THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *Lundi 08 Juillet 2013* **par :**

AURÉLIEN GONZALEZ

Localisation par Vision Multi-spectrale
Application aux Systèmes Embarqués

JURY

PHILIPPE MOUYON	Professeur d'Université	Président du Jury
ROGER REYNAUD	Professeur d'Université	Examinateur
RENÉ ZAPATA	Professeur d'Université	Rapporteur
PAUL CHECCHIN	Maître de Conférences	Rapporteur
JOAN SOLÀ ORTEGA	Ingénieur Consultant	Invité
JONATHAN PIAT	Maître de Conférences	Invité
MICHEL DEVY	Directeur de Recherche	Directeur de Thèse
ANDRÉ MONIN	Chargé de Recherche	Codirecteur de Thèse

École doctorale et spécialité :
EDSYS : Robotique 4200046
Unité de Recherche :
Laboratoire d'Analyse et d'Architecture des Systèmes (UMR 8001)
Directeur(s) de Thèse :
Michel DEVY et *André MONIN*
Rapporteurs :
René ZAPATA et *Paul CHECCHIN*

Remerciements

Je ne saurais débuter ce paragraphe sans mentionner mon directeur et codirecteur de thèse, Michel Devy et André Monin. Et à juste titre ! Je tiens tout d'abord à les remercier de m'avoir offert l'opportunité de réaliser une thèse au sein du LAAS-CNRS. Egalement merci à eux pour leur confiance et l'autonomie qu'ils m'ont accordé, de leur aide et de leur soutien lors des moments de doute. Merci pour leur accompagnement à la suite de ce projet professionnel.

Je souhaiterais exprimer ma reconnaissance aux membres du jury pour l'intérêt porté à mes recherches et remercier particulièrement Philippe Mouyon, Roger Reynaud, René Zapata et Paul Checchin pour leurs remarques constructives et leur lecture pertinente de cet ouvrage.

Je tiens tout particulièrement à remercier Joan Solà pour la qualité de ses travaux de recherche, ses développements et ses manuscrits qui ont été une vrai aide tout au long de ma thèse. Je remercie également chaleureusement Jean-Marie Codol et Cyril Roussillon pour leur disponibilité et leur aide très précieuse, à la fois théorique et technique, contribuant ainsi aux résultats présentés dans cette thèse.

Pour l'intérêt qu'ils ont porté à mes travaux et leur participation à l'amélioration et au portage du *CSLAM* sur système embarqué, je souhaite remercier mes collègues du LAAS, Jonathan, Aurélien, Daniel et Diego.

Le projet SART a été une réelle opportunité d'interagir avec des industriels expérimentés de THALES, FLIR et LATECOERE. Un grand merci à Christian Larrouy, Martial Grimm et Jacques Tran pour leur sympathie et leur professionnalisme.

Pour les excellents moments passés dans le bureau E40, je ne saurais remercier assez mes très bons amis Ahmad Al Sheikh, Rémi Sharrock et Jean-Marie Codol. Merci à Tom, Olivier, Balakrishna, Tatiana, Josselin, Josu, Maialen et Ane pour les très divertissantes pauses déjeuner et café. Merci aussi à mes derniers collègues de bureau et ceux d'à côté : Matthieu, Wassim, Mamadou, Redouane et David.

Un grand merci au LAAS pour m'avoir accueilli durant mes travaux de thèse et à son personnel dévoué et qualifié, pour son aide technique et administrative. Merci également aux groupes SARA et RAP pour leur accueil chaleureux ainsi que leur aide.

Je n'aurais pensé réaliser une thèse sans la qualité des différents enseignements reçus durant ma formation. Je souhaiterais exprimer ma gratitude aux professeurs de l'IUT GEII de Toulouse et de l'IUP SI pour avoir su transmettre leur passion pour la recherche grâce à des cours de qualité. Merci également à mes anciens camarades Tony, Gégé, Fafa et Cricri avec qui j'ai régulièrement l'occasion de me remémorer de bons souvenirs et anecdotes.

Merci à Magellium de m'offrir l'opportunité de poursuivre mon activité dans la *Localisation basée Vision* sur des projets innovants. Je tiens tout particulièrement à remercier l'équipe Robotique & Perception pour leur chaleureux accueil, leur entraide et leur convivialité.

Merci à mes amis d'enfance, Nicolas, Jacques, Marine, Vincent Laetitia, Franck et Emilie. Le temps m'a cruellement manqué ces dernières années pour les voir plus souvent. Merci également à Alexandre, Thierry et Frédo, amis et compagnons de longue date qui m'ont soutenu durant cette aventure.

Je ne saurais conclure sans bien évidemment remercier ma famille pour leur soutien inconditionnel, particulièrement mes parents, mon frère et sa petite famille. Enfin, une pensée pour celle qui partage ma vie, Audrey, et mes accueillants et généreux beaux-parents Albert et Brigitte.

MERCI à tous...

Table des matières

Table des figures

Table des symboles

Optique & Vision

$\mathcal{R}_\mathcal{M}$, $\mathcal{R}_\mathcal{C}$, $\mathcal{R}_\mathcal{I}$	Repère Monde, Repère Caméra, Repère Image
$[X_M, Y_M, Z_M]^\mathsf{T}$	Point 3D exprimé dans le repère Monde
$[X_C, Y_C, Z_C]^\mathsf{T}$	Point 3D exprimé dans le repère Caméra
$[x_I, y_I, 1]^\mathsf{T}$	Point 3D exprimé dans le repère Image
$[x_d, y_d]^\mathsf{T}$	Point 2D dans le repère Image après application du modèle de distorsion
$\boldsymbol{f}$	Focale
(u,v)	Coordonnées d'un pixel dans l'image
u_0, v_0	Centre de l'image exprimé en pixel selon u et v
s_u, s_v	Nombre de pixels par unité de longueur selon u et v
s_θ	Rectangularité du pixel
α_u, α_v	Focale exprimée en pixels
$\mathbf{K}$	Matrice intrinsèque
$k_1, k_2, k_3, ...$	Paramètres de distorsions radiaux
$c_2, c_4, c_6, ...$	Paramètres de correction

Filtrage

$\mathbf{x}_k$	Vecteur de valeurs moyennes de l'état d'un système à l'instant k
$\mathbf{P}$	Matrice de covariance de $\mathbf{x}_k$ à l'instant k
$\mathbf{y}_k$	Etat observé d'un système à l'instant k
$\mathbf{f}(\cdot)$	Fonction d'évolution du système
$\mathbf{h}(\cdot)$	Fonction d'observation du système
$\mathbf{F}, \mathbf{H}$	Matrices jacobiennes des fonctions ci-dessus
$\mathcal{N}(\mathbf{x} - \hat{\mathbf{x}}; \mathbf{X})$	$\mathbf{x}$ suit une une distribution Gaussienne de moyenne $\hat{\mathbf{x}}$ et de variance $\mathbf{X}$
$\mathbf{z}$	Innovation
$\mathbf{Z}$	Covariance de l'innovation
$\mathbf{K}$	Gain du filtre de Kalman

SLAM

$\mathcal{R}$	Vecteur d'état du système
$\mathcal{M}$	Vecteur d'état de l'ensemble des amers
$\mathcal{L}_i$	Amer i
$\mathcal{C}$	Caméra
$\mathbf{u}$	Vecteur de contrôle de la commande
$\mathbf{U}$	Variance de $\mathbf{u}$
v	Bruit de mesure
$\mathbf{R}$	Variance de v
$\mathbf{g}(\cdot)$	Inverse de la fonction d'observation
$\mathbf{G}$	Jacobienne de $\mathbf{g}(\cdot)$

SLAM Visuel

$\mathbf{p} = (p_x, p_y, p_z)^\mathsf{T}$	Vecteur de position
$\mathbf{q} = (q_w, q_x, q_y, q_z)^\mathsf{T}$	Représentation d'une orientation sous forme de quaternion
$\mathbf{v} = (v_x, v_y, v_z)^\mathsf{T}$	Vitesse linaire
$\mathbf{w} = (w_x, w_y, w_z)^\mathsf{T}$	Vitesse angulaire
$\mathbf{ab} = (ab_x, ab_y, ab_z)^\mathsf{T}$	Biais des accéléromètres de la centrale inertielle
$\mathbf{wb} = (wb_x, wb_y, wb_z)^\mathsf{T}$	Biais des gyromètres de la centrale inertielle
$\mathbf{g} = (g_x, g_y, g_z)^\mathsf{T}$	Gravité
Δt	Pas de temps
$Q(.)$	fonction de transformation d'un vecteur en quaternion
$R(.)$	fonction de transformation d'un quaternion en matrice de rotation

Table des acronymes

Projet

SART	Système d'Aide au Roulage Tout-temps
IHM	Interface Homme-Machine
DAL	*Design Assurance Level* : Niveau d'assurance du design
IMU	*Inertial Measurement Unit* : Centrale inertielle
GPS-RTK	*Real Time Kinematic GPS* : GPS Cinématique temps réel

SLAM

SLAM	*Simultaneous Localization and Mapping* : Localisation et cartographie simultanées
MOT	*Mobile Object Tracking* : Suivi des objets mobiles
KF	*Kalman Filter* : Filtre de Kalman
EKF	*Extended Kalman Filter* : Filtre de Kalman étendu
PF	*Particle Filter* : Filtre particulaire
SAM	*Smoothing And Mapping* : Lissage et cartographie
SFM	*Structure From Motion*
EP	*Euclidean Point* : Point Euclidien
IDP	*Inverse Depth Parametrization* : Paramétrisation en distance inverse
AHP	*Anchored Homogeneous Point* : Point homogène ancré
ZNCC	*Zero-mean Normalized Cross Correlation* : Corrélation croisée normalisée centrée
RANSAC	*RANdom SAmple Consensus* : Consensus d'échantillons aléatoires
NEES	*Normalised Estimation Error Squared* : Erreur carrée normalisée de l'estimation
BDD	Base De Données

Systèmes embarqués

DSP	*Digital Signal Processor* : Processeur de signal numérique
SIMD	*Single Instruction on Multiple Data* : Processuer à instruction unique, données multiples
GPU	*Graphics Processing Unit* : Processeur graphique
FPU	*Floating Point Unit* : Unité de calcul en virgule flottante
CPU	*Central Processing Unit* : Unité centrale de traitement
FPGA	*Field-Programmable Gate Array* : Circuit logique programmable
LUT	*Look-Up Table* : Table de correspondance
PLB	*Processor Local Bus* : Bus de processeur local
PCIe	*Peripheral Component Interconnect express* : Composant d'interconnexion de périphériques
BRAM	*Block Random Access Memory* : Mémoire à accès aléatoire par block

Chapitre 1

Introduction

1.1 Introduction Générale

La localisation est une des problématiques majeures des systèmes navigants. Ces dernières années, le succès des *Systèmes de Guidage par Satellite*, ou GPS, a été tel que la plupart des véhicules en sont équipés de série, et ils deviennent une application incontournable des *Smartphones*. Pour quelques dizaines d'euros, il est alors possible de se localiser à une dizaine de mètres avec une erreur de quelques mètres.

Cependant, bien que cette précision soit suffisante pour un véhicule, certaines applications peuvent nécessiter une précision beaucoup plus fine, de l'ordre du mètre ou du centimètre. Une telle précision demande un investissement bien plus important. Actuellement, les systèmes les plus répandus pour une localisation de précision sont le GPS différentiel (DGPS) et le GPS-RTK (Real Time Kinematic). Si la première application est basée sur la mesure de différences entre la position d'une station de référence et une application, la seconde repose sur la mesure de phase des ondes porteuses, et nécessite également une station de référence. Les gammes de prix sont également différentes de celles associées aux GPS grand public, puisque le prix d'un GPS de précision peut varier de quelques centaines à plusieurs dizaines de milliers d'euros. Il est alors difficile d'imaginer le prix d'un système de localisation grand public s'envoler pour une précision de quelques mètres supplémentaires.

Plutôt que d'investir sur un seul équipement de localisation onéreux, des fournisseurs spécialisés pour certaines applications (agriculture, aéronautique, transport en commun...) se sont focalisés sur la fusion des capteurs bas coûts, permettant également l'observation de nouvelles données. Par exemple, l'utilisation de sonars ou de lasers, en plus de l'utilisation d'un GPS, permet de positionner un système localement et de détecter d'éventuels obstacles. Une caméra embarquée sur un véhicule ou un robot peut donner une estimation du mouvement par *Odométrie visuelle* ou bien réaliser une Localisation et Cartographie Simultanées (SLAM). Elle permet aussi de détecter et de suivre les objets mobiles (MOT). Enfin, une centrale inertielle permet de connaître les accélérations, les vitesses de giration et le cap du mobile. La fusion des données acquises par ces capteurs peut rendre une application précise au mètre près pour le coût d'un GPS différentiel bas de gamme. Ces capteurs permettent également une observation plus riche de l'état du système et de son environnement.

Avec la miniaturisation et la baisse des coûts de production, ces capteurs sont omniprésents dans notre environnement quotidien : télévisions, consoles de jeu, téléphones, voitures ... Dernièrement, les capteurs ont envahi les véhicules jusqu'à les rendre totalement autonomes. Ils peuvent parcourir plusieurs centaines de milliers de kilomètres sans aucun accrochage, à l'image de la *Google Car*. Des performances qui peuvent rendre jaloux les autres secteurs d'activités. Afin de réduire les incidents lors de la phases de roulage au sol des gros porteurs, l'aéronautique

s'est penchée sur ces applications de localisation et de détection "bas coût".

1.2 Contexte et Motivations

Les travaux présentés dans cette thèse se placent dans le cadre du projet SART, acronyme pour *Système d'Aide au Roulage Tout-temps*. Il bénéficie d'un financement DGE-FUI (Direction Générale des Entreprises - Fond Unique Interministériel) et il est labellisé par le pôle de compétitivité *Aerospace Valley*. Les développements ont débuté à l'issue du dépôt du brevet *"Method and System for Aircraft Taxiing Assistance"* en avril 2009 sous le n° WO2009044257(A2) [Latécoère, 2009].

Actuellement, la navigation d'un aéronef au sol depuis sa zone de parking vers son point fixe avant le décollage, et son retour à sa zone de parking après l'atterrissage sont contrôlés par le pilote. Celui-ci ne dispose que de moyens visuels directs pour contrôler sa manoeuvre. Une bonne évaluation de l'environnement de l'aéronef, ainsi que la connaissance de sa maniabilité et de ses dimensions sont alors fondamentaux pour éviter les défaillances. Cela est d'autant plus important lorsque les conditions météorologiques sont dégradées (brouillard, pluie, neige ...), entraînant généralement une réduction du trafic aérien ou le déroutage des aéronefs vers des terrains dégagés. La perte de visibilité engendrée par ces conditions accroît le risque de défaillance. La catastrophe aérienne la plus meurtrière de l'histoire a eu lieu sur l'aéroport de Tenerife en 1977 impliquant deux appareils roulant au sol et par temps de brouillard, faisant 583 victimes. Chaque année, une moyenne de 50 accidents lors du roulage est dénombrée. Il s'agit généralement d'incidents mineurs, lors desquels, par exemple, l'extrémité d'un aéronef entre en collision avec un autre objet. Ce risque est d'autant plus important sur les gros porteurs où l'envergure est plus délicate à estimer pour un pilote, et l'angle mort de vision plus important car la cabine de pilotage est plus haute par rapport au sol. Pour les manoeuvres proches de la zone de parking, les pilotes sont guidés par des *Marshallers*, opérateurs au sol en charge du guidage de l'avion. Pour opérer dans des conditions météorologiques dégradées, d'incertitude de localisation ou de perte sur un aéroport complexe (Charles de Gaulle, Francfort, Amsterdam, ...), les pilotes sont aidés par des véhicules dis *"Follow Me"* les précédant sur la voie à suivre. Ces véhicules sont également mis en oeuvre dans de grands aéroports, lorsque le pilote se perd ou est incapable de rejoindre la zone de parking. Cette modalité d'assistance à la navigation peut engendrer des délais importants qui se répercutent sur le trafic au sol et en vol. De plus, une note de sécurité du *National Transportation Safety Board* de septembre 2012 recommande l'installation de systèmes d'aide à la prévention de collisions et permettant la surveillance des bouts d'ailes sur les dernières générations de gros porteurs [NTSB, 2012]. Cette recommandation a été formulée suite à l'investigation de 12 accidents impliquant ce type d'aéronef. La demande d'un système de localisation, de guidage et de détection pour l'aéronautique est donc bien présente.

A ce jour, quelques systèmes existent pour faciliter le guidage et aider à la localisation de l'aéronef au sol. L'un des premiers systèmes destiné à l'aviation commerciale a été développé par Latécoère en 2000. Présenté en figure 1.1 à gauche, ce système nommé ETACS, acronyme pour *External and Taxi Aid Camera System*, est composé de 5 caméras et d'un calculateur numérique. Il permet la transmission haut débit des images vers les écrans des pilotes. Ces images peuvent être également transmises sur les écrans des passagers. Ce système, toujours en service et très apprécié par le personnel naviguant et les passagers, équipe les Airbus long-courriers, et dernièrement l'Airbus A380 [NTSB, 2012].

En 2002, un brevet a été déposé par Boeing pour un système similaire, nommé *Airplane Ground Maneuvering Camera System* [Boeing, 2002], qui équipe de série le dernier Boeing 777. Le système présenté peut comprendre jusqu'à six caméras, dont au moins deux sont placées

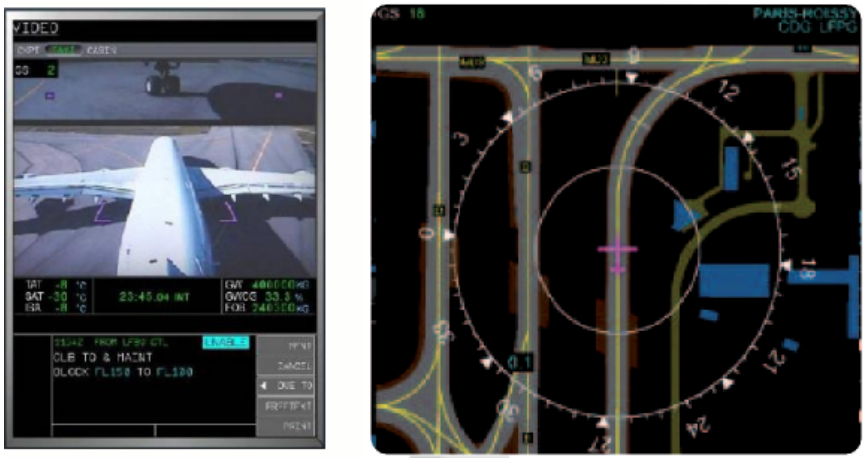


FIGURE 1.1 – Systèmes d'aide à la visualisation de l'aéronef et à sa localisation. A gauche, le système *External and Taxi Aid Camera System* (ETACS) développé par Latécoère permet la visualisation de la quasi-totalité de l'avion grâce à une caméra placée sur la dérive. Des marqueurs sont incrustés dans l'image pour l'aide au guidage (source [NTSB, 2012]). A droite, le système *Onboard Airport Navigation System* (OANS) facilite la localisation de l'aéronef sur un aéroport (source [Thalès, 2010]).

sur la roulette avant et le train principal. Des informations pour l'aide au guidage sont ensuite incrustées dans les images pour l'aide au roulage.

En 2004, Rockwell Collins dépose un brevet pour un système nommé *Enhanced vision system* (EVS). Il se compose d'une caméra infrarouge, d'une centrale inertielle, d'un GPS et d'un radar. L'image provenant de la caméra infrarouge est ensuite enrichie par des informations de guidage (trajectoire programmée, vitesse, vent...) avant d'être affichée sur un Head-Up Display (HUD) [Rockwell-Collins, 2004].

Une des dernières applications en matière de localisation sur aéroport a été brevetée par Airbus en 2009 [Airbus, 2009] et est développée par Thalès sous le nom de *Onboard Airport Navigation System* (OANS) [Thalès, 2010]. Illustré en figure 1.1 à droite, ce système présente au pilote la position courante de l'avion sur une cartographie de l'aéroport. Option sur les derniers Airbus A380, ce système est une aide précieuse quand à la localisation générale de l'avion sur un aéroport, mais il dépend de la précision de la cartographie et du couple centrale inertielle - GPS. Ce système garantit une précision à 5 mètres avec une intégrité à 8 mètres.

Parmi les systèmes d'aide au roulage existants et présentés, la visualisation de l'environnement et la localisation se font par deux systèmes différents. Un des objectifs du projet SART est de présenter au pilote une image fiable par toutes conditions météorologiques afin de lui permettre de déplacer l'aéronef, à faible allure, jusqu'à son point fixe ou jusqu'à sa zone de parking.

1.3 Contributions au projet SART

Le développement d'un tel système implique de grands acteurs du secteur aéronautique et de la vision. Mis en oeuvre par Latécoère et présenté en figure 1.2 à gauche, le système se compose :

– D'une tête de prise de vue présentée en figure 1.2 à droite, et comprenant une centrale inertielle, une caméra couleur et une caméra infrarouge. Ce module *Capteurs*, permettant

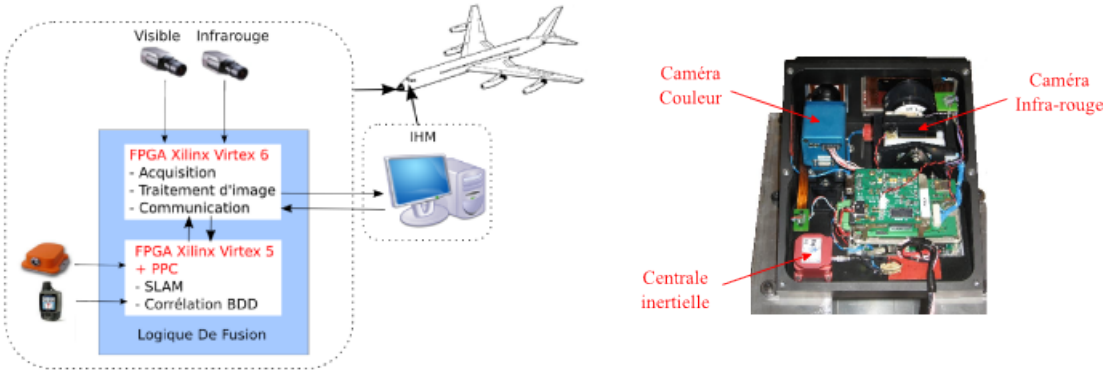


FIGURE 1.2 – Système SART. A gauche, le diagramme de l'architecture du système. A droite, la mise en œuvre dans la tête de prise de vue.

d'acquérir les données sensorielles, est développé par FLIR Systems.

- D'un module *Logique de Fusion*. Cette unité, développée par le LAAS-CNRS a pour but d'acquérir et de traiter les signaux issus des capteurs de vision, d'une centrale inertielle, d'un GPS et d'une base de données contenant la cartographie de l'aéroport. Grâce à ces traitements, elle doit pouvoir localiser le porteur au mètre près et en temps réel, et détecter les éventuels obstacles.
- D'une Interface Homme-Machine (IHM) développée par Thalès Avionics où les différentes données pertinentes permettent d'enrichir l'image présentée au pilote, provenant de la bande spectrale visible ou infrarouge.

Le module de Logique de Fusion est au coeur du système SART. Une des premières tâches fut de définir l'interface de ce module avec les fonctions développées par les partenaires industriels du projet. Nous avons donc participé à la définition de l'architecture logicielle présentée en figure 1.3.

Celle-ci présente les fonctionnalités principales réalisées par la Logique de Fusion.

- Les images sont réceptionnées, corrigées en distorsion et en dynamique. Des traitements plus particuliers peuvent être réalisés sur le contraste de l'image infrarouge, celle-ci étant sur 14 bits. Des indices perceptuels sont extraits de ces images (points d'intérêt, lignes, potentiels obstacles...).
- Une fonction de SLAM/MOT (Simultaneous Localisation And Mapping / Mobile Object Tracking) est réalisée à l'aide d'un filtre de Kalman étendu et utilise ces points d'intérêt, ou autres indices, ainsi que les données provenant d'une centrale inertielle et d'un GPS. Parallèlement, les objets hors-sol sont détectés par *Inverse Perspective Mapping*.
- A la sortie de la fonction de SLAM/MOT, les données de localisation peuvent être appariées à une cartographie locale afin de localiser l'aéronef et les différents objets d'intérêt dans la cartographie globale.
- Ces données sont ensuite transférées à l'IHM qui les incruste sur un écran pour l'aide au pilotage.

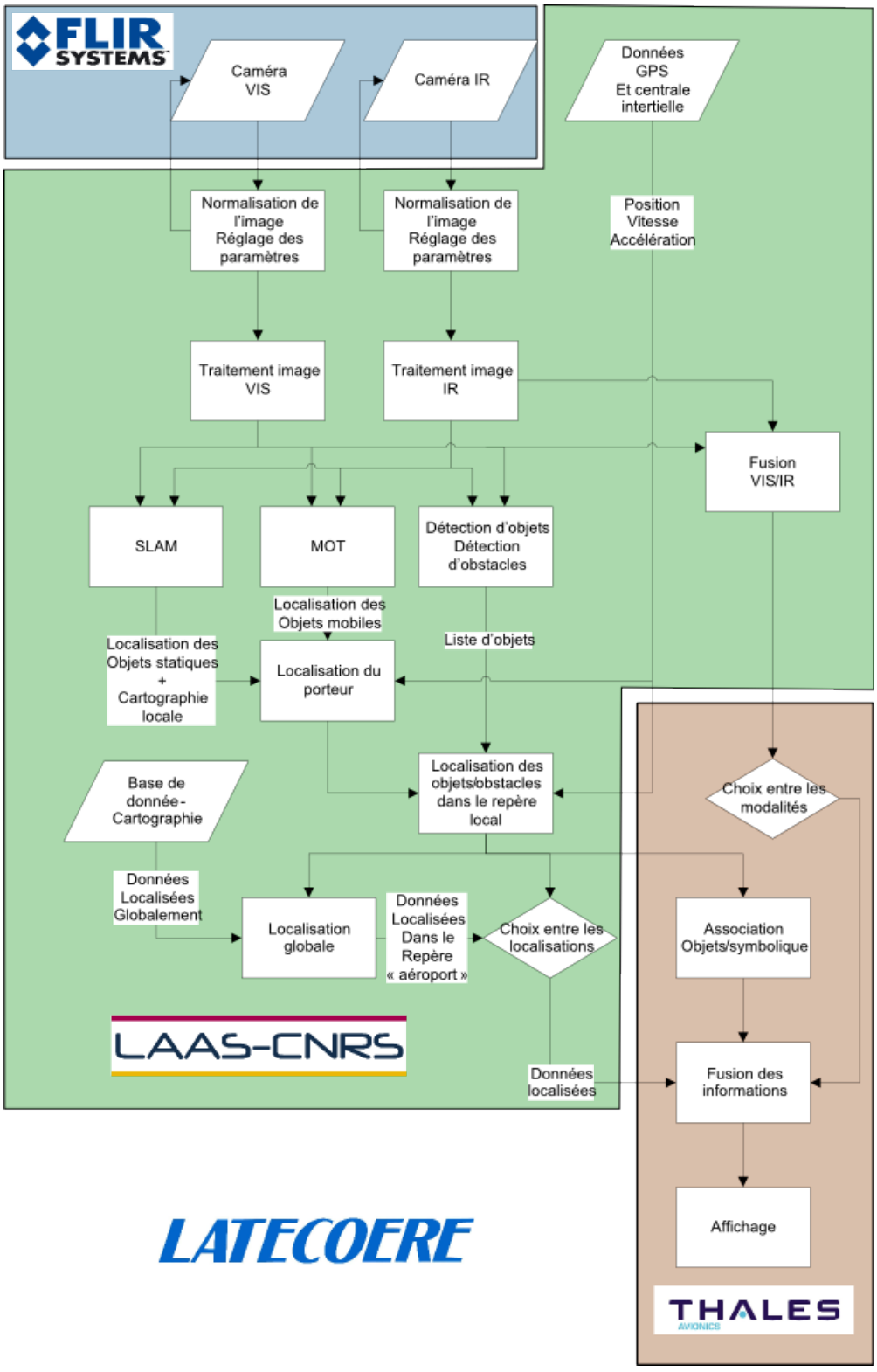


FIGURE 1.3 – Architecture logicielle du module *Logique de Fusion* du système SART.

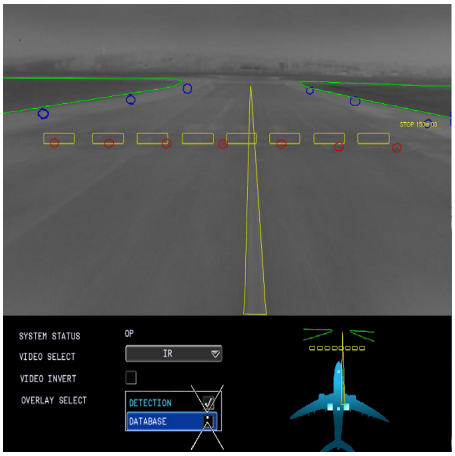


FIGURE 1.4 – Interface Homme-Machine présentée au pilote.

Parallèlement, nous avons participé aux études menant aux choix de la bande spectrale à utiliser par le développement d'algorithmes permettant le traitement et la détection de points dans des images de l'infrarouge proche et lointain.

Le développement de l'algorithme *RT-SLAM* [Roussillon et al., 2011] a mobilisé une équipe de six personnes et doté le LAAS d'un SLAM visuel fonctionnant en temps réel sur des images à 60 Hz et utilisant une centrale inertielle pour la prédiction. Sur ce projet, nous avons contribué au développement de certaines fonctions mathématiques relatives au filtre de Kalman étendu, sur les fonctions de projection et rétro-projection de points à partir d'une caméra *pin-hole* ainsi que sur certaines propriétés relatives à la paramétrisation des points d'intérêt. Des algorithmes spécifiques ont également été développés pour les tests unitaires, prouvant les différentes fonctions grâce à la *SLAM-Toolbox* [Sola et al., 2009].

Une fois un premier prototype de *RT-SLAM* fonctionnel, les efforts ont été portés sur le développement d'un algorithme de SLAM destiné aux systèmes embarqués, nommé *C-SLAM* [Gonzalez et al., 2011]. Les algorithmes de C-SLAM sont largement inspirés de *RT-SLAM*, mais l'ensemble des fonctions ont dû être re-développées afin de produire un logiciel compatible avec les contraintes relatives aux systèmes embarqués avioniques. Ces contraintes interdisent les allocations dynamiques de mémoire, les boucles récursives ou l'utilisation de librairies externes.

Parallèlement, l'interface entre la Logique de Fusion et l'Interface Homme-Machine a dû être développée et testée unitairement. La figure 1.4 présente l'interface développée par Thalès Avionics, enrichie des informations fournies par notre algorithme. Les données de localisation, de détection et de guidage ont dû être intégrées à l'image à transmettre afin d'assurer leur synchronisation et minimiser la perte d'informations.

En dernier lieu, les efforts de développement ont été portés sur l'amélioration de C-SLAM, prenant en compte des données GPS et une base de données cartographique, dont les observations ont été incluses à l'étape de correction du filtre de Kalman étendu. Des données expérimentales à bord du robot d'exploration *Mana* ont été acquises pour l'évaluation de cet algorithme.

Grâce au profilage de celui-ci, nous avons pu déterminer les fonctions coûteuses et certaines ont été portées sur les éléments logiques du FPGA intégrés à la carte. Le co-design du système a pu être mis en place puis évalué dans [Botero et al., 2012]. Simultanément, un transfert de technologie vers Latécoère a été opéré.

1.4 Publications

SLAM visuel monoculaire par caméra infrarouge. A. Gonzalez, M. Devy, J. Solà Ortega. Dans 13ème Congrès ORASIS des Jeunes Chercheurs en Vision, Praz-sur-Arly (France), 6 - 10 juin 2011.

RT-SLAM : A Generic and Real-Time Visual SLAM Implementation. C. Roussillon, A. Gonzalez, J.Solà, J.M. Codol, N. Mansard, S.Lacroix, M. Devy. Dans 8th International Conference on Computer Vision Systems, Sophia Antipolis (France), 20 - 22 septembre 2011.

A C-embedded Algorithm for Real-Time Monocular SLAM. A. Gonzalez, J.M. Codol, M. Devy. Dans 18th International Conference on Electronics, Circuits and Systems, Beyrouth, Liban, 11 - 14 décembre 2011.

Architecture embarquée pour le SLAM monoculaire. D. Botero, A. Gonzalez, M. Devy. Dans 18ème édition de Reconnaissance des Formes et Intelligence Artificielle, Lyon (France), 25 - 27 janvier 2012.

1.5 Organisation du mémoire

Cette thèse traite de la problématique de la localisation par la vision monoculaire visible et infrarouge, l'intégration de différents capteurs, et son application aux systèmes embarqués.

Le chapitre 2 présente les outils utilisés dans la vision par ordinateur. Ainsi, il rappelle les modèles et les opérations usuelles de ce domaine. La localisation étant traitée à partir d'images acquises dans différentes bandes spectrales, il présente également celles-ci dans ce chapitre.

Afin de résoudre la problématique du SLAM, nous avons sélectionné une méthode probabiliste basée sur le filtre de Kalman étendu. Après un état de l'art sur les différentes méthodes mises en application par la communauté, le chapitre 3 justifie notre choix, et présente les outils mathématiques nécessaires à l'implémentation d'une telle solution.

Après avoir présenté différents travaux sur le SLAM monoculaire, le chapitre 4 s'attache aux différentes méthodes implémentées afin d'accélérer et robustifier la localisation, en vue d'un portage sur des systèmes pourvus de faibles ressources. Différents résultats obtenus grâce à un algorithme spécifiquement développé pour les systèmes embarqués sont présentés. La cohérence [1] des résultats est étudiée, et ce chapitre compare la qualité des résultats obtenus avec une caméra visible et infrarouge, avec ou sans l'utilisation de capteurs externes, tels qu'une centrale inertielle ou un GPS.

Le chapitre 5 introduit une méthode pour l'exploitation d'une cartographie pour le SLAM visuel monoculaire. Il précise les étapes nécessaires à l'extraction de primitives dans l'image devant être facilement corrélables à celles de la cartographie. Il présente ensuite comment l'observation de la pose du mobile grâce à cette cartographie peut être intégrée à un SLAM par filtre de Kalman étendu, et expose les résultats obtenus.

Enfin, le chapitre 6 s'intéresse au portage des algorithmes sur des systèmes embarqués. Après une introduction des travaux déjà effectués dans ce domaine particulier, il propose un co-design développé pour l'obtention de résultats en temps réel. L'identification des fonctions coûteuses a pu permettre le développement d'outils et d'accélérateurs spécifiques à notre application. Enfin, ce chapitre évalue les performances fréquentielles de l'algorithme spécifiquement

1. Traduction du terme anglais *"consistency"*, la cohérence définie l'intégrité du filtre par la mesure de l'adéquation entre l'erreur estimée et l'erreur réelle.

développé, avant de traiter de son portage sur un Virtex5 de Xilinx qui comporte un processeur PowerPC cadencé à 400MHz.

Chapitre 2

Vision : du Spectre à l'Image

2.1 Introduction

La vue est notre principal sens servant à nous localiser et à percevoir le monde extérieur. Nos autres sens y contribuent également mais dans une moindre mesure, en s'adaptant généralement lorsque la vue ne fournit plus assez d'informations. Les mal-voyants développent d'ailleurs des capacités sensorielles plus performantes avec l'ouïe ou le toucher.

Il est intéressant de noter que l'oreille interne apporte également sa part d'information car elle est l'organe de l'équilibre, puisqu'elle nous permet de percevoir l'axe vertical. Il est donc naturel de vouloir doter les systèmes mobiles d'un capteur identique à notre vue. Avec la miniaturisation, de plus en plus de systèmes autonomes sont équipés d'une ou plusieurs caméras qui transmettent une information accessible et de qualité. Même si ces capteurs ont une résolution bien plus faible que l'oeil humain, ils peuvent observer plus d'informations car équipés pour voir en dehors de l'unique spectre lumineux. Se pose ensuite le problème de l'extraction et de l'interprétation d'informations contenues dans les images, ce qui a donné naissance à une discipline scientifique, la *Vision par Ordinateur*. On estime le cerveau humain capable de faire environ 300 fois plus d'opérations qu'un ordinateur moderne.

Ce chapitre présente le processus de formation d'une image dans le cadre de la Vision par Ordinateur. Nous aurions pu nous limiter à la formation de l'image au niveau du capteur, mais dans le cadre de cette thèse, nous sommes amenés à traiter différentes composantes spectrales. Il nous parait donc nécessaire d'introduire dans le paragraphe suivant les bandes fréquentielles que nous allons étudier, à savoir le spectre visible et l'infrarouge.

Ces deux spectres nécessitent deux technologies d'acquisition différentes. Nous n'allons pas entrer dans les détails de chaque technologie, mais dans le cadre de l'étude de la chaîne d'acquisition d'une image, nous allons présenter brièvement comment ces capteurs fonctionnent.

Bien que les spectres analysés soient de natures différentes, les systèmes optiques d'une caméra visible et d'une caméra infrarouge peuvent être considérés d'une manière identique. Nous allons présenter enfin comment se modélise une caméra et son système optique afin de convertir une mesure en pixels en une observation sur la géométrie de la scène observée.

2.2 Le spectre visible et infrarouge

2.2.1 Présentation des spectres

Avant de nous intéresser à l'image, il importe de présenter en premier lieu le spectre électromagnétique. En effet, si le spectre visible colorimétrique est celui que nous connaissons car nous possédons le sens adapté à son observation, ce n'est pas le cas pour celui de l'infrarouge.

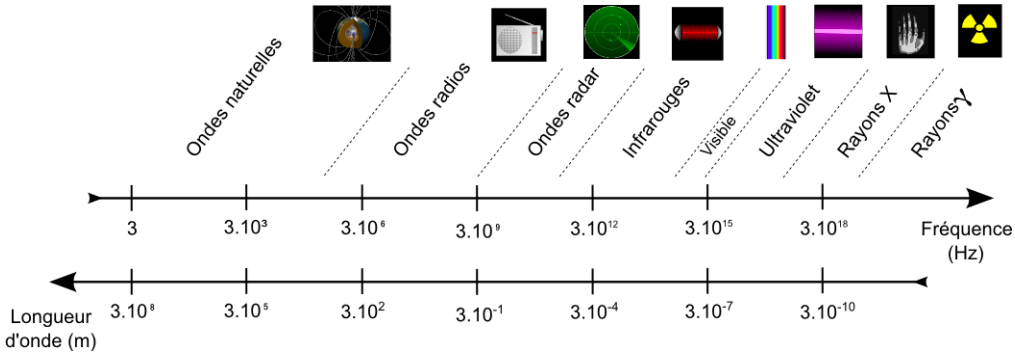


FIGURE 2.1 – Présentation des différentes ondes électromagnétiques, en fonction de leur fréquence et de leur longueur d'onde, ainsi que quelques exemples d'application.

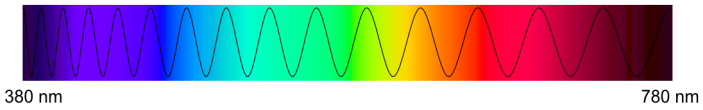


FIGURE 2.2 – Décomposition du spectre de la lumière visible en ses différentes couleurs qui la composent suivant la longueur d'onde.

Le rayonnement électromagnétique peut être décomposé en une composante corpusculaire (photons) et en une composante ondulatoire (onde électromagnétique). Certaines ondes que l'on rencontre au quotidien sont présentées en figure 2.1, telles que les ondes radios, les infrarouges, les ultraviolets, les rayons X, et certainement la plus observée d'entre-elles, l'onde lumineuse. Ces ondes sont regroupées dans différentes bandes fréquentielles qui peuvent ensuite être subdivisées en fonction de leurs propriétés. Si nous prenons l'exemple de la bande THF (pour *Tremendously High Frequency*), celle-ci est divisée en trois groupes :

- Les ondes *micrométriques*, également connues sous le nom d'ondes infrarouges,
- Les ondes *nanométriques*, regroupant le spectre visible et les ultraviolets,
- Les ondes *picométriques*, contenant, quant à elles, les rayons X et les rayons gamma.

Illustré en figure 2.2, le spectre visible fait donc partie de la famille des ondes nanométriques car sa longueur d'onde varie de 380 nm pour le violet à 780 nm pour le rouge.

La fréquence des ondes infrarouges est située juste en-dessous de celle du spectre visible, dont les longueurs d'ondes varient de 780 nm à 1 mm. Tout comme le spectre lumineux est divisé en plusieurs couleurs, on va distinguer trois bandes principales dans celui de l'infrarouge car un objet va avoir une émissivité différente dans chacune de ces bandes. Cependant, ce découpage peut varier en fonction de la norme ou du domaine d'étude (astronomie, télécommunications...). En imagerie thermique, ce découpage peut être attribué à des technologies d'acquisition différentes, et on discerne communément :

- L'*infrarouge proche*, allant de 780 nm à 1,4 μm, très utilisé en télédétection pour la différentiation des surfaces naturelles ou en détection d'incendie ou de points chauds au-delà de 300 °C, mais fournit des images de faible qualité pour des objets autour de 300 °K (température moyenne du corps humain) si aucune source, tel qu'un projecteur infrarouge, n'est utilisée,

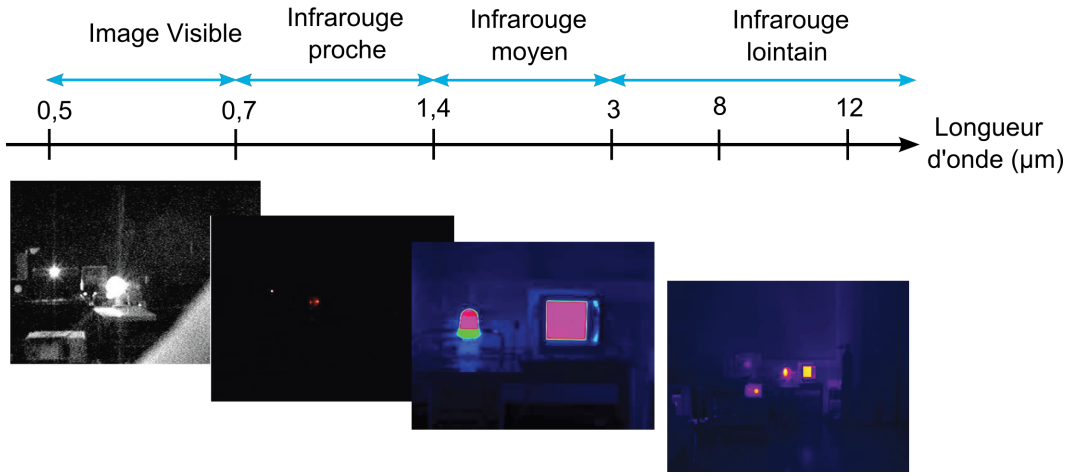


FIGURE 2.3 – Balise lumineuse marquant les bords de pistes d'un aéroport vu par différentes caméras captant le spectre visible à l'infrarouge lointain. Notons que les caméras étaient équipés d'optiques différentes.

- L'*infrarouge moyen*, allant de 1,4 μm à 3 μm, employé dans les applications militaires comme la reconnaissance de signatures thermiques pour le guidage des missiles, ou des applications de spectroscopie spatiale,
- L'*infrarouge lointain*, allant de 3 μm à 1 mm, également connu sous le nom d'*imagerie thermique*, servant à la détection des personnes de manière passive car la longueur d'onde d'émissivité du corps humain se situe aux alentours de 10 μm. La plupart des caméras thermiques sont sensibles aux longueurs variant de 8 μm à 12 μm.

D'une manière générale, à chacune des bandes correspond une technologie différente de capteur. Il est donc essentiel d'étudier la *signature thermique* d'un objet avant de concevoir un système destiné à l'observer. La balise présentée en figure 2.3 est dans un environnement sombre, à proximité d'un corps noir chauffé à 100 °C. Dans le cadre du projet SART, une pré-étude sur les capteurs les plus appropriés pour la détection de ce type d'objets, a conduit à choisir un système visuel multi-spectral, visible et infrarouge lointain.

2.2.2 Présentation des capteurs

Le capteur CCD (*Charge-Coupled Device*, figure 2.4 gauche) est composé d'une matrice de cellules photosensibles. Le nombre d'électrons collectés dans le *puits de potentiel* de chaque photosite est proportionnel à la quantité de lumière reçue pendant le temps d'exposition. A la fin de l'acquisition, les charges sont transférées à un collecteur, qui les transmettra à son tour à un convertisseur. Les diodes de celui-ci permettra de récupérer une tension proportionnelle au nombre d'électrons reçu.
Le capteur CMOS (*Complementarity metal-oxide-semiconductor*, figure 2.4 milieu) est lui aussi composé d'une matrice de photodiodes, où chacune d'entre-elle intègre son propre convertisseur analogique/numérique et amplificateur. Peu à peu, les capteurs CMOS ont remplacés les capteurs CCD, et à l'heure actuelle, ils sont bien plus répandus que les capteurs CCD de part leurs meilleures résolutions, qualité d'image et leurs coûts de fabrication inférieurs.
Il existe deux grandes familles de capteurs pour l'infrarouge thermique :

- Les capteurs refroidis,

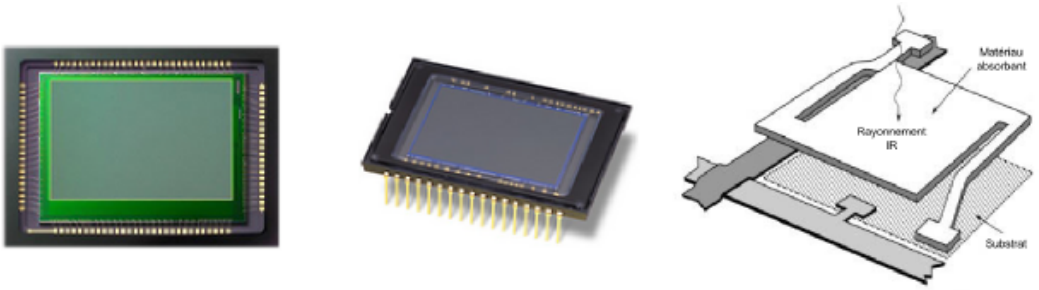


FIGURE 2.4 – Présentation des capteurs communément utilisés. A gauche un capteur CCD. Au centre, un capteur CMOS et à droite, un microbolomètre

– Les capteurs non-refroidis.

Les capteurs refroidis ont un principe de fonctionnement très similaire aux caméras classiques. La différence vient du type de matériaux utilisés pour la conception des photodiodes, où le tellurure de mercure-cadmium (*mercatel* ou *MCT*) est communément utilisé. Le refroidissement de ce capteur par un système cryogénique permet de supprimer les bruits de mesure induits par le réchauffement du capteur lors de son fonctionnement, et ainsi de détecter d'infimes différences de température dans la scène observée. Cependant, la conception complexe de ce capteur devant posséder une isolation thermique parfaite et nécessitant un système de refroidissement font de lui un capteur onéreux.

Les capteurs non-refroidis sont généralement composés d'une matrice de micro-bolomètres. Schématisé en figure 2.4 droite, les micro-bolomètres se composent d'un matériau capable d'absorber le rayonnement infrarouge (généralement à base d'oxyde de vanadium) qui fait varier la résistance électrique de celui-ci. La lecture du courant à ses bornes permet de déterminer la quantité de rayonnement reçu, et donc la température associée. Les caméras non refroidies sont généralement beaucoup moins chères que les caméras refroidies. Leur fabrication est plus rapide et nécessite moins d'étapes, et leur conditionnement sous vide est moins onéreux. Cette technologie a été adoptée pour le projet SART car elle ne nécessite pas de maintenance spécifique et est adaptée aux systèmes embarqués.

2.3 Modélisation des caméras

Notre application concerne la localisation d'un mobile (ici, un aéronef) avec précision à l'aide de la vision. Il est donc important de pouvoir décrire comment un indice visuel de la scène peut se transformer en un pixel de l'image, et inversement : à quoi correspond dans la scène un pixel de l'image. Cette partie décrit ces deux opérations présentées en figure 2.5 : la *projection* et la *rétro-projection*.

Les indices visuels de la scène sont appelés des amers ou objets remarquables. Ils sont facile à identifier dans une image, et l'ensemble des amers détectés dans un environnement forme une *carte*.

2.3.1 Projection d'un point

L'opération de projection a pour but de convertir un amer présent dans le monde 3D en un pixel de l'image. Les caméras imitant le fonctionnement de l'oeil humain, l'analogie avec celui-ci semble évidente. Tout comme la lumière passe au travers du cristallin avant d'activer les capteurs photosensibles de notre rétine qui sont les cônes et les bâtonnets, le rayonnement

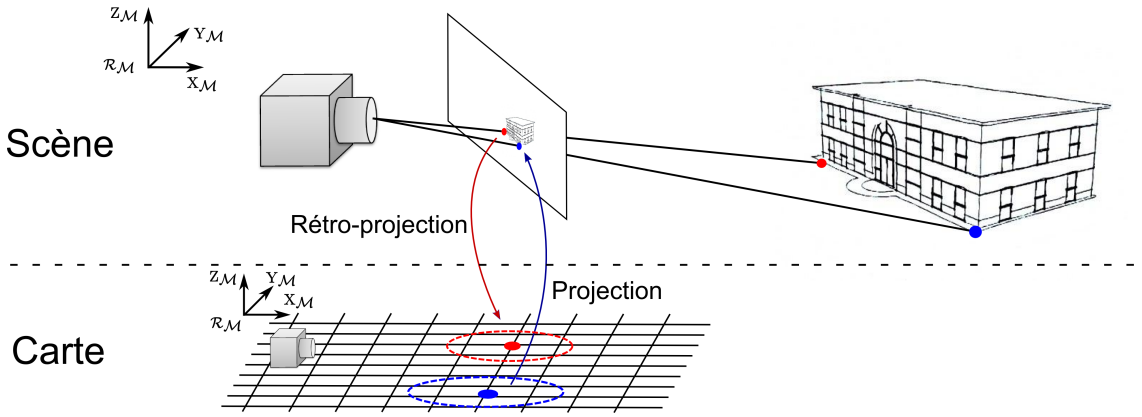


FIGURE 2.5 – L'opération de *projection* a pour but de situer l'image d'un amer référencé dans la carte. L'opération inverse, dite de *rétro-projection* a, quant à elle, pour objectif de positionner un amer dans la carte en fonction de ses coordonnées dans l'image. Notons que durant ces opérations, l'information de profondeur liée à un amer n'est pas directement observable dans l'image. Les ellipses dans la carte représentent cette incertitude sur la profondeur.

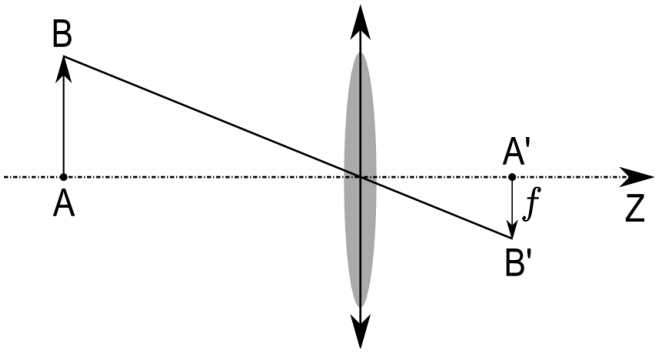


FIGURE 2.6 – Le modèle sténopé ou *pinhole* est le plus utilisé pour la représentation d'une caméra perspective. Dans cette modélisation, les rayons se propagent en ligne droite et les distorsions dues à la lentille et aux défauts du capteur ne sont pas prises en compte.

électromagnétique passe au centre de l'optique de la caméra avant d'arriver sur le capteur, qu'il soit développé pour des fréquences visibles ou infrarouges. Afin de décrire le processus de formation d'une image, c'est-à-dire de trouver la relation entre les coordonnées d'un point 3D de l'espace avec le point 2D associé sur le plan image, plus communément appelé *pixel*, nous devons modéliser la caméra.

Plusieurs modèles géométriques sont utilisés, les plus connus étant les modèles épais, mince et sténopé (ou *pinhole*). Dans les cas pratiques, et notamment avec des optiques standard (pas de zoom, pas de *fish-eye*), le modèle sténopé est suffisant.

En effet, simple et linéaire, il modélise le processus de formation des images. Comme schématisé en figure 2.6, il consiste en un *axe optique* orthogonal à un plan vertical qui contient schématiquement la lentille. Le *centre optique* est l'intersection de l'axe optique avec la lentille, où convergent les rayons. A la *distance focale* $\boldsymbol{f}$ se situe le *plan image*.

La transformation d'un point 3D $\mathbf{P}(X_M, Y_M, Z_M)^\mathsf{T}$ défini dans un repère spatial $\mathcal{R}_\mathcal{M}$ en un pixel 2D (u,v) défini dans le repère image $\mathcal{R}_\mathcal{I}$ se déroule en plusieurs étapes :

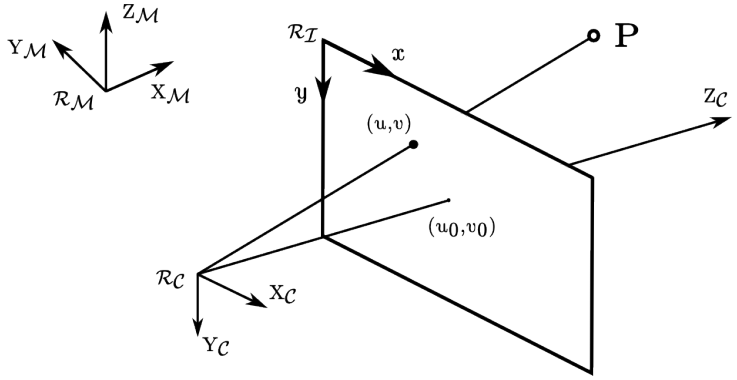


FIGURE 2.7 – Dans la représentation sténopé, un point **P** dans le repère monde $\mathcal{R}_{\mathcal{M}}$ se projette en un point p dans le repère image $\mathcal{R}_{\mathcal{I}}$

.

– Dans un premier temps, on exprime les coordonnées du point 3D dans un nouveau repère associé à la caméra à l'aide d'une matrice de passage homogène :

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix} = \begin{bmatrix} & & & t_x \\ & R_{3\times 3} & & t_y \\ & & & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_M \\ Y_M \\ Z_M \\ 1 \end{bmatrix} \tag{2.1}$$

où R et $(t_x \quad t_y \quad t_z)^\mathsf{T}$ sont respectivement la rotation et la translation entre le repère monde $\mathcal{R}_{\mathcal{M}}$ et le repère caméra $\mathcal{R}_{\mathcal{C}}$. Exactement, les colonnes de R donnent les coordonnées des vecteurs $O_M X_M$, $O_M Y_M$ et $O_M Z_M$, tandis que $(t_x \quad t_y \quad t_z)^\mathsf{T}$ donne les coordonnées de O_M par raport à $\mathcal{R}_{\mathcal{C}}$. Cette matrice est appelée *matrice* ***extrinsèque*** car elle dépend uniquement de paramètres externes à la caméra. Les repères sont présentés en figure 2.7.

– Ensuite, on projette le point défini dans le repère caméra sur le plan image. Cette projection est donc une application injective de $\mathbb{R}^3$ dans $\mathbb{R}^2$. A l'inverse, un point sur le plan image correspond à un rayon dans l'espace. La projection du point 3D dans le plan image est défini par :

$$\begin{bmatrix} x_I \\ y_I \\ 1 \end{bmatrix} \sim s \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix} \tag{2.2}$$

où s est un facteur d'échelle qui correspond à la profondeur du point 3D observé.

– A ce stade, le point est projeté sur le plan image qui est supposé continu et infini. En pratique, le capteur d'une caméra est composé d'une matrice finie d'éléments photosensibles. La dernière étape va donc consister en l'échantillonnage du plan image pour récupérer la coordonnée du pixel (u,v) :

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} s_u & s_\theta & u_0 \\ 0 & s_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_I \\ y_I \\ 1 \end{bmatrix} \tag{2.3}$$

où u_0 et v_0 sont les coordonnées en pixels de l'intersection entre l'axe optique et le plan image (le centre de l'image, ou point principal), s_u et s_v représentent le nombre de pixels

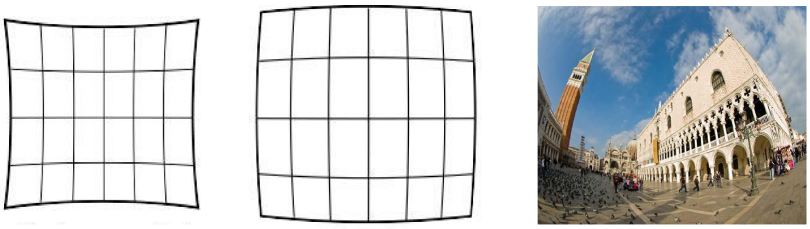


FIGURE 2.8 – A gauche une distorsion de type coussinet et au centre de type barillet. A droite, une photo de Venise soumise à une forte distorsion optique de ce dernier type. Cette distorsion, due à une focale courte, est commune pour les caméras destinées à être embarquées ou à large champ de vue car l'optique est plus facilement miniaturisable. Les optiques de type *fisheye* ont une distorsion poussée à l'extrême.

par unité de longueur et s_θ est une caractéristique liée à la rectangularité du pixel. Ce paramètre décrit donc la possible déformation trapézoïdale du pixel qui pourrait résulter d'un mauvais montage de la caméra (axe optique non orthogonal au plan image), mais est généralement nul.
Couramment, on regroupe la matrice de projection 2.2 et celle d'échantillonnage 2.3 en une seule pour former la *matrice* ***intrinsèque***. Cette dernière contient l'ensemble les paramètres relatifs à l'optique et au capteur de la caméra :

$$\mathbf{K} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_u & 0 & u_0 \\ 0 & s_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

Dès lors, le passage d'un point P de l'espace à un pixel dans l'image peut être uniquement écrit avec la matrice extrinsèque et la matrice intrinsèque :

$$\begin{bmatrix} s \cdot u \\ s \cdot v \\ s \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 & 0 \\ 0 & \alpha_v & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} & R_{3\times3} & & t_x \\ & & & t_y \\ & & & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_M \\ Y_M \\ Z_M \\ 1 \end{bmatrix} \tag{2.5}$$

ou :

$$\begin{bmatrix} s \cdot u \\ s \cdot v \\ s \end{bmatrix} = \mathbf{K} \cdot R \cdot \begin{bmatrix} X_M \\ Y_M \\ Z_M \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \tag{2.6}$$

Néanmoins, nous avons supposé ici que le modèle optique était parfait et que les rayons se propageaient en ligne droite au travers du système optique. Le montage du capteur sur la caméra n'étant pas parfait et la lentille engendrant naturellement des distorsions radiales, nous observons des défauts dans l'image. Comme l'illustre la figure 2.8, deux types de distorsions peuvent être observées : la distorsion en *barillet* et celle en *coussinet*. Pour prendre en compte cette distorsion, nous devons ajouter un terme correcteur avant l'échantillonnage du point. La distorsion est modélisée par une fonction polynomiale d'ordre n. Dans la plupart des cas, il faut estimer k_n jusqu'à l'ordre 2 (polynôme de degré 5) et quelques fois jusqu'à l'ordre 3 (polynôme de degré 7) dans les cas de très grandes *distorsions radiales* :

$$\mathbf{x}_d = \begin{bmatrix} x_d \\ y_d \end{bmatrix} = d(\mathbf{x}_I) = (1 + k_1 r^2 + k_2 r^4 + \dots) \begin{bmatrix} x_I \\ y_I \end{bmatrix} \tag{2.7}$$

avec :

$$r^2 = {x_I}^2 + {y_i}^2$$

Notons que cette équation permet de calculer la position distordue d'un pixel à partir de sa position parfaite, vis-à-vis du modèle sténopé. Comme c'est cette position distordue qui est observée, il est nécessaire d'inverser cette équation pour corriger les coordonnées d'un pixel (voir équation 2.9).

Il existe d'autres types de distorsions (prismatiques, tangentielles...), qui généralement, sont négligeables pour les applications robotiques.

2.3.2 Rétro-projection d'un point

La rétro-projection, processus inverse de la projection, permet de convertir un pixel de l'image en son équivalent dans le monde 3D. Cette étape peut se diviser en trois temps :

- La dépixellisation,
- La correction des distorsions,
- La rétro-projection par une caméra normalisée.

La dépixellisation

La dépixellisation est le processus qui permet de convertir une valeur échantillonnée en son équivalent métrique dans le plan image. Cette opération reste relativement simple car elle consiste en l'inversion de la matrice des paramètres intrinsèques $\mathbf{K}$:

$$\mathbf{x}_d = \begin{bmatrix} x_d \\ y_d \\ 1 \end{bmatrix} = \mathbf{K}^{-1} \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{2.8}$$

avec

$$\mathbf{K}^{-1} = \begin{bmatrix} 1/\alpha_u & -\alpha_\theta/(\alpha_u\alpha_v) & (\alpha_\theta v_0 - \alpha_v u_0/(\alpha_u\alpha_v) \\ 0 & 1/\alpha_v & -v_0/\alpha_v \\ 0 & 0 & 1 \end{bmatrix}$$

La correction

La correction consiste en l'étape inverse de la distorsion de l'image présentée en figure 2.8. Cette étape est beaucoup plus délicate du fait de la nature non-linéaire du modèle de distorsion et de son expression donnant la position distordue en fonction de la position corrigée. Il est possible de résoudre l'équation 2.8 pour calculer (x_I, y_I), mais à l'ordre 3, avec un polynôme de degré 7. Aussi, pour corriger ces distorsions, nous utilisons la solution séduisante proposée par Solà dans sa thèse [Solà, 2007] qui consiste en l'utilisation d'un modèle de correction qui approxime l'inverse du modèle de distorsion :

$$\mathbf{x}_I = c(\mathbf{x}_d) = (1 + c_2 r_d^2 + c_4 r_d^4 + \cdots)\mathbf{x}_d \tag{2.9}$$

avec

$$r_d{}^2 = x_d{}^2 + y_d{}^2$$

Les paramètres de ce modèle sont calculés durant la phase hors-ligne de calibrage, par la minimisation au sens des moindres carrés de la fonction d'erreur :

$$f(\mathbf{c}) = \sum_{i=1}^{N} [\mathbf{x}_d - d(c(\mathbf{x}_d))]^2$$

La rétro-projection

Une fois les coordonnées métriques corrigées $\mathbf{x}_I$ du point dans le plan image obtenues, il suffit de les rétro-projeter dans le monde pour obtenir le point $\mathbf{P}$, qui se situera sur la droite formée par le centre optique et $\mathbf{x}_I$. La distance entre le centre optique et $\mathbf{P}$ est donnée par s, qui n'est pas observable directement par une seule caméra. L'opération de rétro-projection peut se formuler selon :

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \end{bmatrix} = s \begin{bmatrix} x_I \\ y_I \\ 1 \end{bmatrix} \tag{2.10}$$

2.4 Conclusion

Dans ce chapitre, nous avons présenté les notions principales de la Vision par Ordinateur, qui exposent comment l'image se forme sur un capteur d'une caméra. Nous avons évoqué le problème des distorsions qui perturbent les applications pour lesquelles la précision est primordiale, telle que la métrologie ou les méthodes de localisation par vision. Nous avons également introduit les opérations de projection d'un amer et rétro-projection d'un pixel entre l'image et la carte.

Cependant, les points ne sont pas les seules primitives à pouvoir être utilisées dans les applications de SLAM par vision. Smith *et al.* [Smith et al., 2006] et Lemaire *et al.* [Lemaire and Lacroix, 2007] réalisent un SLAM à partir de segments extraits d'une image afin de générer une carte de droites 3D. Dans [Sola, 2010], Solà étudie la cohérence d'un SLAM basé sur un Filtre de Kalman étendu (EKF) avec différents types de paramétrisations pour des points. Dans [Solà et al., 2012], il étend son étude aux lignes.

Berger *et al.* [Berger et al., 2007] font un pas de plus dans la représentation en utilisant des facettes issues de l'image à l'aide d'une application de stéréovision. Une méthode de SLAM basée sur plusieurs plans à partir d'images éparses a été présentée par Argiles *et al.* [Argiles et al., 2011].

Bien que ces approches soient intéressantes, elles demandent plus de pré-traitements et de calculs de la part de l'algorithme, qui peut les rendre incompatibles avec les contraintes temps réel. De manière à garantir de bonnes performances d'exécution sur les systèmes embarqués, nous nous concentrerons sur des amers de type points 3D, perçus dans l'image par des points d'intérêt. Nous verrons en chapitre 4 comment ces points d'intérêt sont identifiés dans l'image et quelles paramétrisations sont proposées pour initialiser sans retard des amers ponctuels 3D dans une approche de type SLAM EKF. Nous évoquerons également l'un des problèmes essentiel du SLAM visuel qui est le suivi et la mise en correspondance des amers. Nous présenterons

les algorithmes permettant d'accélérer grandement ces deux étapes, algorithmes principalement basés sur l'incertitude de localisation de l'amer lors de sa projection dans l'image.

Avant cela, le chapitre suivant introduit le filtre de Kalman, son extension aux systèmes non-linéaires et son application dans le cadre du SLAM.

Chapitre 3

Du Filtrage au SLAM

3.1 Introduction

3.1.1 Localisation et Cartographie Simultanées

SLAM est l'acronyme anglais pour *Simultaneous Localization And Mapping*, soit en français *Localisation et Cartographie Simultanées*. Au même instant, le système essaie de se localiser et de cartographier son environnement. Il se localise sur la base de la carte courante, et il complète cette carte de manière incrémentale. Il pourra mieux se repérer s'il passe par des lieux déjà visités.

Tout comme nous essayons de nous localiser avec des points d'intérêt persistants quand nous visitons une nouvelle ville (clocher d'église, place de la Mairie et son café du Midi...), nous faisons en sorte qu'un système autonome naviguant adopte ce même comportement. Il détecte des zones caractéristiques avec ses capteurs, zones qui ont peu de chance d'évoluer au fil du temps et qu'il est capable de suivre ou de reconnaître dans les perceptions suivantes afin de pouvoir se localiser quelle que soit sa position.

La localisation est un élément essentiel pour un robot autonome ou pour un système mobile équipé de capteurs pour percevoir l'environnement (un humain, un véhicule, un aéronef...). Qu'il soit développé pour l'aide à la personne, la reconnaissance d'itinéraires, l'exploration spatiale, ou pour toute autre application, le robot doit à chaque instant connaître précisément sa position. Si un robot est mal localisé, il peut faire un mouvement dangereux pour un humain ou pour lui-même. Dans notre application, la localisation tient également une place très importante : si l'avion est localisé quelques mètres à côté de sa véritable position, c'est la sortie de piste assurée...

Un exemple est illustré en figure 3.1. A chaque instant, le robot perçoit des amers (notés $\mathcal{L}$ pour la formulation anglaise *Landmark*). S'il a déjà connaissance de ceux-ci, c'est-à-dire s'ils sont déjà dans la carte, il les observe à nouveau pour se localiser et pour affiner sa connaissance du monde, sinon il les initialise dans sa carte pour essayer des les réobserver aux itérations suivantes. Le principe est donc de localiser au mieux notre système à chaque instant.

Le SLAM est donc la méthode que nous allons utiliser pour la localisation de notre système. Dans ce chapitre, nous présentons les différentes approches existantes pour le SLAM et détaillons celle que nous allons implémenter, basée sur un Filtre de Kalman Etendu. Après un bref rappel sur la théorie du filtre de Kalman et de son extension aux systèmes non-linéaires, nous appliquerons ce dernier au SLAM par vision monoculaire.

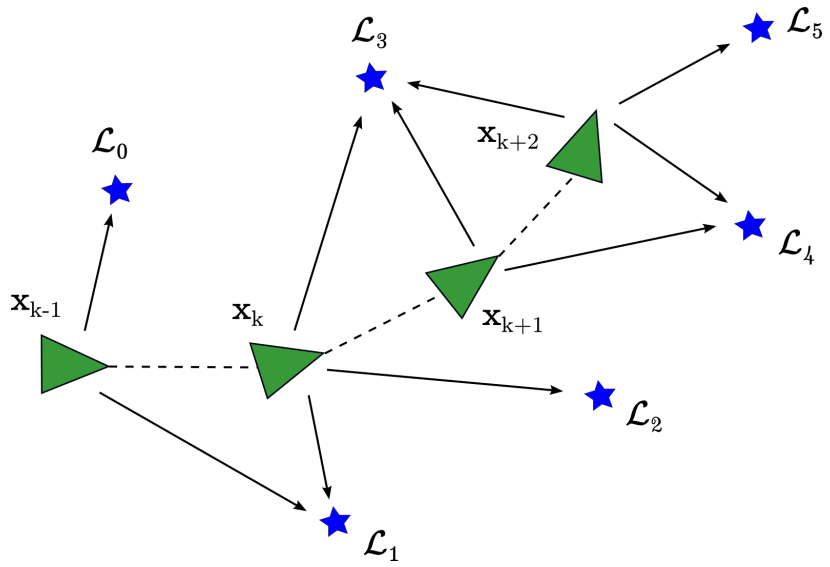


FIGURE 3.1 – Illustration du principe du SLAM. L'objectif est d'améliorer la connaissance sur la position de notre système. Pour cela, à chaque instant, il perçoit des percepts remarquables, ou *amers*, grâce à ses capteurs. Si ce sont des amers qu'il a déjà observés auparavant et donc qui font partie de sa carte, il effectue une *réobservation* de ceux-ci. Ainsi, il affine simultanément sa connaissance sur sa position et sur celle de ces amers. Si ce sont des amers qu'il observe pour la première fois, il les intègre dans sa carte dans le but de les réobserver durant les itérations suivantes.

3.1.2 Le SLAM probabiliste

Depuis plusieurs années, la demande en localisation est forte. Les premiers travaux dans ce domaine ont été réalisés par Smith et Cheeseman [Smith et al., 1988] qui ont proposé l'utilisation d'un Filtre de Kalman Etendu ou EKF pour *Extended Kalman Filter* pour la problématique du SLAM. Les méthodes se sont ensuite grandement développées, notamment autour de capteurs tels que des lasers ou des sonars [Moutarlier and Chatila, 1989].

Initialement, la vision n'était pas considérée car jugée trop complexe, bien que la caméra soit le capteur qui fournisse les informations les plus riches pour un robot. Les applications de SLAM concernaient surtout les télémètres laser [Csorba, 2007], [Betge-Brezetz et al., 1996]. Cependant, avec les progrès de la Vision par Ordinateur, le SLAM visuel connut une grande avancée en 2003 grâce aux travaux de Davison [Davison, 2003] qui fit tourner son application en temps réel sur un ordinateur grand public mais pour de petites scènes. Depuis, la caméra est considérée comme un capteur quasi-indispensable dans les applications de SLAM. En 2007, Davison développe le MonoSLAM [Davison et al., 2007] qu'il embarque avec succès sur un robot humanoïde, pour de la localisation d'intérieur sur de courtes trajectoires. La caméra est souvent couplée à d'autres capteurs (odomètres, capteurs inertiels...) pour essayer de pallier son principal inconvénient qui est l'inobservabilité de la profondeur. En effet, cette contrainte a conduit à l'introduction de méthodes d'initialisation avec retard (un amer est créé quand sa position 3D peut être estimée) ou sans retard (un amer est créé dès qu'il est observé). Pour ce dernier cas, l'amer adopte une représentation adaptée : somme de gaussiennes [Sola et al., 2005], inverse de la profondeur [Montiel, 2006] ou diverses variantes [Sola, 2010].

Nous avons participé au développement de *RT-SLAM*, où Roussillon *et al.* traitent des images provenant d'une caméra 640×480 cadencée à 60Hz en temps réel sur un ordinateur

de bureau [Roussillon et al., 2011]. RT-SLAM intègre également différents capteurs tels que des odomètres, centrale inertielle ou GPS et il permet une localisation avec une erreur que de quelques mètres après plusieurs centaines de mètres parcourus à bord d'un robot d'exploration.

Même si le SLAM EKF perd aujourd'hui du terrain face à des approches plus innovantes orientées optimisation et prenant en entrée un graphe (*Graph-Based SLAM*), il reste toujours le sujet de diverses études, améliorations et implémentations. Ces études portent principalement sur des approches permettant de pallier les limitations du SLAM EKF ; ce dernier est actuellement considéré comme la meilleure approche pour les systèmes embarqués, généralement pourvus de faibles ressources de calcul.

3.1.3 Limitations du SLAM EKF

Le SLAM fondé sur le filtre de Kalman souffre de deux inconvénients majeurs qui sont la **complexité** et la **cohérence**.

La complexité

Pour que le filtre garde une certaine cohérence, il doit intégrer, dans un *vecteur d'état*, les positions du système mobile ainsi que tous les amers identifiés par les capteurs. Les incertitudes sur ces positions ainsi que les corrélations entre-elles doivent également être considérées et sont définies dans une *matrice de covariance*. On saisit bien la difficulté qui va paraître lorsque le nombre d'amers va croître dans la *carte* SLAM qui regroupe le vecteur d'état et la matrice de covariance.

Dans sa thèse, Solà [Solà, 2007] évoque le problème de complexité algorithmique du SLAM. Il évalue la complexité globale du SLAM EKF en $\mathcal{O}(k \cdot n^2)$ avec n le nombre d'amers dans la carte et k le nombre d'amers observés à chaque itération de l'algorithme.

Plusieurs solutions ont été proposées pour réduire cette dépendance au nombre d'amers. La solution proposée par Dissanayake *et al.* [Dissanayake et al., 2001] et adoptée par les dernières approches en SLAM EKF ([Roussillon et al., 2011], [Gonzalez et al., 2011]) consiste à ne retenir que les derniers amers observés. Une fois les amers en dehors du champ de perception du capteur, ils sont supprimés de la carte. Plus exactement, ils sont gardés durant quelques itérations dès qu'ils ne sont plus observés, au cas où cette non identification soit seulement due à une occultation locale. Dans ce cas, ils vont réapparaître dans le champ de vue du capteur. Bien que cette approche permette de réaliser des trajectoires de plusieurs centaines de mètres, elle n'offre pas une localisation précise à cause de l'oubli des lieux par lesquels le mobile est déjà passé. Les approches exploitant ces techniques "d'oubli" sont souvent désignées par *SLAM court-terme*, en opposition au *SLAM long-terme*, capable de détecter une fermeture de boucles lorsque le robot revient sur un endroit déjà exploré.

Une autre solution proposée par Rodriguez-Losada *et al.* [Rodriguez-Losada et al., 2004] est la division de l'espace cartographié en sous-cartes. Cette méthode, nommée *Multi-map*, permet de limiter le nombre d'amers par carte et peut se décomposer en deux approches. Dans [Chong and Kleeman, 1999], une nouvelle sous-carte est créée lorsque l'incertitude globale devient trop importante. Leonard *et al.* [Leonard and Feder, 2001] proposent la création d'une sous-carte lorsque la précédente atteint un nombre d'amers limite. Cette approche multi-cartes est bien adaptée pour le SLAM multi-robots [Vidal-Calleja et al., 2011], [Marquez-Gamez and Devy, 2011]. Son problème principal est le maintien de la cohérence lors de la transition entre les différentes sous-cartes. Les approches inspirées du *Hierarchical SLAM* [Bulata and Devy, 1996], [Estrada et al., 2005] rajoutent une étape d'optimisation hors-ligne pour réestimer les positions relatives inter-cartes, ainsi éventuellement, que les positions des amers perçus dans

plusieurs cartes.

Thrun *et al.* [Thrun et al., 2004] proposent le *Sparse Extended Information Filter* qui utilise le caractère épars de la matrice de covariance en ne sélectionnant, pour la mise à jour du système, que les amers ayant un impact important sur l'état.

En 1997, Csorba [Csorba, 2007] propose un SLAM *robot-centré*. Le système reste donc virtuellement immobile et c'est son environnement qui se déplace. Une telle approche permet de réduire la complexité de l'étape de prédiction du filtre et permet l'obtention d'une matrice de covariance diagonale mais elle élève au carré le nombre d'états du vecteur.

La cohérence

Le second inconvénient du filtre de Kalman est son incohérence. Bien que celui-ci soit optimal pour les systèmes linéaires affectés par des bruits gaussiens, il engendre des problèmes de cohérence dus à la linéarisation autour de la valeur estimée, dans le cas des systèmes non-linéaires. Cette cohérence a été étudiée par Julier *et al.* [Julier and Uhlmann, 2001]. Ils ont démontré que la linéarisation de la fonction d'observation peut engendrer des corrections abusives rendant alors le filtre incohérent. L'observation des amers de la carte est donc un point primordial du SLAM EKF et de nombreux efforts ont été réalisés pour filtrer au mieux les points incohérents. Dans le cas de l'EKF SLAM monoculaire, plusieurs méthodes ont été mises en oeuvre pour filtrer les observations incohérentes dues à de faux appariements, tels que l'utilisation de nouvelles paramétrisations d'amers [Montiel, 2006] [Sola, 2010] et le calcul d'un score de corrélation par ZNCC [Stefano et al., 2005]. En 2009, Civera *et al.* [Civera et al., 2009] proposent l'algorithme One-Point RANSAC qui ne sélectionne que les amers ayant peu de chance de rendre incohérent le filtre de Kalman, c'est-à-dire ceux dont l'incertitude est très faible.

Concernant le SLAM, et plus particulièrement la localisation, ces problèmes de cohérence se répercutent sur l'intégrité de la position du système mobile, et au-delà, des positions des amers. La localisation est intégrée si l'estimée (de sa position et son incertitude) est compatible avec la position réelle du système, c'est-à-dire que la probabilité que cette position réelle soit en dehors de l'ellipsoïde d'incertitude autour de la position estimée est inférieure à une valeur déterminée. On parle alors *d'intégrité* (à 10^{-5}, 10^{-7}...).

Enfin rappelons que dans le cas général, l'erreur sur la position obtenue par le SLAM EKF ne peut qu'augmenter : pour éviter cette dérive, il faut détecter les fermetures de boucle ou fusionner des observations de position absolue (GPS en extérieur ou *map matching* quand on dispose de connaissances a priori sur l'environnement).

Ces inconvénients ont poussé la communauté SLAM à se tourner vers de nouvelles approches.

3.1.4 Les différentes approches

Malgré ses défauts de complexité et de cohérence, l'EKF a été longuement étudié et est toujours très utilisé. Cependant, d'autres solutions ont été explorées afin que le SLAM ne soit plus sujet à l'incohérence et à une trop grande complexité en fonction du nombre d'amers détectés.

L'une des approches considérée est l'utilisation d'un filtre particulaire à la place d'un EKF. Montemerlo *et al.* [Montemerlo et al., 2002] développent l'algorithme FastSLAM basé sur cette méthode. Bien qu'il réduise la complexité grâce à la décorrélation entre les erreurs sur la position du robot et des amers, il fait apparaître un nouveau problème sur la sélection du nombre de particules à définir qui est corrélée à la longueur de la trajectoire. Par ailleurs, le problème de cohérence est toujours présent puisque, lors de l'étape de rééchantillonnage, les

particules subissent une dégénérescence et oublient de manière exponentielle leur passé. Un avantage du filtrage particulaire (PF) vis-à-vis de filtrage de Kalman est l'estimation de la position initiale du robot : le SLAM EKF a besoin d'une position estimée et échoue dès que le filtrage diverge. Le SLAM PF peut traiter le problème du *kidnapping* [Zhang et al., 2009].

Une autre méthode, proposée par Di Marco *et al.* [Di Marco et al., 2001], se base sur l'analyse par intervalles. Cette approche ensembliste n'est pas affectée par les erreurs de linéarisation ou d'échantillonnage et les résultats sont cohérents mais au prix d'un pessimisme sur la méthode obtenue. L'idée est de supposer que l'on dispose d'une équation de modèle et de mesures affectées par des bruits bornés dont un intervalle contenant les bornes est connu. Cette supposition est le principal défaut de cette méthode qui est dès lors très sensible aux données aberrantes. De plus, pour l'obtention de bons résultats, il est nécessaire de corréler les informations issues de plusieurs capteurs différents afin de pouvoir réduire de manière efficace les bornes du système et donc d'augmenter la précision globale. Notons toutefois en France les travaux de Luc Jaulin qui a développé l'algorithme SIVIA (Set Inverter Via Interval Analysis) [Jaulin and Walter, 1993] et le met en application pour le SLAM de robots sous-marins [Jaulin, 2009]. Cet algorithme a été implémenté dans la bibliothèque *IBEX* [Chabert et al., 2012], celle-ci développée par une équipe trans-laboratoire, centrée sur l'Ecole des Mines de Nantes.

Mais la méthode qui suscite le plus d'intérêt aujourd'hui est l'approche par optimisation de graphes. Le GraphSLAM, proposé par Thrun *et al.* [Thrun and Montemerlo, 2005], consiste en un algorithme dit de *Smoothing And Mapping* (SAM) dont le but est de lisser l'ensemble de la trajectoire sous hypothèse gaussienne. Son principe est de définir un état qui contient toutes les positions depuis l'instant initial ainsi que la position des amers. La matrice d'information du système est alors construite de manière incrémentale. Pour retrouver la position du robot et des amers, il faut marginaliser cette matrice, lui rajoutant ainsi des termes. La même année, Dellaert *et al.* [Dellaert and Kaess, 2005] proposent une approche optimisée du SAM, basée sur la factorisation en racine carrée de la matrice d'information et de la jacobienne de la mesure. Cet algorithme fait parti de la famille *Square Root SAM*, ou $\sqrt{\text{SAM}}$. En 2008, Kaess *et al.* [Kaess et al., 2008] proposent une amélioration avec iSAM, basé sur une factorisation QR rapide et incrémentale de la matrice d'information.

Ces méthodes *Graph SLAM* se déclinent en *Pose-Based SLAM* (optimisation de la trajectoire seulement) et *Landmark-Based SLAM* (optimisation de la trajectoire et des amers). Elles s'appliquent quelque soit la nature des observations. Concernant le SLAM visuel, ces méthodes d'optimisation incrémentale s'apparentent aux approches *d'appariement de faisceaux* (*Bundle Adjustment*) bien connues dans la communauté Vision depuis les travaux de Triggs *et al.* [Triggs et al., 2000] et Lourakis *et al.* [Lourakis and Argyros, 2004], popularisés en France par Mouragnon *et al.* [Mouragnon et al., 2009].

Le défaut de ces algorithmes est le temps de traitement, même si Klein *et al.* [Klein and Murray, 2007] ont démontré qu'il était possible d'utiliser ce type d'algorithme sur des systèmes ayant des ressources limitées (un iPhone dans leur cas) pour cartographier de petites scènes.

Bien que ces algorithmes de SLAM basé sur l'optimisation ont grandement évolués ces dernières années, ils demandent des machines modernes, et peu fonctionnent aujourd'hui en temps réel sur des systèmes embarqués. Dans [Strasdat et al., 2010], Strasdat *et al.* concluent que les approches par optimisation de graphes sont les plus performantes sauf dans le cas où l'on dispose de ressources de calcul limitées. Dans ce cas, il est nécessaire d'envisager une solution de SLAM par filtrage.

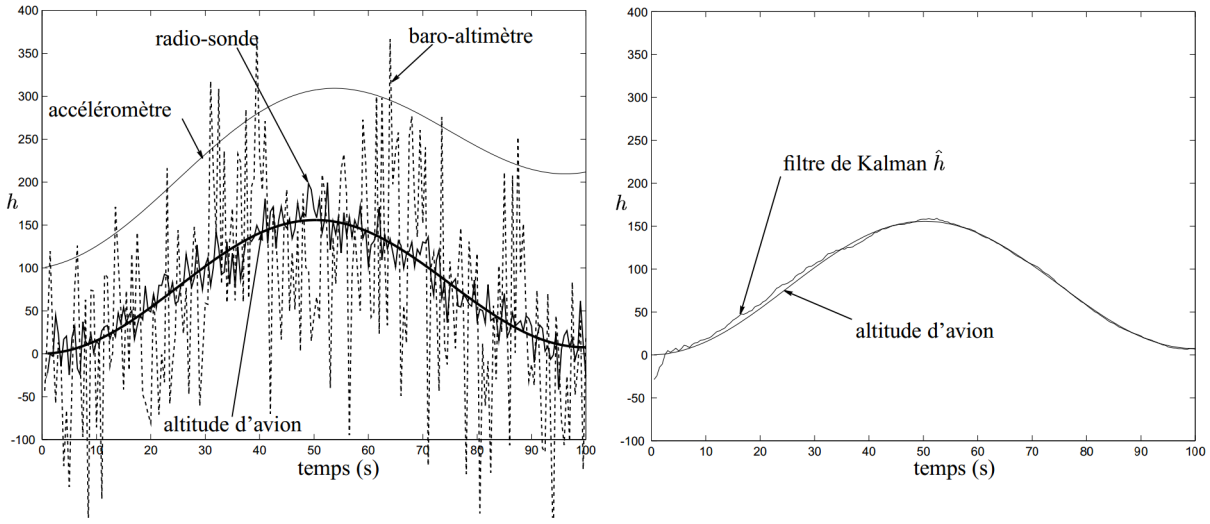


FIGURE 3.2 – Exemple de filtrage présenté dans [Nikiforov, 2000] pour une application de navigation hybride. A gauche, un ensemble de données provenant de capteurs mesurant l'altitude d'un avion, dont un accéléromètre, un baro-altimètre et une radio-sonde. Ces mesures sont bruitées et/ou biaisées. A droite, la comparaison entre le résultat issu du filtre de Kalman et l'altitude réelle. Ce résultat est intègre.

3.2 Le Filtrage

Le filtrage est l'action d'estimer un état en utilisant au mieux l'ensemble des informations que peut nous fournir un système et/ou l'observation de celui-ci. Nous nous intéressons ici à des systèmes dynamiques, et plus particulièrement à l'état d'un système mobile.

Un système considéré peut nous donner une évaluation de son état par des capteurs observant ses paramètres internes. On parlera alors de capteurs *proprioceptifs*. En prenant l'exemple d'une voiture en mouvement, on peut connaître sa vitesse en regardant simplement son compteur. On peut alors connaitre sa position en intégrant cette vitesse. En robotique, la navigation d'un robot fondée sur de telles mesures est qualifiée de *dead-reckoning*. Ce mode de navigation est sujet à des erreurs cumulatives.

Un système mobile peut embarquer des capteurs qui donnent des observations sur l'environnement : il s'agit de capteurs dit *extéroceptifs*, comme une caméra. Ces capteurs peuvent également estimer l'état d'un système en l'observant depuis des points fixes dans l'environnement, ou embarqués sur d'autres systèmes mobiles (radars routier...).

Comme un capteur n'est jamais parfait, la mesure qu'il indique est bruitée et peut également être biaisée. Filtrer l'ensemble des informations disponibles nous permettra de connaître bien plus précisément l'état estimé de notre système. Un exemple est illustré en figure 3.2 où l'objectif est de connaître avec précision l'altitude d'un avion à partir de mesures bruitées et/ou biaisées.

Dans cette section, nous présentons brièvement le concept et la formulation associée au filtrage. Ne faisant qu'aborder la théorie, nous reprenons la notation utilisée dans [Solà, 2007], dont l'ouvrage développe très clairement la formulation du filtre de Kalman et de son extension aux systèmes non-linéaires.

3.2.1 Le filtre de Kalman

Le filtre de Kalman [Kalman, 1960] estime les états d'un système dynamique à partir de mesures incomplètes ou bruitées. C'est l'estimateur optimal pour les systèmes linéaires ayant des distributions gaussiennes. De ce fait, il est utilisé dans de nombreuses technologies : radar, communications, navigation, météorologie, finance...

Grâce à une connaissance de la dynamique de la cible, l'effet du bruit peut être éliminé. Ces données peuvent être calculées pour l'instant présent (filtrage), pour le passé (lissage), ou pour une prévision de l'état futur (prédiction).

Le filtre de Kalman est un estimateur récursif, c'est-à-dire qu'il a uniquement besoin de connaître son état précédent et les mesures en cours (bruitées par définition) pour estimer l'état courant.

Son état est représenté par deux variables :
- $\hat{\mathbf{x}}_k$ qui est le vecteur contenant les valeurs moyennes de l'état du système à l'instant k,
- $\mathbf{P}$ qui est la matrice de covariance de $\mathbf{x}_k$ à l'instant k et représente l'erreur d'estimation .

De plus, le filtre utilise deux fonctions :
- $\mathbf{x}_k = \mathbf{f}_k(\mathbf{x}_{k-1}, w_k)$ qui est la *fonction d'évolution* du système,
- $\mathbf{y}_k = \mathbf{h}_k(\mathbf{x}_k, v_k)$ qui est la *fonction d'observation* du système.

Considérons alors le système linéaire gaussien suivant :

$$\begin{aligned} \mathbf{x}_k &= \mathbf{F}\mathbf{x}_{k-1} + \mathbf{G}w_k \\ \mathbf{y}_k &= \mathbf{H}\mathbf{x}_{k-1} + v_k \\ p(\mathbf{x}_0) &= \mathcal{N}(\mathbf{x}_0 - \hat{\mathbf{x}}_0; \mathbf{P}_0) \\ p(w_k) &= \mathcal{N}(w_k ; \mathbf{Q}) \\ p(v_k) &= \mathcal{N}(v_k ; \mathbf{R}) \end{aligned}$$

où $\mathcal{N}(\mathbf{x} - \hat{\mathbf{x}}; \mathbf{P})$ est une fonction gaussienne de variable $\mathbf{x}$, de moyenne $\hat{\mathbf{x}}$ et de covariance $\mathbf{P}$, $p(\mathbf{x}_0)$ est la densité de probabilité relative à l'état initial du système, $p(w_k)$ la densité de probabilité relative au contrôle du système et $p(v_k)$ la densité de probabilité relative au bruit de mesure.

La philosophie de ce filtre consiste à réaliser une prédiction de son futur état grâce à la connaissance de son état actuel et de la dynamique du système donnée par la fonction d'évolution $\mathbf{f}(\cdot)$.

A l'instant suivant, le filtre évalue son erreur entre sa prédiction et son état réel donné à partir des mesures des différents capteurs. L'évaluation de cette erreur doit lui permettre de réaliser ainsi une estimation de plus en plus fiable pour finalement converger vers la position réelle du système malgré des mesures bruitées.

Le filtre de Kalman se décompose donc en deux étapes : la **prédiction** et la **correction**.

La Prédiction

L'étape de prédiction est donnée par :

$$\begin{aligned} \hat{\mathbf{x}}_{k|k-1} &= \mathbf{F}\hat{\mathbf{x}}_{k-1|k-1} \\ \mathbf{P}_{k|k-1} &= \mathbf{F}\mathbf{P}_{k-1|k-1}\mathbf{F}^\mathsf{T} + \mathbf{G}\mathbf{Q}\mathbf{G}^\mathsf{T} \end{aligned}$$

Ce système se récrit avec la notation simplifiée :

$$\begin{aligned} \hat{\mathbf{x}}^+ &= \mathbf{F}\hat{\mathbf{x}} \\ \mathbf{P}^+ &= \mathbf{F}\mathbf{P}\mathbf{F}^\mathsf{T} + \mathbf{G}\mathbf{Q}\mathbf{G}^\mathsf{T} \end{aligned}$$

La Correction

L'étape de correction est définie par :

$$\begin{aligned} \mathbf{Z} &= \mathbf{H}\mathbf{P}\mathbf{H}^\mathsf{T} + \mathbf{R} \\ \mathbf{K} &= \mathbf{P}\mathbf{H}^\mathsf{T} \cdot \mathbf{Z}^{-1} \\ \hat{\mathbf{x}}^+ &= \hat{\mathbf{x}} + \mathbf{K}(\mathbf{y} - \mathbf{H}\hat{\mathbf{x}}) \\ \hat{\mathbf{P}}^+ &= \hat{\mathbf{P}} - \mathbf{K}\mathbf{Z}\mathbf{K}^\mathsf{T} \end{aligned}$$

où $\mathbf{Z}$ est la matrice de covariance de *l'innovation* donnée par la différence entre la mesure prédite et l'observation : $\mathbf{z} \triangleq \mathbf{y} - \mathbf{H}\hat{\mathbf{x}}$, $\mathbf{K}$ est le *gain* du filtre qui est proportionnel à l'erreur commise, inversement proportionnel à la précision du capteur, et qui corrige l'état de manière optimale lors de son application.

3.2.2 Le filtre de Kalman Etendu

Le filtre de Kalman Etendu, proposé par Jazwinski [Jazwinski, 1970], s'applique aux systèmes non-linéaires. Considérons le système non-linéaire gaussien suivant :

$$\begin{aligned} \mathbf{x}_k &= \mathbf{f}(\mathbf{x}_{k-1}, w_k) \\ \mathbf{y}_k &= \mathbf{h}(\mathbf{x}_k, v_k) \\ p(\mathbf{x}_0) &= \mathcal{N}(\mathbf{x}_0 - \hat{\mathbf{x}}_0; \mathbf{P}_0) \\ p(w_k) &= \mathcal{N}(w_k - \hat{\mathbf{w}}_k; \mathbf{Q}) \\ p(v_k) &= \mathcal{N}(v_k; \mathbf{R}) \end{aligned}$$

La Prédiction

La fonction de prédiction est linéarisée autour de l'estimée du filtre à l'instant courant et du contrôle connu, en fonction de l'état du système et de la perturbation, dont les matrices jacobiennes sont :

$$\mathbf{F}_\mathbf{x} = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}^\mathsf{T}}\right|_{\hat{\mathbf{x}}, \hat{w}} \qquad \mathbf{F}_w = \left.\frac{\partial \mathbf{f}}{\partial w^\mathsf{T}}\right|_{\hat{\mathbf{x}}, \hat{w}}$$

L'étape de prédiction est donnée par :

$$\begin{aligned} \hat{\mathbf{x}}^+ &= \mathbf{f}(\hat{\mathbf{x}}, \hat{w}) \\ \mathbf{P}^+ &= \mathbf{F}_\mathbf{x}\mathbf{P}\mathbf{F}_\mathbf{x}^\mathsf{T} + \mathbf{F}_w\mathbf{Q}\mathbf{F}_w^\mathsf{T} \end{aligned}$$

La Correction

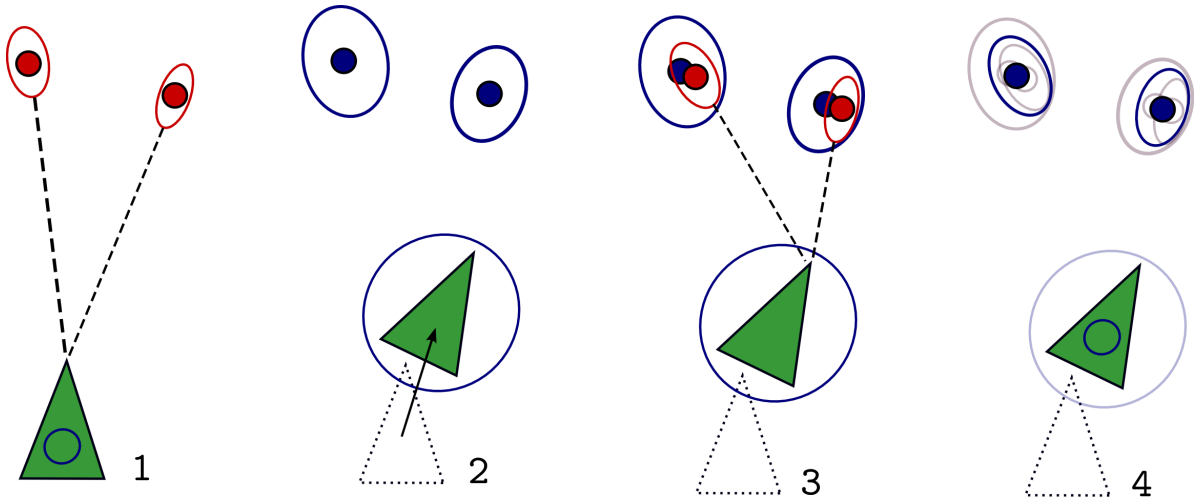


FIGURE 3.3 – Opérations durant une boucle de SLAM EKF, inspirée de [Solà, 2007]. 1 : Le robot perçoit de nouveaux points d'intérêt, et les initialise dans la *carte* en tant qu'amers. 2 : On prédit le déplacement du robot, et les incertitudes de sa pose et de la position des amers augmentent. 3 : Le robot ré-observe les amers déjà présents dans la carte et essaie de les associer aux points observés. 4 : En fonction de l'erreur commise, on met à jour le filtre et les incertitudes diminuent.

La fonction d'observation $\mathbf{y} = \mathbf{h}(\mathbf{x}_k, v_k)$ est linéarisée autour de la meilleur estimée (la prédiction courante) en fonction de l'état du système et du bruit de mesure :

$$\mathbf{H_x} = \left.\frac{\partial \mathbf{h}}{\partial \mathbf{x}^\mathsf{T}}\right|_{\hat{\mathbf{x}},\hat{v}} \qquad \mathbf{H}_v = \left.\frac{\partial \mathbf{h}}{\partial v^\mathsf{T}}\right|_{\hat{\mathbf{x}},\hat{v}}$$

L'étape de correction est définie par :

$$\begin{aligned} \mathbf{Z} &= \mathbf{H_x P H_x^\mathsf{T}} + \mathbf{H}_v \mathbf{R} \mathbf{H}_v^\mathsf{T} \\ \mathbf{K} &= \mathbf{P H_x^\mathsf{T}} \cdot \mathbf{Z}^{-1} \\ \hat{\mathbf{x}}^+ &= \hat{\mathbf{x}} + \mathbf{K}(\mathbf{y} - \mathbf{h}(\hat{\mathbf{x}})) \\ \hat{\mathbf{P}}^+ &= \hat{\mathbf{P}} - \mathbf{K Z K}^\mathsf{T} \end{aligned}$$

3.3 Le Filtre de Kalman Etendu pour le SLAM

D'une manière générale, les étapes principales de l'algorithme de SLAM EKF peuvent se décomposer selon les quatre parties présentées en figure 3.3 :

1. On initialise de nouveaux amers dans la carte à l'instant k,
2. Toujours à l'instant k, on prédit la position du système et des observations sur les amers en les appariant aux observations prédites,
3. A $k+1$ on réobserve les amers présents dans la carte,
4. On corrige la prédiction réalisée, ce qui a pour effet de diminuer l'incertitude globale.

Nous ne considérons ici qu'un SLAM visuel fondé sur des amers 3D ponctuels, observés dans les images par des points d'intérêt. Par ailleurs, nous considérons que l'environnement est statique : les amers sont fixes.

Dans cette section nous résumons la formulation du SLAM EKF et nous reprenons la notation utilisée par [Solà, 2007] dont l'ouvrage en présente très clairement les arcanes.

La *carte* SLAM consiste en un *vecteur d'état* comprenant la pose du robot et les amers cartographiés :

$$X = \begin{bmatrix} \mathcal{R} \\ \mathcal{M} \end{bmatrix} = \begin{bmatrix} \mathcal{R} \\ \mathcal{L}_1 \\ \vdots \\ \mathcal{L}_i \\ \vdots \\ \mathcal{L}_n \end{bmatrix}$$

où $\mathcal{R}$ est le vecteur d'état du système ou du robot qui contient sa *pose*, c'est-à-dire la position $\mathbf{p}$ et l'orientation $\mathbf{q}$ définie sous forme de *quaternion*, et $\mathcal{M}$ l'ensemble des amers $\mathcal{L}_i, i \in [1..n]$ de l'espace. Notons que $\mathcal{R}$ peut également contenir des éléments nécessaires au modèle de mouvement du robot, comme la vitesse linéaire et angulaire, l'accélération... Dans le cadre du filtre de Kalman étendu, la densité *a posteriori* est approximée par des densités gaussiennes avec des matrices de moyennes et de covariances définies par :

$$\hat{X} = \begin{bmatrix} \hat{\mathcal{R}} \\ \hat{\mathcal{M}} \end{bmatrix} = \begin{bmatrix} \hat{\mathcal{R}} \\ \hat{\mathcal{L}}_1 \\ \vdots \\ \hat{\mathcal{L}}_i \\ \vdots \\ \hat{\mathcal{L}}_n \end{bmatrix}$$

et

$$\mathbf{P} = \begin{bmatrix} \mathbf{P}_{\mathcal{R}\mathcal{R}} & \mathbf{P}_{\mathcal{R}\mathcal{M}} \\ \mathbf{P}_{\mathcal{M}\mathcal{R}} & \mathbf{P}_{\mathcal{M}\mathcal{M}} \end{bmatrix} = \begin{bmatrix} \mathbf{P}_{\mathcal{R}\mathcal{R}} & \mathbf{P}_{\mathcal{R}\mathcal{L}_1} & \cdots & \mathbf{P}_{\mathcal{R}\mathcal{L}_i} & \cdots & \mathbf{P}_{\mathcal{R}\mathcal{L}_n} \\ \mathbf{P}_{\mathcal{L}_1\mathcal{R}} & \mathbf{P}_{\mathcal{L}_1\mathcal{L}_1} & \cdots & \mathbf{P}_{\mathcal{L}_1\mathcal{L}_i} & \cdots & \mathbf{P}_{\mathcal{L}_1\mathcal{L}_n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathbf{P}_{\mathcal{L}_i\mathcal{R}} & \mathbf{P}_{\mathcal{L}_i\mathcal{L}_1} & \cdots & \mathbf{P}_{\mathcal{L}_i\mathcal{L}_i} & \cdots & \mathbf{P}_{\mathcal{L}_i\mathcal{L}_n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathbf{P}_{\mathcal{L}_n\mathcal{R}} & \mathbf{P}_{\mathcal{L}_n\mathcal{L}_1} & \cdots & \mathbf{P}_{\mathcal{L}_n\mathcal{L}_i} & \cdots & \mathbf{P}_{\mathcal{L}_n\mathcal{L}_n} \end{bmatrix}$$

où $\mathbf{P}$ la matrice de covariance de la carte SLAM. Elle est carrée et symétrique, ce qui implique que $\mathbf{P}_{\mathcal{R}\mathcal{R}} = \mathbf{P}_{\mathcal{R}\mathcal{R}}^{\mathsf{T}}, \mathbf{P}_{\mathcal{R}\mathcal{M}} = \mathbf{P}_{\mathcal{M}\mathcal{R}}^{\mathsf{T}}$ et $\mathbf{P}_{\mathcal{M}\mathcal{M}} = \mathbf{P}_{\mathcal{M}\mathcal{M}}^{\mathsf{T}}$. $\mathbf{P}_{\mathcal{R}\mathcal{R}}$ est l'incertitude sur la pose du système mobile, $\mathbf{P}_{\mathcal{R}\mathcal{M}}$ décrit les corrélations entre le robot et les amers et $\mathbf{P}_{\mathcal{M}\mathcal{M}}$ les corrélations croisées entre amers, avec $\mathbf{P}_{\mathcal{L}_i\mathcal{L}_i}$ l'incertitude sur la pose de l'amer $\mathcal{L}_i$. La figure 3.4 illustre une représentation de cette carte et de sa matrice de covariance associée.

Le filtrage va ensuite se dérouler selon trois étapes principales :

1. On **prédit** le mouvement du robot selon une loi de commande connue à l'avance et/ou de données issues des capteurs embarqués (généralement proprioceptifs),
2. On effectue la **correction** de cette prédiction en exploitant les observations des points correspondants aux amers déjà connus dans la carte,
3. On **initialise** de nouveaux amers dans la carte.

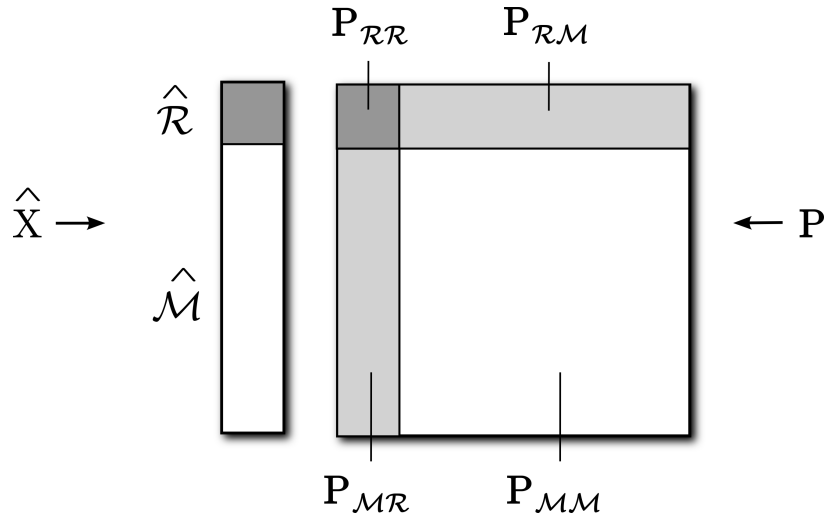


FIGURE 3.4 – La Carte SLAM EKF, et sa matrice de covariance.

3.3.1 Prédiction

La prédiction de la pose du robot à $k+1$ suit la fonction :

$$\mathcal{R}^+ = \mathbf{f}(\mathcal{R}, \mathbf{u})$$

où $\mathbf{u}$ est le vecteur de contrôle supposé gaussien de moyenne $\hat{\mathbf{u}}$ et de variance $\mathbf{U}$: $\mathbf{u} \sim \mathcal{N}(\hat{\mathbf{u}}; \mathbf{U})$. $\mathbf{u}$ peut être donné par la commande ou être observé par des capteurs (odomètres...).
D'après la formulation de l'EKF, l'étape de prédiction est définie par :

$$\hat{\mathcal{R}}^+ = \mathbf{f}(\hat{\mathcal{R}}, \hat{\mathbf{u}}) \tag{3.1}$$
$$\mathbf{P}^+_{\mathcal{RR}} = \mathbf{F}_{\mathcal{R}} \cdot \mathbf{P}_{\mathcal{RR}} \cdot \mathbf{F}_{\mathcal{R}}^{\mathsf{T}} + \mathbf{F}_{\mathbf{u}} \cdot \mathbf{U} \cdot {\mathbf{F}_{\mathbf{u}}}^{\mathsf{T}} \tag{3.2}$$
$$\mathbf{P}^+_{\mathcal{RM}} = \mathbf{F}_{\mathcal{R}} \cdot \mathbf{P}_{\mathcal{RM}} \tag{3.3}$$
$$\mathbf{P}^+_{\mathcal{MM}} = \mathbf{P}_{\mathcal{MM}} \tag{3.4}$$

où les matrices jacobiennes sont :

$$\mathbf{F}_{\mathcal{R}} = \left.\frac{\partial \mathbf{f}}{\partial \mathcal{R}^{\mathsf{T}}}\right|_{\hat{\mathcal{R}}, \hat{\mathbf{u}}} \qquad \mathbf{F}_{\mathbf{u}} = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{u}^{\mathsf{T}}}\right|_{\hat{\mathcal{R}}, \hat{\mathbf{u}}} \tag{3.5}$$

3.3.2 Correction

L'étape de correction du filtre de Kalman se fait par l'application d'un terme additif, produit de l'*innovation* par le *gain*. Dans le cadre du SLAM EKF monoculaire, l'innovation est la différence entre l'observation d'un amer dans l'image et la prédiction de celle-ci. Cette différence est calculée dans le plan image.
La mesure d'un amer i se fait par la fonction d'observation :

$$\mathbf{y}_i = \mathbf{h}(\mathcal{R}, \mathcal{C}, \mathcal{L}_i) + v \tag{3.6}$$

qui dépend de la pose du robot $\mathcal{R}$, de la position de la caméra $\mathcal{C}$ par rapport au robot et de

l'amer $\mathcal{L}_i$ observé. Cette mesure est affectée par un bruit blanc gaussien v de moyenne nulle et de variance R : $v \sim \mathcal{N}(0; \mathbf{R})$. D'après la formulation de l'EKF, l'étape de correction est définie par :

$$\begin{aligned}
\mathbf{z}_i &= \mathbf{y}_i - \mathbf{h}(\hat{\mathcal{R}}, \mathcal{C}, \hat{\mathcal{L}}_i) && (3.7)\\
\mathbf{Z}_i &= \mathbf{H}_i \cdot \mathbf{P} \cdot \mathbf{H}_i^{\mathsf{T}} + \mathbf{R} && (3.8)\\
\mathbf{K}_i &= \mathbf{P} \cdot \mathbf{H}_i^{\mathsf{T}} \cdot \mathbf{Z}_i^{-1} && (3.9)\\
\hat{X}^+ &= \hat{X} + \mathbf{K}_i \cdot \hat{\mathbf{z}}_i && (3.10)\\
\mathbf{P}^+ &= \mathbf{P} - \mathbf{K}_i \cdot \mathbf{Z}_i \cdot \mathbf{K}_i^{\mathsf{T}} && (3.11)
\end{aligned}$$

où la matrice jacobienne est :

$$\mathbf{H}_i = \frac{\partial \mathbf{h}(\hat{\mathcal{R}}, \mathcal{C}, \hat{\mathcal{L}}_i)}{\partial X^{\mathsf{T}}}\bigg|_{\hat{X}} \tag{3.12}$$

3.3.3 Initialisation

L'initialisation consiste en l'ajout d'un nouvel amer dans la carte, tel que :

$$X^+ = \begin{bmatrix} X \\ \mathcal{L}_{n+1} \end{bmatrix}$$

Afin de pouvoir initialiser un nouvel amer dans la carte, la fonction d'observation $\mathbf{h}(\cdot)$ doit être inversée. Posons $\mathbf{g}(\cdot)$ cette fonction telle que :

$$\mathcal{L}_{n+1} = \mathbf{g}(\mathcal{R}, \mathcal{C}, \mathbf{y}_{n+1})$$

Or, comme la profondeur de l'amer n'est pas observable en vision monoculaire, la fonction $\mathbf{h}(\cdot)$ n'est pas inversible. Cette profondeur doit être donnée *a priori*, et $\mathbf{g}(\cdot)$ se récrit :

$$\mathcal{L}_{n+1} = \mathbf{g}(\mathcal{R}, \mathcal{C}, \mathbf{y}_{n+1}, \mathbf{c})$$

où $\mathcal{N}(\mathbf{c} - \hat{\mathbf{c}}; \mathbf{C})$ est une fonction gaussienne de moyenne $\hat{\mathbf{c}}$ et de variance $\mathbf{C}$.

La moyenne peut maintenant être définie par :

$$\hat{\mathcal{L}}_{n+1} = \mathbf{g}(hat\mathcal{R}, \mathcal{C}, \mathbf{y}_{n+1}, \hat{\mathbf{c}})$$

et les jacobiennes par :

$$\begin{aligned}
\mathbf{G}_{\mathcal{R}} &= \frac{\partial \mathbf{g}(\hat{\mathcal{R}}, \mathcal{C}, \mathbf{y}_{n+1}, \hat{\mathbf{c}})}{\partial \mathcal{R}}\\
\mathbf{G}_{\mathbf{y}_{n+1}} &= \frac{\partial \mathbf{g}(\hat{\mathcal{R}}, \mathcal{C}, \mathbf{y}_{n+1}, \hat{\mathbf{c}})}{\partial \mathbf{y}_{n+1}}\\
\mathbf{G}_{\mathbf{c}} &= \frac{\partial \mathbf{g}(\hat{\mathcal{R}}, \mathcal{C}, \mathbf{y}_{n+1}, \hat{\mathbf{c}})}{\partial \mathbf{c}}
\end{aligned}$$

La matrice de covariance $\mathbf{P}$ peut s'écrire :

$$\mathbf{P} = \begin{bmatrix} \mathbf{P} & \mathbf{P}_{\mathcal{L}_{n+1}X}^{\mathsf{T}} \\ \mathbf{P}_{\mathcal{L}_{n+1}X} & \mathbf{P}_{\mathcal{L}_{n+1}\mathcal{L}_{n+1}} \end{bmatrix}$$

avec :

$$\begin{aligned}\mathbf{P}_{\mathcal{L}_{n+1}X} &= \mathbf{G}_{\mathcal{R}}\mathbf{P}_{\mathcal{R}X} \\ \mathbf{P}_{\mathcal{L}_{n+1}\mathcal{L}_{n+1}} &= \mathbf{G}_{\mathcal{R}}\mathbf{P}_{\mathcal{R}\mathcal{R}}\mathbf{G}_{\mathcal{R}}^{\mathsf{T}} + \mathbf{G}_{\mathbf{y}_{n+1}}\mathbf{R}\mathbf{G}_{\mathbf{y}_{n+1}}^{\mathsf{T}} + \mathbf{G}_{\mathbf{c}}\mathbf{C}\mathbf{G}_{\mathbf{c}}^{\mathsf{T}}\end{aligned}$$

Plusieurs formulations ont été proposées pour la fonction $\mathbf{g}(\cdot)$: nous les présenterons dans le chapitre suivant.

3.4 Conclusion

Nous avons décrit dans ce chapitre comment un filtre de Kalman étendu s'applique au problème du SLAM d'une manière générale. Cependant, la communauté SLAM est très active, et elle met à disposition un grand nombre d'algorithmes de différentes méthodes ; les principales (dont RT-SLAM venant du LAAS) sont accessibles sur le site [Open-SLAM, 2009]. Le principal inconvénient du SLAM EKF est l'augmentation de sa complexité lors de l'ajout de nouveaux amers dans la carte. Les algorithmes doivent être optimisés et des techniques de gestion de ces amers sont alors à mettre en place, notamment dans le cas des systèmes embarqués où nous disposons de ressources modestes et ne pouvons que maintenir quelques amers dans la carte. Nous nous rapprochons ainsi d'une part du SLAM court-terme, où les amers ne sont maintenus que quelques instants après leurs disparitions de l'image, et d'autre part de l'odométrie optique, où les amers sont supprimés une fois hors du champ de vue de la caméra. L'odométrie optique a pour seul but d'estimer les déplacements d'un système mobile à partir d'une caméra embarquée.

Le chapitre suivant décrit le déroulement d'un SLAM EKF monoculaire et comment il peut être optimisé par l'emploi de méthodes et d'algorithmes précis. Nous étendrons ensuite son application avec l'utilisation conjointe de plusieurs capteurs.

Chapitre 4

SLAM Visuel multi-capteurs

4.1 Introduction

Plusieurs types de capteurs tels que les sonars, radars, lidars... peuvent être utilisés pour le SLAM. Bien entendu, les caméras se sont rapidement déployées pour traiter de la localisation d'un système mobile, et y devenir un capteur quasi-indispensable. Les progrès technologiques (performances, compacité...), la très large diffusion dans le grand public (coût) et le développement de nouveaux services sur les *smartphones* ont accéléré le processus. Par ailleurs, les caméras peuvent percevoir dans différentes bandes spectrales et peuvent être équipées de différentes optiques, à focales plus ou moins courtes pour des champs de vue plus ou moins élargis. De plus, ces capteurs peuvent s'utiliser seuls, ou couplés pour former des systèmes bi-caméras, stéréovision ou multi-caméras. Le choix pour réaliser un SLAM Visuel est donc vaste et se définit précisément par l'application concernée. Ici, nous souhaitons réaliser un SLAM qui soit robuste quelles que soient les conditions météorologiques, ce qui impose l'utilisation d'au moins une caméra infrarouge. Dans certaines conditions (pluie dense ou neige), les performances se dégradent dans cette bande spectrale ; pour traiter ces situations, mais aussi pour rendre le SLAM visuel robuste dans le cas général, nous ajoutons une caméra visible. De même, le champ de vue doit être important mais le rendu naturel pour le pilote à qui cette image va être présentée. Nous choisirons alors une optique classique de manière à limiter les traitements des distorsions qui engendrent une latence et des artefacts dans l'image.

4.2 Etat de l'art du SLAM monoculaire

A l'origine, le problème du SLAM ne concernait que les robots d'intérieurs pourvus de capteurs de profondeurs (lasers, sonars...). La caméra n'était pas considérée à cause de son inconvénient majeur : la perte de profondeur. Le problème du SLAM visuel se réfère souvent à celui de *Structure From Motion* (SFM) dont l'étude remonte aux années 80 et qui était un thème majeur de la Vision par Ordinateur. A cette époque, il était question de retrouver la géométrie d'une scène via une ou plusieurs caméras la percevant de différents points de vue de manière à pouvoir la reconstruire [Harris and Pike, 1988]. Ces études ont permis de comprendre le processus de vision et de poser les fondements de la géométrie visuelle [Hartley and Zisserman, 2000]. Elles n'abordaient que très rarement la problématique du SLAM.

Même si SLAM et SFM traitent de thématiques proches, la synergie entre recherches en Vision et en Robotique est venue dans la dernière décennie seulement. La vision dans la problématique du SLAM fut approfondie en 1997 par [Neira et al., 1997] s'appuyant sur les travaux de Smith *et al.* [Smith et al., 1988] et Moutarlier *et al.* [Moutarlier and Chatila, 1989].

En 1998, Davison *et al.* [Davison and Murray, 1998] proposent l'utilisation d'un système de stéréovision couplé à des odomètres pour réaliser une cartographie tridimensionnelle de l'environnement. Percevoir un amer de deux points de vue différents au même instant permet de connaître plus précisément et plus rapidement la donnée de profondeur manquante dans l'approche mono-caméra. Cette approche sera étendue à un système stéréoscopique seul par Jung *et al.* [Jung and Lacroix, 2003].

Ce n'est qu'en 2003 que le SLAM monoculaire connaît une grande avancée grâce aux travaux de Davison [Davison, 2003]. Celui-ci propose un algorithme de SLAM, le *MonoSLAM*, utilisant une simple caméra et fonctionnant en temps réel sur un ordinateur grand public. Les performances temps réel ont été obtenues notamment par la gestion du nombre de points d'intérêt extraits des images, des *amers* présents dans la carte SLAM et par l'implémentation de l'algorithme de *Recherche Active* proposé par [Harris and Pike, 1988]. Pour traiter de la non-observabilité de la profondeur en vision monoculaire, Davison introduit une approche particulaire : un amer est introduit dans la carte avec du retard, une fois que la profondeur est connue. Cependant, cette approche ne permet pas de traiter les amers lointains (points sur la ligne d'horizon par exemple). Une fois les points d'intérêt détectés et initialisés dans la carte SLAM, ils doivent être réobservés dans l'image lors de l'étape de correction du filtre, afin de garantir sa bonne cohérence. Au lieu de rechercher le point à réobserver dans l'ensemble de l'image, l'algorithme de Recherche Active essaie de le retrouver uniquement dans la zone d'incertitude obtenue grâce au filtre. Le gain en temps de traitement est donc important, mais cet algorithme s'applique sur un nombre limité de points.

Le problème du SLAM par vision monoculaire a atteint une certaine maturité en 2005 avec l'arrivée des techniques d'initialisations d'amers non retardées dans la carte SLAM. La première proposition fondée sur une somme de gaussiennes, fut formulée par Solà *et al.* [Sola et al., 2005], et finalement résolut de manière plus robuste en 2006 avec la représentation d'amers ponctuels par paramétrisation en inverse de profondeur, ou *Inverse-Depth Paramétrisation* (IDP), énoncé par Montiel [Montiel, 2006]. De nombreuses paramétrisations aujourd'hui existent pour représenter un point, et leurs usages ont été étendus aux lignes. L'ensemble de ces représentations sur la cohérence du filtre a été étudié par Solà [Solà et al., 2012].

Les approches multi-caméras se généralisent et en 2006, Kaess *et al.* [Kaess and Dellaert, 2006] implémentent l'algorithme du SLAM sur un robot mobile équipé d'une ceinture de 8 caméras. En 2007, Mei *et al.* [Mei, 2007] utilisent une caméra omnidirectionnelle catadioptrique associée à un télémètre laser. La même année, Solà *et al.* [Sola et al., 2007] proposent le SLAM bi-caméra qui traite du SLAM depuis un capteur stéréovision, par une adaptation de l'approche mono-caméra, ce qui permet de calculer directement la profondeur pour les points qui sont dans le champ de vue de la stéréovision, tout en intégrant des amers lointains, typiquement à l'infini, idéaux pour recaler l'orientation.

Parmi les récents développements en matière de SLAM monoculaire, l'algorithme *RT-SLAM* [Roussillon et al., 2011], au développement duquel nous avons participé, permet de fusionner au sein d'une même application SLAM une ou plusieurs caméras, odomètres, centrales inertielles et GPS.

Parallèlement, à partir de 2004, la communauté commence à s'intéresser aux capteurs monoculaires infrarouges, technologie devenant de plus en plus accessible. La première utilisation d'un tel capteur dans un système de localisation est proposée par Nygårds *et al.* [Nygårds et al., 2004] qui fusionnent, dans leur application, un GPS-RTK (centimétrique), une centrale inertielle et un système de navigation au sein d'un même filtre de Kalman. Bien qu'il ne cartographie pas directement l'environnement, leur système permet de géolocaliser et de suivre des objets détectés par une caméra visible et une caméra infrarouge thermique.

La première application d'une caméra infrarouge au problème du SLAM est proposée par Folkesson *et al.* [Folkesson and Christensen, 2008]. Ils équipent un robot mobile d'une caméra proche-infrarouge 320×240 cadencée à 10 Hz et d'une centrale inertielle bas coût. Ils soulignent le problème de faible qualité d'une image infrarouge et utilisent alors un descripteur SIFT (*Scale-Invariant Feature Transform*) pour les amers extraits. Vu la faible résolution des images traitées par cette approche, ce type de points n'est pas adapté.

En 2011, nous embarquons, avec *C-SLAM*, un SLAM monoculaire par caméra infrarouge thermique, cadencée à 30 Hz et sans capteurs additionnels [Gonzalez et al., 2011]. La faible résolution de la caméra (160×120) associée à l'aspect diffus des images ne nous permet de parcourir que quelques dizaines de mètres. Nous présentons en 4.7.2 une évolution de cette application par l'ajout de capteurs inertiels et GPS qui permettent de nous rapprocher avec succès de la vérité terrain. Ces résultats sont présentés ci-après.

Ce chapitre est dédié au SLAM monoculaire et multi-capteurs. Tout d'abord, nous présentons comment les différents amers sont représentés dans la carte, car ils sont au coeur du SLAM. Les trois étapes du SLAM EKF que sont la **prédiction**, la **correction** et l'**initialisation** seront présentées sur le SLAM monoculaire, ainsi que les algorithmes implémentés pour obtenir une localisation temps réel et de bonne qualité. Nous exposerons les résultats obtenus et nous présenterons notre contribution au SLAM par vision monoculaire infrarouge. Dans une seconde partie, après un bref état de l'art sur les différentes intégrations de capteurs dans le contexte du SLAM visuel, la fusion des données issues d'une centrale inertielle, puis d'un GPS avec celles acquises par la vision monoculaire, en utilisant le même filtre de Kalman étendu, sera détaillée. Enfin, nous illustrerons cette section par les résultats obtenus sur la fusion de ces capteurs au SLAM monoculaire visible et infrarouge.

4.3 Paramétrisation des points

Lorsqu'un pixel de l'image est choisi pour point de référence, nous devons effectuer une série d'opérations pour le transformer en un amer rajouté à la carte SLAM. Cette opération, dite de *rétro-projection* et détaillée en section 2.3.2, va transformer ce pixel de l'image en point de l'espace, en essayant d'évaluer au mieux sa profondeur. Lors de l'étape de correction du filtre, l'opération inverse doit être réalisée : la *projection*, présentée en 2.3.1. Il est question ici de projeter un amer, dans la position prédite à partir de la carte, dans le plan image afin de pouvoir évaluer l'erreur entre cette projection et l'observation réelle de ce même amer. La manière dont sera paramétré ce point dans la carte va avoir une grande influence sur le résultat du SLAM. Dans cette section, nous présentons les principales paramétrisations de ces points, qui sont également celles utilisées dans [Roussillon et al., 2011] et [Gonzalez et al., 2011]. Pour présenter ces paramétrisations, nous reprenons la formulation proposée dans [Sola, 2010].

Les points adoptent couramment trois types de paramétrisations comme représenté en figure 4.1 :

- Point Euclidien ou *Euclidean Point (EP)*,
- Paramétrisation Inverse de Profondeur ou *Inverse-Depth Parametrisation (IDP)*,
- Point Homogène Ancré ou *Anchored Homogeneous Point (AHP)*.

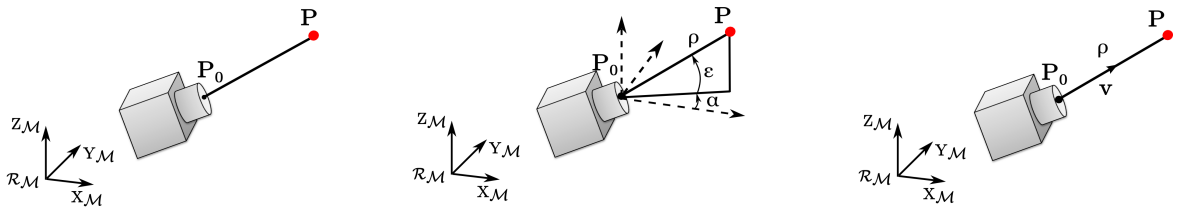


FIGURE 4.1 – Les 3 paramétrisations de points les plus courantes en SLAM visuel. A gauche, la paramétrisation euclidienne où uniquement 3 composantes sont nécessaires pour caractériser un point. Au milieu, la paramétrisation *IDP*, utilisant 6 composantes. A droite, la paramétrisation *AHP* qui fait appel à 7 composantes.

4.3.1 Les points euclidiens

Un point euclidien $\mathbf{p}$ est représenté simplement par trois coordonnées cartésiennes :

$$\mathcal{P}_{\mathbb{E}} = \mathbf{p} = \begin{bmatrix} x & y & z \end{bmatrix}^{\mathsf{T}} \in \mathbb{R}^3$$

La transformation de ce point dans le repère caméra, suivie de sa projection sur le plan image, sont définies par :

$$\underline{\mathbf{u}} = \mathbf{K}\mathsf{R}^{\mathsf{T}}(\mathbf{p} - \mathsf{T}) \in \mathbb{P}^2$$

où $\mathbf{K}$ est la matrice des paramètres intrinsèques, définie en section 2.3, R et T sont la matrice de rotation et le vecteur de translation de la caméra $\mathcal{C}$, par rapport au repère du monde (inverse des paramètres extrinsèques introduits en 2.3.1). Rappelons que $\underline{\mathbf{u}}$ est un point de l'espace projectif, donc $\underline{\mathbf{u}} = (su \quad sv \quad s)^{\mathsf{T}}$.

Les opérations de rétro-projection et transformation inverse sont faites par :

$$\mathbf{p} = \mathsf{R}\mathbf{K}^{-1}\underline{\mathbf{u}} + \mathsf{T}$$

Lorsqu'un point 3D est observé la première fois, z n'est pas observable : dans les premiers travaux sur le SLAM visuel, Davison initialisait z par N particules tirées avec une distribution uniforme entre z_{min} et z_{max}. Solà initialisait z par une série de gaussiennes. Mais ces deux méthodes sont très peu robustes du fait des problèmes de linéarisation.

4.3.2 La paramétrisation IDP

Lors de l'initialisation d'un nouveau point, il est important de pouvoir l'ajouter à la carte SLAM le plus rapidement possible. La paramétrisation euclidienne ne permet pas d'initialiser les points directement car elle engendre des observations fortement non-linéaires [Civera et al., 2008]. Il est important que le point à initialiser puisse avoir une formulation gaussienne afin de l'ajouter directement à la carte. Deux paramétrisations sont donc couramment utilisées : *AHP* et *IDP*.

Un point IDP, acronyme anglais pour *Inverse-Depth Point*, proposée par [Montiel, 2006] [Civera et al., 2008], est défini par 6 composantes :

$$\mathcal{P}_{\mathbb{ID}} = \mathbf{p} = \begin{bmatrix} x_0 & y_0 & z_0 & \epsilon & \alpha & \rho \end{bmatrix}^{\mathsf{T}} \in \mathbb{R}^6$$

Les trois premières composantes $\mathbf{p}_0 = [x_0 \;\; y_0 \;\; z_0]$ correspondent aux coordonnées euclidiennes du centre optique de la caméra lors de l'initialisation du point. Le couple $[\epsilon \;\; \alpha]$ représente l'élévation et l'azimut du rayon optique, reliant le centre optique et le point. Enfin, si l'on note d la distance entre ces 2 points, alors $\rho = 1/d$ est la distance inverse.

A un point IDP $\mathcal{P}_{\mathbb{ID}}$, correspond le point euclidien $\mathbf{p}$:

$$\mathbf{p} = \mathbf{p}_0 + \mathbf{v}^*(\epsilon, \alpha)/\rho \tag{4.1}$$

où $\mathbf{v}^*(\epsilon, \alpha)$ est le vecteur unitaire défini par :

$$\mathbf{v}^*(\epsilon, \alpha) = \begin{bmatrix}\cos(\epsilon)\cos(\alpha) & \cos(\epsilon)\sin(\alpha) & \sin(\epsilon)\end{bmatrix}^{\mathsf{T}}$$

La transformation de ce point dans le repère caméra, suivie de sa projection sur le plan image, sont définies par :

$$\underline{\mathbf{u}} = \mathbf{K}\mathsf{R}^{\mathsf{T}}(\mathbf{v}^*(\epsilon, \alpha) - \rho(\mathsf{T} - \mathbf{p}_0)) \in \mathbb{P}^2$$

L'opération de rétro-projection et transformation inverse sont faites par :

$$\mathcal{P}_{\mathbb{ID}} = \begin{bmatrix}\mathbf{p}_0 \\ (\epsilon, \alpha) \\ \rho\end{bmatrix} = \begin{bmatrix}\mathsf{T} \\ v^*(\mathsf{R}\mathbf{K}^{-1}\underline{\mathbf{u}} + \mathsf{T}) \\ \rho^{\mathcal{C}}\end{bmatrix}$$

où $v^*(\mathbf{v})$ est le vecteur des angles d'élévation et d'azimut (ϵ, α) d'un vecteur directeur $\mathbf{v} = (\mathbf{v}_x \quad \mathbf{v}_y \quad \mathbf{v}_z)^{\mathsf{T}}$:

$$\begin{bmatrix}\epsilon \\ \alpha\end{bmatrix} = v^*((\mathbf{v}_x \quad \mathbf{v}_y \quad \mathbf{v}_z)^{\mathsf{T}}) = \begin{bmatrix}\arctan(\mathbf{v}_z/\sqrt{\mathbf{v}_x^2 + \mathbf{v}_y^2}) \\ \arctan(\mathbf{v}_y/\mathbf{v}_x)\end{bmatrix}$$

La distance inverse $\rho^{\mathcal{C}}$ est définie dans l'espace caméra lors de l'initialisation et doit être fournie comme estimée. Sa relation avec la distance d est donnée par $\rho^{\mathcal{C}} = \|\mathbf{K}^{-1}\underline{\mathbf{u}}\|/d$. Contrairement aux méthodes de distance basée sur une somme de gaussiennes ou un filtre particulaire, cette méthode est plus robuste vis-à-vis de la linéarisation.

4.3.3 La paramétrisation AHP

Un point AHP, acronyme anglais pour *Anchored Homogeneous Point*, est très proche de la paramétrisation IDP. Il est défini par 7 composantes :

$$\mathcal{P}_{\mathbb{AH}} = \begin{bmatrix}\mathbf{p}_0^{\mathsf{T}} & \mathbf{v}^{\mathsf{T}} & \rho\end{bmatrix}^{\mathsf{T}} = \begin{bmatrix}x_0 & y_0 & z_0 & \mathbf{v}_x & \mathbf{v}_y & \mathbf{v}_z & \rho\end{bmatrix}^{\mathsf{T}} \in \mathbb{R}^7$$

A un point AHP $\mathcal{P}_{\mathbb{AH}}$, correspond le point euclidien $\mathbf{p}$:

$$\mathbf{p} = \mathbf{p}_0 + \mathbf{v}/\rho \tag{4.2}$$

La transformation de ce point dans le repère caméra, suivie de sa projection sur le plan image, est définie par :

$$\underline{\mathbf{u}} = \mathbf{K}\mathsf{R}^{\mathsf{T}}(\mathbf{v} - \rho(\mathsf{T} - \mathbf{P}_0)) \in \mathbb{P}^2$$

L'opération de rétro-projection et transformation inverse est faite par :

$$\mathcal{P}_{\mathbb{AH}} = \begin{bmatrix}\mathbf{p}_0 \\ \mathbf{v} \\ \rho\end{bmatrix} = \begin{bmatrix}\mathsf{T} \\ \mathsf{R}\mathbf{K}^{-1}\underline{\mathbf{u}} \\ \rho^{\mathcal{C}}\end{bmatrix}$$

Dans cette paramétrisation, la distance inverse $\rho^{\mathcal{C}}$ doit être également fournie en tant qu'estimée.

4.4 Prédiction

Dans le cas où nous n'avons qu'une seule caméra à notre disposition, le modèle de prédiction le plus simple est le modèle de mouvement à **vitesse constante**. Le vecteur d'état du robot $\mathcal{R}$ est défini selon 13 paramètres :

$$\mathcal{R} = (\mathbf{p} \quad \mathbf{q} \quad \mathbf{v} \quad \mathbf{w})^\mathsf{T}$$

où $\mathbf{p} = (p_x \quad p_y \quad p_z)^\mathsf{T}$ et $\mathbf{q} = (q_w \quad q_x \quad q_y \quad q_z)^\mathsf{T}$ sont respectivement la position et l'orientation du robot dans l'espace ; $\mathbf{v} = (v_x \quad v_y \quad v_z)^\mathsf{T}$ et $\mathbf{w} = (w_x \quad w_y \quad w_z)^\mathsf{T}$ correspondent respectivement à sa vitesse linéaire et angulaire selon chaque axe du repère.

L'orientation du robot est représentée par un quaternion, car les équations liées à la prédiction et à la correction sont simplifiées, et on évite le problème classique de blocage de cardan (ou *Gimbal Lock*). Par contre, cette représentation n'est pas minimale : le vecteur $(q_w \quad q_x \quad q_y \quad q_z)^\mathsf{T}$ doit être normalisé après chaque manipulation.

Le modèle d'évolution du robot $\mathbf{f}$ définie en 3.1 s'exprime simplement par :

$$\mathcal{R}^+ = \begin{bmatrix} \mathbf{p}^+ \\ \mathbf{q}^+ \\ \mathbf{v}^+ \\ \mathbf{w}^+ \end{bmatrix} = \begin{bmatrix} \mathbf{p} + \mathbf{v} \times \Delta t \\ \mathbf{q} \otimes Q(\mathbf{w} \times \Delta t) \\ \mathbf{v} + \mathbf{v}_i \\ \mathbf{w} + \mathbf{w}_i \end{bmatrix}$$

où $\otimes$ est le produit de quaternions, $Q(.)$ la fonction de transformation d'un vecteur en quaternion et $\mathbf{v}_i$ et $\mathbf{w}_i$ sont des perturbations de type impulsion sur la commande du robot en vitesse linéaire et en vitesse angulaire.

Les jacobiennes $\mathbf{F}_\mathcal{R}$ et $\mathbf{F}_\mathbf{u}$ définies en 3.5 s'écrivent sous la forme :

$\mathbf{F}_\mathcal{R} =$

	$\mathbf{p}$	$\mathbf{q}$	$\mathbf{v}$	$\mathbf{w}$
$\mathbf{p}^+$	$\mathbf{I}_{3\times3}$	0	$I_{3\times3} \cdot \Delta t$	0
$\mathbf{q}^+$	0	$\frac{\partial \mathbf{q}^+}{\partial \mathbf{q}}$	0	$\frac{\partial \mathbf{q}^+}{\partial \mathbf{w}}$
$\mathbf{v}^+$	0	0	$\mathbf{I}_{3\times3}$	0
$\mathbf{w}^+$	0	0	0	$\mathbf{I}_{3\times3}$

$\mathbf{F}_\mathbf{u} =$

	$\mathbf{v}_i$	$\mathbf{w}_i$
$\mathbf{p}^+$	0	0
$\mathbf{q}^+$	0	0
$\mathbf{v}^+$	$I_{3\times3}$	0
$\mathbf{w}^+$	0	$I_{3\times3}$

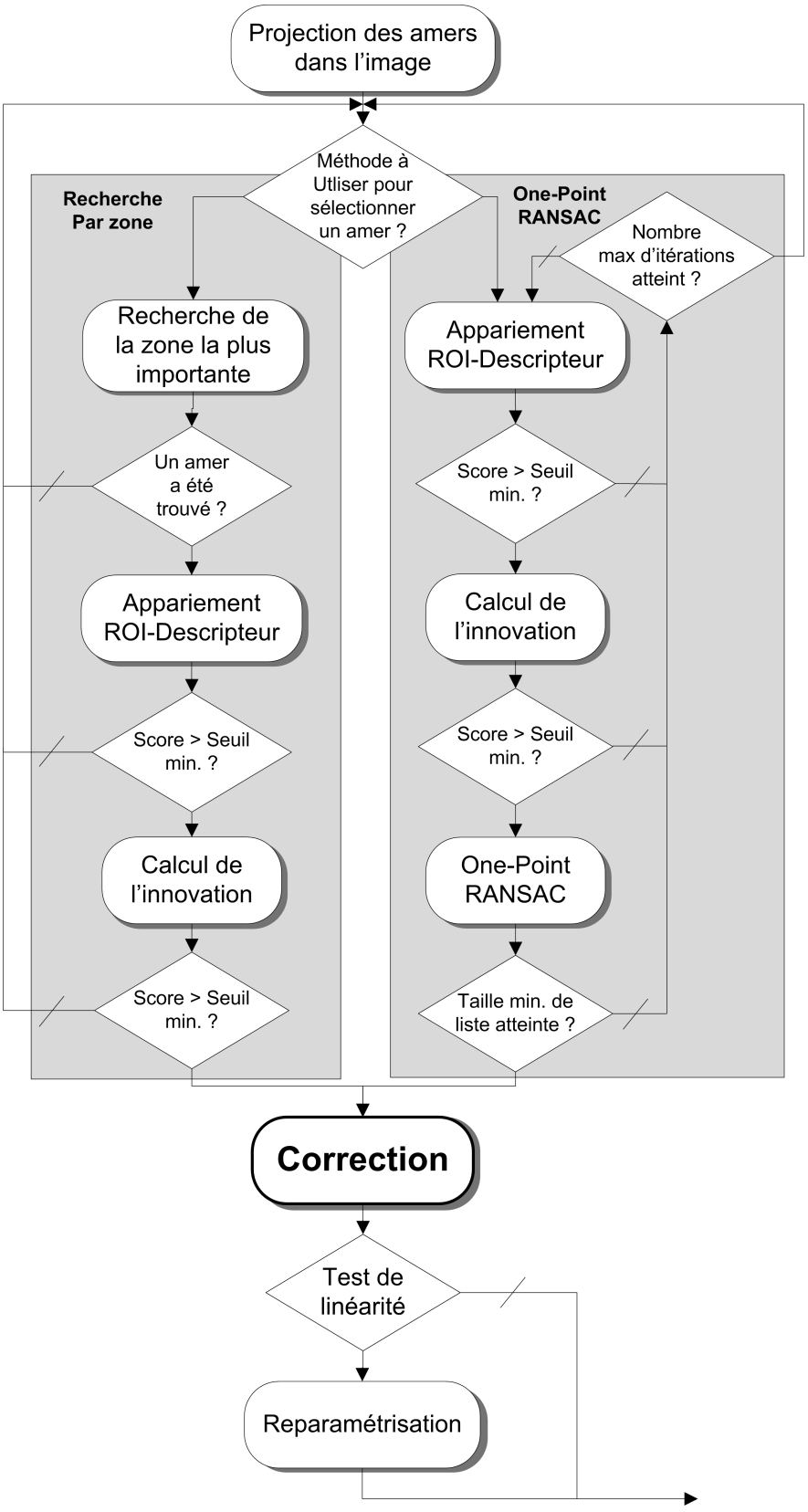


FIGURE 4.2 – Algorithme proposé pour l'étape de correction du filtre de Kalman.

4.5 Correction

La correction du filtre de Kalman se fait par l'application d'un terme, produit de l'*innovation* par le *gain*. L'innovation est directement obtenue par la différence entre la mesure de l'état réel à l'instant k et sa valeur prédite à partir de la valeur estimée de cet état à l'instant $k-1$. Dans le cadre du SLAM EKF visuel, cette innovation est égale à la différence entre la position observée des amers dans l'image et la projection dans ce même plan de leurs positions 3D prédites. Le problème essentiel concerne l'association entre observations et amers présents dans la carte.

Pour la correction, nous proposons l'algorithme présenté figure 4.2. Celui-ci peut être décomposé en six fonctions principales :

1. L'ensemble des amers présents dans la carte est projeté dans l'image. On obtient la moyenne et la covariance associée à chaque point projeté.
2. On choisit une méthode pour la sélection de l'amer à corriger : soit One-Point RANSAC (qui privilégie plutôt les amers les plus précis), soit, au contraire, par Recherche Active (ou *Active Search*) de l'amer qui a la plus grande incertitude.
3. Si un amer est sélectionné, on essaie de le réobserver dans l'image.
4. Si cet amer a été apparié dans l'image, on calcule son innovation.
5. Si la distance de Mahalanobis est inférieure à un seuil déterminé, on **corrige** le filtre avec cette observation.
6. Enfin, si cet amer est de type IDP ou AHP et si le test de linéarité présenté en [Montiel, 2006] est satisfait, l'amer peut être reparamétré en euclidien de manière à occuper moins de place mémoire et à accélérer les calculs.

Dans cette section, nous présentons les différents algorithmes nécessaires pour l'obtention d'un gain de meilleure qualité possible, ainsi que les optimisations proposées pour accélérer les différentes étapes.

4.5.1 Projection

La première étape de la correction consiste en la projection de l'ensemble des amers de la carte SLAM dans le plan image. La fonction de projection dépend de la paramétrisation que l'amer $\mathcal{L}_i$ adopte. En fonction de celle-ci, on effectue la projection associée définie en section 4.3. Cependant, la transformation (R, T) donne la position de la caméra dans le référentiel du monde dans lequel les positions des amers et du robot sont estimées. Cette transformation peut se décomposer en deux : la position du robot $\mathcal{R}$ dans le monde et la position de la caméra $\mathcal{C}$ par rapport au robot.

On obtient la projection $\underline{\mathbf{u}} = \mathbf{h}_i(\hat{\mathcal{R}}, \mathcal{C}, \hat{\mathcal{L}}_i)$ de l'amer dans l'image, de même que les jacobiennes associées à l'ensemble de ces transformations, dont :

$$\mathbf{H}_{\mathcal{R}} = \frac{\partial \mathbf{h}_i(\hat{\mathcal{R}}, \mathcal{C}, \hat{\mathcal{L}}_i)}{\partial \mathcal{R}} \quad \text{et} \quad \mathbf{H}_{\mathcal{L}_i} = \frac{\partial \mathbf{h}_i(\hat{\mathcal{R}}, \mathcal{C}, \hat{\mathcal{L}}_i)}{\partial \mathcal{L}_i} \tag{4.3}$$

Comme la jacobienne $\mathbf{H}_i = \left.\frac{\partial \mathbf{h}_j(\hat{\mathcal{R}}, \mathcal{C}, \hat{\mathcal{L}}_j)}{\partial X^{\mathsf{T}}}\right|_{\hat{X}} = 0$ pour $j \neq i$, alors $\mathbf{H}_i$ est creuse :

$$\mathbf{H}_i = [\mathbf{H}_{\mathcal{R}} \quad 0 \quad \cdots \quad 0 \quad \mathbf{H}_{\mathcal{L}_i} \quad 0 \quad \cdots \quad 0] \tag{4.4}$$

Ainsi, lors du calcul de la covariance de l'innovation $\mathbf{Z}_i$ définie en 3.8, seules les parties $\mathbf{P}_{\mathcal{R}\mathcal{R}}, \mathbf{P}_{\mathcal{R}\mathcal{L}_i}, \mathbf{P}_{\mathcal{L}_i\mathcal{R}} \text{et} \mathbf{P}_{\mathcal{L}_i\mathcal{L}_i}$ de la matrice de covariance $\mathbf{P}$ sont concernées, comme illustré en figure 4.3.

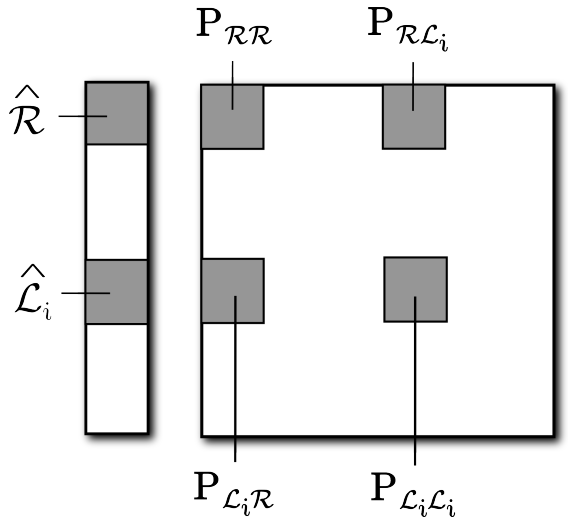


FIGURE 4.3 – Représentation des éléments de la matrice de covariance $\mathbf{P}$ impliqués dans le calcul de l'innovation d'un amer $\mathcal{L}_i$.

La covariance de l'innovation s'exprime alors selon :

$$\mathbf{Z}_i = [\mathbf{H}_{\mathcal{R}} \quad \mathbf{H}_{\mathcal{L}_i}] \cdot \begin{bmatrix} \mathbf{P}_{\mathcal{R}\mathcal{R}} & \mathbf{P}_{\mathcal{R}\mathcal{L}_i} \\ \mathbf{P}_{\mathcal{L}_i\mathcal{R}} & \mathbf{P}_{\mathcal{L}_i\mathcal{L}_i} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{H}_{\mathcal{R}}^{\mathsf{T}} \\ \mathbf{H}_{\mathcal{L}_i}^{\mathsf{T}} \end{bmatrix} + \mathbf{R} \tag{4.5}$$

où $\mathbf{R}$ est le bruit associé au capteur, la caméra dans notre cas. Pour une image nette, la variance du bruit d'observation peut être considérée de l'ordre du pixel :

$$\mathbf{R} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$\mathbf{Z}_i$ permet de calculer l'incertitude sur la valeur prédite de l'observation d'un amer $\mathcal{L}_i$ défini par la formule 3.6 dans le cadre du SLAM monoculaire, en fonction de la pose du robot $\mathcal{R}$ et de celle de la caméra $\mathcal{C}$. Cette incertitude est nécessaire pour appliquer l'approche de Recherche Active. Elle peut alors être récrite sous la forme :

$$\mathbf{y}_i = \mathbf{h}_i(\mathcal{R}, \mathcal{C}, \mathcal{L}_i) + v$$

avec sa Jacobienne définie par :

$$\mathbf{H}_i = \left. \frac{\partial \mathbf{h}_i(\hat{\mathcal{R}}, \mathcal{C}, \hat{\mathcal{L}}_i)}{\partial X^{\mathsf{T}}} \right|_{\hat{X}}$$

Or, nous n'avons pas directement accès à cette mesure qui implique la recherche du point dans l'image. Pour limiter ce temps de recherche, nous nous intéressons aux covariances de l'innovation. En effet, cette matrice est constituée des variances et covariances de la projection du pixel dans l'image :

$$\mathbf{Z}_i = \begin{bmatrix} \sigma_{u_i}^2 & \sigma_{uv_i} \\ \sigma_{vu_i} & \sigma_{v_i}^2 \end{bmatrix} \tag{4.6}$$

Au lieu de traiter l'ensemble de l'image pour recherche l'observation de l'amer $\mathcal{L}_i$, nous pouvons nous limiter à une région très restreinte de l'image, comme illustré en figure 4.4, et délimitée

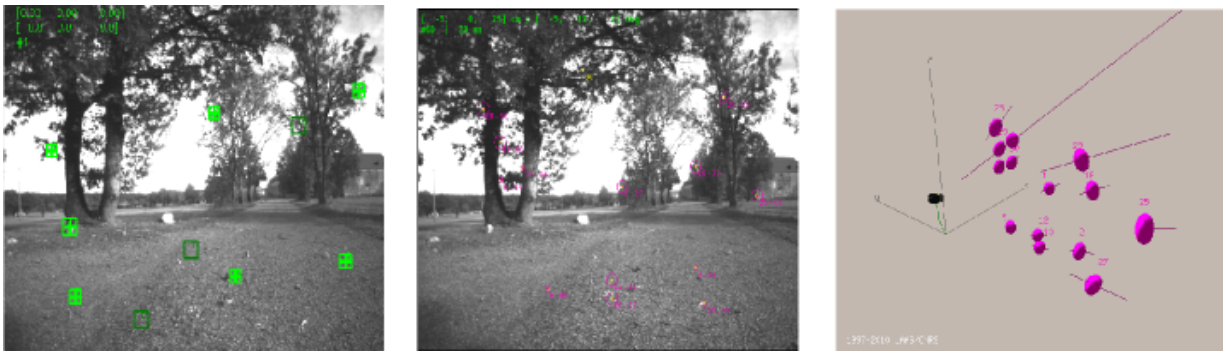


FIGURE 4.4 – Zones de recherche à 3σ centrées sur la prédiction dans l'image. *A gauche*, image issue de *CSLAM*, où la zone de recherche est définie par un rectangle. *Au milieu*, image issue de *RT-SLAM*, où les zones de recherche sont représentées par des ellipses. *A droite*, les zones d'incertitude adoptent une représentation en 3D offerte par *RT-SLAM*.

par $\mathbf{h}_i(\hat{\mathcal{R}}, \mathcal{C}, \hat{\mathcal{L}}_i) \pm 3 \times \sqrt{(\sigma^2_{i_u})}$ et par $\mathbf{h}_i(\hat{\mathcal{R}}, \mathcal{C}, \hat{\mathcal{L}}_i) \pm 3 \times \sqrt{(\sigma^2_{i_v})}$, nous donnant une zone de recherche simplifiée où nous sommes sûrs de trouver la mesure à 98,9%.

La zone d'incertitude à 1σ est elliptique, ses demi-axes a et b sont donnés par les valeurs propres de la matrice de covariance $\mathbf{Z}_i$. Ils satisfont l'équation $u^2/a^2 + v^2/b^2 = 1$, qui peut être récrite sous forme matricielle :

$$\begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} 1/a^2 & 0 \\ 0 & 1/b^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = 1$$

et son orientation θ par :

$$\theta = \frac{1}{2} \cdot \arctan\left(\frac{2 \cdot \sigma_{uv_i}}{\sigma^2_{u_i} - \sigma^2_{v_i}}\right)$$

Une fois les projections réalisées dans l'image, nous devons choisir un amer sur lequel nous allons effectuer l'étape de correction du filtre de Kalman. Pour effectuer ce choix, nous proposons deux méthodes : soit une recherche active de l'amer sélectionné en fonction de l'aire de sa zone d'incertitude, soit par l'algorithme de One-Point Ransac.

4.5.2 Sélection d'un amer

De manière à garantir un résultat optimal, l'ensemble des amers observables doit être corrigé. Or, la correction est l'étape la plus coûteuse du filtre de Kalman. La correction d'un trop grand nombre de points peut rendre notre algorithme incompatible avec les applications temps réel. De plus, au-delà d'un certain nombre de corrections, le gain apporté est minime si les amers ont été correctement choisis. Une politique de sélection d'amers doit être mise en place. Celle-ci peut être basée sur la taille de la zone d'incertitude de l'amer. Plus cette zone est importante et plus l'information apportée par la correction sera notable. Réciproquement, l'information apportée par une correction d'un amer ayant déjà une faible incertitude sera d'autant plus minime, mais sera par contre plus robuste du fait de la bonne estimation de l'amer.

La taille de la zone est proportionnelle aux demi-axes a et b, et selon la formulation 4.6, il suffit de sélectionner l'amer maximisant le déterminant de Z_i.

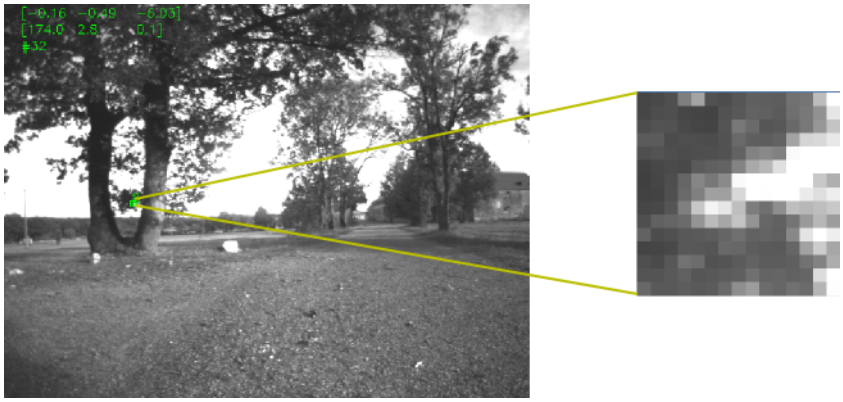


FIGURE 4.5 – Lors de l'initialisation d'un amer dans l'image (à gauche), une zone autour de celui-ci est mémorisée (à droite). Dans notre application, cette zone est un carré de 15 pixels de côté centré sur le pixel d'intérêt. Ce descripteur sert de *signature* de l'amer afin de pouvoir l'identifier d'image en image.

Les méthodes One-Point RANSAC et Recherche Active sont complémentaires. Tandis que la première permet une linéarisation correcte du filtre grâce à un amer robuste, la seconde permet une correction importante de la carte.

4.5.3 Recherche Active

Une fois un amer choisi, on effectue une opération de recherche active (ou *Active Search*). Cette opération nécessite deux paramètres :

- un *descripteur* (ou patch), illustré en figure 4.5 qui est la région de quelques pixels autour du point d'intérêt sélectionné pour être initialisé dans le filtre de Kalman. Cette région est considérée comme la *signature* de cet amer, lui permettant une caractérisation sur le long-terme [Davison et al., 2007].
- la zone de recherche, directement donnée par la matrice de covariance de l'innovation Z_i.

L'objectif de l'opération de recherche active est de corréler le descripteur de l'amer dans la zone de recherche afin de retrouver les coordonnées du pixel dans l'image qui maximise un score de corrélation.

Le score de corrélation est communément obtenu par un calcul de ZNCC, acronyme de *Zero-mean Normalized Cross Correlation* [Stefano et al., 2005] :

$$\mathrm{ZNCC}(p_1,p_2) = \frac{\sum_{x,y}\left(\left[I(u_1-x,v_1-y)-\overline{I}\right]\left[I'(u_2-x,v_2-y)-\overline{I'}\right]\right)}{\sqrt{\sum_{x,y}\left(\left[I(u_1-x,v_1-y)-\overline{I}\right]^2\sum_{x,y}\left(\left[I'(u_2-x,v_2-y)-\overline{I'}\right]^2\right.\right.}} \tag{4.7}$$

avec $\mathrm{ZNCC}(p_1,p_2) \in [-1,1]$ le score obtenu pour l'appariement du patch p_1 avec p_2, I et I' les fonctions Intensité dans les images contenant respectivement p_1 et p_2, et $\overline{I}$ et $\overline{I'}$ leurs moyennes respectives.

Si le résultat du ZNCC est au-delà d'un seuil minimum (dans notre cas supérieur à 0,7), la corrélation peut être considérée comme valide. Le calcul du barycentre sur les 4-voisins de la

FIGURE 4.6 – Images intervenant lors de la corrélation. A gauche, le descripteur correspondant à l'amer lors de son initialisation. Au centre, la prédiction de ce descripteur au moment de la corrélation et après mouvement du système, et à droite, la position dans la zone d'incertitude où ce descripteur prédit a été trouvé en maximisant le score ZNCC.

valeur maximale nous permet d'obtenir une valeur décimale sur u et v, et de réaliser ainsi une corrélation sub-pixellique. Nous obtenons ainsi l'observation $\mathbf{y}_i$ de l'amer $\mathcal{L}_i$ et pouvons calculer son innovation $\mathbf{z}_i$ présentée en 3.7 :

$$\mathbf{z}_i \quad = \quad \mathbf{y}_i - \mathbf{h}(\hat{\mathcal{R}}, \hat{\mathcal{L}}_i) \tag{4.8}$$

Remarque 4.1 *Notre système se déplaçant, rapidement, le descripteur initialement appris lors de la première observation de l'amer ne correspond plus à ce qui est perçu par la caméra. Prenons l'exemple d'un amer initialisé à l'instant t_0 avec son descripteur. Si le mobile se déplace en avançant dans la direction de cet amer, la différence entre le descripteur d'origine et ce qui est perçu dans l'image est de plus en plus imparfaite du fait de l'échelle en particulier, même si l'amer reste visible et de bonne qualité. Une solution proposée par [Davison et al., 2007] pour pouvoir corréler l'amer sur un plus long terme est d'effectuer une prédiction de son apparence à l'instant de sa corrélation à l'instant t_n. Pour cela, nous effectuons une transformation affine sur chaque pixel (u, v) du descripteur enregistré à l'instant t_0 :*

$$\begin{bmatrix} u_{t_n} \\ v_{t_n} \end{bmatrix} = \begin{bmatrix} h \cdot \cos(\alpha) & h \cdot \sin(\alpha) \\ -h \cdot \sin(\alpha) & h \cdot \cos(\alpha) \end{bmatrix} \begin{bmatrix} u_{t_0} \\ v_{t_0} \end{bmatrix}$$

avec h le facteur d'homothétie entre la position du capteur à t_0 et sa position à t_n et α la rotation sur l'axe de lacet entre les deux positions.

Une fois la position prédite de chaque pixel du descripteur obtenue, nous effectuons une interpolation bilinéaire pour récupérer l'intensité de chaque pixel du descripteur en y_n à partir de ses intensités à l'instant t_0. Ces opérations sont illustrées en figure 4.6 : à gauche, le descripteur extrait de l'image lors de l'initialisation du point, au centre, le descripteur prédit au moment de la correction de l'amer correspondant et à droite le lieu de la zone d'incertitude où la corrélation est maximisée.

4.5.4 Distance de Mahalanobis

En principe, comme présenté 4.5.1 en la zone de recherche d'un amer dans l'image est bornée par sa zone d'incertitude à 3σ. Néanmoins, en fonction des algorithmes de recherche, il peut arriver que le point corrélé soit à l'extérieur de cette région. De manière à savoir si le point observé est cohérent avec l'incertitude prédite, nous utilisons la distance de *Mahalanobis*. Comme toute mesure de distance, elle permet de mesurer une différence, ici entre l'observation

d'un point et sa prédiction, et tient également compte du bruit de mesure. Elle permet de détecter ainsi les mesures aberrantes, et dans le cas général, elle est définie par :

$$\mathcal{D}_M = \sqrt{(x-\bar{x})^\mathsf{T}\Sigma^{-1}(x-\bar{x})} \tag{4.9}$$

où x est la mesure d'un état tel que $x = (x_1 \quad x_2 \quad \dots \quad x_n)^\mathsf{T}$, $\bar{x}$ l'estimée courante de cet état $\bar{x} = (\bar{x}_1 \quad \bar{x}_2 \quad \dots \quad \bar{x}_n)^\mathsf{T}$ et Σ la matrice de covariance de $(x-\bar{x})$. Dans notre cas, la différence entre l'observation et la prédiction est directement donnée par l'innovation $\mathbf{z}$ (4.8) et la matrice de covariance entre les deux séries de variables par $\mathbf{Z}$ (4.6). La distance de Mahalanobis s'écrit alors simplement :

$$\mathcal{D}_M = \sqrt{z^\mathsf{T} Z^{-1} z}$$

Selon l'hypothèse d'un filtrage d'un système linéaire gaussien cohérent, $\mathcal{D}_M^2$ suit la loi du χ^2 avec le nombre de degrés de liberté donné par la dimension de $\mathbf{z}$. Dans le cas 2D, nous vérifions généralement que $\mathcal{D}_M \leq 3$ pour être dans la région à 98,9% de probabilité.

4.5.5 One-Point RANSAC

La sélection de l'amer à corriger peut également être réalisée grâce à l'algorithme appelé One-Point RANSAC (pour RANdom SAmple Consensus) [Civera et al., 2009]. Contrairement à un algorithme de RANSAC classique tel que détaillé en section 5.6, l'hypothèse 1-point permet de réduire considérablement le nombre d'hypothèses aléatoires nécessaires pour garantir la cohérence de l'amer à corriger. L'idée est de corriger un amer ayant une très faible innovation, de manière à linéariser le filtre en une valeur sûre afin d'améliorer sa cohérence.

L'algorithme One-Point RANSAC s'effectue selon les étapes suivantes :

- Tout d'abord, définissons $\mathcal{L}_N, N \in [1..n]$ l'ensemble des amers présents dans la carte SLAM. Un sous-ensemble $\mathcal{L}_K, K \in N$ est obtenu après la projection des amers $\mathcal{L}_N$, composé uniquement des amers projetés dans les limites du cadre de l'image. Un amer $\mathcal{L}_i$ est choisi aléatoirement (random) dans l'ensemble $\mathcal{L}_K$, ensemble des amers projetés dans l'image.
- On lui applique l'algorithme de *Recherche Active* 4.5.3 pour obtenir l'observation de ce point.
- Conjointement au calcul de l'innovation, on vérifie que l'observation de l'amer est compatible avec sa prédiction grâce à la distance de Mahalanobis 4.5.4.
- Si ce test est positif, on calcule dans un vecteur temporaire la mise à jour du vecteur d'état du filtre de Kalman, tel que décrit en 3.10 :

$$\hat{X'}^+ \quad = \quad \hat{X} + \mathbf{K}_i \cdot \mathbf{z}_i \tag{4.10}$$

  De cette manière, on réalise la correction du filtre de Kalman uniquement sur le vecteur d'état et nous n'avons pas à gérer la complexité des multiplications matricielles de la mise à jour de la covariance $\mathbf{P}$ du filtre.
- Nous effectuons une seconde projection des amers du sous-ensemble $\mathcal{L}_K$ avec les paramètres issus du nouveau vecteur d'état $\hat{X'}^+$.
- Nous calculons, pour chaque amer, la différence entre les deux projetés et construisons une liste de ceux ayant cette différence inférieure à un seuil. Ainsi nous mémorisons les amers cohérents avec le choix de $\mathcal{L}_i$ pris pour la correction.
- Finalement, quand tous les amers de $\mathcal{L}_K$ ont été traités, nous choisissons l'amer qui possède la liste des points cohérents la plus importante et au-delà d'une valeur minimale.

Pour accélérer cette procédure, comme cela est classique pour RANSAC, nous pouvons arrêter dès qu'on tire un amer $\mathcal{L}_i$ dans $\mathcal{L}_k$ qui permet une correction compatible avec un nombre d'amers supérieur à un certain seuil.

4.5.6 Correction

Nous disposons maintenant de l'ensemble des informations nécessaires pour la correction du filtre de Kalman. En premier lieu, il nous faut calculer le gain du filtre à l'aide de l'équation 3.9. Grâce à la forme éparse des jacobiennes, le calcul du gain $\mathbf{K}_i$ peut être récrit sous la forme :

$$\mathbf{K}_i = \begin{bmatrix} \mathbf{P}_{\mathcal{R}\mathcal{R}} & \mathbf{P}_{\mathcal{R}\mathcal{L}_i} \\ \mathbf{P}_{\mathcal{M}\mathcal{R}} & \mathbf{P}_{\mathcal{M}\mathcal{L}_i} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{H}_{\mathcal{R}}^{\mathsf{T}} \\ \mathbf{H}_{\mathcal{L}_i}^{\mathsf{T}} \end{bmatrix} \cdot \mathbf{Z}_i^{-1}$$

La mise à jour du vecteur d'état est réalisée grâce à l'équation 3.10 et celle de la matrice de la covariance par 3.11 :

$$\begin{aligned} \hat{X}^+ &= \hat{X} + \mathbf{K}_i \cdot \mathbf{z}_i \\ \mathbf{P}^+ &= \mathbf{P} - \mathbf{K}_i \cdot \mathbf{Z}_i \cdot \mathbf{K}_i^{\mathsf{T}} \end{aligned}$$

4.5.7 Reparamétrisation

Cette étape, facultative, permet de modifier la paramétrisation d'un amer. Son but est de réduire le nombre d'états qu'un amer utilise et diminuer, par ce fait, le nombre de calculs lors des opérations matricielles.

Un amer adopte soit une paramétrisation *IDP* de 6 composantes, soit une paramétrisation *AHP* de 7 composantes. Une fois le point reparamétré en euclidien, celui-ci n'utilisera plus que 3 composantes pour pouvoir être défini. Le traitement sera alors plus rapide car le nombre d'éléments à multiplier au sein des opérations matricielles est plus faible. Dans notre cas de SLAM sur un système embarqué, avec une taille fixe du nombre d'états, cette opération offre la possibilité d'initialiser de nouveaux points dans le filtre grâce à l'espace mémoire libéré et donc d'améliorer la précision du SLAM.

Un point AHP ou IDP sera reparamétré si la croyance sur sa position est jugée suffisamment forte et si la linéarisation de la fonction d'observation est considérée comme valide au fil du temps. Nous effectuons donc le test proposé par [Montiel, 2006]. Pour cela, nous évaluons l'expression :

$$L_d = \frac{4 \cdot \sigma_d}{d} |cos\alpha| \tag{4.11}$$

où $\sigma_d = \frac{\sigma_\rho}{\rho^2}$ peut être considéré comme un écart-type normalisé par rapport à la distance d, distance entre l'estimée du point AHP ou IDP dans la carte et celle du centre optique de la caméra $\mathcal{C}$ à l'instant courant :

$$d = \|\mathbf{p} - \mathcal{C}\|$$

et où

$$cos\alpha = \frac{\mathbf{v}^{\mathsf{T}} \times ((\mathbf{p}_0 + \mathbf{v}/\rho) - \mathcal{C})}{d \times \|\mathbf{v}\|}$$

représente le cosinus de l'angle entre les droites formées d'une part, par le centre optique et le point lors de son initialisation et, d'autre part, par le centre optique et le point à l'instant

courant.

Pour que la linéarisation soit valide, on doit avoir $L_d \approx 0$. Il suffit alors de tester si $|L_d|$ est inférieur à un seuil très faible (dans notre cas 0,1). Dans ce cas, on considère que pour ce point, le système est linéarisable en coordonnées cartésiennes. Son codage en AHP ou IDP ne se justifie plus. L'amer est alors reparamétré en euclidien par les fonctions de transformation 4.1 dans le cas d'un amer IDP :

$$\mathbf{p} = \mathbf{p}_0 + v^*(\epsilon, \alpha)/\rho$$

et par 4.2 dans le cas d'un amer AHP :

$$\mathbf{p} = \mathbf{p}_0 + v/\rho$$

4.6 Initialisation

Dans l'idéal, il serait intéressant de traiter l'ensemble des pixels d'une image pour en tirer le maximum d'informations et fournir une robustesse maximale à notre système. Malheureusement, une telle opération prendrait un temps considérable. Nous devons alors nous intéresser à une poignée de pixels remarquables dans les images qui doivent répondre aux critères ci-dessous pour être sélectionnés :

- **Isolé** : Un amer isolé dans une région de l'image apportera subjectivement plus d'informations qu'un ensemble d'amers regroupés dans une même région d'image,
- **Remarquable** : Il doit être repéré facilement dans l'image,
- **Réobservable** : L'amer peut être suivi d'image en image.

Ce processus d'initialisation d'amers dans la carte SLAM est présenté en figure 4.7 et est développé dans les sous-sections suivantes.

Remarque 4.2 *Avant de rajouter un amer dans la carte, il faut d'abord vérifier si celle-ci n'est pas déjà complète, puisque notre vecteur d'état et notre carte SLAM ont des tailles fixe.*

4.6.1 La tessellation

La tessellation, ou *pavage*, est une méthode de partition d'un espace bi- ou tridimensionnel. De manière concrète, nous allons diviser l'image en plusieurs sous-images pour n'avoir à traiter qu'une seule de ces régions. L'objectif de cette division est multiple : en segmentant ainsi l'image, nous obtenons une sorte de grille d'occupation mémorisant les projections dans l'image des amers déjà présents dans la carte. Ainsi, nous connaissons les secteurs de l'image où des amers existants sont déjà présents et nous sélectionnons une région qui en est dépourvue. En effet, il est préférable de répartir au mieux les amers dans l'image de manière à garantir une meilleure observabilité du mouvement. Prenons l'exemple d'un ciel ou d'un horizon très contrasté où des points d'intérêt sont facilement perceptibles. Bien que l'observation de ceux-ci au cours de la séquence peut donner une bonne information sur la rotation du mobile, la translation, quand à elle n'est pas observable. Dans le cas contraire, si nous initialisons des amers régulièrement répartis dans l'image, nous offrons au SLAM une meilleure observation de la scène et du mouvement. Le second avantage est que la détection d'un point d'intérêt dans une région sera beaucoup plus rapide que le traitement de l'image en entier.

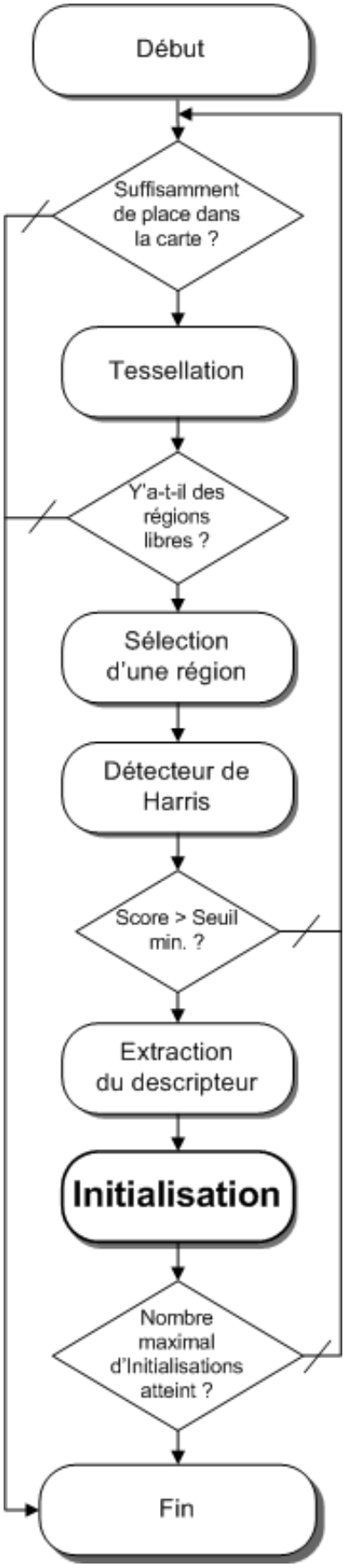


FIGURE 4.7 – Algorithme proposé pour l'initialisation de nouveaux amers.

4.6.2 Le détecteur de points

Lorsqu'une région a été sélectionnée, il faut non seulement pouvoir détecter un point rapidement, mais ce point doit aussi avoir des particularités identifiables, voire uniques, pour pouvoir être réobservé dans les images suivantes. Nous allons nous intéresser aux gradients de l'image et plus particulièrement aux *coins* formés par des intersections. Ces points ont la particularité d'avoir un gradient important qui est orienté dans deux directions. De plus, ces coins sont relativement uniques dans une région de l'image.

Parmi les différents détecteurs existants, celui de *Harris* [Harris and Stephens, 1988] est certainement le plus populaire car il est à la fois robuste et rapide. La première étape de ce détecteur de points est le calcul des dérivées horizontales et verticales pour chaque pixel de l'image :

$$\begin{aligned}\mathbf{I}_u(u,v) &= -I(u-1,v) + I(u+1,v)\\ \mathbf{I}_v(u,v) &= -I(u,v-1) + I(u,v+1)\end{aligned}$$

Ces dérivées servent à la constitution d'une matrice $\mathbf{M}$:

$$\mathbf{M}(u,v) = \mathbf{g}_m(\sigma) \otimes \begin{bmatrix} \mathbf{I}_u^2 & \mathbf{I}_u\mathbf{I}_v \\ \mathbf{I}_u\mathbf{I}_v & \mathbf{I}_v^2 \end{bmatrix}$$

où $\mathbf{g}_m(\sigma)$ est un masque gaussien de taille $m \times m$ et de variance σ^2. De manière concrète, la matrice $\mathbf{M}(u,v)$ contient le score de saillance du pixel (u,v). Le score de Harris pour ce pixel est obtenu par :

$$\mathbf{H}(u,v) = det(\mathbf{M}(u,v)) - k \cdot (\text{trace}(\mathbf{M}(u,v)))^2$$

avec $k \in [0,040, 15]$. Il suffit ensuite de seuiller les scores obtenus pour ne retenir que le ou les meilleurs points dans la région sélectionnée.

Une amélioration de ce détecteur a été implémentée dans notre algorithme, et fut proposée par Shi et Tomasi [Shi and Tomasi, 1994]. Dans cette méthode, le score est obtenu par :

$$\mathbf{S}(u,v) = m_{uu} + m_{vv} - \sqrt{(m_{uu}^2 - m_{vv}^2 + 4 \cdot m_{uv}^2)}$$

Ces détecteurs ont prouvé leurs qualités durant de nombreuses années et sont encore aujourd'hui très utilisés. De très nombreux auteurs ont proposé des améliorations sur la détection des points d'intérêts (par exemple, *Features from Accelerated Segment Test* (FAST) est plus rapide et plus précis que Harris, utilisé notamment dans l'algorithme de *Parallel Tracking And Mapping* (PTAM) [Klein and Murray, 2007]). Comme dans notre application, présentée en chapitre 6, la détection des points est traitée sur un accélérateur matriciel, nous n'avons pas testé d'autres détecteurs que ceux introduits ici : il est connu que les détecteurs de type *Différence de Gaussiennes* (DOG) sont imprécis. Par contre nous avons étudié comment ces détecteurs se comportent sur des images acquises en infrarouge lointain en figure 4.8. Sur cette bande spectrale, les contours des différents objets et éléments de la scène ont tendance à être relativement diffus. Or, plus les bords des objets sont nets et contrastés, plus les points sont de bonne qualité.

Hajebi a comparé les résultats fournis par plusieurs détecteurs sur des images infrarouge bande III [Hajebi and Zelek, 2008]. Elle remarque que, si ces détecteurs fournissent d'excellents résultats pour les images du spectre visible, les résultats sont plus discutables sur les images infrarouges, notamment au vu du nombre de points trouvés et de la robustesse. Elle utilise

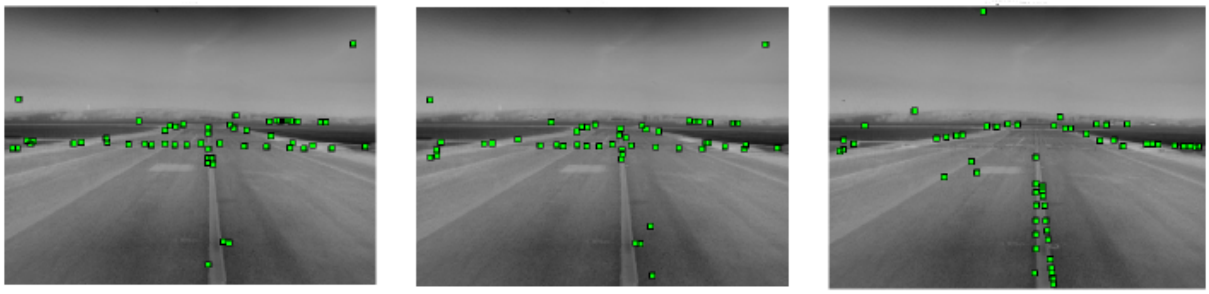


FIGURE 4.8 – Trois détecteurs ont été comparés sur des images infrarouges de notre application. Ces images ont été préalablement corrigées en distortion et leurs histogrammes égalisés. Nous avons paramétré les seuils pour récupérer approximativement le même nombre de points avec chaque détecteur. A gauche le détecteur de points de *Harris*, au centre celui de *Shi et Tomasi* et à droite le détecteur de *Congruence de Phases*.

alors le détecteur par *Congruence de Phases* dans son application de stéréovision qui fournit de meilleurs résultats sur des images infrarouges.

Contrairement aux approches classiques se basant sur l'intensité et supposant que les points d'intérêt sont ceux possédant un gradient maximal, le détecteur par Congruence de Phases [Morrone and Owens, 1987] repose sur un modèle d'énergie local et sélectionne les points d'intérêt où leurs composantes de Fourier sont les plus en phase.

Dans sa thèse, Ducarouge [Ducarouge, 2011] compare également ces détecteurs sur des images infrarouges et remarque que le nombre de points extraits est équivalent lors de l'utilisation d'un détecteur par Congruence de Phases et de Harris. Il note également que les points sélectionnés par ce dernier sont mieux répartis dans l'image que dans le cas d'un détecteur par Congruence de Phases.

Cependant, le calcul de la Congruence de Phases demande beaucoup de ressources opératoires et reste très sensible au bruit. De plus, notre application SLAM ne demande que peu de points comparé à une application de stéréovision par exemple. Un détecteur de Harris ou de Shi-Tomasi suffit pour extraire d'une image infrarouge suffisamment de points contrastés.

4.6.3 Descripteur

La réobservation des points d'image en image est une étape fondamentale du SLAM. En effet, le suivi des amers déjà intégrés dans la carte apporte la stabilité et la convergence du filtre de Kalman, c'est-à-dire la précision du SLAM. La réobservation d'un amer faite quelques pixels au-delà de sa position réelle dans l'image peut entraîner une corruption du filtre et la divergence de l'algorithme.

Le but est de pouvoir identifier, pendant le déplacement du robot, donc de la caméra, les mêmes indices visuels. Pour cela, nous adoptons l'approche de Davison *et al.* [Davison and Murray, 1998], en utilisant des descripteurs relativement petits, de 15 pixels de côté, centrés autour des pixels lors de leur initialisation. Ces descripteurs sont à la fois propres au point détecté et beaucoup plus précis que cet unique point. Appelés également *signature du point*, ces descripteurs servent à caractériser le pixel sur le long terme car ils sont utilisés comme détecteur lors de la recherche du point réalisée par le corrélateur ZNCC présenté en 4.5.3. Afin de suivre l'amer plus longtemps, il est important d'appliquer une transformation à son descripteur, dépendant du point de vue, comme présenté en remarque 4.1. Dans ce domaine des

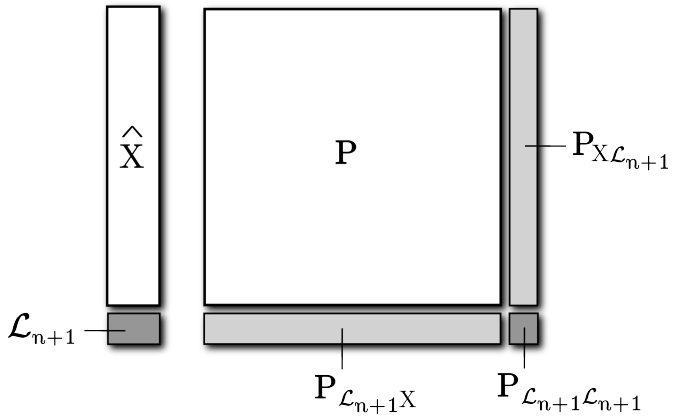


FIGURE 4.9 – Ajout d'un nouvel amer dans la carte.

descripteurs, notons aussi un grand nombre de travaux dans la communauté Vision : SIFT, SURF, Census, BRIEF, ORB... Souvent, comme pour SIFT, ces descripteurs sont plutôt adaptés à la reconnaissance d'objets ou à l'indexation d'images, sur une base d'un apprentissage préalable. La nécessité d'adapter le descripteur au point de vue nous a conduit à conserver le classique voisinage, centré sur le point d'intérêt.

4.6.4 Initialisation

Nous disposons maintenant de l'ensemble des informations pour initialiser un amer $\mathcal{L}_{n+1}$ dans la carte SLAM, qui consiste en l'ajout de celui-ci dans le vecteur d'état X et ses covariances associées dans $\mathbf{P}$ comme illustré en figure 4.9 :

$$X^+ = \begin{bmatrix} \hat{X} \\ \hat{\mathcal{L}}_{n+1} \end{bmatrix} \quad \text{et} \quad \mathbf{P}^+ = \begin{bmatrix} \mathbf{P} & \mathbf{P}_{\mathcal{L}_{n+1}X}^\mathsf{T} \\ \mathbf{P}_{\mathcal{L}_{n+1}X} & \mathbf{P}_{\mathcal{L}_{n+1}\mathcal{L}_{n+1}} \end{bmatrix}$$

La moyenne est donnée par la fonction de rétro-projection définie en fonction du type de paramétrisation utilisé. Elle est définie en 4.3 et d'après 3.3.3, cette fonction s'écrit :

$$\hat{\mathcal{L}}_{n+1} = \mathbf{g}(\hat{\mathcal{R}}, \mathcal{C}, \mathbf{y}_{n+1}, \hat{\mathbf{c}})$$

Les nouvelles covariances de $\mathbf{P}$ s'expriment selon :

$$\begin{aligned} \mathbf{P}_{\mathcal{L}_{n+1}X} &= \mathbf{G}_\mathcal{R}\mathbf{P}_{\mathcal{R}X} \\ \mathbf{P}_{\mathcal{L}_{n+1}\mathcal{L}_{n+1}} &= \mathbf{G}_\mathcal{R}\mathbf{P}_{\mathcal{R}\mathcal{R}}\mathbf{G}_\mathcal{R}^\mathsf{T} + \mathbf{G}_{\mathbf{y}_{n+1}}\mathbf{R}\mathbf{G}_{\mathbf{y}_{n+1}}^\mathsf{T} + \mathbf{G}_\mathbf{c}\mathbf{C}\mathbf{G}_\mathbf{c}^\mathsf{T} \end{aligned}$$

avec :

$$\begin{aligned} \mathbf{G}_\mathcal{R} &= \frac{\partial \mathbf{g}(\hat{\mathcal{R}}, \mathcal{C}, \mathbf{y}_{n+1}, \hat{\mathbf{c}})}{\partial \mathcal{R}} \\ \mathbf{G}_{\mathbf{y}_{n+1}} &= \frac{\partial \mathbf{g}(\hat{\mathcal{R}}, \mathcal{C}, \mathbf{y}_{n+1}, \hat{\mathbf{c}})}{\partial \mathbf{y}_{n+1}} \\ \mathbf{G}_\mathbf{c} &= \frac{\partial \mathbf{g}(\hat{\mathcal{R}}, \mathcal{C}, \mathbf{y}_{n+1}, \hat{\mathbf{c}})}{\partial \mathbf{c}} \end{aligned}$$

FIGURE 4.10 – Equipements expérimentaux utilisés au LAAS pour l'acquisition des données. (a) Robot d'exploration *Mana*. (b) Caméra Flea2 IEEE1394 monochrome développée par *PointGrey*, avec une résolution de 640×480. (c) Caméra *A10* IEEE1394 infrarouge bande III développée par *Flir*, avec une résolution de 160×120. (d) Centrale Inertielle produite par *Xsens*.

où $\mathbf{G}_{\mathcal{R}}$ est la jacobienne de la fonction de rétro-projection associée à la paramétrisation de l'amer par rapport à la pose du robot, $\mathbf{G}_{\mathbf{y}_{n+1}}$ est la jacobienne de la fonction de rétro-projection par rapport au pixel de l'image et $\mathbf{G}_{\mathbf{c}}$ la jacobienne de la fonction de rétro-projection par rapport à la distance inverse donnée *a priori*.

4.7 Résultats SLAM Monoculaire

4.7.1 Résultats SLAM Monoculaire Visible

Dans cette section, nous présentons les résultats qualitatifs de la version *C-SLAM* de l'algorithme présenté ci-dessus [Gonzalez et al., 2011]. Cette version a été spécialement développé pour les systèmes embarqués.

Nous comparerons, dans un premier temps, les trajectoires obtenues dans le cas du visible et de l'infrarouge avec la vérité terrain. Trois séquences ont été sélectionnées pour évaluer l'algorithme : deux séquences d'environ quarante mètres pour évaluer le modèle à vitesse constante dans le cadre du SLAM monoculaire visible et infrarouge. Ces deux séquences ont été tournées sur le parking du LAAS. La troisième séquence a été prise à *Caylus* où le robot fait une boucle d'environ 500 mètres en tout-terrain. Celle-ci est suffisamment longue pour évaluer le SLAM avec un modèle de mouvement inertiel, avec ou sans correction GPS [1]. Notons que même si le robot revient à sa position de départ, il n'y a pas de fermeture de boucle dans la mesure où les amers perçus au début de la séquence sont supprimés de la carte avant la fin de la séquence, cela pour satisfaire des contraintes de mémoire bornée sur un système embarqué. Nous étudierons, ensuite, la cohérence de cet algorithme grâce au score *Normalised Estimation Error Squared* (NEES).

Lors de l'acquisition des séquences, une attention particulière a été portée sur l'horodatage de chaque donnée avec une précision de l'ordre de la micro-seconde, de manière à pouvoir rejouer la séquence en temps réel lors des exécutions différées du SLAM en relisant les images

1. Nous remercions Cyril Roussillon qui a intégré de nombreux outils sur le robot MANA du LAAS-CNRS afin de réaliser ces acquisitions, et qui les a mises à disposition.

stockées sur le disque. Notons que cet horodatage précis est obtenu sans exploiter de logiciels de type *Data Logger* comme *Effibox* (société *Effidence*) ou *RT-MAPS* (société *Intempora*).

Les résultats en terme de vitesse de traitement sont traités dans le chapitre 6 dédié aux systèmes embarqués.

Pour valider de manière expérimentale notre algorithme, nous traitons un jeu de données acquis par le robot d'exploration *Mana* présenté en figure 4.10(a). Dans le cas du SLAM monoculaire seul, nous utilisons le modèle de mouvement à vitesse constante sur les images monochromes acquises par la caméra IEEE1394 (*firewire*) Flea2 de *PointGrey*, avec une résolution de 640×480 et cadencée à 60Hz (figure 4.10(b)).

La carte SLAM peut contenir jusqu'à 300 états, et 8 corrections sont faites à chaque itérations par *Recherche Active*. Quatre amers sont initialisés à chaque itération dans la limite de la place disponible dans la carte. Ces amers adoptent une paramétrisation AHP.

Durant la séquence présentée en figure 4.11(a), le robot a réalisé une boucle d'approximativement 40 mètres. La figure 4.11(b) compare la trajectoire estimée (en rouge) par le SLAM sur le plan (XY) avec la vérité terrain (en bleu) obtenue par un GPS-RTK (centimétrique). Nous pouvons remarquer que cette trajectoire est de bonne qualité, confirmée par l'erreur faite sur X et Y inférieure au mètre (figure 4.11(d)). Cependant, l'algorithme a plus de difficulté à observer la hauteur Z, ce qui est illustré en figure 4.11(c). L'erreur commise sur cet axe est alors plus importante (figure 4.11(d).

Le principal avantage de cette méthode est sa simplicité. En effet, une seule caméra et un simple modèle de mouvement à vitesse constante suffisent. Cependant, ce couple atteint vite ses limites. Le facteur d'échelle n'est pas observable, de même que le cap, qui ont dû être ajustés manuellement. De manière à conserver la consistance du filtre, un nombre suffisant d'appariements de bonne qualité doit être effectué à chaque itération. De plus, la zone de recherche du point par ZNCC peut être importante et augmenter rapidement, ce qui accroît le temps de calcul et le risque de faux-appariements. Dans le cas de faux-appariements, le risque de divergence du SLAM est conséquent, car aucun autre capteur ne peut confirmer ou infirmer les corrections effectuées. Le caractère aléatoire du SLAM peut faire que la même séquence peut diverger ou non, en fonction des points initialisés, de la qualité et du nombre de corrections.

Bien que cette séquence fonctionne en temps réel sur une machine classique, la taille de la carte et le nombre de corrections utilisés dans le cadre de ce test ne permettent pas d'obtenir cette application temps réel lors de son portage sur système embarqué. Cette problématique est étudiée en section 6.4.1.

4.7.2 Résultats SLAM Monoculaire Infrarouge

Dans le cas du SLAM monoculaire infrarouge, nous utilisons le modèle de mouvement à vitesse constante sur les images bande III (8μm - 12μm) acquises par la caméra IEEE1394 (*firewire*) A10 de *Flir*, avec une résolution de 160×120 et cadencée à 30Hz (figure 4.10(c)).

Nous utilisons ici les mêmes paramètres que pour le cas du visible : une carte SLAM de 300 états, quatre amers sont initialisés à chaque itération, et la paramétrisation AHP est adoptée. Par contre, à cause de la faible qualité et résolution des images, nous essayons de corriger 10 amers par image.

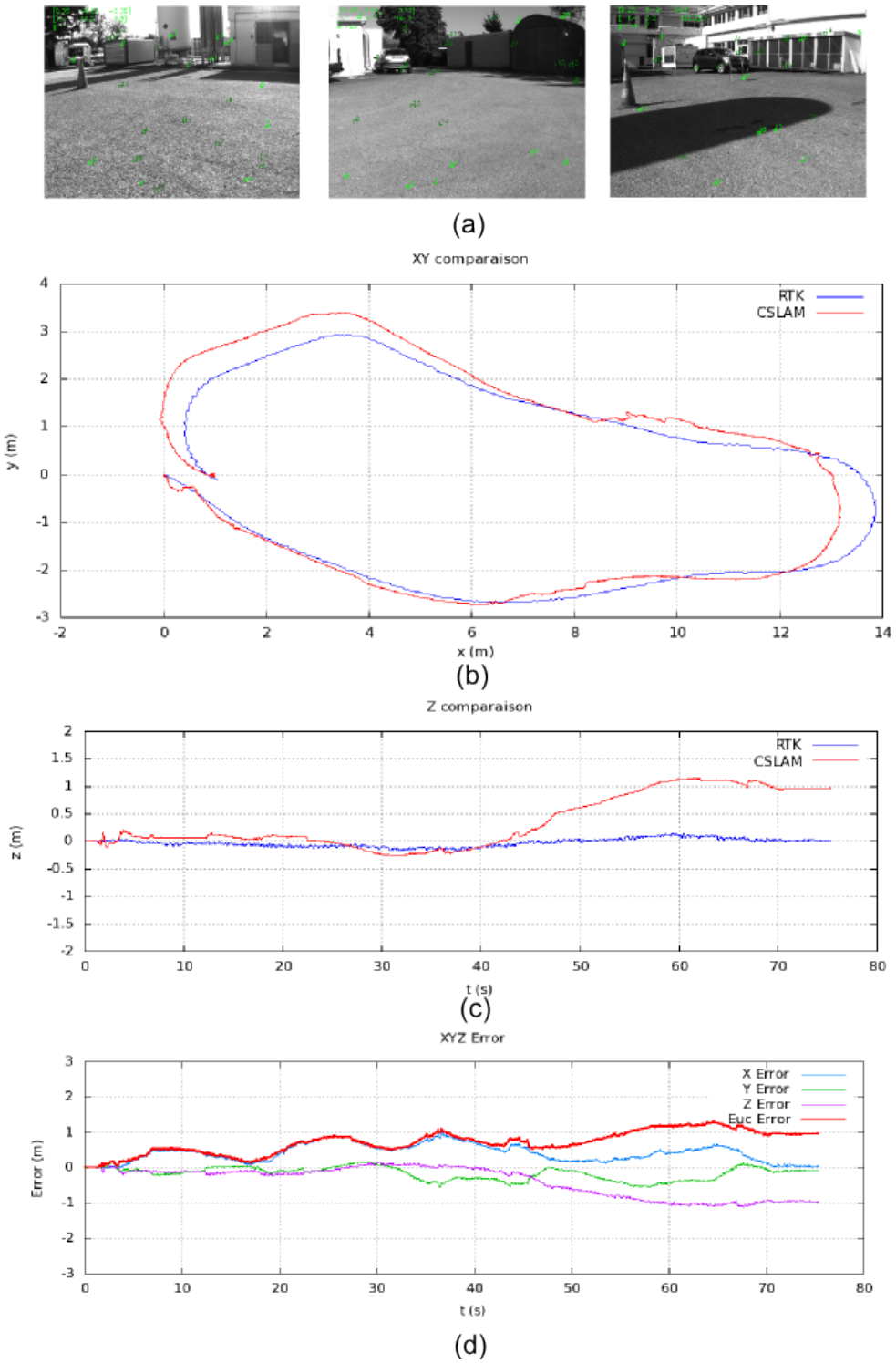


FIGURE 4.11 – Résultats du SLAM EKF Monoculaire visible. (a) Images de la séquence. (b) Trajectoire en XY obtenue (rouge) avec comparaison avec la vérité terrain (bleu). (c) Trajectoire en Z obtenue (rouge) et comparaison avec la vérité terrain (bleu). (d) Erreurs commises.

Après la séquence présentée en figure 4.12(a), le robot a réalisé une trajectoire d'approximativement 30 mètres. La figure 4.11(b) compare la trajectoire estimée par le SLAM sur le plan (XY) avec la vérité terrain obtenue par un GPS-RTK (centimétrique). Contrairement au cas du SLAM monoculaire visible, la qualité de la trajectoire est ici plus discutable, quel que soit l'axe étudié. La figure 4.12(d) qui présente l'ensemble des erreurs commises confirme bien cette précision imparfaite.

Néanmoins, l'information apportée par une caméra infrarouge est loin d'être non-négligeable. Par exemple, cette séquence a été tournée de nuit, où la caméra visible ne procurait aucune donnée exploitable. Plusieurs paramètres influent sur ces résultats, à commencer par la qualité des images infrarouges. L'aspect diffus et la faible dynamique de celles-ci complique la détection de coins de qualité, et les appariements sont plus délicats. Les images sont également peu résolues (160×120) et engendrent plus d'imprécision dans la localisation des amers, et donc du mobile. De plus, toutes les minutes, la caméra applique une Correction de Non Uniformité (ou NUC) afin de corriger certaines disparités au niveau du capteur. Cette opération réalise mécaniquement l'application d'un obturateur pendant une seconde, et signifie donc une perte d'information. Ce "saut" d'image peut alors entraîner une divergence de l'algorithme.

4.8 Le SLAM multi-capteurs

L'avantage du modèle de mouvement à *vitesse constante* pour l'étape de prédiction est sa simplicité. Malheureusement, ce modèle possède plusieurs inconvénients :

- Le facteur d'échelle n'est pas connu. Le paramètre de distance inverse doit être fourni *a priori* et peut conduire à de forts changements dans l'échelle.
- A cause de la faible précision du modèle, les zones de recherche dans l'image sont grandes. Par conséquent, les incertitudes au niveau de la position des amers sont aussi importantes.
- Des zones importantes augmentent également le risque de faux appariements lors de cette même recherche.
- Le modèle de mouvement se base sur la supposition d'une vitesse constante. Les dynamiques élevées sont donc difficiles à observer.

Afin d'assurer une meilleure robustesse aux systèmes de localisation, le nombre et le type de capteurs sont multipliés. Ces capteurs peuvent se diviser en deux grandes familles :

- Les capteurs proprioceptifs renseignant sur les déplacements propres du mobile.
- Les capteurs extéroceptifs informant sur l'environnement entourant le mobile et dont font par exemple partie le Velodyne illustré en figure 4.13 et la caméra.

Pour limiter la divergence due à la perte de profondeur dans le SLAM monoculaire et les différentes dérives inhérentes à l'utilisation d'un modèle à *vitesse constante*, les algorithmes de SLAM visuel ont régulièrement utilisé d'autres capteurs annexes. Avant la proposition de Davison pour l'utilisation d'amers visuels tridimensionnels en 2003 [Davison, 2003], les premiers algorithmes de SLAM visuel de Neira *et al.* [Neira et al., 1997] et Davison *et al.* [Davison and Murray, 1998] intégraient déjà des odomètres.

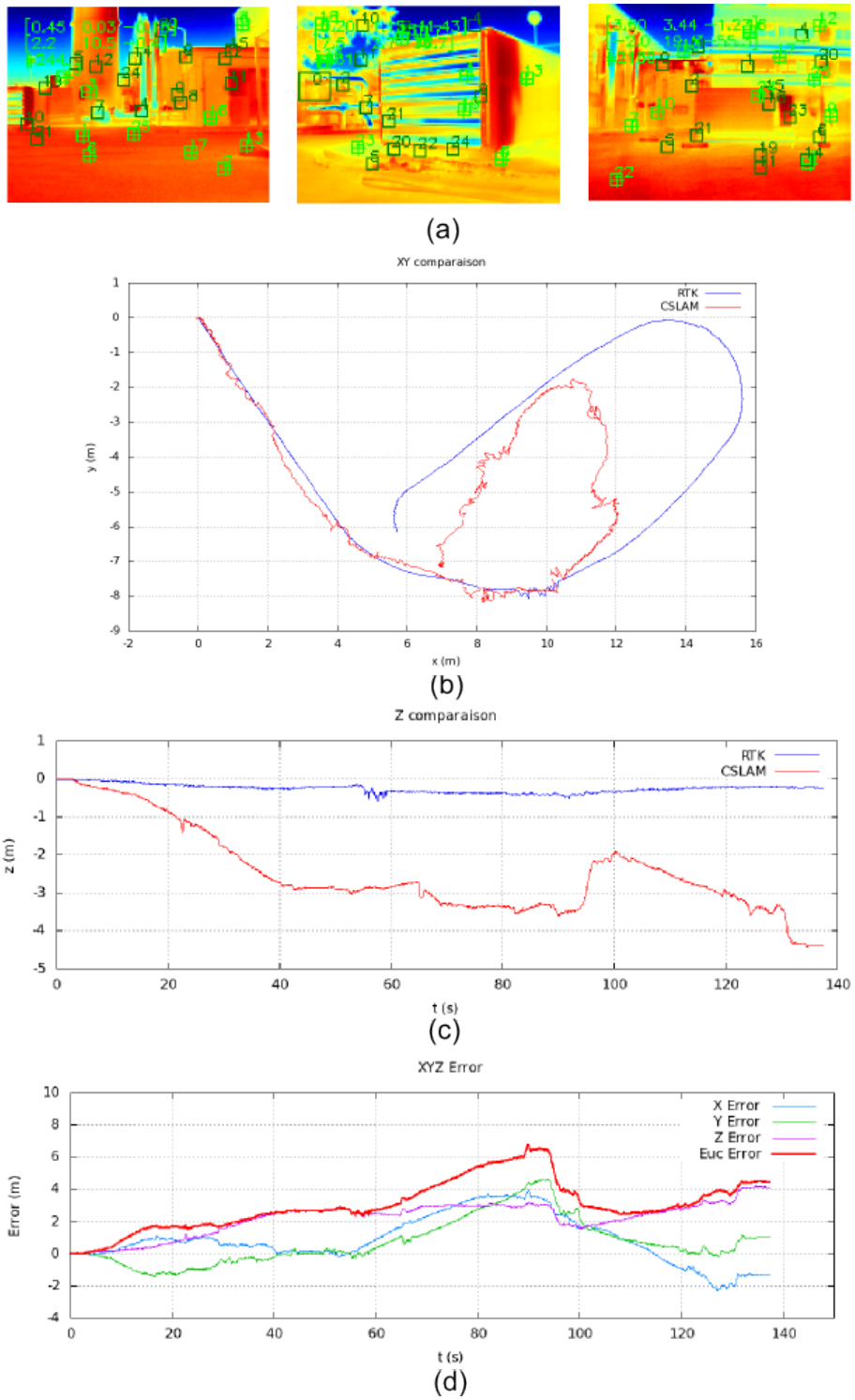


FIGURE 4.12 – Résultats du SLAM EKF Monoculaire infrarouge. (a) Images de la séquence. (b) Trajectoire en XY obtenue (rouge) avec comparaison avec la vérité terrain (bleu). (c) Trajectoire en Z obtenue (rouge) et comparaison avec la vérité terrain (bleu). (d) Erreurs commises.

FIGURE 4.13 – Le Velodyne, monté ici sur le toit de la *Google Car*, est un excellent capteur qui fournit une carte 3D à 360° de son environnement. Mais son coût élevé et son encombrement limite sa diffusion sur des applications spécifiques (sécurité civile ou militaire ...).

En 2004, Nygårds *et al.* [Nygårds et al., 2004] fusionnent, dans leur application, un GPS centimétrique, une centrale inertielle et un système de navigation (compas numérique, altimètre ...) au sein d'un même filtre de Kalman afin de localiser avec précision les objets détectés dans les images.

Agrawal *et al.* [Agrawal and Konolige, 2006] proposent en 2006 un algorithme de localisation fusionnant l'odométrie visuelle par stéréovision et un GPS naturel. Le GPS empêchant la dérive, l'erreur obtenue après une trajectoire d'une centaine de mètres n'est que de quelques mètres.

En 2011, Konolige *et al.* [Konolige et al., 2011] localisent un robot à quelques mètres après une trajectoire de près d'une dizaine de kilomètres. Pour atteindre ce résultat, ils implémentent une odométrie visuelle à partir d'un algorithme de SLAM bi-caméra et en exploitant des descripteurs SIFT. La centrale inertielle est intégrée par *couplage faible* (ou *couplage lâche*) au sein de leur application, c'est-à-dire que ses données sont utilisées pour fournir une estimation sur la position du mobile ; contrairement au couplage *couplage fort* (ou *couplage serré*) où les données brutes de la centrale inertielle sont directement exploitées dans le modèle dynamique du mobile.

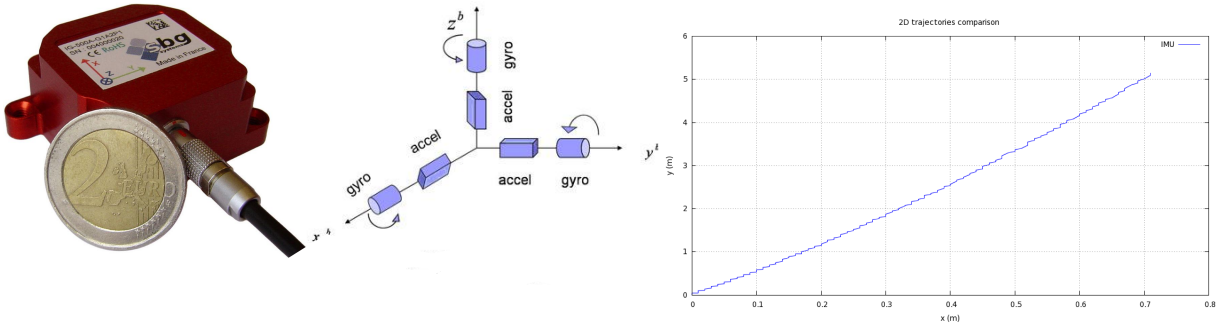
La même année, les algorithmes *RT-SLAM* [Roussillon et al., 2011] et *C-SLAM* sont proposés. Ils permettent de fusionner, au sein d'une même application SLAM, une ou plusieurs caméras, odomètres, centrales inertielles et GPS.

Dans l'application SART, l'aéronef est équipé d'une centrale inertielle bas coût et d'un GPS naturel. Nos évaluations basées sur les acquisitions réalisées par le robot Mana au LAAS, nous exploitons un GPS-RTK pour avoir la vérité terrain ; les données d'un GPS naturel sont alors simulées en bruitant la position fournie par le GPS-RTK par un bruit blanc gaussien. Cependant, en faisant cela, nous ne prenons pas en compte les aberrations qui polluent les mesures GPS (multi-réflexions...). Dans les sous-sections suivantes, nous présentons brièvement ces capteurs et comment nous traitons les données qu'ils fournissent pour assurer une meilleure robustesse à notre algorithme de SLAM.

4.8.1 Présentation des capteurs

La centrale Inertielle

La centrale inertielle est également appelée IMU pour *Inertial Measurement Unit*. La centrale *XSens* présentée en 4.14 (a) et utilisée dans le système SART, intègre trois accéléromètres et trois gyromètres mesurant respectivement les accélérations et les vitesses de giration selon



(a) (b) (c)

FIGURE 4.14 – (a) La centrale inertielle utilisée dans le système SART. (b) Les mesures des accéléromètres et gyromètres suivant le repère de la centrale. (c) Un exemple de divergence bidimensionnelle de notre centrale sur un système immobile.

chaque axe de l'espace (4.14 (b)). Elle intègre également des magnétomètres, mesurant le cap par rapport au Nord magnétique, mais ces magnétomètres ne sont pas considérés dans notre application.

Miniaturisées et très peu onéreuses, ces centrales *Micro Electro-Mechanical Systems* (MEMS) a rendu cette technologie accessible pour de nombreuses applications professionnelles ou grand public. Notons qu'il restera toujours des centrales haut de gamme, donc très onéreuses, réservées aux domaines des hautes technologies tels que l'aérospatiale ou le domaine sous-marin. Les centrales MEMS s'imposent de plus en plus dans notre quotidien et se retrouvent maintenant principalement dans nos smartphones, nos manettes de jeu et consoles portables... Malheureusement, ces centrales inertielles bas coût divergent très rapidement car les biais et les bruits s'intègrent à chaque mesure. En quelques secondes, la mesure n'est plus exploitable (4.14 (c)). Il faut particulièrement veiller au calibrage des biais lors de l'initialisation de notre système.

L'utilisation d'un tel capteur dans notre application va lui conférer un énorme gain en précision et en temps de calcul. Le modèle utilisé et présenté en section 4.8.3 mesure les accélérations linéaires et vitesses angulaires de notre système. Il permet donc d'apporter des informations précieuses sur la vitesse instantanée du système.

Le GPS

Le GPS, acronyme anglais pour *Global Positioning System*, est un système de positionnement par satellites. Il est certainement le capteur de localisation et de guidage le plus connu, maintenant intégré de série dans certains modèles de véhicules.

A partir de l'acquisition de signaux provenant d'au moins 4 satellites différents équipés de plusieurs horloges atomiques, le récepteur GPS calcule le temps de propagation de ces signaux entre les satellites et lui-même ; il connaît donc sa distance par rapport à ceux-ci. Par trilatération, il peut ainsi se situer précisément en trois dimensions sur n'importe quel point du globe. Le GPS fournit alors une coordonnée en longitude, latitude et altitude propre à un référentiel (le plus utilisé étant le référentiel *WGS84*) et avec une précision proche de la dizaine de mètres. Cette précision reste cependant relative au nombre de signaux de satellites différents reçus ainsi qu'à l'environnement du robot pouvant occasionner des réflexions multiples ou bloquer ces signaux GPS.

4.8.2 Prédiction ou Correction ?

Les données issues des capteurs peuvent être traitées à deux emplacements dans le SLAM EKF. Elles vont soit intervenir lors de l'étape de prédiction car elles sont intégrées au modèle de mouvement, soit servir lors de la correction du filtre. Utiliser un capteur pour la prédiction du mouvement va engendrer l'ajout d'états dans le modèle du robot. Toutefois, ce coût est amplement rentabilisé lors des étapes de recherche active. Par ailleurs, Roussillon cite plusieurs conditions à remplir pour pouvoir utiliser un capteur dans le modèle de prédiction [Roussillon and Lacroix, 2012] :

- Il est périodique et de fréquence plus élevée que tout autre capteur intégré dans le filtre,
- Il a un niveau raisonnable de bruit de manière à procurer un point correct de linéarisation,
- Ses données ne doivent être jamais défectueuses : les erreurs du capteur sont cohérentes avec ses incertitudes et doivent avoir des probabilités gaussiennes.

Ainsi, la centrale inertielle couvre l'ensemble de ces conditions pour permettre un couplage fort. Sa fréquence élevée (100 Hz) lui permet de produire des données précises et peu bruitées. Le point de linéarisation du filtre est donc correct et n'induit pas d'erreurs.
Pour le GPS, le même choix existe entre couplage faible (on fusionne l'estimée de position absolue donnée par le GPS) et fort (on exploite les mesures brutes GPS : pseudo-distances, phases...). Ici nous réalisons un couplage faible des données GPS.

4.8.3 Le modèle de mouvement Inertiel

Le modèle de mouvement inertiel est plus complexe que celui à vitesse constante et le vecteur d'état du robot est défini selon 19 paramètres :

$$\mathcal{R} = (\mathbf{p} \quad \mathbf{q} \quad \mathbf{v} \quad \mathbf{ab} \quad \mathbf{wb} \quad \mathbf{g})^\mathsf{T} \tag{4.12}$$

où $\mathbf{p} = (p_x \quad p_y \quad p_z)^\mathsf{T}$ et $\mathbf{q} = (q_w \quad q_x \quad q_y \quad q_z)^\mathsf{T}$ sont respectivement la position et l'orientation du robot dans l'espace, $\mathbf{v} = (v_x \quad v_y \quad v_z)^\mathsf{T}$ est sa vitesse linéaire, $\mathbf{ab} = (ab_x \quad ab_y \quad ab_z)^\mathsf{T}$ et $\mathbf{wb} = (wb_x \quad wb_y \quad wb_z)^\mathsf{T}$ sont respectivement les biais sur les accéléromètres et les gyromètres selon les trois axes, et enfin $\mathbf{g} = (g_x \quad g_y \quad g_z)^\mathsf{T}$ est la gravité.

Le modèle d'évolution $\mathbf{f}(\cdot)$ du robot exploitant les mesures inertielles s'exprime par :

$$\mathcal{R}^+ = \begin{bmatrix} \mathbf{p}^+ \\ \mathbf{q}^+ \\ \mathbf{v}^+ \\ \mathbf{ab}^+ \\ \mathbf{wb}^+ \\ \mathbf{g}^+ \end{bmatrix} = \begin{bmatrix} \mathbf{p} + \mathbf{v} \times \Delta t \\ \mathbf{q} \otimes Q((w_m - \mathbf{wb}) \times \Delta t + t_i) \\ \mathbf{v} + ((R(q) * \Delta t) * (a_m - \mathbf{ab}) + \mathbf{g}) * \Delta t + v_i \\ \mathbf{ab} + a_{bi} \\ \mathbf{wb} + w_{bi} \\ \mathbf{g} \end{bmatrix}$$

où $\otimes$ est le produit de quaternions, $Q(.)$ la fonction de transformation d'un vecteur en quaternion, $R(.)$ la fonction de transformation d'un quaternion en matrice de rotation, a_m et w_m les mesures des accéléromètres et gyromètres, et v_i, t_i, a_{bi} et a_{bi} sont des perturbations de type impulsion.

Les jacobiennes $\mathbf{F}_\mathcal{R}$ et $\mathbf{F}_\mathbf{u}$ définies en 3.5 et peuvent s'écrire sous la forme :

$$\mathbf{F}_{\mathcal{R}} = \begin{array}{c|cccccc} & \mathbf{p} & \mathbf{q} & \mathbf{v} & \mathbf{ab} & \mathbf{wb} & \mathbf{g} \\ \hline \mathbf{p}^+ & \mathbf{I}_{3\times3} & 0 & \mathbf{I}_{3\times3}\cdot\Delta t & 0 & 0 & 0 \\ \mathbf{q}^+ & 0 & \frac{\partial \mathbf{q}^+}{\partial \mathbf{q}} & 0 & 0 & \frac{\partial \mathbf{q}^+}{\partial \mathbf{wb}} & 0 \\ \mathbf{v}^+ & 0 & \frac{\partial \mathbf{v}^+}{\partial \mathbf{q}} & I_{3\times3} & -R\cdot\Delta t & 0 & I_{3\times3}\cdot\Delta t \\ \mathbf{ab}^+ & 0 & 0 & 0 & \mathbf{I}_{3\times3} & 0 & 0 \\ \mathbf{wb}^+ & 0 & 0 & 0 & 0 & \mathbf{I}_{3\times3} & 0 \\ \mathbf{g}^+ & 0 & 0 & 0 & 0 & 0 & \mathbf{I}_{3\times3} \end{array}$$

$$\mathbf{F}_{\mathbf{u}} = \begin{array}{c|cc} & \mathbf{v}_i & \mathbf{t}_i \\ \hline \mathbf{p}^+ & 0 & 0 \\ \mathbf{q}^+ & 0 & \frac{\partial \mathbf{q}^+}{\partial t_i} \\ \mathbf{v}^+ & \mathbf{I}_{3\times3} & 0 \\ \mathbf{ab}^+ & 0 & 0 \\ \mathbf{wb}^+ & 0 & 0 \\ \mathbf{g}^+ & 0 & 0 \end{array}$$

Remarque 4.3 *La centrale mesure les accélérations biaisées et le facteur d'échelle ne peut être observé que dans le cas d'une estimation parfaite des biais. Ceci est une tâche délicate car ils évoluent au cours du temps et ne peuvent être entièrement observés que si la position du robot est connue très précisément. Dans le cas d'un SLAM EKF, cela demande une réobservation des amers après un certain temps, comme à la fin d'une boucle où l'on peut réobserver les amers initialisés au début. Cela pose deux problèmes : la nécessité d'avoir une trajectoire qui boucle et le fait de maintenir une carte contenant la plupart des amers de notre trajectoire. Les ressources demandées dans le cas d'une grande trajectoire sont trop importantes pour maintenir un SLAM correct.*

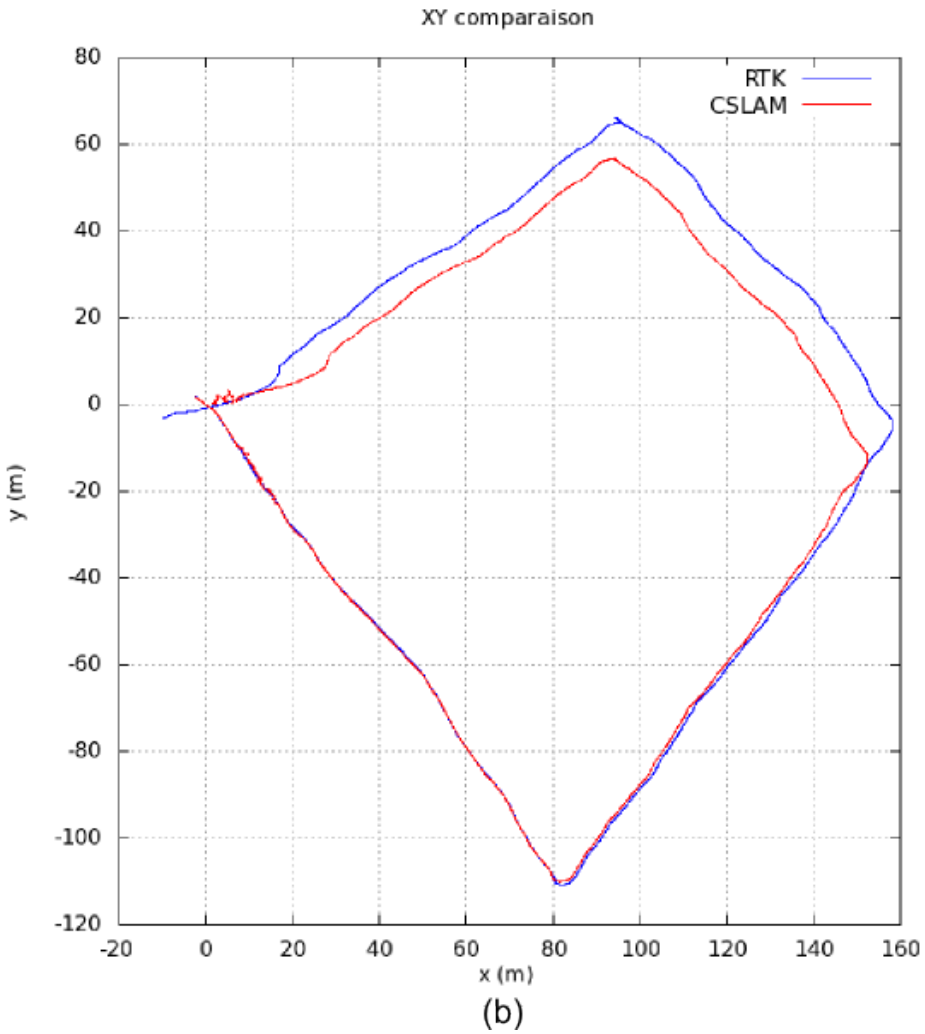
Pour limiter les dérives liées à l'exploitation des données inertielles dans le cas où il n'y a pas de boucle dans la trajectoire, il faut intégrer une mesure absolue de position : ce sera l'apport du GPS, qui donne la position du robot dans le repère global à quelques mètres près.

4.8.4 Résultats SLAM Inertiel

Afin valider de manière expérimentale notre algorithme dans le cas du modèle de mouvement inertiel, nous traitons le jeu de données acquis à *Caylus* par le robot d'exploration *Mana*. Cette séquence a été acquise par la même caméra visible présentée en figure 4.10(b), et le robot était également équipé d'une centrale inertielle bas coût illustrée en figure 4.10(d).

Lors de cette séquence nous choisissons les mêmes paramètres que pour le SLAM monoculaire visible, sauf que 8 corrections corrections sont réalisées à chaque itération, dont quatre par *Recherche Active* et quatre par *One-Point RANSAC*. Ces amers adoptent une paramétrisation AHP et se reparamétrisent en euclidien.

(a)



(b)

FIGURE 4.15 – Résultats du SLAM EKF monoculaire avec couplage serré d'une centrale inertielle. (a) Images de la séquence. (b) Trajectoire en XY obtenue (rouge) avec comparaison avec la vérité terrain (bleu).

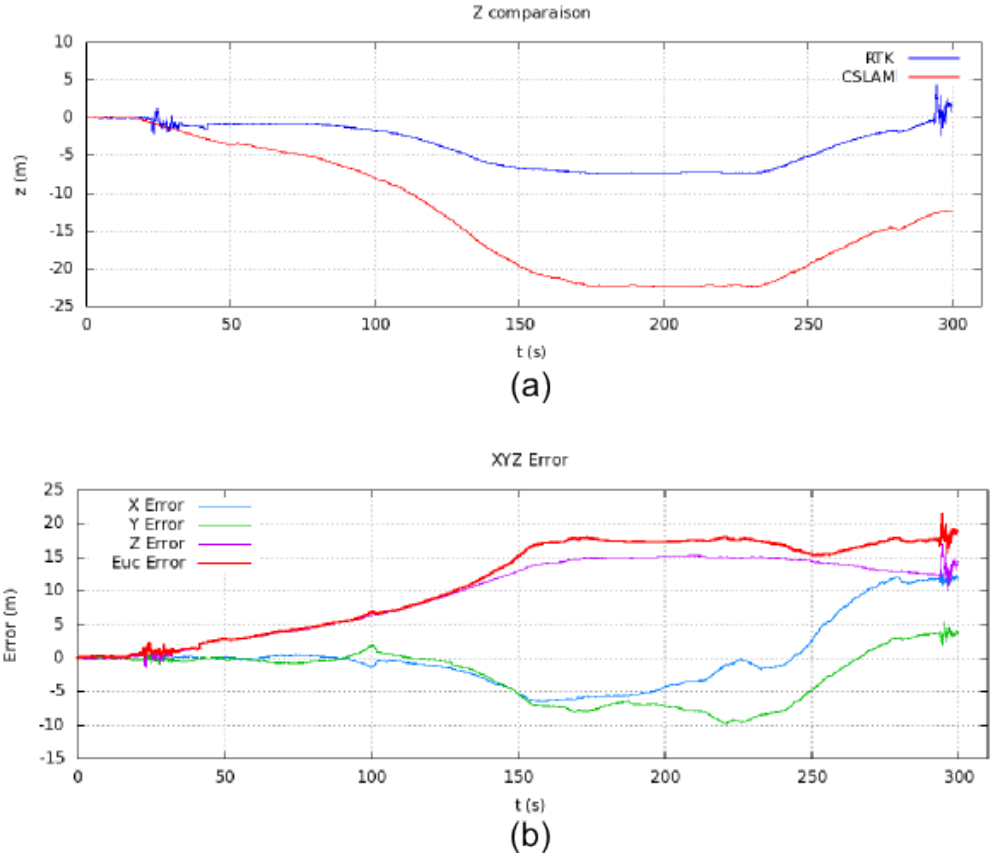


FIGURE 4.16 – Résultats du SLAM EKF monoculaire avec couplage serré d'une centrale inertielle. (a) Trajectoire en Z obtenue (rouge) et comparaison avec la vérité terrain (bleu). (b) Erreurs commises.

Durant la séquence présentée en figure 4.15(a), le robot a réalisé une boucle approximativement de 500 mètres. Sur ces images, les points en vert foncés correspondent amers AHP dans la carte, et ceux en vert clair les amers observés et corrigés. Les amers reparamétrés apparaissent en bleu, foncé s'ils sont dans la carte et clair s'ils ont servi pour la correction du filtre. La figure 4.15(b) compare la trajectoire estimée (en rouge) par le SLAM sur le plan (XY) avec la vérité terrain (en bleu) obtenue par un GPS-RTK, et la figure 4.16(a) sur l'axe Z. Nous pouvons remarquer que cette trajectoire est de bonne qualité, avec une erreur finale inférieure à 4% (4.16(b)).

Ces résultats confirment également l'observabilité incomplète du facteur d'échelle, notamment dans le cas de longue lignes droites, où les biais de la centrale sont observés moins précisément. Seul le cap, non-observable, a dû être ajusté manuellement.

Cette méthode est bien plus robuste aux secousses dues au terrain accidenté générant de fortes accélérations, et bien plus précise que la méthode exploitant le modèle à vitesse constante. De plus, la centrale inertielle permet de réduire considérablement l'ensemble des incertitudes, représentées dans la matrice de covariance $\mathbf{P}$ du filtre. De ce fait, les zones de recherche des amers sont réduites, la corrélation par ZNCC plus précise et plus rapide, et le risque de faux-appariement diminue. Les avantages sont donc multiples dans le cas du système embarqué, où le nombre de corrections nécessaires pour maintenir un filtre cohérent peut être revu à la baisse, et la taille de la carte SLAM peut être diminuée également.

Cependant, il est très délicat de combiner une centrale inertielle avec une caméra infrarouge basse-résolution. En effet, la bonne qualité des images visibles nous permet une bonne observabilité des biais. Dans le cas de l'infrarouge, la faible qualité des appariements rend l'estimation des biais pas assez précise pour obtenir un SLAM correct.

4.8.5 La correction GPS

Si un signal GPS est disponible, un tel capteur ne bornera pas seulement l'erreur de position du mobile ; mais il limitera également l'ensemble des erreurs en comparant la direction de la vitesse locale fournie par le couple caméra-IMU avec la vitesse globale fournie par le GPS.

Dans le cas d'un couplage faible, les capteurs aident à l'étape de correction de l'EKF. De cette manière, le GPS s'intègre simplement dans l'étape de correction du filtre de Kalman. Nous supposons connue la transformation qui permet de passer de la position dans le référentiel WGS84 à la position dans le repère du monde pour le robot. Dans l'application SART par exemple, la position et l'orientation du repère de l'aéroport sont connues dans le référentiel terrestre.

La fonction d'observation définie en 3.6 s'exprime ici simplement en fonction de la position du robot $\mathcal{R}_\mathbf{p}$. Cette équation peut alors être récrite sous la forme :

$$\mathbf{y} = \mathbf{h}(\mathcal{R}_\mathbf{p}) + v$$

avec sa jacobienne définie simplement par la matrice identité :

$$\mathbf{H} = \frac{\partial \mathbf{h}(\mathcal{R}_\mathbf{p})}{\partial X^\mathsf{T}}\Big|_{\hat{X}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

L'innovation, définie en 3.7 est simplement la différence entre la mesure du GPS et la position prédite du robot. Elle peut se récrire selon :

$$\mathbf{z} = \mathbf{y} - \mathbf{h}(\hat{\mathcal{R}}_\mathbf{p})$$

Comme l'observation de la position ne dépend que de la position du robot, le calcul de la covariance de l'innovation $\mathbf{Z}$ définie en 3.8, ne concerne que $\mathbf{P}_{\mathcal{R}_\mathbf{p}\mathcal{R}_\mathbf{p}}$ de la matrice de covariance $\mathbf{P}$ et se simplifie par :

$$\mathbf{Z} = \mathbf{P}_{\mathcal{R}_\mathbf{p}\mathcal{R}_\mathbf{p}} + \mathbf{R}$$

où $\mathbf{R}$ est le bruit associé au capteur GPS. On suppose, à tort, que le bruit de mesure du GPS est blanc et gaussien. De fait, il est très corrélé dans le temps. Par contre, en moyennant plusieurs mesures d'un GPS, et comme celles-ci sont globalement centrées sur la position réelle, la mesure peut être considérée comme blanche et gaussienne.

On calcule ensuite le gain du filtre de Kalman selon 3.9 :

$$\mathbf{K} = \mathbf{P} \cdot \mathbf{H}^\mathsf{T} \cdot \mathbf{Z}^{-1}$$

Finalement, on met à jour le vecteur d'état et la matrice de covariance avec 3.10 et 3.11 :

$$\begin{aligned} \hat{X}^+ &= \hat{X} + \mathbf{K} \cdot \hat{\mathbf{z}} \\ \mathbf{P}^+ &= \mathbf{P} - \mathbf{K} \cdot \mathbf{Z} \cdot \mathbf{K}^\mathsf{T} \end{aligned}$$

4.8.6 Résultats du SLAM monoculaire visible avec fusion Inertiel-GPS

Les mêmes paramètres que pour le cas de l'inertiel ont été utilisés ici : une carte de 300 états, quatre amers sont initialisés à chaque itération. Ces amers adoptent également une paramétrisation AHP et peuvent être reparamétrés en euclidien.

La séquence présentée en figure 4.17(a) est toujours la boucle de 500 mètres réalisée à *Caylus*. Sur ces images, nous affichons les amers observés comme en figure 4.15.

La figure 4.17(b) compare la trajectoire estimée par *CSLAM* (en rouge) et par *RT-SLAM* (en vert) sur le plan (XY) avec la vérité terrain (en bleu) obtenue par un GPS-RTK , et la figure 4.18(a) sur l'axe Z. Les données GPS utilisées pour la correction sont bruitées à 8 m à 99%.

Dans ce cas, le GPS permet de recaler l'ensemble des dérives, et de borner les erreurs comme l'illustre la figure 4.18(b). Globalement, l'erreur reste inférieure à 4 m, sauf vers la fin de la séquence on le terrain devient accidenté et le robot est soumis à de fortes accélérations, ce qui rend les données plus difficilement observables. Le facteur d'échelle devient complètement observable, et le cap du véhicule est extrapolé de ces données.

4.8.7 Résultats du SLAM monoculaire Infrarouge avec fusion Inertiel-GPS

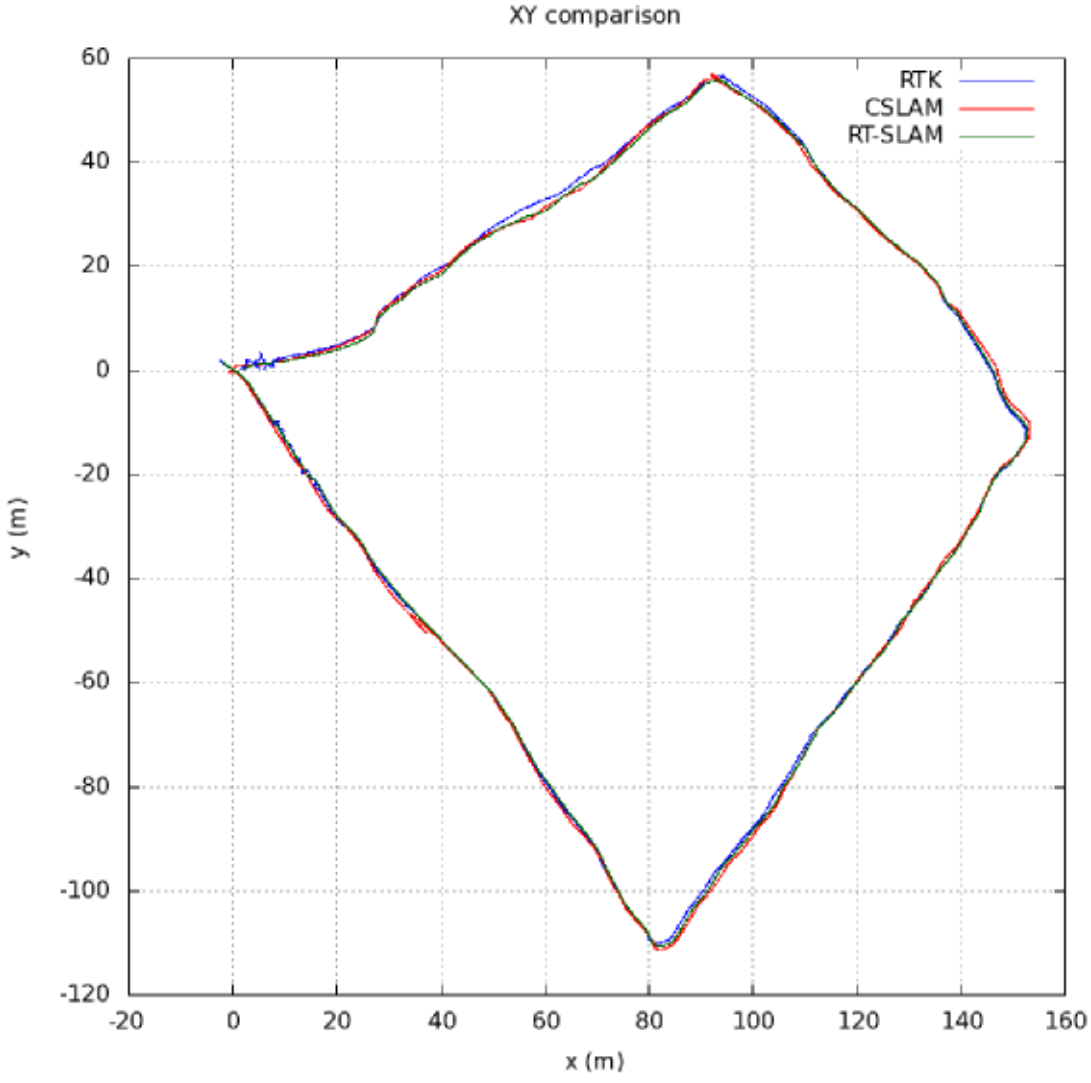
Les mêmes paramètres que pour le cas précédent ont été utilisés ici : une carte de 300 états, quatre amers sont initialisés à chaque itération. Ces amers adoptent également une paramétrisation AHP, mais ne sont pas reparamétrés en euclidien pour limiter la perte de qualité. Pour ces mêmes raisons, nous corrigeons 10 amers par itération.

La figure 4.19(a) présente des images d'une trajectoire d'environ 50 mètres réalisée sur le parking du LAAS de nuit. Sur ces images, la représentation colorimétrique des amers est la même que celles utilisées précédemment.

La figure 4.19(b) compare la trajectoire estimée par le SLAM (en rouge) sur le plan (XY) avec la vérité terrain (en bleu) obtenue par un GPS-RTK, et la figure 4.17(c) sur l'axe Z.

A cause des problématiques introduites par la caméra infrarouge, et soulevées en 4.7.2, nous devons disposer de données GPS de qualité afin de limiter les dérives de notre algorithme. Dans cet exemple, elles sont bruitées à 1,5 m à 99%. Comme illustré en figure 4.20, les divergences sont importantes lors des virages, quand les amers ne sont pas observés suffisamment longtemps pour permettre une convergence de l'algorithme. Dans ces cas, le GPS permet de limiter l'erreur. Cette erreur diminue lors des régimes constants, quand les amers sont observés suffisamment longtemps, et que les corrections GPS permettent un recalage de la position.

(a)



(b)

FIGURE 4.17 – Résultats du SLAM EKF monoculaire avec couplage serré d'une centrale inertielle, et utilisation d'un GPS pour la correction. (a) Images de la séquence. (b) Trajectoire en XY estimée par *CSLAM* en rouge et par *RTSLAM* en vert, avec comparaison avec la vérité terrain (bleu).

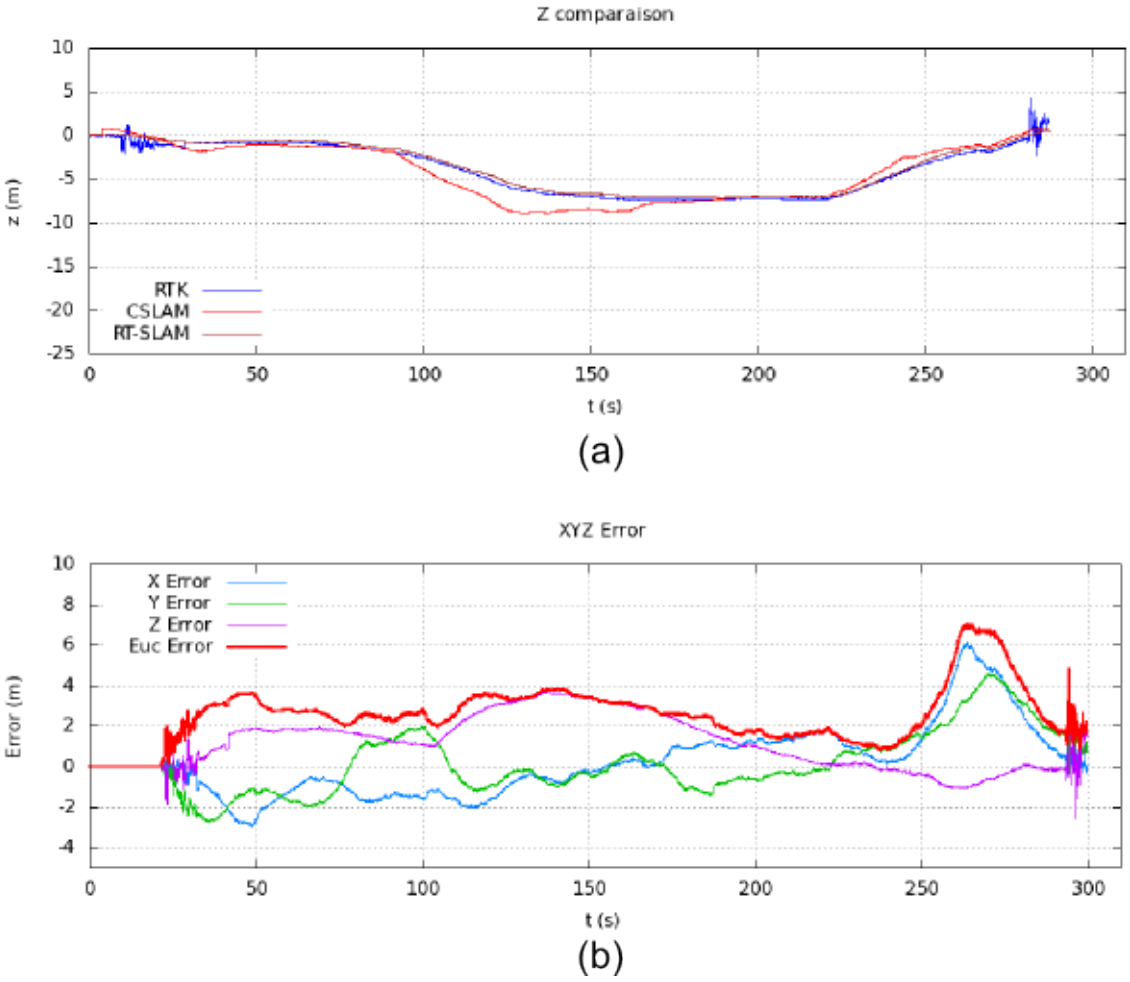
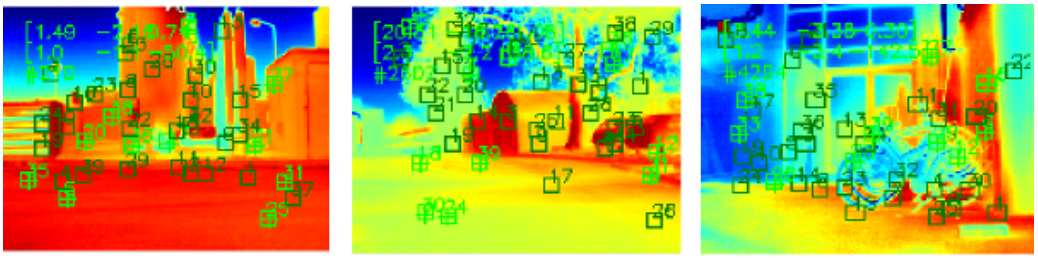
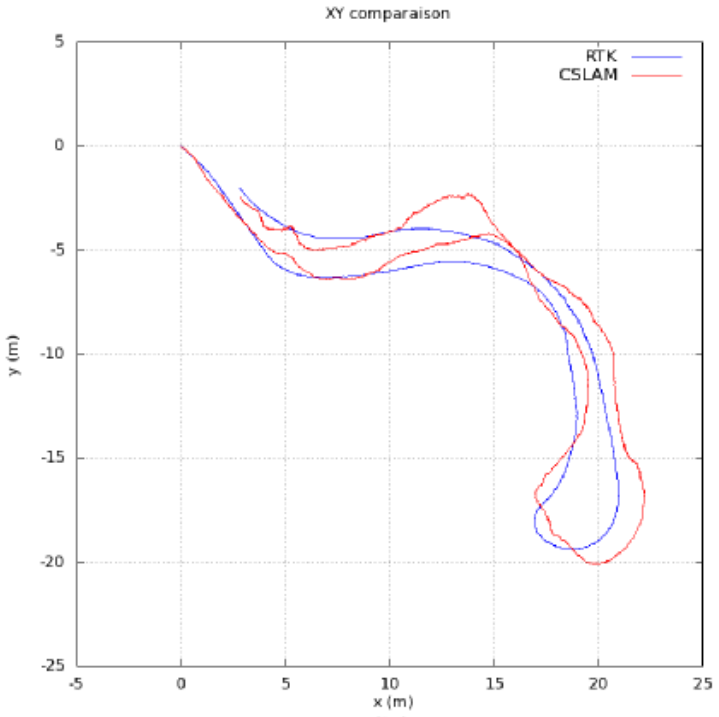


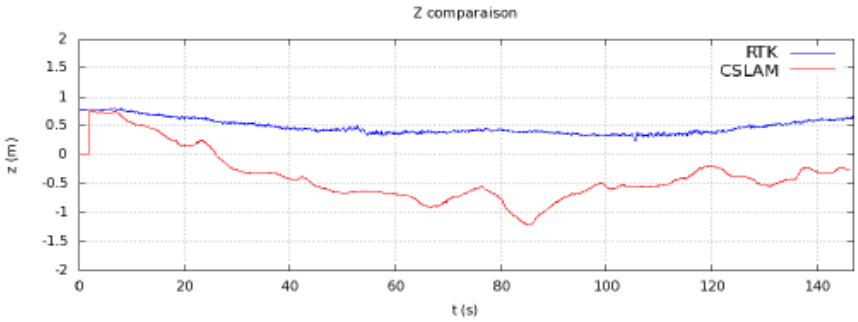
FIGURE 4.18 – Résultats du SLAM EKF monoculaire avec couplage serré d'une centrale inertielle, et utilisation d'un GPS pour la correction. (a) Trajectoire en Z estimée par *CSLAM* en rouge et par *RTSLAM* en marron, avec comparaison avec la vérité terrain (bleu). (b) Erreurs commises.



(a)



(b)



(c)

FIGURE 4.19 – Résultats du SLAM EKF monoculaire infrarouge avec couplage serré d'une centrale inertielle, et utilisation d'un GPS pour la correction. (a) Images de la séquence. (b) Trajectoire en XY obtenue (rouge))et comparaison avec la vérité terrain (bleu). (c) Trajectoire en Z obtenue (rouge) et comparaison avec la vérité terrain (bleu).

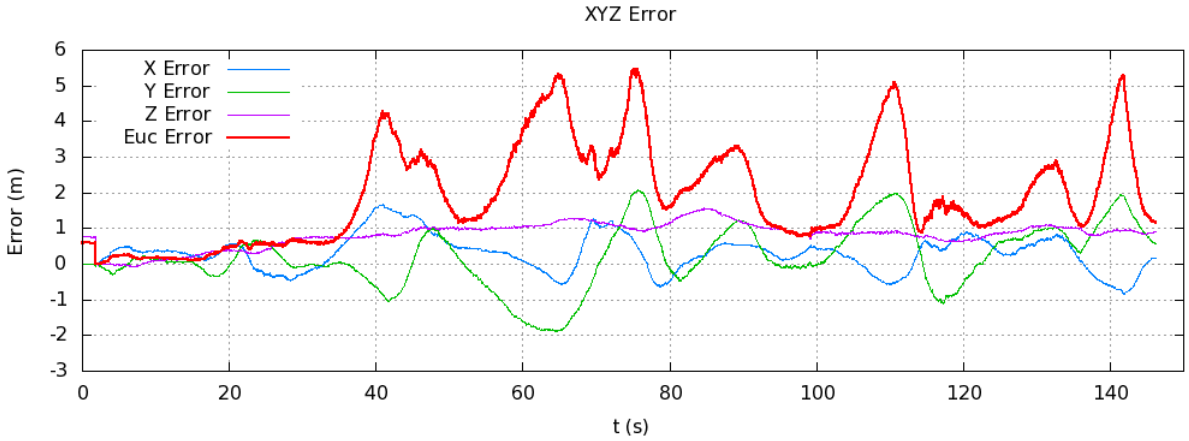


FIGURE 4.20 – Erreurs commises par le SLAM EKF monoculaire infrarouge avec couplage serré d'une centrale inertielle, et utilisation d'un GPS pour la correction.

4.8.8 Validation par le score NEES

Une bonne méthode pour l'évaluation d'un filtre de Kalman consiste en l'évaluation de sa cohérence. Le score NEES, acronyme de *Normalized Estimation Error Squared* a été proposé par Bailey *et al.* [Bailey et al., 2006]. Il est défini par :

$$\epsilon_k = (\mathbf{x}_k - \hat{\mathbf{x}}_k)^\top \mathbf{P}_k^{-1} (\mathbf{x}_k - \hat{\mathbf{x}}_k) \tag{4.13}$$

où $\mathbf{x}_k$ est la valeur de la vérité terrain, $\hat{\mathbf{x}}_k$ l'estimée et $\mathbf{P}_k^{-1}$ l'inverse de la matrice de covariance à l'instant k.

Selon l'hypothèse d'un filtrage d'un système linéaire gaussien cohérent, ϵ_k suit la loi du χ^2 avec le nombre de degrés de liberté donné par la dimension de $\mathbf{x}$. Pour N expériences aléatoires, le score moyen de NEES est calculé par :

$$(\bar{\epsilon})_k = \frac{1}{N} \sum_{i=1}^{N} \epsilon_{ik} \tag{4.14}$$

avec N le nombre de tests de type Monte-Carlo réalisés sur une même séquence. La valeur du NEES peut être considérée comme la distance de Mahalanobis entre la vérité terrain et l'estimée, élevée au carré et prenant en compte le nombre de tests. Cette valeur donne ainsi une idée sur la confiance qu'a l'algorithme de SLAM sur sa localisation. Dans notre cas, nous cherchons à évaluer la cohérence du filtre sur la position du robot, donc nous avons trois degrés de libertés. Les mesures ont été réalisées sur une série de 25 tests aléatoires. La zone de 95% de probabilité est définie par l'intervalle :

$$\chi^2_{3\times 25}(1 - 0.025 \;\; ; \;\; 1 - 0.975) = \{52.94 \;\; ; \;\; 100.84\}$$

La valeur moyenne du NEES doit alors être comprise dans la borne $[2.12 \;\; ; \;\; 4.03]$. En dessous de la borne minimale, le SLAM est sous-confiant : son incertitude sur la localisation est trop grande, au-dessus de la borne maximale, le SLAM est sur-confiant : son incertitude sur sa localisation est trop faible.

Pour réaliser ce test, nous avons utilisé le modèle de mouvement inertiel présenté en section 4.8.3. La correction du filtre est réalisée par les observations issues de la caméra et du GPS,

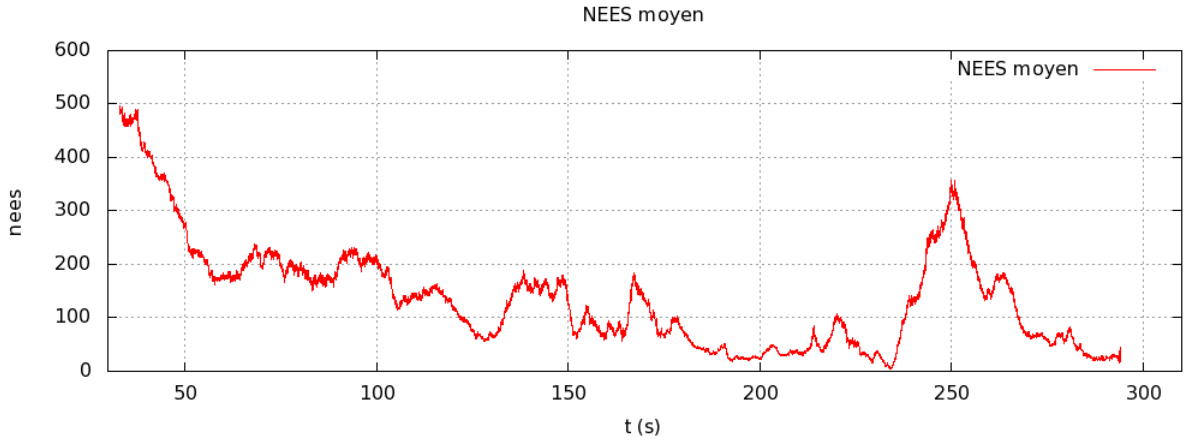


FIGURE 4.21 – Valeurs moyennes du NEES calculées à partir de 25 tests selon la méthode de Monte-Carlo.

dont le modèle de ce dernier est présenté en section 4.8.5. Ces tests ont été effectués sur une série de 15000 images. Cependant, nous ne retenons pas les 30 premières secondes nécessaire au calcul du cap du robot à l'aide des données GPS.

Le résultat présenté en figure 4.21 prouve que le SLAM devient cohérent lors d'un régime constant, quand les données sont de bonne qualité et permettent une meilleure évaluation des biais. Les estimées du vecteur d'état sont plus justes, fournissant des points de linéarisation plus précis au filtre. Lors des régimes transitoires (virages, changements de milieux) et de fortes vibrations ou secousses, l'algorithme a tendance à être optimiste sur la précision de sa localisation, globalement d'un facteur 10.

4.9 Conclusion

Dans ce chapitre, nous avons présenté les différentes étapes d'un SLAM EKF dans un contexte appliqué. Nous avons introduit les différentes optimisations nécessaires à l'obtention d'un SLAM de qualité et répondant aux contraintes de temps réel pour les systèmes embarqués. Un SLAM monoculaire seul avec un modèle à vitesse constante peut être suffisant pour de courtes distances, si les amers peuvent être réobservés lors de fermeture de boucle par exemple. Cependant, pour des applications de localisation en extérieur, ce modèle atteint vite ses limites.

L'intégration d'une centrale inertielle permet un SLAM de meilleure qualité, et le parcours de plusieurs centaines de mètres sans divergence de l'algorithme. Un tel capteur permet de réduire les incertitudes, et facilite ainsi l'étape d'appariement, tout en diminuant le risque de faux-appariements. Néanmoins, l'erreur cumulée en fin de trajectoire peut être importante.

L'utilisation d'un GPS permet alors de recaler l'erreur, grâce à une meilleure observabilité des biais de la centrale et du facteur d'échelle. Le cap du véhicule est également connu grâce à un tel capteur.

Les mesures apportées grâce à une caméra infrarouge peuvent également avoir un grand intérêt, notamment lors de conditions météorologiques dégradées ou pendant la nuit. Cependant, une attention particulière doit être portée sur le choix de celle-ci. L'aspect diffus des images, le manque de dynamique et la faible résolution du capteur peut rendre la problématique de la localisation délicate à traiter. Ces caméras fournissent souvent une donnée codée sur 14 bits et les pré-traitements appliqués sur ces images peuvent engendrer du bruit ou une perte de

qualité. Un capteur infrarouge avec une meilleure dynamique et une résolution plus élevée doit être considérée.

Des séquences vidéos des différents tests sont disponibles à l'adresse :
http ://www.youtube.com/user/thecslam.

De manière à rendre le SLAM temps réel pour les systèmes embarqués, la carte a dû être réduite à quelques centaines d'amers. Dans ce contexte, la problématique du SLAM se rapproche plus d'une problématique d'odométrie optique. L'aspect système embarqué sera traité dans le chapitre 6, qui exposera plus en détails les limitations imposées.

Avant cela, le chapitre suivant propose une implémentation d'un *Map Matching* au sein d'un SLAM EKF, réalisant l'appariement d'indices visuels avec les éléments d'une cartographie qui nous est mise à disposition pour le projet SART.

Chapitre 5

SLAM par pseudo-observations

5.1 Introduction

Avec la multiplication des satellites d'observation de la surface terrestre, la forte demande des *Systèmes d'Information Géographiques* (ou SIG) et la multiplication des appareils nécessitant une cartographie de l'environnement tels que les GPS de nos voitures, il est maintenant facile de se procurer une carte de notre environnement. De nombreux outils et logiciels sont libres d'accès et d'utilisation. Ces bases de données (BDD) cartographiques, souvent aux formats vectoriels ou matriciels, sont déjà grandement utilisées en télédétection où elles servent, par exemple, à corriger les distorsions d'une image satellite. Si nous avons une telle carte à notre disposition, il est alors intéressant de pouvoir l'exploiter dans notre algorithme de localisation. C'est le cas de notre application pour laquelle une base de données aéroportuaire est fournie, comme illustré en figure 5.1.

L'utilisation de telles bases de données dans les travaux de localisation et de navigation restent cependant assez rares pour localiser un mobile dans un environnement, du fait de la précision insuffisante des SIGs, par rapport aux besoins de la localisation. Par ailleurs ce traitement n'est pas simple, puisqu'il nécessite de percevoir l'environnement et d'y détecter des éléments identifiables dans la base de données. Il est plus simple, rapide et efficace d'utiliser une donnée GPS. Cependant, cette donnée peut être inexploitable à cause de multi-réflexions, brouillage hostile... Pour améliorer la localisation GPS et simplifier la localisation vis-à-vis d'une carte SIG, le *map matching* est couramment exploité sur les système de navigation grand public : la position d'un véhicule obtenue par GPS est projetée sur la route la plus proche (avec l'hypothèse qu'une voiture se déplace sur une route), avec possibles erreurs aux abords de carrefours.

Plusieurs travaux proposent une localisation sur la base d'une carte apprise au préalable par SLAM : [Kim and Sukkarieh, 2005] et [Rady et al., 2011] traitent de la localisation d'aéronefs ou de drones, tandis que [Marquez-Gamez and Devy, 2011], [Thomas, 2011] et beaucoup d'autres traitent de la localisation de véhicules ou robots terrestres pour rejouer une trajectoire apprise par des techniques de SLAM ou de SFM. Que cela soit en aérien ou en terrestre, ils réalisent une première navigation d'un milieu inconnu dans de bonnes conditions, qui permet la construction d'une carte de points d'intérêt. Si une information GPS est disponible, les points peuvent être géo-référencés. Lors d'un parcours ultérieur de cette trajectoire de référence dans des conditions dégradées, la carte est alors utilisée et mise à jour pour localiser le mobile précisément.

Plus proche de notre application, pour la localisation en vol d'un drone, Conte *et al.* [Conte and Doherty, 2008] fusionnent dans un filtre de Kalman les données d'une centrale inertielle et d'une approche basée sur l'extraction et l'appariement de segments. Les contours sont extraits

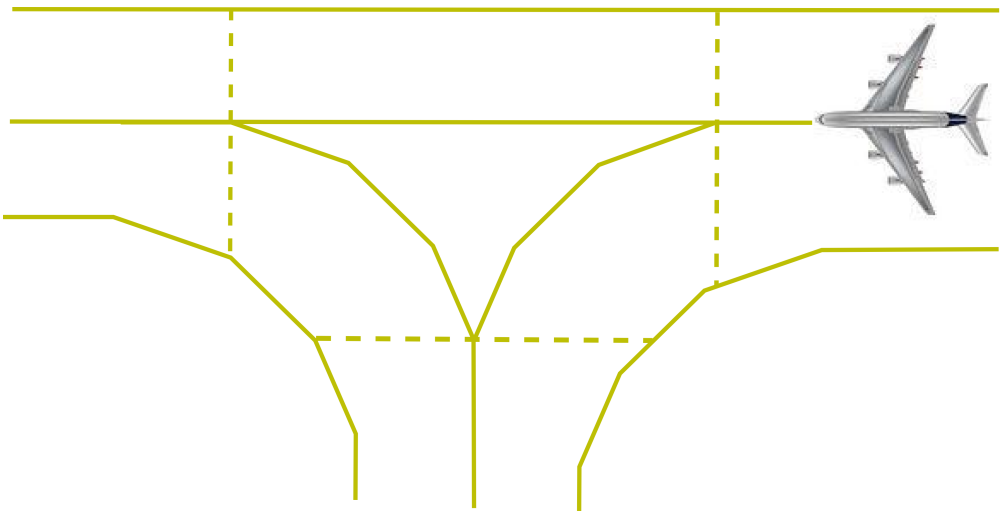


FIGURE 5.1 – Illustration de quelques éléments présents dans la base de données cartographique pour notre application de localisation d'un aéronef dans un aéroport. Les données sont représentées sous forme de polylignes géo-référencées dans un repère propre à cette base de données.

à la fois de l'image issue de la caméra embarquée, et de la cartographie "Google-Earth" de la zone de vol. Une première fusion de données est réalisée au sein d'un premier filtre de Kalman, comprenant les capteurs de vol (baromètres, centrale inertielle) et l'odométrie visuelle provenant d'un suivi de points par la méthode KLT entre deux images successives. La dérive obtenue par ce type de méthode est limitée par l'appariement entre les segments extraits de l'image et les lignes contenues dans une BDD cartographique pré-traitée et embarquée. Lorsqu'un appariement est réalisé, et défini comme valide, l'observation réalisée grâce à la BDD est intégrée dans le filtre de Kalman lors de la correction. Afin d'éviter les sauts et provoquer une divergence de l'algorithme, cette observation est intégrée progressivement. Bien qu'ils permettent d'éviter la divergence de la localisation, de tels appariements restent rares et seulement deux ont été réalisés durant un vol de deux minutes sur quelques dizaines de mètres.

En 2011, Rady *et al.* [Rady et al., 2011] proposent une solution technique pour la localisation absolue basée sur l'apprentissage de caractéristiques topographiques particulières et géo-localisées lors d'un premier vol, et la mise en correspondance de ces caractéristiques lors des vols suivants. L'année suivante, Lim *et al.* [Lim et al., 2012]] utilisent une approche similaire pour la localisation de leur drone. Une reconstruction de la scène par *SFM* est réalisée hors-ligne, reconstruction qui est ensuite utilisée pour la localisation du drone grâce à un appariement 2D-3D.

Notre base de données cartographique concerne l'aéroport de Toulouse-Blagnac. Elle a été obtenue par le *Service de l'Information Aéronautique*. Cette base 2D suppose que l'aéroport est plat : elle recense les principaux éléments au sol, tels que les axes, les bords de pistes (ou *runways*) et des voies de circulation (*taxiways*), les voies de service, ou les points d'arrêt. Y figurent également les parkings et les bâtiments. Toutefois, malgré leur grand intérêt, des objets comme les balises lumineuses ne sont pas signalées. Elles sont pourtant facilement repérables dans les images visibles ou infrarouges, de jour comme de nuit. Dans cette base de données, les différents éléments répertoriés se présentent sous forme de polylignes : l'objet se subdivise en une suite de segments dont les points sont géo-référencés dans un repère propre à la base

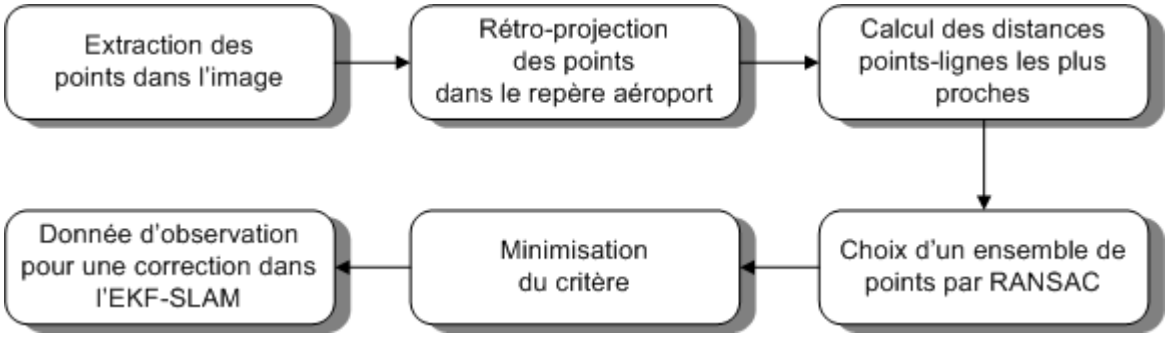


FIGURE 5.2 – Algorithme proposé pour l'appariement points-lignes et l'intégration du résultat dans le SLAM EKF.

de données. Notons que ce repère a été adopté comme le repère de l'aéroport dans lequel nous exprimons toutes les localisations obtenues par GPS ; c'est donc aussi le repère dans lequel est exprimée la carte SLAM.

Dans ce chapitre nous présentons une méthode pour l'utilisation de telles bases de données et comment les appariements entre les points issus d'une caméra et les éléments d'une cartographie peuvent être exploités dans un SLAM EKF.

5.2 Méthode proposée

Pour exploiter, dans notre méthode SLAM, une base de données contenant la cartographie de notre environnement, nous proposons l'algorithme illustré en figure 5.2. Celui-ci peut se décomposer en cinq parties distinctes, qui seront ensuite détaillées dans chaque sous-section :

- La première étape consiste en l'extraction des points d'intérêt dans l'image. Ces points doivent être extraits sur les contours de l'image correspondant aux projections des marquages et des bords de piste de manière à pouvoir les corréler avec les informations de notre BDD.
- Une fois les points extraits de l'image, une étape de rétro-projection dans le repère de la base de données est nécessaire. Chaque point image est rétro-projeté sur le plan du sol, donné par la BDD. Celle-ci est 2D, et notre système évolue sur un aéroport supposé plan : nous faisons donc la supposition d'un sol plan horizontal. La profondeur du point-sol sur le rayon optique correspondant est alors connue si le système est calibré correctement.
- Ensuite, nous calculons les distances orthogonales points-sol/lignes et ne retenons que la distance la plus courte pour chaque point.
- Il se peut que des points-sol n'appartiennent pas à des lignes de la BDD, comme les balises, les obstacles ou les artefacts dans l'image. Ces points peuvent entraîner de faux appariements et doivent donc être filtrés par un algorithme de RANSAC jusqu'à former un ensemble d'appariements cohérents au sens du résidu obtenu après localisation de l'aéronef.
- Une fois un ensemble cohérent de couples point-sol/ligne obtenu, nous calculons les paramètres (θ, t_x, t_y) qui correspondent à la rotation et à la translation 2D à réaliser pour minimiser, au sens des moindres carrés, l'ensemble des distances orthogonales.

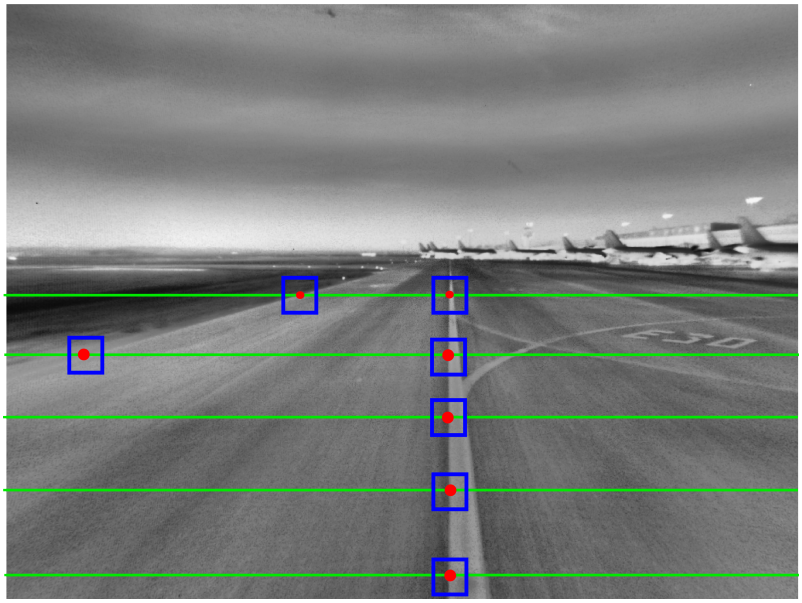


FIGURE 5.3 – Exemple de l'extraction des points par détection de forts gradients dans l'image.

Nous obtenons en sortie de cet algorithme le paramètre de rotation θ et les paramètres de translation t_x et t_y qui produisent ainsi une matrice homogène RT. L'estimation de RT est raffinée par optimisation locale, pour obtenir la corrélation la plus précise sur le plan du sol, entre les points-sol calculés à partir des points extraits sur les contours image, et les lignes de la BDD. Cette nouvelle position ainsi calculée est intégrée à l'étape de correction du Filtre de Kalman Étendu, en tant qu'observation de la position. Nous traitons cette observation de manière similaire à une observation GPS.

Pour résumer : les points image sont des observations directement perçues. Les points 3D, ici sur le sol, en sont déduits grâce à la connaissance de la pose de la caméra et à la supposition d'un sol plan. Les appariements points-sol/lignes permettent de calculer une localisation de l'aéronef dans l'aéroport, information que nous appelons pseudo-observation, dans la mesure où elle est incorporée dans notre filtre comme une donnée observée.

5.3 Détection des points

Pour la détection et le suivi des points, considérons l'approche proposée par Chapuis [Chapuis et al., 2002]. Cette approche, appliquée à la détection de routes à bord d'un véhicule, se décompose en deux méthodes :

- La détection des bords de route dans l'image, reposant sur un Filtre de Kalman mis à jour par les observations des marquages dans les images. Ces observations sont faites par la détection de forts gradients le long de lignes horizontales dans l'image, gradients qui correspondent à des bords de route ou des marquages.
- La reconstruction 3D du profil de la route, réalisée grâce à la connaissance des paramètres intrinsèques et extrinsèques du modèle. Les points sont rétro-projetés sur le plan du sol, puis, par la minimisation au sens des moindres carrés de deux polynômes du second et troisième ordre, le profil de la route est reconstruit.

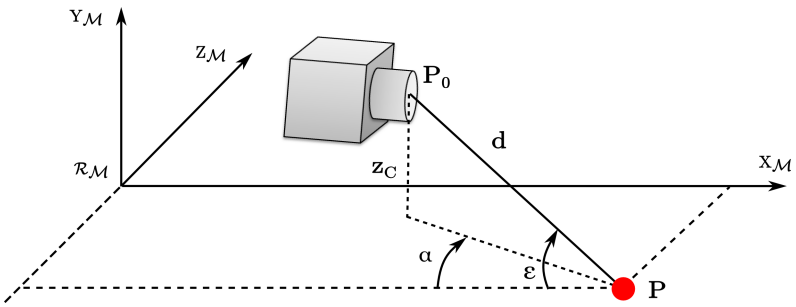


FIGURE 5.4 – Si nous supposons que le point appartient au plan du sol, alors il est facile de connaître ses composantes euclidiennes dans le repère Aéroport, étant également le repère monde $\mathcal{R}_{\mathcal{M}}$ dans notre application.

La méthode que nous proposons pour la détection de marquages reste similaire à celle décrite par Chapuis [Chapuis et al., 2002] et illustré en figure 5.3 : nous cherchons à détecter les forts gradients le long de lignes horizontales dans l'image (mais nous ne générons pas de représentations pour les lignes au sol). Ces gradients correspondent à un changement de milieu dans l'image, tel que le passage de l'herbe à la piste ou de la piste à un marquage. Ces points étant considérés au sol, ils peuvent être facilement rétro-projetés dans la carte SLAM, comme décrit dans la sous-section suivante. Ainsi, ils peuvent directement adopter une paramétrisation euclidienne.

Bien que cette méthode soit intéressante, elle pose quelques inconvénients dans notre application. A cause de la topographie complexe d'un aéroport (multiples lignes qui se croisent à l'approche des parkings, inscriptions des noms des voies de circulation au sol), plusieurs gradients peuvent être présents dans une zone de recherche. Des erreurs lors du calcul du gain du filtre peuvent apparaître si les points extraits dans l'image sont associés aux mauvaises lignes sur le plan du sol.

Une autre difficulté survient lors de la prise d'un virage par l'aéronef. Afin de garder le train principal bien centré sur la piste, le pilote est obligé de survirer : la cabine de pilotage va se retrouver à l'extérieur de la piste, au-dessus de l'herbe. Dans cette configuration, nous avons une perte d'observabilité sur les lignes au sol : notons que ce problème du survirage va se poser quelque soit l'approche choisie pour extraire les bords de piste ou les marquages.

De plus, la méthode que nous proposons n'intègre pas les observations des lignes dans le filtre de Kalman déjà présent pour le SLAM présenté en chapitre 4. Une option aurait été de rajouter les lignes au sol, représentées par des quadriques mises à jour par filtrage de Kalman, dans la carte SLAM, puis d'exploiter la cartographie en appariant les lignes de la carte SLAM avec celles de la carte a priori. Nous avons rejeté cette méthode pour deux raisons :

- La carte SLAM serait devenue hétérogéne, avec des points et des lignes, donc plus complexe à mettre à jour.
- La recherche des appariements entre lignes seraient aussi plus complexe que l'approche adoptée consistant à apparier des points-sol à des lignes de la carte a priori.

5.4 Rétro-projection des points

Une fois les points extraits de l'image, nous devons les rétro-projeter dans le repère de la base de données. Pour cela, nous adoptons la paramétrisation IDP du point présentée en 4.3 :

$$\mathcal{P}_{\mathbb{IDP}} = \mathbf{p} = \begin{bmatrix} x_0 & y_0 & z_0 & \epsilon & \alpha & \rho \end{bmatrix}^{\mathsf{T}} \in \mathbb{R}^6$$

La paramétrisation IDP a un sens lorsque la profondeur de l'amer n'est pas observable. Or, en faisant la supposition que le point se projette sur le plan du sol tel qu'illustré en figure 5.4, la distance d est connue :

$$d = \frac{1}{\rho} = \frac{z_0}{\sin(\epsilon)} \tag{5.1}$$

L'amer peut alors être directement exprimé en euclidien par la fonction de reparamétrisation également présentée en 4.3 :

$$\mathbf{p} = \mathbf{p}_0 + \mathbf{v}^*(\epsilon, \alpha)/\rho \tag{5.2}$$

où $\mathbf{v}^*(\epsilon, \alpha)$ est le vecteur unitaire défini par :

$$\mathbf{v}^*(\epsilon, \alpha) = \begin{bmatrix} \cos(\epsilon)\cos(\alpha) & \cos(\epsilon)\sin(\alpha) & 0 \end{bmatrix}^{\mathsf{T}}$$

En plus de l'utilisation de fonctions déjà développés pour le SLAM monoculaire, comme l'initialisation d'amers à partir de points d'intérêt dans l'image, ou les fonctions de reparamétrisation, l'intérêt de passer par une paramétrisation IDP est la gestion facilitée du passage entre le repère caméra $\mathcal{R}_\mathcal{C}$ et le repère "Aéroport", qui est, dans notre cas, le repère monde $\mathcal{R}_\mathcal{M}$.

5.5 Calcul des distances

L'ensemble des points extraits de la manière décrite en section 5.3 doit être corrélé à la base de données fournie. Celle-ci contient l'ensemble des éléments au sol tels que les bords et milieux de pistes, les intersections, les barres d'arrêt... Nous avons donc à réaliser un appariement points-lignes entre les points issus de l'image et la base de données aéroportuaire, comme présenté en figure 5.5.

L'idée principale est de calculer pour chaque point extrait la distance à chacune des lignes. Pour chaque point, nous ne conservons que la distance la plus courte. Si l'ensemble cartographié est vaste et qu'il recense de nombreux éléments, tel qu'un aéroport, il est alors essentiel de ne s'intéresser qu'au voisinage de la position estimée de l'aéronef. On subdivise alors notre base de données en différentes régions, et on ne calcule les distances points-lignes qu'avec les lignes rattachées aux régions avoisinantes de notre aéronef.

Pour une région donnée, chaque polyligne se subdivise en ses segments qui la composent. L'équation paramétrique de la droite passant par $A = (x_A, y_A)$ et $B = (x_B, y_B)$ d'un segment est donnée par $ax + by + c = 0$, avec :

$$\begin{aligned} a &= y_B - y_A \\ b &= x_A - x_B \\ c &= x_B \cdot y_A - x_A \cdot y_B \end{aligned} \tag{5.3}$$

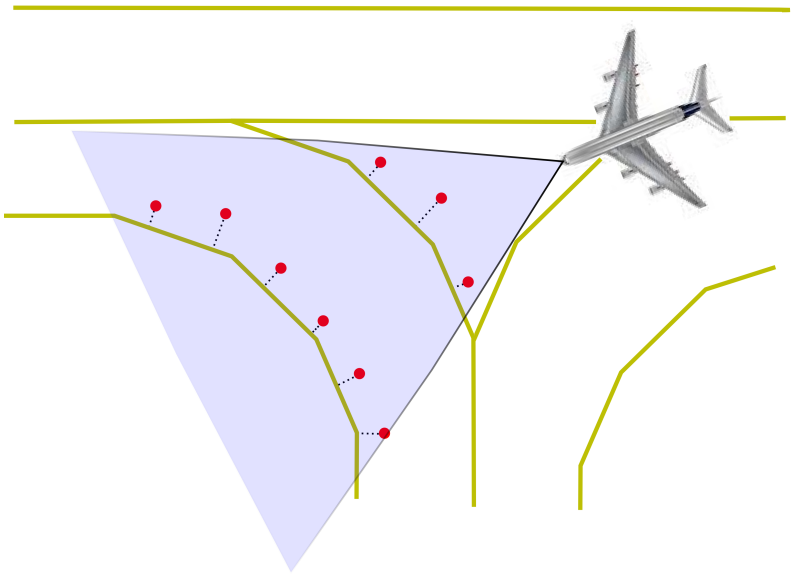


FIGURE 5.5 – Exemple d'appariements point-lignes.

Soit $P_i = (x_{P_i}, y_{P_i})$ un point à apparier à une ligne. La distance orthogonale d_{P_i} de ce point à la droite est donnée par :

$$d_{P_i} = \frac{|a \cdot x_{P_i} + b \cdot y_{P_i} + c|}{\sqrt{a^2 + b^2}} \tag{5.4}$$

Pour chaque point, nous calculons les distances à chaque droite dans une région. Nous ne retenons que les droites qui ont engendré les distances les plus courtes. A chaque couple point-ligne, le projeté orthogonal H_i d'un point P_i à une droite a pour coordonnées :

$$x_{H_i} = \frac{b^2 \cdot x_{P_i} - a \cdot b \cdot y_{P_i} - a \cdot c}{a^2 + b^2} \tag{5.5}$$

$$y_{H_i} = \frac{-a \cdot b \cdot x_{P_i} + a^2 \cdot y_{P_i} - b \cdot c}{a^2 + b^2} \tag{5.6}$$

Nous obtenons donc un ensemble $\mathcal{E}$ de couples points-projetés orthogonaux, et le problème de l'appariement point-ligne devient ainsi un problème d'appariement point-point qui peut être résolu par une méthode telle que l'*Iterative Closest Point* (ICP), simplifié, puisqu'ici, nous sommes en 2D : nous allons estimer une transformation 2D à partir des deux ensembles de points appariés sur le plan du sol.

5.6 Choix d'un ensemble de points cohérents par RANSAC

Dans cet ensemble $\mathcal{E}$, certaines distances peuvent être hors-normes, dues à l'extraction de points sur des lignes perçues dans les images, qui ne correspondent pas à des lignes de la cartographie. Un premier seuillage est réalisé pour éliminer ces points incohérents de l'ensemble. Nous obtenons un second ensemble $\mathcal{E}_F$ filtré des points incohérents (ou *outliers*) auquel nous appliquons un algorithme de RANSAC (pour RANdom SAmple Consensus), de manière à ne retenir qu'un sous-ensemble de points cohérents, ou *inliers*.

Celui-ci peut se décomposer en quatre étapes :

– Un sous-ensemble $\mathcal{E}_i$ de points est choisi parmi l'ensemble $\mathcal{E}_F$ et le modèle RT est estimé avec ce sous-ensemble.
– Ce modèle est appliqué aux autres données, et si un couple correspond bien à ce modèle, il est considéré comme un inlier candidat et est rajouté au sous-ensemble $\mathcal{E}_i$.
– Si suffisamment de points ont été rajoutés à $\mathcal{E}_i$, alors un nouveau modèle est calculé avec ce sous-ensemble. Dans le cas contraire, si peu de points ont été rajoutés à $\mathcal{E}_i$, cela signifie que l'ensemble des couples de base était incohérent avec le reste des couples.
– Une erreur d'appariement est obtenue avec le sous-ensemble $\mathcal{E}_i$

Cette procédure est réitérée i fois, et le sous-ensemble de couples produisant l'erreur la plus faible est retenu.

5.7 Minimisation du critère

Nous cherchons les paramètres θ, t_x, t_y à appliquer à notre ensemble de points pour qu'ils soient appariés aux lignes. Nous avons donc à résoudre un système de la forme :

$$\begin{bmatrix} H_1 \\ \vdots \\ H_n \end{bmatrix} = \mathsf{RT} \cdot \begin{bmatrix} P_1 \\ \vdots \\ P_n \end{bmatrix} \equiv \begin{bmatrix} x_{H_1} & \cdots & y_{H_n} \\ y_{H_1} & \cdots & y_{H_n} \\ 1 & \cdots & 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & t_x \\ \sin(\theta) & \cos(\theta) & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{P_1} & \cdots & x_{P_n} \\ y_{P_1} & \cdots & y_{P_n} \\ 1 & \cdots & 1 \end{bmatrix} \tag{5.7}$$

avec (x_P, y_P) les points-sol, et (x_H, y_H) leurs projections sur les lignes les plus proches de la carte a priori, calculées par les équations 5.5 et 5.6.

A la solution, la relation qui impose la superposition des points H_i et P_i doit être vérifiée :

$$H_i = (\mathbf{R}_\theta \cdot P_i) + \mathbf{T}_{xy} \Leftrightarrow (\mathbf{R}_\theta \cdot P_i) + \mathbf{T}_{xy} - H_i = 0 \tag{5.8}$$

où $\mathbf{R}_\theta$ est la matrice de rotation élémentaire autour de l'axe z, et $\mathbf{T}_{xy}$ la translation sur les axes x et y :

$$\mathbf{R}_\theta = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \qquad \mathbf{T}_{xy} = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

La distance évoquée est simplement égale à la différence entre les nouvelles coordonnées d'un point et son projeté orthogonal :

$$\mathcal{D}(H_i, P_i) = (R_\theta \cdot P_i) + T_{xy} - H_i \tag{5.9}$$

Soit $\mathcal{F}(\theta, t_x, t_y, H_i, P_i)$ la fonctionnelle qui à tout couple (H_i, P_i) associe la valeur définie en 5.9. L'objectif de la méthode de localisation est donc de trouver le vecteur de paramètres (θ, t_x, t_y) qui minimisent le critère suivant :

$$\mathcal{E} = \sum_{i=1}^{n} [\mathcal{D}(H_i, P_i)]^2 = \sum_{i=1}^{n} \mathcal{F}(\theta, t_x, t_y)^2 \tag{5.10}$$

Ce critère n'est pas linéaire. Sa minimisation nécessite donc la mise en oeuvre d'un processus itératif du type Newton-Raphson ou Levenberg-Marquard afin d'estimer le vecteur d'état recherché.

Nous avons réalisé une implémentation dédiée de cette méthode d'optimisation plutôt que d'exploiter une fonction d'une bibliothèque standard, cela pour pouvoir l'intégrer plus facilement dans le système embarqué. Nous rappelons ci-après les principales étapes de cette méthode, qui exploite une linéarisation de la fonctionnelle à minimiser.

Soit A le vecteur d'état correspondant à une rotation et une translation quelconque dans le plan et A_k le vecteur d'état relatif à la rotation et translation déterminé à l'étape k du processus itératif :

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} \theta \\ t_x \\ t_y \end{pmatrix} \qquad \begin{pmatrix} a_{1k} \\ a_{2k} \\ a_{3k} \end{pmatrix} = \begin{pmatrix} \theta_k \\ t_{xk} \\ t_{yk} \end{pmatrix}$$

Le développement limité à l'ordre 1 de la fonctionnelle $\mathcal{F}$ au voisinage de A_k est :

$$\begin{aligned} \mathcal{F}(A, H_i, P_i) &\simeq \mathcal{F}(A, H_i, P_i) + \frac{\partial \mathcal{F}(A, H_i, P_i)}{\partial A}\Big|_{A=A_k} (A - A_k) \\ &\simeq \mathcal{F}(A, H_i, P_i) + \sum_{l=1}^{3} \frac{\partial \mathcal{F}(A, H_i, P_i)}{\partial_{a_l}}\Big|_{A=A_k} (a_l - a_{lk}) \end{aligned}$$

En supposant que A soit la solution recherchée, alors $\mathcal{F}(A, H_i, P_i) = 0$ et nous pouvons écrire l'approximation suivante :

$$-\mathcal{F}(A, H_i, P_i) \simeq \mathcal{F}(A, H_i, P_i) + \frac{\partial \mathcal{F}(A, H_i, P_i)}{\partial A}\Big|_{A=A_k} (A - A_k)$$

En posant $\Delta_{a_l} = a_l - a_l k$ pour $n \in [1, 3]$ et en considérant l'ensemble des n appariements, nous obtenons le système linéaire suivant :

$$\begin{cases} -\mathcal{F}(A_k, H_1, P_1) = \frac{\partial \mathcal{F}(A,H_1,P_1)}{\partial \theta}\Big|_{A=A_k} \Delta_\theta + \frac{\partial \mathcal{F}(A,H_1,P_1)}{\partial t_x}\Big|_{A=A_k} \Delta_{t_x} + \frac{\partial \mathcal{F}(A,H_1,P_1)}{\partial t_y}\Big|_{A=A_k} \Delta_{t_y} \\ \quad \vdots \qquad\qquad \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ -\mathcal{F}(A_k, H_n, P_n) = \frac{\partial \mathcal{F}(A,H_n,P_n)}{\partial \theta}\Big|_{A=A_k} \Delta_\theta + \frac{\partial \mathcal{F}(A,H_n,P_n)}{\partial t_x}\Big|_{A=A_k} \Delta_{t_x} + \frac{\partial \mathcal{F}(A,H_n,P_n)}{\partial t_y}\Big|_{A=A_k} \Delta_{t_y} \end{cases}$$

Ce dernier ayant trois inconnues, il est donc nécessaire de dériver les abscisses et les ordonnées, soit :

$$\begin{cases} -\mathcal{F}(A_k, x_{H_1}, x_{P_1}) = \frac{\partial \mathcal{F}(A,x_{H_1},x_{P_1})}{\partial \theta}\Big|_{A=A_k} \Delta_\theta + \cdots + \frac{\partial \mathcal{F}(A,x_{H_1},x_{P_1})}{\partial t_y}\Big|_{A=A_k} \Delta_{t_y} \\ \quad \vdots \qquad\qquad \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ -\mathcal{F}(A_k, y_{H_n}, y_{P_n}) = \frac{\partial \mathcal{F}(A,y_{H_n},,y_{P_n})}{\partial \theta}\Big|_{A=A_k} \Delta_\theta + \cdots + \frac{\partial \mathcal{F}(A,y_{H_n},y_{P_n})}{\partial t_y}\Big|_{A=A_k} \Delta_{t_y} \end{cases} \tag{5.11}$$

Nous obtenons ainsi deux équations par appariement, donc deux points sont nécessaires et suffisants pour trouver une solution à ce problème. Le système peut être noté matriciellement :

$$[F] = [J][\Delta A]$$

Sa solution au sens des moindres carrés, pour ($n \geqslant 2$), est donnée par :

$$[\Delta A] = ([J]^{\mathsf{T}}[J])^{(-1)}[J]^{\mathsf{T}}[F]$$

Cela revient à calculer le vecteur correctif ΔA qui minimise la somme des distances orthogonales. Connaissant le vecteur correctif ΔA, il est possible de calculer un nouveau vecteur d'état A_{k+1} par la relation suivante :

$$A_{k+1} = A_k + \Delta A$$

Si les conditions initiales A_0, données par les rétro-projections des points, sont relativement proches des lignes à apparier, quelques itérations suffisent pour réaliser la convergence.

Le calcul des dérivées partielles se fait par l'application de la *Règle de la Chaîne*. Soit $\mathcal{F}(A, H_i, P_i)$ la fonctionnelle égale à :

$$\mathcal{F}(A, H_i, P_i) = (\mathbf{R}_\theta \cdot P_i) + \mathbf{T}_{xy} - H_i \tag{5.12}$$

Alors la dérivée partielle de $\mathcal{F}(A, H_i, P_i)$ par rapport à $A = (\theta \quad t_x \quad t_y)^{\mathsf{T}}$ est égale à :

$$\frac{\partial \mathcal{F}(A, H_i, P_i)}{\partial A} = \frac{\partial \mathcal{F}(A, H_i, P_i)}{\partial [H_i \;\; P_i]} \cdot \frac{\partial [H_i \;\; P_i]}{\partial A} \tag{5.13}$$

où :

$$\frac{\partial \mathcal{F}(A, H_i, P_i)}{\partial [H_i \;\; P_i]} = \left[\frac{\partial \mathcal{F}(A,H_i,P_i)}{\partial H_i} \qquad \frac{\partial \mathcal{F}(A,H_i,P_i)}{\partial P_i} \right]$$

et :

$$\frac{\partial [H_i \;\; P_i]}{\partial A} = \begin{bmatrix} \frac{\partial H_i}{\partial A} \\ \frac{\partial P_i}{\partial A} \end{bmatrix}$$

L'expression 5.13 peut ainsi s'écrire :

$$\begin{aligned} \frac{\partial \mathcal{F}(A, H_i, P_i)}{\partial A} &= \left[\frac{\partial \mathcal{F}(A,H_i,P_i)}{\partial H_i} \qquad \frac{\partial \mathcal{F}(A,H_i,P_i)}{\partial P_i} \right] \cdot \begin{bmatrix} \frac{\partial H_i}{\partial A} \\ \frac{\partial P_i}{\partial A} \end{bmatrix} \\ &= \frac{\partial \mathcal{F}(A, H_i, P_i)}{\partial H_i} \cdot \frac{\partial H_i}{\partial A} + \frac{\partial \mathcal{F}(A, H_i, P_i)}{\partial P_i} \cdot \frac{\partial P_i}{\partial A} \end{aligned} \tag{5.14}$$

Or, nous ne connaissons pas $\frac{\partial H_i}{\partial A}$. Néanmoins, nous pouvons utiliser les dérivées $\frac{\partial H_i}{\partial P_i}$ que nous obtenons lors du calcul de la projection orthogonale du point, et $\frac{\partial P_i}{\partial A}$ obtenue lors du calcul des nouvelles coordonnées à l'itération $k+1$. L'expression 5.13 peut se développer finalement en :

$$\frac{\partial \mathcal{F}(A, H_i, P_i)}{\partial A} = \frac{\partial \mathcal{F}(A, H_i, P_i)}{\partial H_i} \cdot \frac{\partial H_i}{\partial P_i} \cdot \frac{\partial P_i}{\partial A} + \frac{\partial \mathcal{F}(A, H_i, P_i)}{\partial P_i} \cdot \frac{\partial P_i}{\partial A} \tag{5.15}$$

Afin de simplifier ces expressions, à chaque étape du processus itératif, on modifie la position du modèle dans son repère $\mathcal{R}_m$. Soit A_k le vecteur d'état trouvé à l'étape k. On calcule alors les points $P_i|_k = \mathbf{R}_{\theta_k} \cdot P_i + \mathbf{T}_{x_k y_k}$ qui deviennent les nouveaux points caractéristiques du modèle.

Pour chaque étape, ceci revient simplement à choisir un repère relatif par rapport à la dernière position calculée. Ainsi, le calcul de toutes les dérivées partielles se font pour un vecteur de paramètres nuls. Le critère à minimiser à l'étape $k+1$ est donc :

$$\mathcal{E}_{k+1} = \sum_{i=1}^{n} [\mathcal{F}(A_k = 0, H_i, P_i|_k) + \sum_{l=1}^{3} \frac{\partial \mathcal{F}(A, H_i, P_i|k)}{\partial a_l}|_{A=0} \Delta_{a_l}]^2 \tag{5.16}$$

Les dérivées partielles sont alors triviales :

$$\frac{\partial \mathcal{F}(A, H_i, P_i)}{\partial H_i} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \frac{\partial H_i}{\partial P_i} = \begin{bmatrix} \frac{b^2}{a^2+b^2} & \frac{-a \cdot b}{a^2+b^2} \\ \frac{-a \cdot b}{a^2+b^2} & \frac{a^2}{a^2+b^2} \end{bmatrix}$$

$$\frac{\partial P_i}{\partial A} = \begin{bmatrix} -y_{P_i} & 1 & 0 \\ x_{P_i} & 0 & 1 \end{bmatrix} \qquad \frac{\partial \mathcal{F}(A, H_i, P_i)}{\partial P_i} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$

L'estimation du vecteur d'état relatif à l'étape $(k+1)$ est obtenue par :

$$\begin{cases} \mathbf{R}_{\theta_{k+1}} & = & \mathbf{R}_{\Delta\theta} \cdot \mathbf{R}_{\theta_k} \\ \mathbf{T}_{x_{k+1} y_{k+1}} & = & \mathbf{R}_{\Delta\theta} \cdot \mathbf{T}_{x_k y_k} + \mathbf{T}_{\Delta x \Delta y} \end{cases}$$

5.8 Intégration au SLAM

Nous disposons maintenant d'une observation de position et d'orientation en deux dimensions que nous pouvons intégrer au filtre de Kalman étendu réalisant le SLAM. Cette observation va être traitée de la même manière qu'une donnée de localisation GPS décrite en 4.8.5 et va donc être utilisée lors de l'étape de correction du filtre. La différence avec le GPS est que nous avons en plus une donnée d'orientation à traiter. Soit la fonction d'observation :

$$\mathbf{y} = \mathbf{h}(A) + v \tag{5.17}$$

la jacobienne de cette fonction d'observation par rapport à la pose du robot est :

$$\mathbf{H} = \frac{\partial \mathbf{h}(A)}{\partial X^{\mathsf{T}}}\Big|_{\hat{X}} = \begin{array}{c|cc} & \mathbf{p} & \mathbf{q} \\ \hline \theta & 0 & \frac{\partial \alpha}{\partial q} \\ t_x & I_{2\times 2} & 0 \\ t_y & & \end{array}$$

L'innovation, définie en 3.7, est la différence entre le résultat issu de la minimisation du critère et la pose prédite du robot. Elle peut être récrite selon :

$$\mathbf{z} = \mathbf{y} - \mathbf{h}(\hat{\mathcal{R}}_{\mathbf{pq}})$$

Le calcul de la covariance de l'innovation $\mathbf{Z}$, définie en 3.8, ne concerne que $\mathbf{P}_{\mathcal{R}_{\mathbf{pq}}\mathcal{R}_{\mathbf{pq}}}$ de la matrice de covariance $\mathbf{P}$ et simplifie son calcul :

$$\mathbf{Z} = \mathbf{H} \cdot \mathbf{P}_{\mathcal{R}_{\mathbf{pq}}\mathcal{R}_{\mathbf{pq}}} \cdot \mathbf{H}^{\mathsf{T}} + \mathbf{R}$$

où $\mathbf{R}$ est un bruit de mesure, qui peut être considéré dans notre cas comme la précision de la cartographie locale. On calcule ensuite le gain du filtre de Kalman selon 3.9 :

$$\mathbf{K} = \mathbf{P} \cdot \mathbf{H}^{\mathsf{T}} \cdot \mathbf{Z}^{-1}$$

Enfin, la mise à jour du vecteur d'état et de la matrice de covariance est réalisée avec 3.10 et 3.11 :

$$\begin{aligned} \hat{X}^{+} &= \hat{X} + \mathbf{K} \cdot \hat{\mathbf{z}} \\ \mathbf{P}^{+} &= \mathbf{P} - \mathbf{K} \cdot \mathbf{Z} \cdot \mathbf{K}^{\mathsf{T}} \end{aligned}$$

5.9 Résultats

Pour valider de manière expérimentale notre algorithme, nous traitons le jeu de données sur la séquence parking du LAAS, déjà utilisé et présenté en 4.7.1. Nous n'avons pas pu, pour diverses raisons, exploiter la base de données de l'aéroport de Toulouse-Blagnac qui était disponible, principalement parce qu'il nous a été impossible d'avoir des résultats de SLAM correct avec les jeux d'images acquis sur cet aéroport. Cela du fait de la qualité des images, que ce soit en infrarouge ou en visible :

- Sur les images infrarouges, des problèmes avec l'algorithme mis en oeuvre pour réduire la quantification de la radiance (de 14 bits à 8 bits).
- Sur les images visibles, des problèmes liés à la caméra elle-même, ainsi que l'utilisation d'un obturateur de type "Rolling Shutter".

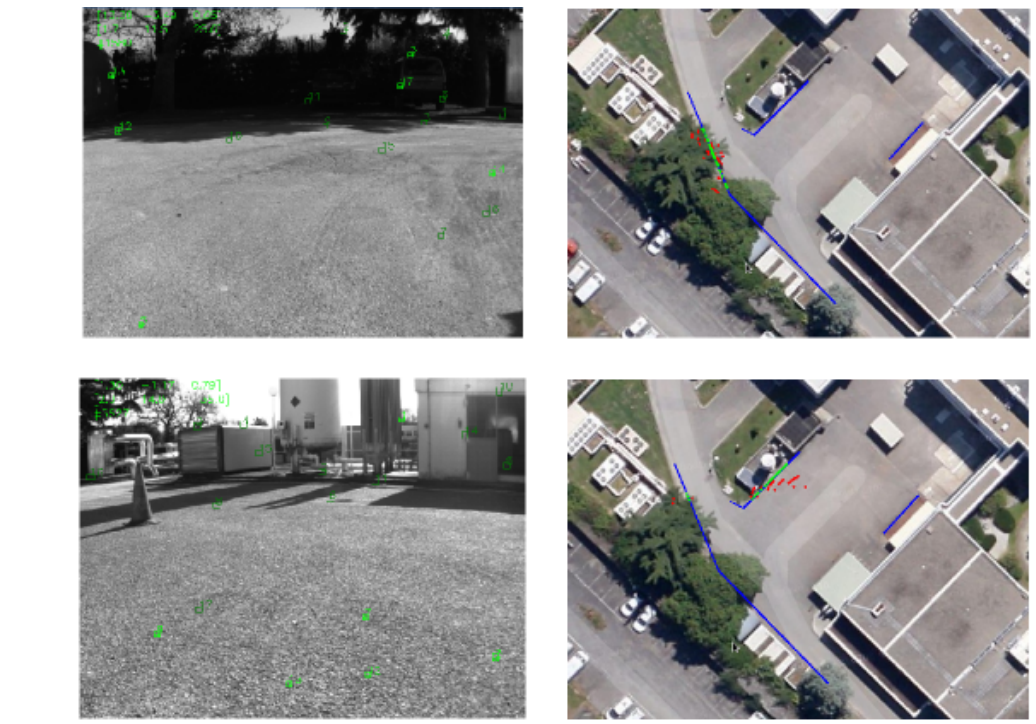
Nous avons donc construit une BDD virtuelle à partir des vues "Google Maps" du LAAS. Nous utilisons un modèle de mouvement inertiel et la correction du filtre est réalisée par les observations visuelles, les données GPS et les appariements avec la base de données. La carte SLAM peut contenir jusqu'à 300 états, et 6 corrections sont faites à chaque itération par *Recherche Active*. Trois amers sont initialisés à chaque image dans la limite de la place disponible dans la carte et d'amers détectés. Ces amers adoptent une paramétrisation AHP.

Durant la séquence présentée en figure 5.6(a), le robot a réalisé une boucle d'environ 40 mètres. Les images de gauche sont les résultats du SLAM monoculaire, tandis que celles de droite présentent les appariements avec la base de données. Dans ces dernières, les segments présents dans la BDD apparaissent en bleu, les points extraits de l'image et projetés au sol sont rouges, et les projetés de ces points sur le segment le plus proche sont présentés en vert sur le segment.

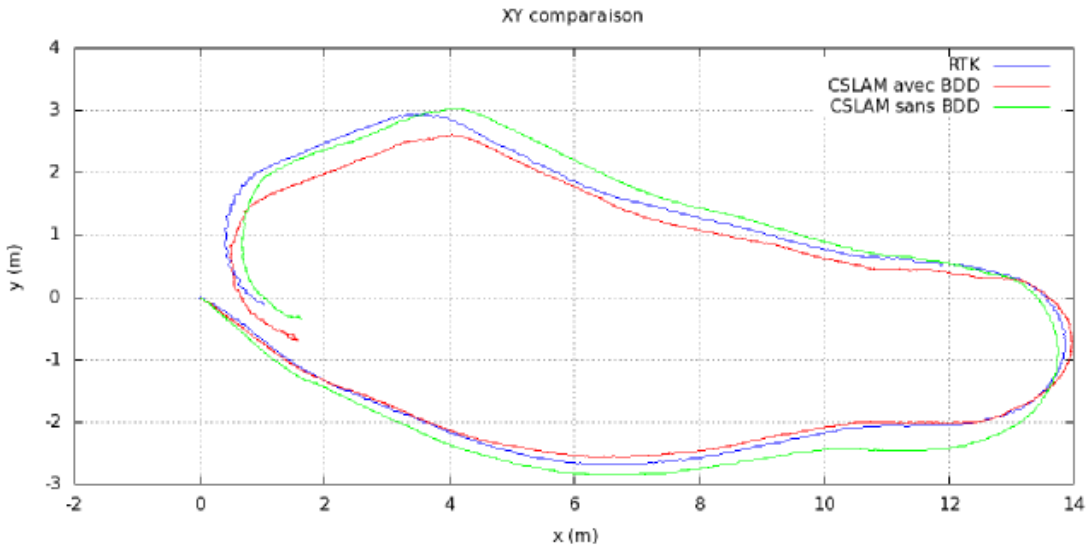
Cette approche est très sensible à la hauteur et aux angles de tangage et de roulis estimés. Pour faciliter les appariements, nous n'avons pas considéré la hauteur de notre robot fournie par le SLAM. Cette valeur provient directement du GPS-RTK. Pour les autres paramètres, nous utilisons les données d'un GPS bruité à 1.5m à 99%.

La figure 5.6(b) fait apparaître que l'utilisation d'une BDD comme observation supplémentaire (en rouge) n'apporte par réellement d'informations supplémentaire par rapport à un SLAM-Inertiel classique avec utilisation d'un GPS (vert) qui est ici assez précis. Nous devons poursuivre ces évaluations afin d'évaluer l'apport des appariements entre primitives observés et éléments de la BDD, cela pour :

- Estimer si l'utilisation de la BDD peut faire diminuer le nombre d'amers à corriger.
- Pallier une perte complète des données GPS, ou une moindre qualité de la localisation GPS.



(a)



(b)

FIGURE 5.6 – Résultats du SLAM EKF monoculaire avec couplage serré d'une centrale inertielle, utilisation d'un GPS et d'une base de donné pour la correction. (a) Captures d'écran durant la séquence. A gauche, le SLAM monoculaire, et à droite l'appariement avec la base de données. (b) Trajectoire en XY obtenue (rouge), sans l'utilisation d'une base de données (vert) et vérité terrain (bleu).

5.10 Conclusion

Nous avons présenté dans ce chapitre une approche pour l'utilisation d'observations issues d'une base de données pour l'étape de correction du filtre de Kalman dans un SLAM monoculaire. Cependant, cette approche est très sensible aux éléments à apparier à la BDD, et une attention particulière doit être portée sur la qualité de leur position. Dans le cas du capteur monoculaire, l'information de profondeur n'est pas directement accessible, et la supposition que les éléments extraits appartiennent au sol peut engendrer des erreurs. Cette supposition est fortement dépendante des différents paramètres de l'algorithme, mais plus particulièrement de la hauteur et des valeurs de tangage et de roulis.

Cependant, cette approche peut être intéressante pour limiter certaines dérives. Par exemple, dans le cas d'un signal GPS indisponible ou de mauvaise qualité sur une application aéroportée, une simple détection de contours peut faire apparaître des éléments directement corrélables à une BDD (routes, cours d'eau...), comme réalisé par Conte *et al.* [Conte and Doherty, 2008] .

Néanmoins, l'ensemble des opérations proposées dans ce chapitre dégradent les performances fréquentielles du SLAM à hauteur d'une dizaine d'images par seconde sur un ordinateur de bureau, et rendent cette approche incompatible avec une utilisation temps réel pour un système embarqué, que nous présentons dans le chapitre suivant.

Chapitre 6

SLAM embarqué et co-design

6.1 Introduction

6.1.1 Problématique

Les applications du SLAM sont aujourd'hui nombreuses et il existe tout autant de méthodes pour résoudre le problème de la localisation. Cependant, les techniques employées sont généralement coûteuses en terme de puissance et de ressources de calcul. Dans la plupart des cas, les données sont enregistrées depuis un robot mobile (ou tout autre véhicule) doté seulement des capteurs et de capacités d'acquisition et de stockage des données sensorielles, puis post-traitées sur des ordinateurs modernes non embarqués. Ceux-ci offrent généralement des capacités de calcul importantes et peuvent intégrer éventuellement des cartes graphiques ou autres accélérateurs de calcul. Pour des démonstrations dans nos laboratoires, ces mêmes robots peuvent être équipés d'un ordinateur portable pouvant disposer d'importantes capacités de calcul. La majorité de la communauté s'intéresse à l'amélioration des algorithmes existants ou au développement de nouvelles méthodes plus robustes et plus précises à l'aide de machines de plus en plus puissantes. Dès lors, intégrer un algorithme de SLAM performant sur un système embarqué pourvu de ressources limitées reste un défi.

L'optimisation de chaque fonction est fondamentale pour en diminuer l'ensemble des coûts, aussi bien en puissance de calcul, qu'en place mémoire, en énergie consommée... Les recherches actuelles dans ce domaine ont pour but de réduire chaque coût engendré sans pour autant dégrader les performances en précision ou en temps de traitement. Développer un SLAM sur des plate-formes embarquables permet ainsi d'augmenter le champ d'utilisation de ces systèmes de localisation.

Les architectures matérielles ne dérogent pas à la version commune de la *Loi de Moore*. Ainsi, les puissances, mémoires et autres composants de l'électronique embarquée, ont vu leur capacités évoluer rapidement ces derniers temps, jusqu'à permettre l'intégration d'un SLAM. Cependant, la configuration d'une telle carte n'est pas la même que celle d'un ordinateur et pour tirer le meilleur parti d'une architecture matérielle, il est nécessaire de penser au couple *Logiciel/Matériel*. Ce type d'optimisation dépend particulièrement de la carte utilisée et de ses composants : processeurs multi-coeurs, FPGA (*Field-Programmable Gate Array*), GPU (*Graphics Processing Unit*), processeurs vectoriels... L'application cible va dicter la configuration à utiliser. Dans le cadre du SLAM, cela implique de définir les fonctions qui nécessitent un processeur et celles qui peuvent être programmées *en dur* sur les autres composants de la carte. Cette réflexion de *co-design* va permettre de tirer le meilleur parti de l'architecture choisie.

6.1.2 Etat de l'art du SLAM embarqué

L'utilisation du SLAM sur des systèmes embarqués est très récente. Ce n'est que depuis quelques années que l'électronique embarquée dispose de suffisamment de ressources pour intégrer un algorithme de SLAM. En 2007, Abrate *et al.* [Abrate et al., 2007] ont réalisé les premiers travaux en la matière. Sur un robot *Khepera* embarquant de simples capteurs infrarouges, ils implémentent un SLAM EKF sur un microprocesseur cadencé à 25 Mhz. La faible puissance de calcul et la mauvaise qualité des détecteurs infrarouges (normalement utilisés comme capteurs de proximité et non comme capteur de profondeur) ont conduit à une optimisation importante de l'algorithme et à l'utilisation d'un modèle de ligne *Hessian* comme primitive, défini par $x\cos\alpha + y\sin\alpha - r = 0$ avec $r \geq 0$. Sur chaque groupe de points détecté est appliqué un appariement non linéaire à ce modèle de ligne afin d'obtenir un nombre l de lignes droites. Leurs résultats expérimentaux montrent l'importance de la caractérisation des capteurs. De plus, le choix des primitives et leur mise en correspondance sont des éléments fondamentaux, surtout dans le cas de capteurs de mauvaise qualité. Deux ans plus tard, Yap *et al.* [Yap and Shelton, 2009] utilisent également des capteurs bas coût, ici les télémètres ultrasons d'un robot *P3-DX* pour une application de SLAM par filtre particulaire, mais qu'ils n'embarquent pas sur le robot. Pour compenser la qualité des données médiocres, ils font l'hypothèse de murs orthogonaux, c'est-à-dire qu'ils exploitent une connaissance contextuelle pour rajouter des contraintes sur le modèle.

En 2008, Gifford *et al.* [Gifford et al., 2008] proposent un SLAM multi-robot. Chaque robot est équipé de six capteurs infrarouges, d'un gyromètre, d'odomètres et d'une carte *Gumstix*. Sur cette carte, cadencée à 600 Mhz, est embarqué un algorithme DP-SLAM (filtre particulaire). Malgré des résultats expérimentaux de bonne qualité obtenus lors du *ICRA Space Robotics Challenge* en 2008, ils relèvent deux inconvénients majeurs : d'abord, la précision du SLAM est fortement liée à l'utilisation de capteurs de qualité ; un compromis est donc à trouver entre l'exactitude des résultats et le coût des capteurs. Ensuite, les faibles ressources disponibles rendent l'algorithme incompatible avec le temps réel, nécessitant 3 secondes pour la mise à jour de 15 particules et 10 secondes pour 25. Magnenat *et al.* [Magnenat et al., 2010] proposent en 2010 un algorithme SLAM basé sur un co-design entre un calculateur et une optimisation de l'algorithme Fast-SLAM 2.0 (association d'un filtre particulaire pour estimer la position du robot et de plusieurs filtres de Kalman pour estimer les cartes associées à ces positions). Leur plate-forme expérimentale est équipé d'un ARM cadencé à 533 MHz et ils utilisent un télémètre infrarouge rotatif de leur conception. Ils notent la nécessité de la bonne paramétrisation de l'algorithme afin de diminuer le temps de calcul et le nombre d'amers dans la carte. Meier *et al.* [Meier et al., 2012] ont développé récemment un drône quadrirotor embarquant un système de stéréovision, une centrale inertielle et un puissant processeur double-coeur *Core2duo* cadencé à 1,86 GHz. Cette puissance de calcul permet la détection d'obstacles grâce au système de stéréovision, et d'embarquer un SLAM basé sur la fusion des données acquises par les caméras et par la centrale. Cette fusion offre une odométrie visuelle de qualité, dont le résultat est intégré dans un filtre de Kalman pour l'estimation de la position du drône. Ils soulignent l'importance de la synchronisation entre les différents capteurs.

Ces différentes études mettent en valeur l'importance de la qualité des informations issues des capteurs pour éviter d'engendrer de faux-appariements ainsi qu'une paramétrisation efficace des différentes méthodes pour obtenir un traitement en temps réel. Des optimisations algorithmiques sont également proposées telles que des paramétrisations d'amer moins onéreuses ou la limitation de ceux-ci dans la carte SLAM. Schröter *et al.* [Schröter et al., 2007] optimisent, par exemple, l'espace mémoire d'un algorithme FastSLAM pour une application de cartographie. Cependant, les algorithmes profitant de l'architecture sur lesquels ils sont

implémentés sont rares. De manière à garantir de bons résultats, que ce soit en terme de précision, de temps de calcul, de consommation... il est nécessaire de concevoir son algorithme en fonction de la carte cible, par exemple en exploitant son système multi-coeur, son processeur vectoriel ou ses composants matériels si celle-ci en est pourvue. Nous nous orientons ainsi vers des *architectures dédiées* et un *co-design* des algorithmes.

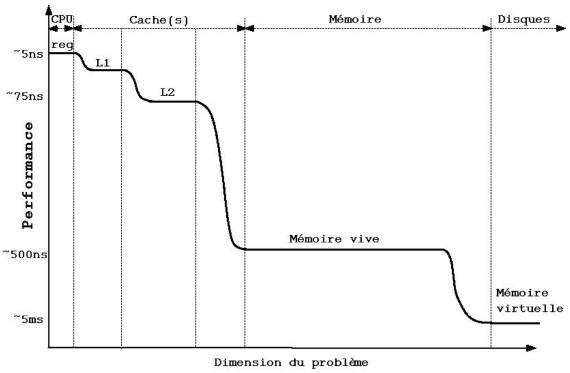
Peu d'implémentations de SLAM sur des architectures embarquées ont été réalisées. De plus, elles ne sont généralement pas en adéquation avec l'architecture cible. Le premier portage d'un SLAM tenant compte de la carte sur laquelle il est implanté a été proposé par la société *Neato* pour le développement de leur aspirateur autonome. Afin de minimiser les coûts de production, ils développent leur propre télémètre laser [Konolige et al., 2008]. Konolige *et al.* n'informent que très peu sur l'architecture employée dans cet article. Cependant, nous pouvons remarquer les efforts réalisés sur la parallélisation des tâches de traitement, de certains calculs et du transfert de données en mémoire par l'utilisation de DSP (Digital Signal Processor). Ils sont également parmi les premiers à noter la puissance consommée afin de garantir une bonne autonomie à leur produit. De plus, ils mettent en avant l'importance du bon calibrage des capteurs.

Parmi les travaux les plus avancés en matière de SLAM visuel embarqué, Vincke *et al.* [Vincke et al., 2012] proposent une adéquation entre algorithme et architecture de manière à garantir un SLAM EKF temps réel sur un système embarqué multiprocesseur bas coût. Ils utilisent une caméra 320×240 et des odomètres. L'architecture comporte un processeur ARM cadencé à 500 MHz (sur lequel est implémenté le filtre de Kalman) disposant d'un co-processeur SIMD, un DSP cadencé à 430 MHz (pour la parallélisation du traitement d'images) et un GPU. En ne portant l'algorithme que sur le processeur ARM et en maintenant 25 amers dans la carte SLAM, le temps de traitement est de 80,8 ms. Lors de l'optimisation des fonctions et du découpage de l'algorithme en blocs fonctionnels, leur gain de temps de traitement est de 78% rendant ainsi leur application temps réel. Ils relèvent que la précision de la localisation dépend du nombre d'amers dans la carte et de la qualité de leurs observations, qui influent directement sur la fréquence de traitement. Une stratégie de suppression des amers permettant l'initialisation de nouveaux a été mise en place. Ils concluent qu'une bonne adéquation entre algorithme et architecture permet de rendre une application SLAM temps réel.

L'utilisation d'architectures massivement parallèles, tels que les FPGA, pour le traitement de données issues de capteurs peut s'avérer très intéressante, surtout quand il s'agit de capteurs délivrant plusieurs millions de données par seconde tel qu'une caméra. Dans sa thèse qui a été effectuée aussi sur le projet SART en parallèle avec nos propres travaux, Botero [Botero G., 2012] propose une architecture matérielle basée sur des FPGAs pour effectuer l'ensemble des traitements relatifs à l'image. La correction géométrique de l'image, la recherche des points de Harris et la mise en correspondance sont faits à la fréquence pixel. Ainsi, le traitement d'image n'engendre plus aucun coût pour un éventuel processeur embarquant un algorithme de SLAM.

Vincke [Vincke, 2012] compare les approches directes et celles qui ont nécessité une adaptation de l'algorithme en fonction de l'architecture cible. Son étude va du capteur jusqu'au calculateur. Il remarque que les approches classiques consistant en l'implémentation d'un code de SLAM existant sur une architecture embarquable, même puissante, peuvent présenter des défauts sévères, en termes de précision, de temps réel, de consommation... L'implémentation optimisée d'un algorithme de SLAM en adéquation avec une architecture embarquée est indispensable.

Ce chapitre s'intéresse à l'algorithme de SLAM EKF présenté dans [Gonzalez et al., 2011], algorithme destiné à être intégré dans des systèmes embarqués, donc conçu pour être facilement modulable en fonction de l'architecture cible. Nous développons ensuite l'architecture cible présentée en [Botero et al., 2012]. Enfin, nous évaluons notre co-design pour le SLAM visuel



(a)

Instruction	FPU Clock Cycles Required
Add, Subtract	6
Multiply	5
Divide	17
Square Root	17
Convert	6
Fused Multiply-Add/Sub	10
Move, Abs, Neg, *etc.*	2
Compare	3

(b)

FIGURE 6.1 – A gauche, le temps d'accès aux différentes mémoires du système, comprenant un registre, deux niveaux de cache (L1 et L2), une Mémoire vive et une Mémoire virtuelle (disque dur...). A droite, le nombre de cycles d'horloge nécessaires à un FPU pour l'opération mathématique concernée sur des nombres simple précision.

embarqué et présentons les résultats obtenus sur le système sélectionné.

6.2 Architecture logicielle

6.2.1 Contraintes

Le développement d'une application logicielle telle que le SLAM, doit satisfaire de nombreuses contraintes, parfois contradictoires. Selon la cible visée, un simple ordinateur puissant ou un système embarqué, la méthode de développement pourra être très différente. De manière générale, un programme doit, en plus de fonctionner correctement, être facilement lisible et compréhensible. Il est donc rare qu'une fonction soit plus longue que quelques dizaines de lignes. Pourtant, Il y a quelques décennies, les recommandations étaient inverses : ne pas réaliser de sous-routines séparées si la procédure n'est appelée qu'une seule fois. Les raisons de ce changement viennent principalement du fait que les projets sont de plus en plus conséquents et l'objectif est mis sur les coûts de développements. Avec des ordinateurs de plus en plus puissants, les optimisations de code ont laissé place à une lisibilité plus aisée. Cette priorité du développement logiciel aux dépens de l'efficacité d'un programme dessert l'utilisateur qui doit investir dans des machines de plus en plus puissantes afin de répondre aux besoins en ressources demandées. Quelquefois, il est nécessaire de trouver un compromis entre les principes avancés de développement d'un logiciel et sa rapidité d'exécution. Ce compromis a d'autant plus de poids dans le développement d'un algorithme dédié aux systèmes embarqués. Un code mal conçu, peu ou pas optimisé, peut engendrer des latences importantes et rendre le produit instable ou inutilisable. De plus, pour optimiser un code de calcul, il faut tenir compte de l'architecture sur laquelle ce dernier sera exécuté.

Des précautions particulières doivent être prises au niveau du stockage et de l'utilisation d'une donnée. En effet, une donnée stockée dans le registre du CPU sera accessible plus rapidement qu'une même donnée stockée en mémoire vive ou sur disque (figure 6.1). Le temps de traitement peut être plus ou moins long en fonction du type de donnée (entier, simple précision, double précision, signée ou non...). Ce temps peut également évoluer en fonction de l'opération réalisée. La figure 6.1 (b) indique le nombre de cycles d'horloge nécessaires au *Floating Point*

Unit (FPU), unité de calcul arithmétique d'un processeur, pour effectuer une opération sur un ou plusieurs nombres en simple précision. Si des opérations très consommatrices de temps et de ressources de calcul sont réalisées régulièrement, tel que des divisions, calculs de racine ou fonctions trigonométriques, il peut être intéressant d'utiliser des *Look-Up Tables* (LUT), c'est-à-dire de stocker des valeurs pré-calculées dans des tableaux exploités en ligne, éventuellement avec un mécanisme d'interpolation si l'échantillonnage des valeurs stockées n'est pas assez dense.

Certaines optimisations, telles que les LUTs ou le passage de double en simple précision, peuvent dégrader les performances en terme de précision, mais augmenter considérablement la rapidité de l'algorithme. Des bons compromis doivent être trouvés : ainsi, dans le cas du SLAM, il est peut être intéressant de perdre un peu en précision en réalisant certaines optimisations, et utiliser le gain en temps et ressources pour corriger des amers supplémentaires, qui apporteront une précision bien plus notoire.

De plus, en fonction de la cible, il est nécessaire de prendre en considération certaines normes de développement. Notre application est destinée à être déployée sur des avions de ligne commerciaux. Les normes de développement pour systèmes embarqués liées à l'aéronautique doivent s'appliquer pour que le produit soit certifiable. Ces contraintes sont décrites dans la norme *DO-178B* [Johnson, 1998]. Cette norme présente cinq niveaux de criticité d'un défaut du système ou sous-système, niveaux notés de A à E :

- Niveau A : ce défaut peut provoquer un problème **catastrophique**. L'erreur ou la perte d'une fonction est critique pour un vol et un atterrissage en sécurité. Risque de crash.
- Niveau B : ce défaut peut provoquer un problème **dangereux**. La panne d'une fonction peut entraîner un impact négatif sur la sécurité ou les performances de vol, ou peut causer des accidents dangereux ou mortels pour les passagers ou le personnel.
- Niveau C : ce défaut peut provoquer un problème **majeur**. Un problème peut entraîner un inconfort des passagers ou une charge de travail supplémentaire significative pour l'équipage.
- Niveau D : ce défaut peut provoquer un problème **mineur**. La panne est remarquée, mais n'a qu'un léger impact, comme une incommodation des passagers ou un changement dans la routine de vol.
- Niveau E : ce défaut peut provoquer un problème **sans effet** sur la sécurité du vol.

Ces cinq niveaux sont aussi appelés DAL, acronyme anglais pour *Design Assurance Level*. Dans le cas de notre système, qui aide le pilote lors de sa phase de roulage au sol, nous devons développer les algorithmes au niveau DAL-D. Ce niveau de criticité engendre des modifications dans le codage. Ainsi, les règles strictes du C-ANSI doivent être respectées, l'allocation dynamique de mémoire, les appels récursifs, ou l'utilisation de librairies externes sont interdits.

6.2.2 RT-SLAM

Nous avons participé durant quelques mois à une implémentation logicielle d'un SLAM EKF en C++ présentée dans [Roussillon et al., 2011]. Cette version reprend les principales étapes du SLAM EKF présentée en chapitre 4 : elle exécute le SLAM en temps réel sur des données venant d'une caméra monochrome cadencée à 60Hz et d'une centrale inertielle cadencée à 100Hz via un couplage serré. Un GPS peut être utilisé pour l'étape de correction. L'architecture de ce logiciel a été conçue pour permettre des extensions vers le multi-robot, les amers hétérogènes, le multi-capteurs... Plusieurs extensions sont en cours de réalisation (bi-cam, SLAM sur points et segments, SLAM multi-cartes...). Notons que cette volonté de réaliser un programme très ouvert, par ailleurs disponible à la communauté en Open Source, a conduit à exploiter

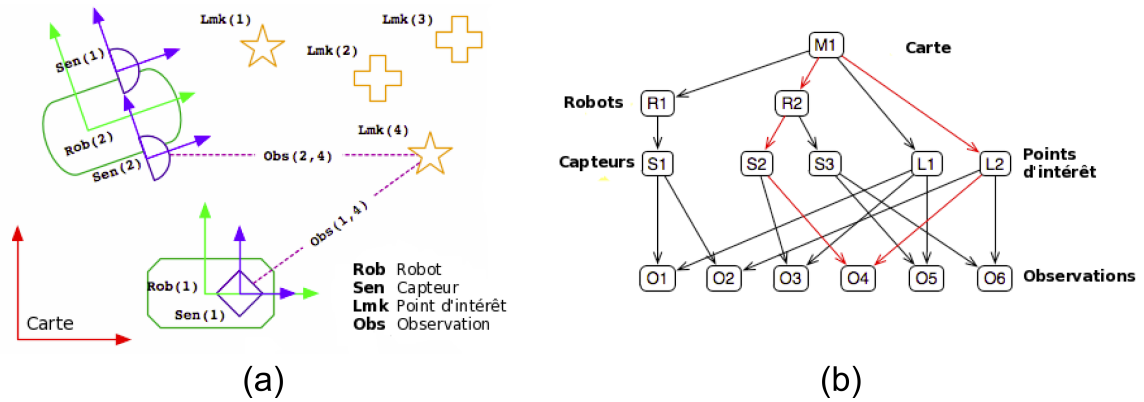


FIGURE 6.2 – Présentation de RT-SLAM. A gauche, les différents éléments pouvant composer RT-SLAM : l'observation d'amers de différentes natures peut être réalisée par plusieurs robots. A droite, l'architecture de haut-niveau de RT-SLAM.

des outils de programmation C++ assez sophistiqués (*héritages multiples, smart-pointers, templates...*) ; la figure 6.2 présente la structure de cette version logicielle. Le package RT-SLAM est disponible à l'adresse : http ://www.openrobots.org/wiki/rtslam.

Le package RT-SLAM intègre les dernières avancées en termes de SLAM EKF visuel. Par exemple, nous pouvons noter l'utilisation de la paramétrisation des amers en AHP et de leur reparamétrisation en Euclidien lors de leur convergence, la gestion des points incohérents (*Outliers*) par la méthode de *One-Point Ransac*, l'*active search* avec prédiction de l'apparence d'un point d'intérêt depuis le point de vue courant par déformation du descripteur, ainsi que la prise en compte du SLAM-bicam.

Certaines optimisations ont été réalisées pour améliorer l'étape de traitement d'image, en utilisant par exemple des *images intégrales* [Viola and Jones, 2001] pour accélérer à la fois le détecteur de points de Harris et une corrélation basée sur un score ZNCC calculé de manière hiérarchique [Stefano et al., 2005].

Le package RT-SLAM a été testé en temps réel de différentes manières : à l'image des travaux de Davison ou de Klein, il a d'abord été évalué pour des applications émergentes sur Smartphones, avec une caméra couplée à une centrale Xsens, déplacée à la main par un opérateur (voir les vidéo sur le site RT-SLAM). Puis, RT-SLAM a été testé en temps réel pour des applications en Robotique tout terrain (*Field Robotics*), sur un robot d'exploration parcourant des distances de plusieurs centaines de mètres et à vitesse élevée [Roussillon and Lacroix, 2012].

Notre participation à ce projet, en forte interaction avec une équipe de développeurs coordonnée par J.Sola, a durée quelques mois. Les règles de développement spécifiques à l'intégration du SLAM sur des systèmes embarqués, nous ont conduit à concevoir une autre architecture logicielle pour le SLAM EKF, qui respecte les normes en vigueur pour les logiciels embarqués sur aéronef.

6.2.3 C-Slam : un SLAM dédié aux systèmes embarqués

En effet, l'implémentation RT-SLAM ne respecte pas les contraintes DAL qui nous sont imposées. Elle a été développée en C++ et non en C, elle réalise des allocations dynamiques, ou utilise des librairies externes telles que *Boost*, *QT*, *OpenCV*... Une seconde implémentation logicielle a dû être développée en C-ANSI, et est présentée en [Gonzalez et al., 2011]. Cette

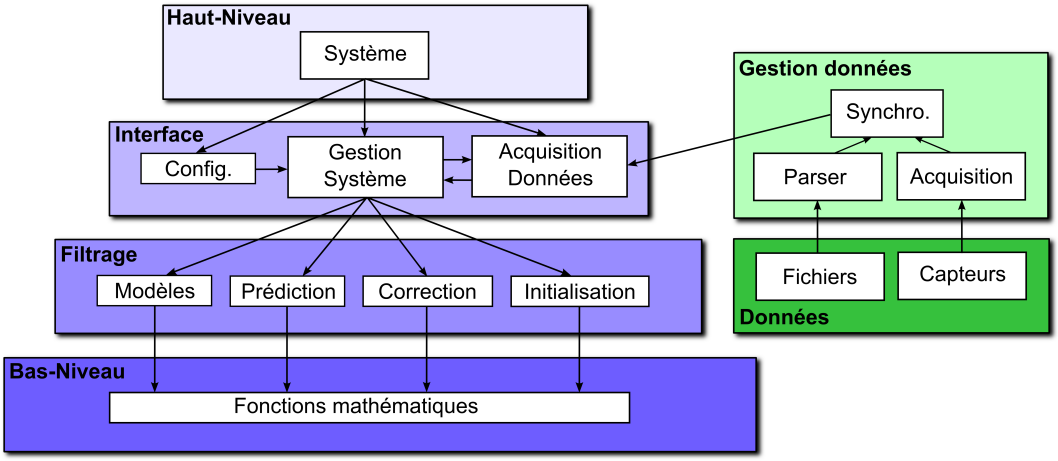


FIGURE 6.3 – Vue d'ensemble de l'architecture de C-SLAM.

nouvelle version du SLAM a deux buts :

- D'une part, elle a été développée pour une cible composée de circuits FPGA et équipée d'un processeur, soit intégré en dur sur la puce avec le FPGA (*hardcore* : PowerPC, VIA, ARM...), soit créé sur le FPGA (*softcore* : MicroBlaze...). Néanmoins, ses blocs fonctionnels restent modulables, et l'algorithme se porte aisément sur divers types de cartes, comme les Virtex5 et 6, les VIA EPIA ou encore les ARM9.
- D'autre part, cette version doit respecter les normes strictes de codage en C (C-ANSI) et les normes *DO-178* de développement pour l'aéronautique embarquée.

Outre la programmation en ANSI-C et ce respect des normes, par rapport à RT-SLAM, nous avons souhaité conserver deux caractéristiques majeures : la modularité et l'architecture hiérarchique. En effet, cette version C-SLAM doit être facilement manipulable en fonction des données d'entrée qui sont, soit des données provenant directement des capteurs, soit des données enregistrées pour un rejeu. Cette modularité vient directement de sa structure organisée en quatre couches comportant des fonctions de haut à plus bas niveau.

L'architecture de ce programme est illustrée en figure 6.3. Celle-ci se divise en deux parties principales : *Acquisition de données* et *Système*.

La partie *Acquisition de données* gère les données provenant, soit directement des capteurs, soit des fichiers images et/ou fichier textes dans le cas d'un rejeu. Ce bloc regroupe, par exemple, une interface pour l'utilisation de fonctions *OpenCV* nécessaires au bon fonctionnement du SLAM lors d'une évaluation sur un ordinateur de bureau. Elle est donc aisément interchangeable pour pouvoir traiter des données procurées par d'autres modules, tels que les résultats des traitements réalisés par le FPGA de la carte. Ainsi, l'interfaçage avec OpenCv se fait facilement via les fonctions de haut niveau que l'on peut désactiver lors du portage sur architecture embarquée.

L'ensemble *Système* est quant à lui divisé en quatre couches :

- La partie **Haut-Niveau** permet de définir le type des entrées (caméra temps réel, données de rejeu, données simulées...). Nous définissons également le type de modèle de mouvement à utiliser (*vitesse constante*, *odométrie*, *inertiel*, *inertiel et GPS*...).
- La section **Interface** réalise les appels des fonctions principales nécessaires au bon déroulement du SLAM. Tout d'abord, elle charge le fichier de configuration qui permet d'attribuer les valeurs des différents paramètres nécessaires aux algorithmes ou au filtre (nombre de points maximum dans la carte, nombre d'amers à initialiser et à corriger à

chaque itération, nombre de corrections par RANSAC ou par Recherche Active, taille des descripteurs, seuils...). Dans ce fichier est également indiqué le type de précision à utiliser (simple ou double). Après l'initialisation des paramètres et des fonctions, cette section gère la boucle principale du programme et veille ainsi au bon fonctionnement du filtre de Kalman étendu. Elle réalise l'appel aux trois sous-fonctions principales de ce filtre qui sont *l'initialisation*, la *prédiction* et la *correction*. Enfin, cette fonction reçoit les différentes données issues des capteurs ou des fichiers, les met en forme et les redistribue en fonction des besoins de chaque étape du filtre.
- Les opérations du filtre de Kalman sont contenues dans le bloc de **Filtrage**. Chaque fonction traite une étape de l'EKF.
- La couche **Bas-Niveau** comporte les différentes fonctions mathématiques, telles que les différentes opérations vectorielles ou matricielles. Ces opérations ont dû être récrites pour ne pas utiliser de librairies externes afin de pouvoir répondre aux contraintes de développement.

Ce programme est conçu pour avoir un module facilement interchangeable par un autre, ou porté sur une partie différente de l'architecture matérielle. Par exemple, au lieu d'utiliser le FPU du processeur, certaines fonctions mathématiques peuvent être directement programmées sur les *éléments logiques* du FPGA, ou utiliser des LUT pré-calculées.

Les tests unitaires ont été prouvés grâce à la *SLAM Toolbox* développée sous MATLAB [Sola et al., 2009], dont s'inspire l'architecture de notre programme. Ceci fait de *C-Slam* un outil simple pour l'expérimentation du SLAM EKF. Comme la version RT-SLAM, mais avec des capacités d'extension moindres ou plus complexes à mettre en oeuvre vu les règles de développement, ce programme est en évolution permanente et est disponible en *Open Source* sur le dépôt : http ://c-slam-laas.googlecode.com/svn/trunk/.

6.3 Le co-design

6.3.1 Introduction

L'objectif du co-design est d'identifier les fonctions coûteuses et/ou parallélisables du SLAM afin de les porter sur les éléments logiques du FPGA.

Remarque 6.1 *Les opérateurs flottants sont plus lents et plus coûteux que les opérateurs en virgule fixe en terme de ressources du FPGA. Ils sont ainsi beaucoup plus lents que les opérateurs flottants très optimisés des processeurs du commerce. Typiquement, un opérateur flottant implémenté en FPGA a un débit de crête (en termes d'opérations par seconde) au moins dix fois inférieur à son équivalent dans un processeur de la même génération [Ligon III et al., 1998] [Underwood, 2004]. Cependant, le fait de pouvoir charger l'ensemble des calculs, de les réaliser à la chaîne et en parallèle*[1] *peut devenir très avantageux en fonction des opérations à réaliser.*

6.3.2 Identification des fonctions coûteuses

La ressource principale qui limite le fonctionnement de notre algorithme est le processeur (ou CPU). Pour identifier les fonctions qui consomment le plus de ressources CPU, nous avons utilisé un outil de profilage. Ainsi, nous déterminons les fonctions qui, une fois portées sur FPGA, vont libérer un nombre important de ressources CPU. De plus, si ces fonctions peuvent être parallélisables, le gain sera plus important.

1. Cette opération est connue sous le terme anglais de *"pipeline"*.

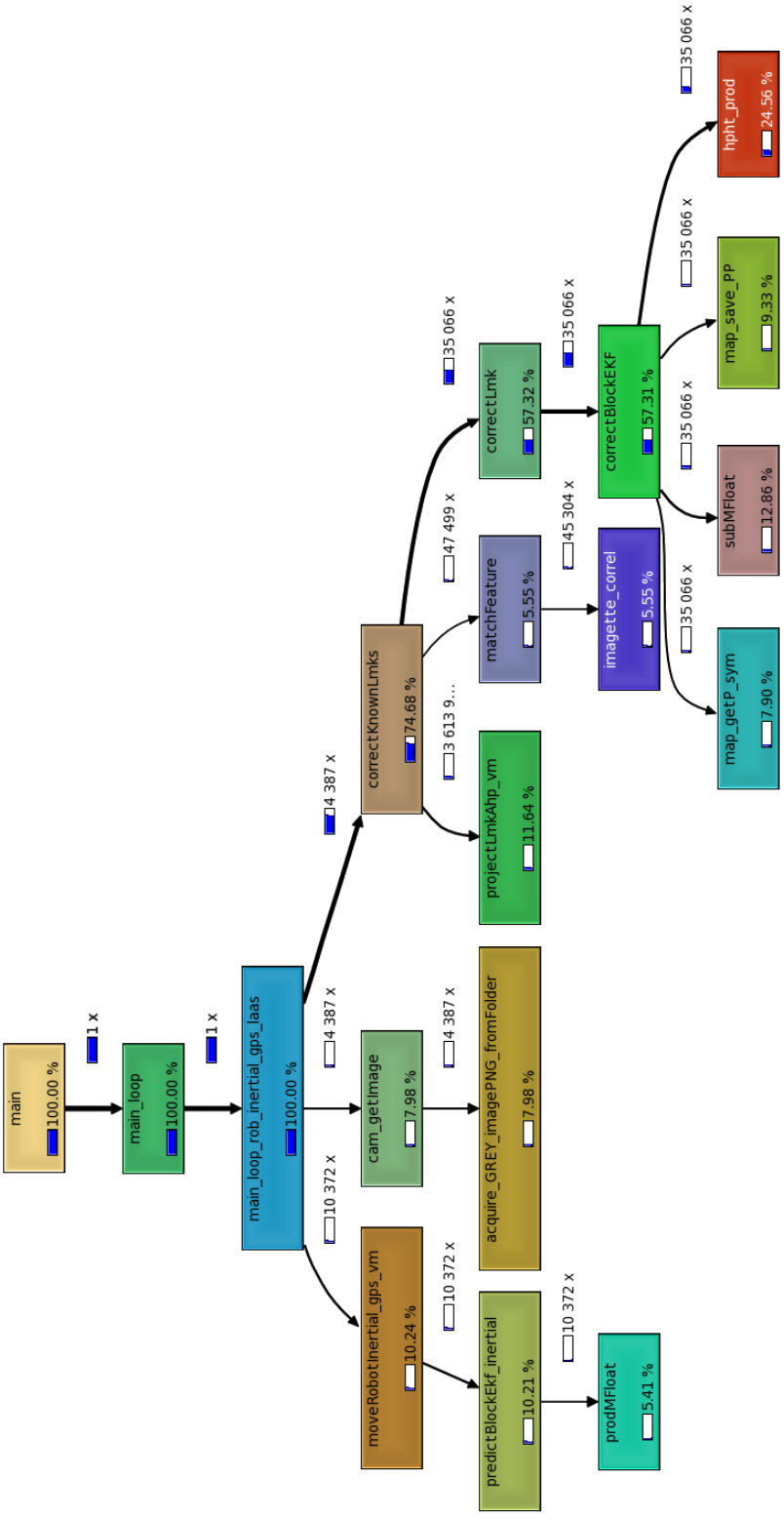


FIGURE 6.4 – Profilage du C-SLAM pour l'identification des fonctions coûteuses.

La figure 6.4 présente le profilage du *C-SLAM*. Il a été réalisé sur quelques centaines d'itérations sur la séquence de *Caylus*. Nous avons utilisé un modèle de mouvement inertiel et des corrections GPS. La carte est limitée à 300 états et 8 corrections sont réalisées à chaque itération, 4 par Recherche Active et 4 par One-Point RANSAC.

Ce résultat confirme que la complexité du SLAM EKF provient de l'étape de correction. En effet, les $^3/_4$ des calculs sont réalisés lors de l'étape de correction des amers. Pour cette étape, une part non-négligeable des ressources est utilisée pour la projection des amers. Bien que cette opération soit peu onéreuse, elle est réalisée pour chaque amer à chaque nouvelle image, soit près de 1000 fois par seconde. De plus, elle fait appel à des fonctions non-linéaires telles que les fonctions trigonométriques ou polynomiales. Cependant, la majorité des ressources est utilisée lors des calculs matriciels, engendrés par la correction des amers. Ces calculs représentent 77% de l'étape de la correction, et sur une vue globale, l'algorithme passe plus de la moitié de son temps à effectuer ces opérations matricielles. A cause du grand nombre d'éléments à traiter, le temps utilisé pour les accès mémoire est également non-négligeable.

Remarque 6.2 *Le tableau ci-dessous évalue le coût de chaque opération matricielle d'un algorithme de SLAM EKF. Nous traitons dans cet exemple un SLAM ayant une carte de 300 états et où 8 corrections sont réalisées à chaque itération. Les amers adoptent une paramétrisation AHP uniquement.*

TABLE 6.1 – Nombre d'opérations mathématiques de base réalisées lors des différentes multiplications matricielles du SLAM EKF.

Opération	Mul.	Add. / Sous.	Total Unit.	Total final
$\mathbf{P}^+_{\mathcal{RR}} = \mathbf{F}_\mathcal{R} \cdot \mathbf{P}_{\mathcal{RR}} \cdot \mathbf{F}_\mathcal{R}^\mathsf{T} + \mathbf{F_u} \cdot \mathbf{U} \cdot \mathbf{F_u}^\mathsf{T}$	15086	14136	29222	29222
$\mathbf{P}^+_{\mathcal{RM}} = \mathbf{F}_\mathcal{R} \cdot \mathbf{P}_{\mathcal{RM}}$	108300	102600	210900	210900
$\mathbf{Z}_i = \mathbf{H}_i \cdot \mathbf{P} \cdot \mathbf{H}_i^\mathsf{T} + \mathbf{R}$	784	732	1516	12128
$\mathbf{K}_i = \mathbf{P} \cdot \mathbf{H}_i^\mathsf{T} \cdot \mathbf{Z}_i^{-1}$	15704	15052	30756	246048
$\hat{X}^+ = \hat{X} + \mathbf{K}_i \cdot \hat{\mathbf{z}}_i$	1200	900	2100	16800
$\mathbf{P}^+ = \mathbf{P} - \mathbf{K}_i \cdot \mathbf{Z}_i \cdot \mathbf{K}_i^\mathsf{T}$	181200	180600	361800	2894400

Cependant, la matrice de covariance $\mathbf{P}$ *est carrée et symétrique. Nous pouvons donc n'en calculer que le triangle supérieur ou inférieur. De plus, l'opération de multiplication de* $\mathbf{P}$ *par une matrice à gauche et cette même matrice transposée à droite peut être simplifiée.*

6.3.3 Architecture Matérielle

Dans [Vincke et al., 2012], Vincke *et al.* concluent que pour obtenir de bons résultats d'un SLAM embarqué, le logiciel doit être en adéquation avec l'architecture. Bien que l'architecture de *C-SLAM* soit modulable, il a été conçu pour que son coeur soit porté sur un processeur d'une carte comprenant un FPGA, afin de pouvoir développer des opérateurs câblés pour exécuter les fonctions coûteuses en temps de traitement sur processeur.

Les FPGAs font partie de la famille des réseaux logiques programmables. Ce sont des circuits composés de nombreuses cellules logiques élémentaires qui peuvent être connectés de manière définitive, ou reconfigurables par programmation. Ces éléments logiques (ou blocs) peuvent être des LUTs, des registres, des multiplexeurs ou tout autre fonction logique.

L'architecture que nous proposons, présentée en [Botero et al., 2012], est illustrée en figure 6.5. Celle-ci peut être divisée en deux parties, qui sont séparées par le *Peripheral Component*

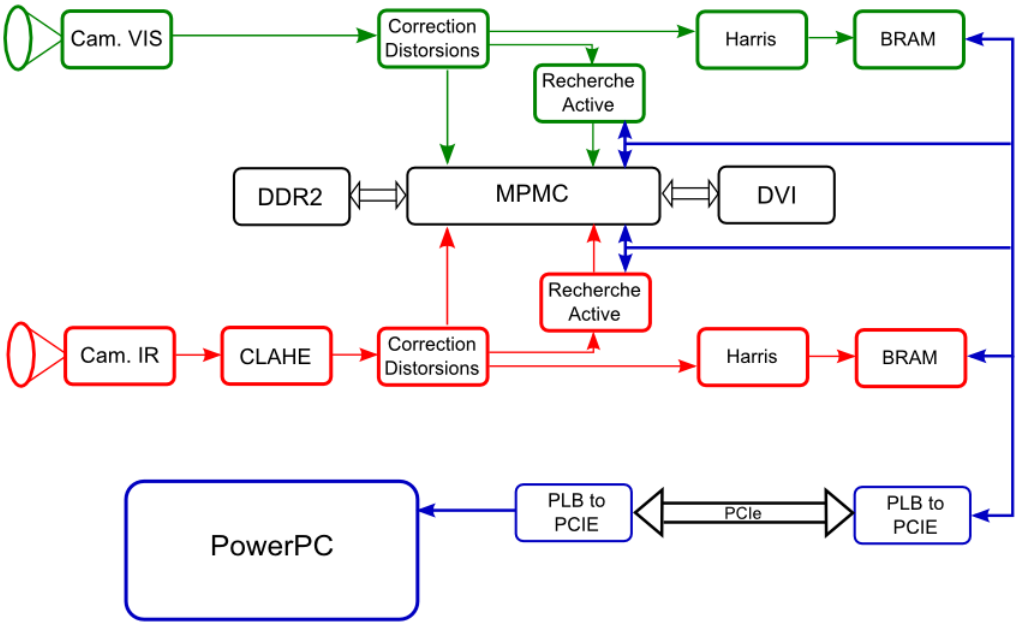


FIGURE 6.5 – Vue d'ensemble de l'architecture de C-SLAM.

Interconnect express (PCIe). Celui-ci est directement connecté au *Processor Local Bus* (PLB) qui a en charge l'arbitrage et le contrôle des communications. D'un côté de ce bus se situe le processeur sur lequel est embarqué le SLAM. Dans le cas d'un Virtex5 de Xilinx, ce processeur est cadencé à 400 MHz et implémente un *Floating Point Unit* (FPU) cadencé à 200 MHz, dont le but est de réaliser les opérations en virgule flottante. Si l'on implémente la partie logicielle de l'algorithme SLAM sur un processeur softcore MicroBlaze sur un Virtex6 par exemple, celui-ci sera cadencé seulement à 200 MHz. Vient ensuite la partie matérielle se situant de l'autre côté du PLB.

Les FPGAs permettent de réaliser un grand nombre d'opérations en parallèle à une fréquence élevée, mais présentent des performances très médiocres dans le cas de calculs en virgule flottante. Ils sont donc d'un grand intérêt pour les fonctions de traitement d'images telles que la rectification géométrique ou l'extraction de point de Harris. Comme illustré en figure 6.4, les fonctions relatives à l'image (recherche de points et appariements) représentent 13.5% de l'ensemble des ressources utilisées. En plus de libérer des ressources, porter ces opérations sur FPGA permet de les rendre parallélisables et peuvent s'exécuter à la fréquence pixel, diminuant ainsi la latence et les temps de calcul. Bien qu'étant effectuée en virgule flottante, les opérations matricielles peuvent être parallélisables. Porter ces opérations sur les blocs logiques peut être très avantageux car ces opérations sont le coeur de la complexité du SLAM EKF.

6.3.4 Accélérateurs matriciels

Les accélérateurs matriciels ont été implémentés dans le but de réaliser les multiplications matricielles qui sont récurrentes dans le cas du SLAM EKF. Le produit de deux matrices étant facilement parallélisable, il est intéressant d'utiliser la structure d'un FPGA pour réaliser des tâches simultanément.

Les opérations à réaliser sont alors *pipelinées*, et en un cycle d'horloge, trois multiplications, trois additions et une accumulation sont effectuées à une cadence de 300 MHz.

Les opérations réalisées sont en virgule flottante simple précision. Bien que le temps pour

chaque opération soit plus long par rapport à l'utilisation d'une virgule fixe, le gain par rapport à l'unique utilisation d'un FPU est important. Celui-ci, cadencé à 200 MHz ne peut exécuter qu'une seule opération à la fois, et peut demander plusieurs cycles d'horloge, comme illustré en figure 6.1 à droite. De plus, les calculs sont réalisés par des éléments extérieurs au processeur, laissant celui-ci libre pour d'autres tâches. Cependant des opérations supplémentaires sont nécessaires pour le transfert des données.

L'accélérateur développé pour ce projet est présenté dans [Tertei and Piat, 2012]. Tertei *et al.* estiment un gain d'un facteur 13 entre la version logicielle du produit de matrice sur PowerPC avec et sans accélérateur matriciel. Cependant, le gain réel peut être plus important si l'on estime que pendant la réalisation des calculs, le processeur a la possibilité de réaliser d'autres opérations. L'architecture développée par D. Tertei au cours de son mastère, a été validée en simulation et est en cours d'intégration avec les autres composantes du SLAM.

6.4 Résultats

6.4.1 Validation sur machine

Avant de porter l'algorithme *C-SLAM* sur une architecture embarquable, il peut être intéressant de savoir en premier lieu s'il peut répondre aux contraintes fréquentielles imposées. Selon les travaux présentés en [Gonzalez et al., 2011], un aperçu de la fréquence de fonctionnement de l'algorithme sur un système embarqué peut être obtenu par le calcul du rapport des fréquences entre le processeur de la machine servant aux tests et celui de l'architecture cible. Ainsi, en réalisant les tests sur une machine classique, l'influence des paramètres sur la fréquence et la cohérence de l'algorithme peuvent être observés plus facilement. Les résultats présentés ci-après ont été réalisés en moyennant 20 expérimentations Monte-Carlo. Nous utilisons un modèle de mouvement inertiel et intégrons des observations GPS bruitées à 8m. Le processeur sur lequel nous exécutons le programme C-SLAM, est cadencé à 2,8 GHz.
Afin de garantir une localisation en temps réel sur notre PowerPC, il nous faut obtenir une fréquence de fonctionnement de 210 Hz, tout en garantissant un résultat cohérent. Les calculs ont été effectués en simple précision, permettant ainsi de réaliser un gain de **10%** par rapport à des calculs en double précision.

La figure 6.6(a) présente la fréquence obtenue pour une à 20 corrections par image. Différentes tailles de carte variant de 70 états à 310 ont été étudiées. Une des premières observations réalisée est l'augmentation de la fréquence de fonctionnement pour un faible nombre de corrections avant une décroissance quasi-logarithmique. Cette augmentation de la vitesse d'exécution s'explique par un SLAM devenant cohérent. En effet, avec peu de corrections d'amers par image, l'information apportée au filtre n'est pas assez pertinente et les incertitudes sont de plus en plus importantes. Ceci engendre des zones de recherche de plus en plus grandes qui font diverger le SLAM. A partir de 5 corrections par itération, l'information apportée est suffisante et les zones de recherche diminuent. Cependant, l'algorithme reste sensible aux faux-appariements. Pour 6 corrections par image, le SLAM conserve sa cohérence dans la quasi-totalité des expérimentations et semble être un bon compromis dans le rapport perte de fréquence / information apportée. Au-delà de 8 à 10 corrections, l'information apportée est minime. Cependant, un nombre supérieur de corrections permet de conforter le filtre, et peut également s'avérer nécessaire dans le cas de capteurs bruités ou de fortes accélérations du mobile.

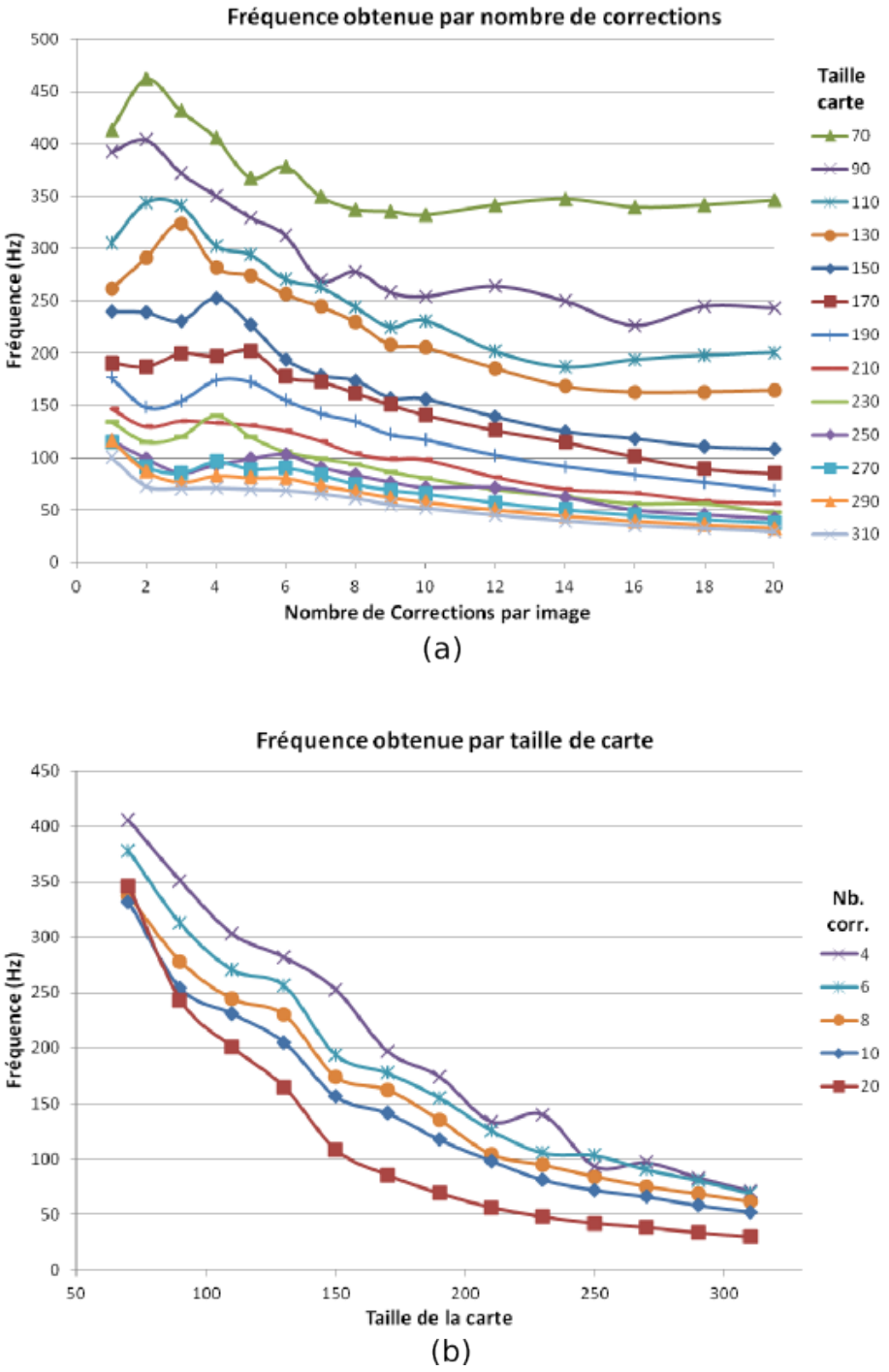


FIGURE 6.6 – (a) Influence du nombre de corrections d'amers par itération sur la fréquence de fonctionnement. Cette influence a été étudiée pour différentes tailles de cartes. (b) Influence de la taille de la carte sur la fréquence de fonctionnement. Cette influence a été étudiée pour différents nombres de corrections.

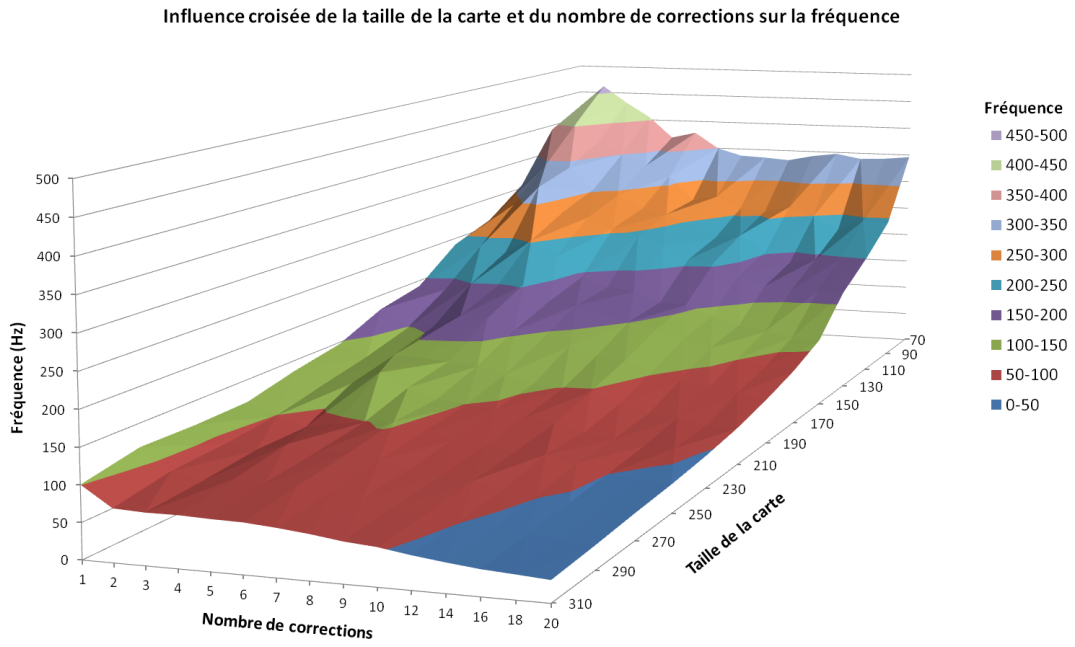


FIGURE 6.7 – Influence croisée de la taille de la carte et du nombre de corrections sur la fréquence de fonctionnement. De manière à garantir une application temps réel sur notre architecture cible, les paramètres sélectionnés doivent permettre une fréquence de fonctionnement de 210 Hz, et donc se trouver au-delà de la zone mauve.

La figure 6.6(b) présente la fréquence obtenue pour une taille de carte variant de 70 à 310 états. Cette fréquence est étudiée pour un nombre différent de corrections par image. Ces courbes permettent de mesurer l'impact de la taille de la carte sur la fréquence de fonctionnement. La décroissance est également quasi-logarithmique. Afin de garantir une application temps réel, une taille de carte variant entre 120 et 150 état peut être considérée. Cela nous permet de maintenir environ 20 amers dans la carte, en plus des états occupés par la paramétrisation du robot. Ce nombre de points est correct pour une application d'odométrie visuelle mais il faut veiller à ce que les observations des amers initialisés dans la carte soient régulièrement répartis dans l'image.

Enfin, la figure 6.7 présente l'influence croisée de la taille de la carte et du nombre de corrections sur la fréquence de fonctionnement. Cette figure permet une observation tridimensionnelle des paramètres affectant la fréquence de fonctionnement. Pour garantir une fréquence de fonctionnement temps réel sur notre architecture cible, ces paramètres doivent appartenir aux zones au-delà de celle en mauve.

Les résultats présentés nous permettent de valider le choix du filtre de Kalman étendu pour une application de SLAM sur système embarqué. Ils confirment également les différents choix algorithmiques et valident une gamme de paramètres permettant l'obtention d'un SLAM temps réel sur une architecture pourvue de ressources limitées.

6.4.2 Validation sur système embarqué

Dans cette section, nous présentons les résultats obtenus lors du fonctionnement du *C-SLAM* sur un Virtex5 de Xilinx. Les algorithmes ont été portés sur le processeur de type PowerPC inclus dans la carte. Celui-ci est cadencé à 400 MHz, et un FPU cadencé à 200 MHz

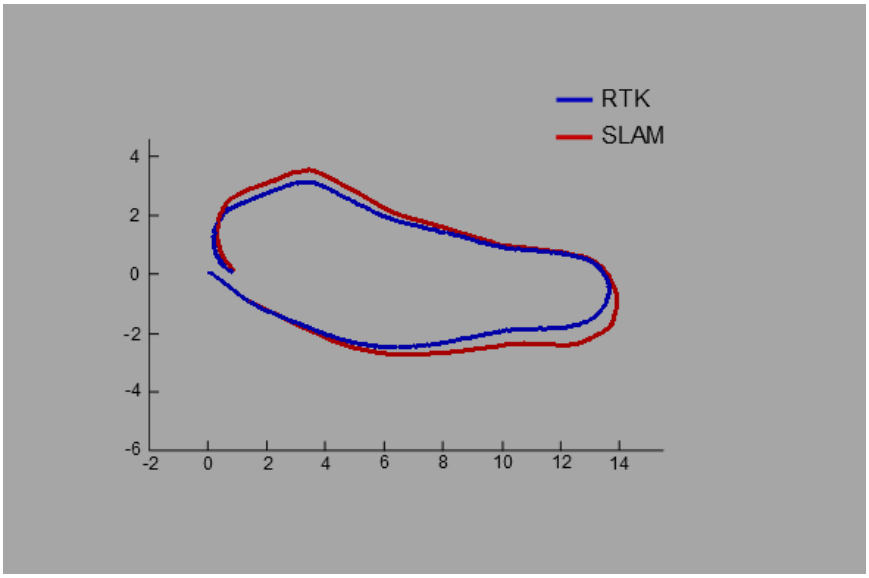


FIGURE 6.8 – Trajectoire obtenue lors de l'exécution de l'algorithme sur notre Virtex5.

est utilisé. Pour ces expérimentations, les images sont traitées sur un ordinateur classique (simulant ainsi les traitements réalisés sur FPGA) et les résultats de ces traitements (points observés et appariés) sont transmis au Virtex5 par PCI-Express [2]. Les accélérateurs matriciels ne sont pas utilisés.

La figure 6.8 représente la trajectoire obtenue après exécution de l'algorithme sur le système embarqué. La séquence étudiée ici a déjà été présentée en figure 4.11, mais contrairement à cette dernière, nous utilisons dans le cas présent un modèle de mouvement inertiel et des observations GPS. La vérité terrain apparait en bleu tandis que la trajectoire obtenue est en rouge. Sur cette séquence, l'erreur moyenne est de 0,38 m et l'erreur maximale est de 0,93 m. Ce résultat confirme ainsi la cohérence de l'algorithme une fois celui-ci porté sur une plate-forme embarquée.

La figure 6.9 représente le pourcentage de temps effectif pris par les opérations réalisées par le Virtex5 pour une carte de 150 états et 5 corrections par image. Le profilage présenté en figure 6.4, sur la base d'une analyse de C-SLAM sur un ordinateur classique, est proche du résultat obtenu ici. Près des $^2/_3$ des ressources sont consacrées aux calculs matriciels, ce qui indique l'importance des accélérateurs matriciels. La gestion de matrices de taille importante impacte de manière significative les accès mémoire. Ainsi, $^1/_4$ du temps est consacré à ces accès. Le reste des ressources utilisé est divisé entre les fonctions de communications, les fonctions de C standard, les fonctions mathématiques (opérations de base, racines, fonctions trigonométriques...), les multiplications matricielles de faibles tailles...

Le résultat des fréquences obtenues lors de l'exécution de *C-SLAM* sur Virtex5 est illustré en figure 6.10 . Celle-ci représente l'influence croisée de la taille de la carte et du nombre de corrections sur la fréquence de fonctionnement. A un facteur près, les résultats sont similaires à ceux obtenus sur ordinateur. Cependant, ils restent en deçà de ceux préalablement estimés.

2. Cette communication par PCI-Express a été mise en oeuvre durant le stage de A. Valade encadré par J. Piat.

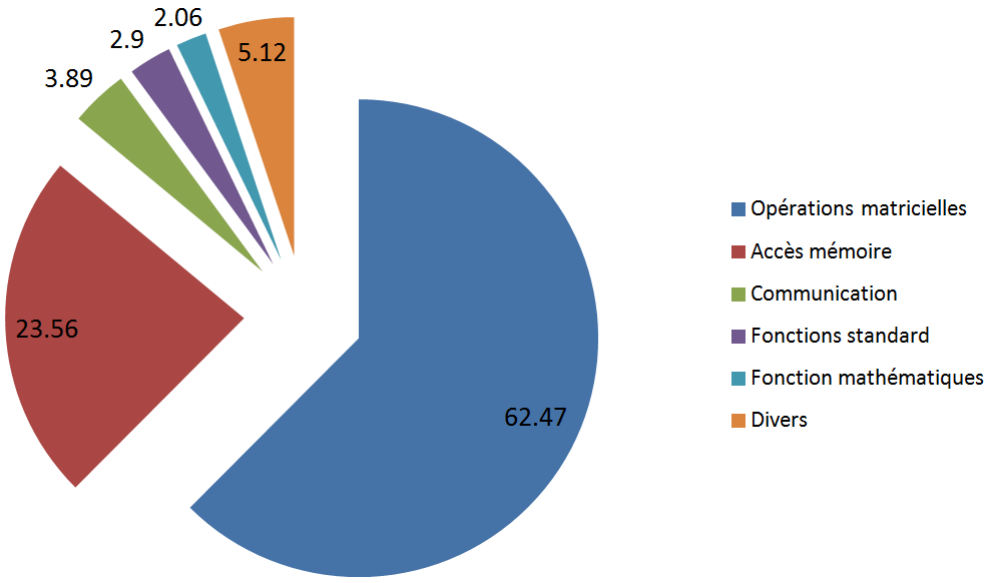


FIGURE 6.9 – Pourcentage de temps passé dans chaque fonctionnalité de l'algorithme.

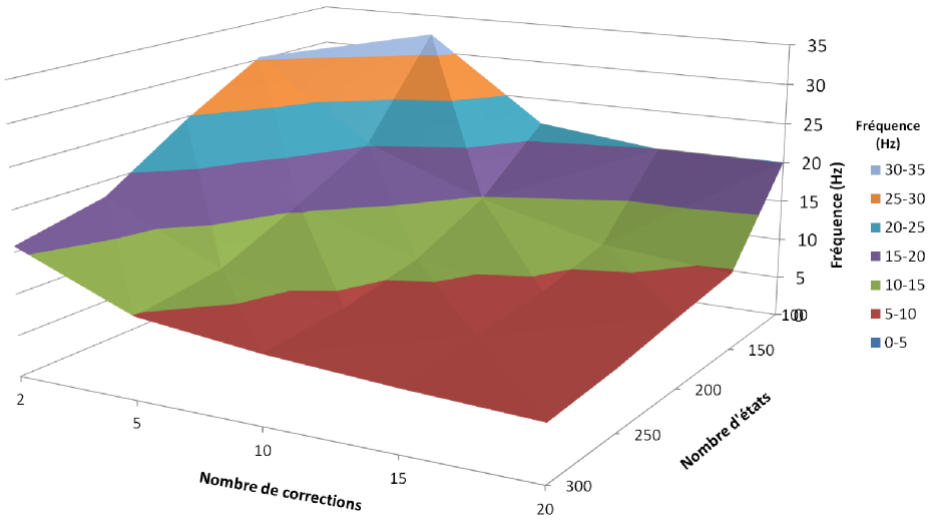


FIGURE 6.10 – Influence croisée de la taille de la carte et du nombre de corrections sur la fréquence de fonctionnement après portage des algorithmes sur le PowerPC et utilisation d'accélérateurs matriciels.

Notre estimation du temps d'exécution de C-SLAM sur un processeur cible Y (ici, Power PC du Virtex5 à 400 MHz) à partir du temps d'exécution sur un processeur X (ici, le processeur i7 de notre portable à 2,8 GHz), estimation obtenue sur la base du rapport des fréquences entre les processeurs X et Y, était donc optimiste ; cela peut s'expliquer par les temps d'accès mémoire (taille du cache, type des mémoires accédées...) et par les caractéristiques différentes des processeurs et des bus de communication. De plus, les calculs sont réalisés par le FPU qui n'est cadencé qu'à 200 MHz.

Grâce au concours actif de J. Piat, des travaux d'intégration de l'algorithme C-SLAM sont toujours en cours sur différentes architectures. A ce jour, des performances de SLAM temps réel sont obtenues sur une carte *Raspberry Pi* (ARM11 cadencé à 700 MHz) et sur une carte *PandaBone*(ARM Cortex-A9 Dual-Core cadencé à 1 GHz) avec une carte SLAM contenant 140 états et en effectuant 6 corrections d'amers par images (toujours avec un traitement d'image déporté). Ces performances, meilleures sur ces cartes, s'expliquent par des tailles et des fréquences des bus mémoires plus élevées que sur Virtex5, des FPU ayant également des cadences supérieures et des caches mémoires plus importants. De plus, le jeu d'instruction est de meilleure qualité sur ARM que sur PowerPC. Notons que les accélérateurs sur FPGA ont été partiellement validés dans le cadre des travaux de thèse de D. Botero ; l'intégration de l'ensemble est en cours.

6.5 Conclusion

Dans ce chapitre, nous avons présenté une implémentation d'un SLAM EKF sur une architecture embarquable pourvue de ressources limitées (puissance de calcul, mémoire, énergie...). Grâce au profilage du code, nous avons pu déterminer les fonctions coûteuses lors de leur exécution sur un processeur et pouvant potentiellement être portées sur les éléments logiques d'un FPGA ; les développements sur FPGA ont été faits par d'autres membres de l'équipe, D. Botero [Botero G., 2012] pour les accélérateurs en traitements d'image, D. Tertei [Tertei and Piat, 2012] pour les accélérateurs des calculs matriciels. Les résultats qualitatifs présentés en section 4.7.1 et l'étude fréquentielle réalisée en amont du portage nous ont permis de valider nos choix algorithmiques. Face au compromis précision / fréquence, ces résultats arrêtent les valeurs des paramètres à utiliser.

Cette étude a permis le portage réalisé avec succès sur un Virtex5 de Xilinx dans le cadre du projet SART décrit au chapitre 1. Cependant, les fréquences obtenues après ce portage sont inférieures à celles estimées au départ. L'utilisation d'accélérateurs matriciels et la libération de ressources au niveau du processeur nous confortent dans l'obtention de fréquences de fonctionnement proches de celles estimées en premier lieu.

Les travaux se poursuivent pour améliorer également les performances du SLAM sur d'autres architectures, offrant dès à présent des performances temps réel (Raspberry, PandaBone...), dans le cadre de nouveaux projets, toujours relatifs au Transport, soit sur aéronef, soit sur automobile.

Chapitre 7

Conclusion

7.1 Conclusion générale

En premier lieu, ces travaux de thèse ont eu pour but d'étudier les méthodes de Localisation et Cartographie Simultanées existants, et d'en concevoir et développer un, adapté aux systèmes embarqués. Il s'avère que cette problématique n'a été que peu abordée par le passé. C'est seulement depuis ces dernières années qu'elle constitue un axe de recherche, et cela pour différentes raisons :

- Les progrès très rapides des algorithmes de SLAM fondés sur la vision, font qu'en moins de 10 ans, le SLAM visuel est sorti des laboratoires, pour être intégré dans de nombreuses applications sur véhicules ou sur des ordinateurs portés par l'homme.
- Contrairement aux méthodes fondés sur la télémétrie, le coût d'un système exploitant le SLAM par la vision est surtout donné par le coût du matériel dédié au calcul, d'où l'intérêt des recherches sur des architectures à ressources limitées, mais suffisantes pour obtenir une précision convenable pour une application donnée.
- Enfin du fait des évolutions très rapides des architectures matérielles (multi-coeurs, FPGA, GPU, DSP...), il existe une grande diversité pour embarquer des algorithmes complexes.

La participation au développement d'un SLAM EKF en C++ fonctionnant avec des images acquises à 60 Hz nous a conforté dans la viabilité du déploiement d'un tel algorithme, fonctionnant en temps réel, sur un processeur cadencé à seulement 400 MHz et atteindre une précision de localisation suffisante. Cependant, l'ensemble des fonctionnalités ont dû être re-développées pour répondre aux contraintes des systèmes embarqués dans l'avionique. Ainsi, nous avons développé *C-SLAM*, un algorithme de localisation basé sur un filtre de Kalman étendu. Grâce à une allocation fixe des mémoires, à l'utilisation d'aucune librairie externe additionnelle et un développement en C standard des différentes fonctions, le risque de défaillances est minimisé. Cet algorithme est ainsi simple d'utilisation et léger pour un portage sur diverses plates-formes.

Différents capteurs ont également été étudiés pour traiter de cette problématique de localisation, et plus particulièrement la caméra infrarouge. Ces caméras, pouvant mesurer les émissions thermiques dans différentes bandes spectrales, sont de plus en plus abordables, et ne sont plus réservées au seul domaine du militaire. Ainsi, elles fournissent une information de nuit ou par météo très dégradée, contrairement aux caméras visibles. Couplées à différents capteurs comme une centrale inertielle et un GPS au sein d'un algorithme de localisation, elles peuvent servir au guidage et repérer les obstacles et dangers potentiels, même par des conditions atmosphériques peu favorables.

Grâce à la mise à disposition d'une base de donnée cartographique de l'environnement dans lequel se déplace notre mobile (un aéronef dans les phases de roulage dans un aéroport),

nous avons présenté une méthode pour l'intégration d'informations connues a priori dans le processus de localisation. Nous avons défini comment extraire des primitives visuelles de l'image, et comment apparier celles-ci à des éléments de la base de données. Ces appariements permettent de localiser notre mobile dans la carte a priori, par une méthode de minimisation d'une erreur dans l'image. Le résultat de cette localisation est intégré dans le filtre de Kalman comme observation de la même manière que les observations fournies par le GPS.

Cependant, la mise en oeuvre d'un algorithme de SLAM sur un système embarqué reste complexe, car les ressources sont limitées. La complexité du SLAM provenant du nombre d'éléments dans la carte (état du robot et amers initialisés dans la carte) et du nombre d'amers corrigés par image, plusieurs stratégies de sélection, de recherche et de correction d'amers ont dû être mises en place ; par ailleurs il est nécessaire de gérer la carte pour supprimer les amers trop anciens, ce qui rend impossible la fermeture de boucle, nous rapprochant ainsi plus d'un système d'odométrie visuelle ou de SLAM à mémoire courte. Bien que les algorithmes de sélection et de recherche d'amers aient été optimisés, ces travaux ne suffisaient pas à l'obtention de résultats en temps réel, donc pour notre application, à 30 Hz sur un Virtex5 doté d'un processeur PowerPC cadencé à 400 MHz. Un co-design logiciel / matériel a dû être développé afin d'accélérer certaines opérations. Après étude des fonctions coûteuses dans le cadre des travaux d'autres membres de l'équipe, certaines opérations ont pu ainsi être portées sur les éléments logiques du FPGA, comme le traitement d'images et certaines opérations matricielles récurrentes.

L'architecture de *C-SLAM*, destinée à être déployé facilement sur des systèmes embarqués, a permis d'évaluer l'algorithme sur des cartes plus modernes et plus puissantes où des fonctionnements robustes et en temps réel ont pu être obtenus.

Les travaux présentés dans cette thèse ont été fortement influencés par le projet SART et par les contraintes industrielles. De ce fait, on pourrait dire que nos résultats sont plus d'ordre technologique que réellement scientifique, et donc, on pourrait s'interroger sur la nature de notre contribution. Cependant, bien que peu étudié par la communauté, nous maintenons que le développement d'un SLAM embarqué pour de grandes trajectoires reste une problématique scientifique d'actualité. Le SLAM visuel embarqué présente un intérêt industriel majeur qui, malgré la puissance embarquée disponible actuellement, n'est pas encore résolu.

7.2 Perspectives

Ces travaux ont permis de prouver la viabilité d'un système de localisation basé sur une plateforme embarquée, et de nouvelles perspectives de développement peuvent alors être considérées :

Localisation multi-spectrale

Les contraintes temporelles du projet SART, ainsi que les problèmes rencontrés dans l'exploitation des images de la caméra développée par FLIR pour ce projet, ne nous a pas permis d'atteindre l'ensemble des objectifs fixés en matière de localisation multi-spectrale. L'utilisation conjointe des amers provenant de la caméra couleur et de la caméra infrarouge est à l'étude. Une approche de type SLAM bi-caméra pourrait être considérée, bien qu'un amer initialisé depuis une image acquise par la caméra couleur ne soit pas forcément observable par la caméra infrarouge, et réciproquement. De plus, grâce à cette approche multi-spectrale, la chrominance et la signature thermique d'un objet peuvent être utilisées conjointement pour identifier celui-ci.

Identification des objets

Une des contraintes du projet SART était de repérer et d'identifier les différents objets présents dans les images visibles et infrarouges, les objets étant soit les amers (balises, bords de pistes...), soit de potentiels obstacles pour l'aéronef (autre aéronef, véhicule, piéton...). Ainsi, ils peuvent être transférés au module IHM qui peut les mettre en évidence dans l'interface présentée au pilote. Les approches classiques sont basées sur la texture des objets. Cependant, une telle opération s'avère complexe. En effet, la texture est inutilisable lors de conditions météorologiques dégradées, et l'image infrarouge en est dépourvue. De plus, les méthodes basées sur l'apprentissage et la classification peuvent être consommatrices de ressources, et donc peu adaptées aux systèmes embarqués.

Vers de nouvelles architectures

Grâce à notre approche de co-design, nous avons pu évaluer la précision d'une localisation par odométrie visuelle et fusion de données acquises par plusieurs capteurs en temps réel. Cela était suffisant dans le cadre du projet SART, vu l'absence de rebouclage dans la trajectoire suivie par un aéronef sur un aéroport.

Afin de se rapprocher d'une application de SLAM à mémoire longue, il est nécessaire de détecter les fermetures de boucles, soit en augmentant la taille de la carte, soit en adoptant une approche multi-cartes, comme cela a été proposé dans différents contextes, par D. Marquez (apprentissage d'une trajectoire longue) et de C. Roussillon et T. Vidal (SLAM multi-robots). Mais cette gestion d'une carte de grande taille ou de plusieurs cartes, accroît également le nombre d'opérations. Les nouvelles architectures multi-coeurs peuvent alors être considérées et analysées pour cette problématique, tout en gardant des opérateurs fortement parallélisables pour les fonctions de traitement d'images.

SLAM par optimisation et systèmes embarqués

Les plate-formes embarquées disposent maintenant de suffisamment de ressources pour intégrer des algorithmes de localisation probabiliste si une adéquation logiciel / matériel est mise en place. Les méthodes basées sur l'optimisation de graphes prouvent, tous les jours un peu plus, leur efficacité face aux méthodes probabilistes. Avec le système PTAM, Klein et Newman ont montré que, sur des machines récentes, les résultats peuvent être précis et obtenus rapidement. Au vu des ressources actuelles disponibles sur les systèmes embarqués, et des progrès réalisés sur ces méthodes, elles peuvent être la source d'études pour leur portage sur ces plates-formes.

Transfert de technologie et industrialisation

Bien que certains développements soient encore nécessaires pour porter le système à maturité, le travail développé sur ce projet a permis de le rendre proche de son industrialisation. Latécoère dispose maintenant de l'ensemble des outils grâce au transfert de technologie opéré dans le cadre de cette collaboration. Mais un accompagnement paraît indispensable pour optimiser la solution proposée dans cette thèse, et surtout, pour l'adapter aux évolutions des algorithmes SLAM et des architectures matérielles disponibles.

Bibliographie

[Abrate et al., 2007] Abrate, F., Bona, B., and Indri, M. 2007. Experimental ekf-based slam for mini-rovers with ir sensors only. In *Preceedings of 3rd European Conference on Mobile Robots, European Conference on Mobile Robots*.

[Agrawal and Konolige, 2006] Agrawal, M. and Konolige, K. 2006. Real-time localization in outdoor environments using stereo vision and inexpensive gps. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*. Vol. 3. IEEE, 1063–1068.

[Airbus, 2009] Airbus. 2009. Brevet : Method and device for aiding the airport navigation. `http://www.faqs.org/patents/app/20090265089`.

[Argiles et al., 2011] Argiles, A., Civera, J., and Montesano, L. 2011. Dense multi-planar scene estimation from a sparse set of images. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 4448–4454.

[Bailey et al., 2006] Bailey, T., Nieto, J., Guivant, J., Stevens, M., and Nebot, E. 2006. Consistency of the ekf-slam algorithm. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. IEEE, 3562–3568.

[Berger et al., 2007] Berger, C., Lacroix, S., et al. 2007. Modélisation de l'environnement par facettes planes pour la cartographie et la localisation simultanées par stéréovision.

[Betge-Brezetz et al., 1996] Betge-Brezetz, S., Hebert, P., Chatila, R., and Devy, M. 1996. Uncertain map making in natural environments. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*. Vol. 2. 1048 –1053 vol.2.

[Boeing, 2002] Boeing. 2002. Brevet : Airplane ground maneuvering camera system. `http://patft.uspto.gov/netacgi/nph-Parser?Sect2=PTO1&Sect2=HITOFF&p=1&u=/netahtml/PTO/search-bool.html&r=1&f=G&l=50&d=PALL&RefSrch=yes&Query=PN/6405975`.

[Botero et al., 2012] Botero, D., Gonzalez, A., Devy, M., et al. 2012. Architecture embarquée pour le slam monoculaire. In *Actes de la conférence RFIA 2012*.

[Botero G., 2012] Botero G., D. A. 2012. Development of algorithms and architectures for driving assistance in adverse weather conditions using fpgas. Ph.D. thesis, INSA de Toulouse.

[Bulata and Devy, 1996] Bulata, H. and Devy, M. 1996. Incremental construction of a landmark-based and topological model of indoor environments by a mobile robot. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*. Vol. 2. IEEE, 1054–1060.

[Chabert et al., 2012] Chabert, G., Araya, I., Neveu, B., Ninin, J., Jaulin, L., Trombettoni, G., and Anthony, B. 2012. Ibex. `http://www.emn.fr/z-info/ibex/`.

[Chapuis et al., 2002] Chapuis, R., Aufrere, R., and Chausse, F. 2002. Accurate road following and reconstruction by computer vision. *IEEE Transactions on Intelligent Transportation Systems*.

[Chong and Kleeman, 1999] Chong, K. S. and Kleeman, L. 1999. Feature-based mapping in real, large scale environments using an ultrasonic array.

[Civera et al., 2008] Civera, J., Davison, A., and Montiel, J. 2008. Inverse depth parametrization for monocular slam. *Robotics, IEEE Transactions on 24,* 5, 932–945.

[Civera et al., 2009] Civera, J., Grasa, O., Davison, A., and Montiel, J. 2009. 1-point ransac for ekf-based structure from motion. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on.* 3498 –3504.

[Conte and Doherty, 2008] Conte, G. and Doherty, P. 2008. An integrated uav navigation system based on aerial image matching. In *Aerospace Conference, 2008 IEEE.* IEEE, 1–10.

[Csorba, 2007] Csorba, C. 2007. Simultaneous localisation and map building.

[Davison, 2003] Davison, A. 2003. Real-Time Simultaneous Localisation and Mapping with a Single Camera. In *Proc. IEEE Int. Conf. on Computer Vision (ICCV).*

[Davison and Murray, 1998] Davison, A. and Murray, D. 1998. Mobile robot localisation using active vision. *Computer Vision-ECCV'98*, 809–825.

[Davison et al., 2007] Davison, A., Reid, I., Molton, N., and Stasse, O. 2007. Monoslam : Real-time single camera slam. *IEEE Trans. Pattern Anal. Mach. Intell. 29*, 1052–1067.

[Dellaert and Kaess, 2005] Dellaert, F. and Kaess, M. 2005. Square root sam. In *Proc. of robotics : science and systems (RSS).* 177–184.

[Di Marco et al., 2001] Di Marco, M., Garulli, A., Lacroix, S., and Vicino, A. 2001. Set membership localization and mapping for autonomous navigation. *International Journal of robust and nonlinear control 11,* 7, 709–734.

[Dissanayake et al., 2001] Dissanayake, M. W. M. G., Newman, P., Clark, S., Durrant-whyte, H. F., and Csorba, M. 2001. A solution to the simultaneous localization and map building (slam) problem. *IEEE Transactions on Robotics and Automation 17*, 229–241.

[Ducarouge, 2011] Ducarouge, B. 2011. Reconstruction 3d infrarouge par perception active. Ph.D. thesis, INSA de Toulouse.

[Estrada et al., 2005] Estrada, C., Neira, J., and Tardós, J. D. 2005. Hierarchical slam : real-time accurate mapping of large environments. *Robotics, IEEE Transactions on 21,* 4, 588–596.

[Folkesson and Christensen, 2008] Folkesson, J. and Christensen, H. 2008. Sift based graphical slam on a packbot. In *Field and Service Robotics.* Springer, 317–328.

[Gifford et al., 2008] Gifford, C., Webb, R., Bley, J., Leung, D., Calnon, M., Makarewicz, J., Banz, B., and Agah, A. 2008. Low-cost multi-robot exploration and mapping. In *Technologies for Practical Robot Applications, 2008. TePRA 2008. IEEE International Conference on.* IEEE, 74–79.

[Gonzalez et al., 2011] Gonzalez, A., Codol, J., and Devy, M. 2011. A c-embedded algorithm for real-time monocular slam. In *Proc. IEEE Int. Conf. on Electronics, Circuits and Systems, Beyrouth (Liban).*

[Gonzalez et al., 2011] Gonzalez, A., Devy, M., Ortega, J. S., et al. 2011. Slam visuel monoculaire par caméra infrarouge. In *ORASIS-Congrès des jeunes chercheurs en vision par ordinateur.*

[Hajebi and Zelek, 2008] Hajebi, K. and Zelek, J. 2008. Structure from infrared stereo images. In *Computer and Robot Vision, 2008. CRV'08. Canadian Conference on.* IEEE, 105–112.

[Harris and Pike, 1988] Harris, C. and Pike, J. 1988. 3d positional integration from image sequences. *Image and Vision Computing 6,* 2, 87–90.

[Harris and Stephens, 1988] Harris, C. and Stephens, M. 1988. A combined corner and edge detector. In *Alvey vision conference*. Vol. 15. Manchester, UK, 50.

[Hartley and Zisserman, 2000] Hartley, R. and Zisserman, A. 2000. *Multiple view geometry in computer vision*. Vol. 2. Cambridge Univ Press.

[Jaulin, 2009] Jaulin, L. 2009. A nonlinear set membership approach for the localization and map building of underwater robots. *Robotics, IEEE Transactions on 25,* 1, 88–98.

[Jaulin and Walter, 1993] Jaulin, L. and Walter, E. 1993. Set inversion via interval analysis for nonlinear bounded-error estimation. *Automatica 29,* 4, 1053–1064.

[Jazwinski, 1970] Jazwinski, A. 1970. *Stochastic processes and filtering theory*. Vol. 63. Academic press.

[Johnson, 1998] Johnson, L. 1998. Do-178b : Software considerations in airborne systems and equipment certification. `http://www.davi.ws/avionics/TheAvionicsHandbook_Cap_27.pdf`.

[Julier and Uhlmann, 2001] Julier, S. J. and Uhlmann, J. K. 2001. A counter example to the theory of simultaneous localization and map building. In *In IEEE Intl. Conf. on Robotics and Automation (ICRA*. 4238–4243.

[Jung and Lacroix, 2003] Jung, I. and Lacroix, S. 2003. High resolution terrain mapping using low attitude aerial stereo imagery. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*. IEEE, 946–951.

[Kaess and Dellaert, 2006] Kaess, M. and Dellaert, F. 2006. Visual slam with a multi-camera rig.

[Kaess et al., 2008] Kaess, M., Ranganathan, A., and Dellaert, F. 2008. isam : Incremental smoothing and mapping. *Robotics, IEEE Transactions on 24,* 6, 1365–1378.

[Kalman, 1960] Kalman, R. 1960. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME–Journal of Basic Engineering 82,* Series D, 35–45.

[Kim and Sukkarieh, 2005] Kim, J. and Sukkarieh, S. 2005. 6dof slam aided gnss/ins navigation in gnss denied and unknown environments. *Journal of Global Positioning Systems 4,* 1-2, 120–128.

[Klein and Murray, 2007] Klein, G. and Murray, D. 2007. Parallel tracking and mapping for small AR workspaces. In *Proc. 6th IEEE and ACM Int. Symp. on Mixed and Augmented Reality*. IEEE Computer Society.

[Konolige et al., 2011] Konolige, K., Agrawal, M., and Sola, J. 2011. Large-scale visual odometry for rough terrain. *Robotics Research*, 201–212.

[Konolige et al., 2008] Konolige, K., Augenbraun, J., Donaldson, N., Fiebig, C., and Shah, P. 2008. A low-cost laser distance sensor. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE, 3002–3008.

[Latécoère, 2009] Latécoère. 2009. Brevet : Système d'aide au roulage tout-temps (sart). `http://worldwide.espacenet.com/publicationDetails/originalDocument?FT=D&date=20090409&DB=&locale=en_EP&CC=WO&NR=2009044257A2&KC=A2&ND=2`.

[Lemaire and Lacroix, 2007] Lemaire, T. and Lacroix, S. 2007. Monocular-vision based slam using line segments. In *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, 2791–2796.

[Leonard and Feder, 2001] Leonard, J. and Feder, H. 2001. Decoupled stochastic mapping [for mobile robot amp ; auv navigation]. *Oceanic Engineering, IEEE Journal of 26,* 4 (oct), 561 –571.

[Ligon III et al., 1998] Ligon III, W., McMillan, S., Monn, G., Schoonover, K., Stivers, F., and Underwood, K. 1998. A re-evaluation of the practicality of floating-point operations on fpgas. In *FPGAs for Custom Computing Machines, 1998. Proceedings. IEEE Symposium on.* IEEE, 206–215.

[Lim et al., 2012] Lim, H., Sinha, S. N., Cohen, M. F., and Uyttendaele, M. 2012. Real-time image-based 6-dof localization in large-scale environments. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on.* IEEE, 1043–1050.

[Lourakis and Argyros, 2004] Lourakis, M. and Argyros, A. 2004. The design and implementation of a generic sparse bundle adjustment software package based on the levenberg-marquardt algorithm.

[Magnenat et al., 2010] Magnenat, S., Longchamp, V., Bonani, M., Rétornaz, P., Germano, P., Bleuler, H., and Mondada, F. 2010. Affordable slam through the co-design of hardware and methodology. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on.* IEEE, 5395–5401.

[Marquez-Gamez and Devy, 2011] Marquez-Gamez, D. A. and Devy, M. 2011. Visual navigation of communicating vehicles in unknown and changing environment. In *Electronics, Circuits and Systems (ICECS), 2011 18th IEEE International Conference on.* IEEE, 673–676.

[Mei, 2007] Mei, C. 2007. Laser-augmented omnidirectional vision for 3d localisation and mapping. Ph.D. thesis, PhD thesis at Ecole Nationale Suprieure Des Mines De Paris Sophia Antipolis.

[Meier et al., 2012] Meier, L., Tanskanen, P., Heng, L., Lee, G., Fraundorfer, F., and Pollefeys, M. 2012. Pixhawk : A micro aerial vehicle design for autonomous flight using onboard computer vision. *Autonomous Robots*, 1–19.

[Montemerlo et al., 2002] Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. 2002. Fastslam : A factored solution to the simultaneous localization and mapping problem. In *In Proceedings of the AAAI National Conference on Artificial Intelligence.* AAAI, 593–598.

[Montiel, 2006] Montiel, J. M. M. 2006. Unified inverse depth parametrization for monocular slam. In *Proc. Robotics : Science and Systems (RSS).*

[Morrone and Owens, 1987] Morrone, M. and Owens, R. 1987. Feature detection from local energy. *Pattern Recognition Letters 6,* 5, 303–313.

[Mouragnon et al., 2009] Mouragnon, E., Lhuillier, M., Dhome, M., Dekeyser, F., and Sayd, P. 2009. Generic and real-time structure from motion using local bundle adjustment. *Image and Vision Computing 27,* 8, 1178–1193.

[Moutarlier and Chatila, 1989] Moutarlier, P. and Chatila, R. 1989. Stochastic multisensory data fusion for mobile robot localization and environment modelling. In *Proc. 5th Int. Symp. on Robotics Research (ISRR).* 207–216.

[Neira et al., 1997] Neira, J., Ribeiro, M., Tardós, J., et al. 1997. Mobile robot localization and map building using monocular vision. In *In The 5th Symposium for Intelligent Robotics Systems.* Citeseer.

[Nikiforov, 2000] Nikiforov, I. 2000. Module : Théorie de la décision et de l'estimation : approche statistique ch. 2-3 : Théorie de l'estimation et exemples.

[NTSB, 2012] NTSB. 2012. National transportation safety board. safety recommendation. `http://www.ntsb.gov/doclib/recletters/2012/A-12-048-049.pdf`.

[Nygårds et al., 2004] Nygårds, J., Skoglar, P., Ulvklo, M., and Högström, T. 2004. Navigation aided image processing in uav surveillance : Preliminary results and design of an airborne experimental system. *Journal of Robotic Systems 21,* 2, 63–72.

[Open-SLAM, 2009] Open-SLAM. 2009. Open-SLAM. `http://openslam.org`.

[Rady et al., 2011] Rady, S., Kandil, A., and Badreddin, E. 2011. A hybrid localization approach for uav in gps denied areas. In *System Integration (SII), 2011 IEEE/SICE International Symposium on*. IEEE, 1269–1274.

[Rockwell-Collins, 2004] Rockwell-Collins. 2004. Brevet : Enhanced vision system (evs). `http://patft.uspto.gov/netacgi/nph-Parser?Sect2=PTO1&Sect2=HITOFF&p=1&u=/netahtml/PTO/search-bool.html&r=1&f=G&l=50&d=PALL&RefSrch=yes&Query=PN/7605774`.

[Rodriguez-Losada et al., 2004] Rodriguez-Losada, D., Matia, F., and Jimenez, A. 2004. Local maps fusion for real time multirobot indoor simultaneous localization and mapping. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*. Vol. 2. 1308 – 1313 Vol.2.

[Roussillon et al., 2011] Roussillon, C., Gonzalez, A., Solà, J., Codol, J., Mansard, N., Lacroix, S., and Devy, M. 2011. Rt-slam : a generic and real-time visual slam implementation. In *Proc. 8th Int. Conf. on Computer Vision Systems (ICVS)*.

[Roussillon and Lacroix, 2012] Roussillon, R. and Lacroix, S. 2012. high rate-localization for high-speed all-terrain robots. In *Communications, Computing and Control Applications, 2012. CCCA'12. Conference on*. IEEE.

[Schröter et al., 2007] Schröter, C., Böhme, H., and Gross, H. 2007. Memory-efficient gridmaps in rao-blackwellized particle filters for slam using sonar range sensors. In *Proceedings of the European Conference on Mobile Robots*. 138–143.

[Shi and Tomasi, 1994] Shi, J. and Tomasi, C. 1994. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*. IEEE, 593–600.

[Smith et al., 2006] Smith, P., Reid, I., and Davison, A. 2006. Real-time monocular slam with straight lines. In *British Machine Vision Conference*. Vol. 1. 17–26.

[Smith et al., 1988] Smith, R., Self, M., and Cheeseman, P. 1988. A stochastic map for uncertain spatial relationships. In *Proceedings of the 4th international symposium on Robotics Research*. MIT Press, Cambridge, MA, USA, 467–474.

[Solà, 2007] Solà, J. 2007. Towards visual localization, mapping and moving objects tracking by a mobile robot : a geometric and probabilistic approach.

[Sola, 2010] Sola, J. 2010. Consistency of the monocular ekf-slam algorithm for three different landmark parametrizations. In *Proc. IEEE. Int. Conf. Robotics and Automation (ICRA)*.

[Sola et al., 2009] Sola, J., Marquez, D., Codol, J., and Vidal-Calleja, T. 2009. An EKF-SLAM toolbox for MATLAB.[Online]. `http://www.joansola.eu/JoanSola/eng/toolbox.html`.

[Sola et al., 2007] Sola, J., Monin, A., and Devy, M. 2007. Bicamslam : Two times mono is more than stereo. In *Robotics and Automation, 2007 IEEE International Conference on*. 4795 –4800.

[Sola et al., 2005] Sola, J., Monin, A., Devy, M., and Lemaire, T. 2005. Undelayed initialization in bearing only slam. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*.

[Solà et al., 2012] Solà, J., Vidal-Calleja, T., Civera, J., and Montiel, J. M. M. 2012. Impact of landmark parametrization on monocular EKF-SLAM with points and lines. *International Journal of Computer Vision 97*, 339–368.

[Stefano et al., 2005] Stefano, L. D., Mattoccia, S., and Tombari, F. 2005. Zncc-based template matching using bounded partial correlation. *Pattern Recognition Letters 26,* 14, 2129 – 2134.

[Strasdat et al., 2010] Strasdat, H., Montiel, J., and Davison, A. 2010. Real-time monocular slam : Why filter ? In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*.

[Tertei and Piat, 2012] Tertei, D. and Piat, J. 2012. Implementing a joint pc-fpga platform for perception algorithms prototyping. Tech. rep.

[Thalès, 2010] Thalès. 2010. Onboard airport navigation system (oans). `http://www.thalesgroup.com/Portfolio/Documents/Aerospace_Product_OANS/`.

[Thomas, 2011] Thomas, F. 2011. Rejeu de chemin et localisation monoculaire : Application du visual slam sur carte peu dense en environnement extérieur contraint. Ph.D. thesis, Université Blaise Pascal-Clermont-Ferrand II.

[Thrun et al., 2004] Thrun, S., Koller, D., Ghahramani, Z., Durrant-Whyte, H., and Ng, A. 2004. Simultaneous mapping and localization with sparse extended information filters : Theory and initial results. *Algorithmic Foundations of Robotics V*, 363–380.

[Thrun and Montemerlo, 2005] Thrun, S. and Montemerlo, M. 2005. The GraphSLAM algorithm with applications to large-scale mapping of urban structures. *International Journal on Robotics Research 25,* 5/6, 403–430.

[Triggs et al., 2000] Triggs, B., McLauchlan, P. F., Hartley, R. I., and Fitzgibbon, A. W. 2000. Bundle adjustment - a modern synthesis. In *Vision algorithms : theory and practice*. Springer, 298–372.

[Underwood, 2004] Underwood, K. 2004. Fpgas vs. cpus : trends in peak floating-point performance. In *International Symposium on Field Programmable Gate Arrays : Proceedings of the 2004 ACM/SIGDA 12 th international symposium on Field programmable gate arrays*. Vol. 22. 171–180.

[Vidal-Calleja et al., 2011] Vidal-Calleja, T. A., Berger, C., Solà, J., and Lacroix, S. 2011. Large scale multiple robot visual mapping with heterogeneous landmarks in semi-structured terrain. *Robotics and Autonomous Systems 59,* 9, 654–674.

[Vincke, 2012] Vincke, B. 2012. Architectures pour des systèmes de localisation et de cartographie simultanées. Ph.D. thesis, Université Paris-Sud.

[Vincke et al., 2012] Vincke, B., Elouardi, A., and Alain, L. 2012. Real time simultaneous localization and mapping : towards low-cost multiprocessor embedded systems. *EURASIP Journal on Embedded Systems 2012,* 1, 5.

[Viola and Jones, 2001] Viola, P. and Jones, M. 2001. Rapid object detection using a boosted cascade of simple features. In *in Proc. IEEE Conf. Comput. Vis. Pattern Recog.* 511–518.

[Yap and Shelton, 2009] Yap, T. and Shelton, C. 2009. Slam in large indoor environments with low-cost, noisy, and sparse sonars. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 1395–1401.

[Zhang et al., 2009] Zhang, L., Zapata, R., and Lépinay, P. 2009. Self-adaptive monte carlo localization for mobile robots using range sensors. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. IEEE, 1541–1546.

Localisation par Vision Multi-spectrale
Application aux Systèmes Embarqués

Mots clés : SLAM, vision multi-spectrale, systèmes embarqués.

La problématique *SLAM* (Simultaneous Localization and Mapping) est un thème largement étudié au LAAS depuis plusieurs années. L'application visée concerne le développement d'un système d'aide au roulage sur aéroport des avions de ligne, ce système devant être opérationnel quelques soient les conditions météorologiques et de luminosité (projet SART financé par la DGE en partenariat avec principalement FLIR Systems, Latécoère et Thales).

Lors de conditions de visibilité difficile (faible luminosité, brouillard, pluie...), une seule caméra traditionnelle n'est pas suffisante pour assurer la fonction de localisation. Dans un premier temps, on se propose d'étudier l'apport d'une caméra infrarouge thermique.

Dans un deuxième temps, on s'intéressera à l'utilisation d'une centrale inertielle et d'un GPS dans l'algorithme de SLAM, la centrale aidant à la prédiction du mouvement, et le GPS à la correction des divergences éventuelles. Enfin, on intègrera dans ce même SLAM des pseudo-observations issues de l'appariement entre des segments extraits des images, et ces mêmes segments contenus dans une cartographie stockée dans une base de données. L'ensemble des observations et pseudo-observations a pour but de localiser le porteur à un mètre près.

Les algorithmes devant être portés sur un FPGA muni d'un processeur de faible puissance par rapport aux PC standard (400 MHz), un co-design devra donc être effectué entre les éléments logiques du FPGA réalisant le traitement d'images à la volée et le processeur embarquant le filtre de Kalman étendu (EKF) pour le SLAM, de manière à garantir une application temps-réel à 30 Hz. Ces algorithmes spécialement développés pour le co-design et les systèmes embarqués avioniques seront testés sur la plate-forme robotique du LAAS, puis portés sur différentes cartes de développement (Virtex 5, Raspberry, PandaBoard...) en vue de l'évaluation des performances.

Multi-spectral Vision Localisation
An Embedded Systems Application

Keywords: SLAM, multi-spectral vision, embedded systems.

The SLAM (Simultaneous Localization and Mapping) problematic is widely studied from years at LAAS. The aimed application is the development of a helping rolling system for planes on airports. This system has to work under any visibility and weather conditions ("SART" project, funding by DGE, with FLIR Systems, Thalès and Latecoère).

During some weather conditions (fog, rain, darkness), one only visible camera is not enough to complete this task of SLAM. Firstly, in this thesis, we will study what an infrared camera can bring to SLAM problematic, compared to a visible camera, particularly during hard visible conditions.

Secondly, we will focus on using Inertial Measurement Unit (IMU) and GPS into SLAM algorithm, IMU helping on movement prediction, and GPS helping on SLAM correction step. Finally, we will fit in this SLAM algorithm pseudo-observations coming from matching between points retrieved from images, and lines coming from map database. The main objective of the whole system is to localize the vehicle at one meter.

These algorithms aimed to work on a FPGA with a low-power processor (400MHz), a co-design between the hardware (processing images on the fly) and the software (embedding an Extended Kalman Filter (EKF) for the SLAM), has to be realized in order to guarantee a real-time application at 30 Hz. These algorithms will be experimented on LAAS robots, then embedded on different boards (Virtex 5, Raspberry Pi, PandaBoard...) for performances evaluation.