Localisation temps-réel d'un robot par vision monoculaire et fusion multicapteurs

Baptiste Charmette

N° d'ordre : DU 2321
EDSPIC : 599

UNIVERSITÉ BLAISE PASCAL - CLERMONT II

Ecole Doctorale
Sciences Pour L'Ingénieur De Clermont-Ferrand

Thèse

présentée par :

BAPTISTE CHARMETTE

pour obtenir le grade de

DOCTEUR D'UNIVERSITÉ

Spécialité : Vision pour la robotique

Localisation temps-réel d'un robot par vision monoculaire et fusion multicapteurs

---

Soutenue publiquement le 14 décembre 2012 devant le jury :

M. Michel DHOME	DR Institut Pascal - CNRS	Président du jury
M. François CHAUMETTE	DR INRIA Rennes	Rapporteur
M. Patrick RIVES	DR INRIA Sophia Antipolis	Rapporteur
Mme Sylvie TREUILLET	MCF Polytech'Orléans	Examinateur
M. Éric ROYER	MCF Univ. d'Auvergne	Examinateur
M. Frédéric CHAUSSE	MCF Univ. d'Auvergne	Directeur de thèse

Remerciements

Je tiens avant toute chose à remercier sincèrement tous ceux qui m'ont aidé d'une quelconque manière à l'élaboration de ce manuscrit.

Tout d'abord je remercie François Chaumette, Patrick Rives, Sylvie Treuillet et Michel Dhome qui ont accepté de faire partie de mon jury de thèse et de prendre du temps pour évaluer à sa juste valeur le travail effectué.

Je remercie également mes encadrants, Eric Royer et Frédéric Chausse qui ont su me guider tout au long de ce travail de thèse, m'apporter leur soutien et leurs idées lorsque j'en avais besoin, tout en me laissant une grande liberté d'action lorsque c'était nécessaire. J'ai pu grâce à eux m'initier au travail de recherche proprement dit, dans ce domaine de la vision et de la robotique qui m'intéresse particulièrement.

Je n'oublie pas bien sûr tous les membres du laboratoire qui m'ont largement aidé à résoudre divers problèmes au fur et à mesure de mes recherches. Je pense en particulier à l'équipe technique, Serge Alizon, François Marmoiton et Laurent Lequièvre qui par leurs expériences et savoir-faire, m'ont permis de mettre en place les différentes expérimentations, me permettant de transformer ce qui ne serait que des théories sur un papier en véritables démonstrations pratiques.

L'ensemble des doctorants du laboratoire m'ont également apporté beaucoup d'aide tant pour résoudre certains problèmes techniques, que par leur bonne humeur et la bonne ambiance qu'ils ont su mettre en place. Merci énormément à Pierre(s), Shuda, Siméon, Jean-Marc, Manu, Bertrand, Alexandre, Laetitia, Vadim, Datta, Thomas, Damien, Guillaume, Gaspard, François, Jonathan, et tous ceux que j'oublie.

Pour finir je remercie ma famille et mes amis, qui ont su m'apporter tout le soutien qui m'était nécessaire lors de la réalisation de ce mémoire. En particulier je remercie celle qui a su être là quand j'en avais vraiment besoin, et m'a permis de faire face aux moments les plus difficiles.

Résumé

Ce mémoire présente un système de localisation par vision pour un robot mobile circulant dans un milieu urbain.

Pour cela, une première phase d'apprentissage où le robot est conduit manuellement est réalisée pour enregistrer une séquence vidéo. Les images ainsi acquises sont ensuite utilisées dans une phase hors ligne pour construire une carte 3D de l'environnement. Par la suite, le véhicule peut se déplacer dans la zone, de manière autonome ou non, et l'image reçue par la caméra permet de le positionner dans la carte. Contrairement aux travaux précédents, la trajectoire suivie peut être différente de la trajectoire d'apprentissage. L'algorithme développé permet en effet de conserver la localisation malgré des changements de point de vue importants par rapport aux images acquises initialement. Le principe consiste à modéliser les points de repère sous forme de facettes localement planes, surnommées patchs plan, dont l'orientation est connue. Lorsque le véhicule se déplace, une prédiction de la position courante est réalisée et la déformation des facettes induite par le changement de point de vue est reproduite. De cette façon la recherche des amers revient à comparer des images pratiquement identiques, facilitant ainsi leur appariement. Lorsque les positions sur l'image de plusieurs amers sont connues, la connaissance de leur position 3D permet de déduire la position du robot.

La transformation de ces patchs plan est complexe et demande un temps de calcul important, incompatible avec une utilisation temps-réel. Pour améliorer les performances de l'algorithme, la localisation a été implémentée sur une architecture GPU offrant de nombreux outils permettant d'utiliser cet algorithme avec des performances utilisables en temps-réel.

Afin de prédire la position du robot de manière aussi précise que possible, un modèle de mouvement du robot a été mis en place. Il utilise, en plus de la caméra, les informations provenant des capteurs odométriques. Cela permet d'améliorer la prédiction et les expérimentations montrent que cela fournit une plus grande robustesse en cas de pertes d'images lors du traitement.

Pour finir ce mémoire détaille les différentes performances de ce système à travers plusieurs expérimentations en conditions réelles. La précision de la position a été mesurée en comparant la localisation avec une référence enregistrée par un GPS différentiel.

Mots-clés : localisation, patchs plan, temps-réel, GPU, robot mobile, appariement

Abstract

This dissertation presents a vision-based localization system for a mobile robot in an urban context.

In this goal, the robot is first manually driven to record a learning image sequence. These images are then processed in an off-line way to build a 3D map of the area. Then vehicle can be —either automatically or manually— driven in the area and images seen by the camera are used to compute the position in the map. In contrast to previous works, the trajectory can be different from the learning sequence. The algorithm is indeed able to keep localization in spite of important viewpoint changes from the learning images. To do that, the features are modeled as locally planar features —named patches— whose orientation is known. While the vehicle is moving, its position is predicted and patches are warped to model the viewpoint change. In this way, matching the patches with points in the image is eased because their appearances are almost the same. After the matching, 3D positions of the patches associated with 2D points on the image are used to compute robot position.

The warp of the patch is computationally expensive. To achieve real-time performance, the algorithm has been implemented on GPU architecture and many improvements have been done using tools provided by the GPU.

In order to have a pose prediction as precise as possible, a motion model of the robot has been developed. This model uses, in addition to the vision-based localization, information acquired from odometric sensors. Experiments using this prediction model show that the system is more robust especially in case of image loss.

Finally many experiments in real situations are described in the end of this dissertation. A differential GPS is used to evaluate the localization result of the algorithm.

Keywords : real-time, GPU, matching, planar feature, image descriptor, robot localization

Table des matières

Liste des figures

Liste des tableaux

Notations

Les notations détaillées ci-dessous concernent des conventions choisies de manière générale dans tout le document. Cependant d'autres notations pourront être ajoutées au fur et à mesure lorsque cela sera nécessaire.

- Les ensembles de valeurs seront représentés par une lettre majuscule, par exemple E, et leur nombre d'élément sera représenté par $|E|$
- Les matrices seront représentées par une lettre majuscule sans mise en forme particulière, par exemple M
- L'opération de transposition qui consiste à inverser les lignes et colonnes d'une matrice, sera symbolisé par la lettre T en exposant de la matrice, par exemple A^T représente la transposée de A
- Les vecteurs seront représentés par une lettre en gras, par exemple $\mathbf{u}$. Ils peuvent également être considérés comme une matrice colonne et à ce titre être utilisés dans des calculs matriciels (par exemple le produit scalaire P entre deux vecteurs $\mathbf{X}$ et $\mathbf{Y}$ peut être exprimé $P = \mathbf{X}^T\mathbf{Y}$.

Une image est assimilée à une fonction d'intensité de $\mathbb{R}^2$ vers $\mathbb{R}$. On peut donc définir pour un pixel de coordonnées $\mathbf{X} = \begin{bmatrix} x \\ y \end{bmatrix}$, la valeur $I(\mathbf{X})$ correspondant au niveau de gris de ce pixel. Par mesure de simplification, on considère également que $\mathbf{x}$ peut être exprimé en coordonnées homogènes. On pose donc que quelles que soient les valeurs réelles x, y et s, on a $I\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = I\left(\begin{bmatrix} sx \\ sy \\ s \end{bmatrix}\right)$.

Introduction

Localisation d'un robot par vision

De nos jours les machines prennent de plus en plus de place dans les applications courantes. Qu'il s'agisse d'appareil ménager, de déplacement de personne, d'exploration spatiale ou d'application militaire, de plus en plus de tâches sont réalisées en utilisant l'assistance d'une machine. On peut citer par exemple les aspirateurs, les régulateurs de vitesse ou encore les véhicules téléguidés qui sont autant d'appareils facilitant le travail de l'utilisateur. Le but principal de ces machines est d'éviter aux êtres humains des taches répétitives ou dangereuses.

Avec le temps ces machines, qui initialement ne fournissaient qu'une simple assistance, sont devenues de plus en plus performantes et ont évolué vers ce qui est communément appelé un robot. Une des principales évolutions ayant permis cette amélioration est la capacité pour ces machines de se déplacer et d'interagir avec leur environnement. C'est ainsi par exemple que l'aspirateur manuel a pu être transformé en robot ménager, capable de se déplacer tout seul dans une pièce pour la nettoyer, ou encore qu'un véhicule commandé à distance a pu devenir parfaitement autonome et réaliser des analyses sur une autre planète automatiquement. Cette capacité nécessite d'utiliser divers instruments observant l'environnement et informant le système des contraintes observées.

Ces instruments, appelés capteurs dans le cadre d'une machine, sont en fait ce qui va former l'équivalent de nos sens pour le robot, lui permettant de percevoir des sons, de détecter des obstacles, de mesurer des distances, etc. Certains capteurs sont très spécifiques à une information donnée, par exemple un capteur de contact ne pourra mesurer que la présence ou non d'un obstacle à l'endroit où il est placé, alors que d'autres sont capables d'enregistrer un signal complexe donnant des informations continues qu'il est nécessaire d'interpréter. C'est le cas en particulier des systèmes basés sur une caméra, où une image est fournie à chaque instant, permettant d'obtenir (moyennant un traitement parfois complexe) de nombreuses informations sur l'environnement.

Si l'on fait un parallèle avec ce qu'est capable de faire un humain, on s'aperçoit que les capteurs correspondent à tous les sens de notre organisme. En particulier la vue est le sens le plus utilisé par l'être humain tant pour s'orienter que pour évaluer d'éventuels obstacles, déterminer la position et la vitesse des objets de l'environnement ou encore reconnaitre certains objets particuliers. Même si les algorithmes mis en œuvre dans un cerveau humain sont bien plus

complexes que ce que l'on parvient à reproduire avec les ordinateurs, cela a le mérite de montrer qu'il existe des solutions pour réaliser de nombreuses tâches à partir des seules informations de vision. De nombreux travaux vont d'ailleurs dans ce sens et tentent de développer de multiples applications telles que la surveillance, la métrologie ou encore de la modélisation 3D.

Pour réaliser un robot autonome une tache particulièrement complexe est de parvenir à le localiser dans son environnement. Cette étape est particulièrement utile si le robot doit être capable de se déplacer et de revenir à son point de départ par la suite. Dans certains cas, en particulier lorsque le robot doit être utilisé dans un milieu où l'homme est peu présent, il est nécessaire de trouver des méthodes complexes de localisation et des capteurs parfois très différents de ce qui est à la portée de l'être humain pour se localiser. Par exemple dans les grandes exploitations agricoles des moissonneuses peuvent être guidées en utilisant le GPS, qui leur donne une position absolue en permanence. D'autres méthodes consistent à placer des systèmes dans l'environnement, par exemple des aimants, que le robot pourra détecter et utiliser pour se déplacer.

Cependant, lorsqu'un robot doit être utilisé en complément ou en remplacement de l'être humain, les informations nécessaires à la localisation sont souvent déjà présentes pour les humains. Elles sont généralement de type visuel. Il semble donc intéressant dans ce contexte d'utiliser des informations de vision pour localiser le robot. De nombreux travaux de recherche vont dans ce sens. Le principal problème consiste à mesurer la distance des objets. Pour cela, certains travaux utilisent plusieurs caméras, d'autres couplent les caméras à des systèmes de perception de distance tels que des télémètres laser ou ultrason. Ces systèmes ont l'avantage de pouvoir utiliser directement les informations visuelles déjà existantes dans l'environnement, sans devoir adapter l'infrastructure. De plus, contrairement aux systèmes utilisant le GPS, ils peuvent être utilisés en ville ou en intérieur même lorsque le ciel est masqué par des immeubles ou un abri, sans perte de précision.

Les transports urbains sont une application pratique de ces systèmes. Elle consiste à avoir de nombreux véhicules autonomes, capables de se déplacer dans un environnement non instrumenté, que les utilisateurs pourront emprunter directement. Il suffira alors d'entrer dans l'ordinateur de bord l'adresse de la destination pour que le véhicule emmène les passagers à l'endroit indiqué. Ensuite, le véhicule peut revenir automatiquement à un autre endroit sans nécessiter de conducteur. Les travaux présentés dans ce mémoire proposent une solution pour qu'un véhicule autonome soit capable de se localiser dans ce contexte. Cela reste une étape indispensable pour parvenir à un système de transport automatisé sûr et fiable.

Problèmes abordés dans cette thèse

L'objectif principal de cette thèse consiste à réaliser un système de localisation par vision permettant à un véhicule de se déplacer dans une zone sans perdre la connaissance de sa position. Le véhicule est d'abord conduit manuellement dans la zone d'utilisation pour modéliser l'environnement. Par la suite, il doit être capable de se déplacer sur une trajectoire dans cette

zone en étant capable si nécessaire (par exemple pour éviter un obstacle ou prendre une autre voie) de s'éloigner de la trajectoire réalisée lors de l'apprentissage.

De nombreux systèmes existants fonctionnent avec une méthode d'appariement de points d'intérêt entre les images. Une fois ces points appariés puis triangulés, le véhicule est positionné par rapport à la référence et peut être commandé pour suivre la trajectoire apprise. La partie consistant, à partir des mises en correspondance, à déterminer la position du véhicule est suffisamment fiable et robuste pour notre application. Cependant elle repose sur le fait que les points ont été correctement appariés initialement. Lorsque le véhicule s'écarte de la trajectoire d'apprentissage, cela peut impliquer un changement de point de vue important qui gène l'appariement des points observés avec leur image de référence. Les travaux dans le cadre de cette thèse se concentrent principalement sur une manière d'améliorer la mise en correspondance, en particulier lorsque le point de vue est sensiblement différent de celui de l'apprentissage.

Pour cela, la carte de l'environnement contenant les points de référence est modifiée. Nous proposons en particulier de modéliser les points de repère de manière à être reconnus lorsque le véhicule les aperçoit depuis un point de vue différent de celui d'origine. De plus, afin de ne pas confondre les points de référence, la position du véhicule est prédite pour anticiper la déformation de ces objets. Cette prédiction est réalisé en considérant le déplacement du véhicule mesurée au fur et à mesure de la trajectoire, ou pour la première image, est supposée connue.

Pour réaliser cette prédiction les informations issues de capteurs odométriques sont utilisées. En effet l'odométrie du moteur et l'angle de braquage du véhicule sont mesurés et peuvent permettre d'anticiper les changements de point de vue, améliorant l'appariement avec les points de référence.

Pour finir la localisation doit être réalisée dans un processus temps réel de manière à connaitre la position du véhicule en chaque instant, sans avoir un déplacement trop important entre deux localisations. L'algorithme a été implémenté sur une architecture de type GPU, permettant d'obtenir les performances désirées.

Contributions et plan de thèse

Pour réaliser tous ces objectifs de localisation par vision malgré un changement de point de vue, il a été nécessaire de considérer les solutions existantes afin d'évaluer leurs limites et les points d'amélioration. Le chapitre 1 permet de décrire les différentes méthodes existantes, tout d'abord pour réaliser la localisation d'un robot dans son ensemble, mais aussi simplement sur les différents moyens de réaliser la mise en correspondance. L'algorithme initial sur lequel est basée la méthode est également décrit.

Afin d'améliorer la reconnaissance de point d'intérêt malgré l'écart à la trajectoire, nous avons défini une modélisation de chaque amer observé sous forme de petite facettes planes dont la texture et l'orientation sont connues. Ces facettes seront nommées patch-plan dans la suite de ce document. Cette modélisation permet d'anticiper les déformations liées aux changements de point de vue et ainsi être capable de reconnaitre plus facilement un point de référence et

ce, même lors d'écart important par rapport à la trajectoire d'apprentissage. L'algorithme de construction de ces patchs et leur utilisation lors de la localisation est décrite dans le chapitre 2. Une comparaison de cet algorithme avec une méthode utilisant un descripteur de référence a également été menée pour montrer l'apport de ces patchs dans un contexte d'appariement seul.

Afin d'obtenir des performances temps-réel de l'algorithme, il est nécessaire de réaliser une implémentation performante utilisant les ressources d'une architecture GPU. Les détails de cette implémentation sont décrits dans le chapitre 3. Une évaluation des performances temporelles de cette implémentation est également réalisée.

L'algorithme utilisant les patchs repose sur la possibilité d'anticiper les déformations liées au changement de point de vue. Dans ce but il est nécessaire de prédire la position du véhicule aussi finement que possible. Pour cela il est intéressant d'utiliser un algorithme basé sur un filtre de Kalman tenant compte à la fois des positions précédentes du véhicule et des informations fournies par des capteurs mesurant l'odométrie des moteurs et l'angle de braquage du véhicule. Cette prédiction est décrite en détail dans le chapitre 4.

Pour finir, afin d'évaluer les performances de nos travaux, plusieurs expérimentations ont été menées. Le chapitre 5 détaille un certain nombre de résultats illustrant le fonctionnement correct de l'algorithme et montrant ses limites dans les cas les plus difficiles.

Cadre de cette thèse : projet ANR CityVIP

Ces travaux de thèse ont été menés dans le cadre du projet ANR CityVIP [1]. Ce projet a pour objectif le développement d'avancées méthodologiques et technologiques dans le domaine de la gestion des déplacements urbains en mode assistance à la conduite mais également en mode conduite automatique. Le projet consiste donc d'une part à développer les algorithmes nécessaires pour atteindre l'objectif, mais également à en faire des expérimentations finales pour démontrer dans un contexte urbain réaliste la pertinence des solutions réalisées.

Les différents partenaires de ce projet sont

- ➢ le laboratoire XLIM de l'Université de Limoges
- ➢ le laboratoire Heudiasyc UMR6599 CNRS/Université de Technologie de Compiègne
- ➢ l'IFSTTAR, Centre de Nantes
- ➢ l'équipe AROBAS de l'INRIA Sophia-Antipolis
- ➢ l'équipe-projet LAGADIC de l'INRIA Rennes-Bretagne Atlantique
- ➢ Le laboratoire LASMEA [2] UMR CNRS/Université Blaise Pascal de Clermont Ferrand dans lequel sont réalisés les travaux de cette thèse
- ➢ Le laboratoire MATIS de l'IGN
- ➢ La société BeNomad spécialisée dans l'édition de logiciels de cartographie et située à Villeneuve Loubet dans les Alpes Maritimes.

1. http://projet_cityvip.byethost33.com
2. devenu Institut Pascal depuis le 1er janvier 2012

Ce projet est découpé en 4 lots :

1. Localisation précise et intègre
2. Conduite Autonome
3. Sécurité des déplacements en environnement urbain
4. Base de données 3D géo-référencées pour la navigation

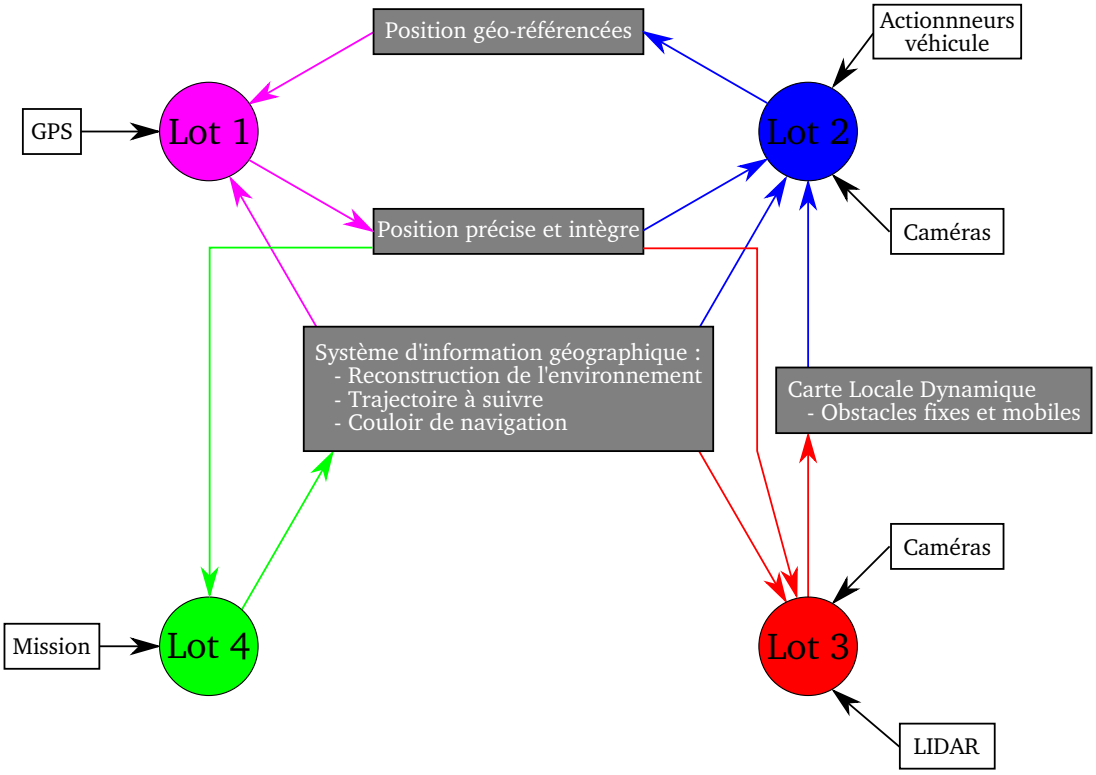


FIGURE 1 – Organisation en lots du projet CityVIP

Chaque lot s'interface avec les autres suivant le schéma de la figure 1. Les travaux décrits dans ce mémoire correspondent plus précisément au Lot 2, qui consiste à guider le véhicule par vision. Il utilise une prédiction de la pose qui peut soit provenir d'une interaction avec le lot 1, comme sur la figure 1, soit être issu directement de l'algorithme décrit dans le chapitre 4.

Chapitre 1

État de l'art

1.1 Stratégie de localisation par vision

La localisation est essentielle pour la mise en œuvre d'un robot autonome. En effet, il est souvent indispensable pour gérer les déplacements d'un véhicule de savoir où il est situé par rapport à son objectif. Pour réaliser cette tâche, de multiples capteurs peuvent être utilisés, tels que le GPS qui donne une position absolue, ou encore l'odométrie permettant de connaitre le déplacement du véhicule. La vision est également un moyen intéressant. C'est en effet celui que les humains utilisent naturellement pour se déplacer. De plus, c'est un capteur à faible cout qui ne nécessite aucune infrastructure spéciale et peut fonctionner tant en intérieur qu'en extérieur. Cependant lorsque l'on essaie d'intégrer la notion de localisation pour un robot, on s'aperçoit que cela peut signifier des choses bien différentes selon l'application.

1.1.1 Stratégie basée sur la position

La localisation d'un robot consiste à déterminer sa position par rapport à un ensemble de points de repère connus. Plus généralement lorsque le robot est mobile, on essaie d'obtenir sa trajectoire dans un repère de référence. L'ensemble des points de repère peut être rassemblé sous la forme d'une carte qui définit également le repère de référence. Se localiser revient donc à se situer dans cette carte. Par exemple connaitre la latitude et la longitude d'un objet permet de savoir où il est situé sur le globe terrestre. Ou encore connaissant le plan d'une pièce, on veut savoir où est situé le robot dans la pièce afin de pouvoir passer la porte. Dans le cadre de la navigation autonome, il est essentiel de connaitre également l'endroit où le robot désire aller dans cette même carte. Cela permet de définir une trajectoire qui sera suivie.

Les points de repère stockés dans la carte doivent être définis comme des éléments que le robot peut percevoir par le biais de ses capteurs. Si le robot utilise une caméra, les repères doivent être associés à des images, que le robot comparera aux images obtenues par sa caméra. Par exemple si le robot sait qu'il y a un arbre à tel endroit, mais qu'aucune information visuelle

ne décrit ce qu'est un arbre, il n'aura aucun moyen de retrouver le point. Par contre si quelque part est indiqué que un arbre doit avoir une apparence sous la forme d'un tronc et de branches et qu'une image de ces données est fournie, le robot pourra comparer les données à celles obtenues par sa caméra.

Pour résumer se localiser consiste à relier les informations perceptibles par les capteurs avec des éléments présents dans une carte, de manière à obtenir sa position sur celle-ci. Ce problème de localisation correspond en fait à deux approches dont l'objectif est différent :

- ➢ l'objectif peut être de connaitre la trajectoire du véhicule en cartographiant l'environnement au fur et à mesure ;
- ➢ lorsque le véhicule doit se déplacer dans un environnement connu, l'objectif est plutôt de connaitre sa position (et sa trajectoire) au sein d'une carte existante.

Selon le cas, la localisation et les contraintes peuvent varier. La première partie décrira brièvement les travaux existant pour créer la carte en même temps que la localisation. Dans une seconde partie, la localisation au sein d'une carte existante sera approfondie.

Cartographie et localisation simultanée

Lorsque un véhicule est utilisé pour explorer une zone inconnue, il ne dispose d'aucune carte existante. Il est donc nécessaire, en plus de la localisation du robot de créer la carte au fur et à mesure. De nombreux travaux à ce sujet sont regroupés sous le terme SLAM (*Simultaneous Localization And Mapping*). Le principe de cette approche est bien expliqué dans le tutoriel en deux parties (Durrant-Whyte and Bailey, 2006) et (Bailey and Durrant-Whyte, 2006).

Les algorithmes de SLAM consistent à repérer des points au fur et à mesure du déplacement, à les trianguler pour les placer dans une même carte, puis à les utiliser pour se positionner dans la carte lors des poses suivantes. Ils sont en particulier beaucoup utilisés avec des capteurs de type télémètres laser ou quelquefois d'autres capteurs du même type tel que le radar (Vivet, 2011). Lorsqu'on utilise de tels capteurs donnant la profondeur, la position 3D des points est connue directement et permet donc d'enrichir la carte.

L'utilisation de la vision peut sembler plus complexe car un point doit être aperçu plusieurs fois pour être reconstruit en 3D. Cependant de nombreuses approches existent et sont résumées dans (Muhammad et al., 2009), qui les classe en fonction des différents types de caméras utilisées (caméra perspective, catadioptrique ou encore paire stéréo), des méthodes de mise en correspondance des points et des algorithmes de calcul de pose. Les algorithmes majoritairement utilisés pour le calcul de pose utilisent soit un filtre de Kalman étendu comme (Davison et al., 2007), soit un ajustement de faisceaux (Triggs et al., 2000).

Dans toutes ces approches la principale difficulté réside dans le fait que au fur et à mesure que le véhicule découvre de nouveaux endroits, il découvre de nouveaux point de repère, des amers, qui doivent être intégrés au processus de reconstruction. Ceci implique que plus la zone de déplacement du robot est importante plus le calcul devient complexe et rend l'algorithme difficile à exécuter dans un processus temps-réel.

Pour pallier ce défaut, il est possible d'utiliser des cartes locales permettant de limiter le nombre de points considérés lors du calcul de pose. Cela permet de diviser la reconstruction du monde en plusieurs sous-ensembles de taille réduite. La difficulté est de garder une cohérence entre les différentes sous-cartes, tout en maintenant à jours les éléments communs entre les cartes comme le font (Pinies and Tardos, 2008) et (Mei et al., 2011).

Une autre méthode permettant de limiter l'augmentation du temps de calcul lors de la localisation consiste à utiliser d'autres capteurs que la vision seule. Par exemple (Se et al., 2002) utilise des données de l'odomètre en plus de vision stéréo, ou encore (Weiss et al., 2011) parviennent à localiser un hélicoptère quadrirotor en corrigeant régulièrement la position fournie par l'algorithme de SLAM avec les données de la centrale inertielle embarquée.

Il est également intéressant de limiter le problème d'augmentation des cartes directement à la source, en limitant la quantité d'information stockée. Pour ce faire, deux méthodes généralement utilisées sont soit de filtrer les points, soit de limiter le nombre de poses stockées en utilisant une optimisation plus performante de la géométrie projective. Ainsi (Strasdat et al., 2010) comparent ces deux méthodes en concluant que, dans le cadre de leurs essais, utiliser une optimisation par ajustement de faisceaux en limitant seulement le nombre de poses de référence semble être le plus efficace. Le résultat est néanmoins nuancé car il ne prend en compte ni de très grandes trajectoires, ni le problème de fermeture de boucle qui permet d'optimiser la reconstruction globale.

Pour éviter que certains points peu fiables lors de la localisation soient ajoutés dans la carte, certains travaux récents tels que (Marquez-Gamez and Devy, 2012) parviennent avec une paire stéréo à différencier directement les objets mobiles, qui ne seront pas ajoutés à la carte, et les amers statiques qui enrichissent la carte. Cela permet de limiter l'accroissement de la taille de la carte en évitant que des amers mobiles inutiles soient ajoutés.

Toutes ces méthodes sont assez intéressantes pour l'exploration de nouvelles zones, mais restent néanmoins limitées à des distances réduites. Malgré les différents types de filtrage utilisés permettant de limiter l'accroissement du nombre de points, celui-ci continue à augmenter interdisant l'utilisation de telles méthodes sur de très grandes distances. De plus la construction de la carte réalisée au fur et à mesure contraint à tout exprimer dans un repère qui ne cesse de bouger car lié aux différentes positions du véhicule. Pour finir ce repère peut être difficile à recaler sur un repère déjà existant, tel que le système du GPS.

Apprentissage préalable

La construction d'une carte peut se faire également de manière totalement séparée de la partie localisation. Cela consiste généralement à piloter le véhicule manuellement, afin d'acquérir une séquence de référence. Puis en utilisant l'ensemble de la référence, les amers peuvent être calculés lors d'une phase hors ligne, ce qui permet d'utiliser des algorithmes plus complexes et de déterminer davantage de paramètres sur les amers. La carte obtenue peut également être recalée sur des données existantes, ou géoréférencée. Ensuite une phase de localisation en ligne

peut-être réalisée, consistant uniquement à comparer l'image courante avec les amers calculés pour localiser le robot au sein de la carte. Par ailleurs la carte étant entièrement construite avant la localisation, la taille des données est parfaitement maitrisée, et il n'y a plus de problèmes d'augmentation du temps de calcul ou de la charge mémoire au fur et à mesure du parcours.

Le travail présenté en (Royer et al., 2007) illustre ce principe, et a servi de point de départ aux travaux de cette thèse. Il s'agit de reconstruire initialement une pose de référence, et d'en sauvegarder uniquement les points d'intérêt reconnaissables, en l'occurrence des points détectés avec l'algorithme décrit par (Harris and Stephens, 1988), qui ont été suivis sur plusieurs images consécutives. Lors de la localisation, une détection des points est à nouveau utilisée sur l'image courante, et ceux-ci sont appariés à ceux de l'image de référence la plus proche. Connaissant la position 2D sur l'image des points de l'image de référence (dont la position 3D est également connue), il est donc possible d'en déduire la pose précise de la caméra.

En utilisant ce genre d'algorithme, il est possible de séparer complètement la localisation et la reconstruction et d'utiliser des capteurs différents pour les deux étapes. Par exemple (Cobzas et al., 2003) utilisent une caméra rotative et un télémètre pour avoir une meilleure reconstruction qui tient compte de la profondeur, alors que la localisation ne nécessite que de simples images d'une caméra. (Kidono et al., 2002) utilisent les mêmes capteurs pour les deux étapes, en l'occurrence une tête stéréo et un odomètre. La carte 3D ainsi réalisée facilite la localisation temps-réel.

D'autres travaux se sont concentrés essentiellement sur la génération de la carte sans se préoccuper dans l'immédiat de la localisation. C'est le cas de (Li and Tsuji, 1999) qui utilisent une caméra placée sur un véhicule observant le coté de la route. Ces vues permettent d'obtenir une cartographie des façades situées au bord de la route. Pour faire une reconstruction encore plus complète, (Craciun et al., 2010) utilisent plusieurs caméras et des télémètres laser pour reconstruire entièrement l'environnement 3D autour du point. (Meilland et al., 2011) parviennent également, en utilisant un ensemble de 6 caméras, à recréer un monde 3D sous forme de sphères virtuelles dans le but de pouvoir par la suite se localiser dans le monde réel en appariant les amers visuels avec le monde reconstruit. L'avantage des sphères virtuelles est de pouvoir se localiser dans le monde reconstruit quelle que soit l'orientation de la caméra, et donc le sens du mouvement.

Ces travaux permettent donc d'obtenir une représentation précise du monde et de se localiser à l'intérieur. Ces représentations peuvent également être géoréférencées, permettant alors de connaitre la localisation absolue du véhicule dans le monde, dans le même repère que, par exemple un GPS. L'intérêt peut être aussi de projeter ce monde sur une carte existante, et de définir de cette manière une référence à suivre simplement en traçant dans la carte la trajectoire voulue. Le véhicule étant ensuite capable de connaitre sa position dans la carte, pourra alors déterminer la commande à appliquer pour reproduire cette trajectoire et se déplacer de façon complètement autonome d'un point initial vers l'arrivée. Définir une trajectoire de telle sorte serait plus difficile dans le cadre d'un algorithme de type SLAM puisque la carte n'existant pas initialement, aucun repère ne peut être utilisé pour définir la trajectoire.

1.1.2 Stratégie basée sur les images

Dans certains cas, le seul objectif du robot consiste à reproduire à l'identique une trajectoire apprise précédemment. Dans ce cas, la localisation précise en terme de distance dans une carte globale n'est pas vraiment nécessaire. En effet, la donnée pertinente est uniquement la position relative par rapport à la trajectoire à suivre.

Au lieu de raisonner en terme de distance physique, il est donc possible de stocker directement la référence sous forme de séquence d'images vues lors de l'apprentissage. Lorsque le robot doit refaire la trajectoire de manière autonome il compare l'image de la caméra avec l'image de référence courante et en déduit la commande à appliquer pour s'en rapprocher. Lorsque l'image de référence est suffisamment proche de l'image courante, on utilise alors l'image suivante de la séquence de référence pour continuer son trajet. Ces stratégies sont souvent qualifiées de topologiques, car contrairement aux méthodes précédentes dites géométriques, elles ne se réfèrent pas aux mesures réelles du terrain.

La principale difficulté consiste à déterminer la commande à appliquer directement à partir des images. Ce problème, appelé asservissement visuel, utilise en général l'algorithme expliqué dans (Chaumette and Hutchinson, 2006) sous le terme de *image jacobian*. Il consiste à faire varier un ensemble de valeurs s mesurées directement sur l'image, vers des valeurs $s*$ de références, généralement les mêmes valeurs mesurées sur l'image de référence à atteindre. La principale différence entre les algorithmes basés sur les images réside dans le choix de ces paramètres s. Plusieurs travaux (Booij et al., 2007), (Goedeme et al., 2007) utilisent la géométrie épipolaire sur des caméras omnidirectionnelles pour déterminer le déplacement du robot. L'intérêt de la caméra omnidirectionnelle est que quelle que soit l'orientation du robot, les points de l'image de référence sont toujours visibles. (Becerra and Sagues, 2008) ont également montré que l'on pouvait utiliser la géométrie épipolaire pour l'asservissement avec des caméras conventionnelles. Finalement, dans (Becerra et al., 2010), les auteurs sont parvenus à utiliser le modèle unifié de caméra fournissant une méthode applicable à tous types de caméra. D'autres travaux (Diosi et al., 2007), (Cherubini and Chaumette, 2009), (Segvic et al., 2009) et (Lopez-Nicolas et al., 2007) combinent l'approche basée sur les images et celles basées sur la position. Ces approches utilisent toujours une séquence d'images de référence à suivre de manière topologique, mais pour se déplacer entre deux images de référence, le déplacement entre la position courante et la dernière position de référence est déterminée localement de manière géométrique.

Une comparaison a été réalisée par (Cherubini et al., 2009), montrant que ces méthodes basées sur les images sont généralement plus précises et plus rapides que les méthodes basées sur la position. De plus, la possibilité d'organiser la séquence d'images sous forme de graphe hiérarchique comme dans (Courbon et al., 2009) permet de localiser rapidement un véhicule et ce, même si sa position est totalement inconnue (cas du *kidnapped robot*). Cependant les véhicules sont contraints de suivre une référence enregistrée en joignant différentes images, et envisager un changement de trajectoire dû par exemple à un évitement d'obstacle est moins évident, nécessitant par exemple un système pour réorienter la caméra comme décrit dans (Cherubini and Chaumette, 2011).

1.1.3 Bilan des différentes stratégies

Pour conclure sur les différentes stratégies de localisation mises en œuvre, on s'aperçoit qu'il n'y a pas une bonne stratégie, mais plusieurs possibilités à choisir en fonction des besoins. Les approches topologiques ont l'avantage de pouvoir commander un véhicule qui reproduira précisément des trajectoires apprises pouvant s'étendre sur de grandes longueurs. Toutefois le véhicule ne pourra pas s'éloigner facilement de sa trajectoire pour éviter un obstacle par exemple. Les approches géométriques utilisant un apprentissage ont une plus grande souplesse, étant capables de gérer un éloignement de la trajectoire, mais en contrepartie d'une précision souvent plus faible.

Pour finir, lorsque le véhicule n'est pas conduit de manière automatique mais doit explorer une zone inconnue, les algorithmes de SLAM peuvent permettre de la cartographier, par exemple pour fournir un apprentissage à une localisation de type géométrique.

Toutes ces approches basées sur la vision présentent tout de même un point commun. Dans tous les cas, il est nécessaire de définir certains amers, généralement des points, détectés sur l'image courante, et de les mettre en correspondance avec des points de référence qui peuvent provenir, soit directement d'une carte (pour les approches de SLAM ou avec apprentissage), soit d'une image de référence (pour les approches topologiques). Pour cela des points d'intérêt sont généralement détectés, puis appariés entre différentes images.

De plus, dans chaque stratégie on peut diviser le problème en 2 parties. La première, la plus complexe est celle du *kidnapped robot*. Il s'agit de retrouver la position du robot (soit dans la carte, soit par rapport à une référence) sans aucun a priori sur la position. Ce cas se présente à l'initialisation, lorsque l'on a encore aucune idée sur la position initiale du robot, ainsi que dans les cas où la localisation a échoué, rendant la position totalement inconnue. Le traitement de ce genre de cas est généralement plus lourd à traiter, mais comme il ne se retrouve que peu souvent, cela n'empêche pas l'utilisation temps-réel de l'algorithme.

Le second problème est la localisation courante, où l'on peut légitimement faire l'hypothèse d'un faible mouvement du robot depuis la localisation précédente. On peut donc avoir une approximation de la pose à priori ; soit directement en prenant la même pose que précédemment, soit de manière plus évoluée en utilisant un modèle d'évolution du robot. Cet a priori peut permettre de restreindre à une petite zone de l'image la recherche d'appariements pour accélérer le processus et atteindre des performances temps-réel.

1.2 Mise en correspondance d'amers

La mise en correspondance d'images consiste, à partir de deux images d'une même scène, à identifier un ensemble de points sur une image et les retrouver sur l'autre image. Les différentes méthodes sont généralement basées sur des correspondances de points entre plusieurs vues qui permettent soit de connaitre sa position dans les approches géométriques, soit de déterminer la commande à réaliser pour les repositionner l'un sur l'autre dans les approches topologiques. La

figure 1.1 montre un exemple d'appariement réalisé sur deux images réelles d'une scène vue de points de vue différents.

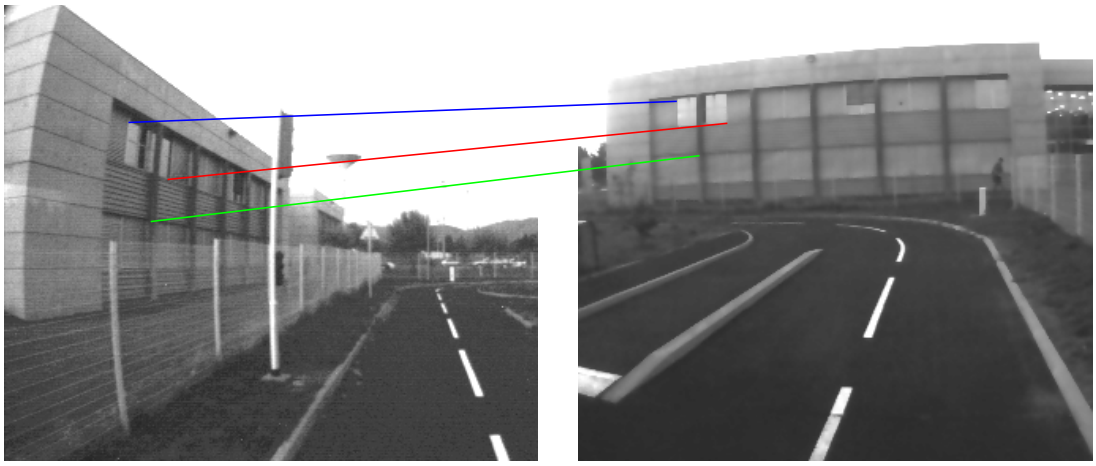


FIGURE 1.1 – Exemple d'appariements entre deux images d'une même fenêtre vue depuis deux points de vue bien différents

Afin d'avoir des correspondances utilisables, la technique d'appariement utilisée doit respecter du mieux possible deux contraintes :

- ➢ être insensible au changement de point de vue et de luminosité. Afin de pouvoir reconnaitre un amer même s'il est vu depuis des positions différentes et dans des conditions d'éclairage différentes.
- ➢ être discriminant pour ne pas apparier deux amers qui, bien que se ressemblant, ne correspondent pas aux mêmes éléments.

Il faut noter que la plupart du temps, un compromis entre ces deux contraintes doit être trouvé. En effet une méthode capable de retrouver un point malgré toutes sortes de changements en luminosité et de point de vue va vraisemblablement associer une fenêtre avec toutes les autres fenêtres d'une même façade, comme si dans l'exemple de la figure 1.1, les 3 points de l'image de gauche était associés aux 3 points de l'image de droite sans discrimination. A l'inverse, un algorithme très discriminant, qui n'apparie un point qu'avec exactement le même objet ne permettra pas de trouver un point si celui-ci a changé de point de vue ou de luminosité.

Dans tous les cas, la mise en correspondance se découpe en plusieurs étapes. Tout d'abord, une détection des points d'intérêt doit être réalisée, afin de trouver tous les points susceptibles d'être reconnus sur chaque image. Ensuite, un descripteur est calculé pour chacun des points ainsi détectés. Finalement, les descripteurs des différents points sont comparés fournissant, pour chaque couple de points, un score d'appariement. Un seuillage sur ce score permet de ne conserver que les meilleurs appariements, c'est-à-dire ceux qui ont le plus de chance de correspondre au même point.

Dans une première partie, nous allons détailler les détecteurs de points les plus courants. Dans une seconde partie les différents descripteurs et leur méthode de calcul de score seront comparés.

1.2.1 Détection des points

La détection des points est la première étape lors de la mise en correspondance. Elle consiste, pour une image, à trouver tous les points qui devraient être facilement reconnaissables lorsqu'on les observera de nouveau.

La principale difficulté est de garantir que, malgré les changements de point de vue ou de luminosité, les mêmes points seront détectés.

La plupart des détecteurs essaient de localiser des coins dans l'image, c'est-à-dire les endroits où les contours de l'image se croisent ou forment un angle.

Certains travaux se sont intéressés aux contours de l'image pour détecter les coins. Il s'agit de (Asada and Brady, 1986; Deriche and Giraudon, 1993), qui commence par détecter les contours puis réalise un seuillage pour en extraire les points d'intérêt. Une autre approche introduite par (Zhang and Zhao, 1995) puis reprise par (Dinesh and Guru, 2004) utilise des modèles de morphologie mathématique pour réaliser leur détecteur.

Mais la majorité des détecteurs sont basés sur la fonction d'intensité en chaque pixel de l'image. Les endroits où se trouvent un point d'intérêt peuvent en effet être vu comme les points où la norme du gradient de cette fonction est maximum. En se basant sur ce principe, les premiers détecteurs ont essayé de se baser sur la dérivée seconde d'une image. Plus précisément, (Beaudet, 1978; Dreschler and Nagel, 1982) passent par l'évaluation de la matrice hessienne d'une image pour détecter les points critiques correspondant aux coins, et d'autres (Kitchen and Rosenfeld, 1982; Zuniga and Haralick, 1983) utilisent une modélisation bicubique de l'image.

Une autre approche initiée par (Moravec, 1977) considère directement le gradient de l'image et essaie d'en trouver le maximum. Ce travail a ensuite été amélioré (Harris and Stephens, 1988) pour réaliser le détecteur de Harris, qui est encore actuellement un détecteur parmi les plus utilisés. Il est basé sur l'évaluation des courbures principales de la fonction d'autocorrélation locale de l'image. Lorsque en un point, les courbures dans deux directions différentes sont élevées, le point considéré correspond à un coin. Un avantage de cet algorithme est que en tout point de l'image, il donne un score de détection, les points ayant les meilleurs scores étant les coins les plus marqués. Il est alors possible de restreindre le nombre de détection de plusieurs manière. Un seuillage peut être réalisé directement sur le score, pour obtenir un nombre variable de détection mais de bonne qualité. Ou bien le nombre de points détecté peut être choisi et de cette manière, seuls les points ayant un score le plus fort seront conservés, leur nombre étant toujours le même. Cette dernière méthode peut également être modifiée en fixant un nombre de points par zone de l'image, permettant ainsi de garantir une répartition homogène des points de l'image. C'est sans doute cette flexibilité dans le tri du résultat qui fait le succès de ce détecteur.

Ces premiers détecteurs, bien que déjà assez performants pour détecter des coins, ne sont pas fondamentalement invariants aux changements d'échelle ou au changement affine. De plus

certains détecteurs essaient de fournir, en plus de la position des différents points une estimation de leur facteur d'échelle, permettant d'évaluer à quelle échelle le point correspond à un coin. Ainsi, en supposant que si le point est revu par exemple à une distance deux fois plus grande, la détection fournira un facteur d'échelle double. La détermination de ce facteur d'échelle repose sur le principe que changer l'échelle d'une image revient à la convoluer par un filtre gaussien. Suivant ce principe (Lowe, 2004) a développé le DoG (*Difference of Gaussian*) qui détecte des coins de manière invariante au changement d'échelle. De même (Mikolajczyk and Schmid, 2001) en utilisant le Laplacien, améliore le détecteur de Harris pour le rendre multi-échelle sous le nom de Harris-Laplace. Le principal défaut de ces détecteurs, est que les multiples filtrages gaussiens pour vérifier de multiples échelles rendent les calculs plus lourds. Le Fast-hessian (Bay et al., 2008) permet de diminuer ces temps de calcul grâce à une approximation du filtrage gaussien.

Afin d'augmenter la robustesse des détecteurs aux transformations affines et projectives, rencontrées particulièrement lors des changement de point de vue, (Mikolajczyk and Schmid, 2004) ont développé deux détecteurs supplémentaires, le Harris-affine et Hessian-affine.

En réaction à ces détecteurs de plus en plus couteux en calcul, (Rosten et al., 2010) a développé récemment un détecteur basé directement sur l'intensité de l'image qui, bien que pas intrinsèquement invariant aux différentes transformations, permet de détecter très rapidement les coins de l'image.

1.2.2 Appariement d'amer locaux

Pour la mise en correspondance, il est nécessaire, après avoir détecté les points, de pouvoir déterminer quels points sont identiques (appelé également points homologues) d'une image à l'autre. Pour cela les méthodes existantes sont basées sur la définition d'un *descripteur* local, c'est-à-dire un ensemble de valeurs représentatives du point et de son voisinage. La comparaison de deux points P_1 et P_2 provenant de deux images I_1 et I_2 différentes, se fait en utilisant un algorithme (par exemple la distance euclidienne) pour mesurer l'écart entre les valeurs de leur descripteur. Les points sont homologues lorsque leurs descripteurs sont proches. Comme pour les détecteurs, la principale difficulté est de pouvoir retrouver un point, même lorsqu'il y a eu un changement de point de vue ou de luminosité.

Par exemple un descripteur local des plus basique pourrait être l'ensemble des valeurs d'intensité de chaque pixel du voisinage du point. Lorsque l'on compare deux points, on peut simplement faire la somme des carrés des différences d'intensité de chaque pixel. Plus cette valeur est faible, plus les points peuvent être considérés comme identiques. Ce descripteur serait toutefois assez peu performant, car le moindre changement de luminosité, ou même un peu de bruit sur l'image, fausserait la comparaison et deux points pourtant identiques auraient un mauvais score (une valeur assez élevée).

Une méthode d'appariement un peu plus évoluée et moins sensible au changement linéaire d'intensité, consiste à prendre comme descripteur le voisinage d'un point mais de lui enlever sa moyenne et de le diviser par l'écart-type des niveaux de gris sur ce voisinage. Pour comparer

les descripteurs, on réalise la somme des produits des descripteurs. On obtient alors lors de la comparaison l'équation 1.1 correspondant à une corrélation centrée normée (ZNCC).

$$ZNCC(P_1, P_2) = \sum_{d \in V} \frac{I_1(P_1 + d) - \overline{I_1}(P_1)}{\sqrt{\sum_{d \in V}(I_1(P_1 + d) - \overline{I_1}(P_1))^2}} \times \frac{I_2(P_2 + d) - \overline{I_2}(P_2)}{\sqrt{\sum_{d \in V}(I_2(P_2 + d) - \overline{I_2}(P_2))^2}} \tag{1.1}$$

$$\text{avec} \begin{cases} V = \{-N, ..., N\} \times \{-N, ..., N\} \text{ le voisinage du point considéré} \\ \overline{I_i}(P_i) = \dfrac{1}{|V|} \displaystyle\sum_{d \in V} I_i(P_i + d) \text{ où } i \in \{1, 2\} \text{ la moyenne du voisinage du point } P_i \end{cases}$$

Cette méthode est invariante en cas de changement affine de luminosité, mais reste très sensible à tout autre type de transformation telle que les translations, rotations, changements d'échelle, changements de point de vue.

Descripteurs robustes aux transformations géométriques

De nombreuses autres méthodes basées sur des descripteurs ont été développées pour rendre ces comparaisons robustes à toutes formes de transformation. Les premiers travaux en ce sens utilisent le moment mathématique (Hu, 1962) ou encore les histogrammes d'intensité (Schiele and Waibel, 1995). Ces deux approches, initialement utilisées sur une image globale, peuvent être restreintes au voisinage d'un point d'intérêt et donc devenir locales. Leurs propriétés les rendent robustes aux changements par rotation et translation. Toutefois, elles restent sensibles aux déformations de type changement d'échelle ou aux transformations affines provoquées par un changement de point de vue. Un autre descripteur beaucoup plus robuste à ces transformations est le très célèbre SIFT (Lowe, 2004). Celui-ci utilise un histogramme de l'orientation des gradients autour du point d'intérêt, couplé à une approche multi-échelle. Il est présenté avec le détecteur DoG décrit précédemment. L'auteur a démontré que ce descripteur est robuste naturellement au changement affine de luminosité, aux rotations, translations et changements d'échelle et à des transformations affines modérées. Pour améliorer la robustesse de ce descripteur (Morel and Yu, 2009) utilisent un modèle de transformation affine. Les résultats montrent une amélioration, néanmoins le cout de calcul déjà non négligeable du SIFT est encore augmenté d'un facteur 5, le rendant inutilisable dans des applications temps-réel. Pour accélérer le calcul des descripteurs SIFT, (Tola et al., 2008) a utilisé un masque circulaire pour définir le voisinage des points. Un autre descripteur très connu est le SURF (Bay et al., 2008), lui aussi est basée sur le SIFT mais utilise la décomposition en ondelettes pour améliorer significativement le temps de calcul tout en conservant des résultats similaires. Plus récemment un descripteur binaire (Calonder et al., 2010) offrant des résultats comparables au SURF en étant plus rapide a été présenté. Ces premiers descripteurs ont été comparés dans plusieurs situations par (Mikolajczyk and Schmid, 2003) en jugeant lesquels sont les plus robustes aux différentes transformations.

Une autre approche pour réaliser des descripteurs invariants aux changements de point de vue est d'utiliser d'autres informations telles que la carte des profondeurs. L'objectif devient alors de recaler non plus des images mais bien des scènes 3D entre elles. (Koser and Koch, 2007) en particulier reconstitue une vue en *ortho-perspective* afin de recaler des scènes 3D dans un contexte de stéréo vision. De la même manière (Wu et al., 2008) décrivent un modèle de patch invariant au point de vue, permettant de reconnaitre une scène 3D.

Le point commun à toutes ces méthodes est que le descripteur d'un point est généré à partir d'une seule image. C'est donc cette unique vue qui est utilisée pour reconnaitre le point observé depuis d'autres points de vue. Ce type d'approche a été initialement créé pour réaliser des recherches de similarité entre deux images issues d'une base de données sans a priori sur le lien entre elles. Cependant, dans le cadre d'une approche de localisation, la plupart des points recherchés ont été observés depuis différents points de vue. Il semble donc dommage de se limiter à une seule vue d'un point pour essayer de le reconnaitre dans la vue courante. L'idée est donc de calculer un descripteur en se basant sur plusieurs images déjà apprises du point observé. Dans cette optique, (Lepetit et al., 2005) utilisent un classifieur pour initialiser un descripteur de point. Reconnaitre le point revient donc à le classer en tant que identique ou non. Une autre approche utilisant un classifieur suivant le même principe est proposé par (Williams et al., 2007). L'originalité de ces travaux est que, au lieu d'utiliser plusieurs observations du même point, les images de différents points de vue sont générées en appliquant une transformation homographique sur l'image d'origine. Ces approches permettent de définir un descripteur indépendant aux changements de point de vue, mais en utilisant de multiples vues du même point.

Descripteurs utilisant une modélisation 3D

Une dernière approche consiste non pas à réaliser un descripteur indépendant du point de vue, mais à utiliser une modélisation 3D de l'amer observé, pour adapter le descripteur à la vue courante. Cela n'est possible que lorsqu'on utilise une stratégie de localisation basée sur la position, où l'on peut prédire la position courante du véhicule et à ce titre, prédire l'image des données 3D que l'on a modélisées. L'avantage de cette méthode est que la modélisation du point recherché ayant été projetée dans la vue courante, on a moins de chance de le confondre avec un objet identique qui serait placé ailleurs. Par exemple lorsque l'on regarde une façade dont toutes les fenêtres sont identiques comme sur la figure 1.2. Un descripteur indépendant du point de vue décrirait toutes les fenêtres (numérotées 1 à 3) de la même manière. Lors de l'appariement, on pourrait donc facilement les confondre (et apparier par exemple la fenêtre 3 de la première image avec la numéro 1 de la seconde image). Tandis que, si l'on connait approximativement la position 3D de la fenêtre par rapport à la caméra, on peut alors projeter l'apparence de la fenêtre tel qu'elle apparait, et la différencier des autres qui, étant situées à une position différente, ont une apparence déformée. Ainsi en recherchant la projection de la fenêtre 3 dans la seconde image, il y a peu de chance de la confondre avec les deux autres.

Cette approche nécessite une modélisation 3D des amers observés pour les reprojeter dans la vue courante. Dans la plupart de ces approches les amers sont modélisés par des surfaces planes,

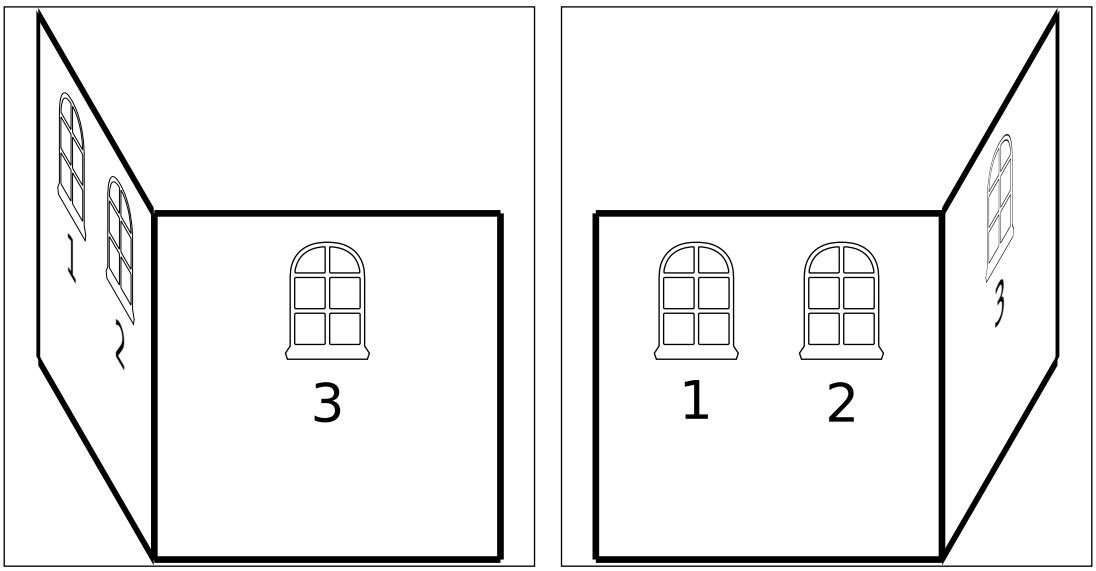


FIGURE 1.2 – Exemple de deux vues observant une maison avec plusieurs fenêtres

hypothèse généralement vraie tant que l'on reste dans un voisinage local. (Molton et al., 2004) ont été les premiers à appliquer cette modélisation dans une utilisation en SLAM. Les amers ne sont donc plus des simples points, mais des patchs, c'est-à-dire des élément de surfaces planes dont l'orientation est calculée à partir de l'apparence du point. Au fur et à mesure du déplacement, et donc du nombre de points de vue depuis lesquels un même point est observé, cette orientation est mise à jour et devient de plus en plus proche de l'orientation réelle du patch. La connaissance de cette orientation permet alors de transformer la première vue du patch pour correspondre à la vue de l'image courante. L'image courante de son côté est bruitée par un filtrage gaussien afin de compenser le flou créé par la déformation sur la première image. La comparaison ainsi réalisée permet de suivre un même point sur de plus grandes distances, en particulier lorsque la caméra s'éloigne de la position où ont été observés initialement les points.

Cette première approche utilise donc les orientations des normales pour améliorer l'appariement entre les points. Cependant le calcul des poses de caméra et la mise à jour de la carte sont toujours basés sur la position des points comme lors des méthodes plus classiques. (Pietzsch, 2008) utilise l'orientation calculée non seulement pour améliorer l'appariement mais l'intègre également dans le filtre de Kalman qui optimise aussi les poses de caméra et la position des points. Ces résultats montrent une nette amélioration de la précision de localisation dans ce contexte.

Une autre utilisation de surfaces localement planes a été réalisée par (Berger and Lacroix, 2008), toujours pour une méthode de SLAM mais en utilisant une paire stéréo. Dans ce cas l'orientation des patchs (appelé *facet* dans ces travaux) est calculée directement avec chaque paire d'image. Il est alors possible de redresser l'image du patch sous la forme d'une vue fronto-parallèle qui est orientée en utilisant le gradient de l'image. Les patchs peuvent ainsi être générés

pour chaque paire d'image, et la mise en correspondance d'amer ne se fait plus entre points mais entre patchs. Le calcul d'orientation étant normalement toujours le même les textures des patchs peuvent être comparées directement beaucoup plus facilement. De plus, lorsque la position peut être prédite, les patchs peuvent également être reprojetés, comme dans les méthodes précédentes pour faciliter l'appariement et la génération des patchs.

Tous ces travaux sont réalisés dans un contexte de SLAM, et l'orientation des patchs (initialisée à partir d'une ou deux poses) est réajustée au fur et à mesure. Dans nos travaux, l'objectif est de déterminer un descripteur à partir de plusieurs vues initiales de la séquence d'apprentissage, puis de le reprojeter pour s'adapter à la vue courante. En effet, dans une approche basée position mais avec un apprentissage, le calcul de la normale des patchs peut être réalisé indépendamment dans une phase hors-ligne. Par la suite la localisation se fait en anticipant sur la déformation que pourra avoir ce patch. Utiliser cette déformation permet d'avoir la possibilité de grandement s'éloigner des points de vue initiaux sans risquer de le confondre avec d'autres amers qui lui ressemblent.

1.3 Algorithme de localisation initial

Nos travaux ont pour objectif d'améliorer les algorithmes de localisation existant. Il n'est cependant pas utile de reprendre l'ensemble de l'algorithme de reconstruction et de localisation. L'algorithme développé auparavant par Éric Royer a été utilisé comme point de départ dans ces travaux. Cette partie reprend donc l'ensemble de cet algorithme qui a servi de base au développement du système décrit par la suite. Cependant, davantage de détails pourront être obtenus à ce propos dans (Royer, 2006).

L'algorithme de localisation utilise une stratégie basée sur la position avec un apprentissage, similaire aux travaux décrits dans la partie 1.1.1. Il consiste à réaliser un premier passage avec le véhicule, afin d'acquérir une séquence d'images d'apprentissage. Cette séquence d'images est analysée lors d'une phase hors-ligne afin de déterminer les poses des caméras et les positions des points de repère qui formeront la carte. Cette partie sera détaillée en 1.3.1. Ensuite lors de la localisation, le véhicule est piloté de manière autonome sur la même trajectoire, et utilise la carte pour déterminer sa position et en déduire le mouvement à réaliser. Cette partie sera décrite en 1.3.2.

1.3.1 Reconstruction

Après acquisition de la séquence d'apprentissage, une carte 3D associée à un seul repère monde est construite. Cette étape de cartographie est connue dans le domaine de vision par ordinateur sous le terme de *Structure from Motion*. De nombreux travaux ont été réalisés en ce sens, tel que (Beardsley et al., 1996), (Nistér, 2000) ou encore (Lhuillier and Quan, 2005). Ces algorithmes ont pour but de modéliser un environnement 3D texturé complet en se basant uniquement sur la séquence vidéo initiale.

Dans le cas de l'algorithme utilisé, seule les poses des images prises par les caméras ainsi que la position de tous les amers repérés sur cette image sont nécessaires.

Afin d'éviter d'être trop lourd en calcul, et d'avoir des calculs mal conditionnés, seulement quelques images seront sélectionnées tout au long de la trajectoire. Ces images, nommées images clef, seront suffisamment distantes pour garantir qu'un déplacement relativement important a été fait entre deux images clef successives. C'est la pose associée à ces images et les points qui y sont détectés, qui seront reconstruits. Par la suite, les autres images seront localisées en se basant sur la reconstruction initiale. L'algorithme se décompose donc en 3 parties :

1. Sélection d'un ensemble d'images clef
2. Calcul de la pose des images-clef et du nuage de points associé
3. Localisation des images non clef

Sélection des images clef

L'objectif de cette sélection est d'avoir une assez grande distance entre deux images successives. En effet le calcul de la géométrie épipolaire est mal conditionné lorsque deux images sont trop proches. Cette sélection peut se faire en utilisant l'odométrie du véhicule, ou bien un GPS et en prenant par exemple une image à chaque mètre d'avancement. Cependant les images clef doivent tout de même permettre de faire des mises en correspondance entre les images. Or l'apparence peut dans certains cas varier beaucoup, particulièrement dans les virages. Il est donc nécessaire de garantir un certain nombre d'amers en commun entre les images successives. La solution retenue n'utilise que les images. Elle consiste à garder une image la plus éloignée possible telle qu'il reste au moins N points communs avec la précédente et au moins M points communs avec les deux images précédentes. Cette heuristique n'est peut-être pas optimale, mais elle garantit un nombre de correspondances suffisant pour faire la reconstruction dans de bonnes conditions.

Calcul de pose

Avec les points suivis sur plusieurs images successives, il devient possible de reconstituer les poses de caméra et les nuages de points 3D observés. Le problème principal est de parvenir à déterminer *simultanément* la pose des caméras et les coordonnées des points. En effet la connaissance du nuage de points permet de déterminer la position d'une caméra, par exemple avec la méthode de Grunert expliquée dans (Haralick et al., 1994). De même trianguler un nuage de points observé par des caméras dont les poses sont connues est un problème résolu, détaillé en particulier en (Hartley and Zisserman, 2004). Afin de résoudre tout le problème simultanément, le calcul de pose se fait de manière incrémentale. La pose de chaque caméra supplémentaire est calculée à partir du nuage de points 3D triangulés précédemment. Puis les nouveaux points 2D vus à la fois depuis la nouvelle pose et les précédentes sont triangulés pour enrichir le nuage de points existants.

La principale difficulté réside dans le calcul des 3 premières poses qui doit se faire avant d'avoir triangulé le premier nuage de points. Un algorithme de type RANSAC (Fischler and Bolles, 1981) utilisant un ensemble de 5 points pour retrouver le déplacement entre les caméras, permet d'avoir une approximation de ces premières poses. Ce calcul est basé sur l'estimation de la matrice essentielle (Nistér, 2004) entre la première et la troisième vue. La matrice essentielle permet ensuite de connaitre le déplacement entre les deux caméras et de trianguler les points appariés entre ces vues. La pose de la seconde caméra est ensuite calculée en utilisant ces points. Les points dont l'erreur de reprojection est assez faible sont considérés comme inliers. Le calcul qui maximise le nombre d'inliers est retenu. On obtient alors une pose initiale pour les trois premières vues qui permettra d'initialiser le calcul incrémental.

A chaque nouvelle pose calculée, un ajustement de faisceaux est réalisé localement pour fusionner la pose avec les caméras calculées précédemment. Un ajustement de faisceaux général est également effectué à la fin sur l'ensemble des poses calculées pour assurer une bonne cohérence de la localisation.

Localisation des images non clef

Après avoir déterminé les poses des caméras clef et les coordonnées du nuage de points 3D, retrouver la position des images non clef est similaire à la localisation. L'algorithme est donc le même et sera détaillé dans la section 1.3.2.

La reconstruction ainsi faite permet d'obtenir les position et orientations prise par la caméra lors de la séquence d'apprentissage et un nuage de point 3D associés. On peut voir un exemple de reconstruction en vue de dessus sur la figure 1.3

1.3.2 Localisation

Une fois que les poses des images clef sont sélectionnées, et que le nuage de points 3D est estimé et associé aux points 2D détectés, toutes les informations nécessaires à la localisation sont réunies. Une seule image suffira alors pour déterminer la position courante de la caméra dans le repère du monde. Pour cela l'algorithme se décompose en 3 étapes successives :

1. la prédiction de la pose,
2. l'appariement des points,
3. le calcul de pose.

Ces différentes étapes vont être détaillées.

Prédiction

Tout d'abord, un modèle de déplacement est utilisé afin d'avoir une pose approchée du véhicule. Ce modèle peut être très simple, par exemple on peut prendre directement la pose précédente. Le temps entre deux images successives étant assez faible par rapport à la vitesse

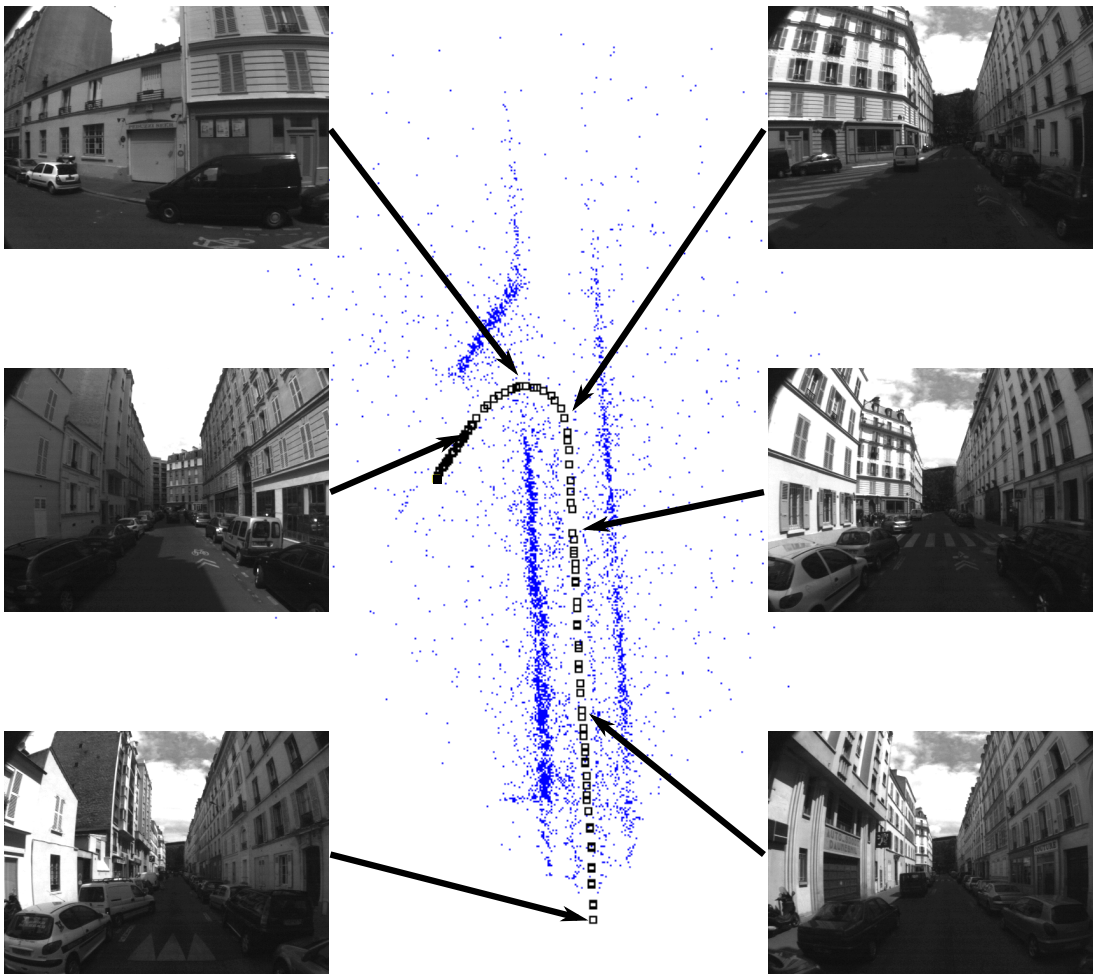


FIGURE 1.3 – Exemple de reconstruction d'une rue à partir d'une séquence d'image.

de déplacement, l'erreur commise n'est pas très importante. Il est également possible d'utiliser un modèle d'évolution du robot et de considérer que le mouvement étant continu, le robot s'est déplacé avec la même vitesse angulaire et linéaire que précédemment. Cela permet d'avoir une approximation généralement plus précise. Dans tous les cas, il est également intéressant d'évaluer l'incertitude sur la position. Plus l'incertitude sur la position sera grande, plus les appariements devront être cherchés dans une zone importante de l'image courante, et l'étape demandera donc plus de calcul.

Lors de la pose initiale du véhicule, lorsque aucune pose n'est encore connue, on utilise alors directement chaque pose clef comme prédiction pour réaliser la localisation sera réalisé. On obtient donc, pour chaque image-clef, une pose calculée grâce à un certain nombre d'appariement. La pose conservée est celle ayant trouvé le plus d'appariement, et permet donc d'initialiser le modèle de prédiction.

Appariement

L'étape d'appariement consiste à rechercher dans l'image courante l'ensemble des points présents dans la carte. Pour cela, la première étape consiste à utiliser un détecteur de points d'intérêt tel que ceux décrit dans la partie 1.2.1. Cependant, s'il fallait comparer tous les points détectés avec tous les points de la carte, l'étape serait particulièrement lourde en calcul. Afin de réduire ce temps de calcul, on commence tout d'abord par sélectionner l'image-clef la plus proche. La prédiction réalisée à l'étape précédente permet de réaliser cette sélection. De plus, au lieu de chercher parmi toutes les images clef, on peut se limiter à celles situées autour de l'image clef sélectionnée précédemment, cela permet d'accélérer la recherche et évite, lorsque la trajectoire d'apprentissage forme une boucle, de suivre une pose qui a été vue dans une autre portion de la trajectoire.

Une fois l'image clef la plus proche repérée, seuls les points qui étaient visibles sur cette image seront comparés aux points détectés sur l'image courante. De plus, grâce à la prédiction de la pose courante, on peut projeter directement ces points sur l'image analysée. L'incertitude associée à la prédiction permet également d'avoir une ellipsoïde de confiance autour de la projection, et définit ainsi une région d'intérêt (ROI) où doit se retrouver le point concerné. En se basant sur l'image des points vus sur l'image clef et sur le voisinage des points détectés à l'intérieur de la zone, on peut calculer un score d'appariement (en utilisant les méthodes de la partie 1.2.2) et après seuillage, déterminer où sont situés précisément ces mêmes points dans l'image courante. Dans le cadre des travaux précédents, la détection des points était faite avec le détecteur de Harris, et l'appariement utilisait une ZNCC. Cependant il est toujours possible d'utiliser d'autres méthodes pour la détection ou l'appariement sans changer l'algorithme.

Après avoir obtenu ces paires de points 2D, on a donc une correspondance entre les points 3D qui étaient sur l'image-clef et les positions 2D de l'image courante où ils ont été retrouvés. C'est cet ensemble de paires qui est utilisé par le calcul de pose.

Calcul de pose

Pour calculer la pose de manière précise, on utilise une méthode d'optimisation itérative pour minimiser l'erreur de reprojection de tous les appariements correct. Cette méthode décrite dans (Araujo et al., 1998), utilise la méthode de Newton-Raphson pour converger vers le minimum. La seule différence avec l'algorithme original est la gestion des faux-appariements, qui dans notre approche sont recalculés à chaque itération et n'interviennent donc pas dans l'évaluation de l'erreur de reprojection.

1.3.3 Bilan

Cette méthode de reconstruction et localisation a fait ses preuves et permet de commander avec succès un robot qui suit une trajectoire apprise. Cependant la localisation reste très dépendante de la trajectoire d'apprentissage et lorsque le robot s'éloigne de celle-ci, par exemple

pour éviter un obstacle, il devient difficile d'apparier les points avec l'image-clef la plus proche, rendant la localisation peu fiable voire impossible lors d'importants écarts. L'utilisation de descripteurs indépendants du point de vue pourrait sembler profitable, mais cela alourdirait le calcul et risquerait de causer de mauvais appariements lorsque de nombreux amers ont la même apparence (ce qui est assez fréquent dans un milieu urbain). Afin d'améliorer la localisation, il semble intéressant de tirer partie d'une connaissance tridimensionnelle des amers. Nous allons montrer que l'usage de patchs permet d'améliorer la situation. De plus, la génération de la carte pouvant être réalisée hors ligne, il est possible de générer plus finement des amers adaptables au fur et à mesure du déplacement. C'est en suivant cet objectif que les travaux décrits au chapitre suivant ont été réalisés.

Chapitre 2

Utilisation de Patchs-plan

2.1 Principe général

Comme on a pu le voir précédemment, la mise en correspondance de points entre deux images nécessite un compromis entre une invariance à de nombreux points de vue, ou une discrimination plus forte. Dans le cadre de la localisation d'un robot, si l'on veut s'éloigner de la trajectoire d'apprentissage, il est pourtant essentiel de pouvoir retrouver les amers appris depuis d'autres points de vue, sans pour autant les confondre entre eux. Par ailleurs, les descripteurs invariants ne tirent partie d'aucun a priori sur la pose de la caméra. Pourtant lorsque l'on commande un véhicule autonome, la position courante du véhicule n'est pas totalement aléatoire, et il est possible de prédire sa position.

En s'intéressant aux algorithmes de SLAM, il a toutefois été montré que des amers pouvaient être suivis sur des distances plus grandes en utilisant des patchs. Ces patchs-plan, décrit dans la partie 1.2.2 sont en réalité la représentation du voisinage d'un point sous la forme d'une zone localement plane dont l'orientation et la texture sont connues. Dans les algorithmes de SLAM, ils sont mis à jour au fur et à mesure de la trajectoire.

Tout comme dans le cadre d'un algorithme de SLAM, il est possible dans notre approche de prédire la position courante. L'utilisation d'une modélisation sous forme de patchs dans notre approche semble donc intéressante puisque leur apparence peut être adaptée à différentes positions du véhicule. Ainsi un amer pourrait être retrouvé malgré d'importantes variations de point de vue, sans pour autant le confondre avec d'autres, puisque qu'aucune invariance aux transformations géométriques ne sera nécessaire. Contrairement aux algorithmes de SLAM, les patchs ne seront néanmoins pas mis à jour au fur et à mesure puisque, dans le cadre de la navigation autonome, le carte est supposée complète au départ, pour permettre de prédire la trajectoire à suivre.

Nos travaux consistent donc à modéliser les amers obtenus par apprentissage sous la forme de patchs-plan, puis à les utiliser lors de la localisation. Après une partie reprenant les notations et modèles de caméra utilisés, la génération de patchs sera détaillée. La partie suivante détaille

l'utilisation des patchs générés lors de la localisation. Pour finir une première expérimentation comparant les performances des patchs avec celles du descripteur SIFT sera présentée.

2.2 Modèle de caméra utilisé

Pour comprendre le principe de projection des patchs et leur création, il est indispensable de décrire le modèle de caméra utilisé et la modélisation du processus de formation d'une images. De plus, selon les ouvrages consultés, différentes conventions pour les projections des points peuvent être utilisées. Cette partie va donc décrire la modélisation des caméras utilisée et introduire les notations qui seront utilisées dans la suite.

Dans le cadre de nos travaux, on considère que les caméras suivent le modèle sténopé, représenté en figure 2.1.

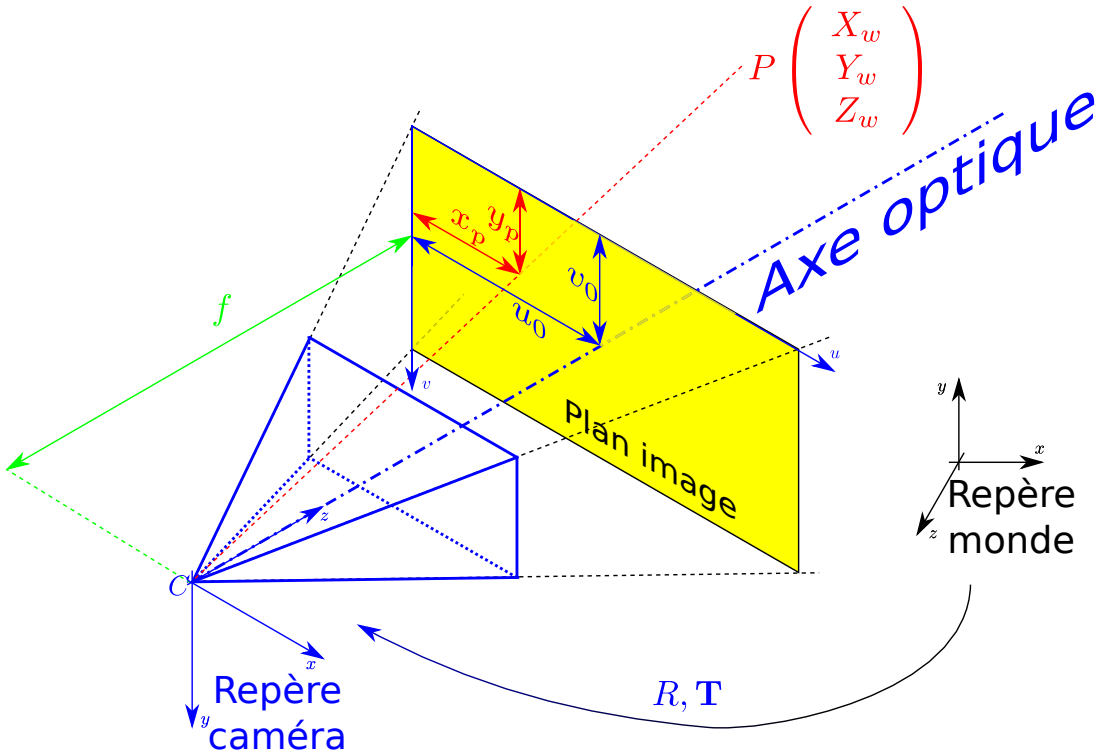


FIGURE 2.1 – Modèle utilisé pour la caméra

2.2.1 Projection des points

Ce modèle considère que tous les points P situés devant l'objectif sont projetés dans le plan image en suivant la droite (PC) où P est le point 3D considéré et $\mathbf{C}$ le centre de la caméra.

Repère monde

Pour retrouver les coordonnées du point dans l'image, on a besoin de connaitre la pose de la caméra. Pour ce faire, on considère un repère absolu, lié au monde, qui servira de repère à la carte générée. Dans ce repère, le point P a pour coordonnées $\mathbf{P_w} = \begin{bmatrix} X_w & Y_w & Z_w \end{bmatrix}^T$. On détermine alors la pose de la caméra sous forme d'un vecteur $\mathbf{T} = \begin{bmatrix} T_x & T_y & T_z \end{bmatrix}^T$ qui correspond aux coordonnées du centre $\mathbf{C}$ de la caméra exprimée dans le repère monde et une matrice R, matrice de rotation donnant l'attitude de la caméra.

Repère caméra

Il permet de définir un repère lié à la caméra comme indiqué sur la figure 2.1, où l'origine est à la position de la caméra, l'axe z est dirigé vers l'avant suivant l'axe optique de la caméra, l'axe y est orienté verticalement vers le bas de la caméra, et l'axe x vers la droite. On peut alors obtenir les coordonnées $\mathbf{P_c}$ du point P exprimées dans le repère caméra avec l'équation 2.1

$$\mathbf{P_c} = R(\mathbf{P_w} - \mathbf{T}) \tag{2.1}$$

Ce qui devient, en passant en coordonnées homogène :

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} R & -R\mathbf{T} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{2.2}$$

Repère image

On obtient ensuite les coordonnées du point dans l'image en utilisant les équations de projection 2.3 (le signe $\equiv$ signifiant une égalité à un facteur multiplicatif près)

$$\begin{cases} x_i = X_c/Z_c \\ y_i = Y_c/Z_c \end{cases} \Leftrightarrow \begin{bmatrix} sx_i \\ sy_i \\ s \end{bmatrix} \equiv \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \tag{2.3}$$

avec s un nombre réel quelconque.

Paramètres intrinsèques

Pour déterminer les coordonnées en nombre de pixels, il est nécessaire de connaitre la matrice K des paramètres intrinsèques de la caméra, exprimée selon l'équation 2.4.

$$K = \begin{bmatrix} f/dx & 0 & u_0 \\ 0 & f/dy & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

avec $\begin{cases} f \text{ la distance focale de l'objectif} \\ dx, dy \text{ respectivement la largeur et hauteur d'un pixel (dans la même unité que f)} \\ u_0, v_0 \text{ les coordonnées du point principal [1] en pixel} \end{cases}$

Il est également d'usage, pour simplifier l'écriture de K, de poser $f_x = f/dx$ pour la focale en x et $f_y = f/dy$ pour la focale en y. On peut remarquer que lorsque les pixels sont carrés, ce qui est généralement le cas, le rapport dx/dy est proche de 1, et par conséquent la focale en x et en y est identique à la focale f exprimée en pixel.

Les coordonnées en pixels du point sont finalement obtenues avec :

$$\begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} = K \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{2.5}$$

.

Bilan

L'ensemble de ces opérations peut être exprimé avec l'équation 2.6

$$\begin{aligned} \begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} &\equiv K \begin{bmatrix} R & -R\mathbf{T} \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \\ \begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} &\equiv \text{Proj} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \end{aligned} \tag{2.6}$$

avec Proj la matrice de projection de taille 3×4 associée à la pose de la caméra .

L'ensemble des valeurs précédentes est résumé dans le tableau 2.1.

Tout cela permet de retrouver, théoriquement, la position d'un point dans l'image. En pratique, on constate quelques erreurs dans ce modèle, particulièrement pour les caméras ayant un grand angle de vue. En effet, dans ce genre d'image, l'objectif provoque une déformation des images qu'il faut également modéliser.

2.2.2 Modélisation de la distorsion

On peut modéliser la déformation de l'objectif en utilisant un modèle de distorsion radiale exploitant un polynôme de degré 5. Le polynôme, de coefficients $(a_1, a_2, a_3, a_4, a_5)$ permet de connaitre la modification de la distance entre le point principal et la projection d'un point sur le

1. Le point principal correspond à l'intersection entre l'axe optique et le plan image

Paramètres intrinsèques de la caméra	
f_x, f_y	Focale en x et y
u_0, v_0	coordonnée du point principal (projection du centre optique)
dx, dy	taille d'un pixel de l'image
K	Matrice des paramètres intrinsèques
Paramètres extrinsèques de la caméra	
R	Matrice de rotation donnant l'orientation de la caméra
$\mathbf{T} = \begin{bmatrix} T_x & T_y & T_z \end{bmatrix}^T$	coordonnée du centre de la caméra dans le repère monde
Proj $= K \begin{bmatrix} R & -R\mathbf{T} \end{bmatrix}$	matrice de projection de la pose de la caméra
Coordonnées du point P	
$\mathbf{P_w} = \begin{bmatrix} X_w & Y_w & Z_w \end{bmatrix}^T$	Coordonnées dans le repère monde
$\mathbf{P_c} = \begin{bmatrix} X_c & Y_c & Z_c \end{bmatrix}^T$	Coordonnées dans le repère caméra
$\mathbf{P_i} = \begin{bmatrix} x_i & y_i \end{bmatrix}^T$	Coordonnées de la projection du point dans le plan image
$\mathbf{P_p} = \begin{bmatrix} x_p & y_p \end{bmatrix}^T$	Coordonnées de la projection du point dans l'image (en pixel)

TABLE 2.1 – Notation utilisée pour les caméras et les points

plan image. Concrètement, pour obtenir une image sans distorsion, il est nécessaire d'appliquer la méthode suivante. Pour un pixel de coordonnées $\begin{bmatrix} x_{p\text{dis}} & y_{p\text{dis}} \end{bmatrix}^T$ dans l'image, on calcule le carré de la distance r au point principal (dans une unité valant 1 pour la distance focale) en suivant l'équation 2.7

$$\begin{aligned} U &= \frac{x_{p\text{dis}} - u_0}{f_x} \\ V &= \frac{y_{p\text{dis}} - v_0}{f_y} \\ r^2 &= U^2 + V^2 \end{aligned} \tag{2.7}$$

On applique ensuite le polynôme sur r^2 pour obtenir la correction de distance dr_{cor} :

$$dr_{cor} = a_5 r^{10} + a_4 r^8 + a_3 r^6 + a_2 r^4 + a_1 r^2 \tag{2.8}$$

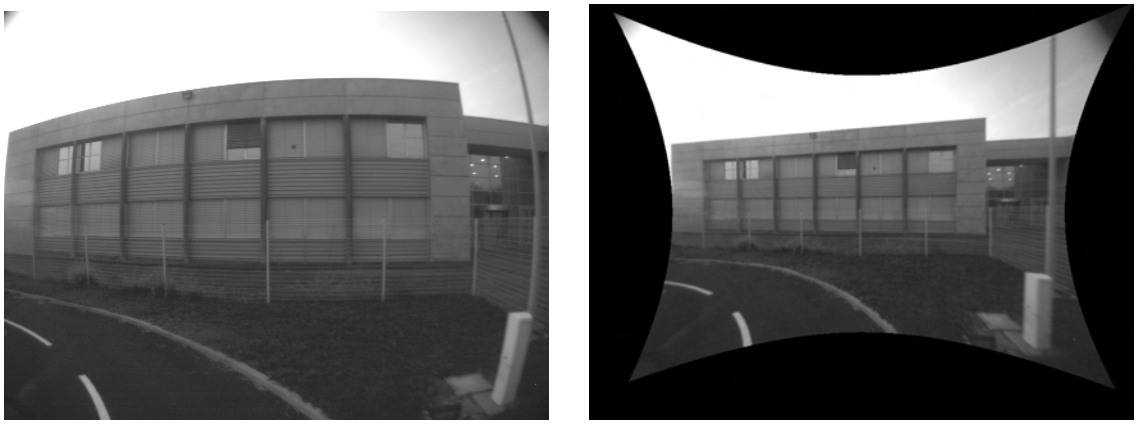
et on applique finalement la nouvelle distance pour obtenir les coordonnées correctes

$$\begin{aligned} x_p &= U(1 + dr_{cor}) f_x + X_{centre} \\ y_p &= V(1 + dr_{cor}) f_x + Y_{centre} \end{aligned} \tag{2.9}$$

avec X_{centre}, Y_{centre} les coordonnées du pixel au centre de l'image, où sera situé le point principal après correction.

Les paramètres intrinsèques de la caméra ainsi que les coefficients de distorsion sont obtenus lors d'une étape d'étalonnage préalable décrite par (Lébraly, 2012). La figure 2.2 montre un

exemple d'image distordue (2.2a) et corrigée (2.2b) en utilisant la procédure décrite. On peut constater que les bords des fenêtres et le poteau qui apparaissent arrondis sur l'image distordue, deviennent parfaitement rectilignes après la correction.



(a) Image distordue (b) Image corrigée

FIGURE 2.2 – Exemple de correction de la distorsion sur une image

Cette étape de correction reste nécessaire pour pouvoir correctement retrouver la position d'un point vu auparavant et le comparer avec une image non distordue.

2.3 Création du patch

2.3.1 Étape de reconstruction

Une reconstruction initiale est réalisée en utilisant l'algorithme décrit dans la partie 1.3.1. On obtient un ensemble de points qui ont été suivis sur plusieurs images-clef avec pour chaque point :

- ses coordonnées 3D ;
- la covariance sur sa position ;
- ses coordonnées 2D dans les images-clef où il a été observé.

De plus la pose de caméra associée à chaque image (clef ou non) est connue. Toutes ces données vont ensuite être combinées pour créer le patch.

2.3.2 Suivi de point

Pour avoir le meilleur modèle possible de patchs, il est préférable d'avoir le plus d'observations possible du même point. Pour cela, les points sont recherchés sur l'ensemble des images

qu'elles aient ou non été sélectionnées comme image clef. Auparavant, afin de ne plus avoir à gérer la distorsion, l'ensemble des images aura été préalablement corrigé avec la méthode décrite en partie 2.2.2. Ceci permettra également, lors de la génération ultérieure des patchs, de créer des vues non distordues sans avoir à recalculer systématiquement le polynôme de degré 5. On considère donc dans la suite de ce manuscrit que les images sont corrigées et qu'on utilise le modèle sténopé équivalent.

Avec le résultat de la cartographie, c'est-à-dire les poses de toutes les images et les positions 3D du patch, l'équation 2.6 fournit la position (en pixel) de chaque point dans chaque image. Lorsque ces positions sont situées dans l'image, un voisinage rectangulaire de 16×16 pixels autour du point est défini et comparé par ZNCC avec le voisinage du point sur l'image clef la plus proche. L'image clef la plus proche est sélectionnée comme étant celle où la position 3D du centre de la caméra est à la plus proche du centre de la caméra courante de manière à comparer le point avec la vue la plus proche de celle recherchée. Ce processus est illustré par la figure 2.3.

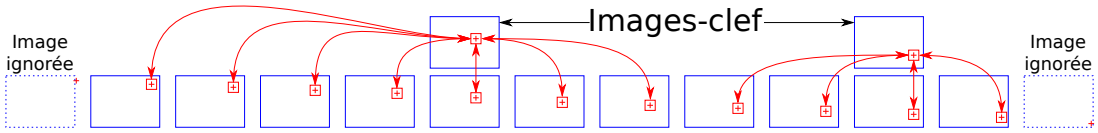


FIGURE 2.3 – Comparaison d'un point avec les images clef les plus proches. Dans cet exemple le point est suivi sur 2 images clef et sa reprojection se retrouve dans 11 images de la séquence complète. Les flèches rouges montrent la ZNCC réalisée entre les deux zones carrées. Les images en pointillées aux extrémités sont celles qui sont ignorées car le point (croix rouge) a été projeté en dehors de l'image

Lorsque le score de la ZNCC est suffisamment élevé (supérieur à 0,8) on considère que le point est bien présent et l'image sera donc utilisée pour générer le patch par la suite. Afin d'autoriser de petites erreurs de reprojection, une ZNCC est également calculée en se décalant d'un pixel dans les 4 directions principales (haut, bas, gauche et droite), et le score maximal est conservé. De cette manière, si le point est à moins de 1 pixel de l'endroit projeté, le score restera assez élevé et l'image sera utilisée pour le calcul du patch.

Avec cette méthode, un point qui n'a été suivi que sur quelques images clef pourra être retrouvé sur de nombreuses images et ce, même si le détecteur de point d'intérêt ne le donnait pas sur les images.

2.3.3 Calcul de normale

Après avoir suivi un point sur plusieurs images, et donc avoir obtenu plusieurs vues de ce point, il est nécessaire de calculer son orientation. Cela permettra, lors de la localisation, de le reprojeter. Pour cela on suppose que chaque point repose sur une surface localement plane. Connaitre l'orientation du plan revient donc à en calculer sa normale.

Pour simplifier le problème, on commence par considérer un point $\mathbf{P_w}$ qui n'est vu que sur deux images, comme sur la figure 2.4.

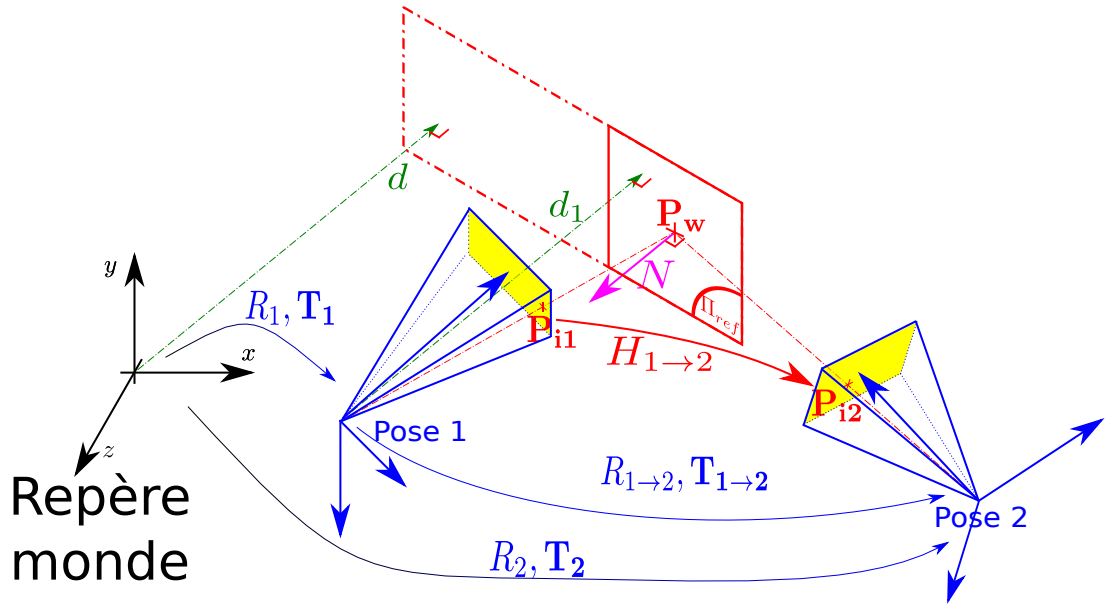


FIGURE 2.4 – Patch vu depuis deux poses différentes.

Homographie induite par un plan

Comme expliqué dans (Hartley and Zisserman, 2004), lorsque deux caméras observent un même plan, une homographie $H_{1\to2}$ peut être définie pour relier les coordonnées en pixel $\mathbf{P_{p1}}$ de chaque point dans la première image à ses coordonnées en pixel $\mathbf{P_{p2}}$ dans la seconde image avec l'équation 2.10.

$$\mathbf{P_{p2}} \equiv H_{1\to2}\mathbf{P_{p1}} \tag{2.10}$$

$H_{1\to2}$ est l'homographie induite par le plan et peut être exprimée par l'équation 2.11

$$H_{1\to2} = K(R_{1\to2} \quad \frac{\mathbf{T_{1\to2}}\mathbf{N_1}^t}{d_1})K^{-1} \tag{2.11}$$

avec

- $R_{1\to2}$ et $\mathbf{T_{1\to2}}$ la matrice de rotation et le vecteur de translation qui transforment les coordonnées dans la caméra 1 en coordonnées dans la caméra 2 selon l'équation $\mathbf{P_{c2}} = R_{1\to2}\mathbf{P_{c1}} + \mathbf{T_{1\to2}}$;
- d_1 la distance algébrique du centre de la caméra 1 au plan ;
- $\mathbf{N_1}$ la normale au plan exprimée dans le repère de la caméra 1 ;

➢ K la matrice des paramètres intrinsèques des caméras (supposée identique pour les deux vues).

Toutes ces valeurs se retrouvent sur la figure 2.4.

Cette équation suppose cependant que la première caméra est à l'origine du repère, ce qui n'est pas notre cas dans le cadre de la localisation. Le calcul de cette homographie doit donc être adapté à notre cas où les matrices de rotation et vecteurs de translation associés aux caméras ne sont connus qu'à partir du repère monde. La démonstration est faite en annexe A, et permet d'obtenir l'homographie reliant $\mathbf{P_{p1}}$ à $\mathbf{P_{p2}}$:

$$H_{1\to 2} = K\left(R_2 R_1^T + R_2(\mathbf{T_1} - \mathbf{T_2})\frac{\mathbf{N}^T R_1^T}{\mathbf{N}^T\mathbf{P_w} - \mathbf{N}^T\mathbf{T_1}}\right)K^{-1} \tag{2.12}$$

avec

➢ R_1,$\mathbf{T_1}$ (resp. R_2,$\mathbf{T_2}$) la matrice de rotation et le vecteur de translation de la première (resp. seconde) caméra qui ont été fournis par la cartographie préalable ;

➢ $\mathbf{N}$ la normale au plan exprimée dans le repère monde.

Méthode d'optimisation utilisée

L'expression de l'homographie selon l'équation 2.12 permet de calculer une fonction de cout qui transforme une image I_1 en image I_2 et compare la différence entre les images. Cette fonction peut s'exprimer par l'équation 2.13

$$C = \sum_{(x,y)\in V_{\mathbf{P_w}}} \left[\left[I_1\left(\mathbf{P_{p1}} + \begin{bmatrix} x \\ y \end{bmatrix}\right) - I_2\left(H_{1\to 2}\left(\mathbf{P_{p1}} + \begin{bmatrix} x \\ y \end{bmatrix}\right)\right)\right]^2 + \left[I_1\left(H_{1\to 2}^{-1}\left(\mathbf{P_{p2}} + \begin{bmatrix} x \\ y \end{bmatrix}\right)\right) - I_2\left(\mathbf{P_{p2}} + \begin{bmatrix} x \\ y \end{bmatrix}\right)\right]^2\right] \tag{2.13}$$

avec V_{P_w} les coordonnées autour du point $\mathbf{P_w}$ (dans notre cas un carré de 16×16 pixels, donc x et y variant de -8 à 8).

Tous les paramètres de cette fonction sont connus à l'exception de l'homographie $H_{1\to 2}$ qui ne dépend que de la normale $\mathbf{N}$. On obtient donc une évaluation de la différence entre deux images pour une valeur de l'homographie (et donc de la normale) donnée. En théorie cette fonction toujours positive doit être nulle lorsque la normale correspond à celle du plan. On va donc utiliser un algorithme d'optimisation par la méthode de Levenberg-Marquardt pour trouver la valeur de $\mathbf{N}$ qui minimise la fonction C.

On remarque également que la fonction C dépend également de $\mathbf{P_w}$, par le biais à la fois du voisinage $V_{\mathbf{P_w}}$ et des projection sur les images $\mathbf{P_{p1}}$ et $\mathbf{P_{p2}}$. Ces valeurs peuvent donc avoir une influence importante sur l'optimisation et pourrait même être optimisée en plus de la normale

dans cette opération. Cependant réaliser un raffinement de la reconstruction des point apporterait une complexité et un cout de calcul important qui semble peu intéressant devant les gains potentiels, la reconstruction initiale étant déjà assez bonne. Par conséquent, bien que ce soit formellement possible et mériterai une étude plus approfondie, cette étape de raffinement des coordonnées de $\mathbf{P_w}$ et des poses de caméra a été ignorée dans ce travail.

Paramétrisation de la normale

La méthode d'optimisation consiste donc à trouver le minimum de la fonction cout lorsque la valeur de la normale varie. Cependant, on peut constater que la valeur de l'homographie (et donc de la fonction cout) ne change pas si la normale est multipliée par un facteur. Cela confirme l'idée assez intuitive que seule l'orientation de la normale a un intérêt, son sens et sa norme pouvant être arbitrairement fixés. De plus lorsque l'on paramètre $\mathbf{N}$ avec ses trois coordonnées, sa norme peut varier dans des proportions énormes. Afin de fixer le sens et la norme de $\mathbf{N}$, il semble intéressant de n'utiliser que 2 paramètres scalaires. Pour rester dans des conditions semblables au schéma de la figure 2.4, on choisit de fixer le sens de la normale comme se dirigeant vers les caméras. Cela revient à dire que l'orientation de la normale ne peut varier qu'en restant dans la demi-sphère dirigée vers la caméra moyenne, et visible sur la figure 2.5.

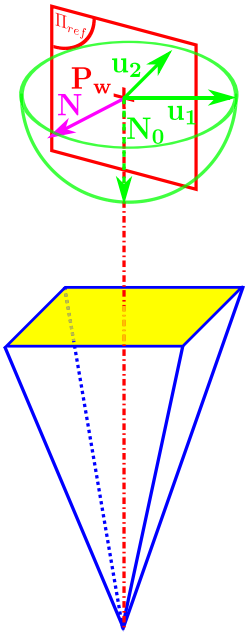


FIGURE 2.5 – Représentation des paramètres de la normale.

Pour pouvoir ensuite définir la normale, on commence par définir le vecteur $\mathbf{N_0}$, qui correspond à la valeur initiale de la normale dans l'algorithme d'optimisation. Il s'agit du vecteur

unitaire suivant la direction qui va du point $\mathbf{P_w}$ au barycentre des centres des caméras de référence. Ensuite, on définit arbitrairement deux vecteurs unitaires $\mathbf{u_1}$ et $\mathbf{u_2}$ orthogonaux à $\mathbf{N_0}$ et orthogonaux entre eux. On obtient alors une base orthonormée $(\mathbf{N_0}, \mathbf{u_1}, \mathbf{u_2})$. La normale peut ensuite être définie en fixant deux paramètres α et β et en utilisant l'expression 2.14

$$\mathbf{N} = \frac{\mathbf{N_0} + \alpha\mathbf{u_1} + \beta\mathbf{u_2}}{\|\mathbf{N_0} + \alpha\mathbf{u_1} + \beta\mathbf{u_2}\|} \tag{2.14}$$

La normale peut prendre alors toutes les directions possibles dans la demi-sphère dirigée vers l'ensemble des caméras de référence. Comme le patch a pu être observé par la caméra, il est logique que la normale soit incluse dans cette demi-sphère.

Généralisation à plusieurs images

La fonction de cout est définie par l'équation 2.13 pour un point vu sur deux caméras. Cependant l'utilisation de davantage d'images (si possible des images prises depuis des points de vue très différents) permet d'améliorer les résultats. Lorsque l'on considère un nombre quelconque d'images, on peut alors utiliser la fonction cout suivante :

$$C = \sum_{n_1 \neq n_2} \sum_{(x,y) \in V_{\mathbf{P_w}}} \left[I_{n_1}\left(\mathbf{P_{pn_1}} + \begin{bmatrix} x \\ y \end{bmatrix}\right) - I_{n_2}\left(H_{n_1 \to n_2}\left(\mathbf{P_{pn_2}} + \begin{bmatrix} x \\ y \end{bmatrix}\right)\right)\right]^2 \tag{2.15}$$

Il s'agit en fait de la même équation que pour deux images, mis à part que dans notre cas, l'homographie inverse $H^{-1}_{n_1 \to n_2}$ n'est pas utilisée. En effet lors du calcul de la somme, la comparaison sera faite avec $H_{n_2 \to n_1}$ qui est égale à l'inverse de $H^{-1}_{n_1 \to n_2}$.

Comme toutes les paires d'images possibles sont considérées dans la somme, la complexité du calcul de la fonction cout varie en $o(N^2)$ par rapport au nombre d'images considérées. Pour garder des performances utilisables dans notre traitement, les calculs de normale sont réalisés en prenant 3 images. Toutefois pour avoir un champ de vue le plus large possible les images utilisées sont celles étant les plus éloignées les unes des autres, garantissant un changement de point de vue le plus marqué possible.

Par ailleurs l'optimisation réalisée dans notre cas, en utilisant un algorithme de Levenberg-Marquardt n'est pas forcément la manière la plus rapide de procéder. En particulier la méthode de composition inverse définie par (Baker and Matthews, 2004), pourrait permettre de retrouver l'homographie plus rapidement à partir de deux images. Cependant, le calcul de normale étant effectué lors d'une phase hors-ligne il n'a pas été utile d'implémenter cette méthode qui reste donc une perspective d'amélioration.

Après avoir réalisé l'optimisation, on obtient donc pour chaque point, une orientation 3D de la surface. Cette orientation permettrait de reprojeter la vue du point dans n'importe quelle vue alentour. Il reste cependant utile de calculer l'apparence du patch en considérant toutes les vues déjà disponibles dans les images de référence.

2.3.4 Calcul des textures

Pour pouvoir comparer un patch avec l'image courante de l'environnement, il est indispensable de savoir quelle texture sera présente sur le plan associé au point.

FIGURE 2.6 – Vue de dessus de la position de la pose virtuelle. On peut voir que la pose moyenne est bien plus proche des vues d'origines que la pose fronto-parallèle.

Cette texture sera décrite par une image Π_{ref}, schématisée sur la figure 2.6. L'image générée devra être une compilation de toutes les vues de référence, c'est-à-dire toutes les images du point trouvées dans la partie 2.3.2, afin de tenir compte de tous les points de vue.

Afin d'avoir une image représentative du patch, il est nécessaire de définir une pose virtuelle qui correspond à la position C de la caméra qui pourrait voir l'image Π_{ref}. Une première idée serait d'utiliser une vue fronto-parallèle, qui donne une image du plan dans le sens de la normale, et donc comme si on faisait face au plan. Cependant de nombreux points suivis sont observés uniquement sur le côté. Par exemple dans un contexte de localisation, les façades des bâtiments sont observées depuis le côté, et pratiquement jamais de face (comme schématisé figure 2.6). Une vue fronto-parallèle serait alors très éloignée des vues d'origine et demanderait un important rééchantillonnage des images pour être générée. De plus, lors de la localisation, il serait nécessaire de faire à nouveau un important rééchantillonnage lorsque le véhicule se situera dans la même zone que sur les vues d'origine. Or les importants rééchantillonnages risquent d'apporter du bruit et du flou sur l'image générée. Pour éviter cela, le centre C de la pose virtuelle est placé à proximité des poses de référence. C est pris comme étant le barycentre des centres des vue d'origine. La pose est orientée de manière à avoir le centre du patch sur l'axe optique. La pose virtuelle utilisée correspond donc à la pose moyenne de la figure 2.6. De cette manière la transformation de chaque image de référence en une vue de la pose virtuelle ne demande pas de rééchantillonnage trop important.

Pour générer l'image Π_{ref}, on utilise l'équation 2.12 pour obtenir l'homographie $H_{vir \to i}$ qui projette chaque image de référence I_i dans la vue virtuelle. On obtient alors pour chaque vue

d'origine une image avec l'équation suivante :

$$\Pi_i \left(\begin{bmatrix} x \\ y \end{bmatrix} \right) = I_i \left(H_{vir \to i} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \right) \tag{2.16}$$

En réalisant la moyenne de toutes ces images, on obtient l'image Π_{ref} souhaitée. Cette méthode permet donc d'utiliser l'ensemble des observations du point 3D. La taille de Π_{ref} peut être choisie arbitrairement. Toutefois, une trop grosse taille risque d'englober une surface trop grande autour du point, qui ne respecte plus l'hypothèse de planéité locale, et une surface trop petite est inutilisable pour comparer l'image du point avec d'autres données. Dans nos expériences un carré de 32 pixels de coté est utilisé. On peut voir un exemple de patch généré sur la figure 2.7.

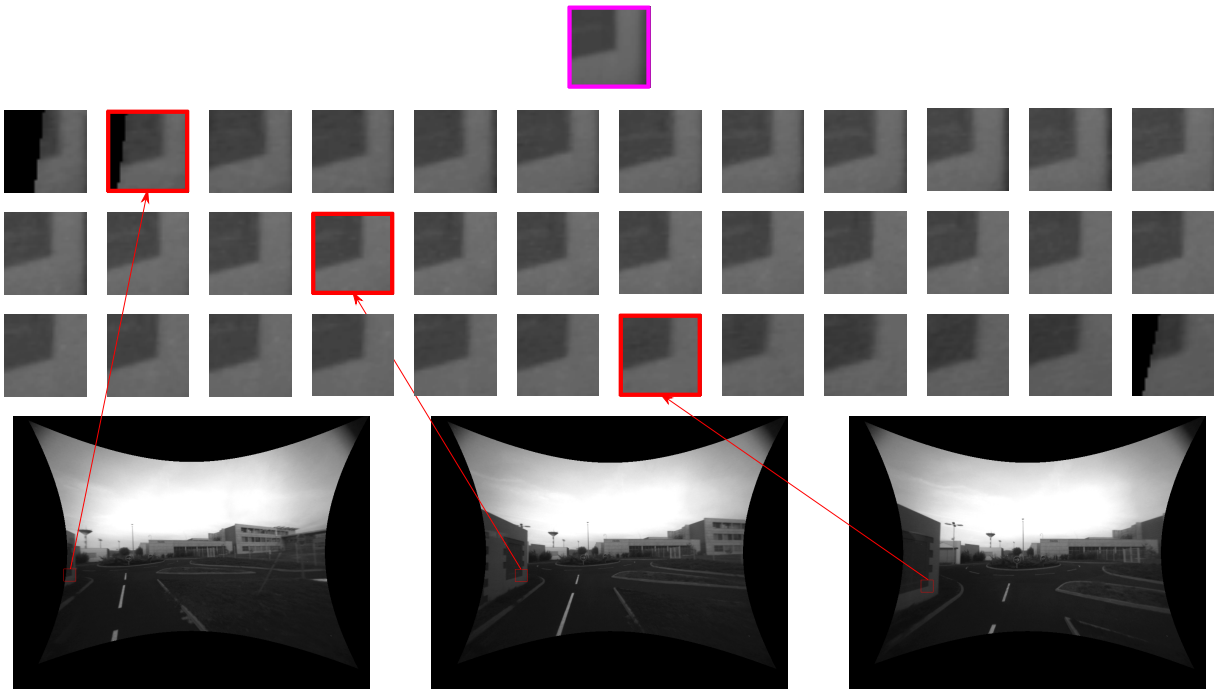


FIGURE 2.7 – Exemple d'image de points suivis et de la texture générée. En haut, l'image encadrée de violet représente la texture générée, l'ensemble des autres petites images sont toutes les vues du point suivi et en dessous 3 images d'où ont été extraites certaines petites images (celles encadrées en rouge)

Avec cette texture, tous les outils nécessaires pour projeter un patch dans une vue courante sont disponibles. Cependant, cette texture peut présenter des défauts, en particulier lors de mauvais appariements. Il est donc intéressant d'évaluer la qualité de la texture et si possible de l'améliorer avant de passer à la phase de localisation.

2.3.5 Évaluation de la qualité

Lorsqu'un point n'est pas vraiment situé sur un plan, ou est issu de mauvais appariements, la moyenne des différentes vues utilisée pour le calcul de la texture peut rendre la texture particulièrement floue et peu exploitable. En particulier si de nombreuses vues différentes sont combinées, on obtient une image homogène grise qui lors d'une ZNCC risque d'obtenir un score assez élevé avec n'importe quel autre point. Cela provoquerait un faux appariement et, s'ils sont trop nombreux il peut en résulter une erreur de localisation. Par ailleurs, de nombreux points ont été suivis. Les reprojeter à chaque itération, en particulier si le patch n'est pas très bon a priori, prendrait beaucoup de temps de calcul et ne serait pas utile. Pour ces raisons, il est nécessaire d'évaluer a priori la qualité de la reconstruction, afin d'éliminer les patchs les moins intéressants.

La manière utilisée pour cette évaluation consiste à comparer l'image Π_{ref} obtenue avec les images de référence qui l'ont générée. En particulier, si une image de référence était issue d'un mauvais appariement, on peut supposer que l'image Π_{ref} générée ne lui ressemble que faiblement. Pour vérifier cela, pour chaque image de référence I_i, Π_{ref} est projetée dans la pose associée en utilisant l'homographie $H_{vir \to i}$ (calculée selon l'équation 2.12 vu précédemment). On obtient alors une image Π_i, correspondant au patch tel qu'il devrait être observé dans l'image I_i. Une ZNCC est alors calculée entre I_i et Π_i pour obtenir un score s_i. Un seuil (0,8 lors de nos tests) est appliqué sur tous les scores ainsi obtenus, et on peut alors définir deux valeurs :

Q_1 Le nombre de vues retrouvées N_{ret}, qui correspond au nombre de valeurs de s_i supérieures au seuil

Q_2 Le rapport entre N_{ret} et le nombre N_{ref} d'images de référence qui ont généré le patch

La première valeur Q_1 permet de savoir sur combien d'images le patch a été vu. Cela peut donner une idée de la pertinence du patch. En effet, un point que l'on a retrouvé sur une longue partie de la trajectoire est un bon point de repère, alors que s'il n'a été trouvé que sur 3 images, il est peu probable qu'il permette d'obtenir une bonne localisation.

La seconde valeur Q_2 est un pourcentage montrant la similarité du patch généré avec les images de référence. Lorsqu'une image est très différente des autres, on a probablement un mauvais appariement et cette valeur diminuera. A l'inverse si toutes les images de référence sont (au changement de point de vue près) quasiment identiques, le patch ressemblera à toutes et sera très fiable.

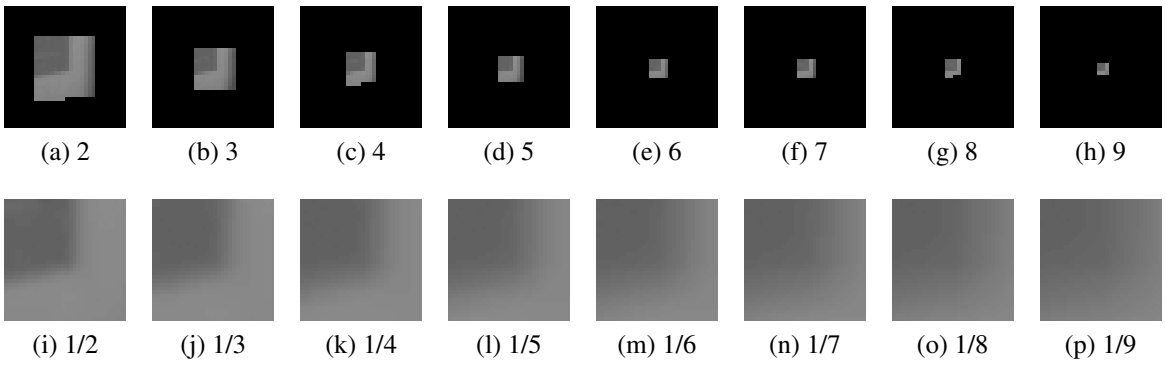
Ces deux valeurs peuvent être seuillées pour conserver uniquement les patchs ayant une qualité minimale. Dans nos tests un patch est considéré comme exploitable s'il est retrouvé sur au moins 3 images et est similaire à plus de 80% des images de référence.

Par ailleurs, plutôt que d'éliminer directement les patchs qui ont un coefficient Q_2 trop faible, il est possible de tenter d'améliorer leur qualité. En particulier si seulement quelques images de référence diffèrent de la texture, on peut supposer que seules ces images sont issues d'un faux appariement. On va alors supprimer ces images de référence et recréer la texture

avec la méthode décrite précédemment et tester à nouveau la qualité. Le patch devrait alors ressembler davantage aux images de référence restantes et avoir un coefficient Q_2 augmenté. Cela reflète une amélioration de la texture qui n'aura pas été bruitée par des images issues d'un faux appariement. Si parmi les images de référence restantes, certaines sont encore différentes du patch, on peut continuer à les supprimer et itérer aussi longtemps que nécessaire afin de supprimer tous les mauvais appariements potentiels qui seraient encore présents.

2.3.6 Evaluation de la zone d'observabilité

Même lorsqu'un patch a une bonne qualité, il n'est pas possible de le retrouver depuis n'importe quel point de vue. Il peut déjà y avoir des occultations telles qu'un mur ou une autre infrastructure qui empêche de le voir depuis certains angles. De plus la distance à laquelle il est observé a aussi une grande influence. Pour illustrer ce phénomène, la figure 2.8 montre le patch décrit précédemment (figure 2.7) tel qu'il apparaitrait à différentes distances d'observation. Lorsque le patch est généré pour une vue beaucoup plus proche que la vue initiale, on s'aperçoit qu'on a très vite une surface unie (qui correspond en fait à un pixel de l'image initiale). De même lorsqu'on s'en éloigne l'image générée ne contient plus que quelques pixels, et cela n'a guère de sens de réaliser un appariement dans ces conditions.



(a) 2 (b) 3 (c) 4 (d) 5 (e) 6 (f) 7 (g) 8 (h) 9

(i) 1/2 (j) 1/3 (k) 1/4 (l) 1/5 (m) 1/6 (n) 1/7 (o) 1/8 (p) 1/9

FIGURE 2.8 – Exemple d'image de patchs à plusieurs distances. La première ligne (de (a) à (h)) s'éloigne de la référence (de 2 à 9 fois la distance initiale) et la seconde ligne (de (i) à (p)) s'en rapproche (distance divisée par 2 à 9).

Pour éviter de tenter d'apparier un patch dans une vue trop éloignée de la texture originale, on va donc définir une zone d'observabilité. Un patch ne sera projeté et analysé que lorsque le centre $\mathbf{C_{courant}}$ de la caméra courante sera dans la zone d'observabilité. Cette zone d'observabilité correspond à la zone où sont situés les centres des caméras de référence, comme le montre la figure 2.9. Seule la position de la caméra est considérée pour le calcul de la zone. Bien entendu, il faudra lors de la localisation, vérifier également en fonction de l'orientation de la caméra, que le patch est bien visible.

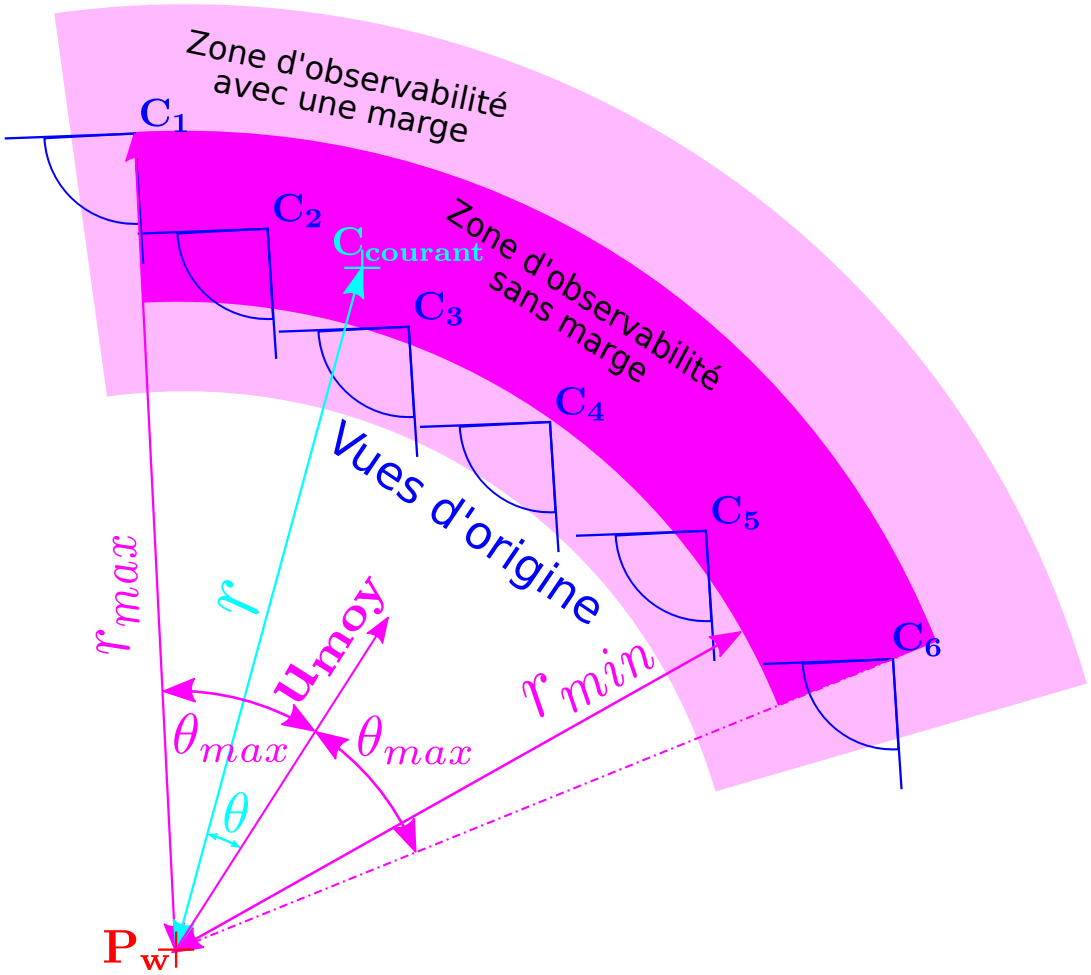


FIGURE 2.9 – Définition de la zone d'observabilité d'un patch en vue de dessus

La zone d'observabilité est définie par rapport à la position centrale $\mathbf{P_w}$ du patch. L'idée est de définir quelques paramètres pour que en connaissant la position $\mathbf{C_{courant}}$ on puisse rapidement savoir si l'on peut observer le patch ou non. Pour être situé dans la zone d'observabilité, il est nécessaire de considérer une contrainte en distance et une contrainte en angle.

Contrainte en distance

Pour la contrainte en distance, on enregistre les distances limites depuis lesquelles le patch a été observé. On obtient les valeurs r_{min} et r_{max} correspondant à la distance entre $\mathbf{P_w}$ et le centre de la caméra de référence, respectivement la plus proche et la plus éloignée. Lorsque l'on connait la position $\mathbf{C_{courant}}$, on calcule la distance r entre ce point et $\mathbf{P_w}$. On se trouve

dans la zone où étaient les poses de référence lorsque la relation 2.17 est vérifiée.

$$r_{min} \leq r \leq r_{max} \tag{2.17}$$

Cela permet d'éviter les cas illustrés sur la figure 2.8 où le patch est observé depuis une distance bien trop importante ou trop faible pour être apparié correctement. Au niveau de la zone d'observabilité elle-même (figure 2.9), cette contrainte permet de définir les deux rayons de la couronne dont une partie formera la zone d'observabilité.

Contrainte en angle

La définition de la contrainte en angle est un peu plus complexe que pour la distance. Afin de faciliter la définition, et dans la mesure où le robot se déplace sur une route, il est préférable de se placer dans le plan du sol. En effet le véhicule ayant une hauteur fixe, l'altitude de la caméra ne devrait pas beaucoup varier pour une position donnée. En se plaçant dans le plan du sol, on va s'intéresser à l'angle depuis lequel le patch a pu être perçu. On utilise les centres des caméras pour lesquels l'angle formé par les vecteurs reliant le point P_w et chacun des deux centre de caméra est le plus élevé possible. Sur la figure 2.9, il s'agit des centres $\mathbf{C_1}$ et $\mathbf{C_6}$. On prend la moitié de l'angle entre les deux caméras que l'on nomme θ_{max}. On considère ensuite le vecteur unitaire $\mathbf{u_{moy}}$, moyenne des vecteurs allant du point à chacun des deux centres. Ce vecteur permet, en utilisant la position $\mathbf{C_{courant}}$, de calculer par un produit scalaire l'angle θ entre le vecteur allant de $\mathbf{P_w}$ à $\mathbf{C_{courant}}$ et $\mathbf{u_{moy}}$. La relation 2.18 est alors vérifié lorsqu'on se trouve dans la zone où étaient les poses de référence.

$$|\theta| \leq \theta_{max} \tag{2.18}$$

Cela permet de déterminer la largeur de l'arc de la couronne formant la zone d'observabilité. En pratique cette contrainte permet d'éliminer les patchs lorsqu'un obstacle empêche de voir un point depuis un trop grand angle.

Marge de la zone

Ces deux relations assez strictes permettent de savoir si un point est situé dans la zone où étaient présentes les caméras de référence. On pourrait considérer qu'un point n'est a priori observable que s'il est dans cette zone et ainsi éviter de rechercher des points trop loin de la zone où ils sont apparus. Toutefois se restreindre strictement à la position des références peut sembler un peu trop contraignant. En effet, l'intérêt de la modélisation sous forme de patchs est de pouvoir les retrouver lorsqu'on s'éloigne de la trajectoire, et donc depuis un point de vue encore jamais utilisé. Pour éviter que l'observabilité ne soit une contrainte trop forte, il est nécessaire de définir une marge autour d'elle. Pour un couple de valeurs (r, θ) donné la contrainte d'appartenance à la zone d'observabilité se définit alors par le système 2.19

$$\begin{cases} r_{min}(1-\rho_r) - \lambda_r \leq r \leq r_{max}(1+\rho_r) + \lambda_r \\ |\theta| \leq \theta_{max}(1+\rho_\theta) + \lambda_\theta \end{cases} \tag{2.19}$$

avec

- ρ_r et ρ_θ paramètres scalaires permettant de définir la marge d'observabilité de manière proportionnelle à la taille existante (par exemple on peut voir un patch deux fois plus loin avec $\rho_r = 1$, ou suivant un angle deux fois plus grand avec $\rho_\theta = 1$)
- λ_r et λ_θ paramètres scalaires dans la même unité que respectivement r et θ permettant de définir une marge constante (par exemple on peut voir un patch à une distance de plus ou moins 1 m et suivant un angle de plus ou moins 5 degrés au-delà de la limite).

Ce test peut être facilement calculé avant de réaliser la projection des patchs et ainsi éviter du calcul inutile. Cependant en jouant sur les paramètres de marge il est facile de moduler la sélectivité de ce test et par conséquent d'améliorer plus ou moins les temps de calcul.

Utilisation de la zone sous forme de facteur

L'approche décrite précédemment permet de définir un critère fixe annonçant si une position est ou non dans une zone d'observabilité. Cependant il peut être intéressant, au lieu de savoir simplement si l'on est dans la zone ou en dehors, d'évaluer si l'on en est proche ou non. Cela permet par exemple de savoir quels sont a priori les meilleurs patchs même si on est dans leurs zone d'observabilité à tous. Il devient alors possible de conserver un nombre fixe de patchs à projeter, peu importe que la position soit réellement dans la zone ou non. Pour cela on définit les valeurs μ_r et μ_θ calculées en utilisant l'équation 2.20

$$\begin{cases} \mu_r = \dfrac{\left| r - \frac{r_{min}+r_{max}}{2} \right|}{\frac{r_{max}-r_{min}}{2}} = \dfrac{|2r - r_{min} + r_{max}|}{r_{max} - r_{min}} \\ \mu_\theta = \dfrac{|\theta|}{\theta_{max}} \end{cases} \tag{2.20}$$

Ces deux valeurs positives sont d'autant plus faibles que $\mathbf{C_{courant}}$ est proche du centre de la zone d'observabilité. En particulier chaque facteur sera compris entre 0 et 1 si l'on est dans la zone d'observabilité et supérieur si l'on en sort. On peut alors utiliser ces valeurs un peu comme le facteur de qualité, pour conserver uniquement les N meilleurs patchs sans forcément avoir un seuil fixé.

2.3.7 Résultat de la création du patch

Pour résumer cette partie, on a pu, à partir d'un amer associé à un certain nombre de vues de référence, le transformer en un patch associé à plusieurs données. On dispose alors pour chaque amer d'un ensemble de valeurs contenant :

- $\mathbf{P_w}$ les coordonnées 3D du point central ;
- $\mathbf{N}$ l'orientation de la normale ;
- Une pose virtuelle d'où est observé le patch de référence ;

- ➢ Π_{ref} l'image (de taille 32×32 pixels) du patch telle que vue par la pose de référence
- ➢ (Q_1, Q_2) deux valeurs constituant le facteur de qualité
- ➢ $(r_{min}, r_{max}, \mathbf{u_{moy}}, \theta)$ un ensemble de 3 valeurs et un vecteur qui forme la zone d'observabilité

Pour chaque amer l'ensemble de ces valeurs est stocké et permettra par la suite de se localiser. C'est uniquement l'ensemble de ces patchs qui constitue la carte de localisation. Toutes les informations précédentes (pose des vues d'origine, images clef et poses associées, etc) ne seront plus utile pour la localisation. Il peut être intéressant de conserver la trajectoire d'apprentissage à des fins de visualisation et de commande, mais celle-ci ne sera pas utilisée lors de la localisation.

2.4 Algorithme de localisation basé sur les patchs

Une fois les patchs calculés, la localisation est réalisée en suivant le schéma de la figure 2.10.

Une première partie consiste à traiter l'image courante de manière à obtenir la position de tous les points d'intérêt et leur descripteur associé. Elle se décompose en une première phase de correction d'image, pour enlever la distorsion grâce à la méthode décrite à la section 1.3.1. La détection des points d'intérêt utilise un détecteur classique tel que ceux décrits en section 1.2.1. Le calcul des descripteurs (qui servent de précalculs à une ZNCC) peut se faire avec une approche classique utilisant les méthodes décrites en 1.2.2 ou d'une manière plus adaptée à l'appariement choisi qui sera décrite par la suite.

Parallèlement, la connaissance de la pose précédente et de son incertitude associée, permet de réaliser une prédiction de la pose courante. Cette prédiction peut être réalisée simplement en reprenant la position précédente et augmentant son incertitude. Il est également possible d'utiliser un modèle de mouvement et d'autres capteurs comme l'odométrie, comme cela sera détaillé au chapitre 4.

La partie concernant directement l'utilisation des patchs se décompose en plusieurs étapes. La première consiste à sélectionner les patchs potentiellement visibles et sera décrit en 2.4.1. Ensuite les patchs sont projetés dans la vue courante comme expliqué en 2.4.2. Et pour finir l'appariement est détaillé en 2.4.3. Après ces étapes, le calcul de pose est réalisé en utilisant la méthode d'optimisation itérative déjà abordée en 1.3.2.

2.4.1 Sélection des patchs potentiellement visibles

Auparavant, dans la phase hors ligne, les patchs ont été triés en fonction de la qualité. On considère donc que tous les patchs sont exploitables et le facteur de qualité ne sera plus utilisé lors de la localisation. La sélection des patchs ne sera donc basée que sur la position courante pour déterminer s'ils sont visibles ou non. La position courante vient du résultat de la pose

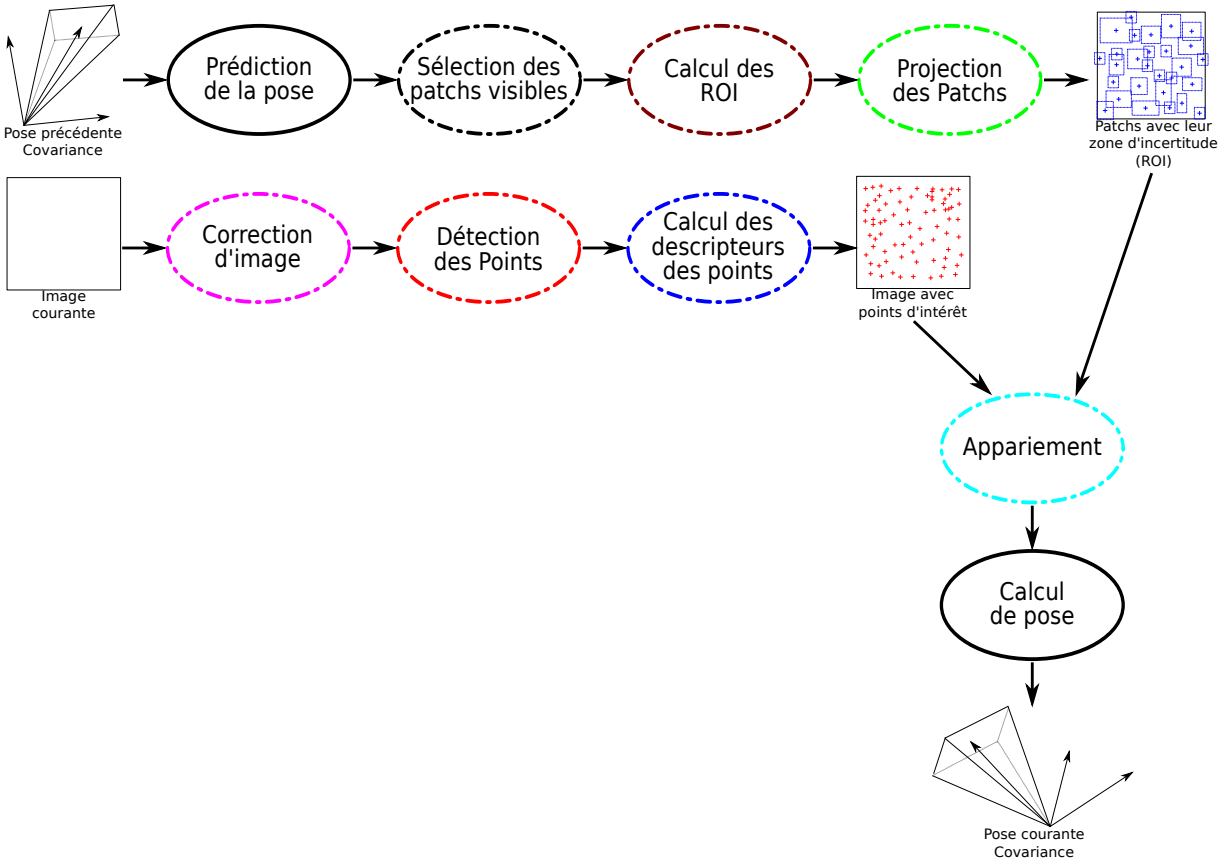


FIGURE 2.10 – Schéma d'ensemble de l'algorithme de localisation

prédite, donc une approximation de la pose réelle (ainsi que son incertitude). En connaissant cette pose on peut évaluer dans un premier temps quels sont les points présents devant la caméra. Cela permet d'éliminer un grand nombre de patchs qui ne seront pas analysés. Par la suite, c'est l'utilisation de la zone d'observabilité qui permettra de sélectionner les patchs potentiellement visibles. L'utilisation de la zone d'observabilité est décrite dans la partie 2.3.6. Tous les patchs dont la pose courante est en dehors de la zone d'observabilité sont alors éliminés. Les patchs restants sont donc devant la caméra et observables. Ce sont eux qui sont considérés comme potentiellement visibles.

Cette méthode assez simpliste est également celle qui pourrait être largement améliorée. Par exemple si on utilise la zone d'observabilité en tant que facteur (avec les valeur μ_r et μ_θ définie par l'équation 2.20), on pourrait coupler ces valeurs avec la qualité et obtenir un facteur unique pour chaque patch. Puis choisir de prendre un nombre fixe de patchs à chaque fois. On pourrait aussi envisager des systèmes plus évolués, par exemple à base de classifieur pour sélectionner a priori les patchs que l'on peut retrouver. Cette partie est en fait le dernier filtre que l'on peut appliquer aux patchs avant de lancer les processus plus lourds de reprojection et d'appariement.

2.4.2 Projection des patchs

Tous les patchs potentiellement visibles qui ont été sélectionnés par l'étape précédente doivent être projetés dans la vue prédite afin de les apparier avec les points de l'image. Pour cela on utilise l'équation 2.12 pour déterminer l'homographie $H_{predit \to vir}$ liant les coordonnées de l'image prédite à celle de la vue de référence Π_{ref}. On peut alors générer une nouvelle image Π_{pred} correspondant à l'image du patch vu depuis la position courante. Il est important de constater que l'image Π_{ref} ayant été transformée par une homographie, la projection obtenue Π_{ref} n'aura vraisemblablement pas la même forme. Par exemple si la texture du patch était représentée, comme dans nos expériences, par une image carrée de 32 pixels de coté, l'image déformée par l'homographie donnera un quadrilatère différent, par exemple aplati sur les bords comme illustré par la figure 2.11. En comparaison, le carré de 32 pixels de coté autour du point

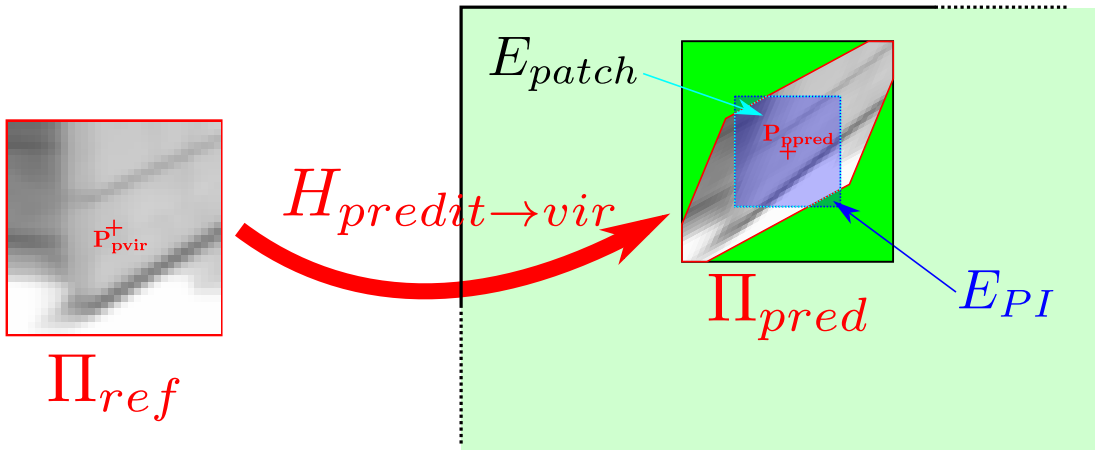


FIGURE 2.11 – Projection d'un patch Π_{ref} dans le plan image de la pose prédite. En rouge est représenté le contour de la reprojection du patch, et en vert un carré de la même taille que Π_{ref} centré sur la projection. On peut voir en particulier que de nombreux pixel de ce carré n'ont pas d'information de texture. Le carré pointillé en bleu autour de la reprojection donne le voisinage du point considéré en prenant un carré de la moitié de la taille de Π_{ref}.

de l'image contient beaucoup plus de valeurs, (représentée en vert sur la figure 2.11). Pour limiter la quantité de données inconnues, la reprojection du patch se limite à un carré de 16 pixels de coté, bien que la texture de référence soit 2 fois plus grande. Toutefois, comme on peut le voir sur la figure 2.11, il peut demeurer des valeurs inconnues dont il faudra tenir compte lors de l'appariement.

Cette opération est donc réalisée sur l'ensemble des patchs potentiellement visibles. De plus, l'homographie permet d'obtenir la position $\mathbf{P}_{\mathbf{ppred}}$ dans l'image vue de puis la pose prédite du centre du patch ($\mathbf{P}_{\mathbf{pvir}}$ sur l'image virtuelle). L'incertitude associée à la prédiction de la pose permet de définir une zone d'incertitude nommé région d'intérêt (ROI pour l'anglais *Region Of Interest*) autour de ce point (tout comme dans la méthode décrite en 1.3.2) où seront recherchés

les points d'intérêt de l'image que l'on tentera d'apparier avec le patch. On peut noter ici que plus la prédiction est précise (et donc son incertitude est réduite), plus les ROI seront de taille réduite.

On obtient donc un ensemble d'images ayant l'apparence de chaque patch dans la vue courante et, associées à ces image, une position et une zone où devraient se trouver ces patchs.

2.4.3 Appariement

L'appariement consiste à comparer d'un coté les patchs projetés et de l'autre les points d'intérêt $\mathbf{PI}$ détectés.

Détermination des appariements potentiel

La première étape consiste à déterminer tous les couples potentiels que l'on veut comparer. Pour cela on utilise la position $\mathbf{P_{ppred}}$ de chaque patch autour de laquelle est définie leur ROI. Tous les points d'intérêt $\mathbf{PI}$ présents dans cette ROI forment un couple avec le patch. Évidemment un même patch va former un couple avec plusieurs points et de même un point peut former un couple avec plusieurs patchs. C'est le score de corrélation qui permettra de les différencier.

Calcul du score d'appariement

Après avoir formé les couples, la comparaison est réalisée avec une ZNCC entre l'image projetée Π_{pred} du patch et le voisinage du point d'intérêt dans l'image courante I_{cour}. Toutefois, on rappelle que Π_{pred} n'est pas forcément de forme carrée (voir figure 2.11), et il faut en tenir compte dans l'équation. Pour cela, on définit les ensembles suivants :

- ➢ E_{patch} l'ensemble des coordonnées des points où la projection d'un patch a une valeur. Il s'agit de la zone entourée de pointillés cyan sur la figure 2.11
- ➢ E_{PI} l'ensemble des coordonnées des points situés dans le voisinage du point d'intérêt. On notera que E_{PI} est la plupart du temps un carré de 16 pixels de coté. Ce n'est pas le cas uniquement lorsque le point d'intérêt est proche du bord de l'image. La figure 2.12 montre un exemple de points détectés avec pour deux d'entre eux (dont un point situé au bord) le voisinage E_{PI} utilisé.

Avec ces notations, la formule de la ZNCC vaut :

$$ZNCC(\Pi_{pred}, I_{cour}) = \frac{\sum_{\delta \in E_\cap} \left[\left(\Pi_{pred}(\mathbf{P_{ppred}} + \delta) - \overline{\Pi_{pred}} \right) \left(I_{cour}(\mathbf{PI} + \delta) - \overline{I_{cour}} \right) \right]}{\sqrt{\sum_{\delta \in E_\cap} \left(\Pi_{pred}(\mathbf{P_{ppred}} + \delta) - \overline{\Pi_{pred}} \right)^2} \sqrt{\sum_{\delta \in E_\cap} \left(I_{cour}(\mathbf{PI} + \delta) - \overline{I_{cour}} \right)^2}} \tag{2.21}$$

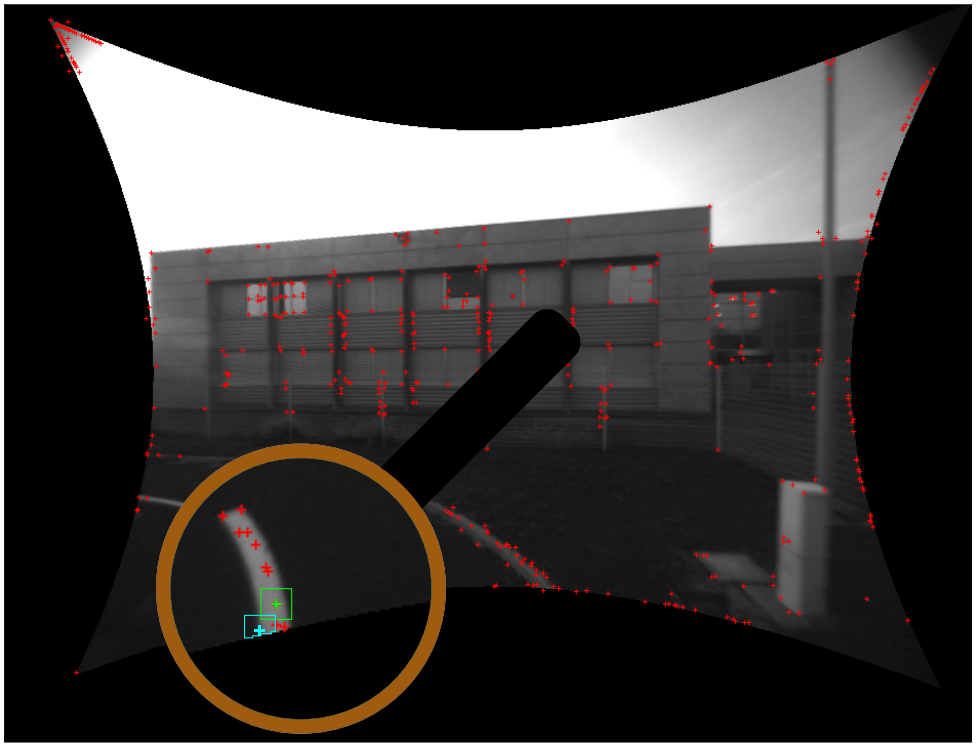


FIGURE 2.12 – Exemple de points détectés sur une image courante. On peut voir dans le zoom deux exemples de points d'intérêt avec le voisinage considéré lors de l'appariement.

$$\text{avec} \begin{cases} E_\cap = E_{patch} \cap E_{PI} & \\ \overline{I_{cour}} = \dfrac{1}{|E_\cap|} \displaystyle\sum_{\delta \in E_\cap} I_{cour}(\mathbf{PI} + \delta) & \text{la moyenne de l'intensité des pixels de } E_\cap \text{ sur l'image courante} \\ \overline{\Pi_{pred}} = \dfrac{1}{|E_\cap|} \displaystyle\sum_{\delta \in E_\cap} \Pi_{pred}(\mathbf{PI} + \delta) & \text{la moyenne de l'intensité des pixels sur la projection du patch restreinte à } E_\cap \end{cases}$$

Cependant cette équation oblige à faire tous les calculs séparément pour tous les couples de points. En effet, même les moyennes sur l'image sont restreintes au voisinage $E_\cap$ qui varie d'un couple à l'autre. Ainsi même lorsqu'un point ou un patch est présent dans plusieurs couples, toutes les moyennes doivent être calculées plusieurs fois, ce qui alourdit le calcul. Néanmoins, on sait que, même si l'homographie a déformé la texture, les patchs reprojetés ont une forme proche d'un carré. En effet, comme on le voit sur la figure 2.11, bien que la projection soit bien déformée, le fait d'avoir réduit le voisinage à un carré de 16 pixels de coté permet d'avoir un voisinage E_{patch} peu différent de E_{PI}. On peut donc approximer l'équation 2.21 sous la forme

de l'équation 2.22

$$S = \frac{\frac{1}{|E_\cap|}\sum_{\delta\in E_\cap}\left[\left(\Pi_{pred}(\mathbf{P_{ppred}}+\delta)-\overline{\Pi_{pred}}\right)\left(I_{cour}(\mathbf{PI}+\delta)-\overline{I_{cour}}\right)\right]}{\sqrt{\frac{\sum_{\delta\in E_{patch}}\left(\Pi_{pred}(\mathbf{P_{ppred}}+\delta)-\overline{\Pi_{pred}}\right)^2}{|E_{patch}|}}\sqrt{\frac{\sum_{\delta\in E_{PI}}\left(I_{cour}(\mathbf{PI}+\delta)-\overline{I_{cour}}\right)^2}{|E_{PI}|}}} \tag{2.22}$$

avec
$$\begin{cases} E_\cap = E_{patch}\cap E_{PI} & \\ \overline{I_{cour}} = \frac{1}{|E_{PI}|}\sum_{\delta\in E_{PI}} I_{cour}(\mathbf{PI}+\delta) & \text{la moyenne de l'intensité des pixels du voisinage } E_{PI} \text{ sur l'image courante} \\ \overline{\Pi_{pred}} = \frac{1}{|E_{patch}|}\sum_{\delta\in E_{patch}} \Pi_{pred}(\mathbf{PI}+\delta) & \text{la moyenne de l'intensité des pixels de la projection du patch (sur } E_{patch}) \end{cases}$$

De cette manière le calcul de la moyenne d'un patch projeté ou du voisinage d'un point d'intérêt peut être séparé de l'appariement. Ce calcul initial correspond en fait à un descripteur que l'on calcule avant de réaliser l'appariement. On peut voir chacun de ces descripteurs comme des images formant un carré de 16 pixels de coté nommé D_{patch} et D_{PI} respectivement pour les patchs et les points d'intérêt. Ils sont calculés suivant les équations 2.23 et 2.24

$$D_{patch}(\mathbf{X}) = \frac{\Pi_{pred}(\mathbf{X})-\overline{\Pi_{pred}}}{\sqrt{\frac{\sum_{\delta\in E_{patch}}\left(\Pi_{pred}(\mathbf{P_{ppred}}+\delta)-\overline{\Pi_{pred}}\right)^2}{|E_{patch}|}}} \tag{2.23}$$

$$D_{PI}(\mathbf{X}) = \frac{I_{cour}(\mathbf{X})-\overline{I_{cour}}}{\sqrt{\frac{\sum_{\delta\in E_{PI}}\left(I_{cour}(\mathbf{PI}+\delta)-\overline{I_{cour}}\right)^2}{|E_{PI}|}}} \tag{2.24}$$

Connaissant ces deux descripteurs, le calcul du score d'appariement S devient alors :

$$S = \frac{\sum_{\delta\in E_\cap}\left[D_{patch}(\mathbf{P_{ppred}}+\delta)D_{PI}(\mathbf{PI}+\delta)\right]}{|E_\cap|} \tag{2.25}$$

En utilisant cette méthode, le calcul des descripteurs devient indépendant de l'appariement. Par conséquent, dans le schéma de la figure 2.10, le calcul de D_{patch} est réalisé dans la partie projection des patchs, et celui de D_{PI} dans la partie calcul des descripteurs de points.

Cet appariement utilise la ZNCC car il n'est pas nécessaire (et même pas souhaité) d'être indépendant du point de vue. En effet les patchs projetés doivent théoriquement être eux-même

adaptés au point de vue, et l'usage d'un descripteur tel que SIFT risquerait de créer une confusion entre plusieurs éléments proches (tels que deux fenêtres côte à côte) sans améliorer l'appariement. La seule difficulté de cet appariement peut provenir de changements de luminosité. La projection des patchs ne tient pas compte des évolutions de la luminosité ambiante et il est important de demeurer aussi robuste que possible à de tels changements. La ZNCC a l'avantage d'être indépendante à un changement affine de luminosité tout en restant assez rapide à calculer.

Élimination des mauvais appariements

Une fois le score calculé pour tous les couples, ils sont d'abord seuillés pour filtrer tous les mauvais appariements. Ce seuillage n'a pas besoin d'être très strict (0,5 lors de nos tests), car il consiste seulement à supprimer les appariements nettement mauvais pour alléger les calculs. Par la suite un tri plus spécifique est réalisé. L'objectif de ce tri est d'éliminer tous les cas où soit un patch, soit un point est présent dans plusieurs paires. Pour cela, tous les couples sont triés dans l'ordre de leur score d'appariement. Ensuite, tous les couples contenant le même patch ou le même point d'intérêt que celui ayant le meilleur score sont éliminés. Puis, on applique à nouveau le même procédé mais en éliminant les couples ayant un conflit avec le second meilleur score. On continue ainsi de suite jusqu'à avoir parcouru toute la liste. A la fin du processus, on obtient une liste telle que chaque point d'intérêt ne puisse être apparié qu'avec un seul patch et chaque patch ne puisse l'être qu'avec un seul point d'intérêt.

Au final on obtient une liste associant les coordonnées 2D des points d'intérêt aux coordonnées 3D des patchs. C'est cette liste qui, après un calcul de pose permet de déterminer la position précise de la caméra courante.

2.5 Premières expérimentations

Avant d'utiliser cette méthode de localisation sur les véhicules, il est intéressant de vérifier que cette reprojection apporte bien un avantage. Pour cela une expérimentation préliminaire a été réalisée. L'objectif de cette expérimentation est de comparer l'utilisation de patchs à une approche classique utilisant un descripteur SIFT, en particulier lorsque l'on s'éloigne de la position initiale. L'idée est donc de mesurer le nombre d'appariements que l'on peut réaliser en utilisant les patchs et de le comparer avec les appariements réalisés en utilisant le descripteur SIFT.

2.5.1 Séquence utilisée

L'objectif principal de la localisation par patch est d'utiliser des amers plans, tels que ceux d'une façade. On considère donc comme scène un poster, ce qui permet par ailleurs de vérifier qualitativement le calcul des normales. La prise de vue est donc réalisée en déplaçant une caméra autour du poster. On obtient alors une séquence vidéo de 45 images, représentant le poster

sur de nombreuses vues différentes, dont certaines sont présentées figure 2.13. Pendant la prise de vues, la caméra était posée sur un trépied et parcourait un arc de cercle autour du poster. Ensuite, une quinzaine d'images ont été prises en faisant bouger la caméra dans tous les sens.

La caméra étant calibrée, toutes les expérimentations ont été réalisées en partant directement des images corrigées, afin de ne pas être perturbées par la distorsion de l'image. Toutes les images montrées dans cette partie (incluant celle de la figure 2.13) seront donc des images corrigées.

L'objectif de l'expérimentation étant d'évaluer la capacité d'appariement de notre méthode, les positions 3D des points ainsi que les différentes poses de la caméra n'ont pas été calculés par la cartographie décrite précédemment. En effet la méthode de reconstruction utilisant des appariements préalables, cela aurait pu biaiser le résultat. La méthode utilisée pour obtenir la référence, c'est-à-dire les différentes poses de la caméra et la position 3D des points recherchés a été réalisée en utilisant 4 cibles de forme connues placées autour de l'image. Ces cibles décrites dans (Lébraly et al., 2010) peuvent être détectées automatiquement sur les images et la position de leur centre dans l'image est calculée précisément. A partir de ces 4 positions, on a pu déterminer la pose des caméras et les coordonnées 3D des cibles dans le repère monde. A partir de ce résultat, on a ensuite calculé l'équation du plan sur lequel repose les cibles et donc l'ensemble des points du poster. Chaque point détecté sur l'image étant également situé dans ce plan, il est alors possible de déterminer sa position 3D en utilisant conjointement l'équation du plan et la pose de caméra.

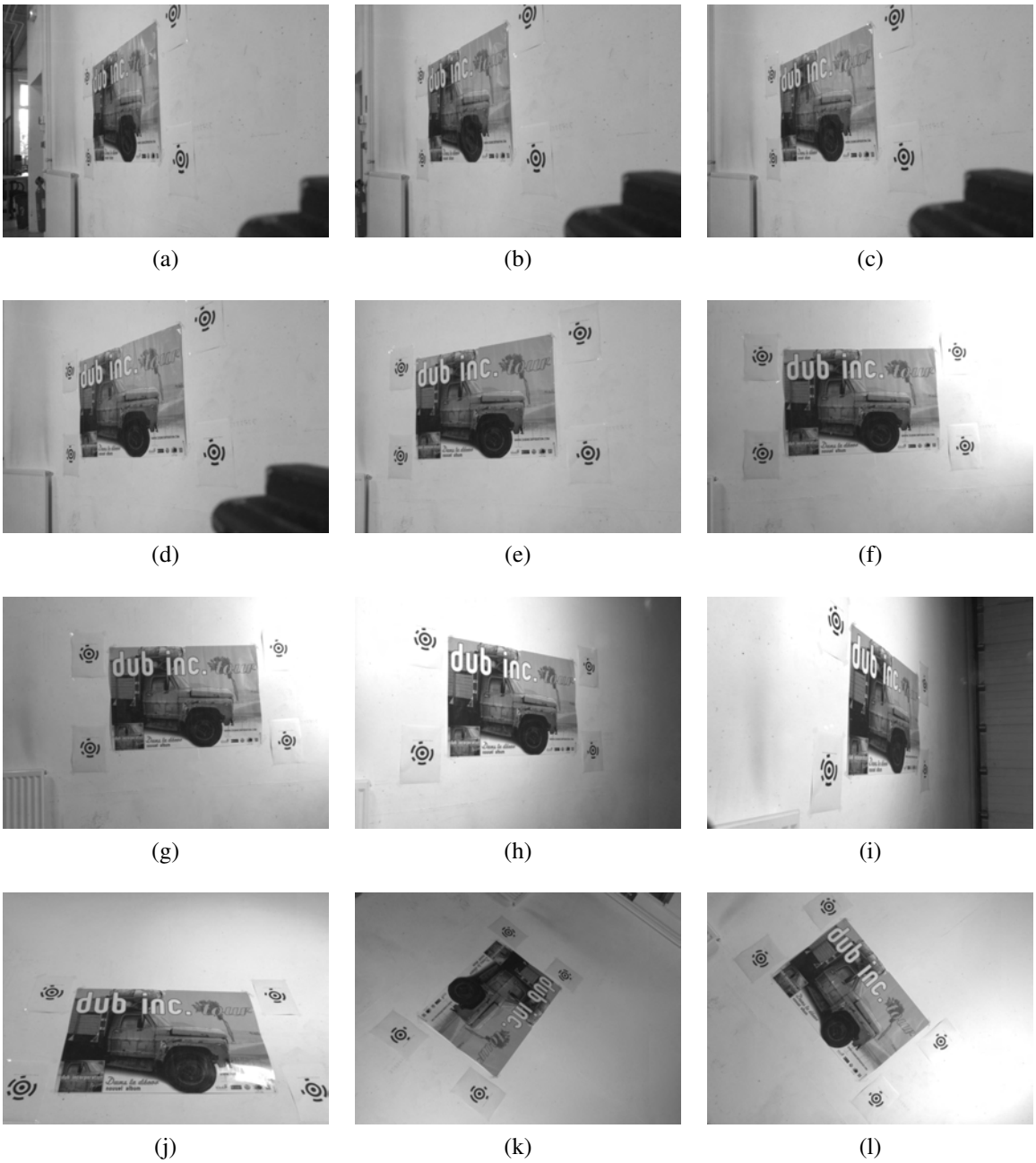
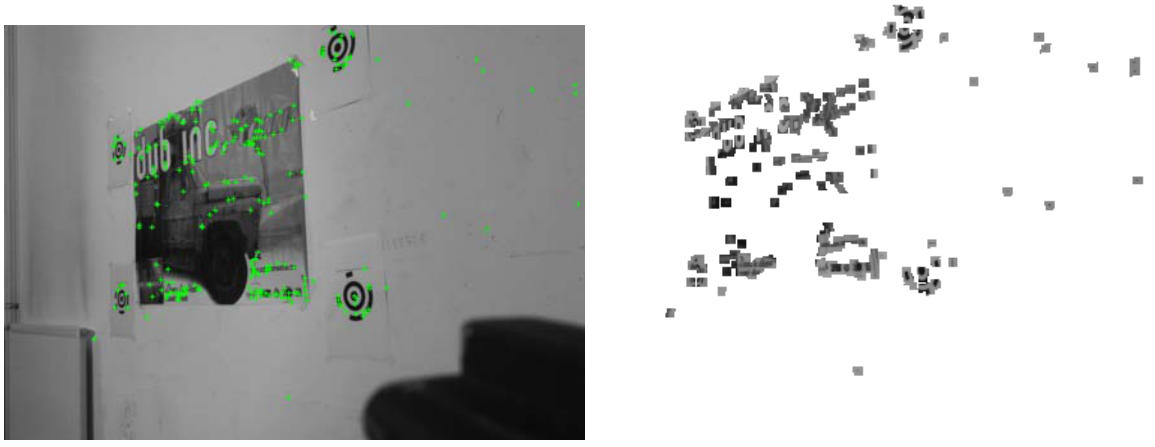


FIGURE 2.13 – Exemple d'images du poster utilisées pour les premiers résultats. Les premières images (de (a) à (i)) ont été enregistrées en réalisant un arc de cercle à hauteur constante. Les dernières images (de (j) à (l)) sont prises en tournant la caméra dans tous les sens. Les images (a) à (c) correspondent aux images de référence utilisées pour le calcul des patchs. L'image (c) est celle utilisée pour le calcul des descripteurs SIFT.

2.5.2 Calcul des points de références

Les points de référence sont l'ensemble des points que l'on essayera d'apparier avec les autres images. Pour les calculer, on a choisi de se limiter aux 3 premières images représentées figure 2.13a à 2.13c.

L'image 2.13c étant la plus proche des autres images de la séquence c'est sur celle-ci que seront calculés les descripteurs SIFT. Les deux autres images seront donc utilisées uniquement pour calculer la normale et la texture des patchs. Le détecteur associé au SIFT, (dont l'implémentation est fournie par l'auteur), est utilisé pour déterminer les points 2D de référence sur l'image avec leur facteur d'échelle. Cela permet de calculer les descripteurs SIFT associés à chacun de ces points. De plus, connaissant l'équation du plan et la pose des caméras, les coordonnées 3D de ces points peuvent être calculées et utilisées pour les projetés sur les autres images de référence. Lorsque le détecteur trouve un point 2D a proximité de la reprojection d'un point de référence, on considère cette détection comme étant une autre vue du point 3D de référence. Les points qui ont ainsi été suivis sur deux ou trois images de référence, sont utilisés pour initialiser l'algorithme de création des patchs. Cette méthode nous permet d'obtenir 257 points de référence dont on possède d'une part le descripteur SIFT et d'autre part un patch texturé généré à partir de deux ou trois vues. Ces points apparaissent sur la figure 2.14a, et les patchs associés sur la figure 2.14b.



(a) Points de référence détectés (b) Patchs de référence vu dans la pose de l'image

FIGURE 2.14 – Points de référence utilisés pour initialiser l'appariement

Afin d'avoir exactement les mêmes références pour les patchs et les descripteurs SIFT, lorsque sur la même image un même point est détecté avec plusieurs facteurs d'échelle, seul celui avec le plus gros facteur d'échelle est retenu. Cela permet de conserver un maximum de

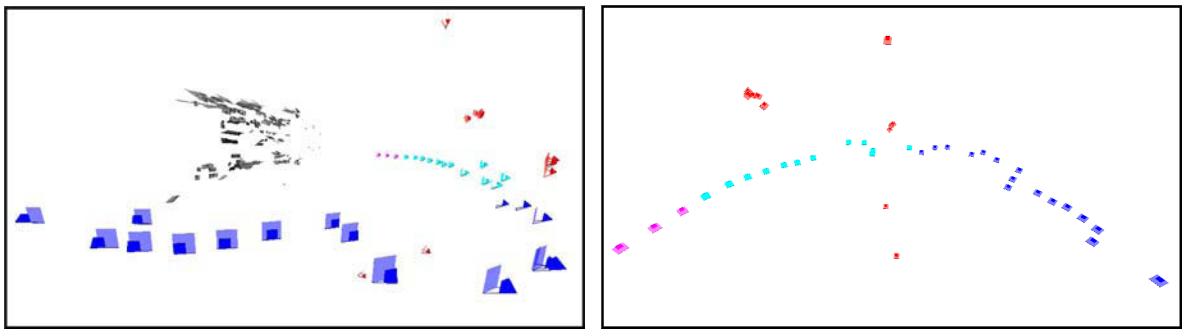
détails dans le descripteur SIFT.

2.5.3 Séquence de test

Une fois les références obtenues, on va utiliser les autres images de la séquence et tenter de suivre les points sur celle-ci. Ayant enregistré 45 images de test, on va diviser la séquence en trois parties :

- ➢ La première partie, assez proche de la référence, est composé de 12 images (dont la dernière est la figure 2.13f). La déformation par rapport à la référence reste assez faible.
- ➢ La seconde partie est composé des 16 images suivantes, qui forment la fin de l'arc de cercle autour du poster. Elle représente une plus forte déformation par rapport à la référence, mais sont prises dans des conditions proches d'une caméra sur un véhicule. Les figures 2.13g et 2.13h montrent respectivement la première et dernière image de cette partie.
- ➢ La dernière partie est composé des 14 images restantes où la caméra bouge de manière chaotique. Les images 2.13j à 2.13l en sont des extraits. Ces images présentent de grandes modifications par rapport à la référence, plus importantes que ce que l'on pourrait avoir dans un contexte de localisation.

On peut visualiser la position des caméras sur les images de la figure 2.15



(a) Vue d'ensemble des caméras et patchs reconstruits (b) Caméra de la séquence de test vue depuis le poster

FIGURE 2.15 – Position des caméras de la séquence de test par rapport au poster. Les caméras correspondant aux poses de référence sont en magenta. Pour l'apprentissage, les caméras de la première partie (la plus proche de la référence) sont en cyan, celles de la seconde partie en bleu foncé et la dernière partie en rouge.

2.5.4 Protocole expérimental

Pour déterminer le nombre d'appariements corrects de chaque point, on procède pour chaque image de la façon suivante. Tout d'abord le détecteur associé au SIFT est appliqué sur l'image

courante. Ainsi on obtient un certain nombre de points détectés qui pourront être appariés. Cependant, les points de référence que l'on recherche ne sont pas systématiquement détectés sur l'image. Le but ici étant d'évaluer le descripteur et non la détection, on ajoute artificiellement la projection de chaque point de référence aux points détectés. En effet, la position 3D des points de référence étant connue, on peut la projeter dans la pose de caméra courante (qui est aussi connue). Le facteur d'échelle associé à ce point, est également calculé en utilisant la distance entre la caméra et le point. On obtient alors pour une image une liste de points candidats à l'appariement contenant l'ensemble des bonnes correspondances. Comme pour les images de référence, on prend soin d'enlever tous les points qui seraient détectés plusieurs fois avec un facteur d'échelle différent. Cela permet de s'assurer que l'on a exactement les mêmes paires qui sont candidates pour l'appariement avec chaque méthode.

Pour réaliser l'appariement, les patchs ont besoin d'une prédiction de la pose de l'image. Celle-ci permet à la fois de reprojeter les patchs dans la vue courante et de définir une ROI afin d'éviter de rechercher toutes les paires possibles de l'image. Pour ne pas défavoriser SIFT, qui n'a pas besoin de prédiction, on utilise les ROI de la même manière pour les deux descripteurs. Ainsi on ne tente pas d'apparier un point avec une référence qui se projette de l'autre coté de l'image. De plus, lors de la localisation, la prédiction de la pose n'est qu'une approximation de la pose réelle. Pour simuler cela, la pose exacte est bruitée en appliquant une variation gaussienne sur la position et l'orientation de la caméra. C'est avec la pose bruitée (et la connaissance de l'écart-type du bruit) que l'on projettera les patchs et évaluera les ROI.

Ayant obtenu l'ensemble des paires possibles entre points d'intérêt et référence, plusieurs scores sont calculés :

- ➢ le score de SIFT en calculant la distance euclidienne entre le descripteur de la référence et celui du point de l'image ;
- ➢ le score d'une ZNCC entre le patch reprojeté et le voisinage du point d'intérêt (comme détaillé section 2.4.3) ;

Dans la description du SIFT, il est également recommandé d'appliquer une méthode pour augmenter la précision. Elle consiste à n'apparier une référence avec un point que si non seulement la distance entre descripteur est plus faible qu'avec les autres points, mais en plus que cette distance soit inférieure à 0,6 fois la seconde distance plus faible avec un autre point. Cette méthode est également utilisée et permet de diminuer le nombre d'appariements potentiels. L'intérêt est d'améliorer la précision d'appariement mais risque de réduire le nombre d'appariements réalisés. Cette méthode est donc appliquée aux paires utilisant le SIFT, et sera nommée SIFT2 lors des résultats.

Dans l'algorithme de localisation, une élimination des conflits est également mise en œuvre pour éviter qu'un même point ou une même référence soit présent dans deux paires distinctes. Cette étape est également réalisée en se basant sur chacun des scores obtenus. Les plus hauts scores sont conservés pour les appariements réalisés avec les patchs, et les plus faibles distances entre descripteurs pour les paires réalisées avec le SIFT. Il est également possible de comparer les différentes méthodes avant de réaliser cette étape. Cela permet alors de mesurer la capacité

des différentes méthodes à éliminer directement les mauvais appariements. La comparaison faite après cette étape montre la capacité des méthodes à avoir des meilleurs scores lors des bons appariements, ce qui n'est pas tout à fait la même chose.

Pour finir, chaque paire est évaluée comme correcte ou non en calculant la distance entre la projection du point de référence (en utilisant la pose non bruitée) et le point d'intérêt auquel il est apparié. Lorsque cette distance est trop importante (supérieure à 1 pixel lors de nos tests), la paire est considérée comme un mauvais appariement, dans le cas contraire ce sera un bon appariement.

2.5.5 Résultats

Une fois les différents appariements obtenus, on va utiliser une valeur de seuil sur le score de la méthode d'appariement. Les paires ayant un score moins bon que le seuil sont éliminées. Ensuite les paires restantes permettent de définir deux valeurs :

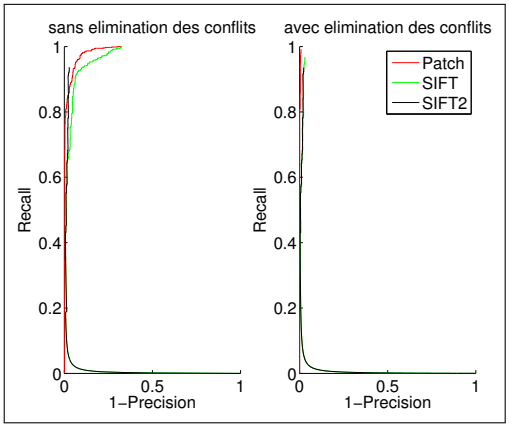
- $R = \frac{\text{nombre d'appariements corrects}}{\text{nombre de références présentes}}$, nommé *recall* qui représente la proportion d'appariements trouvés.
- $(1 - P) = \frac{\text{nombre faux appariements}}{\text{nombre d'appariements réalisés}}$, nommé *1 - Précision* qui représente la proportion d'erreur dans les appariements trouvés.

On fait varier le seuil à toutes les valeurs possibles pour chaque méthode. On obtient ainsi un ensemble de paires de valeurs R et $(1 - P)$ pour chaque méthode. En reportant toutes les valeurs pour tous les seuils dans un même graphe, où l'on place R en ordonnée et $(1 - P)$ en abscisse, on obtient une courbe ROC qui permet de comparer les différentes méthodes de manière indépendante du seuil. Un descripteur sera le meilleur si sa courbe est la plus en haut (R élevé) et à gauche (peu de mauvais appariements). Généralement un seuil très strict permet d'avoir très peu de mauvais appariements, mais également un recall faible. A l'inverse un seuil très large maximisera le recall au détriment de la précision.

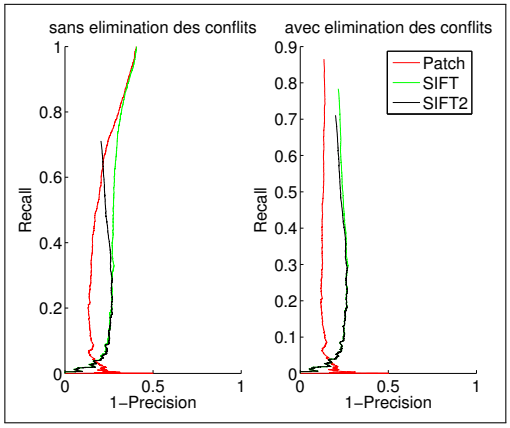
Cas de prédiction idéale

Tout d'abord, on peut observer (figure 2.16) les courbes lorsque l'on ne bruite pas la pose prédite. Les points sont alors projetés directement à la bonne position et les ROI étant très réduites, très peu de mauvais appariements peuvent être réalisés. On s'aperçoit que l'utilisation de patchs a tendance à apporter une meilleure précision qu'avec les autres descripteurs, plus particulièrement après l'élimination des conflits. Cette différence est d'autant plus marquée lorsque l'on est sur des images bien différentes des références (figure 2.16c et 2.16d), puisque c'est là que l'adaptation du patch a le plus d'intérêt.

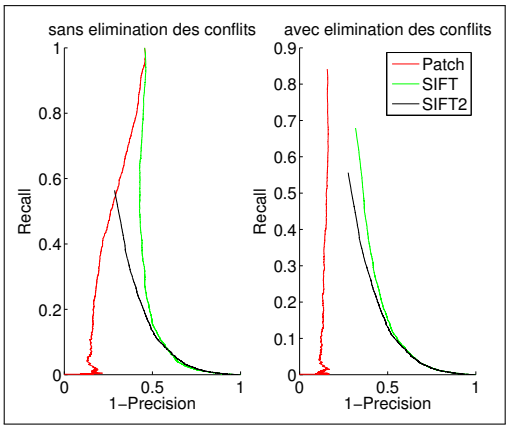
Cependant, ces premiers tests sont clairement avantageux pour le patch puisqu'ils possèdent la pose exacte de l'image. Il faut donc voir son comportement lorsque la prédiction est bruitée, et que les patchs ne se projettent pas exactement avec la même apparence que la cible.



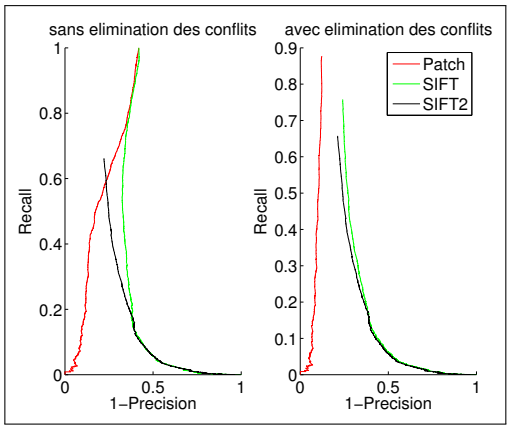
(a) Image de référence



(b) Images de la première partie



(c) Images de la seconde partie

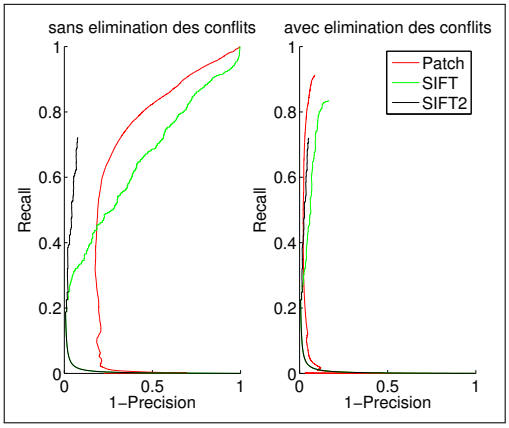


(d) Images de la troisième partie

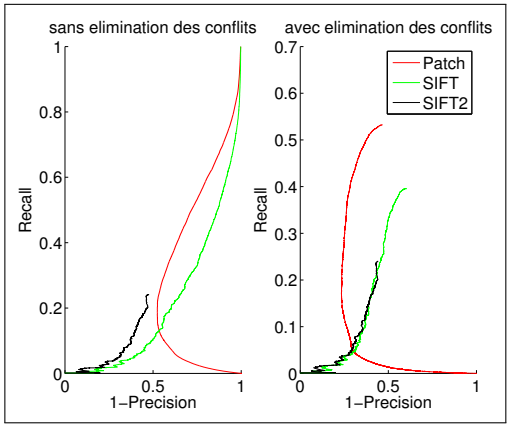
FIGURE 2.16 – Résultat des comparaisons sur le poster sans bruit.

Cas de prédiction réaliste

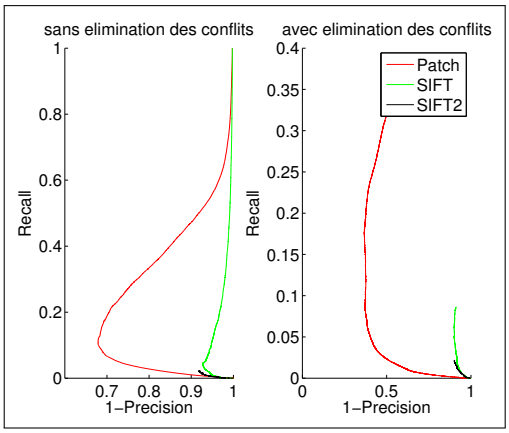
Sur la figure 2.17, on a les mêmes résultats que précédemment mais en utilisant des poses un peu plus bruitées, comparables à ce qu'on devrait avoir lors d'une localisation. On s'aperçoit tout d'abord que sur les poses de référence (figure 2.17a) et celle de la première partie (figure 2.17b), la méthode des patchs est moins précise que SIFT particulièrement lorsque l'optimisation associée est utilisée. Cependant, les patchs parviennent à avoir un *recall* plus important, et donc à conserver la majorité des appariements. Lorsque l'on s'éloigne de la référence, avec les parties 2 et 3 (figure 2.17c et 2.17d), la méthode SIFT devient considérablement moins précise. En effet, en augmentant le bruit de position on a également augmenté la taille des ROI. Ceci implique que de nombreux appariements supplémentaires sont réalisés et que les différentes méthodes ne parviennent pas toujours à les différencier. Lorsqu'on applique l'élimi-



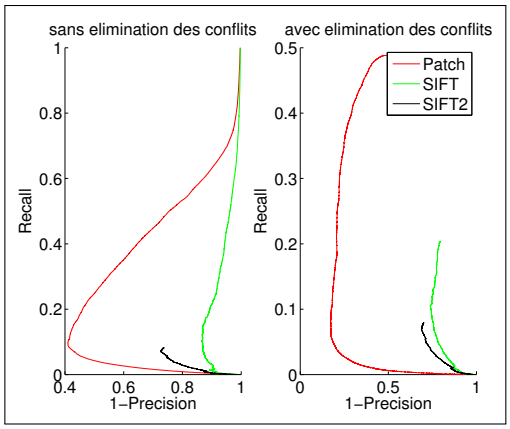
(a) Image de référence



(b) Images de la première partie



(c) Images de la seconde partie



(d) Images de la troisième partie

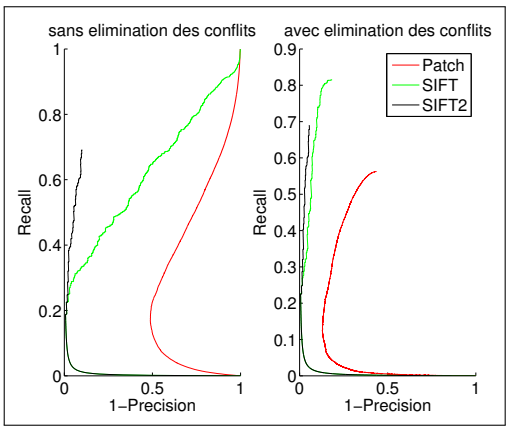
FIGURE 2.17 – Résultat des comparaisons sur le poster avec un bruit de 100 mm et 1 degré

nation des conflits, la précision augmente globalement mais a davantage d'effet sur les patchs, qui obtiennent ainsi un bien meilleur *recall* et une meilleure précision que les autres. Au bilan on s'aperçoit que le bruit imposé n'a que faiblement limité l'erreur d'appariement des patchs. De plus lorsque l'on s'écarte de la référence les patchs permettent d'atteindre un recall beaucoup plus important, c'est-à-dire qu'un maximum d'appariements peuvent être réalisés dans ces conditions.

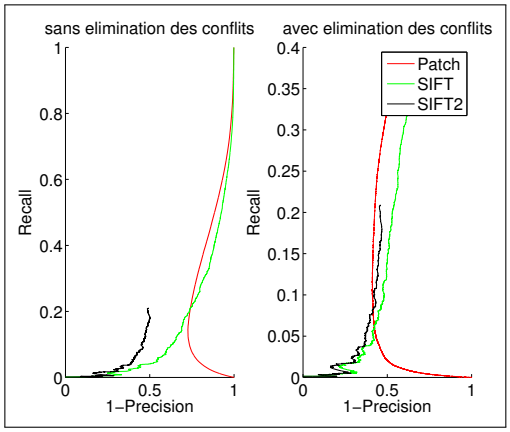
Cas de prédiction fortement bruitée

Pour s'approcher des limites de l'algorithme on a également essayé d'augmenter énormément le bruit de la prédiction, bien au-delà de l'erreur de prédiction que l'on aura lors de la localisation. Cela implique que les ROI soient assez grandes pour recouvrir pratiquement toute

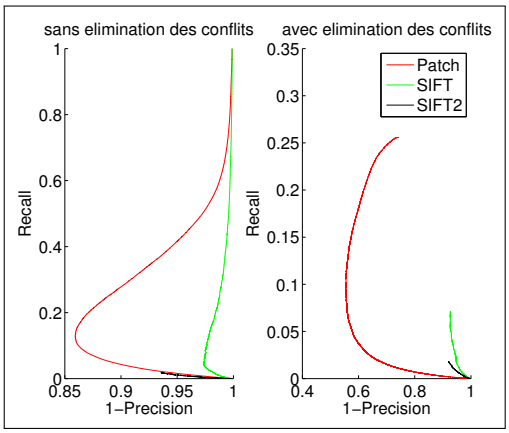
l'image, et les patchs auront vraisemblablement une apparence assez différente de la vue courante. On obtient alors les graphes de la figure 2.18.



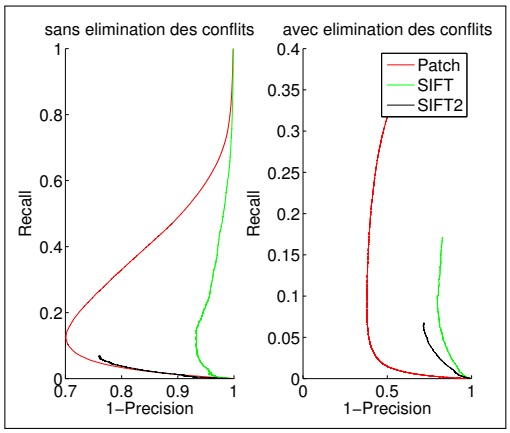
(a) Image de référence



(b) Images de la première partie



(c) Images de la seconde partie



(d) Images de la troisième partie

FIGURE 2.18 – Résultat des comparaisons sur le poster avec un bruit de 500 mm et 10 degrés

Lorsque les images sont assez proches de la référence (figure 2.18a), les patchs reprojetés sont clairement moins précis que le SIFT. Toutefois sur les images plus éloignées des parties 2 et 3 (figure 2.18c et 2.18d), ils semblent conserver une précision meilleure que les autres méthodes. Même avec un bruit important, la projection des patchs dans une vue proche de l'image de test suffit à améliorer les performances. Il faut toutefois nuancer ces résultats car lorsqu'on voit l'échelle de la précision sur les courbes, on s'aperçoit que même après élimination des conflits plus de 40% des appariements sont erronés. Il n'est donc pas garanti qu'un calcul de pose réalisé à partir de ces appariements pourrait être correct.

2.6 Conclusion

Le calcul de normales permet d'avoir des amers beaucoup plus riches que les simples amer-point. On a d'ailleurs pu constater, en le comparant avec le SIFT, que cela permet de s'éloigner grandement de la trajectoire en conservant une meilleure mise en correspondance. L'amélioration est particulièrement sensible en terme de nombre d'appariements réalisés. Cela pourra donc permettre, lors de la localisation de conserver suffisamment d'appariements pour pouvoir localiser le véhicule malgré des écarts importants à la référence. De plus, avec sa capacité à s'adapter à la vue courante, il est assez peu probable de confondre un appariement avec un autre.

Il est cependant important de noter que l'utilisation de patchs nécessite une prédiction de la localisation aussi précise que possible. En effet lorsque la prédiction est très imprécise, de nombreux faux appariements sont réalisés. Cet inconvénient reste toutefois peu gênant dans le cadre d'une localisation, car le véhicule suit une trajectoire connue que l'on peut facilement prédire.

Tout semble donc indiquer que l'utilisation de patchs est une méthode très prometteuse pour la navigation autonome. Néanmoins, cela nécessite des calculs de projection sur chaque image qui sont assez lourds à mettre en œuvre. Afin d'atteindre des performances temps réel, il est donc indispensable d'améliorer les performances de l'algorithme et son implémentation.

Chapitre 3

Implémentation GPU

3.1 Analyse de la durée d'exécution

L'algorithme a tout d'abord été conçu sans se préoccuper de la contrainte temps réel. En effet, l'utilisation de nombreux patchs et leur reprojection un à un est assez lourde en temps de calcul et les moyens tels que la programmation sur GPU sont connus pour résoudre les problèmes de ce type.

Lors des premières exécutions, la localisation du véhicule prend plusieurs secondes à chaque image sur une machine offrant des performances dans la moyenne. Cela rend donc cet algorithme inutilisable en l'état dans un cadre de navigation autonome. Il est en effet impossible de commander un véhicule si sa position n'est connue que 3 secondes après avoir acquis une image sans pouvoir en traiter plus d'une toutes les 3 secondes. Il est donc nécessaire, en travaillant sur l'implémentation elle-même, de parvenir à une accélération du processus. Afin d'améliorer spécifiquement les parties les plus longues, une analyse du temps d'exécution du programme est réalisée. Les temps de calcul sont mesurés avec l'horloge de la machine avec une précision de l'ordre de la microseconde. L'ordinateur a un processeur Intel core quad cadencé à 2,4 GHz, et dispose de 4 Gio de mémoire RAM. L'implémentation n'utilise cependant qu'un seul cœur du processeur. La localisation a été exécutée sur une séquence de 1 140 images en utilisant 15 683 patchs et le temps de chaque itération a été mesuré. Les résultats sont présentés dans le tableau 3.1.

La partie la plus gourmande en calcul est la projection des patchs. Une analyse plus fine de l'exécution montre que c'est plus précisément le processus d'interpolation bilinéaire qui consomme la plupart du temps. Une autre partie qui nécessite presque un tiers du temps est le calcul des descripteurs de PI. Cette partie consiste en fait à extraire une vignette de l'image courante, et à lui soustraire la moyenne et la diviser par l'écart-type. Cette partie nécessite beaucoup de temps parce que prenant des valeurs extraites d'une image, il est difficile d'avoir une mémoire cache assez performante. En effet les changements de ligne avant d'avoir atteint l'extrémité de l'image causent généralement des défauts de cache c'est-à-dire des accès à des

	Moyenne (ms)	Médiane (ms)	Écart-type (ms)	Maximum (ms)	Temps écoulé (%)
Correction d'image	14,95	14,94	0,14	16,97	0,41
Détection des points	30,57	30,39	1,00	33,12	0,83
Sélection des patchs visibles	1,96	1,90	0,26	2,57	0,05
Calcul des ROI	21,94	20,82	6,12	36,76	0,59
Projection des patchs	2403,16	2252,29	706,75	4057,52	65,11
Calcul des descripteurs de PI	1099,75	1197,21	168,54	1614,22	29,80
Appariement	116,49				3,16
Total	3690,83	3497,17	834,26	5581,49	100

TABLE 3.1 – Profiling du code sur CPU. Les couleurs et noms des fonctions correspondent à celle de la figure 2.10. Le calcul du score et des descripteur de PI ayant été réalisé simultanément, il n'a pas été possible d'en déterminer leur valeur à chaque itération. Toutefois la valeur moyenne par image a pu être évaluée séparément. Les parties de l'algorithme mettant moins de 1 ms par itération ont été ignorées

données non présentes dans le cache. De plus il s'agit d'une opération qui est exécutée de manière identique sur tous les pixels du voisinage d'un point. L'implémentation initiale nécessite un parcours de chaque image mais pourrait être parallélisée. Pour finir, ces deux étapes sont appliquées sur l'ensemble des points détectés ou des patchs visibles. Là encore on applique donc de nombreuses fois la même opération sur des données légèrement différentes. Il semble donc intéressant d'appliquer une méthode de parallélisation à cet algorithme.

Pour toutes ces raisons l'implémentation sur GPU (*Graphic Processing Unit*) parait intéressante. En premier lieu, le fait de projeter des patchs correspond aux calculs pour lesquels ont été conçues les cartes graphiques. En effet, les cartes graphiques servent initialement à réaliser de la synthèse d'image 3D, c'est-à-dire projeter des primitives graphiques dans différentes vues. Elles disposent donc de tous les outils nécessaires pour réaliser des interpolations bilinéaires et des calculs de projection le plus efficacement possible. De plus, leur architecture permet une exécution massivement parallèle, c'est-à-dire qu'une même opération peut être exécutée simultanément sur de très nombreuses données. Cela est également intéressant pour traiter des algorithmes sur chaque pixel et sur chaque patch de manière simultanée.

3.2 Etat de l'art

Avec le développement considérable des cartes graphiques depuis quelques années, de nombreux chercheurs ont tenté d'implémenter leurs algorithmes sur GPU. Cette utilisation de GPU à des fins diverses autres que l'affichage est appelée *general-purpose GPU* et a été explorée dans de nombreux domaines. Les différentes méthodes de programmation sur GPU et des implémentations réalisées sont résumées par (Owens et al., 2007). L'auteur explique en particulier les différentes spécificités de cette programmation et le travail nécessaire pour adapter un algorithme existant à cette architecture quel que soit le domaine d'utilisation.

Dans le cadre plus restreint de la vision par ordinateur, les descripteurs les plus courants ont également été implémentés sur GPU. En particulier le descripteur SIFT et son détecteur associé ont été implémentés dans (Sinha et al., 2006). Moyennant un certain nombre d'adaptations de l'algorithme, cette implémentation sur GPU permet un gain de temps de calcul important par rapport à sa version sur CPU. Cette première implémentation sur GPU utilise la librairie OpenGL et la programmation de shader. Par la suite une autre version de SIFT sur GPU (Heymann et al., 2007) divise par 5 le temps de calcul en passant d'une implémentation sur CPU optimisée avec le jeu d'instruction SSE (*Streaming SIMD Extension*) à une implémentation GPU.

Le descripteur SURF, déjà plus rapide que SIFT en version CPU, a également été implémenté en GPU par (Cornelis and Van Gool, 2008). Au lieu d'utiliser la librairie graphique OpenGL comme précédemment, le langage CUDA créé par Nvidia et la librairie CUBLAS associée a permis de faciliter la programmation. Cela lui permet d'obtenir des performances bien meilleures que les implémentations de SIFT précédentes.

Plus récemment (Schweitzer and Wuensche, 2009) une méthode d'appariement complète (calcul des descripteurs et mise en correspondance) sur GPU a été implémentée, toujours en utilisant CUDA. De nombreuses modifications de l'algorithme initial ont néanmoins été nécessaires afin de rester aussi efficace que possible sur la nouvelle architecture. Cependant les performances finales demeurent identiques à la version sur CPU. Au niveau du temps de calcul, l'ensemble de l'algorithme parvient à être exécuté en 3 ms par image. De plus aucune instruction n'est exécutée sur le CPU permettant éventuellement à d'autres programmes de tourner en parallèle.

Tous ces travaux ont montré qu'il est possible d'obtenir de très bonnes performances en implémentant un algorithme sur GPU. Cependant, cette conversion d'une version CPU linéaire à une architecture SIMD (Single Instruction, Multiple Data) n'est pas évidente. Les algorithmes eux-mêmes nécessitent généralement d'être repensés, optimisés et parfois même un peu modifiés pour tenir compte des caractéristiques de la nouvelle architecture.

3.3 Description des outils

Avant de pouvoir détailler l'implémentation réalisée sur GPU, il est nécessaire de bien comprendre le fonctionnement de cette architecture. Cette partie introduit donc tout d'abord ses spécificités avant de décrire les outils utilisés et le vocabulaire associé.

3.3.1 Architecture

Les cartes graphiques qui équipent maintenant tous les ordinateurs ont initialement été réalisées pour pouvoir afficher des données complexes issues de scène 3D. Il s'agit donc de transformer une liste de coordonnées 3D (les *vertex*) organisées sous forme de triangles que l'on doit projeter dans une vue spécifique. La position de chaque vertex est connue indépendamment des autres et ne requiert finalement que peu de mémoire. La carte graphique applique les équations de projection à chaque coordonnée pour savoir où placer chaque vertex. Ensuite, il faudra déterminer la couleur à appliquer à chaque pixel. Pour cela on peut utiliser une texture ou une couleur qu'il faudra interpoler pour déterminer la valeur en chaque point.

La carte graphique doit donc, pour réaliser ces tâches au mieux, être capable d'exécuter une même instruction (la reprojection) sur énormément de données indépendantes en même temps.

L'architecture des cartes graphiques a évolué différemment des processeurs CPU classiques pour se spécialiser dans cette tâche, comme illustré sur la figure 3.1. Les CPU sont capables de

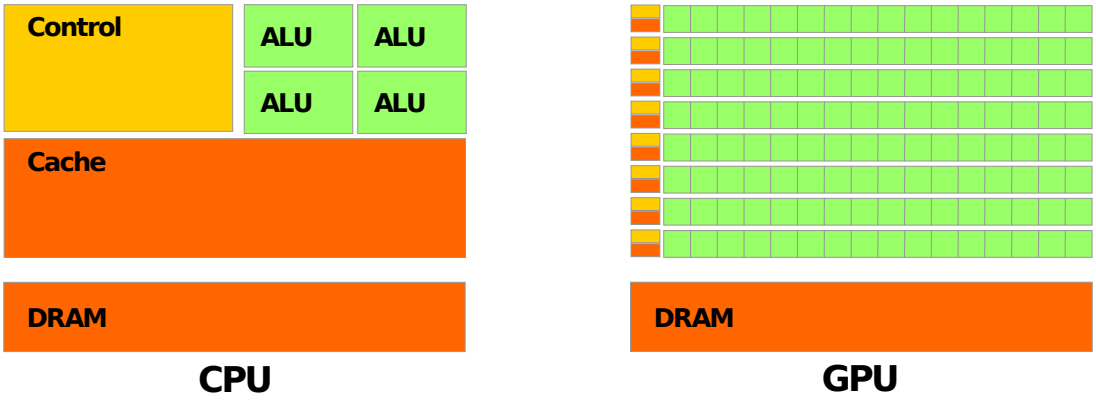


FIGURE 3.1 – Comparaison des architecture des CPU et des GPU. Les unités de calculs (nommé ALU pour *Arithmetic and Logical Unit* en vert) sont beaucoup plus nombreuses sur un GPU au détriment de la mémoire cache et des unité de contrôle (en orange). Image extraite de (Nvidia, 2011)

gérer de nombreuses tâches différentes en parallèle avec beaucoup de données. Le GPU, lui, est contraint de réaliser la même tâche sur de très nombreuses données en parallèle. C'est cela qui lui vaut le qualificatif de massivement parallèle.

En pratique la capacité de calcul est répartie en plusieurs (généralement 4 à 16) multiprocesseurs se décomposant eux-même en de nombreux cœurs (8 à plus de 48). Cela permet d'avoir

énormément de calculs en parallèle, beaucoup plus que sur un CPU qui ne possède actuellement guère plus de 8 cœurs. Toutefois, ces nombreux cœurs disponibles ne peuvent être utilisés que tous ensemble, et ainsi chaque multiprocesseur est contraint d'exécuter la même opération sur tous ses cœurs. De plus l'accès aux données n'est pas très facile, et les ensembles de valeurs sur lesquelles travaille chaque cœur doivent être alignés en mémoire de manière à ce que le énième cœur travaille sur la énième valeur.

D'autre part les cartes graphiques ont souvent besoin de déterminer quelle couleur placer sur chaque pixel en se basant sur des textures ou des images à déformer. Pour répondre à ce besoin, des accès mémoire spéciaux ont été mis en œuvre telle que la mémoire de texture. Celle-ci permet d'utiliser des méthodes d'interpolation câblées directement dans la carte, qui offrent de très bonnes performances pour réaliser ces opérations. De plus l'accès à ces textures doit également se faire de manière bidimensionnelle, car il est courant de n'extraire qu'une partie d'une image. Un système de cache adapté à une utilisation 2D a donc été mis en place, évitant des défauts de cache lorsqu'on change de ligne dans l'image.

Ces différentes particularités semblent donc intéressantes pour notre utilisation. Toutefois, exécuter un programme sur la carte graphique nécessite un certains nombre d'outils bien particuliers.

3.3.2 Outils de programmation

Pour réaliser la programmation sur les cartes graphiques il est nécessaire d'utiliser certains outils qui donnent accès à la carte graphique. La librairie OpenGL par exemple permet d'exécuter certaines portions du code sur la carte graphique. Cependant il s'agit d'un outil de synthèse d'image et détourner son utilisation à des fins autres que de l'affichage n'est pas forcément très facile. Pour pallier ce défaut, le constructeur de carte graphique Nvidia a créé un langage nommé CUDA (*Compute Unified Device Architecture*) très semblable au langage C++ permettant d'exécuter un programme sur une carte graphique sans nécessairement réaliser un affichage. La plupart des outils présents sur les cartes graphiques créées par Nvidia sont accessibles avec cet outil. Les différences fondamentales par rapport aux langages de programmation courants sont liées à l'utilisation des spécificités de la carte graphique telles que la parallélisation massive et les différents types de mémoire. C'est avec cet outil que l'implémentation GPU de notre algorithme a été réalisée. Un dernier outil mis au point plus récemment est appelé OpenCL. Cet outil se présente sous la forme d'une librairie C++ offrant les mêmes possibilités que le langage CUDA. Toutefois, étant développé par un consortium de plusieurs constructeurs, cet outil présente l'avantage de pouvoir être utilisé sur toutes formes de cartes graphiques, sans se limiter à celle d'un constructeur. Du fait de son développement assez récent, nous n'avons pas utilisé cet outil. Toutefois, possédant les mêmes spécificités que CUDA, il serait envisageable d'adapter à l'avenir notre implémentation à OpenCL.

Pour utiliser les différents outils de la carte graphique, il est nécessaire de définir plusieurs notions propres à ce genre de système.

Parallélisation massive

La première notion essentielle est le *kernel*. Comme décrit précédemment, le GPU est conçu pour exécuter un même programme sur un ensemble de données. Pour qu'un même programme soit exécuté simultanément, il est nécessaire de le découper en multiples exécutions (on parle de *thread*) qui réalisent les mêmes instructions mais sur des données différentes. La liste d'instructions exécutées par tous les threads est appelée *kernel*.

Par exemple si on réalise la somme de deux vecteurs à N coordonnées, chaque thread peut réaliser la somme pour une coordonnée. Le kernel sera donc simplement le programme qui lit une coordonnée dans chaque vecteur, additionne les deux valeurs et écrit le résultat. Le programme nécessitera donc N threads pour réaliser la somme complète.

Pour écrire le kernel, il est nécessaire, en plus du code exécuté de savoir combien de threads vont être exécutés. De plus il est nécessaire d'organiser ces threads pour que le programme bien qu'identique, puisse facilement sélectionner les données que chaque thread va traiter. Pour cela, les threads sont regroupés sous forme de `blocs`. Chaque thread connait ses coordonnées au sein de ce bloc. Ainsi, si on reprend l'exemple de la somme de vecteurs, on peut dire que chaque bloc de coordonnées X, se chargera de la coordonnée numéro X. De cette manière, bien que le kernel soit écrit une seule fois et soit parfaitement le même pour tous les threads, chacun d'entre eux s'exécutera sur une donnée différente. Dans la programmation CUDA, les blocs peuvent être des structures à trois dimensions, et chaque thread possède donc 3 coordonnées.

Au niveau matériel chaque bloc sera exécuté sur un même multiprocesseur. Le multiprocesseur utilise l'ensemble de ses cœurs pour exécuter simultanément plusieurs threads. L'ensemble des threads exécutés simultanément est appelé un *warp* et possède une taille fixe pour chaque carte graphique, généralement de 32. Afin d'éviter d'avoir des cœurs au repos, il est préférable d'avoir des blocs d'une taille multiple de la taille d'un warp. Lorsque le nombre de threads dépasse la taille d'un warp, ils sont divisés en plusieurs warps et exécutés à la suite sur le même multiprocesseur. Cependant la carte graphique ne peut tout de même pas gérer trop de threads en attente. Ainsi il n'est pas possible d'avoir plus de 512 threads dans un même bloc. Cela peut sembler assez restrictif, par exemple lorsque l'on veut appliquer un processus sur une image complète de 512×384 pixels, avoir un thread par pixel est bien au-delà de cette limite. Pour pallier cette contrainte, on peut également avoir plusieurs blocs regroupés sous forme d'une *grille*. Si un bloc est toujours exécuté sur un même multiprocesseur (pouvant éventuellement partager des données entre les threads), ce n'est pas forcément le cas des différents blocs d'une même grille. En effet la plupart des cartes graphiques possèdent plusieurs multiprocesseurs. Lorsqu'une grille contient plusieurs blocs, ceux-ci seront exécutés simultanément en utilisant plusieurs multiprocesseurs. De même que précédemment tous les threads d'une même grille exécutent le même kernel, mais chacun d'entre eux connait les coordonnées de son bloc au sein de la grille. La grille est une structure à deux (voire trois sur les cartes plus récentes) dimensions, pouvant contenir plusieurs milliers de blocs sur chaque dimension.

Pour résumer on peut considérer que chaque thread se différencie des autres à travers 5 coordonnées : 3 pour les coordonnées du thread dans le bloc, et 2 pour les coordonnées du bloc

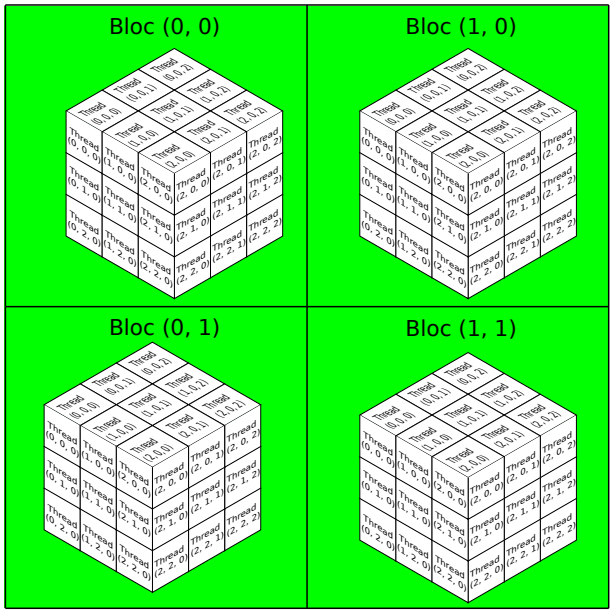


FIGURE 3.2 – Exemple d'organisation en grille et bloc. La grille en vert est composée de 2×2 blocs divisés en 3×3×3 threads

dans la grille. Le diagramme de la figure 3.2 donne un exemple de cette organisation des threads en bloc et grille.

Pour une application de traitements d'images opérant sur une image entière les threads sont généralement associés chacun à un pixel en conservant l'organisation bidimensionnelle de l'image. Chaque bloc représente un sous-ensemble de l'image, et la grille représente le découpage en sous-ensemble. Par exemple pour une image on pourrait prendre des blocs de 8×8 threads (nombre multiple de 32 et inférieur à 512), avec la dernière dimension non utilisée (fixée à 1), et la taille de la grille serait calculée à partir des dimensions de l'image (pour une image de 512×384 pixels, on aurait une grille de 64×48 blocs).

Il est important de garder à l'esprit que lorsque les threads sont plus nombreux que la taille d'un warp, ils seront divisés en plusieurs warps. La répartition des threads entre les warps est faite arbitrairement et il n'y a aucun moyen de savoir quels threads seront exécutés simultanément. De plus l'ordre d'exécution des warps est également inconnu. Il est tout à fait possible que le dernier thread d'un bloc soit exécuté avant le premier. Il est donc nécessaire de programmer les kernels en rendant chaque thread aussi indépendant des autres que possible.

Pour finir, les threads réalisant les mêmes opérations en même temps, les branchements conditionnels doivent être évités autant que possible. En effet, lorsqu'une condition contraint certains threads d'un même bloc à réaliser des opérations différentes des autres, les opérations seront réalisées successivement. Bien que le langage le permette, ce genre d'implémentation peut grandement nuire à l'efficacité de l'implémentation.

Utilisation de la mémoire

La mémoire embarquée dans le GPU est divisée en plusieurs parties en fonction de son usage. Il est important de bien choisir la mémoire adaptée à une application, car selon le cas son utilisation peut être plus ou moins rapide et plus ou moins contraignante.

La mémoire *globale* est la principale mémoire que l'on trouve dans la carte graphique. C'est là que sont transférées les données venant du CPU et également dans cette mémoire que les threads écrivent les résultats qui sont transférés vers le CPU. Elle est accessible par tous les threads en permanence. Cependant il est indispensable d'empêcher deux threads différents d'écrire au même endroit dans cette mémoire car les threads pouvant être exécutés dans n'importe quel ordre ou même simultanément, il est impossible de prévoir le résultat que l'on obtiendrait. L'accès à cette mémoire est assez lent, et doit être réalisé de manière alignée entre les threads. En particulier les threads doivent tous accéder à une valeur distincte et homogène de la mémoire, démarrant à une distance multiple de 2 octets du début de la mémoire. Sur certaines cartes, même l'ordre des threads a de l'importance, c'est-à-dire que le premier thread du bloc doit accéder à la première adresse mémoire. Lorsque cet alignement n'est pas respecté, les accès mémoire seront réalisés en plusieurs étapes, ralentissant considérablement le programme. Malgré ces contraintes, cette mémoire est la plus importante disponible (de l'ordre du gigaoctet).

L'accès à la mémoire *partagée* est beaucoup plus rapide mais le partage ne peut être fait qu'entre threads d'un même bloc. De plus, les informations de la mémoire ne sont pas conservées après la fin de l'exécution d'un kernel. Cette mémoire est donc principalement utilisée pour conserver des valeurs intermédiaires dans un calcul et les partager entre tous les threads d'un bloc. Sa taille est également très limitée (quelques kilooctets). Pour finir, l'accès à cette mémoire est assez contraignant. Elle est en effet divisée en *banques* de données. Chaque mot successif de 32 bits est placé dans une banque différente, avec un maximum de 32 (la taille d'un warp) banques. Chaque thread doit accéder à une banque différente, sans quoi l'accès mémoire est réalisé en plusieurs fois. Sur les cartes les plus récentes il est toutefois possible que plusieurs threads accèdent aux données d'une même banque s'ils accèdent au même mot de 32 bits.

Le dernier type de mémoire est la mémoire locale qui correspond au registre de chaque processeur. Cette mémoire est très rapide, propre à chaque thread mais assez réduite. La taille de cette mémoire, de quelques kilooctets est répartie entre tous les threads d'un bloc. Il est donc important de veiller à ne pas consommer trop de registres lors de l'exécution d'un kernel.

Ces différents types de mémoire correspondent à la mémoire physique de la carte graphique. la programmation CUDA donne accès à deux autres types de mémoire qui sont en réalité des emplacements particuliers de la mémoire globale. Ces emplacements possèdent un cache qui permet de rendre la mémoire plus rapide d'accès. Par contre ils ne peuvent pas être utilisés en écriture directement au sein d'un kernel.

La mémoire *constante* est accessible très rapidement par chaque thread. Comme son nom l'indique elle permet de stocker des valeurs constantes transférées directement depuis le CPU.

Son cache permet d'accéder rapidement à chaque valeur sans contrainte d'alignement des données.

La mémoire de *texture* est une partie de la mémoire globale qui peut être utilisée en lecture par chaque thread sans se préoccuper de l'alignement des données. Elle possède un cache 2D qui la rend particulièrement intéressante pour le traitement des images. L'écriture dans cette mémoire ne se fait qu'en copiant un ensemble de valeurs soit depuis le CPU soit depuis la mémoire globale du GPU. Chaque texture peut être définie avec 1, 2 ou 3 dimensions. Lorsque la texture a deux dimensions, il est possible d'accéder à n'importe quel élément, même à des coordonnées non entières. Dans ce cas, selon la configuration, la valeur du plus proche voisin ou le résultat d'une interpolation linéaire des valeurs alentours sera renvoyé. L'interpolation linéaire étant câblée dans la carte, l'accès à des coordonnées non entières est pratiquement aussi rapide que lorsque les coordonnées sont exactement sur un pixel.

Les flux de traitement (Stream)

En plus de la parallélisation massive lors de l'exécution d'un kernel, il est possible de paralléliser plusieurs processus simultanément. En particulier, il est possible de réaliser simultanément un transfert de mémoire entre CPU et GPU et l'exécution d'un kernel. Cela est d'autant plus intéressant que les transferts de mémoire vers le GPU sont généralement les opérations qui prennent le plus de temps. De plus, au-delà de la parallélisation au sein même du GPU, il est possible de réaliser d'autres opérations sur le CPU pendant que le GPU exécute un kernel.

Pour mettre en œuvre ce genre de parallélisation, il faut que lorsque l'on exécute un calcul sur un kernel ou un transfert de donnée, le GPU n'attende pas la fin de l'exécution pour redonner la main à l'utilisateur. Ainsi il est possible de lancer d'autres exécutions qui pourront se dérouler en parallèle. Cependant, certaines instructions doivent attendre que les premières se soient terminées pour pouvoir s'exécuter, par exemple quand elles attendent le résultat de l'une pour leur calcul. Pour pouvoir conserver cette séquentialité, la notion de *stream* est utilisée. Lorsque l'on souhaite exécuter un transfert ou un calcul qui sera exécuté en parallèle avec d'autres tâches, on le place dans un stream particulier. Toutes les fonctions qui devront être exécutées parallèlement sont appelées dans un stream différent. Les fonctions qui doivent attendre la fin d'un processus pour s'exécuter, seront appelées en utilisant le même stream que le processus précédent. Lorsque tous les appels sont réalisés, le driver du GPU lancera les différents transferts et calcul dès que possible quand les multiprocesseurs seront disponibles.

Il peut arriver également qu'un même processus doive attendre la fin de plusieurs streams pour s'exécuter. Comme il n'est pas possible qu'un même appel appartienne à plusieurs streams cela pourrait poser problème. Pour gérer ce cas, il est possible de définir des événements associés à un stream. On peut ainsi dire que, après l'exécution de multiples fonctions, un évènement sera levé pour un stream choisi. Si l'on place un évènement à la fin de tous les streams concernés, le processus pourra alors se lancer dès que les événements de chaque flux seront activés.

Il est donc possible d'avoir une gestion assez évoluée des différents processus qui tournent sur la carte graphique. Cela est particulièrement utilisé pour masquer les temps de transfert,

en exécutant les premiers calculs sur les premières données transférées en même temps que le transfert de la suite. Cela peut aussi être utilisé pour des tâches différentes, par exemple pour la gestion de l'image et des patchs, l'image peut être transférée de la caméra vers le GPU pendant que les patchs sont projetés dans la vue prédite. Sur les cartes graphiques les plus récentes, il est même possible d'exécuter plusieurs kernels simultanément.

L'ensemble de tous ces outils permet d'implémenter efficacement l'algorithme décrit dans le chapitre précédent.

3.3.3 Outils de communication entre threads

Bien que les threads doivent être aussi indépendants que possible, il est parfois nécessaire de les faire travailler ensemble, en particulier lorsqu'ils utilisent de la mémoire partagée.

La première chose à faire peut être de les forcer à s'exécuter en même temps. En particulier lorsqu'on désire utiliser une valeur qui a été calculée par un autre thread, il est nécessaire que chaque thread attende que les premiers calculs soient terminés avant de lire la mémoire partagée. Pour cela il est possible de créer des *barrières* qui obligent chaque thread à s'arrêter tant que tous les autres threads du même bloc n'ont pas atteint leur barrière. Ces barrières ne peuvent cependant bloquer que les threads du même bloc et il est donc indispensable que les blocs (même regroupés en grille) soient totalement indépendant les uns des autres. Lorsqu'il est indispensable qu'une opération soit exécutée entièrement avant une autre, il est alors nécessaire de réaliser des kernels distincts qui seront exécutés successivement.

Un autre moyen de communication entre les threads est l'utilisation de fonction *atomique*. Ces fonctions permettent de réaliser une fonction simple (incrémentation, addition, récupération du minimum ou maximum, etc) sur une ou plusieurs variables situées en mémoire globale ou partagée en garantissant qu'aucun autre thread n'a pu modifier la variable en même temps. Si deux threads appellent une fonction atomique sur une même variable en même temps, l'un devra attendre que l'autre ait fini le traitement atomique pour le faire à son tour. Pour cette raison l'utilisation de telles fonctions doit être fait avec parcimonie car elles peuvent provoquer certains ralentissements. De plus certaines cartes graphiques plus anciennes ne supportent pas les fonctions atomiques ou en tout cas, pas forcément sur la mémoire partagée.

3.4 Implémentation des patchs plan

Bien que l'algorithme soit déjà décrit, réaliser une implémentation sur GPU n'est pas chose facile. Il faut en effet trouver le moyen de mettre en œuvre tous les outils décrits précédemment pour être le plus efficace possible. De plus il est nécessaire de rester proche de ce qui a déjà été fait pour pouvoir réutiliser les parties existantes, telles que la commande du robot ou le calcul de pose. Ce travail d'adaptation a été réalisé sur la partie de localisation qui utilise les patchs. La

partie servant à générer les patchs-plan, conçue pour être exécutée dans une phase hors ligne, n'est pas modifiée.

Dans un premier temps, il est nécessaire de concevoir le schéma global du programme, les différentes parties qui le composent et la façon dont elles s'agencent. Par la suite chaque partie du programme sera décrite, pour montrer les mesures choisies pour l'utilisation de la mémoire et la répartition des threads.

3.4.1 Schéma global

Avant tout, il est nécessaire de choisir l'organisation du programme. Dans un premier temps il a semblé essentiel de conserver certaines étapes sur le CPU. Le calcul de pose et la prédiction, pour commencer, car il s'agit d'étapes nécessitant peu de temps dans le profiling initial, et qui peuvent à l'avenir évoluer. De plus il est possible, en particulier pour la prédiction, d'utiliser les informations d'autres capteurs, et ces informations ne sont pas disponibles directement sur le GPU.

Au delà de la répartition entre CPU et GPU, il est intéressant de s'interroger sur la répartition entre les différents streams. Comme décrit précédemment il est possible d'exécuter simultanément deux calculs distincts sur le GPU. Le schéma de l'algorithme initial montrait d'ailleurs bien cette approche en deux branches parallèles qui se regroupent au moment de l'appariement. Il semble donc logique d'utiliser des streams différents pour chacune de ces branches.

On obtient le schéma de la figure 3.3.

Le calcul des ROI et la génération des appariements potentiels sont tous deux réalisés sur le CPU car, d'une part il s'agit d'opérations assez légères qui fonctionnent assez bien et, d'autre part il s'agit d'opérations plus difficiles à paralléliser qui n'auraient que peu d'intérêt sur le GPU. De plus le calcul des ROI ne nécessite aucune donnée calculée par le GPU et peut donc s'exécuter en même temps que le calcul des kernels.

Le fait d'avoir deux branches parallèles pour la première partie permet de masquer les transferts de données importantes. En particulier le transfert de l'image, qui peut être réalisé pendant le calcul de projection des patchs. De même la génération des paires potentielles, réalisée sur le CPU, peut s'exécuter pendant que les descripteurs sont calculés sur le GPU.

Cette organisation permet donc de tirer parti d'un premier niveau de parallélisme entre les tâches. Cependant au niveau de chaque étape, il est également nécessaire d'adapter l'algorithme à la nouvelle implémentation. En particulier, il faut gérer correctement l'utilisation de la mémoire et l'enchainement des kernels.

Afin de limiter les temps de transfert au maximum, l'ensemble des données utiles et des tableaux nécessaires sont alloués au début du programme, lors d'une phase d'initialisation. En particulier, l'ensemble des données associées aux patchs, en l'occurrence leur texture et leur pose de référence, sont transférés entièrement sur le GPU à l'initialisation. De même les paramètres intrinsèques de la caméra sont transférés à l'initialisation, utilisés pour une phase de calculs préliminaires afin de n'avoir plus à réaliser quoi que ce soit par la suite. De plus les calculs initiaux permettent autant que possible de déterminer la taille des données utilisées et

d'allouer par avance toute la mémoire nécessaire sur le GPU. Cela permet de diminuer grandement le temps de chaque itération ; cependant, cela empêche de pouvoir charger au fur et à mesure de la localisation les données nécessaires, ce qui peut être gênant pour de grandes trajectoires. En effet l'ensemble des données de la trajectoire doit rester suffisamment peu important pour tenir dans la mémoire du GPU. Pour réaliser des trajectoires très longues il serait nécessaire de trouver un moyen de modifier les données chargées sur la carte et de les mettre à jour régulièrement.

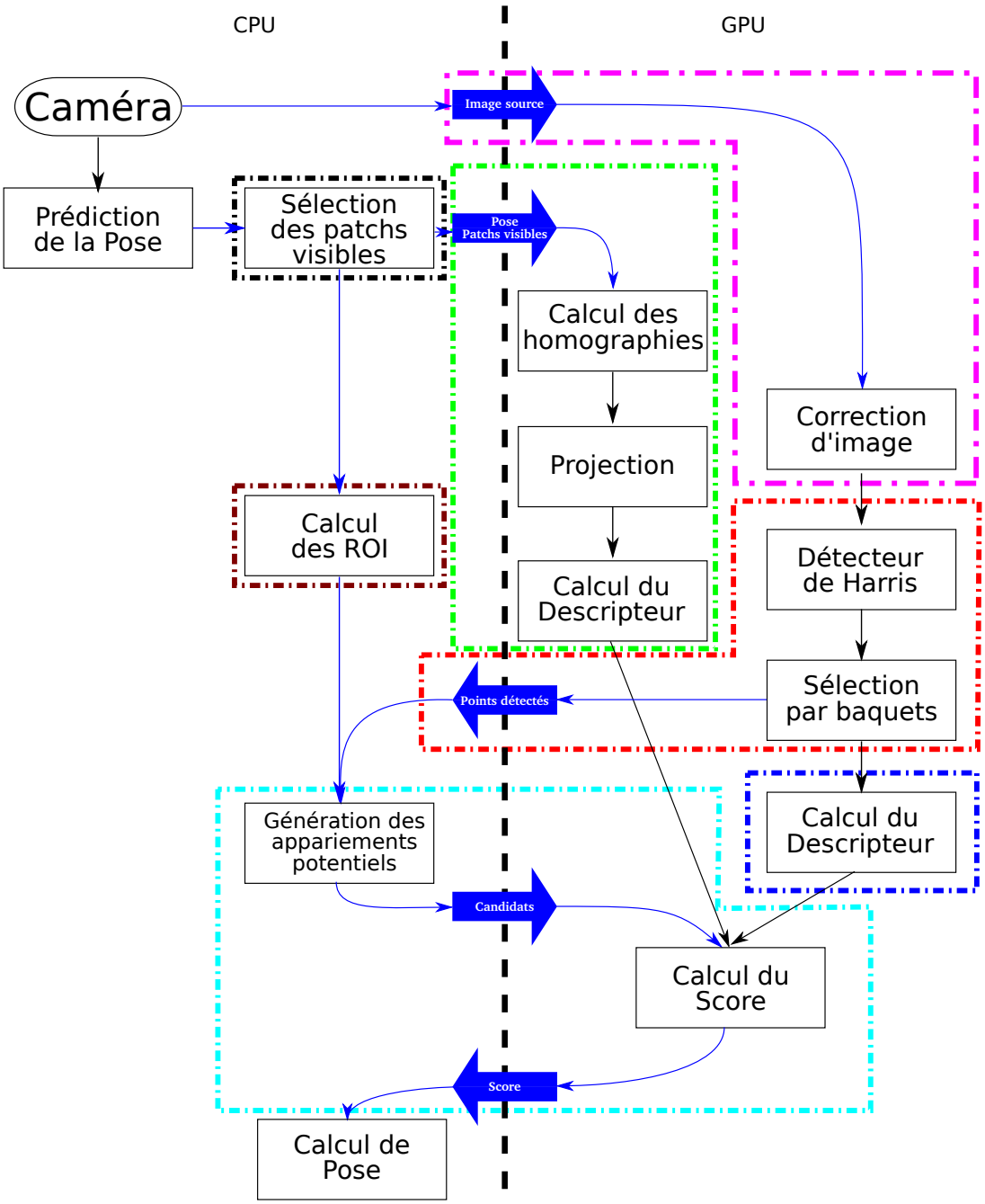


FIGURE 3.3 – Schéma d'ensemble de l'implémentation GPU de l'algorithme de localisation. Les flèches bleues symbolisent les transferts de données (les plus grosses sont pour les transferts entre GPU et CPU), les flèches noires symbolisent une exécution successive sans transfert de données. Les cadres colorés reprennent les étapes de la figure 2.10. Chaque cadre dans la partie GPU correspond à un kernel

3.4.2 Correction de l'image

La partie de la correction de l'image englobe à la fois le transfert de l'image vue par la caméra et la correction de la distorsion.

Pour réaliser cette étape le plus efficacement possible, il est tout d'abord intéressant de noter que la correction est toujours la même. En effet la caméra utilisée pour la localisation ne change pas au cours du temps et par conséquent son étalonnage reste le même. Il est donc inutile de recalculer les corrections à appliquer à chaque pixel à chaque itération. La solution la plus efficace consiste à construire deux tableaux de correction de la taille de l'image corrigée contenant pour chaque pixel de l'image corrigée les coordonnées respectivement en abscisse et ordonnée du point correspondant sur l'image source.

Ainsi à chaque itération, il suffira pour chaque pixel de regarder à la position correspondante dans les tableaux les coordonnées du point et à les lire sur l'image pour obtenir la valeur du pixel correspondant. Cette opération est indépendante pour chaque pixel, la seule différence étant la position à observer dans le tableau et l'endroit où écrire la valeur lue. Chaque pixel peut donc être traité par un thread différent. L'organisation par blocs consiste à avoir des blocs de 8×8 threads, regroupés dans une grille de taille dépendant de la taille de l'image corrigée. L'image corrigée peut être légèrement rognée de manière à garantir que sa taille soit toujours un multiple de 8. Cela permet d'utiliser tous les threads à chaque instant moyennant une perte réduite puisque, en raison de la correction, très peu d'informations sont présentes sur les bords de l'image (voir figure 2.2b pour un exemple d'image corrigée). De plus, les régions qui tombent en dehors de l'image, représentées généralement par du noir seront ici mises à une valeur remarquable, typiquement inférieure à 0. Cela permet par la suite de savoir quels points n'ont pas d'information de couleur.

Au niveau de l'utilisation de la mémoire, les tableaux sont placés directement dans la mémoire globale. Les threads accédant à une valeur dont la position est liée directement à leurs coordonnées, l'alignement en mémoire sera toujours respecté. De plus ne chargeant qu'une seule valeur dans chaque tableau, l'utilisation d'un cache serait inutile, il n'est donc intéressant d'utiliser ni la mémoire texture ni la mémoire constante pour stocker les tableaux de correction.

Par contre pour l'image source reçue par la caméra, le problème est différent. En effet les coordonnées des points à prélever étant issues du calcul d'un polynôme, elles sont rarement entières. Par conséquent il est nécessaire de réaliser une interpolation bilinéaire de l'image pour obtenir la valeur du pixel à mettre dans l'image corrigée. Il est donc indispensable de placer l'image source dans une mémoire texture, où l'interpolation bilinéaire sera réalisée très rapidement. De plus la taille de l'image étant toujours la même cette allocation pourra être réalisée une seule fois et la mémoire réutilisée à chaque itération.

Pour finir l'image corrigée sera écrite directement en mémoire globale. Comme pour les tableaux de correction, les threads n'écrivent qu'une valeur liée à leurs coordonnées et ne poseront donc aucun problème d'alignement.

Cette partie est donc assez bien adaptée à l'implémentation GPU. On a de nombreux threads

parfaitement indépendants qui exécutent une opération simple et n'ont aucun problème d'alignement mémoire. De plus l'utilisation de l'interpolation bilinéaire câblée offre une amélioration certaine. Le seul facteur pouvant pénaliser l'implémentation sur GPU est le transfert de l'image.

3.4.3 Détection des points d'intérêt

La détection des points d'intérêt peut se décomposer en deux parties. La première consiste à calculer le critère de Harris sur toute l'image. Cette opération peut être parallélisée en utilisant un thread par pixel. La seconde partie utilise le critère calculé précédemment pour trouver les maximum locaux et générer la liste des points détectés. Ces deux parties vont être implémentées sous la forme de deux kernels différents.

Calcul du critère de Harris

La première partie exécutée par le kernel *Détecteur de Harris* utilise l'image pour détecter les points d'intérêt. Plusieurs variantes de l'algorithme de Harris ont été développées, comme celle décrite par Schmid et al. (1998) qui permettait d'améliorer la qualité des détections. Dans notre cas, la version utilisée permet d'améliorer la vitesse d'exécution notamment en utilisant des coefficients entiers lors du calcul des masques gaussiens. Pour résumer l'algorithme, on le décompose en plusieurs étapes :

- ➢ On applique un flou en convoluant l'image avec le masque $\frac{1}{16}\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}$;
- ➢ on calcule les gradient g_x et g_y suivant respectivement les axes x et y ainsi que leur produit $g_\times = g_x \times g_y$ en convoluant avec l'image les masques $\begin{bmatrix}-1&0&1\end{bmatrix}$ et $\begin{bmatrix}-1\\0\\1\end{bmatrix}$;
- ➢ On fait la moyenne de chaque gradient en convoluant avec le masque $\frac{1}{256}\begin{bmatrix}1&4&6&4&1\\4&16&24&16&4\\6&24&36&24&6\\4&16&24&16&4\\1&4&6&4&1\end{bmatrix}$ ce qui nous donne les valeurs G_x, G_y et $G_\times$;
- ➢ On calcule l'image critère de Harris avec l'équation $G_x \times G_y - G_\times - 0.04(G_x + G_y)^2$;
- ➢ On ne conserve que les pixels qui ont une valeur de critère supérieure à celle de leurs voisins

Chaque étape correspond à un calcul qui doit être réalisé sur chaque pixel, mettant en œuvre un pixel et ses voisins.

Pour la première étape, le masque doit être convolué directement sur l'image corrigée. Cette image est déjà présente dans la mémoire du GPU et peut être utilisée directement. Cependant plusieurs valeurs sont nécessaires à chaque thread pour appliquer le masque de flou. Par conséquent l'image est placée dans la mémoire de texture de manière à bénéficier du cache 2D associé. Cela se fait sans copie inutile, puisque l'image corrigée est déjà dans la mémoire globale, et la mémoire texture consiste simplement à associer le cache avec une partie de la mémoire, sans avoir besoin de déplacer les données.

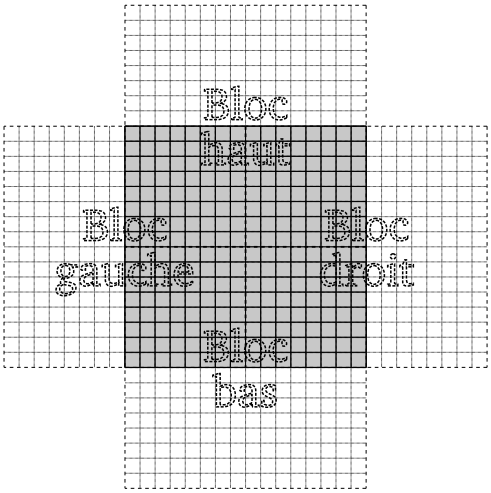
Pour les deux étapes suivantes, il est nécessaire de récupérer non seulement le résultat du pixel courant mais aussi de son voisinage. La valeur obtenue dans le voisinage correspond logiquement à la valeur calculée par un autre thread. Par conséquent il est nécessaire que tous les threads stockent les résultats intermédiaires dans une mémoire partagée, de manière à pouvoir utiliser les résultats les uns des autres. Cependant on a vu précédemment que la mémoire partagée n'est commune qu'aux threads d'un même bloc. Par conséquent, les threads étant à l'extrémité de leur bloc ne peuvent pas accéder aux valeurs calculées pour le pixel voisin, puisque le calcul a été réalisé dans un autre bloc. Tous les threads présents aux extrémités des blocs ne peuvent donc pas obtenir le résultat final. Cependant il ne faut pas faire de trou dans le résultat, chaque valeur de l'image doit avoir une valeur du critère calculé. Pour compenser le fait que les threads du contour soient inutilisés il faut que les blocs se chevauchent, c'est-à-dire que les valeurs soient calculées plusieurs fois (une fois dans chaque bloc nécessaire). Lorsqu'un thread est situé à la bordure de son bloc, il ne fournira pas de résultat final. Il ne calculera que les résultats intermédiaires et sera mis au repos pour les calculs suivants. Concrètement, on obtient l'organisation schématisée figure 3.4.

La partie suivante calculant le critère de Harris, est faite avec les mêmes threads que précédemment. Cette étape ne nécessite pas les valeurs des voisins et n'augmente donc pas le chevauchement.

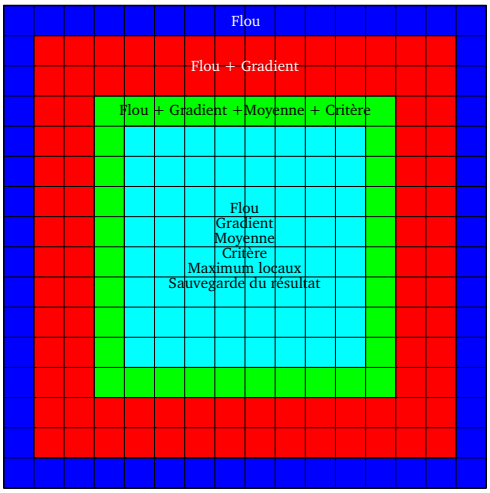
La dernière partie nécessite de comparer une valeur avec ses voisines. Là encore les threads situés en bordure ne peuvent pas réaliser le calcul et seront mis au repos. Les autres threads se contentent de mettre à 0 la valeur dans l'image critère si le score est plus faible qu'un de ces voisins et la valeur du critère autrement (donc lorsque le point est un maximum local). La valeur est écrite directement dans la mémoire globale, afin de conserver le résultat pour l'exécution du kernel suivant.

Une dernière étape avant de terminer le kernel consiste à mettre à 0 les pixels situés à proximité du bord de la correction. Pour cela on regarde dans l'image d'origine si la couleur des pixels voisins est négative, et le score de détection est annulé. Cela permet d'éviter de détecter des points sur la bordure de la correction dus au changement de niveau de gris entre l'absence de valeur (lorsqu'on était en dehors de l'image distordue) et les valeurs correctes à l'intérieur de l'image.

Au bilan, cette partie, bien que largement parallélisée, n'est pas celle qui présente le plus d'améliorations. La parallélisation permet effectivement de calculer le critère sur plusieurs pixels simultanément, mais les calculs sont néanmoins réalisés plusieurs fois pour avoir le ré-



(a) Détail d'un bloc dans la grille utilisée pour le détecteur de Harris. On voit le chevauchement des blocs voisins qui calculeront plusieurs fois le même pixel



(b) Détail des calculs au sein d'un bloc pour le détecteur de Harris. En cyan les threads qui exécutent toutes les opérations et auront un résultat final, en vert ceux qui ne font que les trois premiers calculs intermédiaires, en rouge ceux qui ne font que les deux premiers et en bleu foncé ceux qui ne font que le premier calcul.

FIGURE 3.4 – Organisation de la grille et des blocs pour le kernel détecteur de Harris

sultat dans plusieurs blocs. De plus il est nécessaire pour chaque thread d'accéder plusieurs fois à la mémoire, et bien que ces accès soient réalisés autant que possible de manière alignée, les performances d'accès mémoire sur GPU ne sont pas toujours les plus efficaces. Par ailleurs un certain nombre de threads ne réalisent pas tous les calculs et sont donc nécessairement mis au repos pendant que les autres threads du bloc finissent les calculs. Pour finir, une structure conditionnelle est nécessaire pour la dernière partie ce qui nécessite de séparer en deux parties successives chaque bloc de threads. En conclusion la plus grande capacité de calcul du GPU par rapport au CPU n'est pas vraiment mise en valeur, et on peut s'attendre à avoir sur cette partie un gain en performance moindre que sur les autres.

Sélection des points détectés

Par la suite le kernel `Sélection par baquets` analyse l'image du critère obtenue et en extrait les points détectés. L'objectif est de diviser l'image en plusieurs sous-parties (des baquets) et d'avoir une liste contenant la position des N points détectés avec les meilleurs scores dans chaque baquet. Cela permet d'éviter que tous les points d'intérêt détectés soient regroupés dans une même zone de l'image. La subdivision en baquet permet de contraindre la détection à avoir des points répartis dans toute l'images.

Pour cela on va commencer par subdiviser l'image en un nombre fixe de baquets (8 divisions horizontales et 4 divisions verticales dans nos tests). Chaque bloc de thread se chargera d'un baquet. Ainsi les threads de chaque bloc pourront se partager la liste de points détectés dans le baquet et la mettre à jour. Comme le nombre de points demandés est fixé par baquet, chaque bloc pourra réaliser la recherche indépendamment des autres et mettre sa partie de liste à jour sans interférer avec les autres baquets. Ainsi aucune communication entre blocs ne sera nécessaire.

Au sein d'un bloc, le baquet considéré est subdivisé en autant de parties qu'il y a de threads. Dans notre programme, une subdivision de 32×16 threads était réalisée dans chaque baquet, de manière à ce que chaque thread ne traite qu'un petit nombre de pixels. Chaque thread va donc parcourir la zone de l'image qui lui est associée et, lorsqu'il trouve un score non nul (donc un point détecté), il l'ajoute avec son score dans une liste de points détectés. Cette liste doit être commune à tous les threads du bloc afin d'y regrouper tous les points détectés possibles. La principale difficulté consiste à faire grandir au fur et à mesure la taille de la liste pour y ajouter les points détectés au fur et à mesure qu'ils sont trouvés. Il est en effet impossible de savoir à l'avance combien de points seront détectés, ni leur position. Il faut donc qu'à chaque fois qu'un nouveau point est détecté par un thread, ce dernier remplisse la première case disponible de la liste et empêche les autres threads d'écrire dans la case suivante. Pour réaliser cela il est nécessaire de disposer en permanence du nombre de points déjà détectés. Lorsqu'un thread détecte un nouveau point, il incrémente cette valeur en utilisant une fonction atomique, pour éviter que plusieurs threads modifient cette valeur en même temps, et met à jour la case correspondante du tableau de points détectés. Lorsque tous les threads ont terminé leur parcours, la liste contient tous les points détectés dans le baquet et le score associé.

Une fois obtenue la liste complète des points, il faut encore en extraire les N points ayant les meilleurs scores. Après avoir utilisé une barrière, pour garantir d'avoir obtenu la liste complète des points détectés dans le baquet, obtenir les points avec le meilleur score revient à trier le tableau des points. Pour réaliser le tri en utilisant efficacement tous les threads disponibles, le tri bitonique décrit initialement par (Batcher, 1968) est utilisé. Ce tri consiste à réaliser de nombreuses listes bitoniques (strictement monotone sur deux parties successives) qui sont successivement fusionnées. L'avantage de ce tri est qu'il est assez facile à paralléliser et de nombreuses implémentations sont disponibles, particulièrement pour des architectures massivement parallèles comme le GPU.

Une fois la liste triée, les threads d'indice inférieur à N copient la valeur correspondant à leur indice dans un tableau situé dans la mémoire globale. C'est cette liste qui sera ensuite envoyée au CPU pour réaliser les appariements potentiels.

Cette partie, comme la précédente ne présente au bilan que peu d'améliorations par rapport à la version CPU. En effet la première partie nécessite l'utilisation d'une fonction atomique et de structure conditionnelle pour vérifier qu'un point est détecté. Ces opérations provoquent assez souvent des sérialisations du processus limitant considérablement le gain par parallélisation. La seconde partie avec le tri bitonique est mieux adaptée à l'architecture massivement parallèle du GPU, néanmoins elle nécessite de trier une grande liste alors que la version sur CPU pouvait se

contenter d'en extraire les plus grandes valeurs.

Bilan

Les deux kernels ne sont finalement pas très adaptés à l'implémentation GPU. Comme écrit précédemment, l'utilisation des structures conditionnelles et les contraintes de partage de données sont difficiles à implémenter efficacement sur GPU. De plus, il est nécessaire, après les calculs, de transférer l'ensemble des points détectés sur le CPU. Cela aussi nécessite du temps et devrait donc fortement limiter le gain en performance sur cette partie. Cependant, il reste préférable de laisser ce calcul sur le GPU de manière à pouvoir éviter des transferts depuis le CPU. En effet, si on conservait le détecteur sur le CPU, il serait nécessaire de transférer l'image corrigée au CPU pour pouvoir faire la détection, puis l'envoyer le résultat de la détection au GPU pour calculer les descripteurs des PI. En conservant ces opérations sur GPU, seul le transfert des points détectés est réalisé, et le calcul bien que faiblement optimisé, ne prendra pas plus de temps que la version CPU qui était déjà rapide.

3.4.4 Projection des patchs

La partie concernant la projection des patchs consiste à utiliser les patchs transférés lors de l'initialisation et la pose prédite pour les reprojeter dans la vue prédite. La première difficulté consiste à stocker de manière efficace l'ensemble des patchs. Ensuite, la réalisation de la projection proprement dite se décompose en trois parties, qui correspondent chacune à un kernel différent :

- ➢ Calculer l'homographie à appliquer à chaque patch
- ➢ Appliquer l'homographie sur la texture en mémoire
- ➢ Calculer le descripteur pour faciliter l'appariement

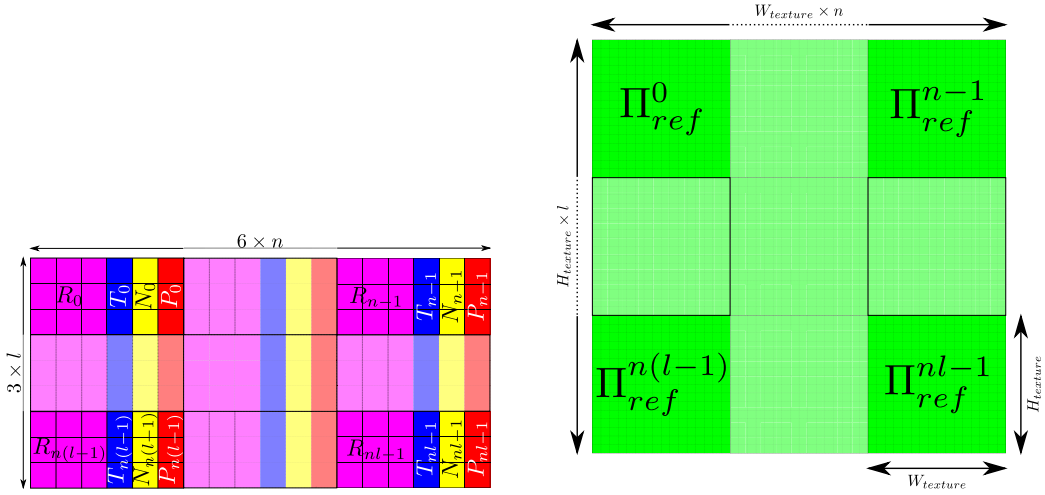
Organisation des patchs en mémoire

Toutes les opérations sont réalisées de manière indépendante pour chaque patch. Il semble donc judicieux de les regrouper pour réaliser toutes les opérations dans un même bloc ou une même grille. Cependant les patchs peuvent être assez nombreux, et le fait de tous les regrouper dans une même grille très grande obligera à séparer le calcul entre plusieurs itérations sur plusieurs multiprocesseurs et n'est donc pas très utile. De plus, tous les patchs ne sont pas forcément visibles à chaque itération. D'ailleurs, le test de visibilité est réalisé justement pour éviter de projeter trop de patchs inutilement. Il est donc intéressant de séparer les patchs et de projeter indépendamment chacun d'entre eux. Toutefois, avoir des patchs individuels ne permettra d'exécuter chaque kernel que sur un nombre réduit de threads ce qui est également néfaste pour les performances. Un compromis a donc été fait en regroupant les patchs par groupes de n

($n = 128$ dans notre implémentation). Ainsi tous les patchs d'un même groupe seront reprojetés en même temps dès qu'il y a au moins un patch visible dans le groupe. Les groupes ne possédant aucun patch visible ne seront pas traités, épargnant autant de temps de calcul.

Les projections de chaque groupe de patchs seront exécutées dans des streams distincts. Ainsi un maximum de multiprocesseurs pourra être utilisé simultanément en parallélisant les différentes projections.

Au niveau de l'organisation en mémoire des patchs, il est nécessaire de pouvoir gérer ces différents groupes. Cependant, pour pouvoir utiliser le même kernel quelle que soit la ligne projetée, les données doivent être organisées entre elles. Dans ce but, les informations associées aux patchs sont stockées sous forme de mosaïques. Chaque texture des 64 patchs est enregistrée côte à côte en mémoire. Chaque ligne correspond ainsi à un groupe différent. Lors de l'appel des kernels, le numéro de la ligne concernée est envoyé en paramètre et permet de retrouver les données. Les informations concernant la pose de référence de chaque patch sont également stockées sous forme de mosaïque organisée de la même manière. Les informations nécessaires de la pose de référence de chaque patch i sont toutefois décomposées sous la forme d'une matrice R_i et de trois vecteurs $\mathbf{T_i}, \mathbf{N_i}, \mathbf{P_i}$ correspondant respectivement à la matrice de rotation et au vecteur de translation associés à la pose de référence, à la normale au plan et aux coordonnées du point au centre du plan. Le schéma de la figure 3.5 montre l'organisation de ces données en mémoire



(a) Poses de référence des patchs

(b) Textures des patchs

FIGURE 3.5 – Organisation de la mémoire stockant les patchs. On note n le nombre de patchs présents dans chaque groupe et l le nombre de groupes de patchs.

Toutes ces données seront utilisées plusieurs fois dans les kernels, et afin de bénéficier du cache 2D, elles seront placées dans la mémoire texture. Pour les textures des patchs, elle devront en plus être interpolées lors de la projection. L'utilisation de l'interpolation câblée est donc

activée pour la mémoire texture les contenant.

La pose prédite obtenue à chaque itération doit également être transférée sur le GPU. Elle est placée en mémoire constante sous la forme d'une matrice R (la matrice de rotation associée à la pose), de matrice R^T la transposée (et inverse) de la matrice précédente et du vecteur $\mathbf{T}$ (vecteur de translation associé à la pose prédite). La matrice de rotation et son inverse sont toutes les deux présentes pour pouvoir accéder plus facilement à la mémoire de manière alignée. La mémoire utilisée est la mémoire constante car ces valeurs ne sont pas modifiées par le GPU, mais sont utilisées de la même manière par tous les threads. De plus la quantité de données est peu importante et tient donc largement dans la mémoire constante (ainsi que dans le cache associé).

Calcul des homographies

Le kernel *Calcul de l'homographie* se charge de calculer pour chaque patch la matrice d'homographie qu'il faudra appliquer aux données. Le calcul de la matrice d'homographie nécessite plusieurs calculs intermédiaires et calculs matriciels qui seront stockés en mémoire partagée.

Au niveau de l'organisation des threads, chaque bloc utilisera 3 threads pour calculer une matrice d'homographie et analysera 32 patchs simultanément (schéma figure 3.6). La grille aura

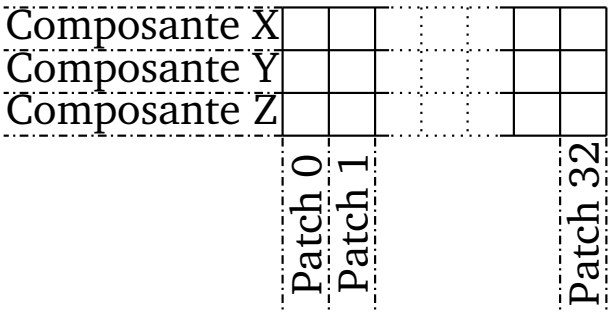


FIGURE 3.6 – Organisation des blocs pour le calcul des homographies

la taille nécessaire pour calculer les matrices de tous les patchs du groupe concerné. Par exemple quand la ligne est remplie avec 128 patchs, la grille aura une taille de $128/32 = 4$.

Le fait d'avoir 3 threads permet de calculer en parallèle chaque composante des vecteurs intermédiaires nécessaire au calcul. Pour les matrices de taille 3×3, chaque ligne est calculée par un thread en réalisant 3 calculs successifs pour chaque colonne. Cela oblige à sérialiser le calcul de chaque colonne mais permet d'éviter d'avoir 6 threads au repos lors du calcul des vecteurs. Les données étant déjà parallélisées sur 32 patchs, il est préférable que chaque calcul d'homographie soit légèrement plus long mais utilise toutes les capacités de calcul disponibles, plutôt que de contraindre à faire plus de blocs avec moins de patchs et de laisser des threads au repos.

Après avoir calculé la valeur de l'homographie pour chaque patch, les valeurs sont enregistrées dans la mémoire globale sous forme d'une mosaïque de matrice 3×3 de même organisation

que les textures de patchs. Ces données pourront être utilisées directement lors de la projection.

Ce kernel permet donc essentiellement de gagner du temps de calcul en calculant les matrices de nombreux patchs simultanément. En théorie le temps de calcul devrait être divisé par le nombre de cœurs disponibles sur la carte graphique. En pratique le gain n'est pas aussi important car les patchs sont reprojetés par groupe et lorsque certains patchs d'un groupe ne sont pas visibles, cela fait des projections qui sont calculées inutilement. Toutefois, les patchs étant regroupés en fonction de leurs positions initiales, la plupart des patchs d'un même groupe sont visibles en même temps et le temps gagné par la parallélisation est plus important que les quelques cas où des patchs non désirés sont reprojetés.

Projection

Le kernel `Projection` utilise les matrices d'homographie calculées précédemment pour transformer la texture des patchs en image dans la vue prédite.

L'image que l'on veut obtenir correspond à une image de 16×16 pixels issue de l'interpolation bilinéaire de la texture. Pour réaliser le calcul chaque thread se chargera d'un pixel de l'image. Chaque bloc aura une dimension de 16×16 et s'occupera d'une image. La grille n'a qu'une dimension de taille n pour considérer tous les patchs d'un groupe.

Chaque thread réalisera donc le produit matriciel de l'homographie calculée précédemment par ses coordonnées dans le bloc. A partir de là il obtient directement les coordonnées 2D du pixel correspondant dans la texture du patch. Cette dernière étant placée dans la mémoire texture, l'accès à l'interpolation bilinéaire correspondant à la valeur est immédiat. Cette valeur est ensuite placée dans la mémoire globale suivant la même organisation en mosaïque que les autres données associées au patch. Lorsque les coordonnées du pixel source se trouvent en dehors de la texture une valeur négative est placée dans l'image. Ceci permettra par la suite de différencier les valeurs de pixel valides de celles qui sont inconnues.

Ce kernel est parfaitement adapté au calcul sur GPU. En effet chaque thread réalise une opération simple (une multiplication matricielle, d'une matrice 3×3 par un vecteur) et écrit une valeur prise dans la mémoire texture. Aucun thread n'est mis au repos, ce qui permet de tirer parti au maximum de la capacité de calcul du GPU. De plus l'utilisation de l'interpolation bilinéaire câblée marque aussi un gros avantage par rapport au CPU. C'est donc cette partie qui montre la plus grande amélioration par rapport à la partie sur CPU. Étant également la partie qui, sur CPU nécessitait le plus de temps de calcul il est donc logique que l'amélioration soit la plus importante sur cette partie.

Calcul du descripteur

Le kernel `Calcul du Descripteur` transforme un ensemble d'images (de taille 16×16) en un ensemble de descripteurs D qui permettront de calculer une ZNCC.

L'ensemble d'images peut se présenter sous deux formes :

- ➢ Une mosaïque de petites images, calculées lors de la projection des patchs ou
- ➢ Un ensemble de coordonnées de points d'intérêt (PI) dont il faut prendre le voisinage dans l'image courante.

Pour chaque image I de l'ensemble, le descripteur D à calculer s'exprime de la forme :

$$D(x) = \frac{I(x) - \overline{I}}{\sqrt{\sum_{x \in E} \left(I(x) - \overline{I}\right)^2 / N}} \tag{3.1}$$

Il est donc nécessaire de calculer la moyenne et l'écart-type de l'image pour ensuite calculer la valeur du descripteur en chaque pixel. Comme pour le kernel précédent chaque thread s'occupe d'un pixel et chaque bloc correspond à un descripteur complet.

La première étape de chaque thread consiste à mettre en mémoire partagée la valeur du pixel correspondant dans l'image. C'est cette étape qui varie selon la forme des valeurs d'entrée (image de patchs ou coordonnées de PI). La principale différence est que la valeur est prise directement en mémoire globale sur la mosaïque dans le cas des patchs ou en lisant les coordonnées du PI et utilisant la mémoire texture où est stockée l'image dans le cas des PI.

Après cette étape on obtient un tableau placé en mémoire partagée de la même taille qu'un bloc contenant pour chaque thread 2 valeurs :

- ➢ Le niveau de gris du pixel I_i correspondant dans l'image source
- ➢ une dernière valeur p_i valant 1 qui servira à compter le nombre de pixel

Un cas particulier se présente lorsque aucune information n'est disponible pour un pixel. Comme expliqué en partie 2.4.2, cela est assez fréquent lorsque la projection d'un patch déborde de sa texture. Ce cas est détecté car une valeur négative a été placée dans l'image. Chaque thread associé à un pixel sans information fixe les 2 valeurs du tableau qui lui sont associées à 0.

Une fois le tableau généré dans la mémoire partagée, le calcul de le moyenne revient à ajouter chaque valeur du tableau. Cela correspond à un algorithme de réduction qui est parallélisé de la manière décrite par (Harris, 2007). L'algorithme consiste à ajouter les valeurs de la moitié des cases avec celles de la seconde moitié. Ensuite le premier quart du tableau est ajouté au second et ainsi de suite jusqu'à n'avoir plus qu'une seule valeur dans la première case, comme décrit sur le schéma de la figure 3.7.

Une fois obtenue la somme de chaque valeur, on obtient le nombre N de pixels de l'ensemble avec la seconde valeur : $N = \sum p_i$. La somme de la première valeur permet de calculer la moyenne $\overline{I} = \frac{\sum I_i}{N}$.

Ensuite, l'écart-type est calculé en appliquant le même algorithme mais sur les valeurs $\left(I_i - \overline{I}\right)^2$

Pour finir chaque thread met la valeur $D(x)$ correspondant à son pixel dans la mémoire globale, suivant la même organisation que précédemment.

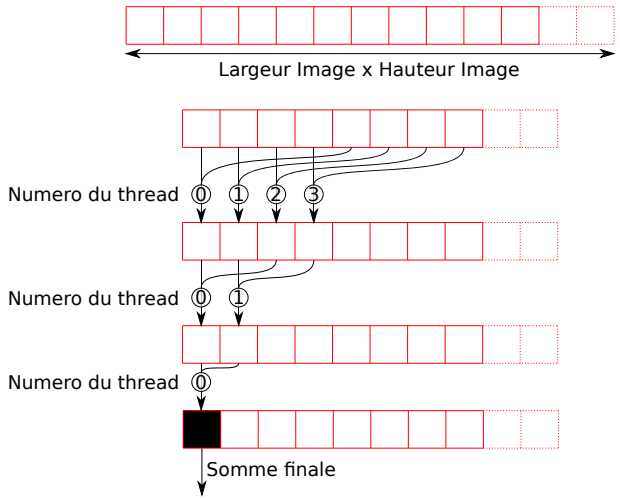


FIGURE 3.7 – Algorithme de réduction parallélisé

Ce kernel permet de réaliser le calcul des descripteurs de manière bien parallélisée, principalement grâce à l'algorithme de réduction. Bien que de nombreux threads soient au repos lors de la réduction, étant regroupés ensemble il ne consommeront pas de calcul supplémentaire et l'optimisation par rapport au CPU reste très importante.

Bilan

L'ensemble de la projection des patchs est constituée de kernels assez performants sur le GPU. L'organisation des patchs permet de ne pouvoir projeter qu'un certain nombre de groupes, afin d'éviter des calculs inutiles. Quelques patchs non visibles pourront être projetés à certaines occasions, mais le gain considérable réalisé grâce à l'utilisation de l'interpolation bilinéaire câblée lors de la projection compense largement cette surcharge. De plus le fait de pouvoir utiliser plusieurs streams et donc d'exécuter la projection de nombreux patchs simultanément apporte encore un avantage au GPU. Cette partie permet donc de tirer grandement parti des avantages du GPU.

3.4.5 Appariement

La partie concernant l'appariement consiste tout d'abord à générer la liste des paires candidates, à transférer cette liste sur le GPU, exécuter le calcul des scores et enfin récupérer les résultats.

La partie concernant la génération des appariements n'a pas changé par rapport à la version CPU précédente. Chaque PI est rangé dans une liste qui les indexe selon leurs coordonnées. Ensuite, pour chaque patch visible, une recherche de tous les PI présents dans la ROI est effectuée, et tous les PI trouvés formeront un appariement potentiel avec le patch considéré.

La liste de tous les appariements possibles, chaque appariement étant formé du numéro du PI et du numéro du patch, est transférée au GPU. Ensuite le kernel `Calcul du Score` calcule la moyenne du produit des descripteurs conformément à l'équation 2.25. Ce calcul est réalisé de manière assez semblable au calcul des descripteurs. Chaque pixel du descripteur correspond à un thread et un bloc de threads analyse un appariement. Chaque thread va lire les valeurs de chaque descripteur et stocker leur produit dans la mémoire partagée. Le même algorithme de réduction que précédemment est utilisé pour réaliser la somme du produit. Le résultat obtenu est ensuite placé en mémoire globale où il pourra être transféré sur le CPU.

Cette partie est assez linéaire, consistant en une succession de transferts, de l'exécution d'un kernel sur les données transférées et du transfert du résultat dans l'autre sens. De plus l'ensemble des appariements et des scores associés représente une grande quantité de données. Dans ce cas, le temps de transfert n'est dans ce cas plus négligeable devant le calcul. Pour finir le CPU est obligé d'attendre la fin du calcul des scores pour pouvoir exécuter la suite.

Pour accélérer le processus, les transferts sont masqués en utilisant les streams. Le fonctionnement consiste à envoyer les appariements potentiels au fur et à mesure de leur recherche. De cette manière, pendant que le calcul s'exécutera sur GPU, il sera encore possible de transférer les candidats suivants. De même, lorsque les scores résultats sont transférés vers le CPU, les candidats transmis entre temps au GPU seront appariés. En pratique dès qu'un ensemble de 128 candidats est repérés, il est transféré dans un tableau de la mémoire globale. Le calcul est ensuite lancé sur le GPU. Pour finir le transfert de ces résultats est également demandé. Cependant ces trois appels sont réalisés dans un même stream. Ainsi ils ne sont pas exécutés directement mais envoyé au GPU et sont exécutés dès que possible. Le CPU pouvant reprendre la main, il peut continuer à générer les appariements potentiels suivants jusqu'à en avoir à nouveau 128. À nouveau les transferts et le kernel sont appelés mais associés à un second stream et à une autre zone de la mémoire globale. Ainsi le calcul du kernel pourra se faire en même temps que le transfert des résultats de la première série. Lorsque les 128 candidats suivants sont disponibles, le CPU attend la fin du premier stream pour relancer le calcul avec les nouvelles données. Ainsi, les deux streams peuvent s'exécuter en alternance sur le GPU tout en conservant la simultanéité du transfert et de l'exécution du kernel. Cela permet de masquer, au moins en partie la latence due au transfert.

Une fois que tous les appariements ont été envoyés, l'ensemble des scores est récupéré et transféré au calcul de pose.

Au bilan ce calcul d'appariement, bien que nécessitant de gros transferts de données, est tout de même bien optimisé. Comme pour le calcul des descripteurs, l'algorithme de réduction reste assez efficace malgré les threads placés au repos. L'utilisation des streams permet de masquer au moins partiellement les transferts et de conserver une amélioration optimale.

3.5 Conclusion

Pour mesurer l'amélioration effectuée par l'implémentation sur GPU, le même profiling que précédemment a été réalisé. Les résultats sont résumés dans le tableau 3.2 Les calculs ont été

	Moyenne (ms)	Médiane (ms)	Écart-type (ms)	Maximum (ms)	Temps écoulé (%)	Gain par rapport au CPU
Correction d'image	3,48	3,48	0,04	3,64	4,49%	4,30
Détection des points	2,92	2,91	0,04	3,16	3,76%	10,49
Sélection des patchs visibles	1,34	1,29	0,25	1,87	1,73%	1,46
Calcul des ROI	23,01	20,94	8,48	46,38	29,70%	0,95
Projection des patchs	4,28	4,20	0,58	5,94	5,52%	561,38
Calcul des descripteurs de PI	0,61	0,61	0,02	0,66	0,78%	1814,42
Appariement	41,43	34,70	17,07	101,72	53,46%	2,81
Total	77,49	68,76	25,36	156,12	100,00%	47,63

TABLE 3.2 – Profiling du code sur GPU. Les couleurs et noms des fonctions correspondent à celle de la figure 2.10. Les parties de l'algorithme mettant moins de 1 ms par itération ont été ignorées

réalisés sur la même séquence que précédemment en utilisant le même ordinateur, qui en plus de son processeur Intel core quad cadencé à 2,4 GHz possède une carte graphique Nvidia geforce GTX 460. Cette carte possède 7 multiprocesseurs de 48 cœurs chacun et une mémoire globale de 1 Gio. Afin de pouvoir mesurer les temps de calcul dans les mêmes conditions, chaque partie (symbolisée en couleur) a été exécutée successivement. Par conséquent l'exécution d'une itération sans la mesure de ces temps de calcul est encore accélérée par rapport à ces valeurs. En particulier, le transfert de l'image peut avoir lieu en même temps que la projection des patchs ou encore le transfert des points détectés se réalise en même temps que le calcul des descripteurs de PI.

Le tableau des résultats 3.2 montre également dans la dernière colonne le rapport de gain entre CPU et GPU. On s'aperçoit que les gains les plus importants sont réalisés lors de la projection des patchs et du calcul des descripteurs. Ce résultat était attendu puisque le CPU était particulièrement peu optimal sur ces opérations et c'est justement pour les optimiser que l'utilisation du GPU a été choisie au départ. L'utilisation de la mémoire texture avec un cache 2D

et une interpolation câblée est le facteur prédominant dans ces opérations. De plus de nombreux calculs sont exécutés en même temps, rendant encore plus favorable l'implémentation sur GPU. Par ailleurs les seuls transferts de données comptabilisés dans ces deux opérations sont de taille assez réduite, et ont peu d'influence sur le temps de calcul.

A l'inverse la fonction d'appariement présente une optimisation moindre. Pourtant la partie sur GPU est pratiquement identique à celle du calcul de descripteurs qui elle, a permis une grande optimisation. Cela s'explique car d'une part la partie génération des appariements potentiels est toujours effectuée sur le CPU, et n'a donc pas été améliorée. De plus le calcul des différents scores est modulé par la nécessité de faire les transferts vers le GPU. Le transfert est certes masqué par l'utilisation de streams mais il est toutefois contraint d'attendre que les appariements soient générés par le CPU pour s'exécuter. De plus un autre facteur de ralentissement de cette partie est lié au nombre d'exécutions. En effet cette partie est exécutée pour chaque appariement ce qui correspond au nombre de patchs visibles multiplié par le nombre de PI moyen dans leur ROI. Pour avoir un ordre d'idée, dans la séquence utilisée pour l'analyse, 59 107 appariements sont générés en moyenne à chaque itération alors que seulement 15 683 patchs sont présents dans la séquence (dont seulement 60% en moyenne sont observables et donc projetés à chaque itération). Ainsi la projection des patchs est appelée 6 fois moins que le calcul du score. Le fait de devoir se répéter un certain nombre de fois, en étant limité par la vitesse du CPU pour générer les paires rend l'amélioration par rapport à la première implémentation moins importante. D'autre part, on peut constater que le temps de cette partie varie beaucoup, comme le montre l'écart-type des temps de plus de 17 ms. Ceci s'explique par le fait que le nombre d'appariements peut grandement varier d'une image à l'autre selon la répartition des points détectés et le nombre de patchs visibles. Une meilleure amélioration de cette partie pourrait d'ailleurs être réalisée en limitant cette variation, par exemple en diminuant les tailles des ROI.

La partie concernant la correction d'image et la détection présente un gain assez moyen par rapport au CPU. Ces fonctions étaient déjà assez rapides sur le CPU et il est difficile d'accélérer indéfiniment tous les processus. De plus ces deux parties ne sont exécutées qu'une seule fois par itération, et n'ont donc pas été une priorité dans l'amélioration réalisée. Le transfert des données vers le CPU prend également du temps et, contrairement aux autres transferts, celui-ci ne peut pas être masqué par une exécution sur le GPU simultané. Tout cela explique que l'amélioration apportée par le GPU reste assez faible sur ces opérations.

Pour finir les deux parties restées sur le CPU ont assez peu évolué. La différence de temps peut s'expliquer par la modification de la quantité de données dans le cache du CPU. En effet la mémoire du processeur est vidée de toutes les informations concernant la pose et la texture des patchs. Seules les données concernant les zones d'observabilité sont présentes et donc vraisemblablement accessibles plus rapidement. Pour le calcul des ROI, le fait d'être légèrement plus lent vient sans doute de l'organisation sous forme de liste. Les listes sont générées depuis les informations du GPU qui sont initialement organisées sous forme de tableaux distincts par baquets moins efficace que dans la version initiale où un seul tableau contenait toutes les données. C'est probablement cela qui explique la faible augmentation de temps entre les deux versions.

Lorsque l'on fait le bilan, on s'aperçoit qu'un facteur 45 a été gagné par rapport à l'implémentation CPU. Plus important encore, alors que l'appariement nécessitait plusieurs secondes pour chaque image, il est réduit à une moyenne inférieure à 80 ms. En mesurant plus globalement le temps de localisation (sans mesurer les temps intermédiaires), la localisation complète (en intégrant prédiction et calcul de pose) peut se faire sur la même séquence avec 90 ms de moyenne, et une durée maximum de 178 ms. Cela signifie que le véhicule pourrait fonctionner à 5 fps sans perdre une seule image, et à près de 12 fps en dehors des cas limites. Cette application devient donc utilisable dans un contexte de localisation temps-réel.

3.6 Perspectives

Cette implémentation en GPU permet donc d'atteindre des performances temps-réel. Cependant, cette implémentation n'est qu'une première réalisation et pourrait être encore améliorée. Tout d'abord on constate sur la dernière analyse, que la partie concernant le calcul des ROI, encore exécutée sur GPU nécessite près de 30% du temps de calcul. Cette partie, qui était négligeable lors de la première implémentation sur CPU, deviendrait donc intéressante à optimiser et pourrait également être portée sur GPU. Plus généralement toutes les parties encore exécutée sur le CPU pourraient être portées sur le GPU évitant ainsi les transferts de données intermédiaires et multipliant si nécessaire les streams pour les réaliser simultanément.

Un autre point négligé dans cette implémentation pourrait également devenir un facteur limitant. Il s'agit de l'utilisation de la mémoire. Dans le cadre de cet algorithme, l'ensemble des patchs est transféré dans une phase initiale dans la mémoire du GPU, de manière à être tous disponibles à chaque instant. Ceci reste possible tant que la trajectoire concernée est limitée à une petite zone de recherche ne contenant que quelques milliers de patchs. Si l'on désirait faire évoluer le véhicule suivant une trajectoire de plusieurs kilomètres, le nombre de patchs générés lors de l'apprentissage deviendrait énorme et ne pourrait plus tenir entièrement dans la mémoire du GPU. Pour pouvoir traiter de plus longues séquences, il serait alors nécessaire de pouvoir charger et décharger les patchs de la mémoire du GPU au fur et à mesure de la localisation. Ainsi, en se basant par exemple sur les zones d'observabilité, on pourrait définir plusieurs zones dans la zone considérée. A chaque zone est associé un certain nombre de patchs observables. En fonction de la position courante du véhicule, les patchs observables dans les zones alentour seraient chargés dans la mémoire et ceux des zones éloignées enlevés. Ainsi la mémoire du GPU ne serait jamais saturée même lorsque la zone d'évolution est importante. La difficulté de cette méthode reste toutefois dans la mise en œuvre du système d'évaluation des patchs à éliminer ou conserver dans la zone courante et dans le transfert effectif des données qui ne doit pas perturber la localisation.

Pour finir l'implémentation sur GPU permet de libérer le CPU de nombreuses tâches et de bénéficier ainsi d'une plus grande puissance de calcul disponible. Cette puissance de calcul pourrait être utilisée pour réaliser d'autres opérations complémentaires permettant de fiabiliser la localisation. Par exemple on pourrait implémenter une méthode d'odométrie visuelle qui

ajouterait des informations à la pose courante à chaque itération tout en réutilisant un certain nombre de résultats déjà calculé dans l'algorithme, tels que la position des points d'intérêt de l'image. Une autre amélioration intéressante pourrait être de diminuer le nombre de patchs projetés à chaque itération. Pour cela une méthode devrait être définie pour trier les patchs a priori afin de savoir s'ils sont intéressants à projeter ou non. Une fois les patchs triés, il est possible de ne projeter qu'un nombre fixe de patchs (ou de groupe de patchs). Cette méthode empêcherait sans doute un certain nombre de bons appariements de se faire, mais tant que suffisamment de patchs de référence sont appariés pour se localiser, l'intérêt d'avoir davantage de points reste discutable. La principale difficulté pour cette solution consiste à définir un facteur qui permettrait de savoir a priori si un patch (ou même un groupe de patchs) est plus intéressant à rechercher qu'un autre. Cela permettrait par la suite de diminuer le nombre de reprojection et d'appariement et donc d'accélérer encore le processus global.

Chapitre 4

Prédiction de la pose

4.1 Introduction

Les méthodes précédentes permettent d'obtenir une localisation assez rapide. Cependant les premières utilisations sur véhicule ont montré que parfois, la précision peut varier énormément. Cela est dû à plusieurs raisons.

Tout d'abord le temps de localisation est dépendant du nombre de patchs visibles. En effet, lorsque de nombreux patchs sont présents devant la caméra, cela fait autant de projection, et lorsqu'on les regroupe avec les points d'intérêt détectés de nombreux appariements potentiels doivent être analysés. Ainsi selon l'endroit où le véhicule est présent, la localisation peut être plus ou moins rapide. Lorsque la localisation est assez lente, le véhicule se déplace davantage entre deux images et la prédiction devient alors moins précise. La reprojection des patchs, alors moins proche de l'image courante rend l'appariement plus difficile, ce qui peut se ressentir sur la localisation.

Une autre difficulté particulièrement gênante est due aux faux appariements. Dans les algorithmes basés sur des images clef, seul un nombre réduit de points d'une image-clef est comparé à l'image courante. Avec l'utilisation des patchs, les images-clefs n'étant plus utilisées, l'ensemble des patchs observables est comparé à l'image courante. En pratique un certain nombre de patchs issus de mauvais appariements lors de l'initialisation présentent une texture floue qui obtient un score d'appariement moyen avec n'importe quelle image. Le facteur de qualité n'est malheureusement pas assez performant pour détecter ces mauvais patchs, et aucun critère a priori n'a pu être déterminé pour les éliminer. Lorsque la prédiction est correcte, les patchs correctement reprojetés ayant un fort score avec les points qui leur correspondent vont permettre lors de l'élimination des conflits de faire disparaitre les appariements contenant ces mauvais patchs. Mais lorsque la prédiction est imprécise, les patchs présentent des scores moins importants avec leur point correspondant et certains patchs plus uniformes présentent des scores d'appariement plus élevés qui, lors de l'élimination des conflits supprimeront les appariements corrects. Lorsqu'ils sont trop nombreux ces faux appariements peuvent provoquer

une importante erreur de localisation (en particulier perte d'intégrité) qui ne serait pas détectée par le système. La prédiction se basant sur les localisations successives, il devient dès lors très difficile de retrouver la bonne position.

Dans ces deux cas, le problème vient essentiellement d'un trop grand nombre de patchs et surtout d'une prédiction moins précise. L'utilisation du facteur de qualité qui permet déjà de filtrer quelques patchs a priori et de la zone d'observabilité qui supprime à chaque itération les patchs éloignés ont déjà permis de limiter un peu le nombre de patchs. Cependant, de nombreux amers restent présents et ne peuvent être éliminés sans risquer de perdre l'avantage de la méthode.

Pour pallier les problèmes mentionnés précédemment sans perdre davantage de patchs, la prédiction de la localisation doit donc être améliorée. Pour éviter les problèmes d'erreur de localisation qui rendent la trajectoire non intègre, une solution consiste à comparer la localisation avec les informations qui viendraient d'autres capteurs. En particulier l'odométrie qui mesure la vitesse et l'angle de braquage des roues du véhicule est généralement disponible sur le véhicule et permet de vérifier que les informations venant des deux capteurs (la caméra et l'odométrie) sont cohérentes. De plus les informations fournies par l'odométrie permettent de prédire plus précisément la position. Cette amélioration permet également de réduire l'incertitude sur la prédiction et donc la taille des ROI. Cela permet de diminuer le nombre d'appariements à calculer réduisant d'autant le risque de faux appariements. Cette méthode semble donc, dans un premier temps, celle qui permettra de réduire les difficultés.

4.2 État de l'art

L'utilisation de plusieurs capteurs pour se localiser est assez commune dans le domaine de la robotique. En effet chaque capteur présente des points forts et des points faibles. L'utilisation d'algorithmes permettant de fusionner ces données permet donc de bénéficier des avantages de tous ces capteurs sans en subir les inconvénients. Cette partie présente donc brièvement les différents capteurs couramment utilisés avec leurs avantages et inconvénients. Par la suite, un rapide résumé des méthodes de fusion utilisées sera réalisé.

4.2.1 Méthode d'acquisition des données capteurs

Les données fournies par les capteurs généralement utilisés en robotique peuvent être divisés en deux groupes :

- ➢ les données fournissant une position absolue, qui fournissent une position directement par rapport au monde extérieur, Par exemple les informations fournies les GPS donnent une latitude et longitude absolue dans une repère terrestre ;
- ➢ les données fournissant une évaluation du déplacement, et ne mesurant donc que l'évolution depuis une position précédente, par exemple les données fournies par les capteurs

proprioceptifs tel que l'odométrie qui évalue la distance parcourue depuis la dernière mesure.

Les données fournissant une position absolue permettent de connaitre sa position en permanence avec une précision constante, mais elles sont généralement assez imprécises ou très couteuse. Un GPS par exemple pourra donner une position au sol avec une précision de l'ordre du mètre. Pour obtenir une précision plus importante avec ce genre de système, il est nécessaire d'utiliser une base fixe de position connue pour corriger les incertitudes du GPS et obtenir un système de GPS différentiel beaucoup plus couteux. De plus la cadence de mesure est également assez faible, et la position, si elle est absolue et ne diverge pas à long terme, n'est pas disponible à tout moment. Pour finir ces systèmes sont généralement assez dépendants de l'environnement. Les systèmes de GPS ont en effet besoin de percevoir les satellites et ne peuvent donc pas être utilisés en environnement intérieur, et seront également difficile à utiliser dans les cas de canyon urbain.

Les données fournissant un déplacement sont assez complémentaires. Tout d'abord les informations sont généralement disponibles avec une cadence élevée, et leur précision est assez bonne même pour des capteurs bas-cout. Cependant, le fait de se situer par rapport à la position précédente fait que les erreurs même minimes sont intégrées au cours du temps et provoquent une dérive de la position. De plus des systèmes comme l'odométrie ne peuvent pas toujours obtenir une information complète sur la position. En particulier lorsque seul le mouvement des roues est utilisé, la position est connue uniquement dans le plan du sol, et il n'est pas possible de mesurer des changements d'altitude.

Les algorithmes basés sur la vision, et plus généralement utilisant toute sorte de capteur extéroceptif comme les télémètres laser, peuvent également être considérés comme des capteurs. Selon le cas, le capteur présentera les avantages des capteurs proprioceptifs ou des capteurs absolus. Par exemple les algorithmes de SLAM ou d'odométrie visuelle se positionnent au fur et à mesure du déplacement, et donnent une position assez fiable par rapport à la pose précédente, tout comme les capteurs proprioceptifs. Par contre dans les approches utilisant une carte, telles que notre approche avec des patchs provenant d'une reconstruction préalable de l'environnement, la localisation à chaque itération est faite de manière absolue dans la carte de référence, permettant de classer la méthode comme un capteur absolu. De plus, de nombreuses propriétés sont semblables au GPS, la cadence de mesure n'est pas très élevée et la précision de localisation varie peu, puisqu'elle n'est liée qu'à la précision de localisation initiale.

4.2.2 Méthode de fusion

Pour tirer parti des avantages de chaque capteur, un bon moyen consiste à fusionner les données. Pour réaliser cette fusion, de nombreuses approches ont déjà été développées. Les plus populaires dans le cadre de la localisation d'un véhicule sont le filtre à particules (ou méthode de Monte Carlo séquentielle) et les dérivées du filtre de Kalman (Kalman, 1960).

Le filtre à particules consiste à évaluer l'état d'un système par le comportement de multiples

échantillons appelés particules, qui vont évoluer selon le modèle du système utilisé. L'avantage de ce filtre est qu'il ne nécessite aucune hypothèse de linéarité sur le système étudié. L'inconvénient est qu'il peut être assez lourd à mettre en œuvre, en particulier lorsque de nombreuses particules sont utilisées, ce qui est nécessaire pour avoir une bonne précision et robustesse.

Plus ancien, le filtre de Kalman a déjà été largement utilisé dans des contextes de localisation temps réel. Par exemple (Rezaei and Sengupta, 2007) utilise un filtre de Kalman pour fusionner les données venant de capteur DGPS, de l'odométrie du véhicule et d'un gyroscope. Contrairement au filtre à particule, le filtre de Kalman suppose que le système est linéaire, et lorsque le système ne l'est pas, les approximations nécessaires pour le linéariser ne sont pas toujours très réalistes.

Dans les deux cas (filtre à particule et filtre de Kalman), la modélisation du système consiste à écrire un ensemble de valeurs dans un *vecteur d'état* $\mathbf{X}$ qui correspond à l'état que l'on désire connaitre du système. Il peut s'agir de la position, de l'angle de cap du véhicule, de sa vitesse, etc. Pour déterminer les valeurs de ce vecteur d'état, on s'appuie sur un modèle d'évolution du système modélisé par une *équation d'état*. Par exemple on peut supposer que le véhicule avance toujours en ligne droite à la même vitesse et l'équation d'état consistera donc à modifier la position du robot suivant le vecteur vitesse courant. On considère aussi un *vecteur d'observation* $\mathbf{Y}$, constitué des valeurs fournies par les capteurs. La fonction d'observation définit l'équation à appliquer au vecteur d'état pour obtenir les mesures des capteurs. Pour finir, on peut également tenir compte de la commande $\mathbf{u_k}$ envoyée au robot. Toutes les notations utilisées dans la modélisation du système et les approches de fusion sont résumées dans la table 4.1.

Dans le cas d'un système linéaire, les équations reprenant ce modèle sont détaillées dans le système d'équations 4.1.

$$\begin{cases} \mathbf{X_{k+1}} = A_k\mathbf{X_k} + B_k\mathbf{u_k} + M\mathbf{w_k} \\ \mathbf{Y_k} = H\mathbf{X_k} + \mathbf{v_k} \end{cases} \tag{4.1}$$

Les vecteurs $\mathbf{w_k}$ et $\mathbf{v_k}$ sont formés de variables aléatoires, sont mutuellement indépendants et suivent chacun une distribution de probabilité normale et centrée. On leur associe à chacun une matrice de covariance, noté Q_k et R_k. Ces deux matrices doivent être évaluées en considérant l'erreur commise par la modélisation pour la première et l'incertitude de mesure des capteurs pour la seconde.

L'objectif des algorithmes de filtrage est de connaitre à chaque itération k (pour laquelle des mesures ont été effectuées), la valeur $\mathbf{X_k}$ de ce vecteur d'état ainsi que l'incertitude P_k associée.

Dans les algorithmes de localisation existant, le filtre de Kalman reste l'outil le plus utilisé pour réaliser une fusion. Nous avons donc choisi de l'utiliser également dans nos travaux, car tout en restant assez simple à mettre en œuvre, il reste bien adapté à notre situation. C'est donc cet algorithme que l'on va décrire plus en détail dans cette partie. Dans le cas du filtre

Notation	Détail	Dimension
	Valeurs constante	
I	Matrice identité	$d_X \times d_X$
$\mathbf{0_d}$	Vecteur nul de taille d	d
	Paramètres du système	
$\mathbf{X_k}$	Vecteur d'état du robot à l'instant k	d_X
P_k	Matrice de covariance mesurant l'incertitude du vecteur d'état	$d_X \times d_X$
$\mathbf{u_k}$	Commande envoyée au robot	d_u
$\mathbf{w_k}$	Bruit de modèle	d_w
Q_k	Matrice de covariance du bruit de modèle	d_w
$\mathbf{Y_k}$	Vecteur d'observation	d_Y
$\mathbf{v_k}$	Bruit de mesure	d_Y
R_k	Matrice de covariance du bruit de mesure	d_Y
	Système linéaire	
A	Matrice représentant l'évolution du système	$d_X \times d_X$
B	Matrice représentant l'interaction du vecteur de commande	$d_X \times d_u$
M	Matrice représentant l'interaction du bruit dans le système	$d_X \times d_u$
H	Matrice représentant la fonction d'observation	$d_Y \times d_X$
	Filtre de Kalman	
$\mathbf{X_{k_2\|k_1}}$	Estimation de $\mathbf{X_{k_2}}$ en tenant compte des données disponibles à l'instant k_1	d_X
$P_{k_2\|k_1}$	Estimation de P_{k_2} en tenant compte des données disponibles à l'instant k_1	$d_X \times d_X$
K_k	Gain de Kalman	$d_X \times d_Y$
	Système non linéaire	
f	Fonction non linéaire représentant l'évolution du système en fonction de $\mathbf{X_k}$,$\mathbf{u_k}$ et $\mathbf{w_k}$	d_X
h	Fonction non linéaire donnant l'observation $\mathbf{Y_k}$ en fonction de $\mathbf{X_k}$ et $\mathbf{v_k}$	d_Y
J_k^{fX}	Jacobienne de f par rapport à $\mathbf{X_k}$	$d_X \times d_X$
J_k^{fw}	Jacobienne de f par rapport à $\mathbf{w_k}$	$d_X \times d_w$
J_k^{hX}	Jacobienne de h par rapport à $\mathbf{X_k}$	$d_Y \times d_X$
J_k^{hv}	Jacobienne de h par rapport à $\mathbf{v_k}$	$d_Y \times d_v$

TABLE 4.1 – Notations utilisées pour la modélisation d'un système

de Kalman, P_k correspond à la matrice de variance-covariance de l'estimation du vecteur. A chaque itération, le processus se décompose en deux étapes :

1. La *prédiction*, qui calcule $\mathbf{X_{k|k-1}}$ et $P_{k|k-1}$, estimation de X_k et P_k à partir des états et observations précédents ;
2. La *correction* qui calcule $\mathbf{X_{k|k}}$ et $P_{k|k}$, estimation de X_k et P_k en tenant compte de l'observation $\mathbf{Y_k}$ réalisée à l'itération k, et donc corrige les estimations faites par la prédiction.

Dans la version initiale du filtre de Kalman, on suppose que l'équation d'état et la fonction d'observation sont linéaires. L'imprécision des données est considérée comme un bruit aléatoire. L'étape de prédiction peut s'écrire alors avec le système d'équations suivant :

$$\begin{cases} \mathbf{X_{k|k-1}} = A\mathbf{X_{k-1|k-1}} + B\mathbf{u_k} \\ P_{k|k-1} = AP_{k-1|k-1}A^T + MQ_kM^T \end{cases} \tag{4.2}$$

Cette étape permet en tout instant d'obtenir une prédiction de l'état du système. La nouvelle position est déterminée directement par l'équation d'état. Le filtre de Kalman permet en plus de tenir compte des approximations de ce modèle et d'en déduire une matrice de covariance. C'est cette matrice qui permettra, malgré les imprécisions, de rester intègre dans la prédiction en augmentant l'incertitude sur l'état.

Lorsque de nouvelles observations sont disponibles, l'état prédit est corrigé et l'incertitude réduite avec le système suivant :

$$\begin{cases} K_k = P_{k|k-1}H^T \left(HP_{k|k-1}H^T + R_k\right)^{-1} \\ \mathbf{X_{k|k}} = \mathbf{X_{k|k-1}} + K_k \left(\mathbf{Y_k} - H\mathbf{X_{k|k-1}}\right) \\ P_{k|k} = \left(I - K_kH\right) P_{k|k-1} \end{cases} \tag{4.3}$$

Ces équations reposent sur l'utilisation du gain de Kalman K_k qui permet de pondérer la confiance à accorder à la prédiction et aux mesures. Plus les mesures seront incertaines (donc la matrice de variances-covariance R_k a des coefficients élevés), plus le gain de Kalman sera faible et le filtre accordera donc plus d'importance à la prédiction. L'incertitude $P_{k|k}$ du vecteur d'état ne variera alors quasiment pas. A l'inverse si les mesures sont très précises, K_k aura une valeur très proche de H^{-1} (en supposant que H est carré et inversible). On s'aperçoit alors que le vecteur d'état prendra directement la valeur déduite des mesures et l'incertitude associée devient très faible.

Cette méthode d'analyse présente l'avantage d'être assez rapide à mettre en œuvre et donne des résultats assez intéressants pour filtrer une position. En effet, seules quelques opérations matricielles sont réalisées à chaque itération, ce qui ne nécessite que peu de temps de calcul. La seule contrainte pour utiliser ce modèle consiste à modéliser le système sous forme d'équation d'état et de fonction d'observation linéaire. Cette contrainte est assez restrictive car il arrive fréquemment qu'un système évolue de manière non linéaire. Une première solution consiste à approximer cette évolution par une fonction linéaire et à mettre un bruit de modèle suffisamment important pour rester intègre. Cependant, cette modélisation reste peu valable et présente l'inconvénient d'augmenter grandement l'incertitude sur l'état.

Pour éviter de devoir mettre des incertitudes trop grandes, et réaliser une approximation dans de bonnes conditions, le filtre de Kalman étendu (EKF) a été défini. L'objectif est de pouvoir utiliser une représentation d'état non linéaire, décrite par le système d'équation 4.4.

$$\begin{cases} \mathbf{X_{k+1}} = f(\mathbf{X_k}, \mathbf{u_k}, \mathbf{w_k}) \\ \mathbf{Y_k} = h(\mathbf{X_k}, \mathbf{v_k}) \end{cases} \tag{4.4}$$

Dans l'EKF, ces fonctions sont appliquées directement à la place des opérations avec les matrice A, B et H dans le calcul des vecteurs $\mathbf{X_{k|k-1}}$ et $\mathbf{X_{k|k}}$. Au niveau du calcul d'incertitude, il n'est toutefois pas possible de réaliser le remplacement de manière immédiate. Les fonctions sont alors linéarisées en utilisant le développement de Taylor au premier ordre. Pour cela, les matrices jacobiennes de chaque fonction sont calculées à chaque itération. On définit ainsi 4 matrices :

$$\begin{aligned} J_k^{fX} &= \left.\frac{\partial f}{\partial \mathbf{X}}\right|_{\mathbf{X}=\mathbf{X_k}} & J_k^{hX} &= \left.\frac{\partial h}{\partial \mathbf{X}}\right|_{\mathbf{X}=\mathbf{X_k}} \\ J_k^{fw} &= \left.\frac{\partial f}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{0_{d_w}}} & J_k^{hv} &= \left.\frac{\partial h}{\partial \mathbf{v}}\right|_{\mathbf{v}=\mathbf{0_{d_v}}} \end{aligned} \tag{4.5}$$

La linéarisation des fonctions se fait alors de la manière suivante :

$$\begin{aligned} \mathbf{X_{k+1}} &\approx f(\mathbf{X_k}, \mathbf{u_k}, \mathbf{0_{d_w}}) + J_k^{fX}(\mathbf{X_{k+1}} - \mathbf{X_k}) + J_k^{fw}\mathbf{w_{k+1}} \\ \mathbf{Y_{k+1}} &\approx h(\mathbf{X_k}, \mathbf{0_{d_v}}) + J_k^{hX}(\mathbf{X_{k+1}} - \mathbf{X_k}) + J_k^{hv}\mathbf{v_{k+1}} \end{aligned} \tag{4.6}$$

Cela consiste a approcher chaque fonction par une application linéaire au voisinage de l'état précédent. On obtient ainsi une relation linéaire que l'on peut utiliser pour calculer l'incertitude de l'état. Les équations du filtre de Kalman sont alors modifiées selon le système 4.7.

$$\begin{aligned} &\text{Prédiction :} \begin{cases} \mathbf{X_{k|k-1}} = f\left(\mathbf{X_k}, \mathbf{u_k}, \mathbf{0_{d_w}}\right) \\ P_{k|k-1} = J_k^{fX} P_{k-1|k-1} \left(J_k^{fX}\right)^T + J_k^{fw} Q_k \left(J_k^{fw}\right)^T \end{cases} \\ &\text{Correction :} \begin{cases} K_k = P_{k|k-1} \left(J_k^{hX}\right)^T \left[J_k^{hX} P_{k|k-1} \left(J_k^{hX}\right)^T + J_k^{hv} R_k \left(J_k^{hv}\right)^T\right]^{-1} \\ \mathbf{X_{k|k}} = \mathbf{X_{k|k-1}} + K_k \left[\mathbf{Y_k} - h\left(\mathbf{X_{k|k-1}}, \mathbf{0_{d_v}}\right)\right] \\ P_{k|k} = \left(I - K_k J_k^{hX}\right) P_{k|k-1} \end{cases} \end{aligned} \tag{4.7}$$

Contrairement au filtre de Kalman précédent où les matrices A, B et H étaient constantes, les jacobiennes ont besoin d'être recalculées à chaque instant. Cela permet de conserver une linéarisation réalisée autour de l'itération précédente, et donc proche de l'état courant du système. Cela permet d'éviter que l'hypothèse de linéarisation soit trop forte. En pratique, la valeur du vecteur d'état à l'itération courante n'est pas accessible directement. On va donc linéariser au-

tour de la prédiction courante. Le calcul des jacobiennes devient alors :

$$\begin{aligned} J_k^{fX} &= \left.\frac{\partial f}{\partial \mathbf{X}}\right|_{\mathbf{X}=\mathbf{X_{k|k-1}}} & J_k^{hX} &= \left.\frac{\partial h}{\partial \mathbf{X}}\right|_{\mathbf{X}=\mathbf{X_{k|k}}} \\ J_k^{fw} &= \left.\frac{\partial f}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{0_{d_w}}} & J_k^{hv} &= \left.\frac{\partial h}{\partial \mathbf{v}}\right|_{\mathbf{v}=\mathbf{0_{d_v}}} \end{aligned} \tag{4.8}$$

Cette approche est la plus utilisée dans le domaine de la robotique. Elle a l'avantage d'être facile à mettre en œuvre même sur des systèmes non linéaires. Cependant elle suppose que les fonctions non linéaires du système soient dérivables, pour pouvoir en calculer les jacobiennes et que la linéarisation au premier ordre soit une bonne approximation. Ces deux hypothèses ne sont pas toujours vérifiées et par conséquent, ce filtre peut présenter des problèmes dans ces cas là. De plus le filtre ne garantit pas une convergence systématique. En effet le fait de changer d'approximation à chaque itération lors du calcul des jacobiennes, peut faire sortir l'estimation de la zone de linéarité et rendre le modèle linéarisé invalide.

Pour pallier certains de ces défauts, sans recourir au filtre à particules, d'autres variantes du filtre de Kalman ont été développées. Ces méthodes, bien moins utilisées sont décrites en détail et comparées dans un contexte de localisation par (Mourllion et al., 2005).

Le filtre de Kalman présenté précédemment permet de filtrer un vecteur d'état en fonction d'observation. Cependant dans le cadre de fusion de données, il est important de noter que plusieurs capteurs peuvent fournir des informations différentes. Dans ce cas, une équation d'observation différente est modélisée pour chaque ensemble de mesure, et l'étape de correction propre à chaque vecteur d'observation est réalisée successivement.

4.3 Modélisation du système

Pour utiliser l'algorithme de fusion défini précédemment, il est nécessaire de définir un vecteur d'état et de faire le bilan des mesures disponibles pour en déduire les fonctions d'observation associées à chaque capteur.

4.3.1 Vecteur d'état

Pour commencer, il convient de définir les paramètres du vecteur d'état que l'on désire utiliser. Les valeurs contenues dans le vecteur correspondent à celles que l'on a besoin de prédire et de mettre à jour. Dans le contexte de localisation d'un véhicule, il s'agit généralement de la position du robot. Dans le cadre de l'algorithme de vision, il est également nécessaire de connaitre l'orientation de la caméra, et donc du véhicule. Ainsi le vecteur d'état doit nécessairement contenir la position et l'angle de cap du véhicule. Cependant, bien que la localisation

se fasse dans un monde 3D, le déplacement reste dans le plan de la route. En effet la localisation s'appliquant sur un véhicule terrestre, sa hauteur par rapport à la route va rester constante. De même les angles de roulis et tangage sont négligeables et peuvent être considérés comme constants. Ainsi les seuls paramètres que l'on désire connaitre sont la position dans le plan du sol et l'angle de cap suivi.

Pour définir le vecteur d'état, il est également utile de ne pas se limiter aux seules valeurs que l'on désire connaitre. En effet un certain nombre de paramètres, bien que non utilisés par les algorithmes extérieurs, sont nécessaires pour définir l'état du véhicule. Par exemple un véhicule situé à une position précise à l'arrêt n'est pas dans le même état que s'il a une vitesse. Ces informations doivent également être conservées dans le vecteur d'état et seront utilisées pour prédire la position du véhicule au fur et à mesure du temps. Le véhicule se déplaçant dans un plan, seules les valeurs de vitesse linéaire et angulaire sont nécessaires, l'angle de cap étant déjà pris.

On obtient donc un vecteur d'état composé de 5 valeurs :

- Deux coordonnées x et z, permettant de placer le véhicule dans le plan du sol,
- L'angle de cap θ, donnant l'orientation du véhicule dans le plan,
- la vitesse linéaire $\overline{v}$, valeur algébrique correspondant à l'avancement du véhicule vers l'avant,
- la vitesse angulaire $\dot{\theta}$, représentant l'évolution de l'angle de cap au cours du temps.

Avec ce vecteur d'état, il est nécessaire de définir les fonctions d'observation qui permettent de relier les valeurs d'état aux mesures réalisées sur le système. Dans notre cas, ces mesures proviennent de deux sources différentes. Tout d'abord l'algorithme de vision, développé au chapitre 2. Ensuite, l'odométrie est également accessible sur le véhicule et peut être utilisée pour améliorer la prédiction et la localisation.

4.3.2 Mesure de la vision

L'algorithme de vision permet d'avoir une estimation de la position et de l'angle de cap du robot dans le monde 3D. Cependant, comme expliqué précédemment, ces données peuvent présenter quelques erreurs et ne sont disponibles qu'à une faible cadence de mesure. On peut donc assimiler ces données aux mesures d'un capteur de type absolu tel qu'un GPS. En effet la position est fournie dans un référentiel fixe, et si celui-ci est correctement construit, la localisation ne devrait pas présenter de dérive.

Une donnée de localisation consiste en un ensemble de coordonnées 3D pour la position et des trois angles d'Euler pour l'orientation. Dans la mesure où le vecteur d'état est limité au plan du sol, seules les coordonnées en x et z et l'angle Ry seront utilisés. Par mesure de simplicité, on considère donc que la mesure de vision ne renvoie que 3 valeurs. De plus, la vision est centrée sur la caméra, et renvoie donc la position de celle-ci. Cependant une étape d'étalonnage permet de connaitre la distance entre la caméra et le milieu de l'essieu arrière ainsi que l'angle

entre l'orientation de la caméra et la direction du véhicule. Ainsi, l'algorithme de localisation par vision renvoie le vecteur de mesure suivant :

$$\mathbf{Y_k^{vis}} = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} \tag{4.9}$$

Les valeurs correspondent directement aux trois premières valeurs du vecteur d'état. Ainsi la fonction d'observation associée à la vision est linéaire et est décrite par la matrice H_{vis} valant :

$$H_{vis} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{4.10}$$

L'algorithme de vision fournit une matrice de covariance de l'erreur de localisation calculée à partir de l'erreur de reprojection des amers. C'est cette matrice qui est utilisée directement pour définir R_k^{vis}.

Ce capteur est donc assez simple à mettre en œuvre et la fonction d'observation est linéaire. Cependant il ne permet d'accéder qu'aux informations de position courante. L'utilisation de l'odométrie permet d'avoir une meilleure connaissance de la vitesse d'avancement.

4.3.3 Mesure de l'odométrie

Les véhicules sont équipés d'odométrie, plus précisément de capteurs capables de mesurer la vitesse des roues du véhicule ainsi que l'angle de braquage. Ces données sont disponibles très rapidement et sont assez précises, ce qui est également complémentaire avec la vision.

Fonction d'observation

Pour trouver la fonction d'observation de ces mesures, il est nécessaire dans un premier temps de modéliser le véhicule. Du fait des informations fournies par les capteurs (angle de braquage des roues avant, et vitesse des roues arrière), le modèle tricycle est utilisée. Ce modèle est représenté sur la figure 4.1, et considère une roue directionnelle à l'avant et deux roues motrices à l'arrière.

On note l la distance entre l'essieu avant et arrière du véhicule et e la largeur de l'essieu arrière. Les capteurs fournissent les vitesses linéaires odo_g et odo_d respectivement de la roue droite et gauche du véhicule ainsi que l'angle Δr de braquage de la roue avant. Ces valeurs sont symbolisées en rouge sur la figure 4.1. Le vecteur d'observation formé par l'odométrie vaut donc :

$$\mathbf{Y_k^{odo}} = \begin{bmatrix} \text{odo}_g \\ \text{odo}_d \\ \Delta r \end{bmatrix} \tag{4.11}$$

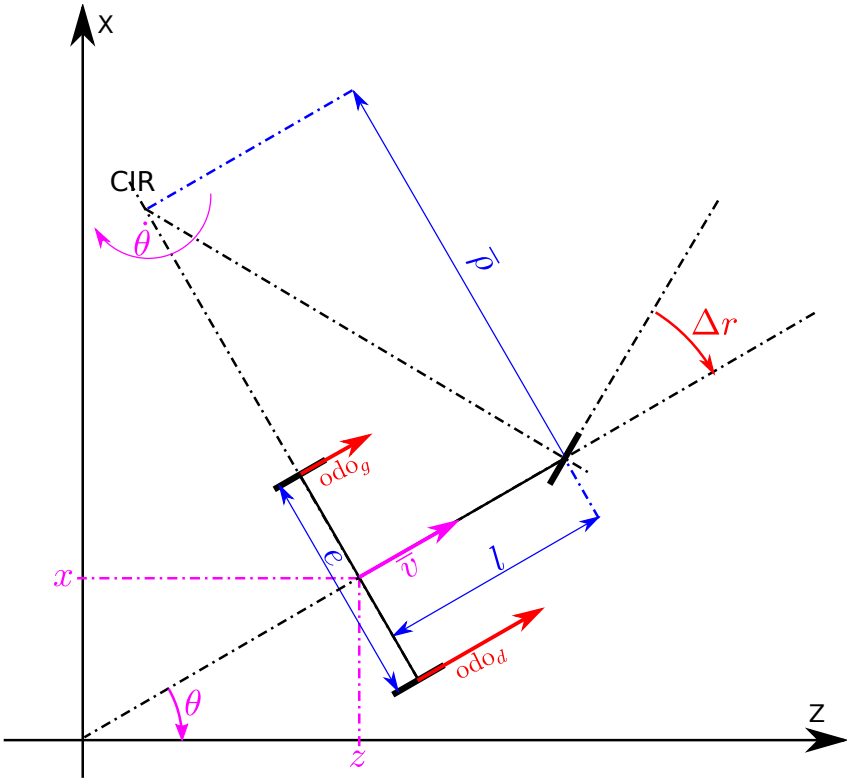


FIGURE 4.1 – Schéma en vue de dessus du véhicule (modèle tricycle)

Pour obtenir la fonction d'observation nécessaire au filtre de Kalman, il est nécessaire de relier ces valeurs au vecteur d'état. Pour rappel le vecteur d'état contient la position courante du véhicule (ses coordonnées x et z), son angle de cap θ, la vitesse linéaire $\overline{v}$ du véhicule et sa vitesse angulaire $\dot{\theta}$. Toutes ces valeurs sont représentées en magenta sur la figure 4.1.

Tout d'abord, il est nécessaire de connaitre la position du centre de rotation du véhicule. Ce centre de rotation, qui peut varier avec le temps, est appelé centre instantané de rotation (CIR). Etant le centre de rotation du véhicule, il est situé par définition à l'intersection des normales aux vecteurs vitesses de chaque point du véhicule. On suppose que le véhicule se déplace sans glissement, par conséquent la direction de la vitesse de chaque roue est dans le sens de roulement et le CIR se trouve donc à l'intersection des axes de rotation des roues. En utilisant le triangle rectangle (tracé sur la figure 4.1) passant par le CIR, le centre du véhicule et la roue avant, on peut déterminer la valeur algébrique $\overline{\rho}$ correspondant à la distance algébrique du centre du robot au CIR avec l'équation 4.12

$$\overline{\rho} = -\frac{l}{\tan \Delta r} \tag{4.12}$$

Par ailleurs, on sait que $\overline{\rho}$ permet de relier la vitesse angulaire et linéaire par la relation :

$$\overline{\rho} = -\frac{\overline{v}}{\dot{\theta}} \tag{4.13}$$

On en déduit donc directement la valeur de Δr :

$$\begin{aligned} \tan \Delta r &= \frac{l\dot{\theta}}{\overline{v}} \\ \Delta r &= \arctan(\frac{l\dot{\theta}}{\overline{v}}) \end{aligned} \tag{4.14}$$

Avec cette même relation, on obtient les valeurs des vitesses de chaque roue :

$$\begin{cases} \text{odo}_g = (\frac{\overline{v}}{\dot{\theta}} + \frac{e}{2})\dot{\theta} &= \overline{v} + \dot{\theta}\frac{e}{2} \\ \text{odo}_d = (\frac{\overline{v}}{\dot{\theta}} - \frac{e}{2})\dot{\theta} &= \overline{v} - \dot{\theta}\frac{e}{2} \end{cases} \tag{4.15}$$

Ces calculs permettent donc de définir une fonction d'observation h_{odo} associée à l'odométrie avec la relation :

$$\mathbf{Y_k^{odo}} = h_{odo}(\begin{bmatrix} x_k \\ z_k \\ \theta_k \\ \overline{v}_k \\ \dot{\theta}_k \end{bmatrix}, \mathbf{v_k}^{odo}) = \begin{bmatrix} \overline{v}_k + \dot{\theta}_k\frac{e}{2} \\ \overline{v}_k - \dot{\theta}_k\frac{e}{2} \\ \arctan\left(\frac{l\dot{\theta}_k}{\overline{v}_k}\right) \end{bmatrix} + \mathbf{v_k}^{odo} \tag{4.16}$$

Bruit de mesure

Le bruit de mesure, symbolisé par $\mathbf{v_k}^{odo}$ est ici directement ajouté aux mesures. Il est supposé gaussien et centré et sa matrice de covariance R^{odo} est constante et donnée directement par la précision du capteur. Dans le cadre de notre utilisation, les données sont mesurées indépendamment, leur covariance sera donc nulle. Pour avoir une estimation de la variance des valeurs on réalise un tracé des valeurs de l'odométrie au cours du temps, afin d'évaluer le bruit de mesure. On obtient les courbes de la figure 4.2. En mesurant globalement la hauteur des oscillations, on considère qu'il s'agit de la valeur de 3 fois l'écart-type. On obtient ainsi $3\sigma_{\text{odo}_g} \approx 0.1$ m/s (figure 4.2a) pour la vitesse des roues arrière, et $3\sigma_{\Delta r} \approx 0.01$ rad (figure 4.2b) pour l'incertitude sur l'angle de braquage. On obtient alors la matrice de variance/covariance suivante :

$$R^{odo} = \begin{bmatrix} \sigma^2_{\text{odo}_g} & 0 & 0 \\ 0 & \sigma^2_{\text{odo}_d} & 0 \\ 0 & 0 & \sigma^2_{\Delta r} \end{bmatrix} = \begin{bmatrix} 0.001 & 0 & 0 \\ 0 & 0.001 & 0 \\ 0 & 0 & 0.00001 \end{bmatrix} \tag{4.17}$$

Linéarisation de la fonction

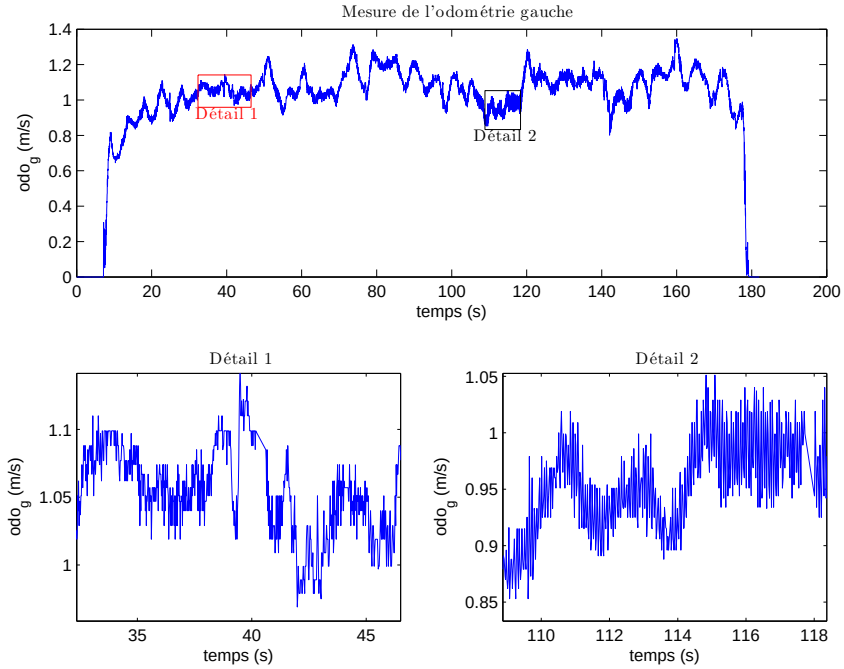
On peut remarquer que la fonction h_{odo} n'est dans ce cas pas linéaire, à cause de la présence de la fonction arc tangente. Il est donc nécessaire, pour utiliser ce modèle dans un filtre de Kalman, de la linéariser à chaque itération conformément au filtre de Kalman étendu. Pour cela, on doit donc calculer les deux jacobiennes J_k^{hX} et J_k^{hv}. Le bruit de mesure étant ajouté directement aux mesures, la fonction reste linéaire par rapport au vecteur $\mathbf{v_k}^{odo}$. Par conséquent la jacobienne J_k^{hv} est constante et égale à la matrice identité de taille 3×3.

La matrice J_k^{hX} est calculée en passant par la définition de la jacobienne :

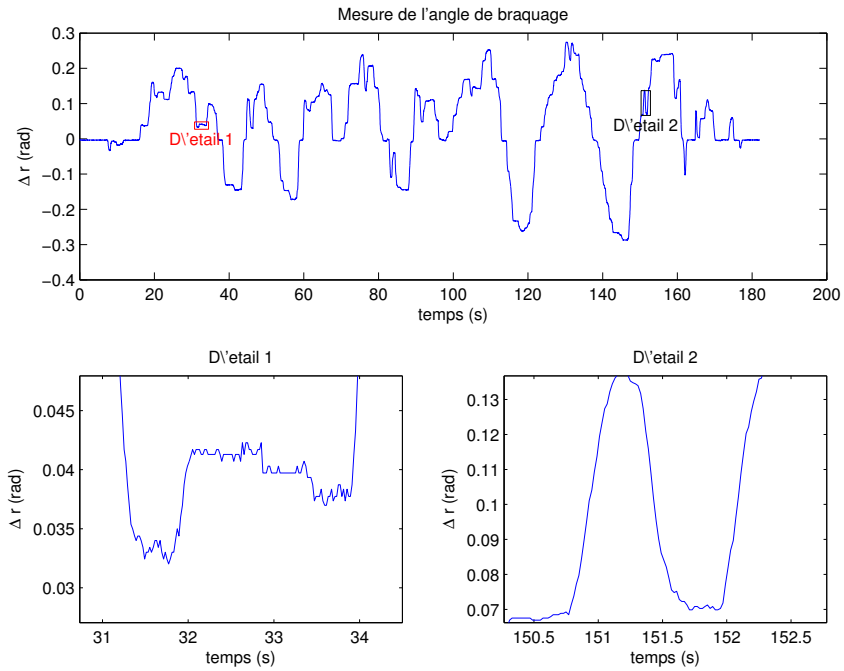
$$J_k^{hX} = \begin{bmatrix} \frac{\partial \text{odo}_g}{\partial x} & \frac{\partial \text{odo}_g}{\partial z} & \frac{\partial \text{odo}_g}{\partial \theta} & \frac{\partial \text{odo}_g}{\partial \overline{v}} & \frac{\partial \text{odo}_g}{\partial \dot{\theta}} \\ \frac{\partial \text{odo}_d}{\partial x} & \frac{\partial \text{odo}_d}{\partial z} & \frac{\partial \text{odo}_d}{\partial \theta} & \frac{\partial \text{odo}_d}{\partial \overline{v}} & \frac{\partial \text{odo}_d}{\partial \dot{\theta}} \\ \frac{\partial \Delta r}{\partial x} & \frac{\partial \Delta r}{\partial z} & \frac{\partial \Delta r}{\partial \theta} & \frac{\partial \Delta r}{\partial \overline{v}} & \frac{\partial \Delta r}{\partial \dot{\theta}} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & e/2 \\ 0 & 0 & 0 & 1 & -e/2 \\ 0 & 0 & 0 & \frac{-l\dot{\theta}}{\overline{v}^2+l^2\dot{\theta}^2} & \frac{l\overline{v}}{\overline{v}^2+l^2\dot{\theta}^2} \end{bmatrix} \tag{4.18}$$

Il faut également faire un cas particulier lorsque le véhicule est immobile. En effet, en l'absence de vitesse, il est impossible de calculer l'angle de braquage des roues. Ce phénomène se remarque par l'apparition d'une forme indéterminée pour les deux dernières valeurs de la dernière ligne de la matrice. Il s'agit d'un cas singulier. En effet lorsque le véhicule est immobile (vitesse linéaire et angulaire nulle), il est impossible de prédire l'angle de braquage des roues. Pour éviter des problèmes lors de l'implémentation, lorsque la vitesse angulaire et linéaire du véhicule est faible, ces deux valeurs sont fixées à 0.

On peut constater, en regardant l'équation 4.18 que les trois premières valeurs du vecteur d'état, qui correspondent à la position du véhicule, n'ont aucune influence sur les mesures. Ainsi, si l'odométrie nous permet de mettre à jour l'estimation de la vitesse du véhicule, les valeur de position prédite par l'équation d'état ne sont pas modifiée. La connaissance sur la vitesse reste tout de même une information intéressante dans le modèle, permettant en particulier d'obtenir une prédiction de la position suivante plus fiable. La localisation précise du véhicule dans l'environnement ne sera mesurée que par la vision.



(a) Vitesse de la roue gauche



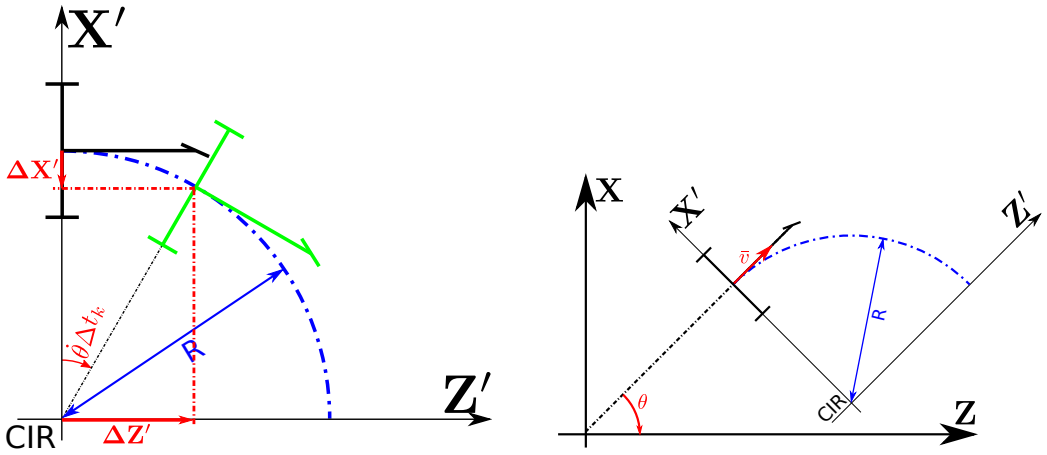
(b) Angle de braquage

FIGURE 4.2 – Représentation des données de l'odométrie

4.3.4 Modèle d'évolution

Après avoir défini les différentes fonctions d'observation associées à chaque capteur, il est nécessaire de définir le modèle d'évolution du véhicule. C'est ce modèle qui permet de prédire les valeurs du vecteur d'état à un instant t, à partir de l'état à un instant précédent.

Dans notre cas, le vecteur d'état contient la vitesse angulaire et linéaire du véhicule ainsi que sa position. Pour modéliser le mouvement du véhicule, on va supposer que sa vitesse reste constante au cours du temps, et ainsi déduire la nouvelle position en considérant l'avancement du véhicule. Pour obtenir l'équation d'état, on utilise toujours le modèle tricycle de la figure 4.1. Cependant pour écrire l'équation du mouvement, on va se placer dans un repère centré sur le CIR et orienté de telle manière que l'axe $\mathbf{X}'$ passe par le robot. Le véhicule est représenté dans ce repère sur la figure 4.3.



(a) Modèle du véhicule dans le repère centré sur le CIR

(b) Position du repère centré sur le CIR par rapport au repère global

FIGURE 4.3 – Représentation du véhicule en mouvement

Z

Équation d'état

Le temps Δt_k entre la prédiction et la dernière itération est supposé suffisamment court pour que le mouvement puisse être assimilé à un mouvement circulaire de rayon R autour du CIR qui reste à une position fixe. Ainsi, la rotation effectuée par le véhicule correspond à un angle de $\dot{\theta}\Delta t_k$. En observant la figure 4.3a, on en déduit l'expression du déplacement du robot suivant les axes X' et Z' :

$$\begin{bmatrix} \Delta X' \\ \Delta Z' \end{bmatrix} = \begin{bmatrix} R\cos(\dot{\theta}dt) \\ -R\sin(\dot{\theta}dt) \end{bmatrix} - \begin{bmatrix} R \\ 0 \end{bmatrix} = \begin{bmatrix} R(\cos(\dot{\theta}dt) - 1) \\ -R\sin(\dot{\theta}dt) \end{bmatrix} \tag{4.19}$$

La figure 4.3b permet d'écrire le lien entre les axes $\mathbf{X}'$, $\mathbf{Z}'$ et le repère global :

$$\mathbf{X}' = \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} \qquad \mathbf{Z}' = \begin{bmatrix} -\sin\theta \\ \cos\theta \end{bmatrix} \tag{4.20}$$

Pour obtenir la valeur du rayon R, on utilise également l'hypothèse que la vitesse linéaire $\overline{v}$ est constante. Ainsi la distance parcourue par le véhicule vaut $\overline{v}\Delta t_k$, et correspond à la longueur de l'arc de cercle de rayon R et d'angle $\dot{\theta}\Delta t_k$. Ce qui donne la relation $R = \frac{\overline{v}}{\dot{\theta}}$

On en déduit donc l'équation du déplacement $\mathbf{dpl}$:

$$\begin{aligned} \mathbf{dpl} &= R(\cos(\dot{\theta}\Delta t_k) - 1)\mathbf{X}' - R\sin(\dot{\theta}\Delta t_k)\mathbf{Z}' \\ &= R(\cos(\dot{\theta}\Delta t_k) - 1)\begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} - R\sin(\dot{\theta}\Delta t_k)\begin{bmatrix} -\sin\theta \\ \cos\theta \end{bmatrix} \\ &= \begin{bmatrix} R\Big(\cos\theta\cos(\dot{\theta}\Delta t_k) - \cos\theta + \sin\theta\sin(\dot{\theta}\Delta t_k)\Big) \\ R\Big(\sin\theta\cos(\dot{\theta}\Delta t_k) - \sin\theta - \cos\theta\sin(\dot{\theta}\Delta t_k)\Big) \end{bmatrix} \\ &= \begin{bmatrix} \frac{-\overline{v}}{\dot{\theta}}\Big(\cos(\theta - \dot{\theta}\Delta t_k) - \cos\theta\Big) \\ \frac{-\overline{v}}{\dot{\theta}}\Big(\sin(\theta - \dot{\theta}\Delta t_k) - \sin\theta\Big) \end{bmatrix} \end{aligned} \tag{4.21}$$

Tout cela nous donne au final l'équation d'état suivante :

$$\mathbf{X_{k+1}} = \begin{bmatrix} x_{k+1} \\ z_{k+1} \\ \theta_{k+1} \\ \overline{v}_{k+1} \\ \dot{\theta}_{k+1} \end{bmatrix} = \begin{bmatrix} x_k - \frac{\overline{v}_k}{\dot{\theta}_k}\Big(\cos(\theta_k - \dot{\theta}_k\Delta t_k) - \cos\theta_k\Big) \\ z_k - \frac{\overline{v}_k}{\dot{\theta}_k}\Big(\sin(\theta_k - \dot{\theta}_k\Delta t_k) - \sin\theta_k\Big) \\ \theta_k + \dot{\theta}_k\Delta t_k \\ \overline{v}_k \\ \dot{\theta}_k \end{bmatrix} + \mathbf{w_k} \tag{4.22}$$

Bruit de modèle

L'équation d'état n'est pas totalement réaliste. En effet l'hypothèse de vitesse constante n'est pas vérifiée puisque le véhicule accélère et ralentit (ne serait-ce qu'au démarrage et à l'arrêt) et réalise des virages par moment, modifiant ainsi sa vitesse angulaire. Cependant ces modifications sont progressives et restent assez faibles entre deux itérations pour que la prédiction réalisée ne soit pas trop mauvaise. Afin de permettre aux mesures de corriger la prédiction, il est nécessaire d'associer une incertitude au modèle de prédiction. Pour ce faire on va considérer qu'un bruit constant s'ajoute à chaque valeur du vecteur d'état. Ces bruit sont supposés gaussiens, centrés et décorrélés entre eux.

Au niveau de l'implémentation de l'EKF, cela signifie que la matrice de covariance Q_k associée au bruit de modèle restera constante à chaque itération. Dans notre implémentation, on a pris $Q_k = 0.00001I$ avec I la matrice identité de taille 5×5.

Linéarisation de l'équation d'état

L'équation d'état n'est pas linéaire, du fait des fonctions trigonométriques. Par conséquent, pour l'utiliser dans le filtre de Kalman étendu, il est nécessaire d'en calculer les jacobiennes en fonction du bruit et en fonction de l'état.

Le bruit étant ajouté directement aux variables d'état, la fonction est linéaire par rapport à $\mathbf{w_k}$. Par conséquent la jacobienne J_k^{fw} est constante et égale à la matrice identité de taille 5×5.

Le calcul de la jacobienne J_k^{fX} est lui beaucoup plus complexe. Le calcul, détaillé en annexe B permet d'obtenir la valeur suivante :

$$J_k^{fX} = \begin{bmatrix} 1 & 0 & \frac{\overline{v}_k}{\dot{\theta}_k}\left(s_{\alpha_k} - s_{\theta_k}\right) & \frac{1}{\dot{\theta}_k}\left(c_{\theta_k} - c_{\alpha_k}\right) & \frac{\overline{v}_k}{\dot{\theta}_k^2}\left(-c_{\theta_k} - \Delta t_k \dot{\theta}_k s_{\alpha_k} + c_{\alpha_k}\right) \\ 0 & 1 & \frac{-\overline{v}_k}{\dot{\theta}_k}\left(c_{\alpha_k} - c_{\theta_k}\right) & \frac{1}{\dot{\theta}_k}\left(s_{\theta_k} - s_{\alpha_k}\right) & \frac{\overline{v}_k}{\dot{\theta}_k^2}\left(-s_{\theta_k} + \Delta t_k \dot{\theta}_k c_{\alpha_k} + s_{\alpha_k}\right) \\ 0 & 0 & 1 & 0 & \Delta t_k \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.23}$$

avec

$$\begin{aligned} c_{\theta_k} &= \cos\theta_k & s_{\theta_k} &= \sin\theta_k \\ c_{\alpha_k} &= \cos(\theta_k - \dot{\theta}_k \Delta t_k) & s_{\alpha_k} &= \sin(\theta_k - \dot{\theta}_k \Delta t_k) \end{aligned}$$

Lorsque le véhicule se déplace en ligne droite, la vitesse angulaire se rapproche de 0. Une forme indéterminée apparait alors dans la matrice. Il est donc nécessaire d'en calculer la limite dans ce cas, pour modifier l'implémentation lorsque la valeur devient faible (inférieure à un seuil défini a priori). On obtient alors :

$$\lim_{\dot{\theta}_k \to 0} J_k^{fX} = \begin{bmatrix} 1 & 0 & -\overline{v}_k \Delta t_k c_{\theta_k} & -\Delta t_k s_{\theta_k} & \frac{\overline{v}_k \Delta t_k^2}{2} c_{\theta_k} \\ 0 & 1 & -\overline{v}_k \Delta t_k s_{\theta_k} & \Delta t_k c_{\theta_k} & \frac{\overline{v}_k \Delta t_k^2}{2} s_{\theta_k} \\ 0 & 0 & 1 & 0 & \Delta t_k \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.24}$$

4.4 Implémentation

Au bilan toutes ces modélisations permettent d'implémenter un filtre de Kalman étendu qui prédira la position du véhicule à chaque instant. Pour cela, à chaque donnée image reçue, la position du véhicule est prédite pour initialiser l'algorithme de localisation. Puis le résultat de la localisation est envoyé au filtre qui, par le biais de la matrice H_{vis}, met à jour la position. De même lorsque des données venant de l'odomètre sont disponibles, une prédiction suivi de la mise à jour utilisant la fonction h_{odo} permet d'améliorer la valeur d'état du véhicule. Cela permet

d'avoir en permanence une position aussi précise que possible, tout en conservant l'incertitude sur la position.

Cependant, au niveau de l'implémentation on constate une difficulté supplémentaire, liée au temps de localisation nécessaire en utilisant les données image. En effet l'analyse d'une image avec la reprojection des patchs et l'appariement pour déterminer la position du véhicule prend un temps non négligeable pendant lequel plusieurs données provenant de l'odométrie sont disponibles. Ainsi la mesure de la vision suite à une image perçue à un temps t_1^{image} n'est disponible qu'à un temps t_1^{vis}. Pendant ce temps de calcul, l'état du filtre de Kalman a pu évoluer en fonction de l'odométrie. On peut voir un schéma représentant l'arrivée des différentes données au cours du temps sur la figure 4.4. Ainsi, une donnée de position arrivant à un temps t_1^{vis} correspond en fait à la position du véhicule à l'instant t_1^{image} où l'image a été perçue et doit donc servir à mettre à jour le modèle pour cet instant. Les données de l'odométrie arrivées entre t_1^{image} et t_1^{vis} (nommées t_2^{odo} à t_5^{odo} sur la figure 4.4) devront être prises en compte dans le calcul après avoir le calcul de cette position.

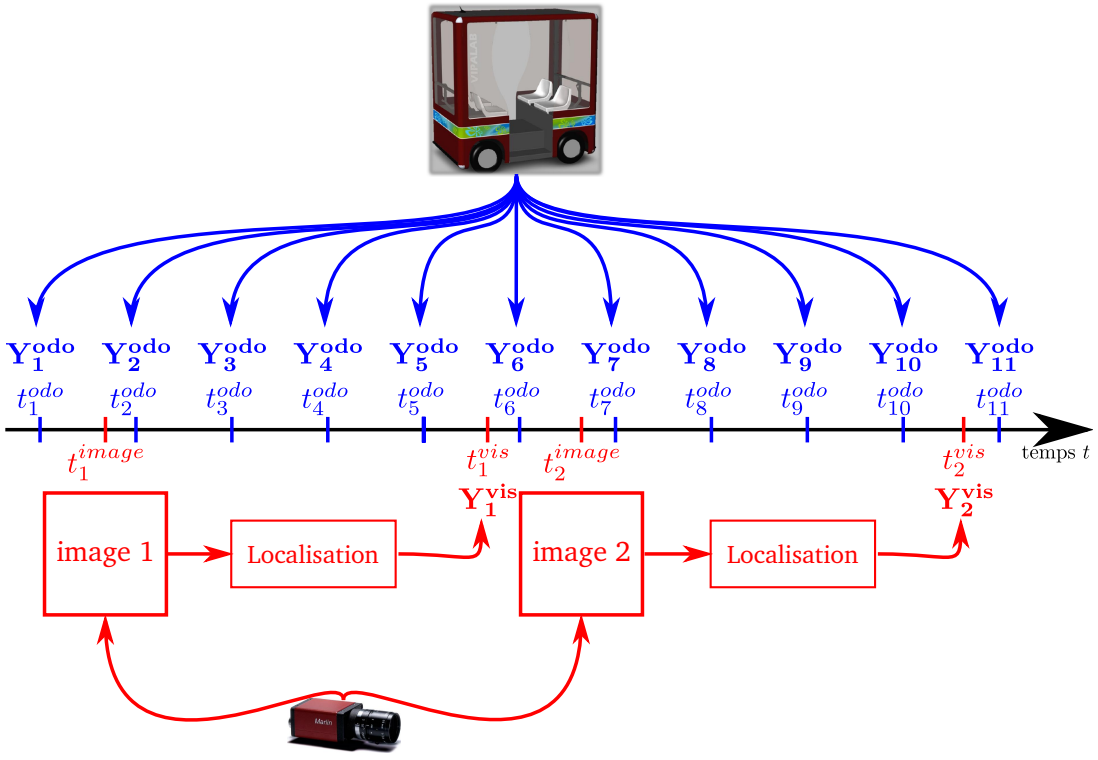


FIGURE 4.4 – Représentation de l'arrivée des différentes informations au cours du temps. En particulier on peut constater le temps entre l'enregistrement d'une image et la détermination de sa position. Pour l'odométrie le temps de calcul de la position est suffisamment faible pour être négligé.

Pour réaliser cette tâche, on utilise la notion de liste d'observation, et on stocke au fur et à

mesure l'ensemble des mesures de position calculées (par l'odométrie ou par vision), le vecteur d'état du filtre après la mise à jour et le temps auquel correspond la mesure. Ainsi quand au temps t_1^{vis}, une donnée vision arrive, fournissant une mesure datée t_1^{image}, la mise à jour peut se faire en partant de la donnée précédente la plus récente, en l'occurrence sur le schéma celle de t_1^{odo}. Ensuite la position au temps t_1^{image} ayant été mise à jour, la prédiction au temps t_2^{odo} est refaite, et la mise à jour associée à $\mathbf{Y_2^{odo}}$ est recalculée en tenant compte de la nouvelle prédiction. La modification est ensuite reportée sur la mesure suivante, et ainsi de suite jusqu'à avoir parcouru toute la liste d'observation. Cela permet de conserver la meilleure prédiction possible en ne perdant aucune mesure temporelle.

4.5 Résultats expérimentaux

L'implémentation ainsi décrite permet donc de réaliser une fusion entre l'odométrie et la vision. Les premiers résultats ont montré une précision comparable à celle de la vision seule, et donc pas de gain mesurable au niveau de la prédiction. Cependant l'utilisation du filtre de Kalman permet d'avoir une méthode peu couteuse en temps de calcul qui reste négligeable par rapport au temps de calcul de la pose et dont les bénéfice reste à être mesurée lors des expérimentations.

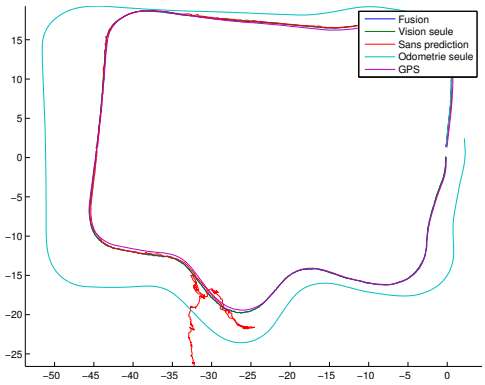
4.5.1 Localisation simple

Pour évaluer l'intérêt du filtre de Kalman, et de la fusion par rapport à l'utilisation de la seule vision, une première étude a été réalisée. Comme lors des essais précédents, une trajectoire d'apprentissage a été réalisée pour calculer les normales. La trajectoire, d'une distance d'environ 200 m, a permis de générer 23 320 patchs ayant un bon critère de qualité. Par la suite, le véhicule a été conduit manuellement en longeant la trajectoire d'apprentissage en restant décalé d'environ 1,5 m à gauche. Ensuite la localisation en se basant sur les patchs a été réalisée avec 3 méthodes pour générer la pose prédite :

- ➢ Sans prédiction en prenant directement la pose précédente avec une variance fixe
- ➢ Vision seule utilisant le filtre de Kalman mais pas de donnée d'odométrie
- ➢ Fusion utilisant l'algorithme complet décrit précédemment

La localisation a par ailleurs été également estimée en utilisant uniquement les données de l'odométrie. Dans ce cas, la trajectoire obtenue est connue uniquement dans le plan du sol. La figure 4.5 montre une vue de dessus de la trajectoire suivie ainsi que la référence. La trajectoire fait le tour de la zone, en suivant la file de droite lors de l'apprentissage et la file de gauche lors de la localisation. La référence est fournie par un GPS différentiel de précision centimétrique. La superposition des résultats avec la référence a nécessité des changements de repère qui seront détaillés dans la partie 5.2.2.

(a) Trace GPS réalisée lors de la référence et de la localisation. Les traces ont été recalées sur une vue virtuelle (extraite du simulateur ASROCCAM) de la zone de test et sont placée dans le repère géographique (le nord vers le haut).

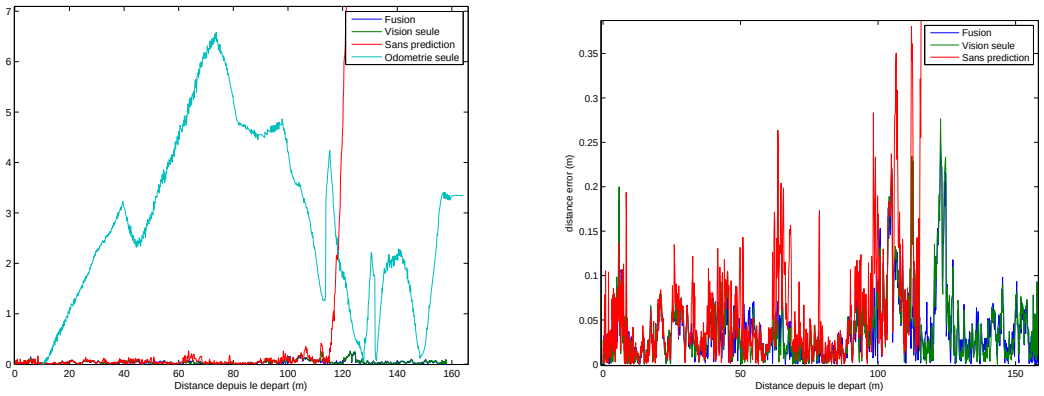


(b) Vue de dessus de la trajectoire réalisée pour tester la fusion avec odométrie. Les trajectoires sont placées dans le repère monde de la vision.

FIGURE 4.5 – Trajectoire réalisée pour tester la fusion avec odométrie

On peut voir que l'odométrie présente une très forte dérive qui rendrait la localisation très imprécise si elle était utilisée seule. La trajectoire fusionnée, tout comme celle utilisant la vision est par contre beaucoup plus précise et suit correctement les données GPS. La trajectoire sans prédiction fonctionne correctement au début de la séquence mais se perd à la fin au niveau du rond-point. Sur le début de la séquence, les trajectoires calculées par localisation sont difficiles à différencier, ce qui semble indiquer une précision similaire. Pour s'en convaincre, on a mesuré en chaque instant l'erreur de localisation. La mesure de cette erreur est faite de manière indépendante de l'erreur de reconstruction et sera détaillée en partie 5.2.3. On obtient les courbes représentées figure 4.6. Hormis l'odométrie qui, à cause de sa dérive, reste très distante de la pose précise, les courbes de localisation présentent toute une erreur ayant le même ordre de grandeur. Les valeurs moyennes des erreurs de localisation pour chaque méthode ont été calculées. Afin de ne pas avoir de biais lié au moment où la localisation sans prédiction s'est perdue, seule les 110 premiers mètres de la trajectoire sont analysés, pour obtenir les valeurs du tableau 4.2.

On peut voir que l'utilisation d'un filtre de Kalman apporte une très légère amélioration de la précision, puisque moyenne et écart-type sont tous plus élevés sans prédiction. Par contre l'utilisation de l'odométrie n'a aucune influence notable sur l'erreur de localisation dans ce cas.



(a) Erreur de localisation en 2D (en considérant les projections dans le plan du sol)

(b) Erreur de localisation en 3D

FIGURE 4.6 – Erreur de localisation avec chaque modèle. L'odométrie n'est disponible qu'en 2D, et à cause de la dérive reste à une grande distance de la position de référence.

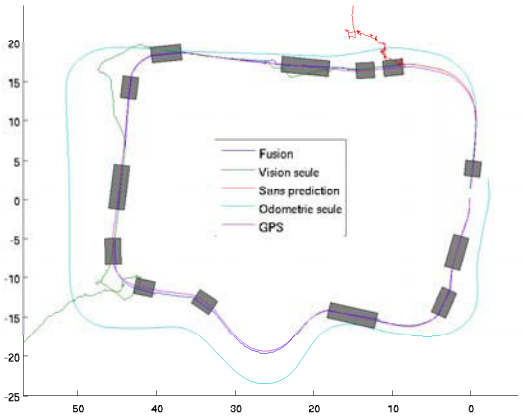
	moyenne(m)	écart-type(m)	médiane (m)	maximum (m)
Odométrie	3.423	1.657	3.414	6.584
Fusion	0.035	0.030	0.027	0.198
Vision seule	0.035	0.030	0.026	0.202
Sans prédiction	0.051	0.050	0.035	0.350

TABLE 4.2 – Valeur de l'erreur de localisation en utilisant les différents modèles. Les valeurs considérées ont été limitées aux 110 premiers mètres de la trajectoire.

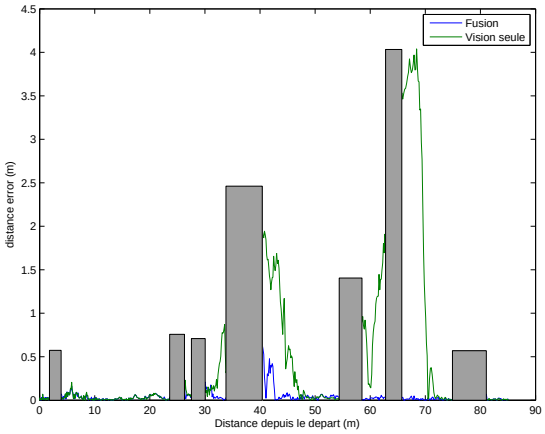
4.5.2 Localisation avec absence d'image

Si la fusion n'apporte pas de précision dans ce premier cas, c'est parce que les poses de caméra arrivent régulièrement et permettent de mettre à jour la position sans nécessiter un modèle de prédiction complexe. De plus, il s'agit d'un test réalisé hors ligne, mais lorsque les données images sont moins fréquentes ou que, pour une raison quelconque (sous ou sur-exposition du capteur, problème de transfert des images) les données images cessent de parvenir, il est important de conserver une bonne prédiction. Pour simuler ce comportement, l'essai a été renouvelé sur la même trajectoire mais en supprimant quelques images pour simuler un problème ponctuel. La figure 4.7 montre la vue de dessus de la trajectoire localisée (a) et l'erreur de localisation (b).

On s'aperçoit que l'absence ponctuelle de données empêche la localisation de se faire au-delà du troisième virage en utilisant une prédiction avec la vision seule, et ne passe même pas le premier virage sans prédiction. De plus, même dans la partie précédente, la mesure de position utilisant la vision seule est moins précise que celle utilisant la fusion. En mesurant l'erreur de



(a) Vue de dessus de la trajectoire calculée.



(b) Erreur de localisation pour la prédiction avec et sans odométrie. La distance n'a été mesurée que sur la première moitié de la trajectoire (avant que la localisation sans odométrie ne se perde). L'erreur de localisation sans prédiction n'est pas montrée car la localisation n'était pas réalisée dans ce cas.

FIGURE 4.7 – Résultat de la localisation lorsque des images sont manquantes. Les rectangles gris sur chaque image correspondent aux parties sans images

localisation de la figure 4.7b, on obtient les résultats du tableau 4.3.

	moyenne(m)	écart-type(m)	médiane (m)	maximum (m)
Odométrie	3.353	1.811	2.917	6.583
Fusion	0.040	0.067	0.022	0.574
Vision seule	0.426	0.877	0.034	3.916
Sans prédiction	13.804	15.249	5.412	43.091

TABLE 4.3 – Valeur de l'erreur de localisation en utilisant les différents modèles lorsque des images sont manquantes. Les valeurs considérées ont été limitées à la première moitié de la trajectoire.

La différence entre la méthode avec et sans odométrie s'explique car lorsque de nombreuses données sont indisponibles, en particulier si un virage a lieu pendant cette indisponibilité, la prédiction devient erronée. L'algorithme de localisation utilisant cette prédiction pour reprojeter les patchs, l'appariement est moins facile à faire et moins d'appariements corrects sont trouvés. A l'inverse lorsque l'odométrie est utilisée, les virages peuvent être anticipés et la prédiction devient plus fiable, permettant aux patchs correctement reprojetés de s'apparier avec les points d'intérêt qui leur correspondent. Ce phénomène est bien visible sur la figure 4.8, représentant le

nombre de patchs correctement appariés au fur et à mesure de la localisation et l'on voit que ce nombre chute grandement lors de la localisation avec la vision seule. Par contre dans les zones

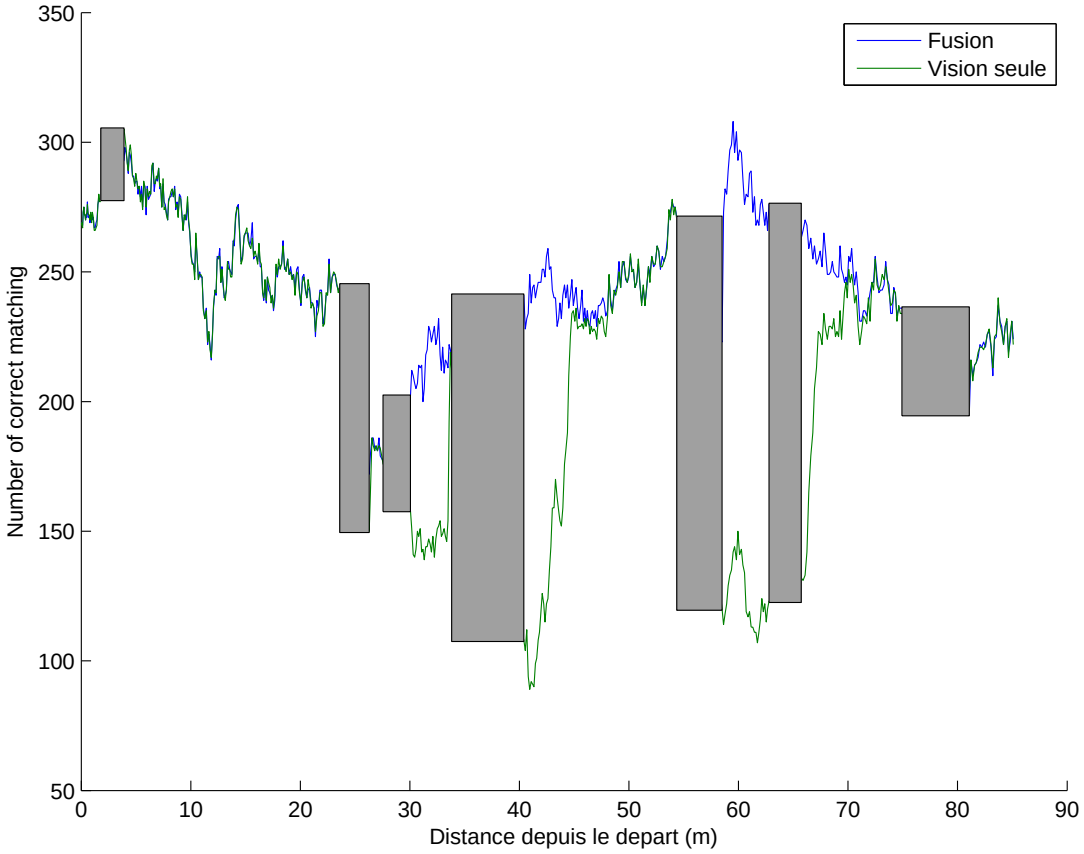


FIGURE 4.8 – Nombre d'appariements corrects au fur et a mesure de la trajectoire. Les rectangles gris correspondent aux parties sans images.

où toutes les images sont disponibles, la prédiction reste bonne dans les deux cas et le nombre d'appariements, ainsi que la précision sont les mêmes avec les deux méthodes.

Un autre intérêt à l'utilisation de la méthode de fusion est de permettre une bonne estimation de la pose du véhicule en chaque instant, en particulier entre deux acquisitions d'image. Lorsque le véhicule est piloté de manière autonome, il est nécessaire de connaitre sa position très fréquemment afin d'envoyer les ordres aux actionneurs régulièrement. L'utilisation du filtre de Kalman et de la fusion avec l'odométrie permet de le faire quel que soit le temps depuis la dernière image et en particulier de mesurer immédiatement l'effet des actionneurs dans les virages grâce aux informations de l'odométrie sans attendre la localisation venant de l'image suivante.

Pour conclure, on a ainsi obtenu un algorithme fonctionnel et peu sensible aux perturbations des différents capteurs en bénéficiant des avantages de la vision et de l'odométrie. L'utilisation du filtre de Kalman permet, sans alourdir le calcul, d'obtenir une position fiable utilisable pour commander le véhicule en chaque instant, ainsi qu'un ordonnancement efficace permettant de conserver l'ensemble des informations disponibles.

4.6 Perspectives

L'utilisation d'un modèle de prédiction tel que celui-ci permet d'avoir un système pleinement fonctionnel. Cependant plusieurs hypothèses ont été faites pour modéliser le véhicule et certaines pourrait être reconsidérées. Tout d'abord pour le filtre lui-même, les modèles utilisés pourraient être perfectionnés pour obtenir une meilleure évaluation de la position et de l'incertitude sur celle-ci. Mais aussi au niveau de l'utilisation de la prédiction, avoir une meilleure information pourrait aider à simplifier l'algorithme de vision et optimiser les calculs.

4.6.1 Amélioration du modèle d'évolution

Tout d'abord le modèle d'évolution a été choisi sous hypothèse de vitesse constante avec un bruit constant sur chaque valeur du vecteur d'état. Ces valeurs ont été fixées arbitrairement en fonction des expérimentations. Cependant une étude plus poussée du bruit provoqué pourrait permettre d'améliorer cette évolution. En particulier le bruit pourrait être considéré comme une accélération et, en développant les équations, être intégré plus finement au modèle, ce qui permettrait d'avoir une estimation de l'incertitude sur la position plus réaliste.

De même il est possible dans les équations du filtre de Kalman, d'intégrer les ordres donnés par la commande. Cela permet d'ajuster la prédiction du véhicule en anticipant non seulement les mouvements réalisés mais aussi ceux que l'on désire réaliser.

Pour finir le déplacement du véhicule a été entièrement projeté dans le plan de la route. Pour cela les données concernant le roulis et le tangage, disponibles avec la vision ont été ignorées. Modéliser le mouvement en 3D du véhicule permettrait de prédire l'intégralité de la pose du véhicule. Cependant le filtre deviendrait sensiblement plus complexe, en particulier au niveau de l'interprétation des valeurs de l'odométrie.

4.6.2 Utilisation plus poussée de la prédiction

Une autre amélioration qui pourrait être réalisée suite à cette implémentation consiste à utiliser la prédiction pour améliorer davantage la localisation. En effet, la prédiction étant devenue plus fiable il serait possible au lieu de ne l'utiliser que pour reprojeter les patchs, de prévoir lesquels ont le plus de chance d'être retrouvés. En triant a priori les patchs en fonction de cette probabilité il devient possible de ne projeter qu'un nombre limité de patchs à chaque itération

et ainsi garantir un temps constant entre l'acquisition de l'image et l'accès au résultat du calcul de la pose.

Cette méthode empêcherait sans doute un certain nombre de bons appariements de se faire, mais tant que suffisamment de patchs de référence sont appariés pour se localiser, l'intérêt d'avoir davantage de points reste discutable. De plus les patchs peu intéressants seraient ignorés, permettant de gagner à la fois du temps de calcul et d'éviter de faux appariements. La principale difficulté pour cette solution consiste à définir un facteur qui permettrait de savoir a priori si un patch (ou même un groupe de patchs) est plus intéressant à rechercher qu'un autre en fonction de la pose prédite.

Chapitre 5

Résultats expérimentaux de la localisation

Après avoir réalisé avec succès les tests préliminaires, le système a été mis en œuvre dans des conditions réelles.

5.1 Matériel utilisé

Pour réaliser les expérimentations, il est nécessaire de disposer au minimum d'une caméra et d'un ordinateur qui exécute l'algorithme. De plus, la fusion nécessite d'embarquer le dispositif à bord d'un véhicule disposant d'odométrie. Pour finir un GPS différentiel est utilisé comme capteur de référence, fournissant une position considérée comme exacte. L'algorithme peut fonctionner avec toute sorte de matériel, toutefois les dernières expérimentations ont essentiellement utilisé le matériel décrit dans cette partie. Si tout ou partie du matériel diffère lors d'une expérience, cela sera précisé au cas par cas.

5.1.1 Caméra

Les caméras utilisés sont des caméra CMOS noir et blanc. Les images fournies ont une résolution de 1024×768 pixels, mais pour accélérer le traitement, sont ré-échantillonnées à une résolution deux fois moindre de 512×384 pixels. Ce rééchantillonnage est réalisé directement par le GPU en même temps que la correction d'image présentée en partie 3.4.2. Les valeurs des pixels de l'image corrigée sont prises directement sur l'image d'origine (avant rééchantillonnage), permettant ainsi de ne pas nécessiter davantage de temps tout en réduisant l'image.

La caméra peut fournir des images à une fréquence de 15 images par seconde, mais cette vitesse est généralement réduite à 7,5 images par seconde pour laisser le temps à l'algorithme de réaliser tous les traitements nécessaires.

L'objectif utilisé est un objectif à très grand angle. En tenant compte de la distorsion du capteur, le champ de vue réel atteint 130 degrés dans la largeur de l'image. L'utilisation d'un tel objectif permet de réduire les risques d'occultation. Par exemple un objet présent à 2 m

de l'objectif n'occupera que peu de place sur l'image, réduisant le nombre de points d'intérêt masqués. De plus cela permet d'avoir une bonne visibilité sur les façades des immeubles au bord de la route. Les façades sont généralement planes et les patchs observés sont généralement bien reconstruits et plus faciles à reconnaitre.

Pour finir, la caméra dispose d'un diaphragme automatique permettant d'augmenter ou réduire l'intensité lumineuse perçue en fonction de la lumière ambiante. Ce dispositif est sensé limiter les risques de sous-exposition ou sur-exposition de l'image. Cependant en pratique, la situation n'est pas toujours bien maitrisée, en particulier lorsque le soleil est assez bas et peut éclairer directement les façades d'un coté de la rue en laissant celles de l'autre coté dans l'ombre. Dans ce cas la dynamique de la caméra n'est généralement pas suffisante pour fournir une information correcte des deux façades, et il faut alors choisir de sous exposer ou sur-exposer un coté de la rue.

5.1.2 Matériel informatique

Pour utiliser l'algorithme dans un système embarqué, il est nécessaire de disposer d'un ordinateur portable assez puissant pour faire fonctionner le système dans de bonnes conditions. En particulier l'appariement étant implémenté sur GPU, la carte graphique doit être performante et compatible avec le langage CUDA utilisé. Le processeur est utilisé pour gérer l'arrivée des données, le calcul de pose et, dans le cas de navigation autonome, la génération et l'envoi de la commande.

L'ordinateur est équipé d'un processeur Intel Core i5 avec 2 cœurs physiques cadencé à 2,5 GHz et de 8 Gio de mémoire RAM. Il s'agit d'un processeur milieu de gamme. Au niveau de la carte graphique, il s'agit d'une carte Nvidia GeForce GTX 485M, possédant un multiprocesseur avec 48 cœurs et 2 Gio de mémoire, cette carte assez performante permet de faire fonctionner l'algorithme en temps-réel.

5.1.3 Véhicule

Le véhicule sur lequel la plupart des expérimentations ont été réalisées est le Vipalab (photo de la figure 5.1). Ce véhicule électrique est capable de transporter 4 personnes à des vitesses inférieures à 20 km/h. Il dispose de deux caméras placées au niveau du toit, l'une orientée vers l'avant du véhicule et l'autre vers l'arrière. Lors de nos expérimentations, seule la caméra située à l'avant est utilisée pour se localiser. Le véhicule dispose également d'odométrie capable de fournir toutes les 20 ms les informations sur la vitesse du moteur des roues arrières et l'angle de braquage à l'avant.

5.1.4 GPS

Pour mesurer la vérité terrain lors des expérimentations, un GPS différentiel est également utilisé. Ce GPS, situé sur le toit du Vipalab, est couplé à une base fixe pour fonctionner comme

FIGURE 5.1 – Le Vipalab

GPS différentiel ayant une précision centimétrique. Les données GPS sont disponibles à une fréquence de 10 Hz.

5.2 Méthode d'évaluation des résultats

5.2.1 Reconstruction initiale

Pour pouvoir calculer l'orientation des patchs et se localiser, il est nécessaire de disposer d'une reconstruction 3D de l'environnement. Cette reconstruction est générée en utilisant l'algorithme décrit dans la partie 1.3.1. Cependant on constate que cette reconstruction est loin d'être parfaite. En particulier une dérive de la reconstruction se produit, et lorsque l'on revient à la position de départ après avoir parcouru une boucle, la reconstruction peut montrer une erreur importante.

D'autre part cette reconstruction étant réalisée uniquement avec des données vision, le repère est défini à un facteur d'échelle près. Pour s'approcher d'une mesure métrique, il est possible d'utiliser la distance parcourue totale, par exemple en utilisant l'odométrie. Cependant ce

facteur d'échelle peut varier au fur et à mesure de la trajectoire, ce qui explique en partie la dérive de la reconstruction.

Pour améliorer la qualité de la reconstruction, l'algorithme a été modifié pour pouvoir utiliser les deux caméras du véhicule. Cet algorithme, détaillé dans (Lébraly, 2012), suppose que la distance entre les deux caméras est connue. Ainsi, en utilisant conjointement les points repérés avec chacune des deux caméras, il devient possible de corriger la dérive du facteur d'échelle. De plus un algorithme optimisant l'ensemble des paramètres en se basant sur la fermeture des boucles, c'est-à-dire en corrigeant les erreurs de bouclage lors d'un second passage au même endroit, permet également d'obtenir une reconstruction plus fiable. Cette méthode avec deux caméras a été utilisée dans plusieurs expérimentations. Néanmoins les deux caméras ont été utilisé uniquement pour générer la reconstruction initiale. Les vues de la caméras arrière n'ont pas été utilisés pour calculer les patchs utilisé lors de la localisation.

5.2.2 Changement de repère du GPS

Pour pouvoir comparer les données obtenues lors de la localisation avec la vérité terrain fournie par le GPS, il est nécessaire de réaliser un changement de repère. En effet les données de localisation sont fournies dans le repère de la reconstruction qui est indépendant du repère géoréférencé utilisé par le GPS.

Afin de pouvoir superposer les trajectoires vision et GPS, il est donc nécessaire de trouver la transformation géométrique qui permet de passer d'un repère à l'autre. Pour cela on se base sur la reconstruction initiale que l'on compare à la trace GPS.

On commence tout d'abord par décaler la localisation pour la placer à la même position que le GPS. Ceci est fait en mesurant la distance dans le véhicule entre la caméra et le récepteur GPS et la direction par rapport à l'axe de la caméra dans lequel ce déplacement doit être fait. Les positions fournies par la vision étant orientées, il est alors possible d'y appliquer la rotation et translation ainsi mesurées pour replacer la localisation à l'endroit où se trouve le GPS. Cette transformation est également réalisée lors de la localisation pour replacer la pose de caméra au milieu de l'essieu arrière où sont mesurées les données de l'odométrie.

Connaissant les localisations faites par vision et par le GPS du véhicule, les données peuvent alors être recalées les unes sur les autres en calculant une matrice de rotation, une translation et un changement d'échelle qui recale au mieux les points. On obtient ainsi une matrice de transformation qui permet de recaler au mieux l'ensemble de la trajectoire d'apprentissage entre les données vision et les données GPS.

Cette matrice de passage peut ensuite être appliquée aux trajectoires de test pour placer la référence et les données de localisation dans le même repère. C'est avec cette méthode que l'on peut tracer des vues de dessus de la trajectoire, telle que la figure 4.5b dans la partie 4.5.1.

Cependant si cette méthode de recalage globale permet d'obtenir des résultats visuellement corrects, lorsque l'on veut mesurer précisément l'erreur de localisation, cela ne suffit pas. En effet, la reconstruction initiale n'est pas parfaite, et présente généralement une dérive globale qui fait que la transformation de l'ensemble de la trajectoire n'est pas parfaite. Si l'on prend la

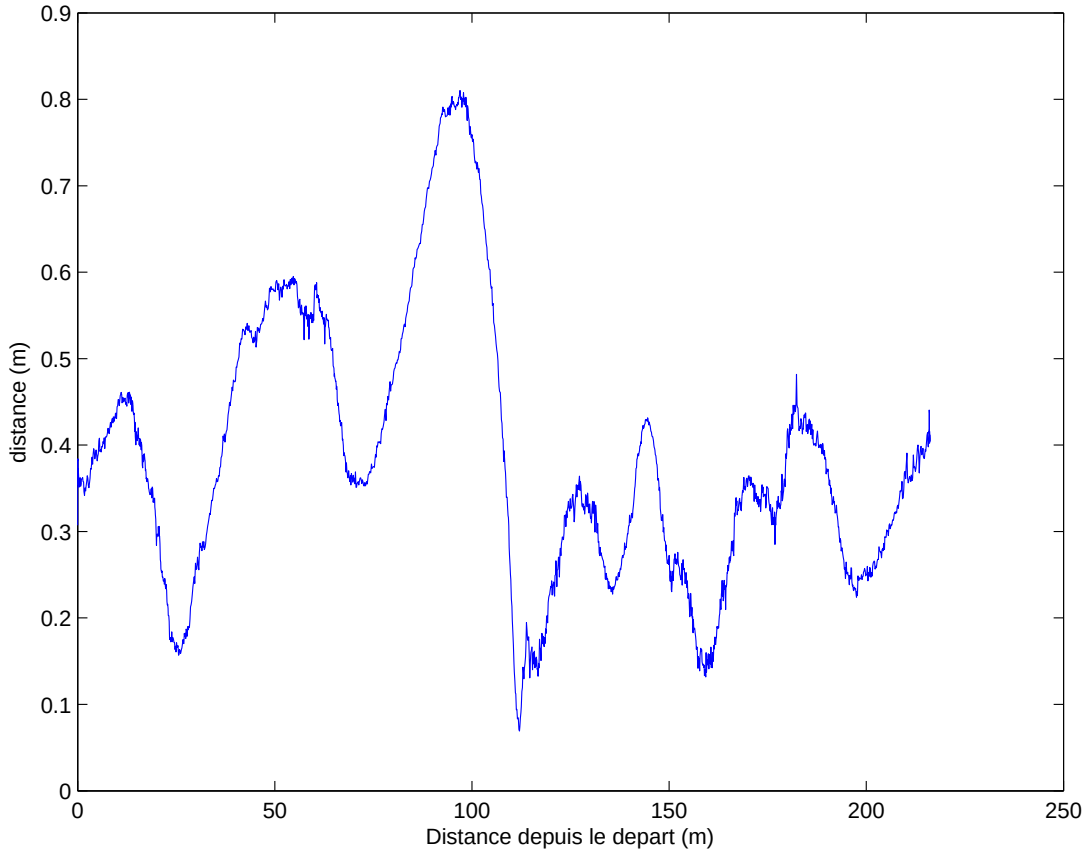


FIGURE 5.2 – Distance entre la trajectoire recalée et le GPS pour la pose de référence.

même trajectoire que celle présentée dans la partie 4.5.1, et que l'on mesure à chaque position de la trajectoire d'apprentissage, la distance entre la position GPS et la position mesurée par la vision, on obtient la courbe de la figure 5.2. On constate que cette distance, qui représente une erreur de recalage, peut atteindre plus de 80 cm, avec une valeur moyenne autour de 40 cm. Il n'est donc pas possible d'évaluer précisément des erreurs de localisation en utilisant ce genre de transformation. En effet toute erreur inférieure à 1 m serait négligeable devant la seule erreur de reconstruction.

5.2.3 Calcul de l'erreur de localisation

Pour pouvoir mesurer l'erreur de localisation, de manière indépendante de l'erreur de reconstruction, il est donc nécessaire de trouver un autre moyen.

En effet, même si en position absolue et recalée sur le GPS, la position est très incertaine, cela n'implique pas que la localisation elle-même soit mauvaise. En effet, généralement, et en

particulier dans le cas de navigation autonome, la position absolue du véhicule n'a pas besoin d'être précise. Seule sa position localement, par rapport à la trajectoire d'apprentissage, doit être connue. C'est d'ailleurs cet écart à la trajectoire qui est utilisé lorsque le véhicule est piloté de manière autonome.

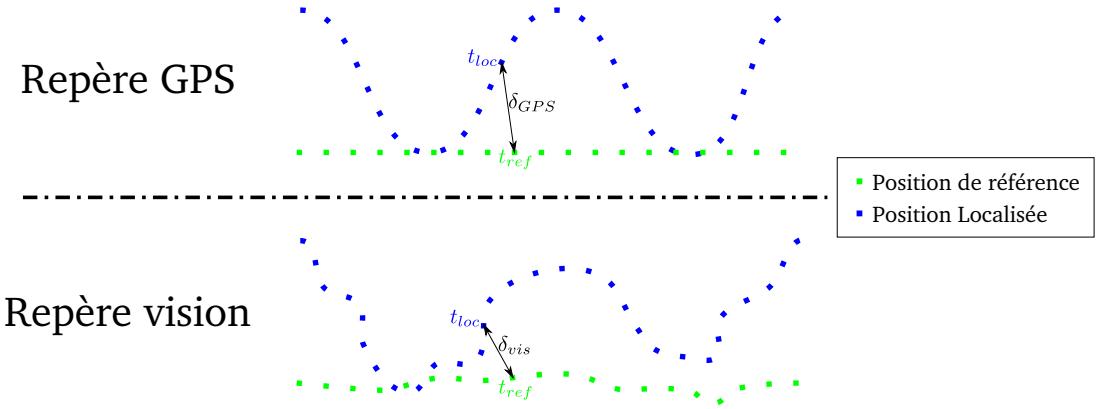


FIGURE 5.3 – Mesure de l'écart à la référence, utilisée pour calculer l'erreur de localisation.

Ainsi, ce qui est intéressant de mesurer est non pas l'erreur de position par rapport à un repère global, mais plutôt l'erreur d'évaluation de la distance δ entre la position localisée et la trajectoire d'apprentissage. Un exemple est montré sur la figure 5.3. Pour ce faire, à chaque instant t_{loc} où la localisation est réalisée, on détermine, en utilisant le GPS de référence, la position la plus proche de la trajectoire d'apprentissage. Cette position a été atteinte à un instant t_{ref}. Comme les données de localisation par GPS et par vision sont datées, il est possible d'obtenir la position du véhicule aux instant t_{ref} et t_{loc} avec le GPS et avec la vision. En mesurant la distance entre ces positions on obtient les valeurs δ_{GPS} et δ_{vis} qui, sont toutes deux l'évaluation de l'écart à la référence.

L'erreur de localisation est alors la différence entre δ_{GPS}, l'écart obtenu par le GPS qui nous sert de référence, et δ_{vis}, écart obtenue en utilisant notre algorithme de localisation. On obtient ainsi une évaluation de l'erreur en chaque instant et on peut tracer des courbes telles que celles de la figure 4.6 dans la partie 4.5.1.

5.3 Évaluation des limites de l'algorithme

5.3.1 Conditions expérimentales

Zone d'expérimentation

Pour se placer dans des conditions proches de l'environnement urbain, des expérimentations ont été menées sur la plateforme PAVIN (Plate-forme d'Auvergne pour Véhicules INtelligent). Cette plateforme issue d'un projet conjointement initié en 2005 par le CNRS et le LASMEA est

un environnement de 5 000 m² constitué de 317 mètres de voirie, d'un rond-point, de carrefours et de façades représentant un milieu urbain. La figure 5.4 présente une photo de cette plateforme.

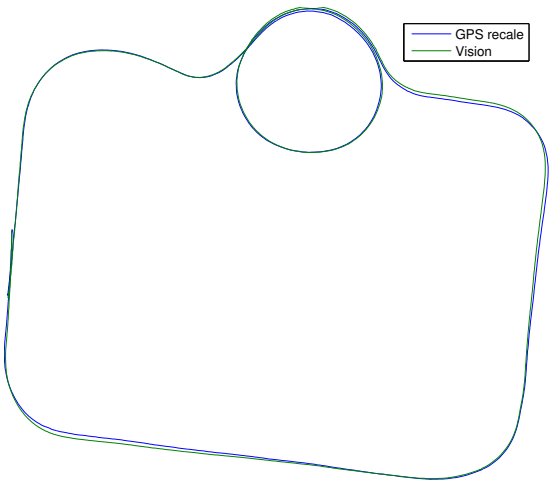
FIGURE 5.4 – Photo de la plateforme PAVIN (juillet 2012)

Cette plateforme, bien que représentative d'un milieu urbain, a cependant un environnement assez dégagé, permettant d'utiliser le GPS différentiel en conservant une précision centimétrique. Ainsi il est possible de connaitre la véritable position du véhicule en utilisant ce capteur de référence.

Reconstruction initiale

Pour réaliser la reconstruction de l'environnement, à partir de laquelle les patchs seront calculés, une trajectoire d'apprentissage est d'abord enregistrée. La trace GPS de cette trajectoire a été superposée à une vue de dessus synthétique de pavin sur la figure 5.5a. Le véhicule est conduit manuellement sur la route faisant le tour de PAVIN, en restant autant que possible dans la file de gauche. Un tour complet du rond-point a également été réalisé. Ceci permet de passer plusieurs fois à une même position, ce qui permet en utilisant un algorithme de bouclage,

(a) Trace GPS de la trajectoire superposée à une vue synthétique de l'environnement



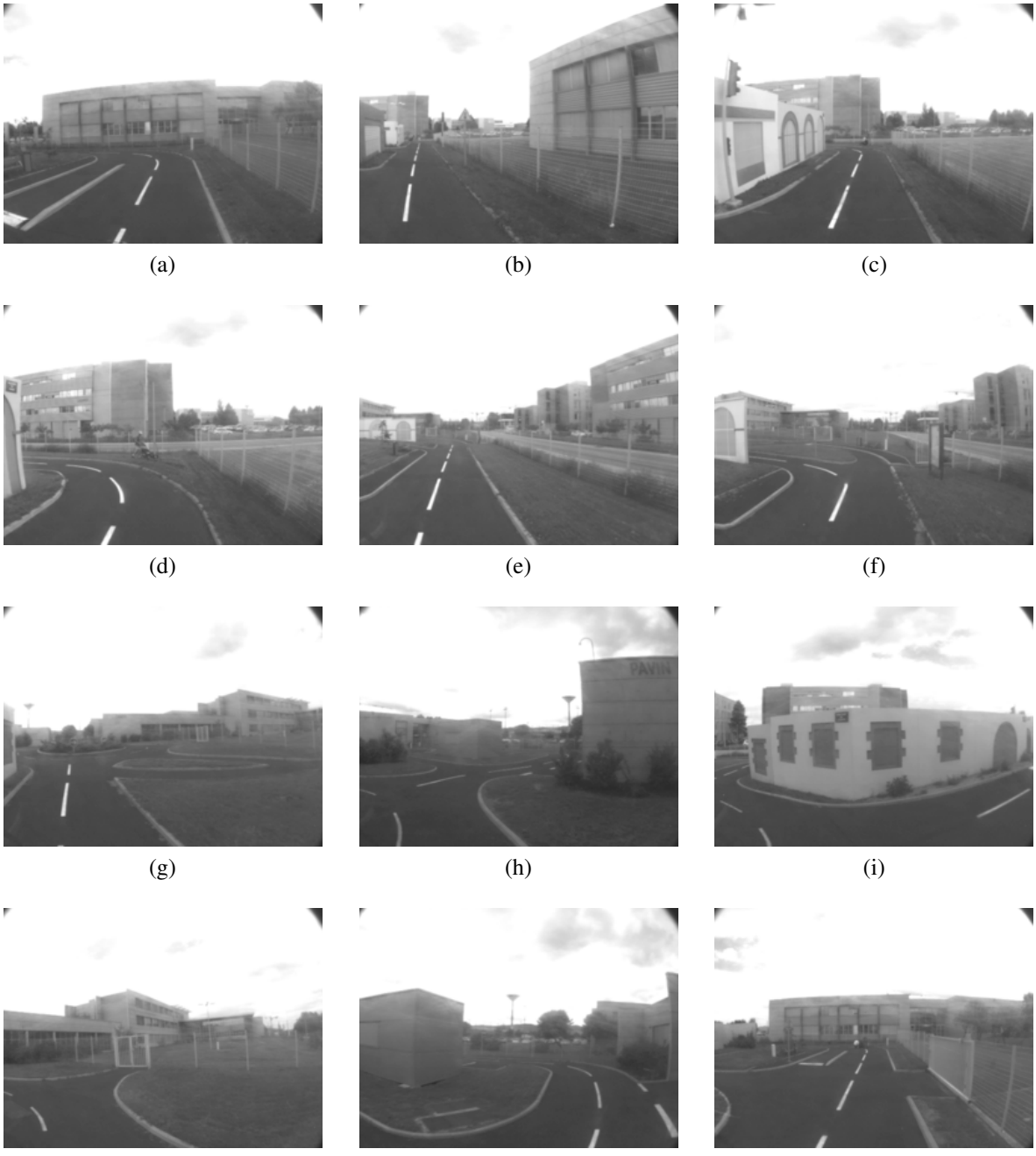
(b) Reconstruction réalisée, comparée avec les données GPS

FIGURE 5.5 – Vue de dessus de la trajectoire d'apprentissage

d'améliorer la reconstruction. De plus, le fait d'avoir un tour complet donne une vue initiale de l'environnement en sens inverse. Le tour de la zone chevauche la zone de départ afin de pouvoir réaliser une fermeture de boucle. Quelques images de cette séquence sont représentées sur la figure 5.6.

La reconstruction est ensuite réalisée en utilisant les deux caméras à bord du véhicule. De plus l'algorithme de fermeture de boucle optimise les paramètres extrinsèques des caméras, c'est-à-dire leur position relative l'une par rapport à l'autre, permettant d'obtenir une reconstruction avec une très faible dérive, comme on peut le constater sur la vue de dessus de la reconstruction en figure 5.5b.

Cette reconstruction est ensuite utilisée pour générer les patchs et se localiser. En particulier cet apprentissage a été utilisé pour les expérimentations de la partie 4.5.1. Après calcul, 23 320 patchs sont utilisables pour la localisation.



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) (k) (l)

FIGURE 5.6 – Images de la séquence d'apprentissage utilisée sur PAVIN

Méthode d'expérimentation

L'objectif de ces premières expérimentations consiste à évaluer la précision et la robustesse de l'algorithme. En particulier le véhicule est placé dans des conditions difficiles où la localisation n'est pas toujours possible. Pour cela, le véhicule est piloté manuellement et la localisation est réalisée à partir des images acquises. La navigation n'est pas autonome dans ce cas.

Le GPS différentiel est également enregistré simultanément avec les images et permet, grâce aux méthodes décrites dans la partie précédente, d'analyser la précision de l'algorithme pour chaque séquence.

Les différentes trajectoires réalisées s'éloignent un peu de la trajectoire d'apprentissage, pour évaluer les performances de l'algorithme lors de changements de point de vue. La figure 5.7 montre les traces GPS de trois trajectoires réalisées :

- ➢ Zigzag où le véhicule zigzague entre les deux voies de circulation ;
- ➢ Décalage où le véhicule prend un chemin parallèle dans le troisième virage ;
- ➢ Inversion où le véhicule fait demi-tour au rond-point et prend la route en sens inverse.

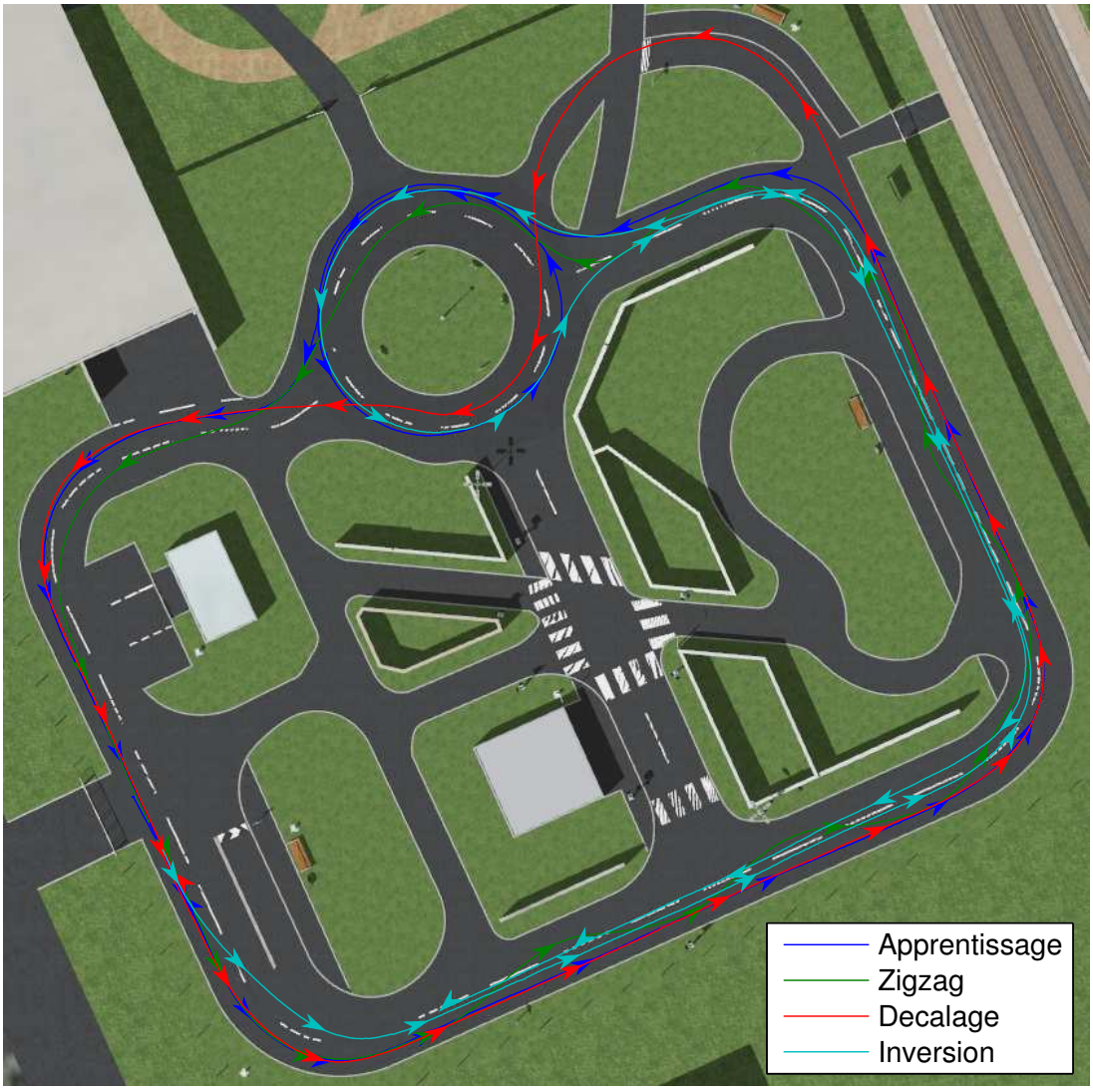
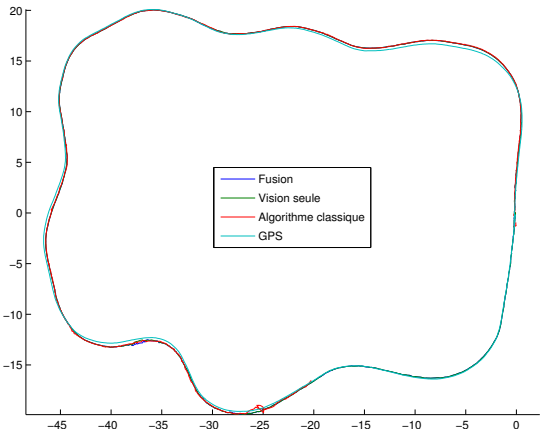
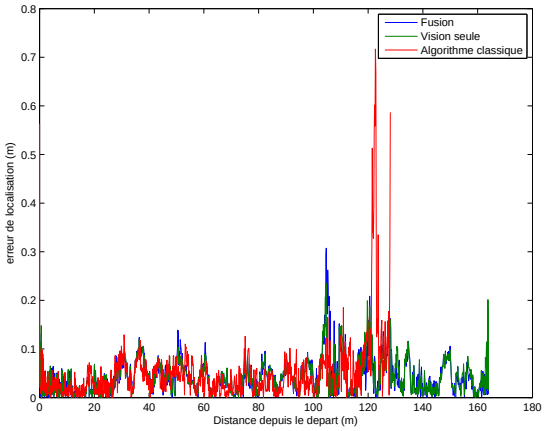


FIGURE 5.7 – Traces GPS de quelques trajectoires réalisées superposées à la plateforme



(a) Vue de dessus de la localisation lors du Zigzag



(b) Erreur de localisation en 3D lors du zigzag

FIGURE 5.8 – Résultats de la localisation pour la trajectoire en Zigzag

5.3.2 Zigzag

Dans le chapitre précédent, lors de l'évaluation de l'intérêt de la fusion, nous avons déjà montré que la localisation pouvait être assez précise malgré un écart latéral d'environ 1,5 m. Pour rendre les choses plus complexes, le zigzag va non seulement avoir régulièrement un écart latéral, mais également provoquer de nombreux changements de cap, le véhicule réalisant successivement des virages à gauche et à droite.

De plus, pour évaluer l'apport de l'algorithme, la localisation est également réalisée avec un algorithme classique de localisation n'utilisant pas de patchs des points sur les images clé. Il s'agit de l'algorithme antérieur à ces travaux décrit en 1.3.2. La reconstruction utilisée est la même que pour nos travaux, seule la localisation est basée sur l'utilisation des images clés d'origine et du repérage des points d'intérêt présents sur les images clefs. Ainsi les conditions pour se localiser sont identiques pour les deux algorithmes.

On obtient les résultats de la figure 5.8 et la table 5.1.

	moyenne(m)	écart-type(m)	médiane (m)	maximum (m)
Fusion	0.043	0.036	0.035	0.312
Vision seule	0.045	0.036	0.037	0.238
Algorithme classique	0.048	0.063	0.034	0.716

TABLE 5.1 – Valeur de l'erreur de localisation pour la trajectoire en zigzag

La précision reste donc assez bonne, environ 4 cm d'erreur en moyenne, et moins de 32 cm dans le pire des cas. Lorsque l'on compare avec l'algorithme classique, on s'aperçoit que la précision reste semblable. L'algorithme classique a juste eu quelques problèmes à la sortie du

rond-point, lié au fait que, voulant suivre la référence, il cherchait à utiliser les images-clef du tour du rond-point. Hormis ce détail à la fin de la séquence, la précision reste du même ordre que celle fourni par l'algorithme avec les patchs. On peut donc en déduire que la précision des deux algorithmes reste la même.

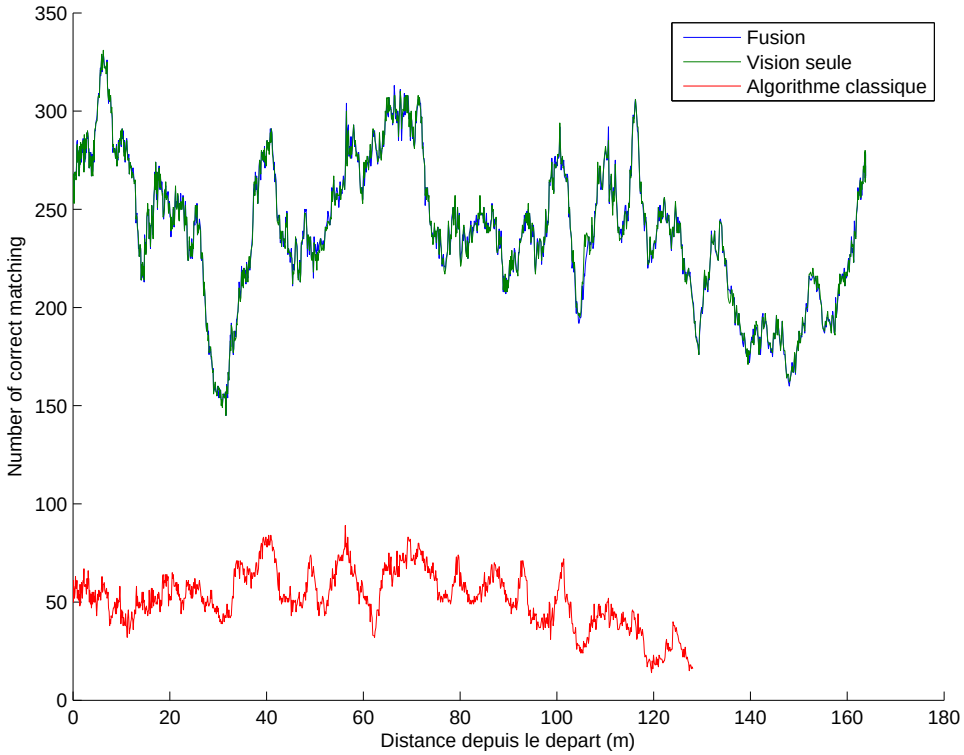


FIGURE 5.9 – Nombre d'inliers sur la trajectoire en zigzag

Pour évaluer la robustesse de l'algorithme, il est également possible de visualiser le nombre de points correctement appariés, nommés inliers. La figure 5.9 montre le nombre d'inliers au fur et à mesure de la trajectoire. Ce nombre reste assez élevé malgré l'écart à la trajectoire. Sur ce point il y a une grande différence avec l'algorithme classique. Cela s'explique en partie parce que, du fait de la reprojection davantage de patchs sont reconnus par rapport au nombre de points sans reprojection. Néanmoins il faut noter que l'algorithme classique fonctionnant en se basant sur les images clef, va limiter la recherche de correspondance aux points de référence qui étaient dans l'image clef la plus proche. Alors que n'utilisant plus les images clef, tout les patchs visibles vont être recherchés, quelle que soit l'image clef dans laquelle ils ont été vus. Ainsi ayant un ensemble de référence recherché beaucoup plus important, il est logique que davantage de points inliers soient retrouvés. Cependant cela permet d'avoir une meilleure robustesse. En effet même si la moitié des inliers était perdue, par exemple à cause d'une occultation, il en

resterait suffisamment pour se localiser avec les patchs, alors que la position serait très peu précise en utilisant l'algorithme classique.

Il est également intéressant de visualiser la projection des patchs au fur et à mesure de la trajectoire. Les figures 5.11 et 5.12 montrent quelques images de la séquence avec la reprojection des patchs. Les images ont été prises aux positions placées sur la vue de dessus de la figure 5.10.

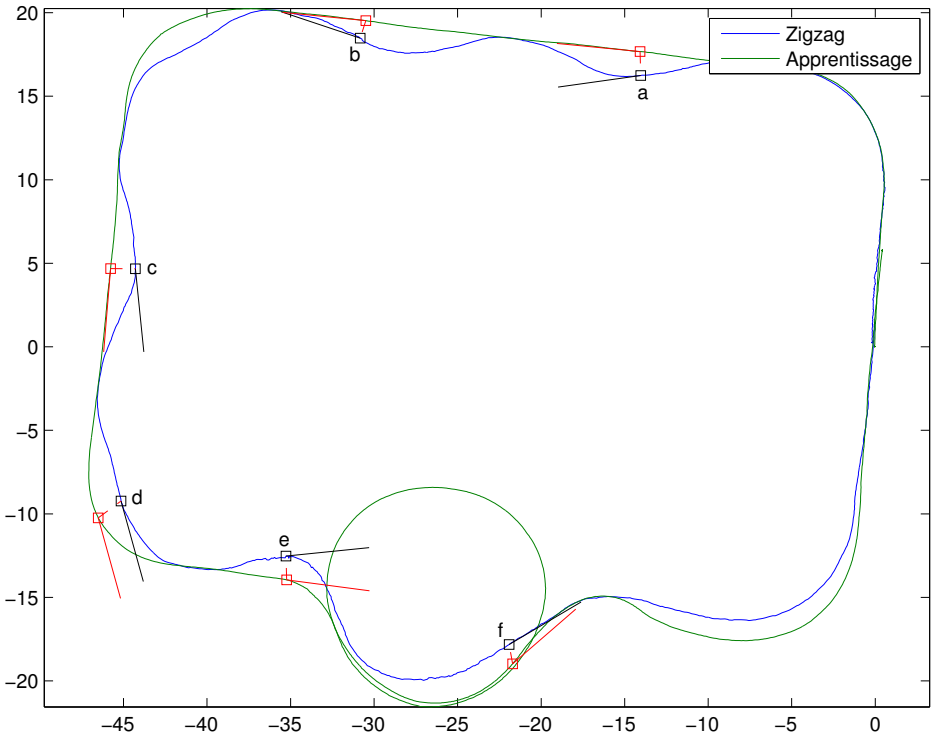
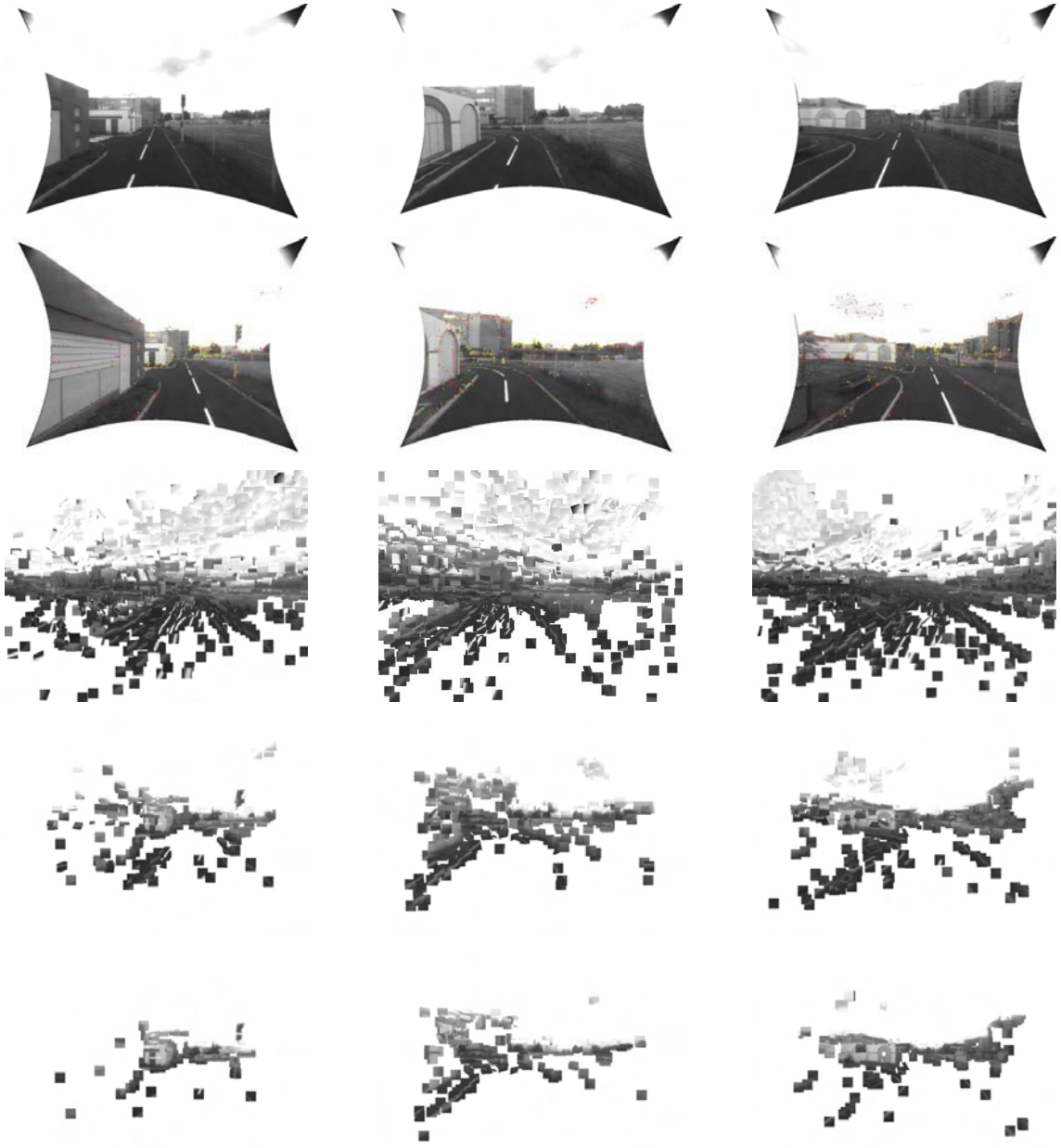


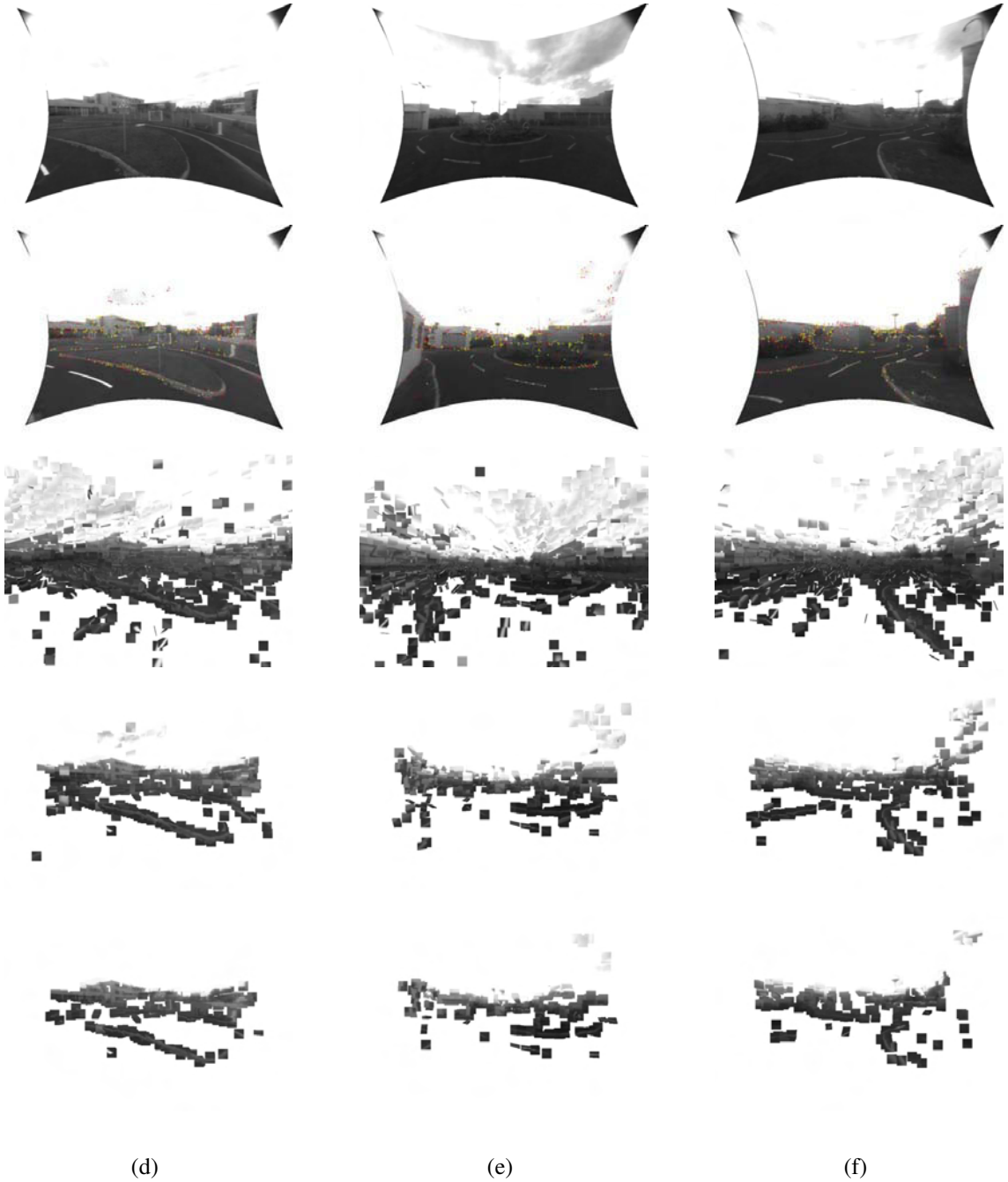
FIGURE 5.10 – Vue de dessus de la trajectoire lors du zigzag. Les positions marquées sont celles où ont été extraites les images des figures 5.11 et 5.12. Pour chaque position, un carré noir représente la position du véhicule, avec un trait pour symboliser la direction de l'axe optique, et un carré rouge est placé sur la position de référence la plus proche (en terme de position et d'angle de vue).

Ces différentes images permettent de se faire une bonne idée de l'évolution de l'algorithme. En particulier avec l'image regroupant tous les patchs a priori visibles, on s'aperçoit que beaucoup d'entre eux sont mal positionnés ou inutilisables. L'appariement préalable permet de filtrer efficacement tous ces patchs, pour obtenir une projection plus proche de la réalité. L'image générée correspond bien à l'image courante, même si l'image de référence était assez différente.



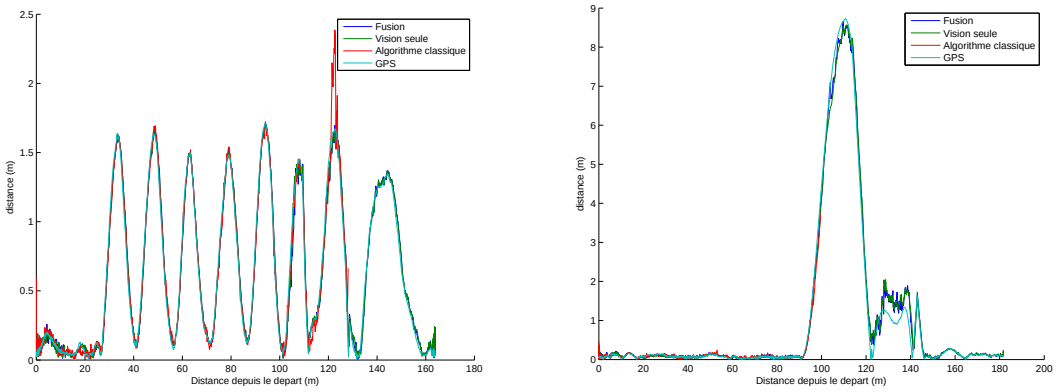
(a) (b) (c)

FIGURE 5.11 – Images extraites de la localisation lors du zigzag. Dans chaque colonne, on a successivement l'image vue, avec les points détectés marqués en jaune si appariés et en rouge sinon. Ensuite la projection de tous les patchs a priori visibles, puis la projection des patchs appariés et pour finir les patchs inliers.



(d) (e) (f)

FIGURE 5.12 – Images extraites de la localisation lors du zigzag. Dans chaque colonne, on a successivement l'image vue, avec les points détectés marqués en jaune si appariés et en rouge sinon. Ensuite la projection de tous les patchs a priori visibles, puis la projection des patchs appariés et pour finir les patchs inliers.



(a) Distance à la référence lors du Zigzag

(b) Distance à la référence lors de la trajectoire avec décalage

FIGURE 5.13 – Distance des trajectoires réalisées par rapport à l'apprentissage. Les mesures sont faites à la fois avec le GPS différentiel, et avec les données de localisation.

5.3.3 Décalage

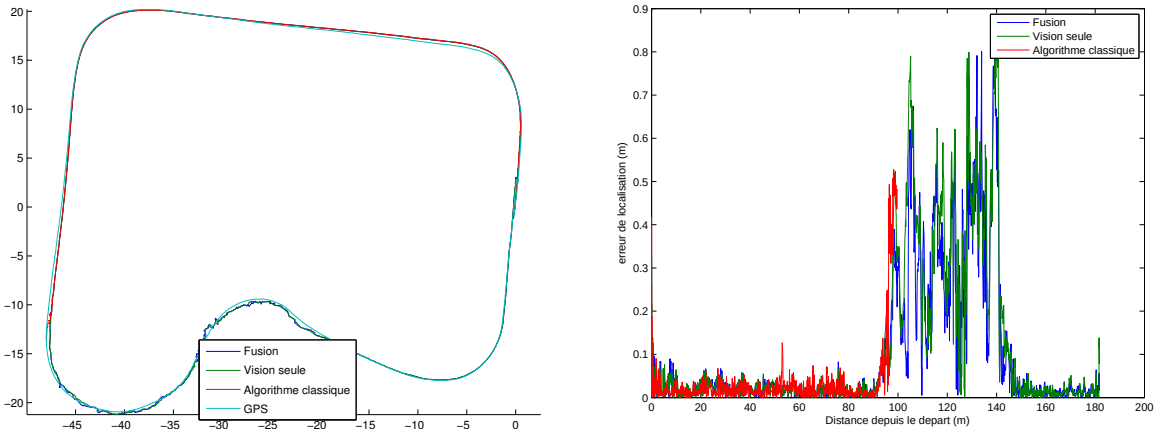
L'expérience précédente montre que le véhicule peut se déplacer et avoir une rotation importante par rapport à l'apprentissage. Cependant lorsque seule une rotation est effectuée, les points ne sont certes plus à la même position sur l'image, mais (mise à part la déformation de l'objectif), leur apparence n'a pas changé. En effet, pour avoir réellement un changement de point de vue il faut que le véhicule soit à une certaine distance de la position initiale. Lors du zigzag, il s'éloigne en passant d'une voie de circulation à une autre, soit une distance maximale de 1,75 m dans l'extremum d'un zigzag.

Pour visualiser plus précisément l'effet de la trajectoire, cette expérimentation, nommé décalage, prend un chemin différent pour s'éloigner à plus de 8,5 m de la référence. A titre de comparaison, la figure 5.13a montre l'évolution de la distance à la référence lors du zigzag, et la figure 5.13b lors de cette expérience. De plus, après s'être décalé dans le chemin, le rond-point est pris par la gauche, et le tour est donc fait par l'autre coté par rapport à la référence. La trace GPS de cette expérimentation est symbolisée en rouge sur la figure 5.7.

Les résultats de cette expérimentation sont montrés sur la figure 5.14 et la table 5.2.

	moyenne(m)	écart-type(m)	médiane (m)	maximum (m)
Fusion	0.108	0.163	0.026	0.836
Vision seule	0.126	0.188	0.026	0.820
Algorithme classique	0.040	0.086	0.017	0.531

TABLE 5.2 – Valeur de l'erreur de localisation pour la trajectoire avec décalage



(a) Vue de dessus de la localisation

(b) Erreur de localisation en 3D

FIGURE 5.14 – Résultats de la localisation pour la trajectoire avec décalage

On peut voir que dans ce cas, la localisation est plus bruitée et plus difficile, particulièrement dans les parties bien éloignées de la trajectoire. Au niveau de l'algorithme classique, la localisation n'a plus été possible lorsque le véhicule réalisait le décalage. Ainsi les mesures de l'erreur affichées dans le tableau 5.2 ne considèrent que la partie ayant une localisation, et sont donc assez bons. Si on découpe la trajectoire en deux parties, avant et pendant le décalage, on obtient les résultats des tables 5.3 et 5.4. On s'aperçoit encore que là aussi la précision est très semblable sur les parties où l'algorithme classique a pu se localiser. Néanmoins le fait de ne pas pouvoir fournir de localisation lors de la sortie de la route montre que l'algorithme classique est moins robuste que celui utilisant les patchs dans les cas d'éloignement de la trajectoire. En effet dans ce cas-là le nombre d'inliers présents chute, comme le montre la figure 5.15, et si les patchs parviennent toujours à fournir une localisation (certes moins précise), l'algorithme classique lui n'a plus assez d'appariements pour calculer une position.

	moyenne(m)	écart-type(m)	médiane (m)	maximum (m)
Fusion	0.024	0.028	0.018	0.327
Vision seule	0.027	0.042	0.017	0.315
Algorithme classique	0.023	0.032	0.015	0.413

TABLE 5.3 – Valeur de l'erreur de localisation sur la partie précédent le décalage

Comme précédemment, il est intéressant de voir la reprojection des amers sur certaines positions pour visualiser les informations utilisées par la localisation. Les figures 5.17 et 5.18 représentent les images des patchs projetés aux positions représentées sur la figure 5.16.

Bien que sur la figure une image de référence ayant un angle de vue proche soit placé, il convient de se rappeler que l'algorithme n'utilise pas les données d'une seule image pour se

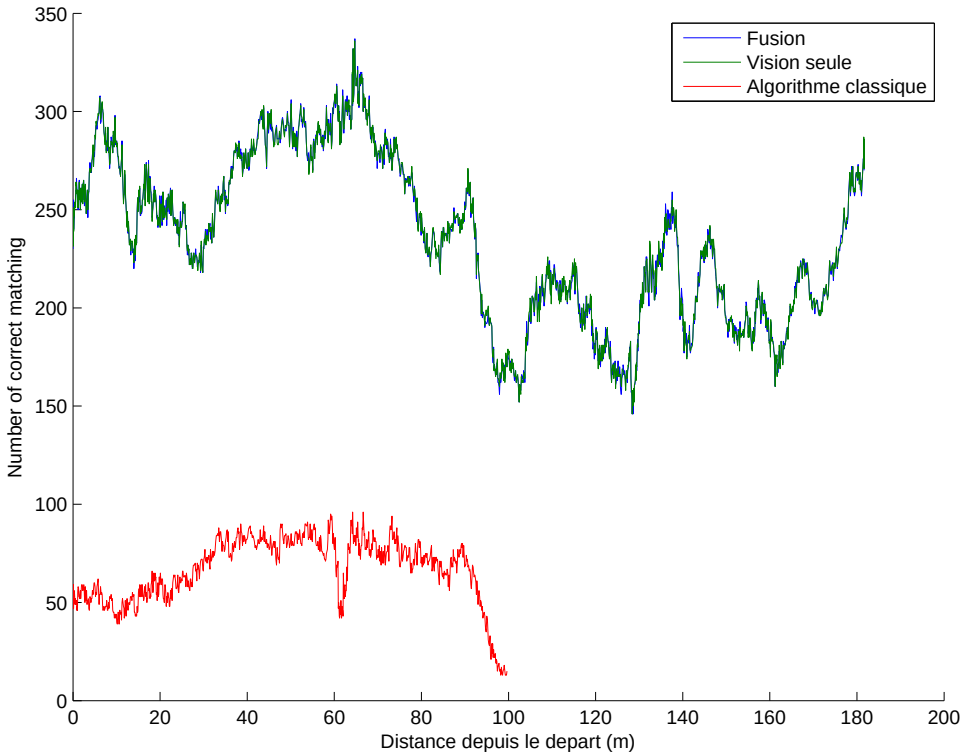


FIGURE 5.15 – Nombre d'inliers sur la trajectoire avec décalage

	moyenne(m)	écart-type(m)	médiane (m)	maximum (m)
Fusion	0.292	0.179	0.278	0.836
Vision seule	0.344	0.196	0.348	0.820
Algorithme classique	0.263	0.192	0.251	0.531

TABLE 5.4 – Valeur de l'erreur de localisation sur la partie avec un fort décalage

localiser. En effet l'ensemble des patchs (a priori) visibles sont projetés pour la localisation. Ainsi, même lorsque l'on est passé loin de la référence, des patchs venant de plusieurs positions de l'apprentissage peuvent être utilisés pour se localiser.

On peut voir que outre l'éloignement à la trajectoire d'apprentissage, la présence d'obstacles tels que le rond-point peut occulter certains amers, rendant la localisation plus difficile et expliquant que la localisation soit moins précise.

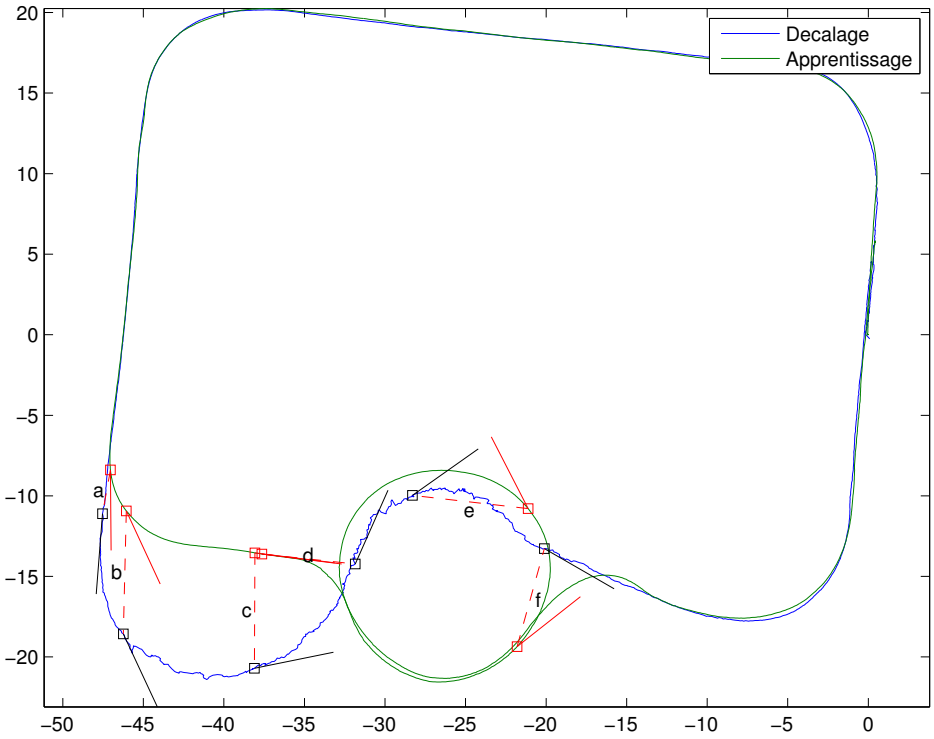
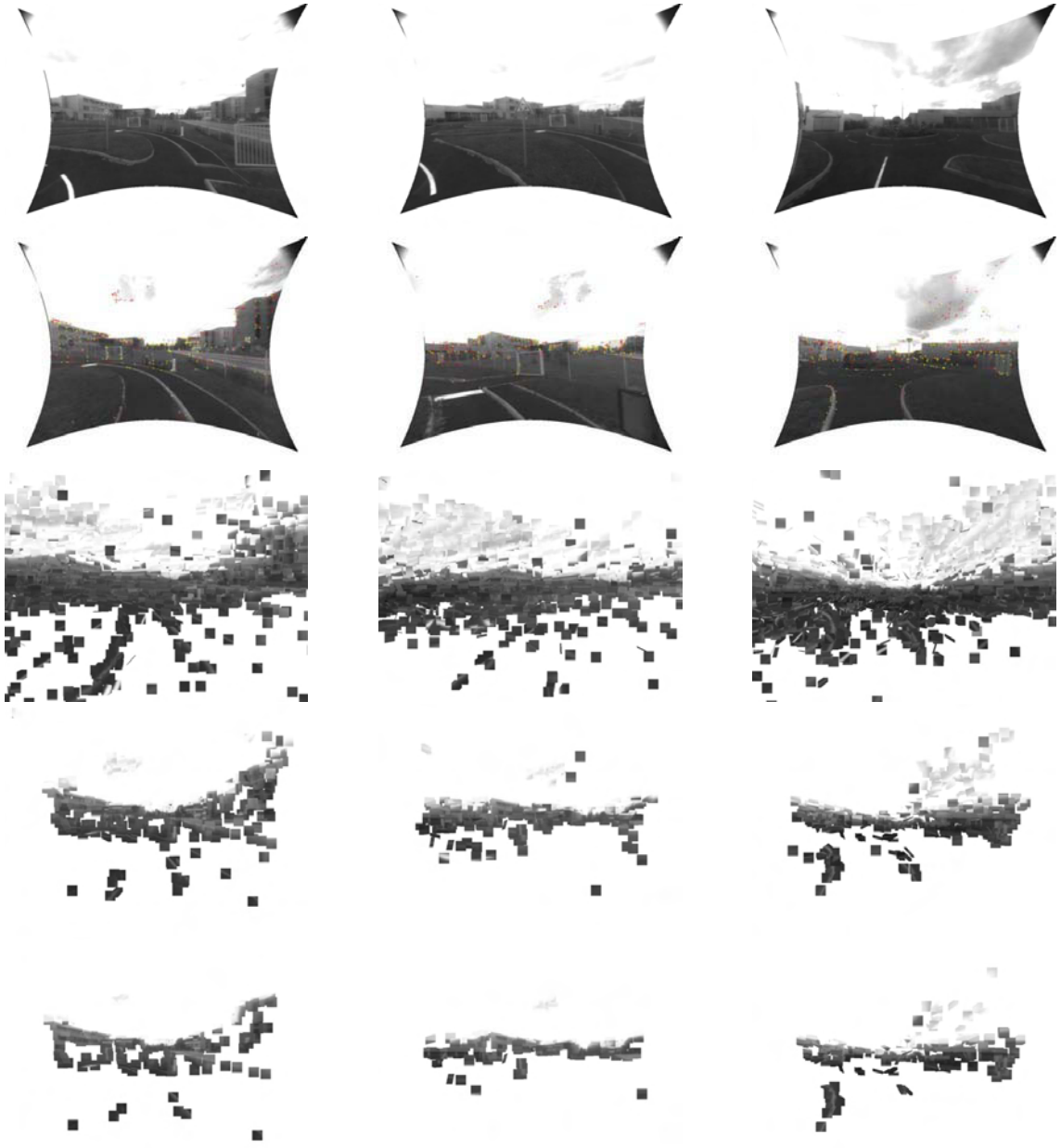
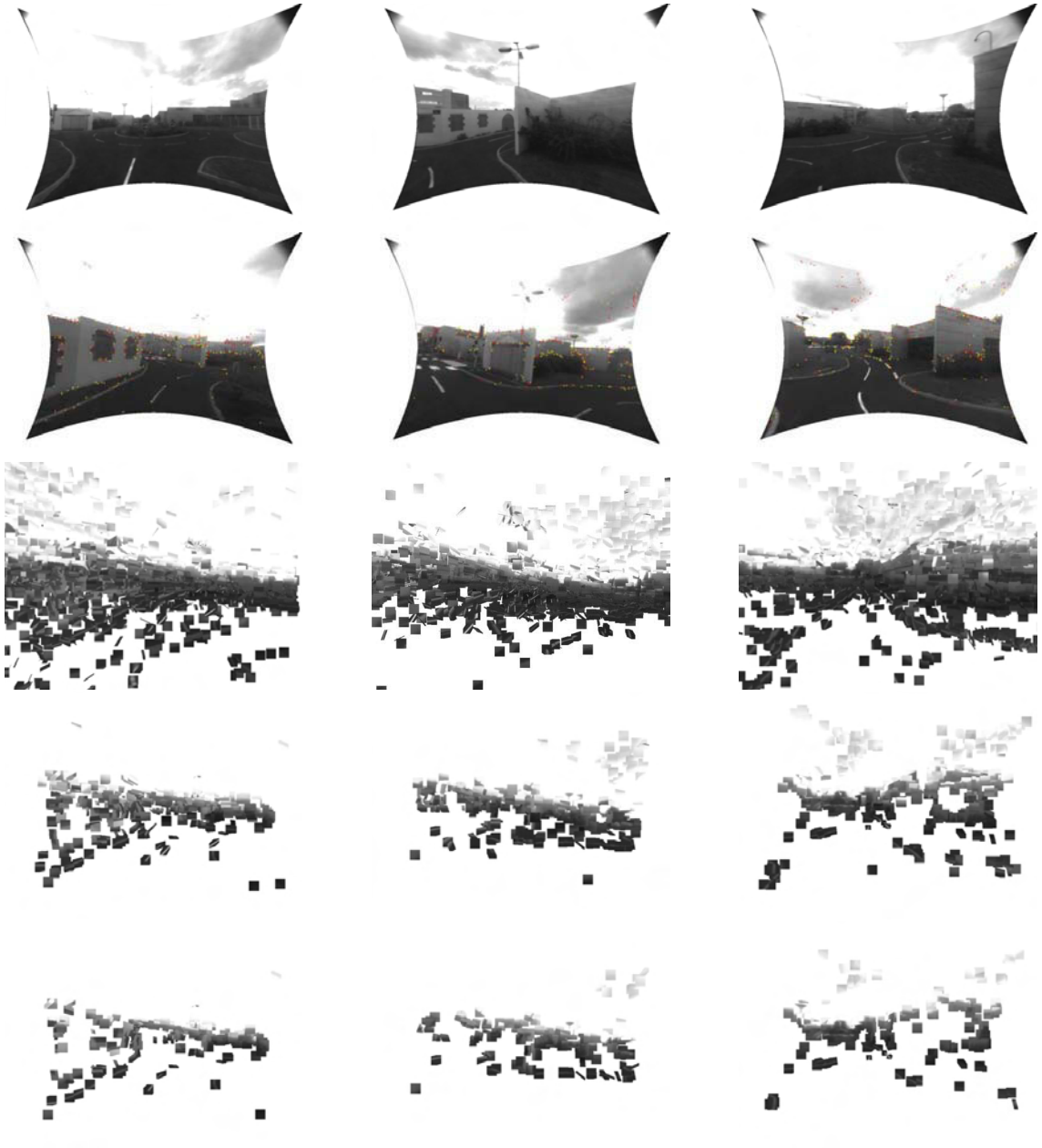


FIGURE 5.16 – Vue de dessus de la trajectoire lors du décalage. Les positions marquées sont celles où ont été extraites les images des figures 5.17 et 5.18. Pour chaque position, un carré noir représente la position du véhicule, avec un trait pour symboliser la direction de l'axe optique, et un carré rouge est placé sur la position de référence la plus proche (en terme de position et d'angle de vue). Le trait en pointillé permet de montrer la correspondance entre la position courante du véhicule et l'image référence la plus semblable.



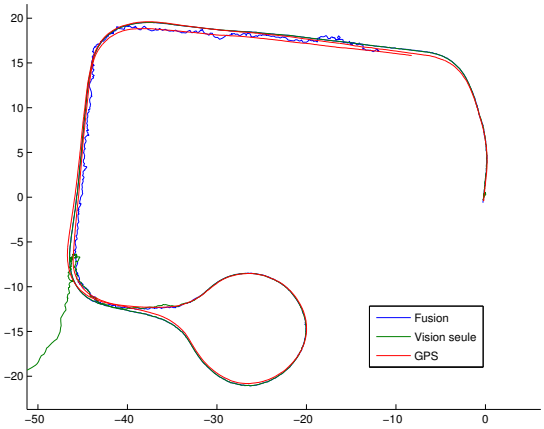
(a) (b) (c)

FIGURE 5.17 – Images extraites de la localisation lors du décalage. Dans chaque colonne, on a successivement l'image de référence, l'image vue (avec les points détectés marqués en jaune si appariés et en rouge sinon), la projection de tous les patchs a priori visibles, la projection des patchs appariés et pour finir les patchs inliers.

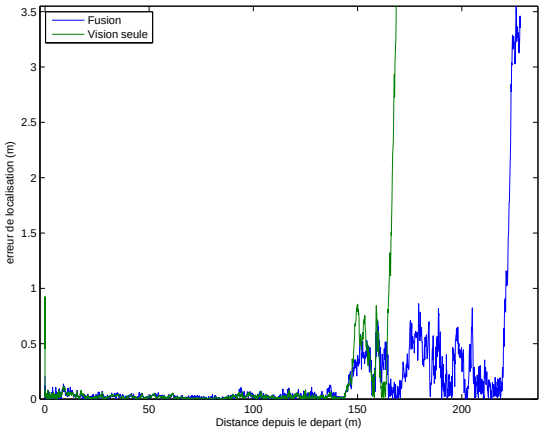


(d) (e) (f)

FIGURE 5.18 – Images extraites de la localisation lors du décalage. Dans chaque colonne, on a successivement l'image de référence, l'image vue (avec les points détectés marqués en jaune si appariés et en rouge sinon), la projection de tous les patchs a priori visibles, la projection des patchs appariés et pour finir les patchs inliers.



(a) Vue de dessus de la localisation



(b) Erreur de localisation en 3D. Par soucis de visibilité, l'échelle verticale a été tronquée bien que l'erreur de vision seule monte bien plus haut.

FIGURE 5.19 – Résultats de la localisation pour la trajectoire avec inversion

5.3.4 Inversion

Pour finir, afin de visualiser la capacité de reprojection de l'algorithme, une séquence a été réalisée en faisant faire un demi-tour au véhicule. Lors de l'apprentissage, un tour complet du rond-point a été réalisé. Cette trajectoire reprend le tour du rond-point mais au lieu de faire un tour complet pour continuer sur la route, le véhicule repart en sens inverse. Ainsi dans le rond-point, plusieurs amers ont pu être vus dans les deux sens, mais à partir de la route en sens inverse la localisation n'est possible qu'en inversant certains patchs. Au niveau de l'algorithme classique, aucune adaptation n'étant faite des points de référence, il est impossible de se localiser lorsque le véhicule roule en sens inverse. Par conséquent les résultats concernant cette expérience présenté sur la figure 5.19 et la table 5.5 n'affiche pas l'algorithme classique.

	moyenne(m)	écart-type(m)	médiane (m)	maximum (m)
Fusion	0.178	0.464	0.036	3.497
Vision seule	17.603	42.715	0.034	226.279

TABLE 5.5 – Valeur de l'erreur de localisation pour la trajectoire avec inversion

Tout d'abord on s'aperçoit que la localisation avec la vision seule ne parvient pas à trouver une position sur la route après le dernier virage. Lorsque l'odométrie est utilisée, et donc que les prédictions sont tout de même meilleures, la localisation est possible mais devient moins précise. On peut effectivement voir sur la courbe 5.19b que l'erreur moyenne de localisation, tant que l'on est sur la même trajectoire (sur les 130 premiers mètres) est assez faible et aug-

mente considérablement sur la suite de la trajectoire, lorsque l'on circule en sens inverse de l'apprentissage.

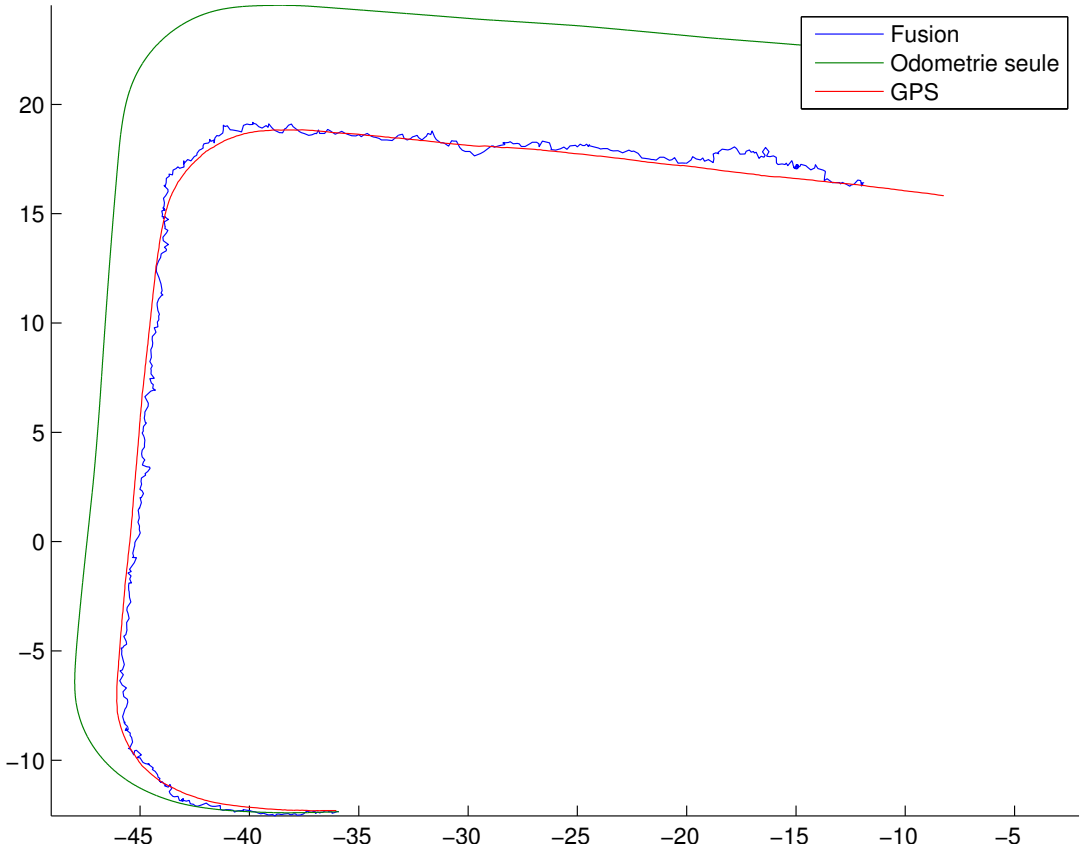


FIGURE 5.20 – Vue de dessus de la localisation lorsque le véhicule circule en sens inverse

Pour se convaincre que ce n'est pas uniquement l'odométrie qui fournit la localisation, la figure 5.20 montre une vue de dessus de la localisation avec fusion, comparée à la position que l'on aurait avec seulement l'odométrie sur la partie où le véhicule se déplace en sens inverse. Comme l'odométrie ne fournit une localisation que relativement à la pose précédente, on l'initialise à la même position que celle de la vision. On peut voir que la dérive est telle que la localisation reste très imprécise et décalée par rapport à la trajectoire de référence, contrairement à l'utilisation de l'algorithme de fusion, où la vision a corrigé la position. En pratique, les deux systèmes se complètent, l'odométrie étant assez précise pour mesurer la distance parcourue et donc l'avancement du véhicule, et la vision étant particulièrement fiable pour évaluer l'orientation (ou angle de cap) du véhicule. Les deux systèmes se complètent donc bien et permettent de réaliser la localisation malgré les conditions difficiles dues à l'inversion.

Pour finir on peut visualiser les patchs projetés pour différentes vues de cette inversion. La figure 5.21 montre les positions où ont été prises les images, et le figure 5.22 et 5.23 les images correspondantes. Les vues de référence ne sont pas affichées dans ce cas car la trajectoire étant inversée, aucune ne correspond vraiment à ce qui est affiché sur l'image courante.

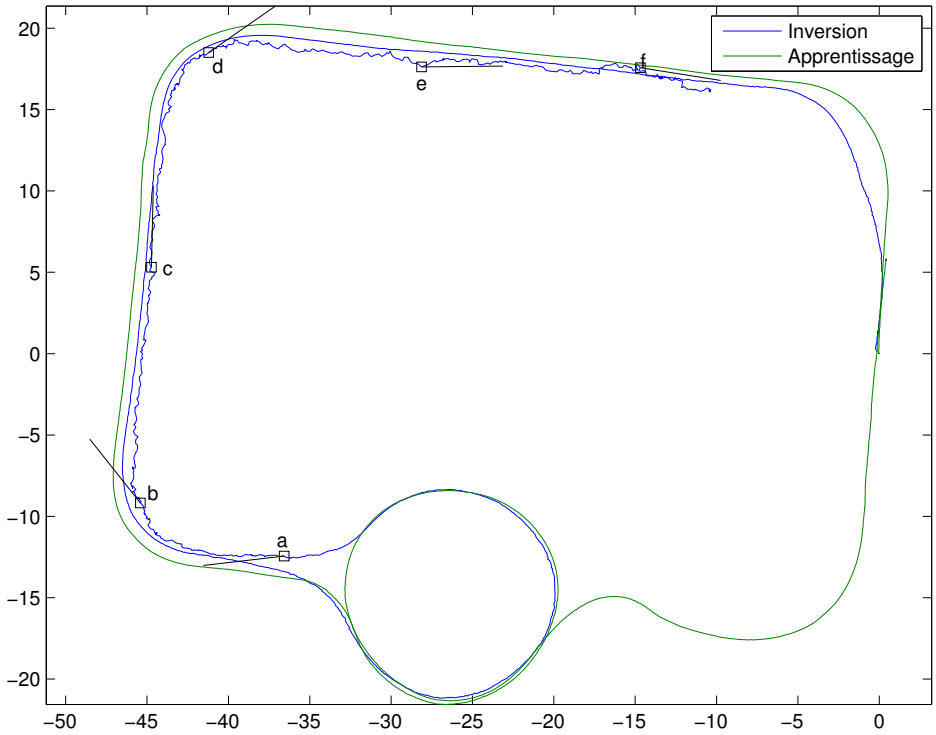
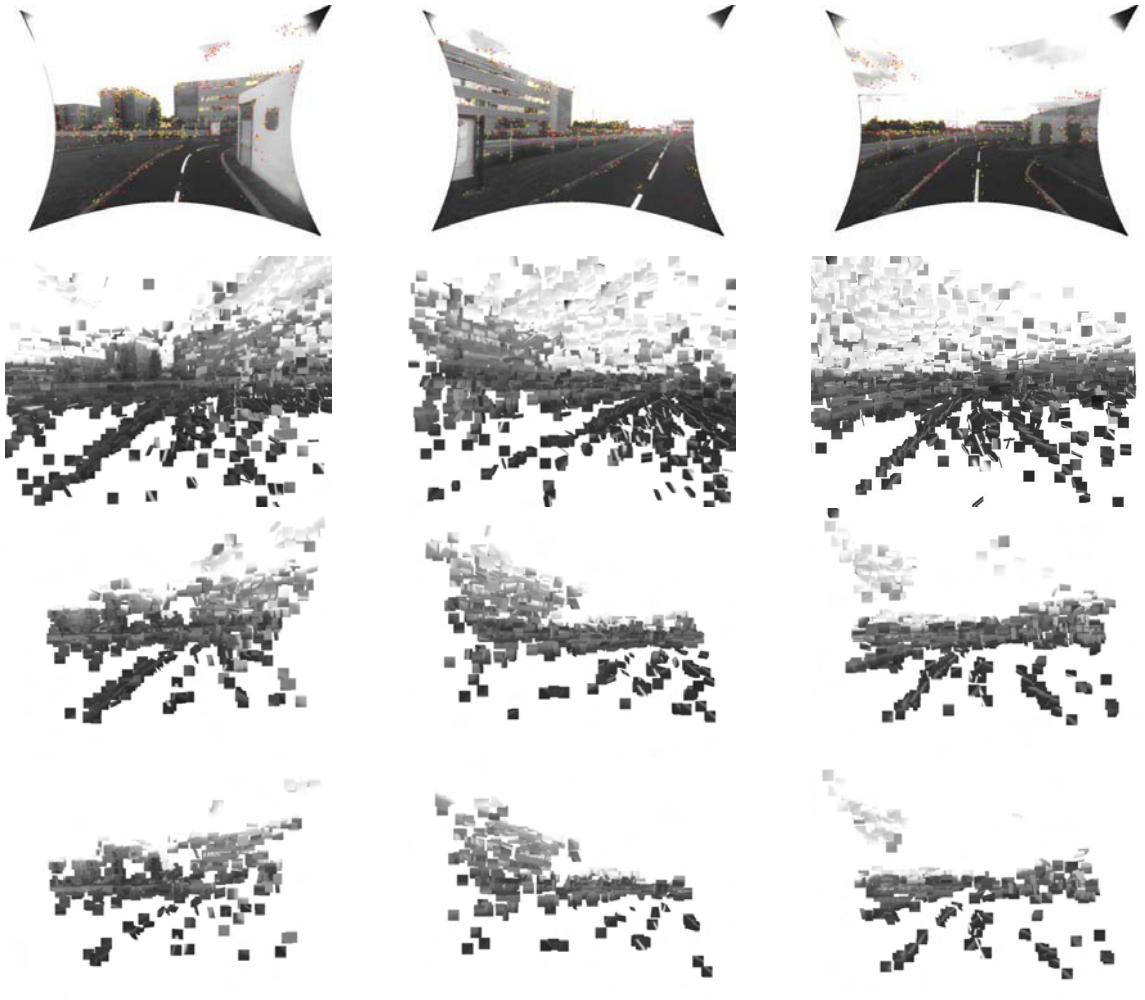
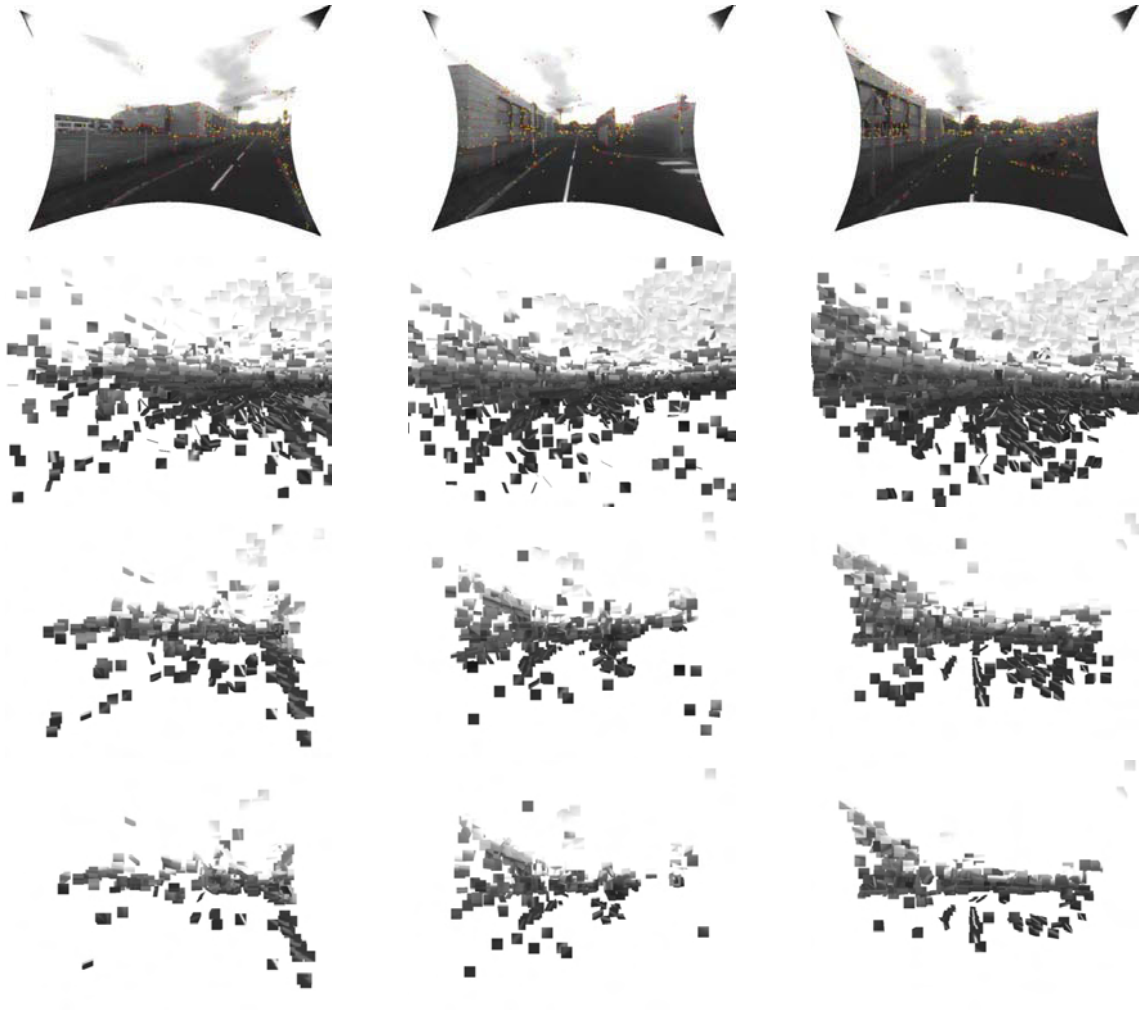


FIGURE 5.21 – Vue de dessus de la trajectoire avec inversion. Les positions marquées sont celles où ont été extraites les images de la figure 5.22. Pour chaque position, un carré noir représente la position du véhicule, avec un trait pour symboliser la direction de l'axe optique



(a) (b) (c)

FIGURE 5.22 – Images extraites de la localisation lors de l'inversion. Dans chaque colonne, on a successivement l'image vue (avec les points détectés marqués en jaune si appariés et en rouge sinon), la projection de tous les patchs a priori visibles, la projection des patchs appariés et pour finir les patchs inliers.



(d) (e) (f)

FIGURE 5.23 – Images extraites de la localisation lors de l'inversion. Dans chaque colonne, on a successivement l'image vue (avec les points détectés marqués en jaune si appariés et en rouge sinon), la projection de tous les patchs a priori visibles, la projection des patchs appariés et pour finir les patchs inliers.

5.4 Démonstration finale

Pour conclure les résultats obtenus, une démonstration finale a été réalisée dans le cadre du projet ANR Cityvip. Cette démonstration, réalisée dans le centre-ville de Clermont-Ferrand, a consisté à faire fonctionner en milieu réel le véhicule de manière autonome.

5.4.1 Zone d'évolution

Lors de cette démonstration le véhicule évolue sur la place de Jaude, une place publique fréquentée par de nombreux piétons et entourée par de grands bâtiments. La figure 5.24 montre des photos de la zone.

Le fait d'être sur une place publique comme celle-ci a permis de constater la robustesse de l'algorithme face à plusieurs contraintes.

Tout d'abord la présence de piétons se déplaçant suivant des mouvements aléatoires nécessite une bonne résistance aux occultations. En effet, en passant devant le véhicule ils masquent certaines parties de l'image, et donc certains amers qui auraient pu être aperçus.

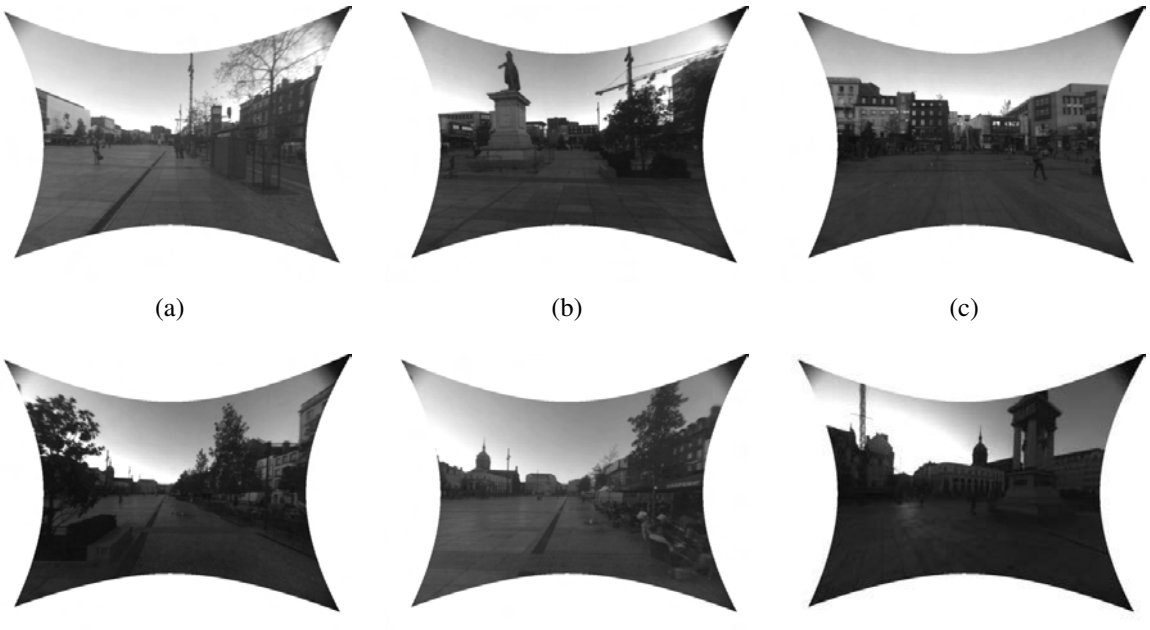
D'autre part, être sur une grande place entourée par des immeubles provoque de fortes contraintes en luminosité, et des changements sensibles entre le matin et l'après-midi. Le soleil changeant de direction au cours de la journée, les façades ensoleillées ne sont pas les mêmes à toute heure et les ombres projetées varient également.

5.4.2 Conduite autonome

L'objectif de la démonstration, outre de montrer l'efficacité de localisation de l'algorithme, était de faire fonctionner le véhicule de manière autonome. Pour cela, il a été nécessaire d'utiliser une loi de commande définie par (Lenain et al., 2005). Cette loi de commande a été utilisée comme une boite noire, à laquelle on fournit la position courante, l'écart latéral à la trajectoire et la courbure de la trajectoire. A partir de ces informations la loi de commande détermine l'angle de braquage des roues à envoyer au système de manière à minimiser la distance à la trajectoire à suivre. On définit également deux vitesses, une appliquée lorsque le véhicule est en ligne droite, et une autre plus réduite lorsque les roues sont braquées d'un angle important.

L'algorithme est également conçu pour permettre au véhicule de s'éloigner de la trajectoire d'apprentissage. Pour le faire de manière autonome, il est alors nécessaire de définir une trajectoire à suivre qui sera différente de celle de l'apprentissage. Cette trajectoire doit être définie dans le même repère que la trajectoire d'apprentissage. Pour cela, une petite application a été réalisée, montrant d'une part la trajectoire d'apprentissage et la position des points d'intérêt et permettant à l'utilisateur de cliquer sur des points par lesquels la trajectoire de commande passera. Les points sont ensuite reliés par une courbe hermitienne, de manière à obtenir une courbe continue qui peut être suivie par le véhicule.

FIGURE 5.24 – Photos de la place de Jaude où les démonstrations finales du projet ANR CityVIP ont été réalisées



(a) (b) (c)

(d) (e) (f)

FIGURE 5.25 – Image de la séquence d'apprentissage de la place Jaude. On peut voir en particulier sur l'image (b) la statue derrière laquelle a été fait l'apprentissage.

Dans le cadre de notre expérimentation, la trajectoire d'apprentissage a été enregistrée en réalisant le tour complet de la place, en passant en particulier derrière la statue à son extrémité. La figure 5.25 montre des images extraites de l'apprentissage.

Pour la trajectoire de commande, afin d'illustrer l'éloignement que peut avoir le véhicule, la trajectoire coupe le tour de la place et passe devant la statue, fait quelques zigzags sur le coté avant de revenir à la position initiale. Ces deux trajectoires ont été positionnées sur une vue synthétique de la place sur la figure 5.26.

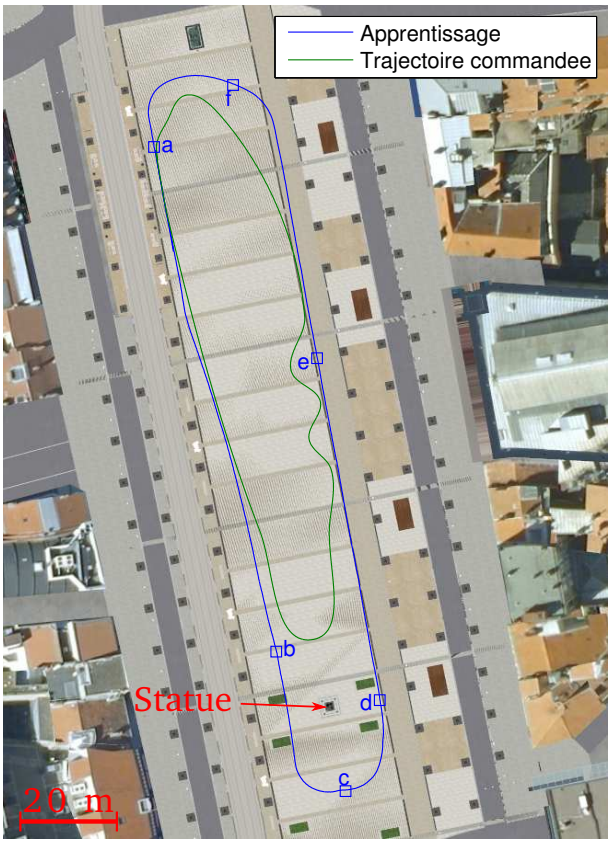


FIGURE 5.26 – Vue de dessus des trajectoires suivies sur la place de Jaude. Les positions représentées sont celles où ont été prises les images de la figure 5.25.

5.4.3 Résultats

Dans un premier temps, en utilisant une séquence acquise en pleine journée, la localisation a échoué. Cela était dû principalement aux fortes contraintes en terme d'illumination, le soleil rendant les façades très brillantes d'un coté alors que celles dans l'ombre apparaissent très sombres. Il a donc été nécessaire de prendre une référence en soirée, lorsqu'aucune ombre n'était présente pour parvenir ensuite à se localiser. Par la suite l'utilisation de l'algorithme

de localisation même durant une journée ensoleillée est possible, mais la référence doit être réalisée dans de bonnes conditions d'illumination.

En utilisant la référence acquise dans de bonnes conditions de luminosité, la localisation a été possible. Des premières expérimentations où le véhicule était piloté à la main, ont montré qu'il était possible de s'éloigner de l'apprentissage jusqu'au coté opposé de la place, (soit à plus de 20 m de l'apprentissage) sans perdre la localisation. On pouvait constater, dans les cas où l'on s'éloigne beaucoup de la trajectoire que l'orientation du véhicule était calculée par la vision alors que l'avancement était fourni par l'odométrie.

Par la suite, le véhicule a été utilisé de manière autonome sur une période continue d'une demi-journée, sans être interrompu involontairement. Le véhicule a parfaitement suivi la trajectoire définie malgré le changement de position, en particulier dans le virage réalisé devant la statue contrairement à l'apprentissage. Malheureusement, il n'a pas été possible, lors de la localisation d'enregistrer les informations du GPS, et par conséquent on ne peut pas évaluer quantitativement les distances mesurées par la vision. La même trajectoire étant répétée plusieurs fois, on a pu constater que les mêmes positions étaient reprises entre les différents tours.

La commande autonome fonctionne également bien, malgré les passages de piétons et autres changements d'environnement. Le véhicule pouvait se déplacer à environ 2 m/s dans les lignes droites. Lorsque des obstacles se présentaient devant le véhicule, le télémètre LASER forçait la commande à s'arrêter. Puis lorsque le chemin était totalement libre, le véhicule repartait sans avoir perdu sa localisation, montrant que le système de prédiction pouvait s'adapter à ce type d'arrêt brutal.

Pour finir, cette démonstration a permis de réaliser une fusion des données de localisation au sein d'un processus fourni par les collaborateurs du projet (laboratoire XLIM) et décrit dans (Bétaille et al., 2012). La pose calculée par vision était envoyée à leur algorithme qui, par la suite affichait une vue virtuelle du véhicule et évaluait l'incertitude de la position en tenant compte de tous les capteurs disponibles.

5.5 Difficultés rencontrées

Pour compléter cette partie sur les résultats de l'algorithme, il est nécessaire de dresser le bilan de toutes les difficultés rencontrées au cours des différents essais.

5.5.1 Luminosité

La première difficulté, principalement rencontrée lors de la démonstration finale est liée à la luminosité de la scène. En particulier lorsque le soleil est assez bas, ou en présence de vitres très réfléchissantes, il peut y avoir un fort contraste entre les éléments de la scène éclairés directement par le soleil et ceux situés dans l'ombre. Ce fort contraste rend la reconstruction peu précise, et les patchs calculés sont de mauvaise qualité et difficiles à reconnaitre si la localisation se fait avec une luminosité différente. Pour éviter ce problème il est nécessaire d'enregistrer la

trajectoire d'apprentissage à un moment de la journée où il y a peu d'ombre, ou par temps couvert. Lorsque la référence a été enregistrée dans de bonnes conditions, la localisation peut se faire même si la luminosité est moins bonne.

Un autre problème lié à l'éclairement se rencontre lorsque le véhicule fait face au soleil. La caméra se trouve alors éblouie et plus aucun amer ne peut être retrouvé. Lorsque ce phénomène se produit temporairement, par exemple dans un virage, la navigation par odométrie permet de continuer sur la trajectoire, mais si le véhicule doit rester face au soleil l'algorithme de localisation finit par échouer complètement. Une solution consiste à utiliser la seconde caméra située à l'arrière du véhicule, en supposant que les caméras ne seront jamais éblouies simultanément.

5.5.2 Erreur de localisation

Un autre problème rencontré principalement lors des expériences sur PAVIN se retrouve lorsque pour une raison quelconque, la prédiction donne une pose erronée. De part la nature de l'algorithme, de nombreux patchs sont recherchés dans l'image et, malgré le facteur de qualité évalué lors de leur construction, plusieurs d'entre eux sont erronés et ressemblent à une surface floue qui obtient un score d'appariement correct avec n'importe quelle image.

Tant que la prédiction n'est pas trop éloignée de la réalité, ces "mauvais patchs" sont suffisamment peu nombreux par rapport aux inliers, le calcul de pose n'est pas perturbé et les rejette en tant que mauvais appariement.

Cependant la situation est différente lorsque la prédiction est imprécise et que la projection n'est donc pas réalisé dans la vue correspondant à la pose courante. Dans ce cas, peu de patchs parviennent à s'apparier avec les points correspondant de l'image, et les "mauvais patchs" eux, s'apparient avec des points qui n'ont rien à voir avec leur réelle position 3D. Le calcul de pose se base alors sur ces appariements, devenus majoritaires et trouve une position totalement fausse en les considérant comme des inliers. Ainsi la pose obtenue est non seulement erronée, mais du fait de la présence de ces patchs elle peut avoir une incertitude réduite et être située très loin de la pose réelle. Comme un certain nombre de patchs apparaissent comme inliers, la situation n'est pas détectée et le modèle d'évolution mis à jour avec cette pose va continuer à donner des mauvaises prédictions. Ainsi, dans les cas où la localisation échoue, comme lors de l'inversion avec le modèle de prédiction sans odométrie, le véhicule est localisé loin de sa trajectoire et l'algorithme continue à le croire de plus en plus loin sans pouvoir détecter le problème.

Une solution à ce problème serait de réduire le nombre de patchs recherchés dans l'image, soit en se basant sur les inliers précédents, ou en sélectionnant a priori leur qualité de manière plus fine. Ainsi les "mauvais patchs" seraient éliminés ou au moins suffisamment peu nombreux pour ne pas influer sur le calcul de pose.

5.5.3 Charge de calcul

Pour finir une dernière difficulté à garder à l'esprit est liée à la charge de calcul de l'algorithme. Les performances temps-réel ont pu être atteintes grâce à la programmation sur GPU.

Cependant, cela implique de disposer d'un ordinateur possédant une carte graphique assez performante pour exécuter l'algorithme. De plus, l'algorithme tente d'apparier l'ensemble des patchs sur chaque image et dans ce but, l'ensemble des patchs reconstruit doit être stocké en mémoire ce qui, là encore représente une charge supplémentaire sur la machine.

Cette charge de calcul est d'autant plus importante que le nombre total de patchs dans la zone est grand. Ainsi si le véhicule doit se déplacer dans une très grande zone, par exemple un quartier complet, de très nombreux patchs seront calculés, et même la machine utilisée dans nos expérimentations risque de ne plus avoir assez de mémoire et de puissance de calcul. Il serait alors nécessaire de découper la zone en plusieurs parties associées chacune à un groupe restreint de patchs. Ainsi seuls les groupes correspondants à la zone où se situe le véhicule seraient chargés en mémoire.

La charge de calcul est également importante lors de la reconstruction et du calcul des patchs. En effet, pour une zone telle que la place de Jaude il a fallu une dizaine d'heures pour calculer l'ensemble des patchs d'apprentissage. Cette partie étant réalisée hors ligne, aucun effort particulier n'a été réalisé pour diminuer ce temps. Néanmoins utiliser les techniques décrites pour l'optimiser et éventuellement le passer sur GPU pourrait permettre de diminuer ce temps et faciliter ainsi les expérimentations futures.

Conclusion et Perspectives

Principales contributions

Le travail présenté dans cette thèse a permis l'amélioration d'un système de localisation d'un robot par vision, utilisable dans le cadre de navigation autonome. Comme dans le système précédent, le robot doit d'abord être conduit dans la zone considérée de manière à obtenir une cartographie 3D de la scène. Ensuite l'algorithme de localisation est utilisé comme un *capteur intelligent* fournissant une pose du véhicule et l'incertitude associée à partir de l'image de la caméra.

La première amélioration par rapport au système précédent est la modélisation plus fine sous forme de patchs plan de l'environnement. Cette opération permet, en prédisant les mouvements du véhicule lors de la localisation, d'anticiper les déformations des amers et ainsi de les reconnaitre plus facilement malgré des changements de point de vue. Cette adaptabilité rend l'algorithme plus robuste aux changements de point de vue, tout en restant suffisamment discriminant pour ne pas confondre deux amers qui se ressembleraient. Avec cette reconstruction certaines informations complémentaires sur les amers telles que leur zone d'observabilité ou leur qualité permet d'améliorer la mise en correspondance pour limiter le traitement aux amers les plus pertinents.

Ensuite, la phase de localisation utilise cette modélisation pour transformer les patchs en une image semblable à celle observée par la caméra. Cette adaptation est possible en combinant les informations 3D acquises lors de la reconstruction et une prédiction de la pose courante basée sur le modèle d'évolution du robot. Les premières expérimentations ont montré que, au prix d'une prédiction correcte de la pose, l'appariement peut être plus fiable qu'en utilisant le descripteur SIFT connu pour être robuste aux changements de point de vue.

L'algorithme nécessitant de nombreux calculs, en particulier pour adapter les patchs à la vue courante, il a été implémenté sur une architecture massivement parallèle (le GPU), de manière à pouvoir être utilisé en temps réel à bord d'un véhicule. L'algorithme s'adapte particulièrement bien à ce type d'architecture et permet de tirer partie des nombreuses fonctions intrinsèques au GPU.

Pour finir afin de disposer d'une prédiction aussi fiable que possible en toutes circonstances, la localisation par vision a été fusionnée avec une localisation par odométrie pour obtenir un système amélioré tirant partie des avantages des deux capteurs tout en limitant leurs inconvé-

nients respectifs.

Plusieurs expérimentations sur des données réelles ont également été réalisées et ont montré que l'algorithme est bien adapté au fonctionnement en navigation autonome. Il permet entre autre de s'éloigner grandement de la trajectoire d'apprentissage, par exemple pour éviter un obstacle, sans perdre sa localisation, et se montre robuste à des contraintes telles que l'occultation. La précision de la localisation a également pu être validée par une comparaison avec un récepteur GPS différentiel.

Perspectives

Si les améliorations apportées à ce système ont montré leur efficacité, il demeure quelques points qui peuvent encore être améliorés. Tout d'abord, réaliser la navigation autonome en ville ne se limite pas à une tache de localisation, et de nombreux autres travaux concernant la planification des trajectoires, la détection et l'évitement d'obstacles ou encore la mise à jour de la carte 3D doivent être menés.

Même en se limitant à la localisation du véhicule, plusieurs points peuvent être améliorés sur notre système. Il s'agit essentiellement d'apporter une meilleure robustesse au système, ou d'augmenter le champ d'application de ce système, par exemple en lui permettant de fonctionner dans des environnements plus vastes.

Problème de la pose initiale

L'ensemble de l'algorithme nécessite d'avoir une prédiction de la pose courante pour pouvoir obtenir une localisation précise. Cependant au tout début de la localisation, lorsque la première image est acquise aucune prédiction n'est possible et il est donc impossible de se localiser dans ces conditions. Pour réaliser nos expérimentations, cette prédiction initiale est fournie à l'algorithme et il est donc nécessaire de lancer la localisation en partant toujours du même endroit. Il serait intéressant de trouver une méthode permettant de trouver sa position sans aucun a priori initial. Pour cela plusieurs pistes peuvent être explorées, telles que réaliser la localisation à partir de plusieurs prédictions possibles (en se basant sur la trajectoire à suivre) et conserver celle qui semble la plus fiable (avec le plus d'inlier), utiliser une méthode classique utilisant les sacs de mots visuels qui résolvent ce problème ou encore utiliser un autre capteur tel que le GPS pour initialiser sa position. Cela permettrai de pouvoir démarrer le véhicule de n'importe quelle position à proximité de la trajectoire et même, éventuellement d'être capable de pouvoir réinitialiser sa position s'il s'égare complètement.

Amélioration des patchs

La modélisation des patchs utilisée a le mérite de poser les bases d'un système s'adaptant à une pose prédite, et les premières expérimentations montrent que cette méthode est utilisable.

Néanmoins cette modélisation peut être affinée de manière à rendre encore plus fiable les patchs calculés.

Facteur de qualité

Tout d'abord le facteur de qualité utilisé, s'il permet d'enlever un certains nombre de patchs issus de mauvais appariements, n'est malgré tout pas parfait. En particulier de nombreux patchs quasiment uniformes parviennent à avoir un bon score d'appariement avec n'importe quelle image, et à ce titre ils ne sont donc pas très discriminants. Une amélioration possible serait donc de calculer le facteur de qualité, non seulement en comparant le patch avec les points qui l'ont créé, mais aussi en analysant par exemple le contraste de la texture, ou encore le champ de vue d'où il a été aperçu. Cela permettrait de supprimer davantage de mauvais patchs et ainsi limiter les risques de localisation erronée.

Texture

Pour améliorer intrinsèquement les patchs, il peut aussi être intéressant de réfléchir à l'intérêt de ne conserver qu'une seule image décrivant un même amer. En particulier lorsqu'un même point a été suivi sur une grande distance, son apparence entre des points de vue très éloignés peut être différente de celle vue de très près. Ainsi il serait possible de considérer plusieurs textures pour un même patch et d'utiliser l'une ou l'autre en fonction de la projection que l'on veut réaliser. Cette méthode pourrait fonctionner aussi si un point était suivi sur des points de vue très différents, sa texture serait générée pour plusieurs angles de vue, et l'image la plus proche de la prédiction serait alors reprojetée. L'intérêt serait de conserver une image proche de la réalité, sans trop de déformations en toutes circonstances, et en limitant grandement le flou provoqué par le calcul de la moyenne.

Génération des patchs

Actuellement, les patchs sont générés en se basant sur des points suivis sur plusieurs images successives. Cependant, le suivi ayant été réalisé avec une ZNCC, lorsque le point de vue commence à changer, le point est perdu et ainsi les images du point prises depuis d'autres points de vue ne sont pas utilisées. Il pourrait être intéressant, après avoir généré initialement les patchs, de refaire un suivi sur les images d'apprentissage et si d'autres vues sont trouvées, de relancer un calcul de normale et de la texture du patch. Cela permettrait d'avoir une modélisation plus fiable des patchs et donc globalement une meilleure qualité de reconstruction.

De plus il serait possible d'utiliser les images provenant des deux caméras embarquées à bord du véhicule lors de l'apprentissage. Ainsi les patchs seraient générés à partir d'images du point provenant de points de vue très différents (vue depuis l'avant et l'arrière du véhicule). Là encore cela pourrait améliorer le calcul des normales et de la texture des patchs. Plus encore, la

reconstruction entière réalisée auparavant pourrait être optimisée en prenant compte les appariements entre les deux caméras. En effet, si chaque amer est suivi sur une plus grande période, et successivement sur la caméra avant et arrière, la position du véhicule et les positions relatives des caméras peuvent être connues avec une bonne précision et ainsi limiter considérablement la dérive faite lors de la reconstruction.

Amélioration de la robustesse au changement de lumière

Une difficulté inhérente au système de localisation par vision est due au changement de luminosité. L'algorithme décrit dans ce document en subit également les conséquences, en particulier sur les trajectoires en extérieur où par exemple la position du soleil varie au cours de la journée.

Localisation bicaméra

Une première possibilité pour éviter les cas où le système est face au soleil consiste à utiliser les deux caméras embarquées à bord du véhicule. L'utilisation d'un système bicaméra avec l'approche par patch pourrait déjà améliorer la reconstruction comme décrit précédemment. Mais son utilisation lors de la localisation pourrait également permettre de limiter les contraintes liées à l'illumination. Un travail similaire a déjà été réalisé en utilisant la méthode précédente mais l'intérêt des patchs-plan serait de pouvoir utiliser les mêmes amers avec les deux caméras, en projetant correctement les patchs sur chaque image. Les caméras étant orientées dans des directions opposées, le soleil ne peut normalement (hors cas de réflexion sur des vitres) pas gêner simultanément les deux caméras.

Descripteur insensible au changement de lumière

Pour pouvoir être plus robuste au changement de luminosité, il pourrait également être intéressant d'utiliser des descripteurs robustes à ces changements. L'appariement par ZNCC utilisé dans cet algorithme est certes robuste aux changements affines de luminosité mais il pourrait être envisageable, en se basant sur les descripteurs existants de chercher des méthodes d'appariement capables d'intégrer des changements plus divers liés par exemple aux ombres portées.

Utilisation dans des environnements plus vastes

Le système actuel est limité à des environnements de taille réduite, à cause de la forte charge mémoire nécessaire pour garder tous les patchs actifs lors de la localisation. Néanmoins il serait possible de modifier l'utilisation des patchs pour diminuer cette charge et permettre ainsi l'utilisation de l'algorithme dans des environnements plus larges.

Diviser l'environnement

Une première possibilité pour gérer des environnements plus vastes serait de subdiviser ces environnements en plusieurs zones. Chaque zone serait associée à un certain nombre de patchs. A chaque fois qu'un véhicule entre dans une zone, les patchs associés à la nouvelle zone sont chargés en mémoire à la place de ceux de l'ancienne. Ainsi on obtient à chaque fois un système moins lourd en mémoire car limité à une petite partie. L'inconvénient est que certains patchs seront présents dans plusieurs zones et donc apparaitront en double. De plus une certaine charge de calcul sera monopolisée lors des changements de zone. Il est donc nécessaire d'avoir une gestion de la mémoire anticipant les zones traversées pour charger à l'avance les patchs de la zone suivante.

Sélection des patchs en ligne

Une autre solution qui pourrait alléger grandement la charge de calcul à chaque image serait de sélectionner les patchs en ligne. L'algorithme actuel considère tous les patchs visibles et réalise la reprojection de chacun d'entre eux. Cependant utiliser tant d'amers pour se localiser est discutable. Il serait intéressant de trier les patchs à chaque itération de manière à savoir a priori lesquels sont les plus intéressants à reprojeter. Ainsi on peut choisir de se limiter à un nombre N fixe de reprojections pour chaque itération et ce, même si d'autres patchs auraient pu s'apparier. Cela aurait en plus l'avantage de fixer le temps de calcul de chaque itération, indépendamment de la position du véhicule.

Pour trier les patchs à chaque itération, deux pistes peuvent être explorées. La première consiste à se baser sur les positions a priori du véhicule et des patchs. En regardant leur zone d'observabilité, leur qualité et éventuellement d'autres critères on pourrait (par exemple en utilisant une analyse en composantes principales) déterminer une valeur proportionnelle à l'intérêt du patch pour l'image courante. Les N patchs ayant les meilleurs scores seraient alors ceux qui sont reprojetés.

Un second moyen de sélectionner les patchs courants serait de réaliser une heuristique qui consiste à ne garder que les patchs qui étaient inliers à l'itération précédente, d'enlever ceux qui ont été outlier et de les remplacer par d'autre patchs de l'ensemble pris soit aléatoirement, soit en fonction d'un critère de qualité. L'intérêt de cette méthode est que l'on conserve les patchs qui sont réellement utiles à la localisation ; toutefois il faut veiller à ce que de nouveaux patchs arrivent, en particulier si l'on risque de s'éloigner de la zone courante.

Gestion des changements de l'environnement

Une dernière difficulté de l'algorithme est liée à sa dépendance à l'environnement. En effet, le système utilise uniquement des amers enregistrés lors de la reconstruction préalable. Cependant, avec le temps, certains amers peuvent disparaitre et les images initiales devenir assez

différentes de l'image de l'environnement quelques temps plus tard. Il est donc intéressant de trouver d'autres moyens de se localiser malgré les changements de l'environnement.

Odométrie visuelle

Une première solution est particulièrement intéressante lorsque les changements sont restreints à quelques positions, par exemple la présence d'un nouveau véhicule stationné masquant une partie de l'environnement, ou des travaux d'entretien réalisés sur une partie de la route empruntée. Elle consiste à utiliser, en plus de l'algorithme de localisation par patch, une méthode d'odométrie visuelle. En effet la localisation actuelle réalise de nombreuses tâches (détection des points, appariement des amers) qui pourraient être utiles à un algorithme d'odométrie visuelle. Cela permettrait de pouvoir se localiser avec des amers visuels qui ne sont pas intégrés dans la carte et d'en déduire, non pas une position absolue comme l'algorithme actuel, mais une bonne estimation du déplacement du véhicule. En particulier l'angle de rotation serait vraisemblablement estimé avec une meilleure précision que celle réalisée à l'aide des capteurs odométriques utilisés. Ensuite, un processus de fusion proche de celui utilisé avec ces capteurs permettrait d'intégrer les résultats de l'algorithme et rendrait le système plus robuste aux occultations des patchs.

L'algorithme de localisation par patchs restera nécessaire pour fournir une position absolue, dans le repère de la carte, et il faudra donc conserver des parties de la carte visibles pour éviter une trop grande dérive liée à la navigation par odométrie.

Mise à jour de la carte

Une solution pour pallier les changements de l'environnement serait de mettre à jour la carte. En particulier il faudrait être capable de détecter que certains patchs ne sont plus visibles, et donc les éliminer et surtout pouvoir acquérir de nouveaux amers au fur et à mesure de la trajectoire et, lorsqu'ils sont observés suffisamment longtemps pour être considérés comme fixes, de les intégrer dans la carte. Cela nécessite d'intégrer un processus de type SLAM, de mémoriser les amers qui ont pu être suivis sur de nombreuses images et d'intégrer le calcul des patchs au fur et à mesure, et ce en plus de l'algorithme de localisation. La principale difficulté réside dans le choix des critères pour déterminer quand une donnée doit être intégrée à la carte ou quand elle doit être supprimée. En particulier cela doit permettre de faire un compromis pour conserver une carte suffisamment riche et à jour pour se localiser, sans avoir trop de données pour ne pas alourdir la mémoire et le temps de calcul de la localisation.

Publications dans le cadre de cette thèse

Article de revue

Vision-based robot localization based on efficient matching using planar features • Baptiste Charmette, Éric Royer et Frédéric Chausse • EURASIP Journal on Image and Video Processing • soumis le 1/08/2012

Article de conférence internationales

Robot Localization using efficient planar features matching • Baptiste Charmette, Éric Royer, Frédéric Chausse et Laurent Lequièvre • International Conference on Intelligent Robots and Systems (IROS 2012), workshop on planning, perception and navigation for intelligent vehicles • Vilamoura, Algarve, Portugal

Efficient planar features matching for robot localization using GPU • Baptiste Charmette, Éric Royer et Frédéric Chausse • Computer Vision and Pattern Recognition Workshops (CVPR 2010), Sixth IEEE Workshop on Embedded Computer Vision • San Francisco, États-Unis

Matching Planar Features for Robot Localization • Baptiste Charmette, Éric Royer et Frédéric Chausse • International Symposium on Visual Computing (ISVC 2009) • Las Vegas, États-Unis

Article de conférence nationales

Mise en correspondance rapide de patchs plan pour la localisation d'un robot en utilisant le GPU • Baptiste Charmette, Frédéric Chausse et Éric Royer • Congrès des jeunes chercheurs en vision par ordinateur (ORASIS 2011) • Praz-sur-Arly, France

Mise en correspondance de patchs plan pour la localisation d'un robot • Baptiste Charmette, Éric Royer et Frédéric Chausse • Reconnaissance des Formes et Intelligence artificielle (RFIA 2010) • Caen, France

Annexe A

Homographie induite par un plan

A.1 Introduction

Dans le livre de Hartley and Zisserman (2004), l'équation de l'homographie induite par un plan est donnée sous la forme suivante :

$$H_{1\to 2} = K(R_{1\to 2} - \frac{\mathbf{T_{1\to 2}}{\mathbf{N_1}}^t}{d_1})K^{-1} \tag{A.1}$$

avec

- $R_{1\to 2}$ et $\mathbf{T_{1\to 2}}$ la matrice de rotation et le vecteur de translation qui transforment les coordonnées dans la caméra 1 en coordonnées dans la caméra 2 selon l'équation $\mathbf{P_{c2}} = R_{1\to 2}\mathbf{P_{c1}} + \mathbf{T_{1\to 2}}$;
- d_1 la distance algébrique du centre de la caméra 1 au plan ;
- $\mathbf{N_1}$ la normale au plan exprimé dans le repère de la caméra 1 ;
- K la matrice des paramètres intrinsèques des caméras (supposée identique pour les deux vues).

Cette formule s'applique lorsque le repère de la première caméra est considéré comme l'origine du repère, et en utilisant les conventions (sur les matrices de rotation et vecteurs de translation) définies dans le livre.

Pour obtenir l'équation en suivant notre convention, et dans le cas où le repère monde n'est pas le même que celui de la première caméra, on va retrouver cette équation.

A.2 Données initiales

Plusieurs données sont supposées connues initialement comme les poses de caméra et l'équation du plan. Ces valeurs sont représentées sur la figure 2.4.

A.2.1 Pose de caméras

On suppose les poses de chaque caméra connues, c'est-à-dire que pour chacune des deux caméras on connait les matrices de rotation R_1,R_2 et les vecteurs de translation $\mathbf{T_1}$,$\mathbf{T_2}$ qui permettent de passer du repère monde au repère caméra selon le modèle décrit dans la partie 2.2. On suppose également que la matrice de paramètres intrinsèques K est la même pour chaque caméra.

A.2.2 Plan

L'homographie étant induite par un plan, on suppose que les points de chaque image sont situés sur un même plan Π. Ce plan passe par un point $\mathbf{P_w}$ connu. La normale $\mathbf{N} = (N_x, N_y, N_z)^T$ à ce plan est également connue et exprimée dans le repère monde.

Pour définir l'équation du plan on crée le vecteur $\mathbf{\Pi} = (N^T, -d)^T = (N_x, N_y, N_z, -d)^T$. Tout point de coordonnées $(x, y, z, 1)^T$ dans le repère monde appartenant au plan Π vérifie l'équation de la forme A.3

$$\mathbf{\Pi}^T \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0 \tag{A.2}$$

avec d une valeur constante. En particulier, le point $\mathbf{P_w}$ appartient à ce plan, par conséquent, on peut calculer la valeur d en posant :

$$\begin{aligned} 0 &= \Pi^T \mathbf{P_w} \\ d &= N_x X_w + N_y Y_w + N_z Z_w = \mathbf{N}^T \mathbf{P_w} \end{aligned} \tag{A.3}$$

donc d est distance algébrique du plan a l'origine (et la normale est supposée orientée vers l'origine).

A.3 Définition de l'homographie

L'homographie induite par le plan est la transformation qui permet de relier les coordonnées $\mathbf{P_{p1}}$ de chaque point dans la première image à ses coordonnées $\mathbf{P_{p2}}$ dans la seconde image avec l'équation suivante :

$$\mathbf{P_{p2}} \equiv H_{1\to 2} \mathbf{P_{p1}} \tag{A.4}$$

Nous savons que cette transformation est une homographie et peut donc s'exprimer sous la forme d'une matrice carré de taille 3×3 s'appliquant aux coordonnées homogènes. C'est la valeur de cette homographie que l'on va recalculer.

A.4 Calcul de l'homographie

Pour trouver l'homographie, il faut exprimer la relation entre les coordonnées d'un point dans le repère de la première image avec ses coordonnées dans la seconde image. Pour cela, on va dans un premier temps trouver la relation permettant de passer des coordonnées dans la première image aux coordonnées 3D dans le repère monde. Ensuite en projetant les coordonnées du repère monde dans la seconde caméra on pourra obtenir la relation liant les coordonnées de chaque image.

A.4.1 Passage des coordonnées dans l'image 1 vers le monde

D'après le modèle de caméra et l'équation 2.3, les coordonnées homogènes $\mathbf{P_{c1}}$ du point 3D $\mathbf{P_w}$ dans le repère lié à la caméra 1 sont liées aux coordonnées homogènes $\mathbf{P_{i1}}$ dans l'image par la relation :

$$\mathbf{P_{c1}} = \begin{bmatrix} sx_{i1} \\ sy_{i1} \\ s \\ 1 \end{bmatrix} \qquad \mathbf{P_{c1}} = \begin{bmatrix} s\mathbf{P_{i1}} \\ 1 \end{bmatrix} \tag{A.5}$$

avec s une valeur réelle constante. En posant $\rho = 1/s$, on obtient :

$$\rho\mathbf{P_{c1}} = \begin{bmatrix} \mathbf{P_{i1}} \\ \rho \end{bmatrix} \tag{A.6}$$

En inversant l'équation 2.1 reliant les coordonnées dans le repère caméra à celle du repère monde, on obtient :

$$\mathbf{P_w} = \begin{bmatrix} R_1^T & \mathbf{T_1} \\ 0 & 1 \end{bmatrix} \mathbf{P_{c1}} \tag{A.7}$$

$$\rho\mathbf{P_w} = \begin{bmatrix} R_1^T & \mathbf{T_1} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{P_{i1}} \\ \rho \end{bmatrix} \tag{A.8}$$

Cela forme une infinité de solutions variant en fonction de ρ. En effet tous les points situés sur une même droite passant par le centre de la caméra se projettent au même endroit dans l'image. En l'absence d'autres informations, on ne pourrait donc pas connaitre complètement les coordonnées du point. Cependant on sait que le point $\mathbf{P_w}$ appartient au plan Π. En utilisant cet a priori la position du point devient unique. On peut donc trouver la valeur de ρ, en appliquant la

valeur de $\mathbf{P_w}$ dans l'équation du plan A.3

$$\begin{aligned}
0 &= \mathbf{\Pi}^T \mathbf{P_w} \\
0 &= \begin{bmatrix} \mathbf{N}^T & -d \end{bmatrix} \begin{bmatrix} R_1^T & \mathbf{T_1} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{P_{i1}} \\ \rho \end{bmatrix} \\
0 &= \begin{bmatrix} \mathbf{N}^T & -d \end{bmatrix} \begin{bmatrix} R_1^T \mathbf{P_{i1}} + \rho \mathbf{T_1} \\ \rho \end{bmatrix} \\
\rho d &= \mathbf{N}^T R_1^T \mathbf{P_{i1}} + \rho \mathbf{N}^T \mathbf{T_1} \\
\rho &= \frac{\mathbf{N}^T R_1^T \mathbf{P_{i1}}}{d - \mathbf{N}^T \mathbf{T_1}} \\
\rho &= \frac{\mathbf{N}^T R_1^T \mathbf{P_{i1}}}{\mathbf{N}^T \mathbf{P_w} - \mathbf{N}^T \mathbf{T_1}}
\end{aligned} \tag{A.9}$$

On a alors une solution unique pour les coordonnées $\mathbf{P_w}$:

$$\rho \mathbf{P_w} = \begin{bmatrix} R_1^T & \mathbf{T_1} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{P_{i1}} \\ \frac{\mathbf{N}^T R_1^T \mathbf{P_{i1}}}{\mathbf{N}^T \mathbf{P_w} - \mathbf{N}^T \mathbf{T_1}} \end{bmatrix} \tag{A.10}$$

A.4.2 Projection du repère monde vers le repère de l'image 2

On écrit l'équation de reprojection dans la seconde image. Du fait des coordonnées homogènes, l'équation est définie à un facteur d'échelle près, on peut donc ignorer le facteur ρ devant $\mathbf{P_w}$ et on obtient :

$$\begin{aligned}
\mathbf{P_{i2}} &\equiv \begin{bmatrix} R_2 & -R_2 \mathbf{T_2} \end{bmatrix} \mathbf{P_w} \\
\mathbf{P_{i2}} &\equiv \begin{bmatrix} R_2 & -R_2 \mathbf{T_2} \end{bmatrix} \begin{bmatrix} R_1^T & \mathbf{T_1} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{P_{i1}} \\ \rho \end{bmatrix} \\
\mathbf{P_{i2}} &\equiv \begin{bmatrix} R_2 R_1^T & R_2(\mathbf{T_1} - \mathbf{T_2}) \end{bmatrix} \begin{bmatrix} \mathbf{P_{i1}} \\ \frac{\mathbf{N}^T R_1^T \mathbf{P_{i1}}}{\mathbf{N}^T \mathbf{P_w} - \mathbf{N}^T \mathbf{T_1}} \end{bmatrix} \\
\mathbf{P_{i2}} &\equiv \left(R_2 R_1^T + R_2(\mathbf{T_1} - \mathbf{T_2}) \frac{\mathbf{N}^T R_1^T}{\mathbf{N}^T \mathbf{P_w} - \mathbf{N}^T \mathbf{T_1}} \right) \mathbf{P_{i1}}
\end{aligned} \tag{A.11}$$

On obtient ainsi la relation entre les coordonnées 2D dans l'image 1 et celle dans l'image 2. Pour avoir la relation directement depuis les coordonnées en pixel, on utilise la matrice de paramètres

intrinsèques K.

$$\begin{aligned}
\mathbf{P_{i2}} &= \left(R_2 R_1^T + R_2(\mathbf{T_1} - \mathbf{T_2}) \frac{\mathbf{N}^T R_1^T}{\mathbf{N}^T \mathbf{P_w} - \mathbf{N}^T \mathbf{T_1}} \right) \mathbf{P_{i1}} \\
\mathbf{P_{p2}} &= K \left(R_2 R_1^T + R_2(\mathbf{T_1} - \mathbf{T_2}) \frac{\mathbf{N}^T R_1^T}{\mathbf{N}^T \mathbf{P_w} - \mathbf{N}^T \mathbf{T_1}} \right) \mathbf{P_{i1}} \\
\mathbf{P_{p2}} &= K \left(R_2 R_1^T + R_2(\mathbf{T_1} - \mathbf{T_2}) \frac{\mathbf{N}^T R_1^T}{\mathbf{N}^T \mathbf{P_w} - \mathbf{N}^T \mathbf{T_1}} \right) K^{-1} \mathbf{P_{p1}}
\end{aligned} \tag{A.12}$$

A.4.3 Valeur de l'homographie

On déduit du résultat la valeur de l'homographie reliant $\mathbf{P_{p1}}$ à $\mathbf{P_{p2}}$:

$$H_{1\to 2} = K \left(R_2 R_1^T + R_2(\mathbf{T_1} - \mathbf{T_2}) \frac{\mathbf{N}^T R_1^T}{\mathbf{N}^T \mathbf{P_w} - \mathbf{N}^T \mathbf{T_1}} \right) K^{-1} \tag{A.13}$$

On peut remarquer que bien que $\mathbf{P_w}$ fasse partie de l'équation, l'homographie reste la même pour tout point du plan Π.

En effet tout point $\mathbf{P_w}'$ appartenant à Π mais différent de $\mathbf{P_w}$ pourrait s'écrire

$$\mathbf{P_w}' = \mathbf{P_w} + \mathbf{u_\Pi} \tag{A.14}$$

avec $\mathbf{u_\Pi}$ un vecteur du plan Π

Cependant dans l'équation de l'homographie, $\mathbf{P_w}$ n'apparait que à travers le calcul de la valeur $\mathbf{N}^T\mathbf{P_w}$. Si on applique le calcul en $\mathbf{P_w}'$ on obtient :

$$\begin{aligned}
\mathbf{N}^T\mathbf{P_w}' &= \mathbf{N}^T(\mathbf{P_w} + \mathbf{u_\Pi}) \\
\mathbf{N}^T\mathbf{P_w}' &= \mathbf{N}^T\mathbf{P_w} + \mathbf{N}^T\mathbf{u_\Pi} \\
\mathbf{N}^T\mathbf{P_w}' &= \mathbf{N}^T\mathbf{P_w} \quad \text{car } \mathbf{u_\Pi} \text{ appartient au plan donc } \mathbf{N}^T\mathbf{u_\Pi} = 0
\end{aligned} \tag{A.15}$$

Donc l'équation de l'homographie est bien la même pour tous les points du plan Π.

Annexe B

Linéarisation de la fonction d'état

Dans le modèle du filtre utilisé, l'équation d'état utilisée lors de la prédiction consiste à considérer la vitesse du véhicule constante. On a donc obtenu l'équation d'état décrite par l'équation 4.22 et rappelée ci-dessous :

$$\mathbf{X_{k+1}} = \begin{bmatrix} x_{k+1} \\ z_{k+1} \\ \theta_{k+1} \\ \overline{v}_{k+1} \\ \dot{\theta}_{k+1} \end{bmatrix} = \begin{bmatrix} x_k - \frac{\overline{v}_k}{\dot{\theta}_k}\Big(\cos(\theta_k - \dot{\theta}_k \Delta t_k) - \cos\theta_k\Big) \\ z_k - \frac{\overline{v}_k}{\dot{\theta}_k}\Big(\sin(\theta_k - \dot{\theta}_k \Delta t_k) - \sin\theta_k\Big) \\ \theta_k + \dot{\theta}_k \Delta t_k \\ \overline{v}_k \\ \dot{\theta}_k \end{bmatrix} + \mathbf{w_k} \tag{B.1}$$

Du fait de la présence des fonctions trigonométriques, cette fonction n'est pas linéaire, et il est donc nécessaire d'en calculer la jacobienne J_k^{fX}.

B.1 Jacobienne

Le calcul de la jacobienne nécessite de calculer les dérivées partielles pour chaque élément de la fonction en fonction des autres, en supposant que chaque élément est indépendant des autres. On obtient ainsi la matrice suivante :

$$J_k^{fX} = \begin{bmatrix} \frac{\partial x_{k+1}}{\partial x_k} & \frac{\partial x_{k+1}}{\partial z_k} & \frac{\partial x_{k+1}}{\partial \theta_k} & \frac{\partial x_{k+1}}{\partial \overline{v}_k} & \frac{\partial x_{k+1}}{\partial \dot{\theta}_k} \\ \frac{\partial z_{k+1}}{\partial x_k} & \frac{\partial z_{k+1}}{\partial z_k} & \frac{\partial z_{k+1}}{\partial \theta_k} & \frac{\partial z_{k+1}}{\partial \overline{v}_k} & \frac{\partial z_{k+1}}{\partial \dot{\theta}_k} \\ \frac{\partial \theta_{k+1}}{\partial x_k} & \frac{\partial \theta_{k+1}}{\partial z_k} & \frac{\partial \theta_{k+1}}{\partial \theta_k} & \frac{\partial \theta_{k+1}}{\partial \overline{v}_k} & \frac{\partial \theta_{k+1}}{\partial \dot{\theta}_k} \\ \frac{\partial \overline{v}_{k+1}}{\partial x_k} & \frac{\partial \overline{v}_{k+1}}{\partial z_k} & \frac{\partial \overline{v}_{k+1}}{\partial \theta_k} & \frac{\partial \overline{v}_{k+1}}{\partial \overline{v}_k} & \frac{\partial \overline{v}_{k+1}}{\partial \dot{\theta}_k} \\ \frac{\partial \dot{\theta}_{k+1}}{\partial x_k} & \frac{\partial \dot{\theta}_{k+1}}{\partial z_k} & \frac{\partial \dot{\theta}_{k+1}}{\partial \theta_k} & \frac{\partial \dot{\theta}_{k+1}}{\partial \overline{v}_k} & \frac{\partial \dot{\theta}_{k+1}}{\partial \dot{\theta}_k} \end{bmatrix} \tag{B.2}$$

B.1.1 Cas simple

Les variables étant supposées indépendantes, le calcul des dérivées par rapport à x_k et z_k ne pose pas de problèmes. En effet, ces deux paramètres n'apparaissent que dans une seule équation et sous forme d'addition sans coefficient. Les dérivées sont donc nulles la plupart du temps et égales à 1 lorsque le paramètre apparait. On a donc pour les dérivées par rapport à x_k et z_k les résultats suivants :

$$\frac{\partial x_{k+1}}{\partial x_k} = 1 \qquad \frac{\partial z_{k+1}}{\partial x_k} = 0 \qquad \frac{\partial \theta_{k+1}}{\partial x_k} = 0 \qquad \frac{\partial \overline{v}_{k+1}}{\partial x_k} = 0 \qquad \frac{\partial \dot{\theta}_{k+1}}{\partial x_k} = 0$$
$$\frac{\partial x_{k+1}}{\partial z_k} = 0 \qquad \frac{\partial z_{k+1}}{\partial z_k} = 0 \qquad \frac{\partial \theta}{\partial z_k} = 1 \qquad \frac{\partial \overline{v}_{k+1}}{\partial z_k} = 0 \qquad \frac{\partial \dot{\theta}_{k+1}}{\partial z_k} = 0$$

Les expression de θ_{k+1}, $\overline{v}_{k+1}$ et $\dot{\theta}_{k+1}$ sont également très simples et ne font pratiquement pas intervenir les autres variables. Les dérivées de ces fonctions sont donc également immédiates :

$$\frac{\partial \theta_{k+1}}{\partial \theta_k} = 1 \qquad \frac{\partial \theta_{k+1}}{\partial \overline{v}_k} = 0 \qquad \frac{\partial \theta_{k+1}}{\partial \dot{\theta}_k} = \Delta t_k$$
$$\frac{\partial \overline{v}_{k+1}}{\partial \theta_k} = 0 \qquad \frac{\partial \overline{v}_{k+1}}{\partial \overline{v}_k} = 1 \qquad \frac{\partial \overline{v}_{k+1}}{\partial \dot{\theta}_k} = 0$$
$$\frac{\partial \dot{\theta}_{k+1}}{\partial \theta_k} = 0 \qquad \frac{\partial \dot{\theta}_{k+1}}{\partial \overline{v}_k} = 0 \qquad \frac{\partial \dot{\theta}_{k+1}}{\partial \dot{\theta}_k} = 1$$

Pour finir, les coordonnées x_{k+1} et z_{k+1} sont directement proportionnelle à la vitesse linéaire $\overline{v}_k$. Leur dérivée par rapport à cette dernière est donc directement le coefficient de proportionnalité :

$$\frac{\partial x_{k+1}}{\partial \overline{v}_k} = -\frac{1}{\dot{\theta}_k}\Big(\cos(\theta_k - \dot{\theta}_k \Delta t_k) - \cos\theta_k\Big)$$
$$\frac{\partial z_{k+1}}{\partial \overline{v}_k} = -\frac{1}{\dot{\theta}_k}\Big(\sin(\theta_k - \dot{\theta}_k \Delta t_k) - \sin\theta_k\Big)$$

B.1.2 Dérivée des coordonnées par rapport à l'angle

Pour calculer les 4 dérivées restantes, on va poser $\alpha_k = \theta_k - \dot{\theta}_k \Delta t_k$. Cela permet de définir les fonctions suivantes :

$$c_{\theta_k} = \cos\theta_k \qquad s_{\theta_k} = \sin\theta_k$$
$$c_{\alpha_k} = \cos\alpha_k = \cos(\theta_k - \dot{\theta}_k \Delta t_k) \qquad s_{\alpha_k} = \sin\alpha_k = \sin(\theta_k - \dot{\theta}_k \Delta t_k)$$

α_k étant à un décalage près identique à θ_k, on a $\frac{\partial \alpha_k}{\partial \theta_k} = 1$. Les dérivées par rapport à θ_k de chaque fonction sont alors :

$$\frac{\partial c_{\theta_k}}{\partial \theta_k} = -\sin\theta_k = -s_{\theta_k} \qquad \frac{\partial s_{\theta_k}}{\partial \theta_k} = \cos\theta_k = c_{\theta_k}$$
$$\frac{\partial c_{\alpha_k}}{\partial \theta_k} = -\sin\alpha_k = -s_{\alpha_k} \qquad \frac{\partial s_{\alpha_k}}{\partial \theta_k} = \cos\alpha_k = c_{\alpha_k}$$

On remplace ensuite dans les expressions de x_{k+1} et z_{k+1} :

$$\begin{aligned} x_{k+1} &= x_k - \frac{\overline{v}_k}{\dot{\theta}_k}\left(c_{\alpha_k} - c_{\theta_k}\right) \\ z_{k+1} &= z_k - \frac{\overline{v}_k}{\dot{\theta}_k}\left(s_{\alpha_k} - s_{\theta_k}\right) \end{aligned} \tag{B.3}$$

on obtient alors les dérivées suivantes :

$$\begin{aligned} \frac{\partial x_{k+1}}{\partial \theta_k} &= -\frac{\overline{v}_k}{\dot{\theta}_k}\left(-s_{\alpha_k} + s_{\theta_k}\right) = \frac{\overline{v}_k}{\dot{\theta}_k}\left(s_{\alpha_k} - s_{\theta_k}\right) \\ \frac{\partial z_{k+1}}{\partial \theta_k} &= -\frac{\overline{v}_k}{\dot{\theta}_k}\left(c_{\alpha_k} - c_{\theta_k}\right) \end{aligned} \tag{B.4}$$

B.1.3 Dérivée des coordonnées par rapport à la vitesse angulaire

Pour la dérivée par rapport à $\dot{\theta}_k$, on utilise les mêmes fonctions que précédemment. Cependant, la dérivée partielle de α_k n'est plus égale à 1, mais à une valeur constante :

$$\frac{\partial \alpha_k}{\partial \dot{\theta}_k} = -\Delta t_k$$

Cela donne pour les dérivées des fonctions liées à α_k :

$$\frac{\partial c_{\alpha_k}}{\partial \dot{\theta}_k} = -\Delta t_k(-\sin\alpha_k) = \Delta t_k s_{\alpha_k} \qquad \frac{\partial s_{\alpha_k}}{\partial \dot{\theta}_k} = -\Delta t_k \cos\alpha_k = -\Delta t_k c_{\alpha_k}$$

Les valeurs d'état étant considérées comme indépendantes, les dérivées de c_{θ_k} et s_{θ_k} sont nulles.

On pose par ailleurs les valeurs u_x et u_z, les numérateurs du second terme de x_{k+1} et z_{k+1}.

$$\begin{aligned} u_x &= -\overline{v}_k\left(c_{\alpha_k} - c_{\theta_k}\right) \\ u_z &= -\overline{v}_k\left(s_{\alpha_k} - s_{\theta_k}\right) \end{aligned}$$

Leur dérivée u'_x et u'_z par rapport à $\dot{\theta}_k$ vaut donc :

$$\begin{aligned} u'_x &= -\overline{v}_k \Delta t_k s_{\alpha_k} \\ u'_z &= \overline{v}_k \Delta t_k c_{\alpha_k} \end{aligned}$$

On remplace ensuite dans les expressions de x_{k+1} et z_{k+1} :

$$\begin{aligned} x_{k+1} &= x_k + \frac{u_x}{\dot{\theta}_k} \\ z_{k+1} &= z_k + \frac{u_z}{\dot{\theta}_k} \end{aligned} \tag{B.5}$$

On en déduit la valeur de la dérivée :

$$\begin{aligned} \frac{\partial x_{k+1}}{\partial \dot{\theta}_k} &= \frac{u'_x \dot{\theta}_k - u_x}{\dot{\theta}_k^2} = \frac{1}{\dot{\theta}_k^2}\left(-\overline{v}_k \Delta t_k s_{\alpha_k} \dot{\theta}_k + \overline{v}_k \left(c_{\alpha_k} - c_{\theta_k}\right)\right) = \frac{\overline{v}_k}{\dot{\theta}_k^2}\left(-\Delta t_k \dot{\theta}_k s_{\alpha_k} + c_{\alpha_k} - c_{\theta_k}\right) \\ \frac{\partial z_{k+1}}{\partial \dot{\theta}_k} &= \frac{u'_z \dot{\theta}_k - u_z}{\dot{\theta}_k^2} = \frac{1}{\dot{\theta}_k^2}\left(\overline{v}_k \Delta t_k c_{\alpha_k} \dot{\theta}_k + \overline{v}_k \left(s_{\alpha_k} - s_{\theta_k}\right)\right) = \frac{\overline{v}_k}{\dot{\theta}_k^2}\left(\Delta t_k \dot{\theta}_k c_{\alpha_k} + s_{\alpha_k} - s_{\theta_k}\right) \end{aligned} \tag{B.6}$$

B.1.4 Résultat final

Toutes les dérivées ayant été calculées, on obtient alors la jacobienne suivante :

$$J_k^{fX} = \begin{bmatrix} 1 & 0 & \frac{\overline{v}_k}{\dot{\theta}_k}\left(s_{\alpha_k} - s_{\theta_k}\right) & \frac{1}{\dot{\theta}_k}\left(c_{\theta_k} - c_{\alpha_k}\right) & \frac{\overline{v}_k}{\dot{\theta}_k^2}\left(-c_{\theta_k} - \Delta t_k \dot{\theta}_k s_{\alpha_k} + c_{\alpha_k}\right) \\ 0 & 1 & \frac{-\overline{v}_k}{\dot{\theta}_k}\left(c_{\alpha_k} - c_{\theta_k}\right) & \frac{1}{\dot{\theta}_k}\left(s_{\theta_k} - s_{\alpha_k}\right) & \frac{\overline{v}_k}{\dot{\theta}_k^2}\left(-s_{\theta_k} + \Delta t_k \dot{\theta}_k c_{\alpha_k} + s_{\alpha_k}\right) \\ 0 & 0 & 1 & 0 & \Delta t_k \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.7}$$

C'est ce résultat qui est utilisé dans le modèle. On remarque toutefois que lorsque la vitesse angulaire $\dot{\theta}_k$ est nulle, l'implémentation du calcul ne peut se faire avec l'équation B.7. Ce cas est pourtant assez fréquent, puisqu'il correspond aux moments où le véhicule se déplace en ligne droite. On va donc chercher l'expression de la jacobienne lorsque la vitesse angulaire se rapproche de 0.

B.2 Limite pour une vitesse angulaire nulle

Lorsque $\dot{\theta}_k$ se rapproche de zéro, on constate que le calcul de la jacobienne en utilisant l'équation B.7 donne des formes indéterminées. En effet, lorsqu'on détaille les calculs on obtient :

$$\begin{aligned} \lim_{\dot{\theta}_k \to 0} \alpha_k &= \lim_{\dot{\theta}_k \to 0} \theta_k - \dot{\theta}_k \Delta t_k = \theta_k \\ \lim_{\dot{\theta}_k \to 0} c_{\alpha_k} &= \lim_{\alpha_k \to \theta_k} c_{\alpha_k} = \cos\theta_k = c_{\theta_k} \\ \lim_{\dot{\theta}_k \to 0} s_{\alpha_k} &= \lim_{\alpha_k \to \theta_k} s_{\alpha_k} = \sin\theta_k = s_{\theta_k} \end{aligned} \tag{B.8}$$

Ce qui nous donne pour les dérivées des coordonnées par rapport à l'angle et à la vitesse, des formes indéterminée de type $\frac{0}{0}$.

B.2.1 Réécriture des fonctions intermédiaires

Pour lever les indéterminations on va exprimer les équations différemment. Tout d'abord, on va exprimer les valeurs c_{α_k} et s_{α_k} sous forme de sommes :

$$\begin{aligned} c_{\alpha_k} = \cos\left(\theta_k - \dot{\theta}_k \Delta t_k\right) &= \cos\left(\theta_k\right)\cos\left(\dot{\theta}_k \Delta t_k\right) + \sin\left(\theta_k\right)\sin\left(\dot{\theta}_k \Delta t_k\right) \\ &= c_{\theta_k}\cos\left(\dot{\theta}_k \Delta t_k\right) + s_{\theta_k}\sin\left(\dot{\theta}_k \Delta t_k\right) \\ s_{\alpha_k} = \sin\left(\theta_k - \dot{\theta}_k \Delta t_k\right) &= \sin\left(\theta_k\right)\cos\left(\dot{\theta}_k \Delta t_k\right) - \cos\left(\theta_k\right)\sin\left(\dot{\theta}_k \Delta t_k\right) \\ &= s_{\theta_k}\cos\left(\dot{\theta}_k \Delta t_k\right) - c_{\theta_k}\sin\left(\dot{\theta}_k \Delta t_k\right) \end{aligned} \tag{B.9}$$

On utilise ensuite les développements limités (autour de $\dot{\theta}_k = 0$) suivants :

$$\begin{aligned} \cos\left(\dot{\theta}_k \Delta t_k\right) &= 1 - \frac{\dot{\theta}_k^2 \Delta t_k^2}{2} + o(\dot{\theta}_k^2) \\ \sin\left(\dot{\theta}_k \Delta t_k\right) &= \dot{\theta}_k \Delta t_k + o(\dot{\theta}_k^2) \end{aligned}$$

On utilise ensuite ces équations pour chaque terme de la jacobienne.

B.2.2 Première forme indéterminée

Les premiers calculs concernent les termes de la forme d'une différence de fonction trigonométrique divisée par $\dot{\theta}_k$. Il s'agit des dérivées de x_{k+1} et z_{k+1} par rapport à l'angle θ_k et par rapport à la vitesse linéaire $\overline{v}_k$.

On développe les calculs tout d'abord pour la dérivée de x_{k+1} par rapport à la vitesse li-

néaire :

$$\begin{aligned}
\lim_{\dot{\theta}_k \to 0} \frac{\partial x_{k+1}}{\partial \overline{v}_k} &= \lim_{\dot{\theta}_k \to 0} \frac{1}{\dot{\theta}_k} \left(c_{\theta_k} - c_{\alpha_k} \right) \\
&= \lim_{\dot{\theta}_k \to 0} \frac{1}{\dot{\theta}_k} \left[c_{\theta_k} - \left(c_{\theta_k} \cos\left(\dot{\theta}_k \Delta t_k \right) + s_{\theta_k} \sin\left(\dot{\theta}_k \Delta t_k \right) \right) \right] \\
&= \lim_{\dot{\theta}_k \to 0} \frac{1}{\dot{\theta}_k} \left[c_{\theta_k} - c_{\theta_k} \cos\left(\dot{\theta}_k \Delta t_k \right) - s_{\theta_k} \sin\left(\dot{\theta}_k \Delta t_k \right) \right] \\
&= \lim_{\dot{\theta}_k \to 0} \frac{1}{\dot{\theta}_k} \left[c_{\theta_k} - c_{\theta_k} \left(1 + o(\dot{\theta}_k) \right) - s_{\theta_k} \left(\dot{\theta}_k \Delta t_k + o(\dot{\theta}_k) \right) \right] \\
&= \lim_{\dot{\theta}_k \to 0} \frac{1}{\dot{\theta}_k} \left[c_{\theta_k} - c_{\theta_k} - \dot{\theta}_k \Delta t_k s_{\theta_k} + o(\dot{\theta}_k) \right] \\
&= \lim_{\dot{\theta}_k \to 0} \frac{1}{\dot{\theta}_k} \left[-\dot{\theta}_k \Delta t_k s_{\theta_k} + o(\dot{\theta}_k) \right] \\
&= -\Delta t_k s_{\theta_k}
\end{aligned} \tag{B.10}$$

On fait un calcul similaire pour la dérivée de z_{k+1} par rapport à la vitesse angulaire :

$$\begin{aligned}
\lim_{\dot{\theta}_k \to 0} \frac{\partial z_{k+1}}{\partial \overline{v}_k} &= \lim_{\dot{\theta}_k \to 0} \frac{1}{\dot{\theta}_k} \left(s_{\theta_k} - s_{\alpha_k} \right) \\
&= \lim_{\dot{\theta}_k \to 0} \frac{1}{\dot{\theta}_k} \left[s_{\theta_k} - \left(s_{\theta_k} \cos\left(\dot{\theta}_k \Delta t_k \right) - c_{\theta_k} \sin\left(\dot{\theta}_k \Delta t_k \right) \right) \right] \\
&= \lim_{\dot{\theta}_k \to 0} \frac{1}{\dot{\theta}_k} \left[s_{\theta_k} - s_{\theta_k} \cos\left(\dot{\theta}_k \Delta t_k \right) + c_{\theta_k} \sin\left(\dot{\theta}_k \Delta t_k \right) \right] \\
&= \lim_{\dot{\theta}_k \to 0} \frac{1}{\dot{\theta}_k} \left[s_{\theta_k} - s_{\theta_k} \left(1 + o(\dot{\theta}_k) \right) + c_{\theta_k} \left(\dot{\theta}_k \Delta t_k + o(\dot{\theta}_k) \right) \right] \\
&= \lim_{\dot{\theta}_k \to 0} \frac{1}{\dot{\theta}_k} \left[s_{\theta_k} - s_{\theta_k} + \dot{\theta}_k \Delta t_k c_{\theta_k} + o(\dot{\theta}_k) \right] \\
&= \lim_{\dot{\theta}_k \to 0} \frac{1}{\dot{\theta}_k} \left[\dot{\theta}_k \Delta t_k c_{\theta_k} + o(\dot{\theta}_k) \right] \\
&= \Delta t_k c_{\theta_k}
\end{aligned} \tag{B.11}$$

Les valeurs des dérivées de x_{k+1} et z_{k+1} par rapport à l'angle θ_k sont les mêmes que les dérivées de respectivement z_{k+1} et x_{k+1} par rapport à la vitesse angulaire à un facteur ($-\overline{v}_k$ et $\overline{v}_k$) près. La limite en $\dot{\theta}_k = 0$ est donc la même au facteur près :

$$\begin{aligned}
\lim_{\dot{\theta}_k \to 0} \frac{\partial x_{k+1}}{\partial \theta_k} &= -\overline{v}_k \Delta t_k c_{\theta_k} \\
\lim_{\dot{\theta}_k \to 0} \frac{\partial z_{k+1}}{\partial \theta_k} &= -\overline{v}_k \Delta t_k s_{\theta_k}
\end{aligned} \tag{B.12}$$

B.2.3 Seconde forme indéterminée

L'autre forme indéterminée concerne les dérivées des coordonnées par rapport à la vitesse angulaire. On commence donc par le cas de x_{k+1} :

$$
\begin{aligned}
\lim_{\dot{\theta}_k \to 0} \frac{\partial x_{k+1}}{\partial \dot{\theta}_k} &= \lim_{\dot{\theta}_k \to 0} \frac{\overline{v}_k}{\dot{\theta}_k^2} \left(-\Delta t_k \dot{\theta}_k s_{\alpha_k} + c_{\alpha_k} - c_{\theta_k} \right) \\
&= \lim_{\dot{\theta}_k \to 0} \frac{\overline{v}_k}{\dot{\theta}_k^2} \left(-\Delta t_k \dot{\theta}_k \left(s_{\theta_k} \cos\left(\dot{\theta}_k \Delta t_k\right) - c_{\theta_k} \sin\left(\dot{\theta}_k \Delta t_k\right) \right) \right. \\
&\qquad \left. + \left(c_{\theta_k} \cos\left(\dot{\theta}_k \Delta t_k\right) + s_{\theta_k} \sin\left(\dot{\theta}_k \Delta t_k\right) \right) - c_{\theta_k} \right) \\
&= \lim_{\dot{\theta}_k \to 0} \frac{\overline{v}_k}{\dot{\theta}_k^2} \left(-\Delta t_k \dot{\theta}_k s_{\theta_k} \cos\left(\dot{\theta}_k \Delta t_k\right) + \Delta t_k \dot{\theta}_k c_{\theta_k} \sin\left(\dot{\theta}_k \Delta t_k\right) \right. \\
&\qquad \left. + c_{\theta_k} \cos\left(\dot{\theta}_k \Delta t_k\right) + s_{\theta_k} \sin\left(\dot{\theta}_k \Delta t_k\right) - c_{\theta_k} \right) \\
&= \lim_{\dot{\theta}_k \to 0} \frac{\overline{v}_k}{\dot{\theta}_k^2} \left(\left(c_{\theta_k} - \Delta t_k \dot{\theta}_k s_{\theta_k} \right) \cos\left(\dot{\theta}_k \Delta t_k\right) + \left(s_{\theta_k} + \Delta t_k \dot{\theta}_k c_{\theta_k} \right) \sin\left(\dot{\theta}_k \Delta t_k\right) - c_{\theta_k} \right) \\
&= \lim_{\dot{\theta}_k \to 0} \frac{\overline{v}_k}{\dot{\theta}_k^2} \left(\left(c_{\theta_k} - \Delta t_k \dot{\theta}_k s_{\theta_k} \right) \left(1 - \frac{\dot{\theta}_k^2 \Delta t_k^2}{2} + o(\dot{\theta}_k^2) \right) \right. \\
&\qquad \left. + \left(s_{\theta_k} + \Delta t_k \dot{\theta}_k c_{\theta_k} \right) \left(\dot{\theta}_k \Delta t_k + o(\dot{\theta}_k^2) \right) - c_{\theta_k} \right) \\
&= \lim_{\dot{\theta}_k \to 0} \frac{\overline{v}_k}{\dot{\theta}_k^2} \left(c_{\theta_k} - \Delta t_k \dot{\theta}_k s_{\theta_k} - \frac{\dot{\theta}_k^2 \Delta t_k^2}{2} c_{\theta_k} + \frac{\dot{\theta}_k^3 \Delta t_k^3}{2} s_{\theta_k} + o(\dot{\theta}_k^2) \right. \\
&\qquad \left. + \dot{\theta}_k \Delta t_k s_{\theta_k} + \Delta t_k^2 \dot{\theta}_k^2 c_{\theta_k} + o(\dot{\theta}_k^2) - c_{\theta_k} \right) \\
&= \lim_{\dot{\theta}_k \to 0} \frac{\overline{v}_k}{\dot{\theta}_k^2} \left(\frac{\dot{\theta}_k^2 \Delta t_k^2}{2} c_{\theta_k} + \frac{\dot{\theta}_k^3 \Delta t_k^3}{2} s_{\theta_k} + o(\dot{\theta}_k^2) \right) \\
&= \frac{\overline{v}_k \Delta t_k^2}{2} c_{\theta_k}
\end{aligned}
\tag{B.13}
$$

On utilise ensuite la même méthode pour le cas de z_{k+1} :

$$\begin{aligned}
\lim_{\dot{\theta}_k \to 0} \frac{\partial z_{k+1}}{\partial \dot{\theta}_k} &= \lim_{\dot{\theta}_k \to 0} \frac{\overline{v}_k}{\dot{\theta}_k^2} \left(\Delta t_k \dot{\theta}_k c_{\alpha_k} + s_{\alpha_k} - s_{\theta_k} \right) \\
&= \lim_{\dot{\theta}_k \to 0} \frac{\overline{v}_k}{\dot{\theta}_k^2} \Big(\Delta t_k \dot{\theta}_k \left(c_{\theta_k} \cos\left(\dot{\theta}_k \Delta t_k\right) + s_{\theta_k} \sin\left(\dot{\theta}_k \Delta t_k\right) \right) \\
&\qquad + \left(s_{\theta_k} \cos\left(\dot{\theta}_k \Delta t_k\right) - c_{\theta_k} \sin\left(\dot{\theta}_k \Delta t_k\right) \right) - s_{\theta_k} \Big) \\
&= \lim_{\dot{\theta}_k \to 0} \frac{\overline{v}_k}{\dot{\theta}_k^2} \Big(\Delta t_k \dot{\theta}_k c_{\theta_k} \cos\left(\dot{\theta}_k \Delta t_k\right) + \Delta t_k \dot{\theta}_k s_{\theta_k} \sin\left(\dot{\theta}_k \Delta t_k\right) \\
&\qquad + s_{\theta_k} \cos\left(\dot{\theta}_k \Delta t_k\right) - c_{\theta_k} \sin\left(\dot{\theta}_k \Delta t_k\right) - s_{\theta_k} \Big) \\
&= \lim_{\dot{\theta}_k \to 0} \frac{\overline{v}_k}{\dot{\theta}_k^2} \Big(\left(\Delta t_k \dot{\theta}_k c_{\theta_k} + s_{\theta_k} \right) \cos\left(\dot{\theta}_k \Delta t_k\right) \\
&\quad + \left(\Delta t_k \dot{\theta}_k s_{\theta_k} - c_{\theta_k} \right) \sin\left(\dot{\theta}_k \Delta t_k\right) - s_{\theta_k} \Big) \\
&= \lim_{\dot{\theta}_k \to 0} \frac{\overline{v}_k}{\dot{\theta}_k^2} \Bigg(\left(\Delta t_k \dot{\theta}_k c_{\theta_k} + s_{\theta_k} \right) \left(1 - \frac{\dot{\theta}_k^2 \Delta t_k^2}{2} + o(\dot{\theta}_k^2) \right) \\
&\qquad + \left(\Delta t_k \dot{\theta}_k s_{\theta_k} - c_{\theta_k} \right) \left(\dot{\theta}_k \Delta t_k + o(\dot{\theta}_k^2) \right) - s_{\theta_k} \Bigg) \\
&= \lim_{\dot{\theta}_k \to 0} \frac{\overline{v}_k}{\dot{\theta}_k^2} \Bigg(\Delta t_k \dot{\theta}_k c_{\theta_k} + s_{\theta_k} - \frac{\dot{\theta}_k^3 \Delta t_k^3}{2} c_{\theta_k} - \frac{\dot{\theta}_k^2 \Delta t_k^2}{2} s_{\theta_k} + o(\dot{\theta}_k^2) \\
&\qquad + \Delta t_k^2 \dot{\theta}_k^2 s_{\theta_k} - \dot{\theta}_k \Delta t_k c_{\theta_k} + o(\dot{\theta}_k^2) - s_{\theta_k} \Bigg) \\
&= \lim_{\dot{\theta}_k \to 0} \frac{\overline{v}_k}{\dot{\theta}_k^2} \left(-\frac{\dot{\theta}_k^3 \Delta t_k^3}{2} c_{\theta_k} + \frac{\dot{\theta}_k^2 \Delta t_k^2}{2} s_{\theta_k} + o(\dot{\theta}_k^2) \right) \\
&= \frac{\overline{v}_k \Delta t_k^2}{2} s_{\theta_k}
\end{aligned} \tag{B.14}$$

B.2.4 Résultat final

En utilisant les résultats obtenus ci-dessus, on obtient alors la limite de la jacobienne suivante :

$$\lim_{\dot{\theta}_k \to 0} J_k^{fX} = \begin{bmatrix} 1 & 0 & -\overline{v}_k \Delta t_k c_{\theta_k} & -\Delta t_k s_{\theta_k} & \frac{\overline{v}_k \Delta t_k^2}{2} c_{\theta_k} \\ 0 & 1 & -\overline{v}_k \Delta t_k s_{\theta_k} & \Delta t_k c_{\theta_k} & \frac{\overline{v}_k \Delta t_k^2}{2} s_{\theta_k} \\ 0 & 0 & 1 & 0 & \Delta t_k \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.15}$$

Bibliographie

Araujo, H., Carceroni, R., and Brown, C. (1998). A fully projective formulation to improve the accuracy of lowe's pose-estimation algorithm. *Computer Vision and Image Understanding*, 70(2) :227–238.

Asada, H. and Brady, M. (1986). The curvature primal sketch. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (1) :2–14.

Bailey, T. and Durrant-Whyte, H. (2006). Simultaneous localization and mapping (slam) : Part ii. *Robotics & Automation Magazine, IEEE*, 13(3) :108–117.

Baker, S. and Matthews, I. (2004). Lucas-kanade 20 years on : A unifying framework. *International Journal of Computer Vision*, 56(3) :221–255.

Batcher, K. (1968). Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, pages 307–314. ACM.

Bay, H., Ess, A., Tuytelaars, T., and Van Gool, L. (2008). Surf : Speeded up robust features. *Computer Vision and Image Understanding*, 110(3) :346–359.

Beardsley, P., Torr, P., and Zisserman, A. (1996). 3d model acquisition from extended image sequences. *Computer Vision—ECCV'96*, pages 683–695.

Beaudet, P. (1978). Rotationally invariant image operators. In *Proceedings of the International Joint Conference on Pattern Recognition*, pages 579–583.

Becerra, H., Courbon, J., Mezouar, Y., and Sagues, C. (2010). Wheeled mobile robots navigation from a visual memory using wide field of view cameras. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 5693–5699. IEEE.

Becerra, H. and Sagues, C. (2008). A sliding mode control law for epipolar visual servoing of differential-drive robots. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 3058–3063. IEEE.

Berger, C. and Lacroix, S. (2008). Using planar facets for stereovision SLAM. In *IROS*, pages 1606–1611.

Booij, O., Terwijn, B., Zivkovic, Z., and Krose, B. (2007). Navigation using an appearance based topological map. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3927–3932. IEEE.

Bétaille, D., Peyraud, S., Renault, S., Ortiz, M., Meizel, D., and Peyret, F. (2012). Using road constraints to progress in real time nlos detection. In *IEEE Intelligent Vehicles Symposium(IV 2012)*, Alcala de Henares, (Spain).

Calonder, M., Lepetit, V., Strecha, C., and Fua, P. (2010). Brief : Binary robust independent elementary features. *Computer Vision–ECCV 2010*, pages 778–792.

Chaumette, F. and Hutchinson, S. (2006). Visual servo control. i. basic approaches. *Robotics & Automation Magazine, IEEE*, 13(4) :82–90.

Cherubini, A. and Chaumette, F. (2009). Visual navigation with a time-independent varying reference. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 5968–5973. IEEE.

Cherubini, A. and Chaumette, F. (2011). Visual navigation with obstacle avoidance. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 1593–1598. IEEE.

Cherubini, A., Colafrancesco, M., Oriolo, G., Freda, L., and Chaumette, F. (2009). Comparing appearance-based controllers for nonholonomic navigation from a visual memory. In *ICRA 2009 Workshop on safe navigation in open and dynamic environments : application to autonomous vehicles*, Kobe, Japan, Japon.

Cobzas, D., Zhang, H., and Jagersand, M. (2003). Image-based localization with depth-enhanced image map. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 2, pages 1570–1575. IEEE.

Cornelis, N. and Van Gool, L. (2008). Fast scale invariant feature detection and matching on programmable graphics hardware. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, 2008. CVPR Workshops 2008*, pages 1–8.

Courbon, J., Mezouar, Y., and Martinet, P. (2009). Autonomous navigation of vehicles from a visual memory using a generic camera model. *Intelligent Transportation Systems, IEEE Transactions on*, 10(3) :392–402.

Craciun, D., Paparoditis, N., and Schmitt, F. (2010). Multi-view scans alignment for 3d spherical mosaicing in large-scale unstructured environments. *Computer Vision and Image Understanding*, 114(11) :1248–1263.

Davison, A., Reid, I., Molton, N., and Stasse, O. (2007). Monoslam : Real-time single camera slam. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6) :1052–1067.

Deriche, R. and Giraudon, G. (1993). A computational approach for corner and vertex detection. *International journal of computer vision*, 10(2) :101–124.

Dinesh, R. and Guru, D. (2004). Mathematical morphology based corner detection scheme : a non-parametric approach. *Kolkata, India, December*.

Diosi, A., Remazeilles, A., Segvic, S., and Chaumette, F. (2007). Outdoor visual path following experiments. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 4265–4270. IEEE.

Dreschler, L. and Nagel, H. (1982). Volumetric model and 3d trajectory of a moving car derived from monocular tv frame sequences of a street scene. *Computer Graphics and Image Processing*, 20(3) :199–228.

Durrant-Whyte, H. and Bailey, T. (2006). Simultaneous localization and mapping : part i. *Robotics & Automation Magazine, IEEE*, 13(2) :99–110.

Fischler, M. and Bolles, R. (1981). Random sample consensus : a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6) :381–395.

Goedeme, T., Nuttin, M., Tuytelaars, T., and Van Gool, L. (2007). Omnidirectional vision based topological navigation. *International Journal of Computer Vision*, 74(3) :219–236.

Haralick, B., Lee, C., Ottenberg, K., and Nolle, M. (1994). Review and analysis of solutions of the three point perspective pose estimation problem. *International Journal of Computer Vision*, 13(3) :331–356.

Harris, C. and Stephens, M. (1988). A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK.

Harris, M. (2007). Optimizing parallel reduction in cuda. *NVIDIA Developer Technology*.

Hartley, R. I. and Zisserman, A. (2004). *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN : 0521540518, second edition.

Heymann, S., Muller, K., Smolic, A., Frohlich, B., and Wiegand, T. (2007). Sift implementation and optimization for general-purpose gpu. In *Proceedings of the International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*.

Hu, M. (1962). Visual pattern recognition by moment invariants. *Information Theory, IRE Transactions on*, 8(2) :179–187.

Kalman, R. (1960). A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(Series D) :35–45.

Kidono, K., Miura, J., and Shirai, Y. (2002). Autonomous visual navigation of a mobile robot using a human-guided experience. *Robotics and Autonomous Systems*, 40(2) :121–130.

Kitchen, L. and Rosenfeld, A. (1982). Gray-level corner detection. *Pattern Recognition Letters*, 1(2) :95–102.

Koser, K. and Koch, R. (2007). Perspectively invariant normal features. In *ICCV 2007*, pages 1–8.

Lébraly, P. (2012). *Étalonnage de cameras à champs disjoints et reconstruction 3D - Application à un robot mobile*. PhD thesis, Institut Pascal (CNRS - Université Blaise Pascal). Jury : Omar Ait-Aider, François Chaumette, Michel Dhome, Richard Hartley, Eric Royer, Peter Sturm.

Lébraly, P., Clément, D., Ait-Aider, O., Royer, E., and Dhome, M. (IROS 2010). Flexible Extrinsic Calibration of Non-Overlapping Cameras Using a Planar Mirror : Application to Vision-Based Robotics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Lenain, R., Thuilot, B., Cariou, C., and Martinet, P. (2005). Model predictive control for vehicle guidance in presence of sliding : application to farm vehicles path tracking. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 885–890. IEEE.

Lepetit, V., Lagger, P., and Fua, P. (2005). Randomized trees for real-time keypoint recognition. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 775–781. IEEE.

Lhuillier, M. and Quan, L. (2005). A quasi-dense approach to surface reconstruction from uncalibrated images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(3) :418–433.

Li, S. and Tsuji, S. (1999). Qualitative representation of scenes along route. *Image and vision computing*, 17(9) :685–700.

Lopez-Nicolas, G., Bhattacharya, S., Guerrero, J., Sagues, C., and Hutchinson, S. (2007). Switched homography-based visual control of differential drive vehicles with field-of-view constraints. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 4238–4244. IEEE.

Lowe, D. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2) :91–110.

Marquez-Gamez, D. and Devy, M. (2012). SLAM visuel avec détection et suivi d'objets mobiles par une approche de segmentation/classification. In *Actes de la conférence RFIA 2012*, pages 978–2–9539515–2–3, Lyon, France. Session "Posters".

Mei, C., Sibley, G., Cummins, M., Newman, P., and Reid, I. (2011). Rslam : A system for large-scale mapping in constant-time using stereo. *International journal of computer vision*, 94(2) :198–214.

Meilland, M., Comport, A., and Rives, P. (2011). Dense visual mapping of large scale environments for real-time localisation. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 4242–4248. IEEE.

Mikolajczyk, K. and Schmid, C. (2001). Indexing based on scale invariant interest points. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 1, pages 525–531. IEEE.

Mikolajczyk, K. and Schmid, C. (2003). A performance evaluation of local descriptors. In *CVPR*.

Mikolajczyk, K. and Schmid, C. (2004). Scale & affine invariant interest point detectors. *International journal of computer vision*, 60(1) :63–86.

Molton, N., Davison, A., and Reid, I. (2004). Locally planar patch features for real-time structure from motion. In *BMVC*.

Moravec, H. (1977). Towards automatic visual obstacle avoidance. In *IJCAI*.

Morel, J. and Yu, G. (2009). Asift : A new framework for fully affine invariant image comparison. *SIAM Journal on Imaging Sciences*, 2(2) :438–469.

Mourllion, B., Gruyer, D., Lambert, A., and Glaser, S. (2005). Kalman filters predictive steps comparison for vehicle localization. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 565–571. IEEE.

Muhammad, N., Fofi, D., and Ainouz, S. (2009). Current state of the art of vision based slam. In *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 7251, page 14.

Nistér, D. (2000). Reconstruction from uncalibrated sequences with a hierarchy of trifocal tensors. *Computer Vision-ECCV 2000*, pages 649–663.

Nistér, D. (2004). An efficient solution to the five-point relative pose problem. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(6) :756–770.

Nvidia (2011). Nvidia cuda c programming guide v4.1. *Nvidia Corp*.

Owens, J., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A., and Purcell, T. (2007). A survey of general-purpose computation on graphics hardware. In *Computer graphics forum*, volume 26, pages 80–113. Wiley Online Library.

Pietzsch, T. (2008). Planar features for visual slam. *KI 2008 : Advances in Artificial Intelligence*, pages 119–126.

Pinies, P. and Tardos, J. (2008). Large-scale slam building conditionally independent local maps : Application to monocular vision. *Robotics, IEEE Transactions on*, 24(5) :1094–1106.

Rezaei, S. and Sengupta, R. (2007). Kalman filter-based integration of dgps and vehicle sensors for localization. *Control Systems Technology, IEEE Transactions on*, 15(6) :1080–1088.

Rosten, E., Porter, R., and Drummond, T. (2010). Faster and better : A machine learning approach to corner detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(1) :105–119.

Royer, E. (2006). *Cartographie 3D et localisation par vision monoculaire pour la navigation autonome d'un robot mobile*. PhD thesis, Université Blaise Pascal - Clermont-Ferrand II.

Royer, E., Lhuillier, M., Dhome, M., and Lavest, J. (2007). Monocular vision for mobile robot localization and autonomous navigation. *International Journal of Computer Vision*, 74(3) :237–260.

Schiele, B. and Waibel, A. (1995). Gaze tracking based on face-color. In *In International Workshop on Automatic Face-and Gesture-Recognition*. Citeseer.

Schmid, C., Mohr, R., and Bauckhage, C. (1998). Comparing and evaluating interest points. In *Computer Vision, 1998. Sixth International Conference on*, pages 230–235. IEEE.

Schweitzer, M. and Wuensche, H.-J. (2009). Efficient Keypoint Matching for Robot Vision using GPUs. In *Fifth IEEE Workshop on Embedded Computer Vision (ECV'09)*.

Se, S., Lowe, D., and Little, J. (2002). Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks. *The international Journal of robotics Research*, 21(8) :735–758.

Segvic, S., Remazeilles, A., Diosi, A., and Chaumette, F. (2009). A mapping and localization framework for scalable appearance-based navigation. *Computer Vision and Image Understanding*, 113(2) :172–187.

Sinha, S., Frahm, J., Pollefeys, M., and Genc, Y. (2006). GPU-based video feature tracking and matching. In *EDGE, Workshop on Edge Computing Using New Commodity Architectures*, volume 278. Citeseer.

Strasdat, H., Montiel, J., and Davison, A. (2010). Real-time monocular slam : Why filter ? In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2657–2664. IEEE.

Tola, E., Lepetit, V., and Fua, P. (2008). A fast local descriptor for dense matching. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. Ieee.

Triggs, B., McLauchlan, P., Hartley, R., and Fitzgibbon, A. (2000). Bundle adjustment—a modern synthesis. *Vision algorithms : theory and practice*, pages 153–177.

Vivet, D. (2011). *Perception de l'environnement par radar hyperfréquence. Application à la localisation et la cartographie simultanées, à la détection et au suivi d'objets mobiles en milieu extérieur.* These, Université Blaise Pascal - Clermont-Ferrand II.

Weiss, S., Scaramuzza, D., and Siegwart, R. (2011). Monocular-slam–based navigation for autonomous micro helicopters in gps-denied environments. *Journal of Field Robotics*, 28(6) :854–874.

Williams, B., Klein, G., and Reid, I. (2007). Real-time slam relocalisation. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE.

Wu, C., Clipp, B., Li, X., Frahm, J., and Pollefeys, M. (2008). 3D model matching with Viewpoint-Invariant Patches (VIP). In *CVPR 2008*, pages 1–8.

Zhang, X. and Zhao, D. (1995). Morphological algorithm for detecting dominant points on digital curves. In *Proceedings of SPIE*, volume 2424, page 372.

Zuniga, O. and Haralick, R. (1983). Corner detection using the facet model. In *Proc. Conf. Computer Vision and Pattern Recognition*, pages 30–37. Washington, DC.