour le suivi et la cartographie. La caméra est suivie à l'aide d'alignement d'image directe, tandis que la géométrie est estimée sous la forme de cartes semi-denses profondeur, obtenue par filtration sur de nombreuses comparaisons pixel par pixel stéréo,l'approche est utilise les

FIGURE 1.4: Exemple d'une scène reconstruite en 3D à l'aide de DTAM

principales étapes suivantes :

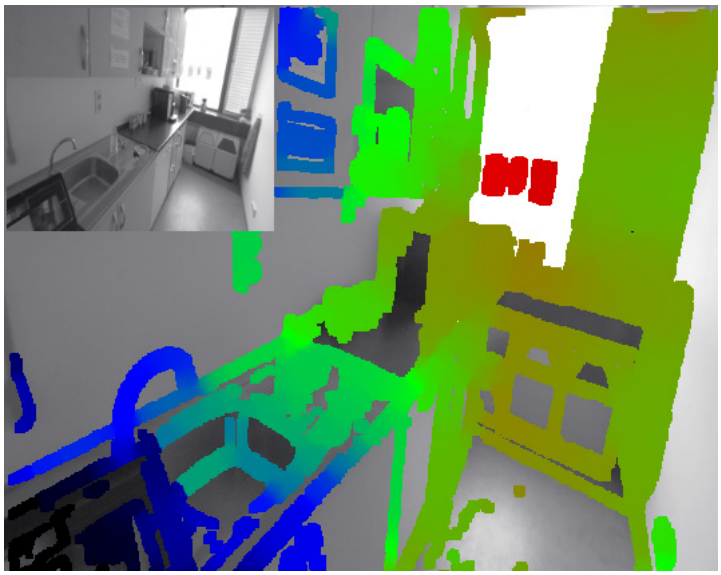


FIGURE 1.5: Exemple d'usage de LSD-SLAM

ORB-SLAM (2015)

ORB-SLAM[9] est un algorithme de SLAM visuel qui utilise le détécteur de zones d'intérêts ORB[10]. Il est capable de calculer en temps réel la trajectoire de la caméra et une reconstruction 3D clairsemée de la scène dans une grande variété d'environnements, allant de petites séquences tenues à la main d'un bureau à une voiture conduite autour de plusieurs pâtés de maisons.

Comme on peut le voir sur la figure 1.7, il est constitué de trois modules qui s'exécutent en parallèle :

Suivi Ce fil d'exécution localise la position et l'orientation de la caméra pour chaque nouvelle image et décide de l'ajout ou non de cette dernière dans une base d'images clés.

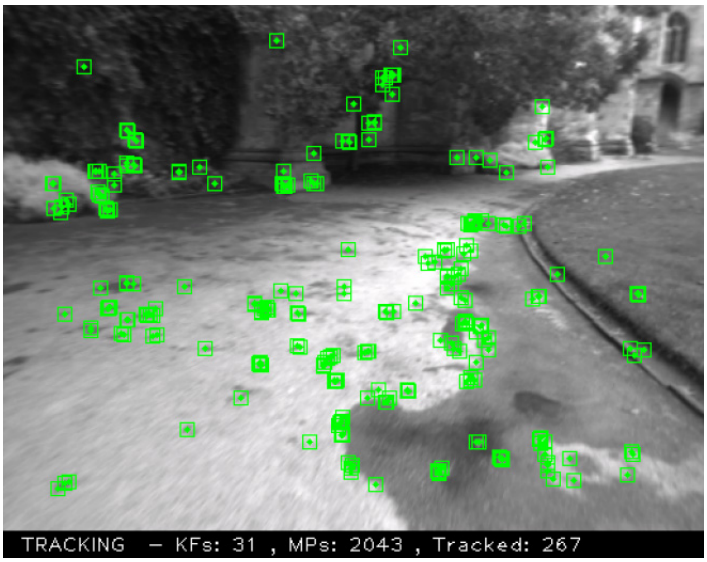


FIGURE 1.6: Exemple d'usage d'ORB-SLAM

Cartographie locale Ce fil d'exécution traitent les images clés et exécute un ajustement de faisceaux local pour obtenir une reconstruction de l'environnement optimale.

Fermeture de boucles Ce fil d'exécution tente de trouver des boucles fermées à chaque nouvelle image clé.

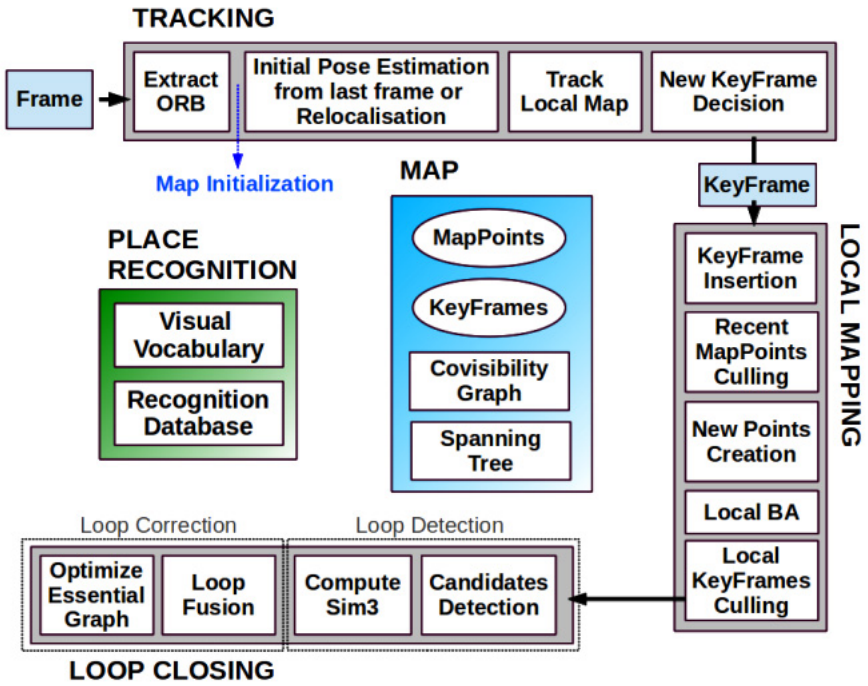


FIGURE 1.7: Schéma des différents modules d'ORB-SLAM

1.2.2 Inconvénients

Parmi les inconvénients du SLAM monoculaire se trouve le manque de robustesse face aux mouvements brusques et l'ambiguïté de l'échelle. C'est pour cela que dés les années 90, des chercheurs ont proposés de combiner les données des caméras avec celles

des centrales inertielles et d'autres capteurs[11] pour exploiter leurs caractéristiques complémentaires[12] :

- Les centrales inertielles sont incapables de distinguer entre un changement d'inclinaison et l'accélération que subit le corps auxquels elles sont attachées, à cause du principe d'équivalence d'Einstein[13].
- Les centrales inertielles ont une grande incertitude pour des mouvements à faible vitesse et une moindre incertitude pour des mouvements à grande vitesse.
- Les caméras peuvent suivre des points d'intérêts avec une très grande précision pour des mouvements à faible vitesse. Cette précision diminue avec l'augmentation de la vitesse.
- Dans une image projective, celle d'une caméra, on ne peut différencier entre un mouvement de translation et un mouvement de rotation[12].
- Pour une caméra, un objet proche avec une vitesse relative faible est semblable à un objet lointain avec une grande vitesse relative[12].

1.3 Algorithmes d'estimation

Les algorithmes de fusion de données et d'estimation peuvent être réparties en deux catégories :

1.3.1 Estimation par lots

Les méthodes d'estimation par lots, ou Batch Estimation en Anglais, sont des techniques qui consistent à optimiser un ensemble de paramètres, trajectoires et positions de points d'intérêts en utilisant toutes les données de la caméra et de la centrale inertielle, afin de minimiser l'erreur de reprojection. L'algorithme classiquement utilisé est la méthode des moindres carrés non-linéaires de Levenberg-Marquardt[14].

Les reconstructions par estimation par lots sont généralement plus précises que les méthodes récursives, mais elles souffrent d'une complexité de calcul plus importante, qui est proportionnel au cube du nombre de paramètres à optimiser. On peut dès lors distinguer les méthodes hors-ligne (très lente) qui utilisent un maximum de paramètres pour la reconstruction 3D et l'estimation de la trajectoire et les méthodes en-ligne, qui réduisent le nombre de paramètres à optimiser pour traiter le flux de données en temps réel.

Les méthodes hors-ligne sont utilisés en fin de trajectoire, après avoir fini l'acquisition de toutes les données. Elles sont généralement utilisées pour la reconstruction de modèles 3D texturés de l'environnement.

Les méthodes en-ligne, contrairement aux méthodes hors-ligne où l'ensemble des images et des données inertielles est connu à l'avance, traitent le flux des données au fur et à mesure qu'elles arrivent. L'idée est d'utiliser une approche incrémentale reconstruisant petit à petit l'environnement et la trajectoire du système. Cette approche est le plus souvent utilisée pour résoudre le problème du SLAM, on peut citer les travaux fait en [15, 16, 17, 18].

1.3.2 Estimation récursive

Les méthodes d'estimation récursive sont des techniques qui consistent à optimiser la trajectoire et la position des points d'intérêts en n'utilisant à chaque fois que l'état précédant et les nouvelles données visuelles et inertielles.

EKF

Le filtre de Kalman est un filtre à réponse impulsionnelle infinie qui estime les états d'un système dynamique à partir d'une série de mesures incomplètes ou bruitées. Le filtre a été nommé d'après le mathematicien et informaticien américain d'origine hongroise Rudolf Kalman.

Kalman propose une solution récursive au filtrage des données linéaires. Cette méthode, améliorée ensuite par Kalman lui-même et Bucy[19], ouvre de nouvelles pistes de recherche dans le domaine de la navigation autonome des robots mobiles. L'approche de base du filtre est basée sur un cycle récursif nécessitant trois hypothèses pour assurer un fonctionnement, prouvé théoriquement, optimal :

- Un modèle d'évolution linaire du système
- Une relation linaire entre l'état et les mesures
- Un bruit blanc gaussien

Filtre Particulaire

Le filtre particulaire, aussi connu sous le nom de méthode de Monté-Carlo séquetielle, sont des techniques sophistiquées d'estimation de modèles fondées sur la simulation.

Les filtres particulaires sont généralement utilisés pour estimer des réseaux bayésiens et constituent des méthodes 'en-ligne' analogues aux méthodes de Monte-Carlo par chaînes de Markov qui elles sont des méthodes 'hors-ligne' (donc à posteriori) et souvent similaires aux méthodes d'échantillonnage séquentiel par importance (SIS).

S'ils sont conçus correctement, les filtres particulaires peuvent être plus rapides que les méthodes de Monte-Carlo par chaînes de Markov. Ils constituent souvent une alternative aux filtres de Kalman étendus avec l'avantage qu'avec suffisamment d'échantillons, ils approchent l'estimé Bayésien optimal. Ils peuvent donc être rendus plus précis que les filtres de Kalman. Les approches peuvent aussi être combinées en utilisant un filtre de Kalman comme une proposition de distribution pour le filtre particulaire.

1.4 Approches proposées

Pour les approches proposées, nous nous sommes principalement inspirés des travaux effectués en [20, 21]. Les principales différences entre les approche proposées et celle faite en [20, 21] sont dans l'algorithme de SLAM (ORB-SLAM au lieu de PTAM). Pour la première approche nous avons opté d'utiliser l'observateur glissant SASMO pour la fusion et l'estimation des données et pour le second nous avons utilisé le filtre de Kalman étendu (EKF).

Les capteurs utilisés sont les suivants :

Caméra monoculaire Elle est fixée à l'avant du quadrirotor pour voir la scène, sans être gênée par les hélices du quadrirotor. Les images reçus seront traitées par ORB-SLAM.

Son modèle d'observation est le suivant :

$$h_{ORB-SLAM} = \begin{bmatrix} \lambda x & \lambda y & \lambda z & \phi & \theta & \psi \end{bmatrix}^T \tag{1.1}$$

Avec :

λ, l'échelle des estimations d'ORB-SLAM.

Centrale inertielle Elle est fixée au centre de gravité du quadrirotor pour qu'elle puisse mesurer les accélérations linéaires et les vitesses angulaires sans erreurs. Nous l'utilisons pour effectuer deux types de mesures.

Le premier type de mesures consiste à intégrer les accélérations linéaires pour obtenir les vitesses de translation et à utiliser directement les mesures des vitesses angulaires.

Le modèle d'observation est le suivant :

$$h_{IMU-1} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} & \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T \tag{1.2}$$

Le deuxième type de mesures consiste à utiliser le fait que lorsque le quadrirotor est au repos, l'accéléromètre mesure seulement l'accélération due à la gravité et comme on peut le voir sur la figure 1.8 nous pouvons l'utiliser pour mesurer les angles de roulis et tangage comme décrit dans l'équation(1.3).

$$\begin{aligned} \phi &= atan(\tfrac{a_y}{a_z}) \\ \theta &= atan(-cos(\phi) \cdot \tfrac{a_x}{a_z}) \end{aligned} \tag{1.3}$$

Avec a_x, a_y et a_z les mesures filtrées de l'accéléromètre suivant les axes X, Y et Z, respectivement.

Cependant, les mesures des accélérations n'étant pas parfaites, à cause du bruit et des fortes vibrations engendrés par la rotation des quatre rotors, nous ne pouvons les utiliser directement. C'est pour cela que nous utilisons un filtre passe-bas de Butterworth[22] pour les filtrer et ne laisser que les composantes statiques qui représentent l'accélération due à la gravité.

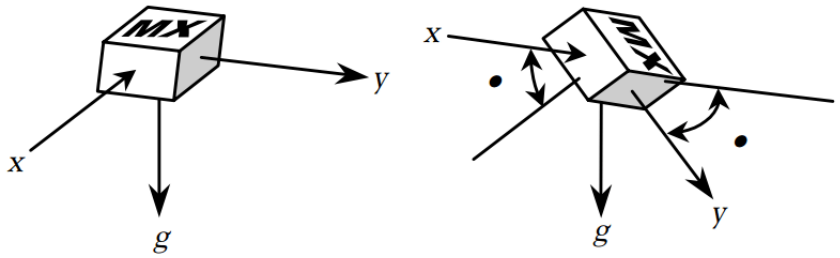


FIGURE 1.8: Mesure des angles en utilisant la gravité

Le modèle d'observation est le suivant :

$$h_{IMU-2} = \begin{bmatrix} \phi & \theta \end{bmatrix}^T \tag{1.4}$$

Sonar ultrason Il est fixé sur la châssis du quadrirotor et est dirigé vers le sol pour qu'il puisse mesurer la hauteur à laquelle se trouve le drone.

Le modèle d'observation est le suivant :

$$h_{Sonar} = z \tag{1.5}$$

Les approches proposées peut être divisée en trois parties, figure **??** :

- ORB-SLAM
- Algorithme de fusion de donnée et d'estimation
- Estimateur de l'échelle

1.4.1 ORB-SLAM

Nous utilisons l'algorithme de SLAM Monoculaire open-source ORB-SLAM pour estimer la position et l'orientation du quadrirotor, ainsi que le position des points d'intérêts dans l'environnement. Cependant, les estimations données par cette algorithme ne sont pas métrique, c'est-à-dire, que l'échelle de ces estimations n'est identique à l'échelle réelle. Pour résoudre ce problème, nous combinons ces données avec celles de la centrale inertielle et du sonar pour estimer l'échelle et pouvoir les fusionner à l'aide de l'observateur.

1.4.2 Algorithme de fusion de donnée et d'estimation

Nous utilisons deux algorithmes de fusion de donnée et d'estimation différents : SASMO et EKF.

Observateur glissant (SASMO)

SASMO acronyme Anglais de Stochastic Adaptative Sliding Mode Observer est un observateur par mode de glissement qui assure, à condition que le système vérifie certaines conditions, que l'erreur quadratique moyenne de l'estimation soit exponentiellement bornée.

Nous allons utiliser l'approche décrite dans les travaux effectués en [23].

Soit un système non-linéaire non-autonome décrit l'équation différentielle d'Itô[24] :

$$\begin{cases} dx_t = (Ax_t + h(x_t)u_t)dt + f(x_t, \zeta_t)dt + g_1(x_t, t)dv_t \\ dy_t = Cdx_t + g_2(x_t, t)dw_t \end{cases} \tag{1.6}$$

Où :

$x_t \in \mathcal{R}^n$, est le vecteur d'état,

$u_t \in \mathcal{R}^p$, est la vecteur de commande,

$\zeta_t \in \mathcal{R}^n$, est le vecteur des perturbations déterministes non mesurables,

$A \in \mathcal{R}^{nn}$, est la matrice de transition qui représente la partie linéaire du système,

$C \in \mathcal{R}^{nn}$, est la matrice de d'observation,

$h : \mathcal{R}^n \to \mathcal{R}^{np}$, est une fonction qui de commande,

$f : \mathcal{R}^n\mathcal{R}^n \to \mathcal{R}^n$, est une fonction qui représente les non-linéarités et les incertitudes du systèmes,

$g_1 : \mathcal{R}^n\mathcal{R}^+ \to \mathcal{R}^{nv}$, est une fonction qui représente l'intensité du bruit du processus,

$g_2 : \mathcal{R}^n\mathcal{R}^+ \to \mathcal{R}^{nw}$, est une fonction qui représente l'intensité du bruit de mesure,

$dv_t \in \mathcal{R}^v$, est un processus stochastique à moyenne nulle et variance dt qui représente l'incrément d'un processus de Weiner $v_t \in \mathcal{R}^v$[25],

$dw_t \in \mathcal{R}^w$, est un processus stochastique à moyenne nulle et variance dt qui représente l'incrément d'un processus de Weiner $w_t \in \mathcal{R}^w$[25].

On admettant les hypothèses suivantes :

- La paire (A, C) est observable. Ce qui implique l'existence d'un gain $K \in \mathcal{R}^{nn}$ tel que les valeurs propres de la matrice $A_0 = A - KC$ soient à valeur réelle négative.
- La fonction f est séparable, i.e. $f = f_1 + f_2$.
- La fonction f_1 est Lipschitzienne :

$$||f_1(x_1) - f_1(x_2)|| \leq l||x_1 - x_2|| \quad \forall x_1, x_2 \in \mathcal{R}^n \tag{1.7}$$

  où $l \in \mathcal{R}^+$ est une constante qui doit être déterminée.
- f_2 est une fonction incertaine, non mesurable, déterministe et bornée qui peut être modélisée comme suit :

$$f_2 = C^T \zeta(t, x_t, u_t) \leq \bar{\zeta} \tag{1.8}$$

On peut démontrer que l'observateur suivant est stable en probabilité[26] :

$$d\hat{x}_t = (A\hat{x}_t + h(\hat{x}_t)u_t + f_1(\hat{x}_t))dt + K(Ce_t)dt + S(\hat{x}_t, y_t, \phi_t, \hat{\eta}_t)dt \tag{1.9}$$

Et qu'il reconstruit l'état du système à partir des mesures en utilisant le gain adaptatif glissant donné par :

$$S = P^{-1}C^T(\sum_{i=1}^{N} \hat{\eta}_i \Gamma_i(e))\frac{\gamma Ce_t}{||\gamma Ce_t| - \phi_t|} \tag{1.10}$$

L'algorithme d'adaptation est basé sur l'espérance de l'erreur d'estimation :

$$d\hat{\eta}_i = \gamma_0 E||Ce_i||^i dt \tag{1.11}$$

Cette observateur nous donne une estimation telle que l'erreur quadratique moyenne soit exponentiellement bornée.

$$\begin{cases} \lim\limits_{t \to +\infty} Sup||e_t||^2 \leq \lambda_{min}^{-1}(P)B_1(\frac{\lambda_{min}(Q)}{\lambda_{max}(P)} - l) \\ si \qquad\qquad l\lambda_{max}(P) < \lambda_{min}(Q) \end{cases} \tag{1.12}$$

Où :

$e_t \in \mathcal{R}^n$, est l'erreur d'observation définie comme $e_t = x_t - \hat{x}_t$,

$\Gamma_i(e) : \mathcal{R}^+ \to \mathcal{R}^+$, est définie comme $\Gamma_i(e) = \frac{E||Ce||^i}{||Ce||}$,

$\phi_t : \mathcal{R}^+ \to \mathcal{R}^+$, est définie comme $\phi_t = \gamma exp(-\beta t)$,

$B_1 \in \mathcal{R}+$, est définie comme $B_1 = \beta_1^2 \lambda_{max}(P) + \beta_2 \lambda_{max}(K^T P K)$,

$\gamma, \gamma_0, \beta, \beta_1, \beta_2 \in \mathcal{R}^+$, sont des constantes de réglage,

$P = P^T, Q = Q^T \in \mathcal{R}^{nn}$, sont des matrices définies positives telles que :

$$(A - KC)^T P + P(A - KC) = -2Q \tag{1.13}$$

Filtre de Kalman Etendu (EKF)

Pour un système de la forme :

$$\begin{cases} x_k = f(x_{k-1}, u_k) + w_k \\ z_k = h(x_k) + v_k \end{cases} \tag{1.14}$$

Où :
x, est la vecteur d'état,
u, est la commande,
z, est la mesure,
w et v, sont des bruits gaussiens à moyennes nulles et de variances σ_w et σ_v respectivement.

l'estimation se fait suivant deux étapes distinctes :

Prédiction La phase de prédiction utilise l'état estimé de l'instant précédent pour produire une estimation de l'état courant.

$$\begin{cases} \hat{x}_{k|k-1} = f(\hat{x}_{k-1|k-1}, u_k) \\ P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k \end{cases} \tag{1.15}$$

Où :

$$F_k = \frac{\partial f}{\partial x}|_{\hat{x}_{k-1|k-1}, u_k} \tag{1.16}$$

Correction Dans l'étape de correction, les observations de l'instant courant sont utilisées pour corriger l'état prédit dans le but d'obtenir une estimation plus précise.

$$\begin{cases} \hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k(z_k - h(\hat{x}_{k|k-1})) \\ P_{k|k} = (I - K_k H_k) P_{k|k-1} \end{cases} \tag{1.17}$$

Où :

$$H_k = \frac{\partial h}{\partial x}|_{\hat{x}_{k|k-1}} \tag{1.18}$$

Et :

$$\begin{cases} S_k = H_k P_{k|k-1} H_k^T + R_k \\ K_k = P_{k|k-1} H_k^T S_k^{-1} \end{cases} \tag{1.19}$$

1.4.3 Estimateur de l'échelle

L'un des inconvénients majeurs des algorithmes de SLAM visuel monoculaire est que l'échelle, $\lambda \in \mathcal{R}$, de la carte obtenue ne peut être déterminée sans utiliser d'autres capteurs ou d'informations sur des objets présents dans la scène. Pour pouvoir utiliser un algorithme de SLAM visuel monoculaire dans la navigation il est nécessaire d'estimer la valeur de cette échelle.

Pour cela nous utilisons un algorithme basé sur la maximum de vraisemblance comme dans [21].

D'un côté, nous utilisons les mesures d'altitude données par le sonar que nous dérivons pour obtenir la vitesse d'ascension/descente , $\dot{z}^i_{sonar}$, et les mesures de l'accéléromètre que nous intégrons pour obtenir les vitesses horizontales, $\dot{x}^i_{accel}$ et $\dot{y}^i_{accel}$. Nous regroupons ces mesures dans le vecteur X^i, avec i qui représente le numéro de la mesure. De l'autre côté, nous utilisons les estimations de position données par ORB SLAM que nous dérivons pour obtenir les vitesses de translation, $\dot{x}^i_{orb}$, $\dot{y}^i_{orb}$ et $\dot{z}^i_{orb}$. Nous regroupons ces mesures dans le vecteur Y^i

Nous supposons que les deux mesures sont indépendantes et distribuées selon une loi normale :

$$\begin{aligned} X^i &\sim \mathcal{N}(\mu_i, \sigma_X^2) \\ Y^i &\sim \mathcal{N}(\lambda\mu_i, \sigma_{orb}^2) \end{aligned} \tag{1.20}$$

Pour estimer l'échelle λ, nous utilisons la méthode du maximum de vraisemblance. Le logarithme de la vraisemblance des paramètres μ_i et λ et donnée par :

$$ln\mathcal{L}(\mu_1, \mu_2, \ldots, \mu_N, \lambda) \propto \frac{1}{2} \sum_{i=1}^{N} \left(\frac{||X^i - \mu_i||^2}{\sigma_X^2} + \frac{||Y^i - \lambda\mu_i||^2}{\sigma_{orb}^2} \right) \tag{1.21}$$

la valeur optimale de l'échelle, $\lambda*$, et des valeurs moyennes de l'altitude, μ_i*, est obtenue lorsque la vraisemblance atteint son maximum, i.e. :

$$\begin{cases} \frac{\partial ln\mathcal{L}(\mu_1,\mu_2,\ldots,\mu_N,\lambda)}{\partial \mu_1} = 0 \\ \\ \frac{\partial ln\mathcal{L}(\mu_1,\mu_2,\ldots,\mu_N,\lambda)}{\partial \mu_2} = 0 \\ \\ \vdots \\ \frac{\partial ln\mathcal{L}(\mu_1,\mu_2,\ldots,\mu_N,\lambda)}{\partial \mu_N} = 0 \\ \\ \frac{\partial ln\mathcal{L}(\mu_1,\mu_2,\ldots,\mu_N,\lambda)}{\partial \lambda} = 0 \end{cases} \tag{1.22}$$

En prenant la dérivée par rapport à μ_i, on peut calculer la valeur optimale de cette dernière en fonction de λ :

$$\begin{gathered} \frac{\partial ln\mathcal{L}(\mu_1,\mu_2,\ldots,\mu_N,\lambda)}{\partial \mu_1} \propto \frac{\mu_i - X^i}{\sigma_X^2} + \frac{\lambda^2\mu_i - \lambda Y^i}{\sigma_{orb}^2} = 0 \\ \\ \implies \mu_i = \frac{\lambda\sigma_X^2 Y^i + \sigma_{orb}^2 X^i}{\lambda^2\sigma_X^2 + \sigma_{orb}^2} \end{gathered} \tag{1.23}$$

En prenant cette fois-ci la dérivée par rapport à λ et en remplaçant les μ_i par leurs expressions nous obtenons :

$$\frac{\partial ln\mathcal{L}(\mu_1, \mu_2, \ldots, \mu_N, \lambda)}{\partial \lambda} \propto \sum_{i=1}^{n} \frac{(\lambda X^i - Y^i)^T(\sigma_{orb}^2 X^i + \lambda\sigma_X^2 Y^i)}{(\lambda^2\sigma_X^2 + \sigma_{orb}^2)^2} = 0 \tag{1.24}$$

Ce qui nous donne, après la résolution de l'équation, la valeur optimale de l'échelle, λ^* :

$$\lambda^* = \frac{s_{xx} - s_{yy} + \sqrt{(s_{xx} - s_{yy}^2 + 4s_{xy}^2)}}{2\sigma_{orb}^{-1}\sigma_X s_{xy}} \tag{1.25}$$

Avec :
$s_{xx} := \sigma_X^2 \sum_{i=1}^{N} (Y^i)^2$
$s_{yy} := \sigma_{orb}^2 \sum_{i=1}^{N} (X^i)^2$
$s_{xy} = \sigma_X \sigma_{orb} \sum_{i=1}^{N} X^i \cdot Y^i$

1.5 Conclusion

Dans ce chapitre, nous avons défini ce qu'est le SLAM et en particulier le SLAM visuel monoculaire. Puis, nous avons présenté l'état de l'art de ce dernier, les inconvénients des approches utilisées et quelques uns des algorithmes de fusion de données utilisés pour les combler. Enfin, nous avons expliqué les deux approches que nous avons proposé pour résoudre le problème du SLAM visuel monoculaire et que nous allons implémenter et simuler au chapitre suivant.

Chapitre 2

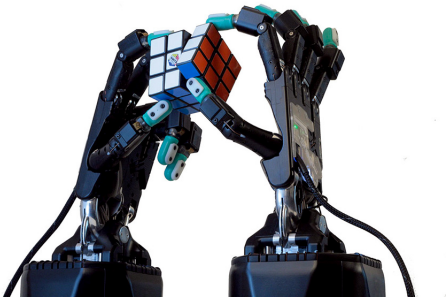
Simulation

2.1 Introduction

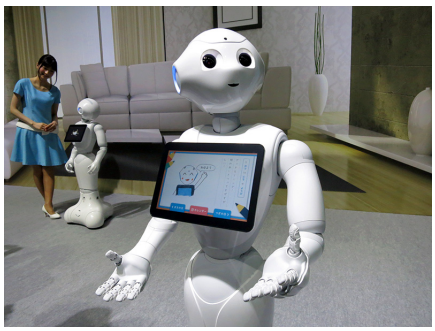
Dans ce chapitre, nous allons implémenter les deux approches proposées au chapitre précédent sous ROS. Puis, nous allons les simuler sous Gazebo pour un quadrirotor. Enfin, nous finirons par une étude comparative entre les deux approches.

2.2 ROS

ROS[27] (Robot Operating System) est un meta-système d'exploitation, quelque chose entre le système d'exploitation et le middleware, pour robots né d'une collaboration entre le milieu industriel et celui de la recherche.



(a) Shadow Hand, main robotique humanoide de la société The Shadow Robot Company



(b) Pepper, robot humanoïde développé par la société Aldebaran

(c) AscTec Hummingbird, quadrirotor de la société Ascending Technologies

FIGURE 2.1: Exemples de robots utilisant ROS

Il fournit des services proches d'un système d'exploitation (abstraction du matériel, gestion de la concurrence, des processus...) mais aussi des fonctionnalités de haut niveau (appels asynchrones, appels synchrones, base centralisée de données, système de paramétrage du robot...).

Il est composé de bibliothèques réutilisables qui sont conçus pour fonctionner de façon indépendante. Les bibliothèques sont enveloppés d'une couche de passage de messages mince qui leur permet d'être utilisés par et d'utiliser d'autres nœuds de ROS. Les messages sont transmis entre pairs et ne sont pas basées sur un langage de programmation spécifique. Ils peuvent être écrits en C++, Python, C, LISP, Octave ou tout autre langue pour lequel un wrapper ROS est disponible.

ROS peut être séparé en trois groupes :

- les outils utilisés pour créer, lancer et distribuer des logiciels basés sur ROS (roscore, roslaunch, catkin)
- les clients ROS pour des langages : roscpp (C++) et rospy (python)
- les packages contenant des programmes pour ROS utilisant un ou plusieurs clients ROS

Les outils et les principaux clients ROS (roscpp et rospy) sont publiés sous les termes de la licence BSD[28]. De nombreux packages sont publiés pour ROS avec diverses licences open source (BSD[28], MIT[29]). Ces packages permettent de lancer des applications, des algorithmes ou encore des programmes pour interfacer ROS avec des robots. L'outil de simulation Gazebo[30] est directement intégré à ROS.

2.2.1 Concepts fondamentaux

Les concepts fondamentaux de ROS sont : le master, les nœuds, les topics et les messages.

Master

Le Master est un service de déclaration et d'enregistrement des nœuds qui permet ainsi à des nœuds de se connaitre et d'échanger de l'information, figure 2.2.

Le Master comprend une sous-partie très utilisée qui est le serveur de paramètres. Celui-ci, comme son nom l'indique, est une sorte de base de données centralisée dans laquelle les nœuds peuvent stocker de l'information et ainsi partager des paramètres globaux.

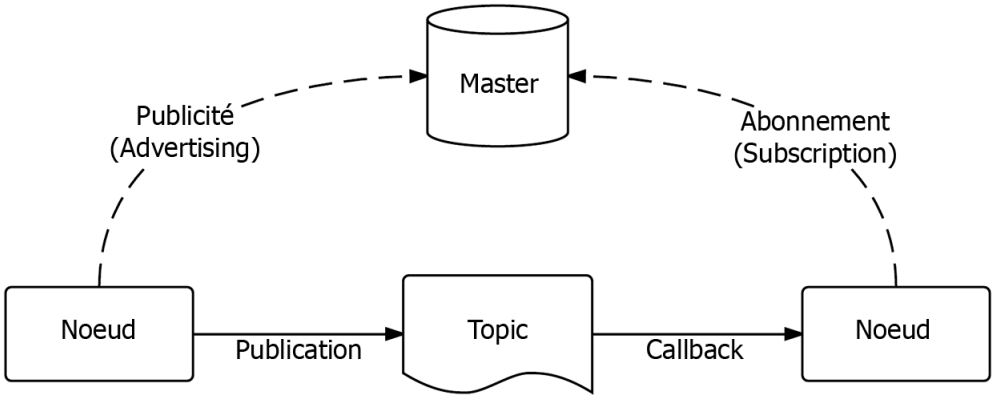


FIGURE 2.2: Schéma de fonctionnement du master

Nœuds

Dans ROS, un nœud est une instance d'un exécutable. Un nœud peut correspondre à un capteur, un moteur, un algorithme de traitement, de surveillance, etc. Chaque nœud qui se lance se déclare au Master. On retrouve ici l'architecture microkernel où chaque ressource est un nœud indépendant.

Topics

Un topic est un système de transport d'information basé sur le principe de l'abonnement/publication (subscribe / publish). Un ou plusieurs nœuds pourront publier de l'information sur un topic, sous forme de message, et un ou plusieurs nœuds pourront lire cette information sur ce topic. Le topic est en quelque sorte un bus d'information asynchrone, un peu comme un flux RSS. Cette notion de bus many-to-many asynchrone est essentielle dans le cas d'un système distribué.

Le topic est typé, c'est-à-dire que le type d'information qui est publiée est toujours structuré de la même manière.

Messages

Un message est une structure de donnée composite. Un message est composé d'une combinaison de types primitifs (chaines de caractères, booléens, entiers, flottants, etc) et de messages (le message est une structure récursive). Par exemple un nœud représentant un servomoteur d'un robot, publiera certainement son état sur un topic (selon ce que vous aurez programmé) avec un message contenant par exemple un entier représentant la position du moteur, un flottant représentant sa température, un autre flottant représentant sa vitesse ...

2.2.2 Gazebo

Gazebo[30] est un simulateur physique multi-robots pour les environnements en 3D. Il est capable de simuler une population de robots, de capteurs et d'objets de manière précise et efficace dans des environnements complexes d'intérieur et d'extérieur. Il génère des signaux réalistes et des interactions physiques plausibles entre les différents objets.

Parmi ces caractéristiques clés on trouve :

- La présence de plusieurs moteurs physiques
- Un large choix de modèles de robots et d'environnement, avec la possibilité de créer ses propres modèles.
- Un large éventail de capteurs
- La présence d'interfaces de programmatiques et graphiques pratiques

2.3 Implémentation de l'approche de SLAM

Dans cette section nous allons nous intéresser à l'implémentation des approches de SLAM décrites au chapitre précédent.

Pour cela, nous utilisons l'outil de simulation Gazebo, car il offre beaucoup d'avantages par rapport à Matlab. Il nous permet de faire des expériences et des essais physiques réalistes facilement et rapidement et avec de bon graphismes, contrairement à Matlab.

Pour le modèle du quadrirotor, nous utilisons le package "Hector Quadrotor"[32]. Ce dernier contient un modèle de quadrirotor, figure 2.4, équipé de différents capteurs tels qu'un caméra monoculaire, un télémètre laser, une centrale inertielle, un sonar et un

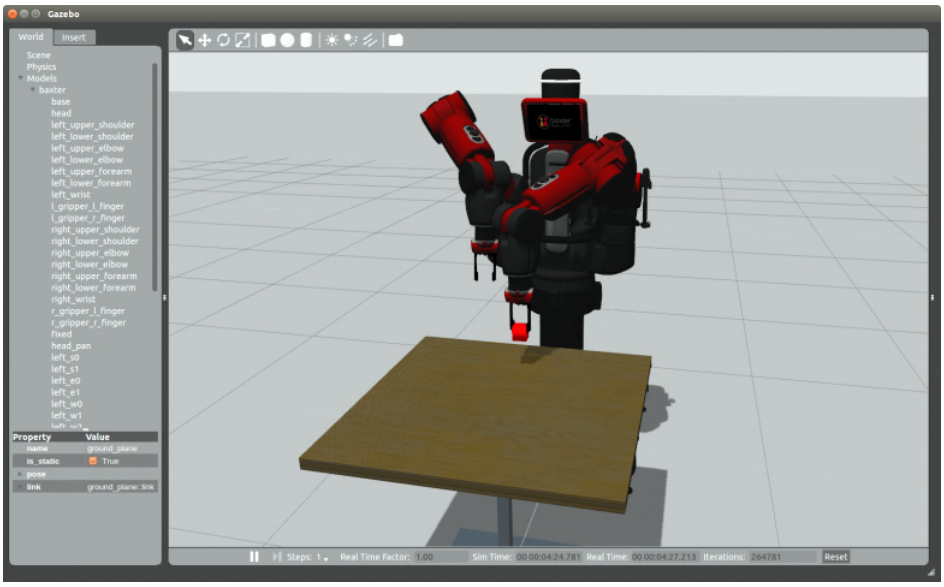


FIGURE 2.3: Exemple de simulation du robot Baxter[31] sous Gazebo

altimètre. Ce package contient aussi des interfaces de commandes pour interagir avec le modèle du quadrirotor.

Pour commander ce dernier, nous utilisons l'interface de commandes en vitesse qui s'abonne sur le topic "/command/twist" qui contient des références de vitesses de translation et de rotation à suivre, car on n'a pas pu trouver certains paramètres nécessaires pour le commander directement en utilisant les signaux PWM des 4 rotors. Nous avons implémenter dans un nœud sous ROS nommé "sliding_mode_velocity_controller" une commande par mode glissant pour asservir la position et l'orientation du quadrirotor en utilisant l'interface de commande en vitesse. Ce nœud s'abonne sur le topic "/reference" pour obtenir le point de référence à atteindre. Il calcul la commande par mode de glissement et la publie sur le topic "/command/twist" comme on peut le voir dans la figure 2.5.

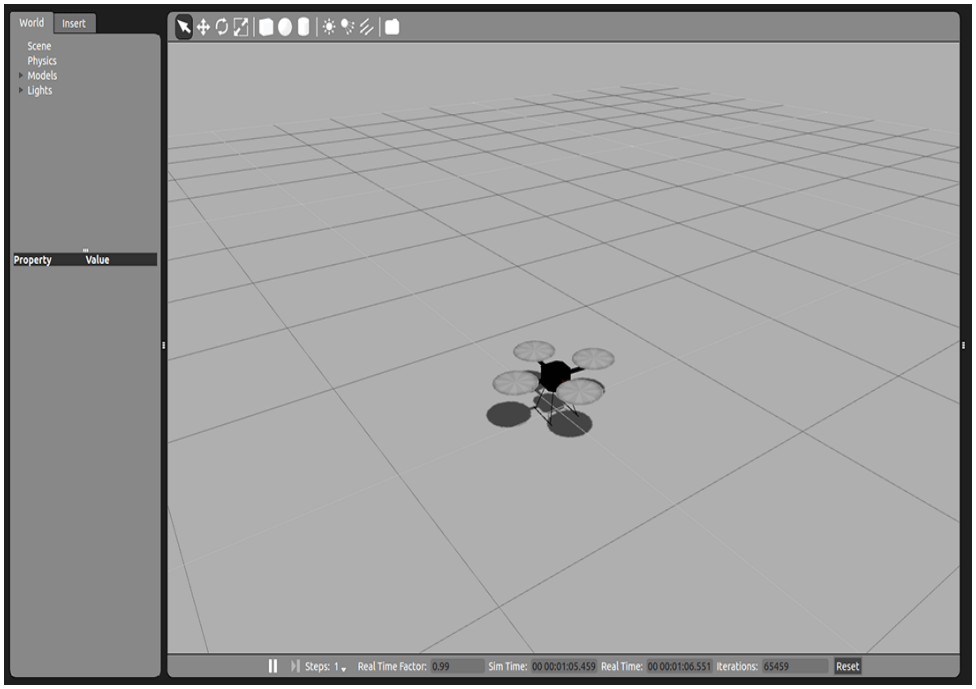


FIGURE 2.4: Modèle du quadrirotor du package "Hector Quadrotor" sous Gazebo

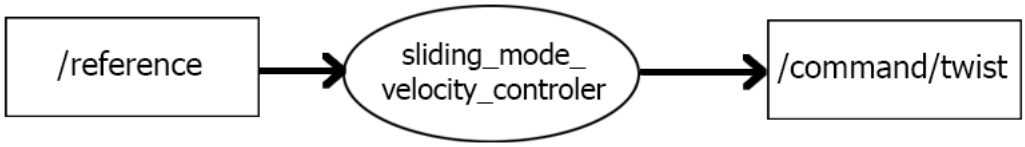


FIGURE 2.5: Schéma de la commande

Nous créons un environnement en 3D sous Gazebo pour pouvoir y faire évoluer le quadrirotor, figure 2.6.

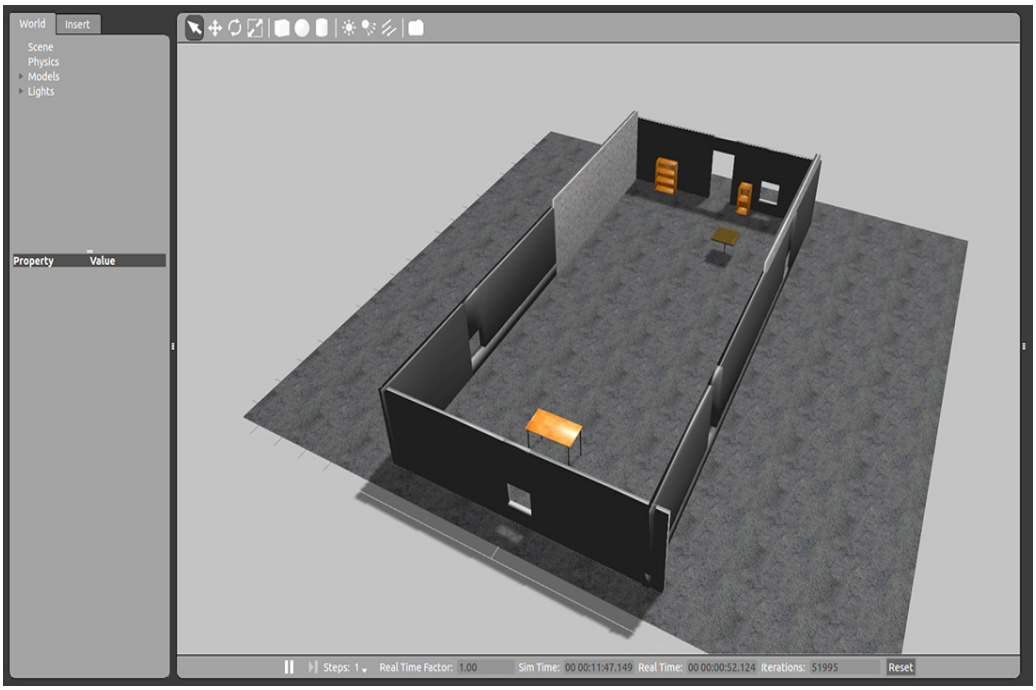


FIGURE 2.6: Environnement 3D de la simulation sous Gazebo

Nous utilisons le nœud open-source de ORB-SLAM nommé "orb_slam", qui va traiter les images provenant de la caméra attachée au quadrirotor, présentes sur le topic "/front_cam/camera/image", et qui va détecter les points d'intérêts, figure 2.7 et publier ensuite la position de ces derniers ainsi que celle de la caméra et son orientation définies par rapport à un repère dont l'origine est fixée au point où se trouvait la caméra lorsque l'algorithme s'est initialisé, c'est-à-dire, lorsqu'il a pris deux images clés assez distantes l'une de l'autre pour pouvoir exécuter l'initialisation par la méthode des 5 points[9].

Nous créons enfin un nœud qui implémente l'algorithme d'estimation, nommé "ekf_slam" dans le cas du EKF et "sasmo_slam" dans le cas de l'observateur glissant SASMO, qui reçoit les estimations de la position et de l'attitude donnée par ORB-SLAM, les mesures de l'accélération linéaire et de la vitesse angulaire provenant de la centrale inertielle, la mesure de l'altitude provenant du sonar et les mesures des commandes et qui nous donne l'estimation finale de la position et de l'attitude du quadrirotor, comme on peut le voir sur la figure 2.8.

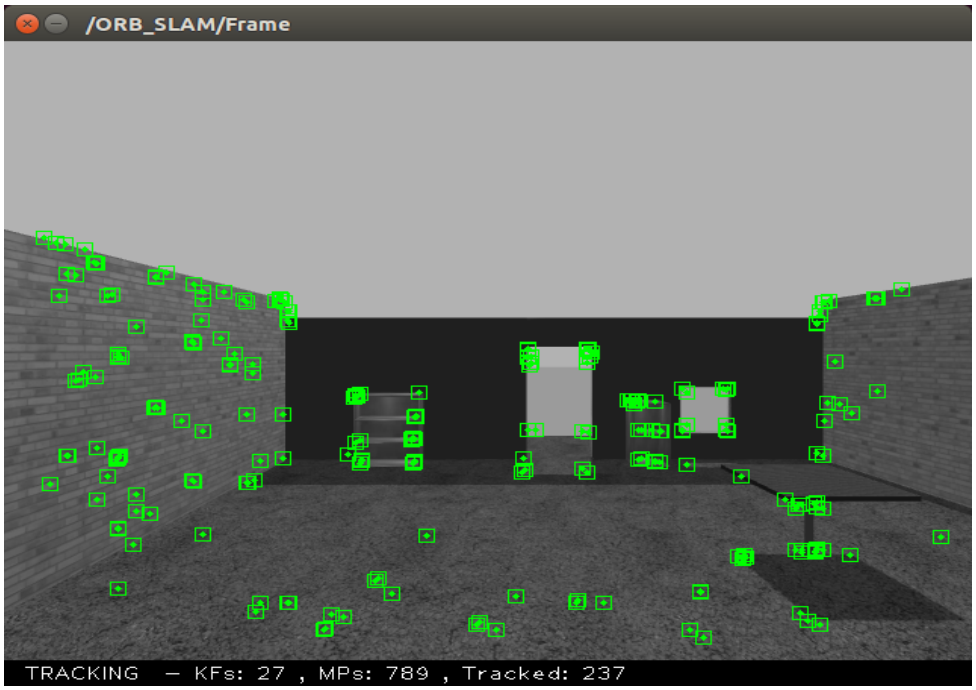


FIGURE 2.7: Détection des points d'intérêts par orb_slam

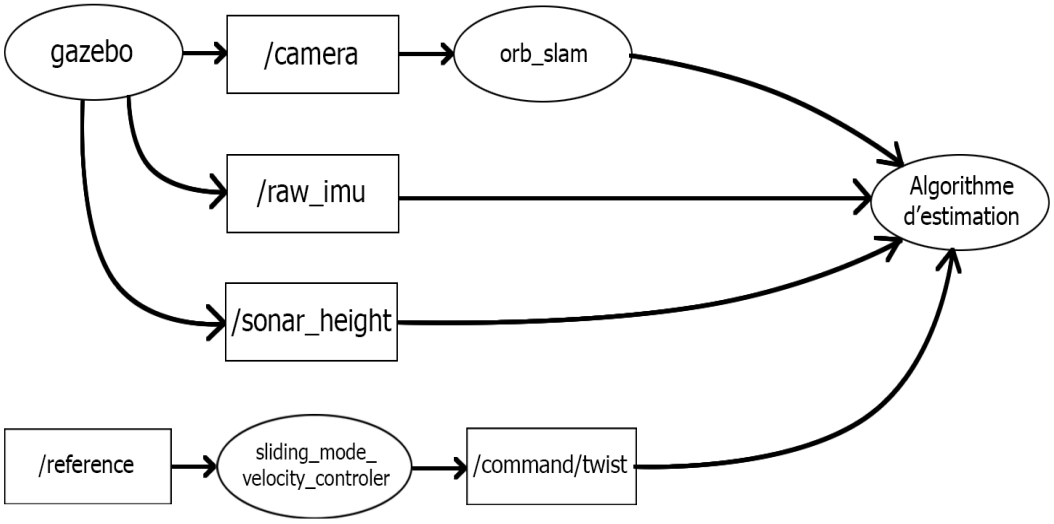


FIGURE 2.8: Les nœuds et les topics utilisés pour réaliser l'approche de SLAM proposée

Pour la simulation, nous procédons de la manière suivante :

- Nous commençons par lancer Gazebo ainsi que tout les nœuds décrits précédemment.
- Ensuite, nous attendons que l'algorithme ORB-SLAM se lance, cela prend en moyenne 12 secondes.
- Après cela, nous faisons suivre au quadrirotor des mouvements de translation pour que ORB-SLAM puisse initialiser la position et l'orientation de la caméra, ainsi que la carte et les premiers points d'intérêt.
- Ensuite, nous faisons suivre au quadrirotor d'autres mouvements de translation pour pouvoir faire l'estimation de l'échelle, car elle dépend des mouvements du quadrirotor.

– Enfin, après la convergence de l'échelle, nous donnons au quadrirotor comme référence les sommets d'un carré centré à l'origine et de longueur de côté égale à 2.

2.3.1 Approche SASMO

Nous pouvons voir les résultats de la simulation sur les figures 2.9, 2.10, 2.11, 2.12, 2.13 et 2.14.

Nous constatons qu'entre le début de la simulation et l'initialisation d'ORB-SLAM à l'instant $t = 18s$, l'estimation de la position suivant X (respectivement Y), augmente (respectivement diminue) malgré que la position du quadrirotor est fixe. Cela est dû aux bruits qui entache les mesures provenant de l'accéléromètre de la centrale inertielle.

Nous remarquons aussi qu'à l'instant où ORB-SLAM s'initialise, les estimations données par l'observateur glissant divergent, puis convergent de nouveaux après quelques instants. Cela peut s'expliquer par le fait que l'initialisation de l'erreur d'estimation à l'instant d'initialisation d'ORB-SLAM introduit un grand gain de glissement qui diminue rapidement dés que l'erreur se stabilise.

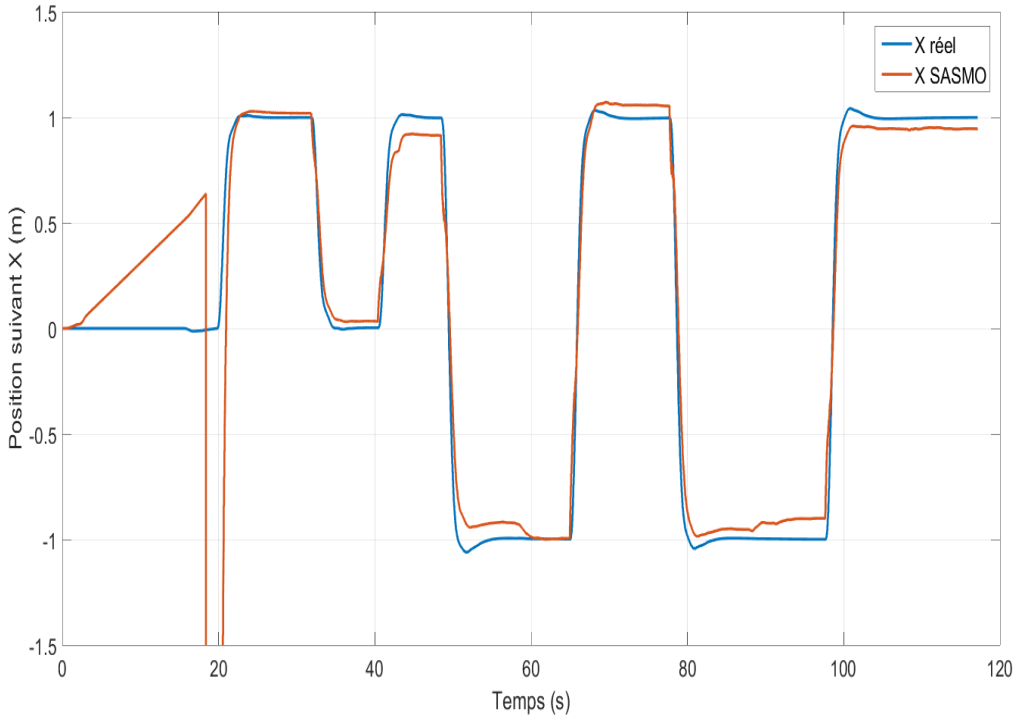


FIGURE 2.9: Evolution de la position suivant X du quadrirotor en fonction du temps

Après que l'échelle se soit stabilisée, nous constatons que les estimations des positions suivent bien l'allure des position réelles. Il persiste néanmoins une erreur statique qui est due au fait que l'estimation de l'échelle n'ait pas convergé vers la valeur réelle. Pour ce qui est de l'attitude, les estimations des angles de Roulis et de Tangages sont assez proches de angles réels. Cependant, pour l'estimation de l'angle de Lacet, nous remarquons qu'il persiste de faibles erreurs qui sont dues aux bruits des mesures.

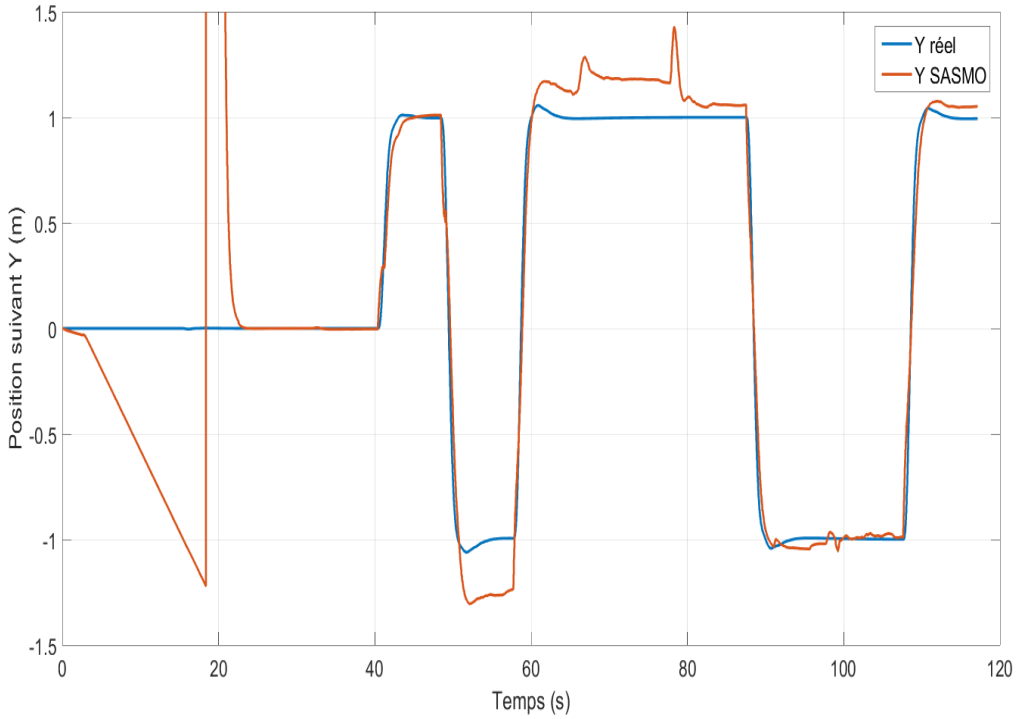


FIGURE 2.10: Evolution de la position suivant Y du quadrirotor en fonction du temps

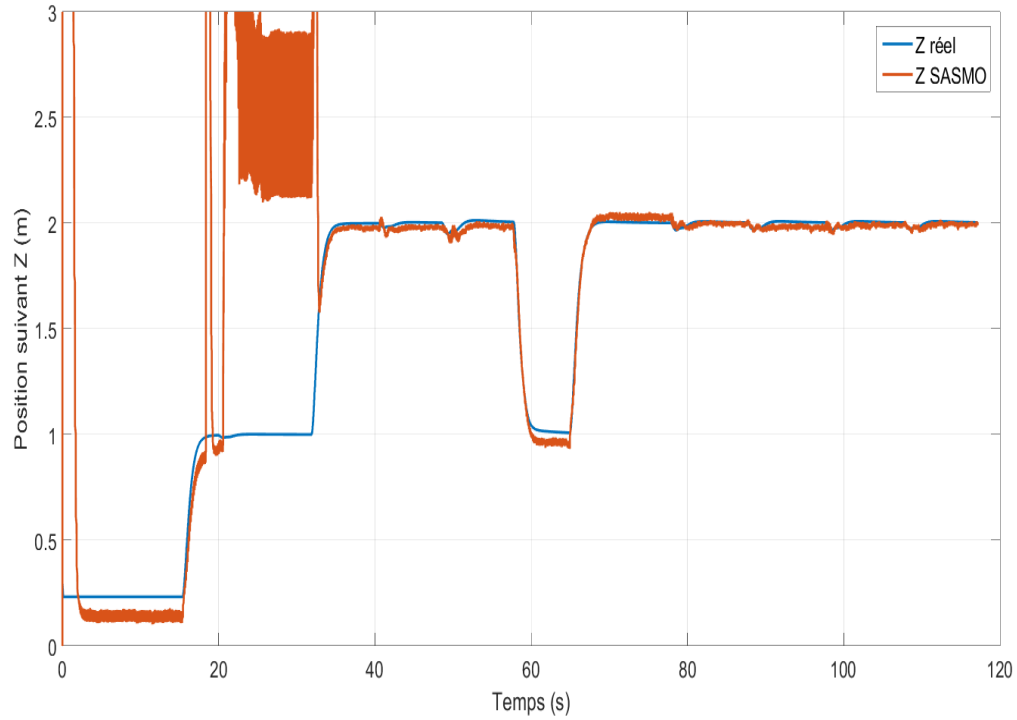


FIGURE 2.11: Evolution de la position suivant Z du quadrirotor en fonction du temps

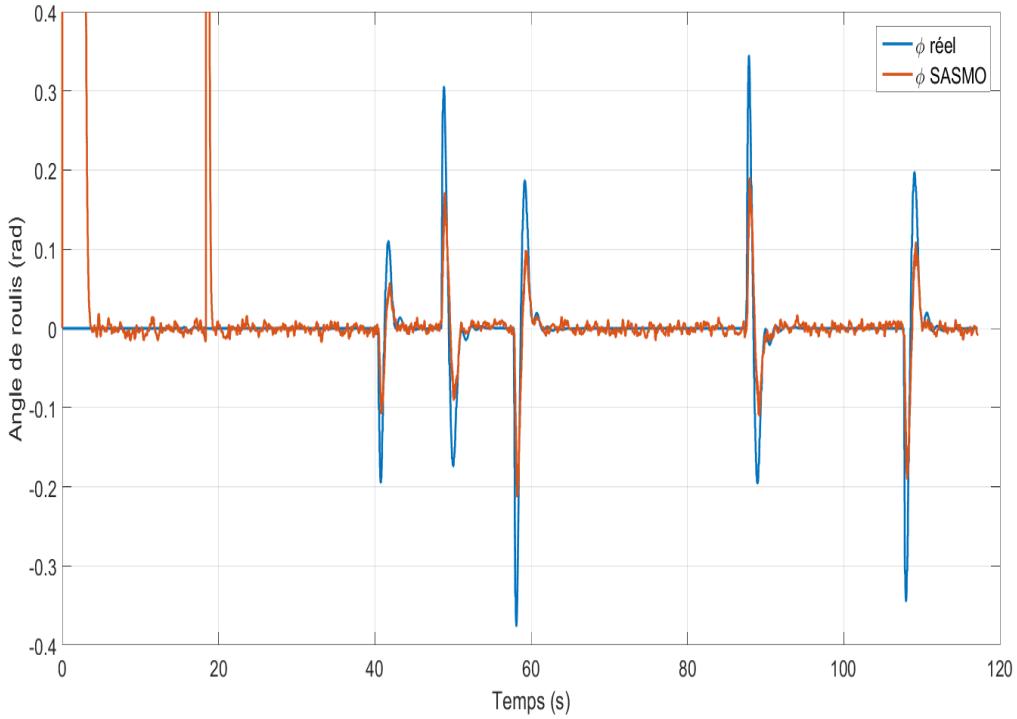


FIGURE 2.12: Evolution de l'angle de Roulis du quadrirotor en fonction du temps

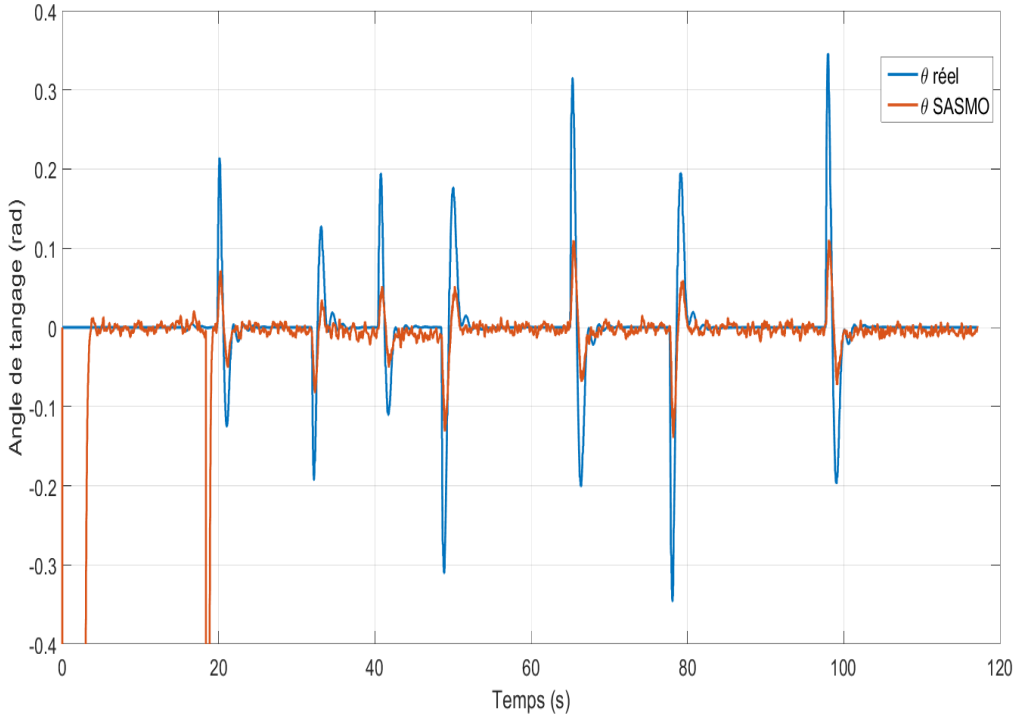


FIGURE 2.13: Evolution de l'angle de Tangage du quadrirotor en fonction du temps

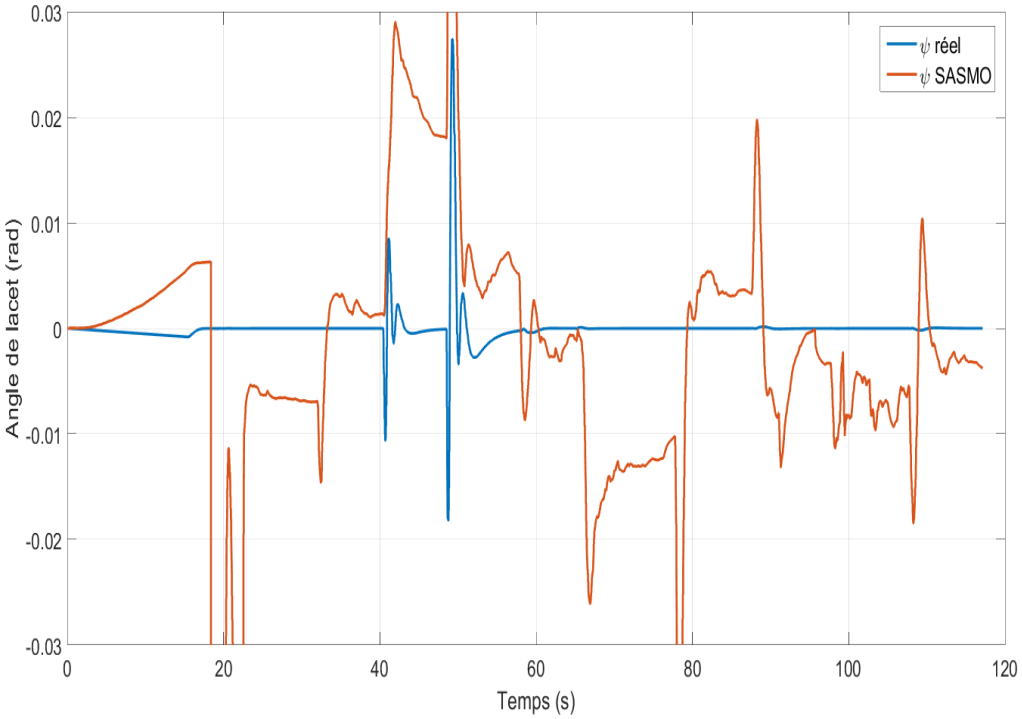


FIGURE 2.14: Evolution de l'angle de Lacet du quadrirotor en fonction du temps

2.3.2 Approche EKF

Nous pouvons voir les résultats de la simulation sur les figures 2.15, 2.16, 2.17, 2.18, 2.19 et 2.20.

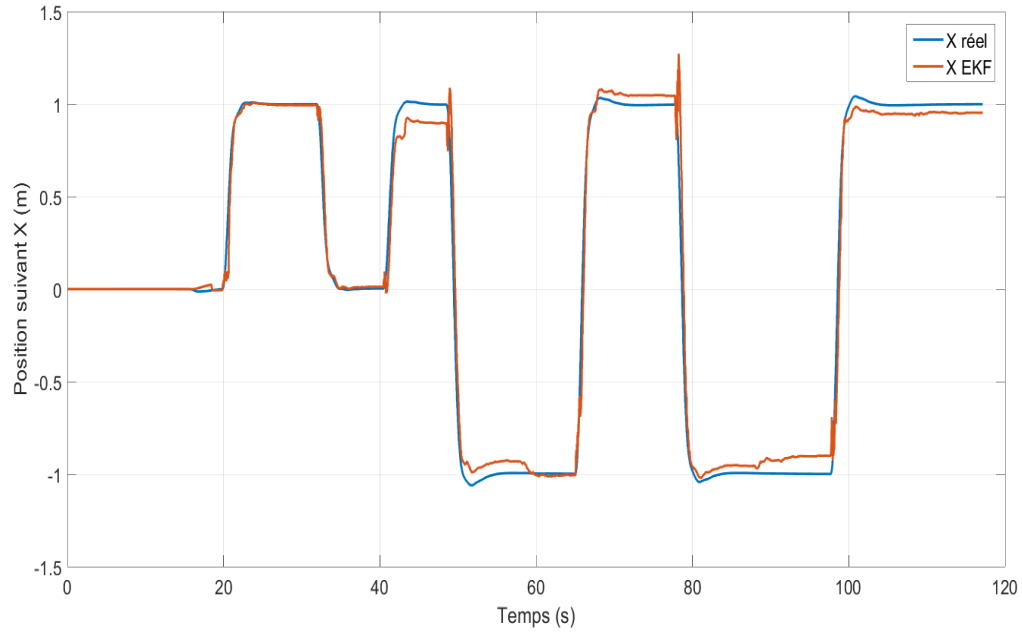


FIGURE 2.15: Evolution de la position en X du quadrirotor en fonction du temps

Nous constatons qu'entre le début de la simulation et le lancement d'ORB-SLAM,

l'estimation de la position suivant X et Y restent nulles, contrairement à l'approche précédente.

Nous remarquons aussi qu'à l'instant où ORB-SLAM s'initialise, les estimations données par l'EKF ne divergent pas, contrairement à l'approche précédente.

Après que l'échelle se soit stabilisée, nous constatons que les estimations des positions suivent bien l'allure des position réelles. Il persiste néanmoins une erreur statique qui est due au fait que l'estimation de l'échelle n'ait pas convergé vers la valeur réelle.

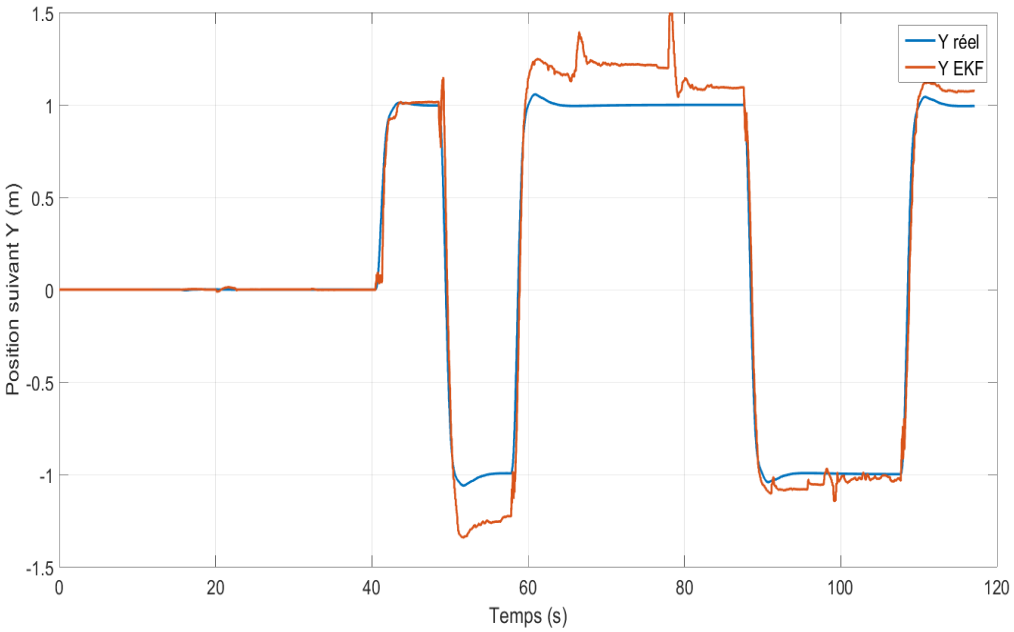


FIGURE 2.16: Evolution de la position en Y du quadrirotor en fonction du temps

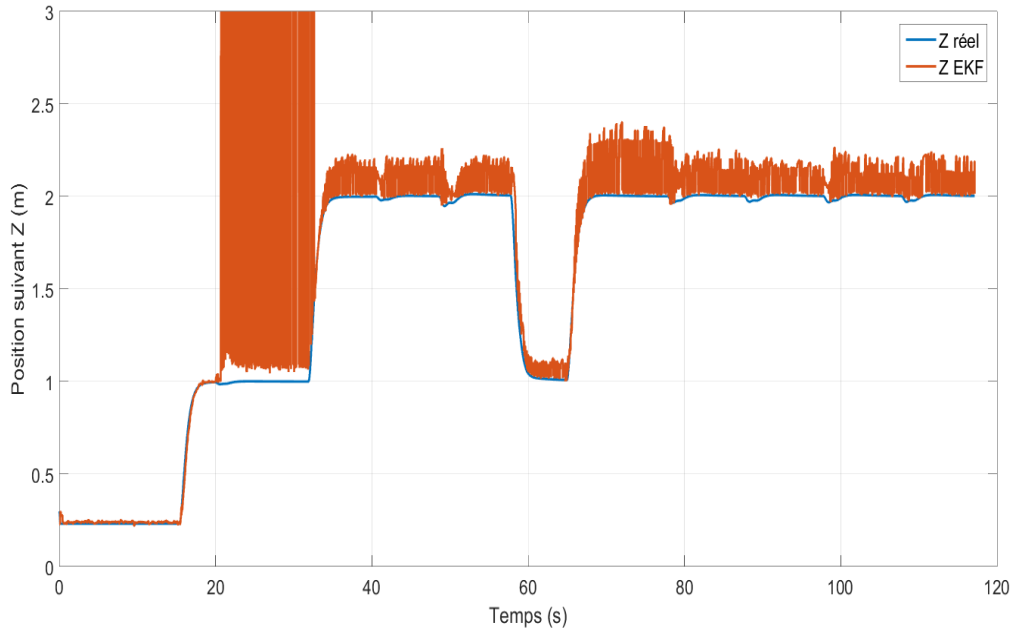


FIGURE 2.17: Evolution de la position en Z du quadrirotor en fonction du temps

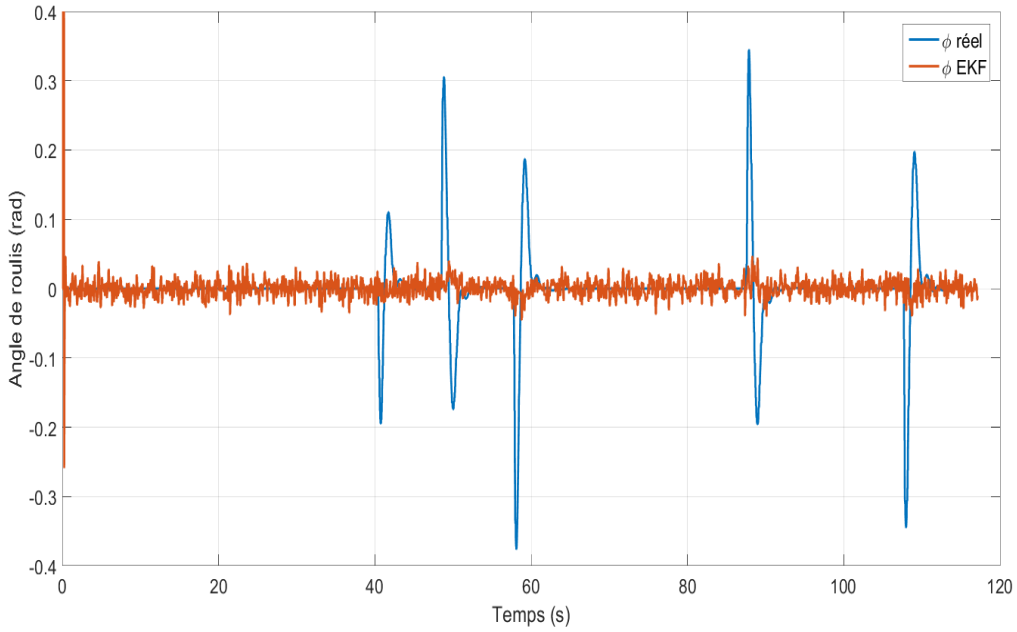


FIGURE 2.18: Evolution de l'angle de Roulis du quadrirotor en fonction du temps

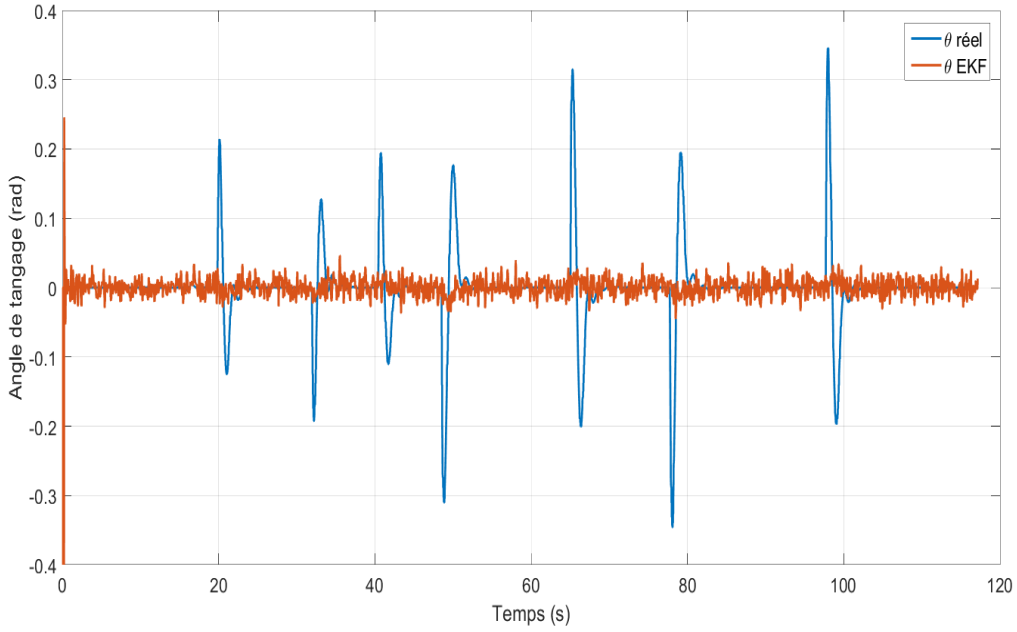


FIGURE 2.19: Evolution de l'angle de Tangage du quadrirotor en fonction du temps

Nous constatons aussi que l'estimation de l'altitude est entachée de beaucoup d'oscillations, contrairement à l'approche précédente, et cela à cause de la différence entre la mesure donnée par ORB-SLAM avec correction de l'échelle et celle donnée par le sonar.

Pour ce qui est de l'attitude, les estimations des angles de Roulis et de Tangages sont très bruitées et ne suivent pas l'allure des angles réels, contrairement à l'approche précédente. Cependant, pour l'estimation de l'angle de Lacet, nous remarquons que cette dernière suit bien l'allure de la valeur réelle, mais avec une faible erreur statique.

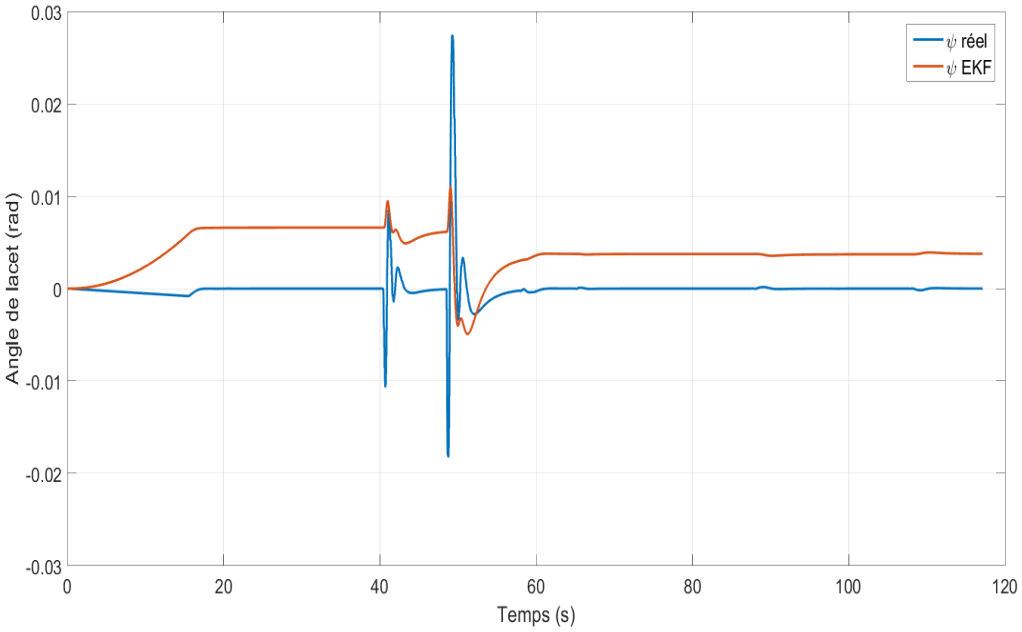


FIGURE 2.20: Evolution de l'angle de Lacet du quadrirotor en fonction du temps

2.3.3 L'échelle

Nous constatons que la convergence de l'échelle est lente, car l'estimateur utilisé à besoin d'avoir assez de données pour converger.

Nous pouvons voir qu'à partir de l'instant $t = 60s$, l'échelle se stabilise pour les trois axes.

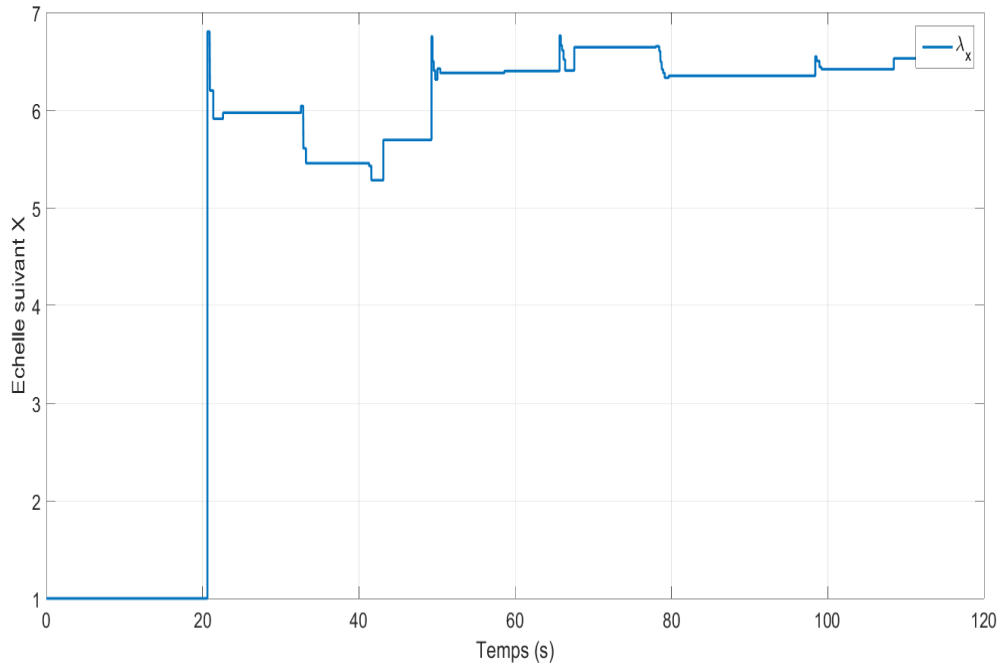


FIGURE 2.21: Évolution de l'estimation de l'échelle suivant X

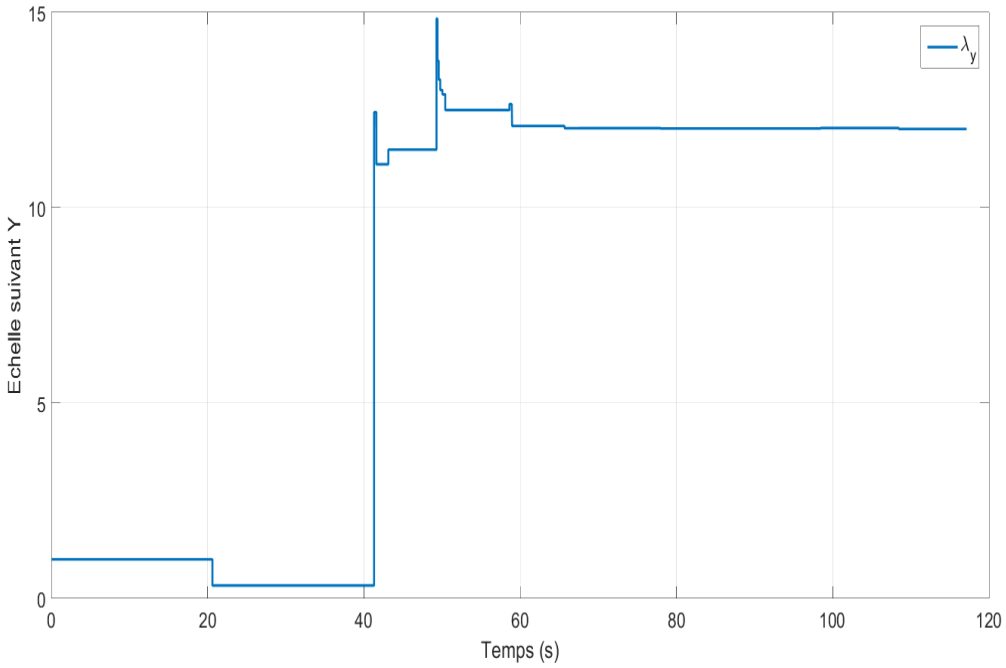


FIGURE 2.22: Évolution de l'estimation de l'échelle suivant Y

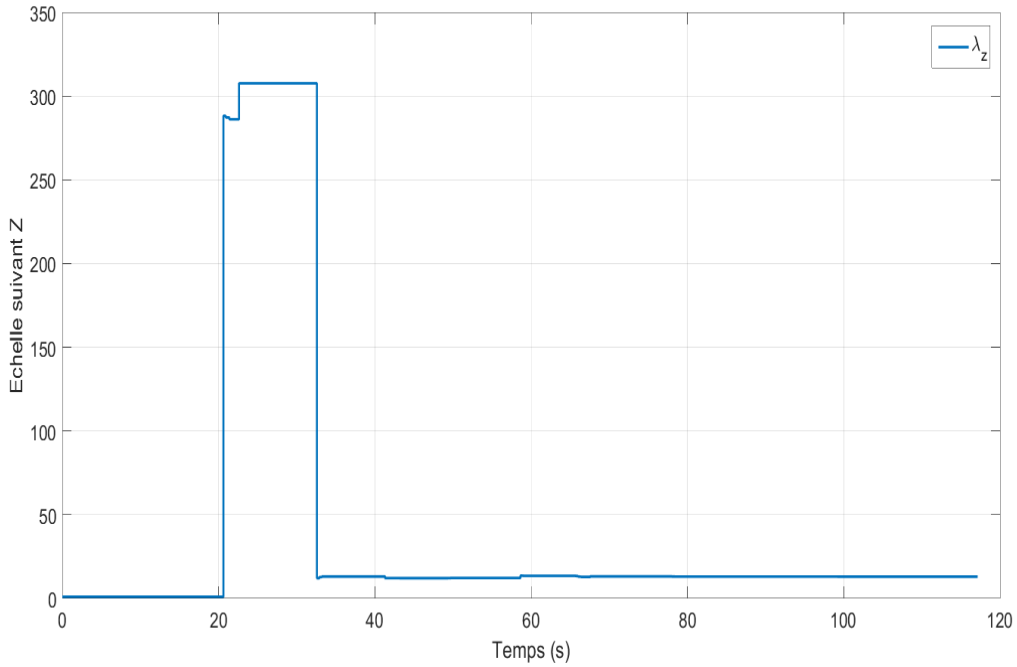


FIGURE 2.23: Évolution de l'estimation de l'échelle suivant Z

2.3.4 Carte de l'environnement

La carte donnée par notre approche est indépendante de l'algorithme d'estimation utilisé. Elle ne dépend que des estimations d'ORB-SLAM et de l'estimation de l'échelle. Comme nous pouvons le voir, figures 2.24, 2.25 et 2.26, la carte est assez fidèle à l'environnement 3D crée et utilisé pour notre simulation.

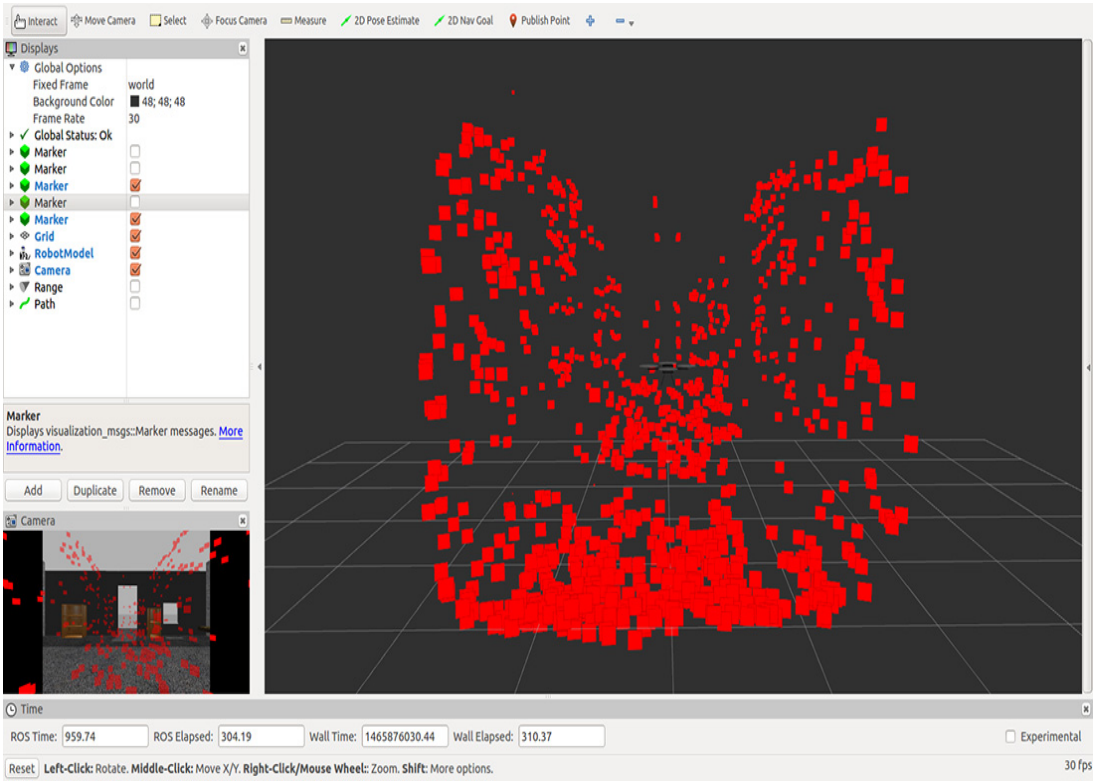


FIGURE 2.24: Vue de dos de la carte des amères générée par notre approche

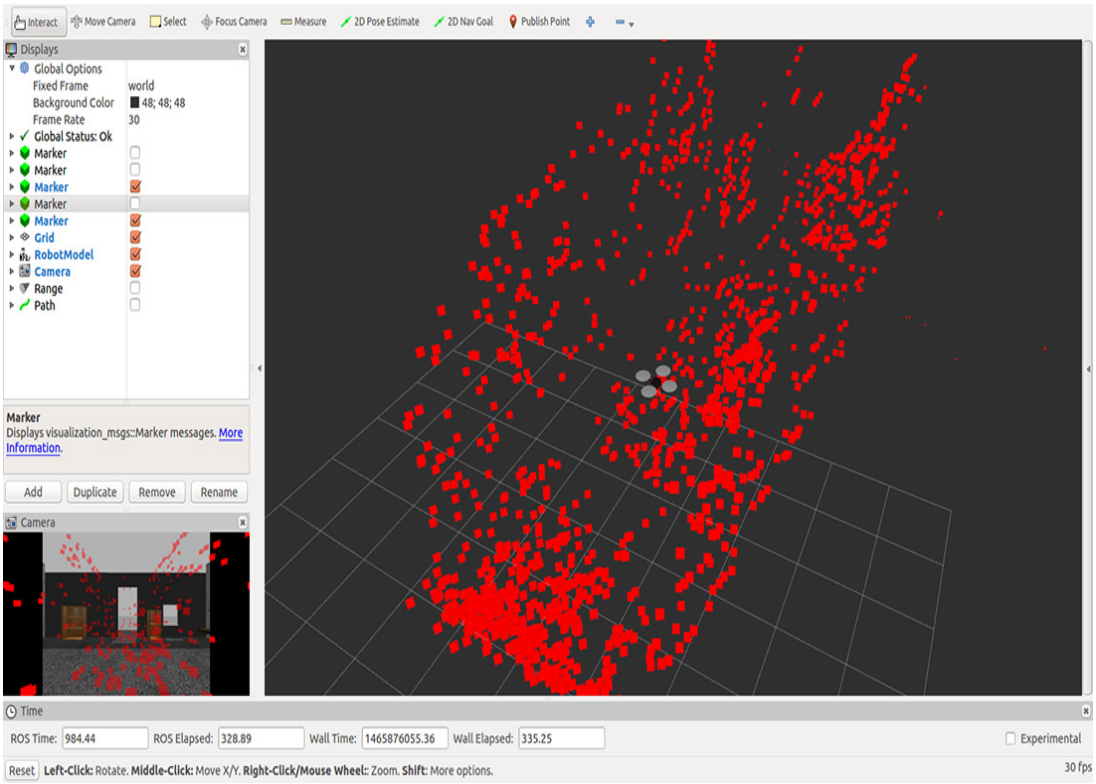


FIGURE 2.25: Vue en perspective de la carte des amères générée par notre approche

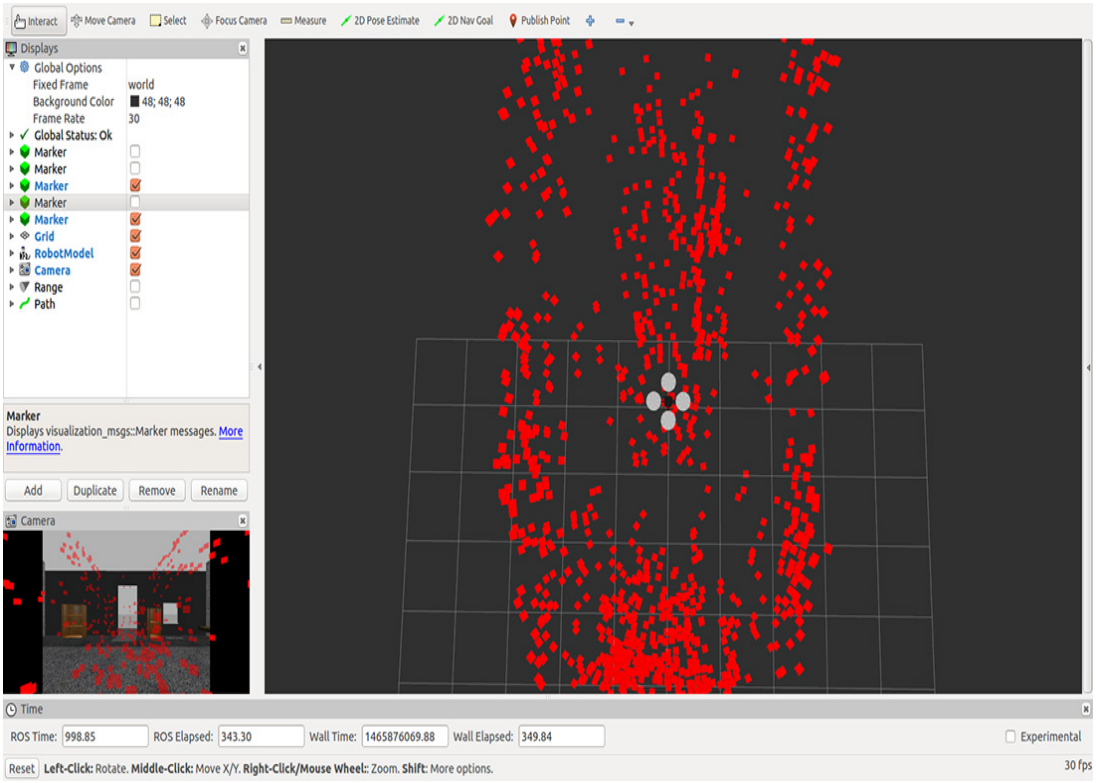


FIGURE 2.26: Vue de dessus de la carte des amères générée par notre approche

2.3.5 Comparaison entre les deux approches

Nous allons nous intéresser dans cette partie à la comparaison quantitative entre les deux approches.

Nous ne nous intéresserons qu'aux estimations faite à partir de l'instant où l'échelle s'est stabilisée, i.e. à partir de $t = 60s$.

Pour cela, nous allons utiliser quelques indicateurs d'écart.

Erreur Quadratique Moyenne (RMSE)

L'erreur quadratique moyennes, ou RMSE (Root Mean Squared Error), est défini comme suit :

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}{N}} \tag{2.1}$$

Où :

$\hat{y}$, est la valeur estimée,

y, est la valeur réelle,

N, le nombre de mesures.

Elle est très utile pour comparer plusieurs estimateurs, notamment lorsque l'un d'eux est biaisé.

Les résultats obtenus pour chaque variable sont résumés dans le tableau 2.1 et nous pouvons les visualiser sur la figure 2.27.

Variable	SASMO	EKF
X	0.0926	0.0971
Y	0.1276	0.1533
Z	0.0240	0.1577
ϕ	0.0299	0.0515
θ	0.0441	0.0619
ψ	0.0099	0.0037

TABLE 2.1: RMSE des estimations des deux approches

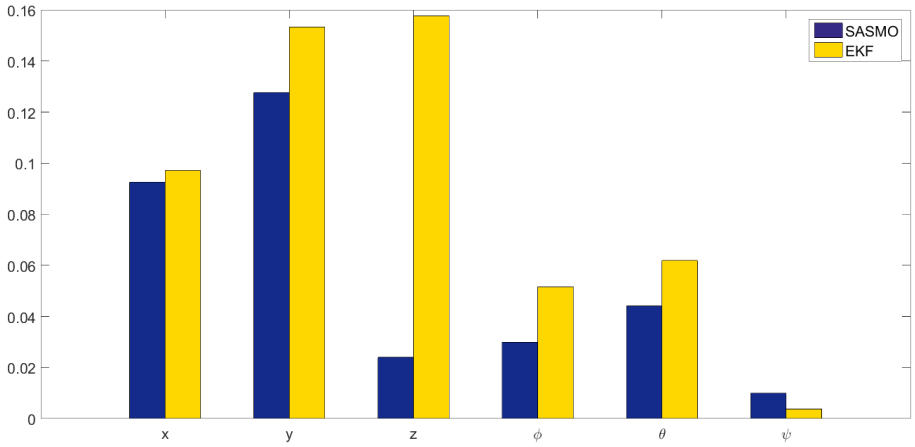


FIGURE 2.27: Diagramme en barre des erreurs quadratique moyennes

Nous remarquons que les erreurs quadratiques moyennes des estimations de SASMO est plus petite que celles des estimations de l'EKF, mis à part l'erreur quadratique moyenne de l'estimation de l'angle de lacet *psi*.

Nous remarquons aussi que pour l'altitude, l'erreur quadratique moyenne de l'estimation de SASMO est beaucoup plus petite que celle de l'EKF.

Tout cela indique que SASMO présente de meilleures estimations que celles de l'EKF.

Erreur Absolue Moyenne (MAE)

L'erreur absolue moyenne, ou MAE (Mean Absolute Error), est défini comme suit :

$$MAE = \frac{\sum_{i=1}^{N} |\hat{y}_i - y_i|}{N} \tag{2.2}$$

Où :

$\hat{y}$, est la valeur estimée,

y, est la valeur réelle,

N, le nombre de mesures.

Les résultats obtenus pour chaque variable sont résumés dans le tableau 2.2 et nous pouvons les visualiser sur les figures 2.28 et 2.29.

Variable	SASMO	EKF
X	$1.6207 \cdot 10^{-05}$	0.0638
Y	$2.2319 \cdot 10^{-05}$	$2.6820 \cdot 10^{-05}$
Z	$0.4207e \cdot 10^{-05}$	$2.7587 \cdot 10^{-05}$
ϕ	$0.5230 \cdot 10^{-05}$	$0.9012 \cdot 10^{-05}$
θ	$0.7713 \cdot 10^{-05}$	$1.0838 \cdot 10^{-05}$
ψ	$0.1731 \cdot 10^{-05}$	$0.0653 \cdot 10^{-05}$

TABLE 2.2: MAE des estimations des deux approches

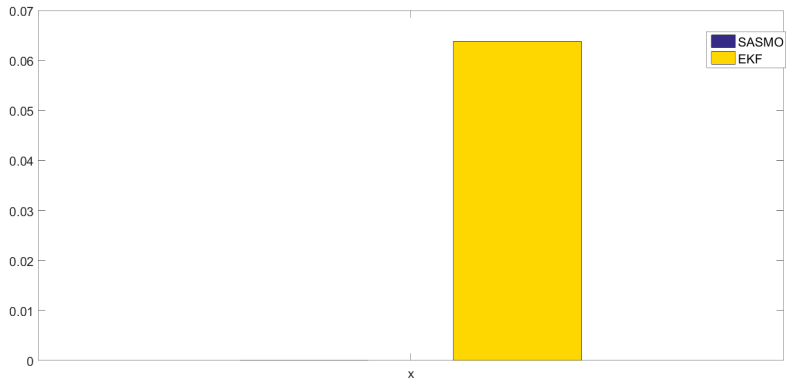


FIGURE 2.28: Diagramme en barre de l'erreur absolue moyenne suivant X

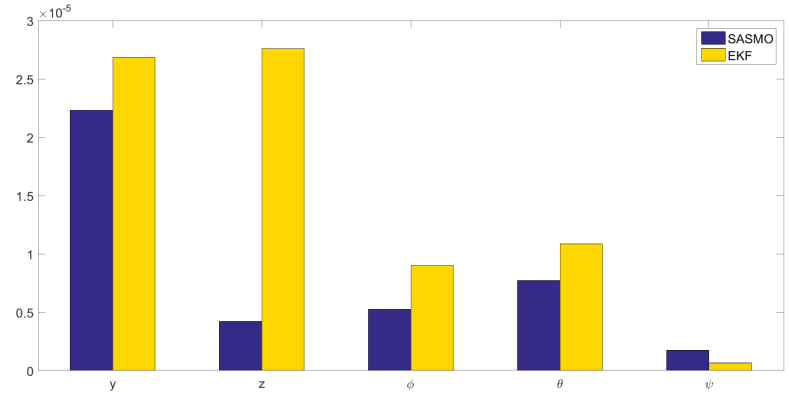


FIGURE 2.29: Diagramme en barre des erreurs absolues moyennes suivant les autres variables

Nous remarquons que les erreurs absolues moyennes des estimations de SASMO est plus petite que ceux des estimations de l'EKF, mis à part l'erreur absolue moyenne de l'estimation de l'angle de lacet *psi*.

Tout cela indique que SASMO présente de meilleures estimations que celles de l'EKF.

L'écart-type

L'écart type, σ, est une mesure de dispersion des données. Elle est défini comme suit :

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(y_i - \bar{y})}{N}} \tag{2.3}$$

Où :

y, est la valeur réelle,

$\bar{y}$, est la valeur moyenne,

N, le nombre de mesures.

Les résultats obtenus pour chaque variable sont résumés dans le tableau 2.3 et nous pouvons les visualiser sur la figure 2.30.

Variable	**SASMO**	**EKF**
X	0.0905	0.0963
Y	0.1050	0.1286
Z	0.0226	0.0795
ϕ	0.0299	0.0515
θ	0.0440	0.0619
ψ	0.0083	$6.7407 \cdot 10^{-05}$

TABLE 2.3: Écarts-types des estimations des deux approches

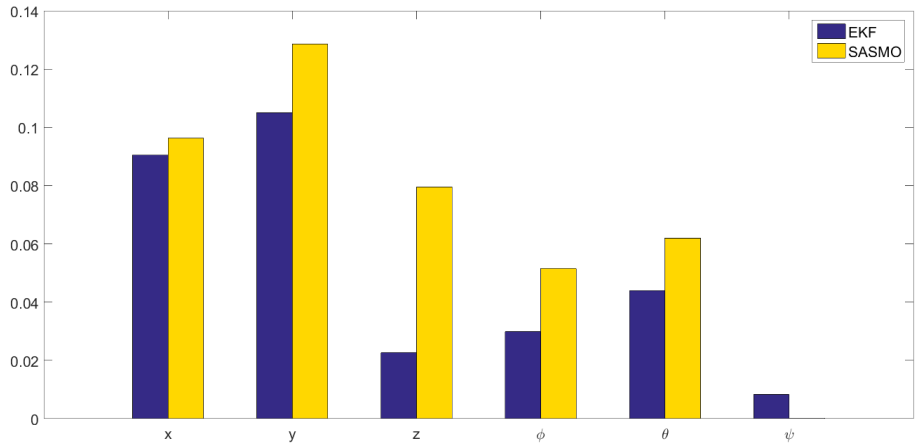


FIGURE 2.30: Diagramme en barre des écarts-types des erreurs

Nous constatons que les écarts-types des estimations de SASMO est inférieur ou égal à ceux de l'EKF, mis à part l'écart-type de l'estimation de l'angle de lacet ψ.

Cela indique bien que les estimations de SASMO sont moins dispersées que celles de l'EKF.

2.4 Conclusion

Dans ce chapitre, nous avons présenté ROS et ses notions de bases, ainsi que Gazebo. Nous avons ensuite implémenté en utilisant ces dernier les deux approches de SLAM monoculaire visuel présentées au chapitre précédent. Puis, nous les avons simulé et fait une comparaison qualitative. Enfin, nous avons fait une comparaison quantitative des deux approches en utilisant quelques indicateurs d'écart et nous avons obtenu que l'approche SASMO est meilleure que l'approche EKF.

Conclusion Générale et Perspectives

Le comportement dynamique d'un procédé peut être entièrement décrit par l'évolution de ses variables d'état. Pour la commande et le diagnostic d'un système dynamique, la connaissance de ces variables est capitale. Or les grandeurs directement mesurées (sorties) ne couvrent généralement pas la totalité des états dynamiques du procédé. Ce problème peut être résolu, sous certaines conditions, en introduisant un observateur d'état dont la tâche sera de fournir une estimation du vecteur d'état du système étudié.

Ce travail s'est centré autour de la réalisation d'une étude comparative entre deux observateurs : le filtre de Kalman étendu (EKF) et un observateur glissant stochastique et adaptatif (SASMO) dans un contexte de SLAM visuel Monoculaire.

Nous avons commencé ce mémoire par une brève introduction sur le problème du SLAM en général et celui du SLAM visuel monoculaire en particulier, où nous avons défini ce qu'ils sont, présenté leurs état de l'art et proposés deux approches différentes pour les résoudre ; l'une en utilisant le filtre de Kalman étendu (EKF) et l'autre en utilisant un observateur glissant stochastique et adaptatif (SASMO).

Nous avons ensuite présenté brièvement ROS et Gazebo que nous avons utilisé pour implémenter et simuler les deux approches proposées de manière réaliste. Enfin, nous avons réalisé une comparaison qualitative et quantitative entre ces deux dernières.

Dans le perspective d'une continuité de ce travail, nous proposons :
- La réalisation d'une étude comparative avec d'autres scénarios qui incluent mais ne sont pas limités à l'occlusion de la caméra, l'introduction de perturbations comme le vent, etc.
- L'utilisation d'autres méthodes pour l'estimation de l'échelle

Bibliographie

[1] Strasdat H., Montiel J.M.M. et Davi-son A.J. « Real-time monocular slam : Why filter ? » In : *IEEE Int. Conf. Robotics and Automation (ICRA)*. 2010.

[2] Hugh Durrant-Whyte et Tim Bailey. « Simultaneous localization and mapping: part I ». In : *Robotics & Automation Magazine, IEEE* 13.2 (2006), p. 99–110.

[3] Andrew J Davison et al. « MonoSLAM: Real-time single camera SLAM ». In : *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 29.6 (2007), p. 1052–1067.

[4] Georg Klein et David Murray. « Parallel tracking and mapping for small AR workspaces ». In : *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*. IEEE. 2007, p. 225–234.

[5] Miroslav Trajković et Mark Hedley. « Fast corner detection ». In : *Image and vision computing* 16.2 (1998), p. 75–87.

[6] Bill Triggs et al. « Bundle adjustment—a modern synthesis ». In : *Vision algorithms: theory and practice*. Springer, 1999, p. 298–372.

[7] Richard A Newcombe, Steven J Lovegrove et Andrew J Davison. « DTAM: Dense tracking and mapping in real-time ». In : *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE. 2011, p. 2320–2327.

[8] Jakob Engel, Thomas Schöps et Daniel Cremers. « LSD-SLAM: Large-scale direct monocular SLAM ». In : *Computer Vision–ECCV 2014*. Springer, 2014, p. 834–849.

[9] Raul Mur-Artal, JMM Montiel et Juan D Tardos. « ORB-SLAM: a versatile and accurate monocular SLAM system ». In : *Robotics, IEEE Transactions on* 31.5 (2015), p. 1147–1163.

[10] Ethan Rublee et al. « ORB: an efficient alternative to SIFT or SURF ». In : *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE. 2011, p. 2564–2571.

[11] Thierry Viéville et Olivier D Faugeras. « Cooperation of the inertial and visual systems ». In : *Traditional and non-traditional robotic sensors*. Springer, 1990, p. 339–350.

[12] Peter Corke, Jorge Lobo et Jorge Dias. « An introduction to inertial and visual sensing ». In : *The International Journal of Robotics Research* 26.6 (2007), p. 519–535.

[13] A DÉCOUVRIR ÉGALEMENT. « La théorie de la relativité restreinte et générale ». In : (1990).

[14] Jorge J Moré. « The Levenberg-Marquardt algorithm: implementation and theory ». In : *Numerical analysis*. Springer, 1978, p. 105–116.

[15] Gabe Sibley et al. « Vast-scale outdoor navigation using adaptive relative bundle adjustment ». In : *The International Journal of Robotics Research* (2010).

[16] Kurt Konolige et al. « View-based maps ». In : *The International Journal of Robotics Research* (2010).

[17] Frank Dellaert et Michael Kaess. « Square Root SAM: Simultaneous localization and mapping via square root information smoothing ». In : *The International Journal of Robotics Research* 25.12 (2006), p. 1181–1203.

[18] Paul Furgale, Timothy D Barfoot et Gabe Sibley. « Continuous-time batch estimation using temporal basis functions ». In : *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE. 2012, p. 2088–2095.

[19] Rudolph E Kalman et Richard S Bucy. « New results in linear filtering and prediction theory ». In : *Journal of basic engineering* 83.1 (1961), p. 95–108.

[20] Jakob Engel, Jürgen Sturm et Daniel Cremers. « Accurate figure flying with a quadrocopter using onboard visual and inertial sensing ». In : *IMU* 320 (2012), p. 240.

[21] Jakob Engel, Jürgen Sturm et Daniel Cremers. « Scale-aware navigation of a low-cost quadrocopter with a monocular camera ». In : *Robotics and Autonomous Systems* 62.11 (2014), p. 1646–1656.

[22] D Gordon E Robertson et James J Dowling. « Design and responses of Butterworth and critically damped digital filters ». In : *Journal of Electromyography and Kinesiology* 13.6 (2003), p. 569–573.

[23] Reza Raoufi et Hamid Khaloozadeh. *Stochastic/Adaptive Sliding Mode Observer for Noisy Excessive Uncertainties Nonlinear Systems*. Liege, Belgium : 15th PSCC, 2005.

[24] Bernt Øksendal. *Stochastic differential equations*. Springer, 2003.

[25] Bernt Oksendal. *Stochastic differential equations: an introduction with applications*. Springer Science & Business Media, 2013.

[26] Fakhreddin Abedi, Wah June Leong et Sarkhosh Seddighi Chaharborj. « A notion of stability in probability of stochastic nonlinear systems ». In : *Advances in Difference Equations* 2013.1 (2013), p. 1–8.

[27] *ROS*. 27 mai 2016. URL : `http://www.ros.org/`.

[28] *Licence BSD*. 27 mai 2016. URL : `http://www.freebsd.org/copyright/license.html`.

[29] *Licence MIT*. 27 mai 2016. URL : `https://opensource.org/licenses/MIT`.

[30] *Simulateur Gazebo*. 28 mai 2016. URL : `http://gazebosim.org/`.

[31] *Robot Baxter*. 10 juin 2016. URL : `http://www.rethinkrobotics.com/baxter`.

[32] *Package Hector Quadrotor*. 11 juin 2016. URL : `http://wiki.ros.org/hector_quadrotor`.