Vision stéréoscopique temps-réel pour la navigation autonome d'un robot en environnement dynamique

Maxime Derome

NNT : 2017SACLS156

Vision stéréoscopique temps-réel pour la navigation autonome d'un robot en environnement dynamique

Thèse de doctorat de l'Université Paris-Saclay préparée à l'Université Paris-Sud

École doctorale n°580 Sciences et technologies de l'information et de la communication (STIC)

Spécialité de doctorat : Mathématiques et informatique

Thèse présentée et soutenue à Palaiseau, le 22 juin 2017, par

Maxime Derome

Composition du Jury :

Fabien Moutarde Professeur, Mines ParisTech	Président
Michel Devy Directeur de Recherche CNRS, LAAS Toulouse	Rapporteur
Vincent Frémont Maître de Conférences HDR, UTC Compiègne	Rapporteur
Samia Bouchafa-Bruneau Professeur des Universités, Université d'Evry	Examinatrice
Guy Le Besnerais Directeur de Recherche, ONERA Palaiseau (DTIS)	Directeur de thèse
Aurélien Plyer Ingénieur de Recherche, ONERA Palaiseau (DTIS)	Encadrant (Invité)
Martial Sanfourche Ingénieur de Recherche, ONERA Palaiseau (DTIS)	Encadrant (Hors-jury)

Thèse de doctorat

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Vision stéréoscopique temps-réel pour la navigation autonome d'un robot en environnement dynamique

Mots clés : vision stéréoscopique, robotique, flot optique, flot de scène, détection d'objets mobiles, temps-réel, GPU

Résumé : L'objectif de la thèse est de concevoir un système de perception stéréoscopique embarqué, permettant une navigation robotique autonome en présence d'objets mobiles.
Pour pouvoir naviguer en environnement inconnu et en présence d'objets mobiles quelconques, nous avons adopté une approche purement géométrique reposant sur l'estimation de la disparité stéréo, du flot optique et du flot de scène. Pour assurer une couverture maximale du champ visuel nous avons employé des méthodes d'estimation denses qui traitent chaque pixel de l'image. Enfin, puisque les algorithmes utilisés doivent s'exécuter en embarqué sur un robot, nous nous sommes efforcé de sélectionner ou concevoir des algorithmes rapides, pour nuire au minimum à la réactivité du système.
Cette thèse présente plusieurs contributions. Nous avons proposé une nouvelle approche pour l'estimation du flot de scène, en combinant des algorithmes d'odométrie visuelle, d'appariement stéréo et de flot optique. Son implémentation sur GPU permet une estimation du flot de scène à cadence vidéo sur les images du dataset KITTI. Une méthode de détection dense d'objets mobiles a aussi été proposée, en modélisant statistiquement et en propageant toutes les erreurs de mesures.
Enfn nous avons validé expériementalement sur un petit robot terrestre ces alogrithmes de perception, en les couplant à un module de commande prédictive. L'expérience montre que le système de perception proposé est suffisamment rapide pour permettre au robot de détecter un objet mobile et d'adapter à la volée sa trajectoire afin d'éviter une collision.

Title : Real-time stereovision for autonomous robot navigation in dynamic environment

Keywords : stereovision, robotics, optical flow, scene flow, mobile object detection, real-time, GPU

Résumé This thesis aims at designing an embedded stereoscopic perception system that enables autonomous robot navigation in dynamic environment (with mobile objects).
To be able to navigate in unknown environments and among any type of mobile objects, we adopt a geometric approach based on the estimation of the stereo disparity, the optical flow and the scene flow.
To ensure the best possible coverage of the field of view, we employ dense methods that process every pixel in the image. Finally, since the algorithms must be compliant with an embedded platform, we careful choose and design the algorithms so they are fast enough to keep a certain level of reactivity.
This thesis presents several contributions. We have proposed a new approach for the scene flow estimation, by combining visual odometry, stereo matching and optical flow algorithms. Its implementation on GPU allows video rate processing on KITTI dataset images. We also propose a new dense method for mobile object detection, by modeling and propagating every measurement error.
Finally, we have experimentally validated these perception algorithms with a small robot, by coupling them with a model predictive control algorithm. The experiment shows that the proposed perception system is fast enough to allow the robot to detect a mobile object and correct its trajectory in order to avoid the collision.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

Remerciements

Je tiens d'abord à remercier l'Office National d'Etudes et de Recherches Aérospatiales (ONERA) ainsi la Direction Générale de l'Armement (DGA) du ministère de la défense, qui ont financé cette thèse.

Le bon déroulement de cette thèse est en grande partie dû à mon directeur de thèse Guy et mes encadrants Martial et Aurélien que je remercie chaleureusement pour m'avoir soutenu à la fois sur le plan technique, scientifique et humain tout au long de ma thèse.

Merci à tous mes collègues du département DTIS (ex-DTIM), et notament à Françoise pour l'animation matinale en pause café, à Fabrice pour le coaching sportif avec Elise, Thibaut et Sémy. Merci aux apprentis, doctorants et post-docs qui ont contribué à la bonne ambiance du labo : Elyse, Flora, Hélène, Isabelle, Marcella, Alexandre, Anthelme, Calum, David, David, Guillaume, Hicham, Joris, Maxime, Maxime, Nicolas, Riadh, Robin, Semy et Thibaut. Merci également aux stagiaires dont Robin qui fut mon premier co-bureau.

Enfin, merci à mes proches pour leur soutien : à mes amis, à ma famille et à kedvesem.

Table des matières

Chapitre 1

Introduction

Sommaire

1.1 Contexte : l'émergence de la robotique autonome

1.1.1 Explosion de la robotique et de ses applications

Le domaine de la robotique est en plein essor, comme le montre l'engouement du grand public pour les drones tels que le modèle AR.Drone de Parrot qui s'est vendu à plus de 700 000 exemplaires [www.leparisien.fr, 2014]. Les drones faisant désormais partie du paysage, la législation française s'est même adaptée et depuis l'arrêté du 11 avril 2012, certains types de drones civils sont autorisés dans l'espace aérien français. Cependant l'utilisation des drones ne se restreint pas au grand public, et concerne aussi des entreprises telles que Dronotec qui propose les services de télé-pilotes de drones à des fins d'inspections techniques de bâtiments et d'expertise après sinistre [www.dronotec.com, 2015], ou EDF qui emploie des drones pour la maintenance de lignes haute tension dont l'accès peut être difficile [Yacou, 2013].

De manière générale, la robotique trouve des applications dans le milieu industriel, dès qu'il s'agit d'inspection ou de maintenance en environnement dangereux ou difficilement accessible pour un opérateur humain. Ces applications bénéficieraient de systèmes robotiques capables de naviguer de façon autonome, par exemple pour couvrir des zones plus grandes et en continu en utilisant un plus grand nombre de robots simultanément. Un autre avantage de la navigation robotique autonome, serait de servir de mode de secours en cas de perte de contrôle ou de communication avec le télé-pilote. Ceci pourrait éviter des incidents comme celui du 22 décembre

2015, où un drone téléopéré filmant l'épreuve de slalom de la coupe du monde de ski alpin s'est écrasé juste derrière le skieur Marcel Hirscher [Duran, 2015] comme illustré Figure 1.1.

FIGURE 1.1 – Crash du drone filmant la descente du skieur Marcel Hirscher (cf. article [Duran, 2015] dont est extraite cette image).

1.1.2 Des systèmes robotiques autonomes matures ?

Des systèmes robotiques autonomes existent déjà, et ont atteint un degré de maturité plus ou moins élevé. Par exemple en domotique : les robots aspirateurs existent depuis plus de 10 ans sur le marché, et sont capables de nettoyer tout un étage en naviguant de façon autonome tout en évitant les obstacles. Ces robots sont dotés de capteurs à ultrasons pour détecter les obstacles à proximité, et naviguent de manière aléatoire ou avec une trajectoire pré-calculée pour ceux qui sont capables de se géolocaliser et de mémoriser les obstacles détectés aux précédents passages. Cependant leur utilisation est limitée à un environnement intérieur relativement simple voire connu (pour ceux dotés de fonction de mémorisation d'obstacle), et leur perception se limite souvent aux capteurs à ultrasons dont la portée maximum est typiquement de quelques mètres. Plus impressionnant, dans le domaine de la voiture autonome : la Tesla modèle S intègre de nombreux capteurs (à ultrasons, radar, caméra frontale), et propose un mode en auto-pilote pour une navigation autonome sur autoroute sans que le conducteur n'ait besoin d'avoir les mains sur le volant. La Tesla a déjà fait ses preuves avec plus de 200 millions de km parcourus en mode auto-pilote sur l'ensemble des Tesla modèle S en circulation, avec un seul accident mortel,

survenu en mode de navigation autonome [www.theguardian.com, 2016]. Cependant le mode d'auto-pilote de la Tesla ne fonctionne que sur autoroute, donc encore une fois en environnement très structuré : sur route, avec marquage au sol et signalisation, et les seuls objets mobiles présents sont les autres voitures, camions ou motos. Un tel système ne convient donc pas pour la navigation autonome de n'importe quelle plate-forme robotique (terrestre ou aérienne) en environnement non structuré et en présence de tout type d'objets mobiles (piétons, véhicules, animaux, robots). Un autre exemple de voiture autonome est la *Google Car* qui permet une conduite en ville, sans conducteur. L'utilisation de ses véhicules est autorisée dans le Nevada sous certaines conditions, et sur les 500 000 km effectués en 2013 avec des google cars (dont 1600 km en totale autonomie), seuls 2 accidents légers ont été notés [Juhan, 2013]. Cependant ces voitures nécessitent une connaissance de l'environnement urbain dans lequel elles évoluent, et les trajets doivent être préalablement effectués de manière classique avec un conducteur aux commandes [Juhan, 2013]. Une telle approche n'est donc pas utilisable dans un scénario d'exploration et de navigation en environnement inconnu.

1.2 Choix du capteur pour la perception d'environnement dynamique : banc stéréo

1.2.1 Avantages et inconvénients du banc stéréo

Pour naviguer de façon autonome, un robot doit pouvoir percevoir son environnement, ce qui requiert d'utiliser un ou des capteurs extéroceptifs. Parmi la variété de capteurs extéroceptifs existants (capteurs à ultrasons, radars, laser, kinect, caméra,...), c'est sur l'emploi d'un banc stéréo que se focalisent nos travaux. Un banc stéréo est constitué de deux caméras fixées à une tige rigide de sorte à avoir des champs de vue recouvrants, et comporte plusieurs avantages :

- c'est un capteur léger et peu coûteux, et qui convient donc à des plate-formes type micro-drone
- c'est un capteur passif (qui ne projette pas d'onde lumineuse ou sonore sur l'environnement pour fonctionner) qui est donc adapté pour des missions de surveillance ou d'exploration furtive
- il donne accès à la fois à une information géométrique (carte de profondeur), et à l'apparence des éléments présents dans l'environnement
- il permet une égo-localisation en estimant le déplacement du capteur par odométrie visuelle
- il fonctionne en environnement intérieur et extérieur.

Toutefois l'emploi d'un banc stéréo comporte aussi des inconvénients. Comme tous les systèmes de vision passive, il dépend des contrastes visibles dans la scène ce qui impose des conditions d'emploi (jour, pas de fumée, pas de brouillard). Enfin sa portée pratique est limitée car son utilisation repose sur le principe de triangulation stéréo dont la précision devient très faible pour des objets distants. Même si

le capteur stéréo n'est pas suffisant en toutes situations, il permet d'accéder à une information suffisante pour la naviguation visuelle en milieu encombré dans de nombreux cas (c'est d'ailleurs la raison pour laquelle de nombreuses espèces animales l'ont adopté) et son étude est donc pertinente.

1.2.2 Les verroux techniques liés au banc stéréo

La vision stéréoscopique a fait l'objet de nombreux travaux, et chaque année des méthodes plus précises apparaissent et figurent en tête des classements d'algorithmes de stéréovision comme KITTI [Geiger et al., 2012] (voir détail à la Section 2.3) et Middlebury [Scharstein and Szeliski, 2002]. Cependant, la plupart des efforts de recherche en vision stéréo (et en vision « monoculaire » également) porte sur l'amélioration de la précision et non sur l'accélération de la vitesse de calcul des algorithmes. Cela se manifeste sur les benchmarks publics, par des temps de calcul annoncés atteignant parfois plusieurs dizaines de minutes pour estimer une carte de profondeur (cf. tableau KITTI 2012 Stéréo), ce qui n'est absolument pas compatible avec une exécution sur un système robotique.

1.3 Objectifs et démarche de cette thèse

1.3.1 Objectifs de cette thèse

Dans cette thèse nous souhaitons concevoir un système de perception stéréo capable de fonctionner sur un robot pour lui permettre de naviguer de façon autonome en environnement dynamique (c'est-à-dire incluant une scène non statique comprenant des objets en mouvement).
Afin d'assurer la meilleure couverture possible de l'ensemble du champ visuel, nous travaillons sur des **algorithmes de vision stéréo dense**, c'est-à-dire des algorithmes dont l'estimation porte sur chaque pixel de l'image.
Puisque l'on souhaite détecter tout type d'objets (e.g. des piétons, véhicules, mais aussi des robots, animaux, ou projectiles de taille suffisamment grosse), nous avons choisi d'adopter une **approche purement géométrique** en proposant un système de perception en environnement dynamique fondé sur l'estimation d'attributs tels que des positions et vitesses en 3D. Ceci, par opposition à une approche sémantique qui consisterait à utiliser des détecteurs d'objets (piétons, cycliste, voiture, etc.) faisant appel à des méthodes d'apprentissage (SVM, réseaux de neurones, etc.) pour détecter et modéliser les objets mobiles perçus dans la scène observée. Cependant rien n'empêche de combiner ces deux approches par exemple pour apprendre à reconnaître un objet quelconque en mouvement et le suivre dans le temps, ou bien en utilisant l'approche géométrique d'abord pour fournir une prédétection à des algorithmes de reconnaissance d'objets.
D'autre part, nous voulons concevoir un système qui puisse être utilisé en **conditions réelles**, par exemple en **milieu urbain**. C'est pourquoi nous évaluons les performances des algorithmes sur le benchmark KITTI avec des images acquises à

partir d'un banc stéréo fixé sur le toit d'une voiture qui se déplace en milieu urbain. Nous souhaitons aussi tester nos algorithmes sur des plate-formes robotiques plus petites comme celles disponibles au sein du laboratoire AtExPA (Atelier Expérimental pour la Perception Autonome) de l'ONERA/DTIM, notamment sur le robot DORA (cf. Figure 1.2) qui possède un ordinateur pour effectuer tous les traitements de manière embarquée.

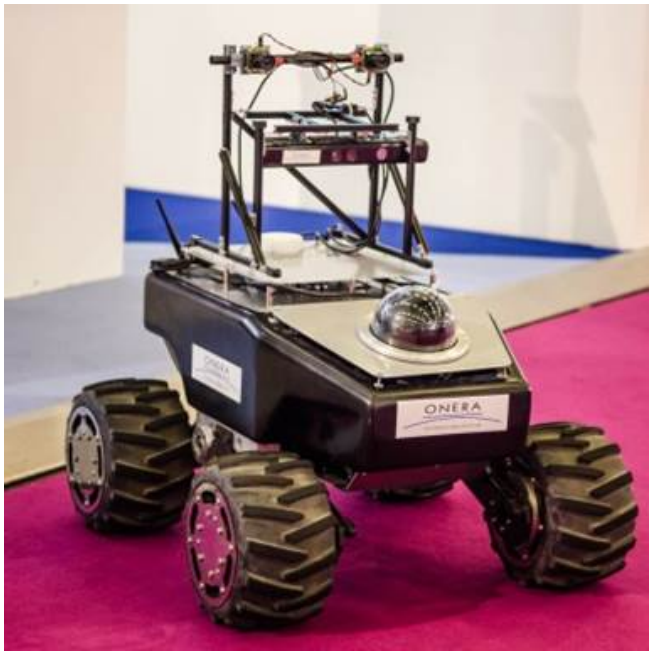

FIGURE 1.2 – Le robot DORA que nous utilisons est une plateforme Summit XL ©Robotnik, qui intègre une carte graphique Nividia GT640 et un processeur Intel quad-core i7. Le banc stéréo doté d'une baseline de 18cm, est configuré pour enregistrer des images VGA à 20Hz.

Enfin nous attachons dans cette thèse une très grande importance à la vitesse d'exécution de nos algorithmes dont le temps de calcul doit rester suffisamment faible pour autoriser une **utilisation embarquée "temps-réel"** (c'est-à-dire à cadence vidéo, ou du moins à plusieurs fps « frame-per-second »). La réactivité du système robotique dépend de la vitesse de nos algorithmes de perception, et pour éviter une collision avec d'éventuels obstacles mobiles, ceux-ci doivent être détectés suffisamment rapidement.
L'objectif de ma thèse est donc de concevoir et de valider expérimentalement des algorithmes permettant à un robot équipé d'un banc stéréo, de fournir en temps-réel une représentation 3D statique et dynamique de l'environnement dans lequel il évolue.

1.3.2 Organisation du manuscrit de thèse

Ce manuscrit est organisé de la manière suivante : tout d'abord nous présentons au premier chapitre le modèle de capteur stéréo utilisé, ainsi que les principaux algorithmes permettant d'obtenir une carte de profondeur à partir des images enregistrées par les caméras gauche et droite du banc stéréo. Nous voyons donc dans le **Chapitre 2 comment estimer la position en 3D et par rapport au banc stéréo, des points observés à l'instant t**. Cette information statique n'est pas suffisante pour percevoir et modéliser un environnement dynamique comportant des éléments mobiles. Pour cela il faut également pouvoir estimer le mouvement entre deux instants d'acquisition, des points observés et du banc stéréo. C'est l'objet du **Chapitre 3 qui traite de l'estimation du mouvement : de la caméra, des pixels dans l'image, et des points en 3D**.À partir des informations statiques et dynamiques acquises avec les méthodes décrites dans les deux premiers chapitres, on peut déduire le mouvement apparent induit par le déplacement de la caméra, et le comparer au mouvement estimé pour chaque pixel de l'image. Si pour un pixel, il y a une différence entre ces deux mouvements, alors soit ce pixel appartient à un objet mobile, soit cet écart est dû à des erreurs d'estimation. Pour pouvoir prendre une décision et considérer ou non qu'un point appartient à un objet mobile, il faut donc au préalable caractériser ces erreurs d'estimation qui interviennent tout au long de la chaîne de traitement. **Le début du Chapitre 4 aborde la question de la modélisation et de la propagation des erreurs d'estimation**. Une fois que l'on sait modéliser et propager ces incertitudes on peut adopter une approche statistique et, pour un critère bien choisi, on peut effectuer un test du χ^2 pour déterminer si un pixel respecte ou non l'hypothèse d'une scène statique. En pratique plusieurs critères peuvent être utilisés pour détecter si un pixel appartient à un élément dynamique. **La suite du Chapitre 4 compare l'utilisation de différents critères pour la détection dense d'objets mobiles à partir d'un banc stéréo en mouvement**. La chaîne de traitement présentée dans les chapitres 2 à 4, a pour vocation de s'exécuter sur un robot, en temps-réel. Pour tourner à une vitesse proche de la cadence vidéo, les algorithmes massivement parallèles utilisés dans notre chaîne de traitement, sont programmés sur carte graphique (GPU). **Le Chapitre 5 décrit l'implémentation sur GPU de nos algorithmes temps-réel ainsi qu'une validation expérimentale du système de perception embarquée, pour la navigation autonome d'un robot**.

Chapitre 2

Perception 3D à Partir d'un Banc Stéréoscopique

Sommaire

L'objectif de ce chapitre est de présenter les notions de base sur la stéréoscopie, et un ensemble de méthodes d'appariement stéréo dense pour sélectionner une méthode efficace et compatible avec une utilisation embarquée en temps-réel. Ce chapitre n'a pas pour objectif de lister de façon exhaustive toutes les méthodes de stéréo, mais présente un échantillon de méthodes représentatif des approches couramment utilisées.

2.1 Modélisation du banc stéréo et triangulation

2.1.1 Description d'un banc stéréo

Un banc stéréoscopique est constitué de deux caméras synchronisées, fixées sur une tige rigide, qui permettent d'acquérir à chaque instant des images de la même scène selon deux points de vue. Dans notre cas, nous utilisons un banc stéréo horizontal avec une caméra gauche et une caméra droite séparées d'une *baseline* (i.e. la distance entre les deux caméras) mesurant typiquement entre 15cm et 20cm. Les caméras que nous utilisons (comme celles présentées Figure 2.1) disposent d'un *global shutter*, d'une focale de 4mm et d'une définition 640x480 pixels.

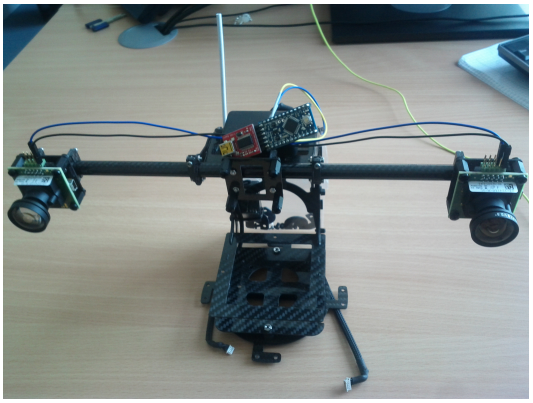

FIGURE 2.1 – Exemple de banc stéréo utilisé sur les plates-formes robotiques du laboratoire AtExPA (ONERA/DTIM).

Si l'on parvient à effectuer un appariement stéréo, en trouvant la position dans l'image droite d'un point repéré dans l'image gauche, alors on peut estimer les coordonnées 3D de ce point par triangulation (cf. sous-section 2.1.3). Pour cela, il faut connaître le modèle de distorsion des lentilles (cf. sous-section 2.1.2), ainsi que la position relative des deux caméras (cf. sous-section 2.1.3). Définissons d'abord les conventions et le modèle de formation d'image considérés.

2.1.2 Convention de repère et modèle de formation de l'image

Convention de repère

Parmi les deux caméras qui constituent le banc stéréo, nous choisissons celle de gauche comme référentiel. Nous considérons le repère caméra comme étant le repère orthogonal direct dont l'origine est le centre optique de la caméra et les trois axes répondent à la convention suivante :

- l'axe z est colinéaire à l'axe optique et orienté vers la scène observée ;
- l'axe x (respectivement y) est colinéaire à l'axe des lignes (respectivement des colonnes) de l'image.

Le repère caméra est illustré dans la Figure 2.2 ci-dessous.

Modèle de formation de l'image

Nous considérons comme modèle de formation d'image, le modèle sténopé ou "trou d'épingle" (*pinhole*). Dans ce modèle, la caméra est schématiquement réduite à son centre optique, et l'image est formée par la projection dans le plan focal image (situé à une distance f du centre optique) des rayons de la scène observée (cf. Figure 2.3). Ainsi un point 3D de coordonnées (X, Y, Z) observé par la caméra a pour projection dans le plan focal image, le point :

$$(X_{nd}, Y_{nd}, f) = f\left(\frac{X}{Z}, \frac{Y}{Z}, 1\right) \tag{2.1}$$

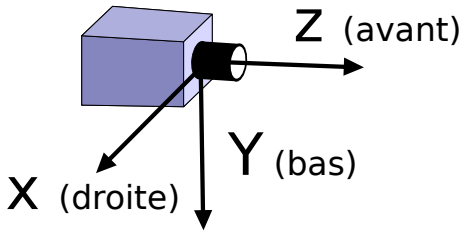

FIGURE 2.2 – Repère caméra défini selon notre convention "image"

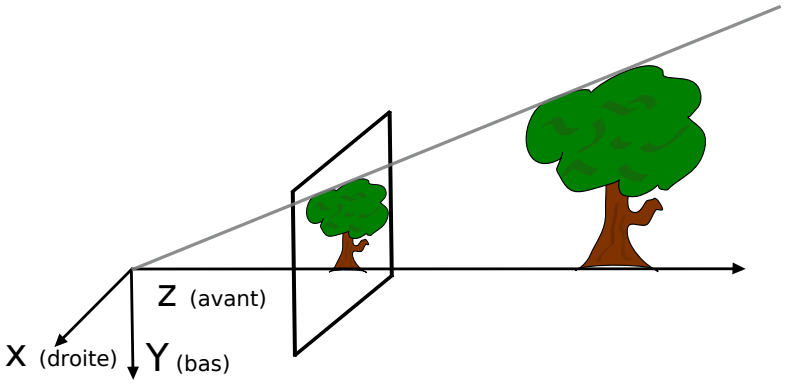

FIGURE 2.3 – Modèle sténopé ou "trou d'épingle" (*pinhole*) de formation d'image

Correction de la distorsion

Étant donnée que la lentille de la caméra induit une distorsion de l'image (cf. Figure 2.4), il est nécessaire de modéliser celle-ci. On suppose dans notre cas que les objectifs utilisés sur les caméras ne causent pas de distorsions trop importantes, et donc qu'on peut les représenter avec le modèle de Brown [Brown, 1966]. On modélise donc les distorsions par des polynômes en X_{nd} et Y_{nd} :

$$X_d = (1 + k_1\rho_{nd}^2 + k_2\rho_{nd}^4 + k_5\rho_{nd}^6)X_{nd} + 2k_3X_{nd}Y_{nd} + k_4(\rho_{nd}^2 + X_{nd}^2) \tag{2.2}$$

$$Y_d = (1 + k_1\rho_{nd}^2 + k_2\rho_{nd}^4 + k_5\rho_{nd}^6)Y_{nd} + 2k_4X_{nd}Y_{nd} + k_3(\rho_{nd}^2 + Y_{nd}^2) \tag{2.3}$$

où

$$\rho_{nd} = \sqrt{X_{nd}^2 + Y_{nd}^2} \tag{2.4}$$

Du capteur aux pixels de l'image

La caméra enregistre l'image sous la forme d'une matrice de pixels grâce à un capteur CCD ou CMOS contenant une grille de cellules photosensibles (pas nécessairement carrées). Ainsi, pour obtenir les coordonnées pixelliques (x, y) dans l'image,

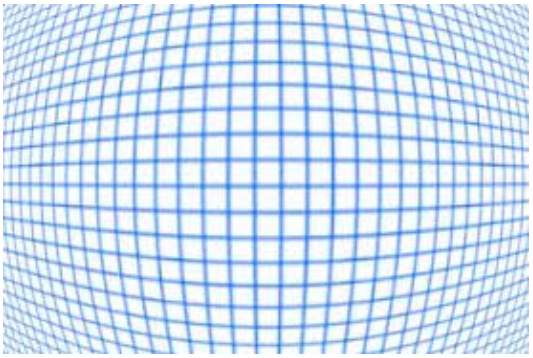

FIGURE 2.4 – Illustration du phénomène de distorsion de la lentille : lorsque la lentille est affectée d'un distorsion dite "en barillet", l'image d'une grille régulière est déformée selon le motif ci-dessus. Ce type de déformation peut se modéliser par les équations polynômiales introduites dans le texte.

il reste à appliquer une transformation affine :

$$x = \alpha X_d + x_0 \tag{2.5}$$

$$y = \beta Y_d + y_0 \tag{2.6}$$

où (x_0, y_0) correspond aux coordonnées dans l'image du point principal, i.e. du point d'intersection de l'axe optique avec le capteur CCD ou CMOS.

Les paramètres $\{k_{i\in[1..5]}, \alpha, \beta, x_0, y_0\}$ sont appelées paramètres intrinsèques de la caméra. En pratique on les estime une fois pour toute en effectuant une calibration de la caméra, à partir d'une série de vues d'une mire damier dont on connaît les dimensions (cf. [Tsai, 1986]).

2.1.3 Rectification et triangulation stéréo

Rectification stéréo

L'intérêt d'un banc stéréo est de pouvoir trianguler les points observés dans l'image gauche et l'image droite. Pour cela il faut d'abord effectuer les appariements stéréo, en cherchant pour un point dans l'image gauche, la position du point homologue dans l'image droite. Dans le cas idéal où les repères caméras diffèrent uniquement d'une translation selon l'axe X, le correspondant d'un pixel de l'image gauche se trouve le long de la même ligne dans l'image droite. En pratique les caméras ne sont jamais parfaitement orientées dans la même direction et leur plans focaux ne sont donc pas coplanaires. Le principe de la rectification stéréo est de définir deux caméras virtuelles obtenues par des rotations des caméras gauches et droite autour de leurs centres optiques, de telle sorte que leur plans focaux soient confondus, comme illustré Figure 2.5. La transformation à appliquer aux images initiales pour obtenir une paire d'images rectifiées, dépend de la position relative des caméras, soit six paramètres (pour la rotation et la translation). Ces paramètres,

dits *extrinsèques*, peuvent être estimés au préalable par calibration du banc stéréo à l'aide d'une mire damier [Fusiello and Irsara, 2008] et nécessite au préalable la correction des distorsions de chaque caméra.

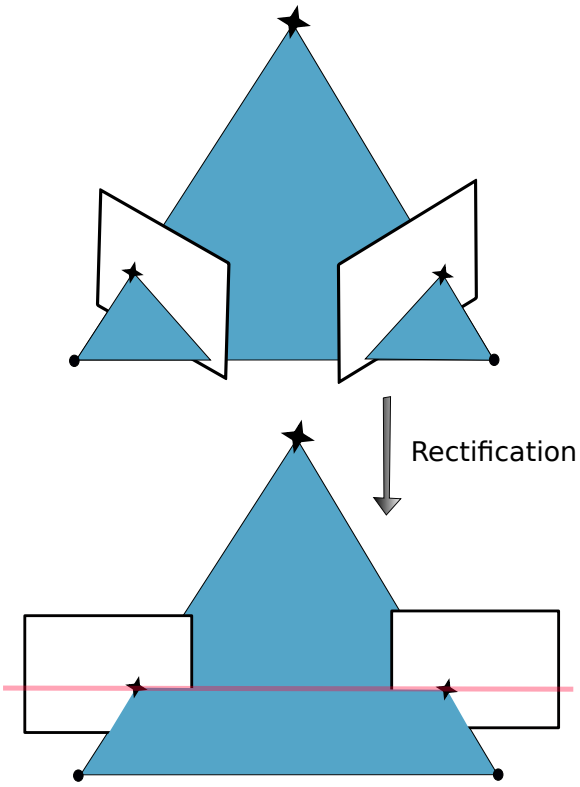

FIGURE 2.5 – Rectification stéréo : les images sont transformées en appliquant une rotation autour des centres optiques afin de rendre les plans images coplanaires.

On supposera désormais que l'on dispose de paires d'images stéréo rectifiées et pour lesquelles les distorsions des lentilles ont été corrigées. Écrivons dans ce contexte, les équations de triangulation.

Traingulation stéréo

Soit un point P de l'espace, de coordonnées (X, Y, Z), observé par deux caméras formant un système stéréoscopique rectifié et dont les centres optiques respectifs O_G et O_D sont distants de b (baseline). Soit f la distance focale (en mètre) supposée identique pour les deux caméras. On note f_x et f_y, les distances focales horizontale et verticale exprimées en pixel, qui s'écrivent (en reprenant les notations des équations (2.5) et (2.6)) :

$$f_x = \alpha f \quad et \quad f_y = \beta f \tag{2.7}$$

On note (X_G, Y_G, f) (respectivement (X_D, Y_D, f))) les coordonnées dans le repère caméra gauche (resp. caméra droite) de la projection de P définie à l'équation (2.1), dans le plan image de la caméra gauche (resp. de la caméra droite). On note (x_G, y_G) et (x_D, y_G) leur coordonnées (en pixels) dans l'image de gauche et de droite. On suppose qu'on utilise des paires d'images stéréo rectifiées, donc :

$$Y_G = Y_D, \quad Z_G = Z_D, \quad y_G = y_D$$

La disparité est une grandeur en pixel, négative, définie par :

$$d = x_D - x_G. \tag{2.8}$$

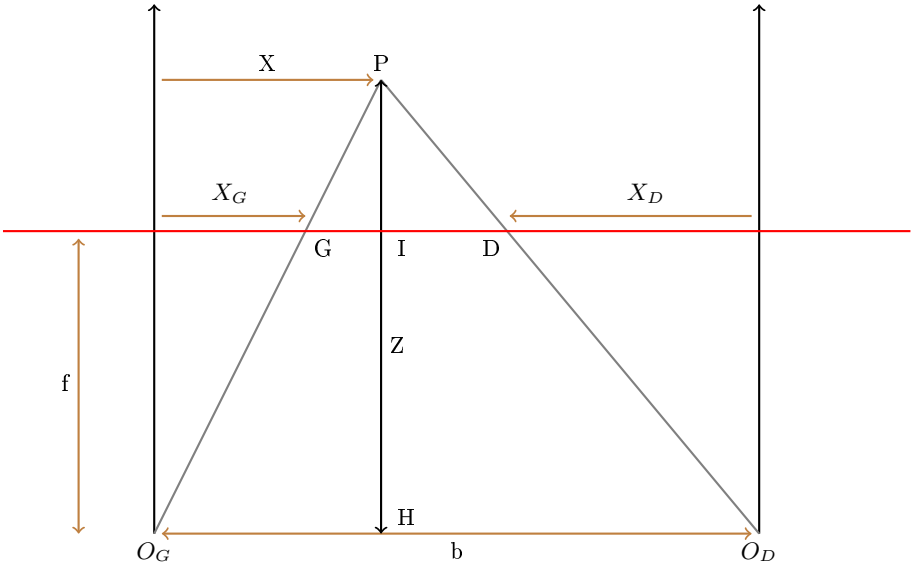

FIGURE 2.6 – Triangulation dans le cas d'un système stéréo rectifié. Sans perte de généralité on représente ici uniquement les dimensions X et Z.

Par application du théorème de Thales à partir de la Figure 2.6, on obtient :

$$\frac{PI}{PH} = \frac{GD}{O_G O_D}, \quad \text{soit : } \frac{Z-f}{Z} = \frac{b+X_D-X_G}{b}$$

On a donc : $-\frac{f}{Z} = \frac{X_D-X_G}{b}$, et par ailleurs : $d = \alpha(X_D - X_G)$. On en déduit que :

$$d = -\frac{bf_x}{Z} \quad et \quad Z = -\frac{bf_x}{d} \tag{2.9}$$

De même on peut écrire :

$$X_G = f\frac{X}{Z} \quad et \quad Y_G = f\frac{Y}{Z} \tag{2.10}$$

Or d'après les équations (2.5), (2.6) et (2.7) :

$$X_G = \frac{f\,(x - x_0)}{f_x} \quad et \quad Y_G = \frac{f\,(y - y_0)}{f_y} \tag{2.11}$$

Finalement on obtient l'équation de triangulation stéréo, en combinant les équations (2.9), (2.10) et (2.11) :

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = -\frac{b}{d} \begin{pmatrix} x - x_0 \\ \frac{f_x}{f_y}(y - y_0) \\ f_x \end{pmatrix} \tag{2.12}$$

Par la suite nous serons parfois amenés à "inverser" cette équation en projetant un point (X, Y, Z) dans l'image pour obtenir ces coordonnées en pixel. Pour cela on définit l'opérateur $\Pi(.)$ à partir de l'équation précédente :

$$\begin{pmatrix} x \\ y \end{pmatrix} = \Pi(X, Y, Z) = \frac{1}{Z} \begin{pmatrix} f_x & 0 & x_0 \\ 0 & f_y & y_0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix} \tag{2.13}$$

Bilan

Nous avons vu dans cette section comment modéliser un banc stéréo et effectuer des corrections et rectifications de la paire d'images stéréo. Ceci afin de faciliter l'appariement stéréo (le correspondant d'un point de l'image gauche, dans l'image droite se situe alors sur la même ligne) qui est nécessaire pour trianguler un point et estimer ses coordonnées 3D dans le repère de la caméra gauche.

Dans la section suivante nous présentons plusieurs algorithmes d'appariement stéréo dense, couramment utilisés dans la littérature, qui permettent d'estimer pour chaque pixel de l'image gauche son correspondant dans l'image droite.

2.2 Méthodes et algorithmes d'appariement stéréo dense

Comme on l'a déjà dit, l'objectif de cette section n'est pas de faire une présentation complète des algorithmes de stéréovision, mais de rappeler quelques approches de base pour introduire les méthodes qui seront comparées dans la suite du chapitre en termes de compromis entre vitesse et précision.

L'objectif des algorithmes de stéréo dense est d'estimer pour chaque pixel (x, y) de l'image gauche I_G, la position $(x + d, y)$ qui lui correspond dans l'image droite I_D (cf. Figure 2.7). Ce qui revient à estimer pour chaque pixel, sa disparité d qui servira à calculer ses coordonnées 3D par triangulation (cf. équation (2.9)). La littérature présente de nombreux algorithmes de stéréo dense, que Scharstein and Szeliski [Scharstein and Szeliski, 2002] classifient notamment en deux catégories : les approches locales que nous présentons à la sous-section 2.2.1, et les approches globales que nous présentons à la sous-section 2.2.2. Depuis cette taxonomie, de nouveaux algorithmes de stéréo dense ont été proposés, notamment l'algorithme *ELAS* [Geiger et al., 2010] qui ne peut pas être considéré comme une approche locale ni globale, et que nous incluons à la liste des algorithmes que nous testons et comparons

dans la section 2.3. Quel que soit la méthode d'appariement stéréo, le principe reste le même : rechercher les pixels qui se ressemblent dans les deux images, au regard d'une certaine mesure de similarité. Cette mise en correspondance se faisant entre pixels, la disparité estimée est à valeur entière. Pour passer outre cette limitation due à la quantification de l'image en pixels, un post-traitement est généralement appliqué à la carte de disparité pour effectuer un raffinement sous-pixellique. Ces méthodes de post-traitement de la carte de disparité sont présentées dans la sous-section 2.2.3.

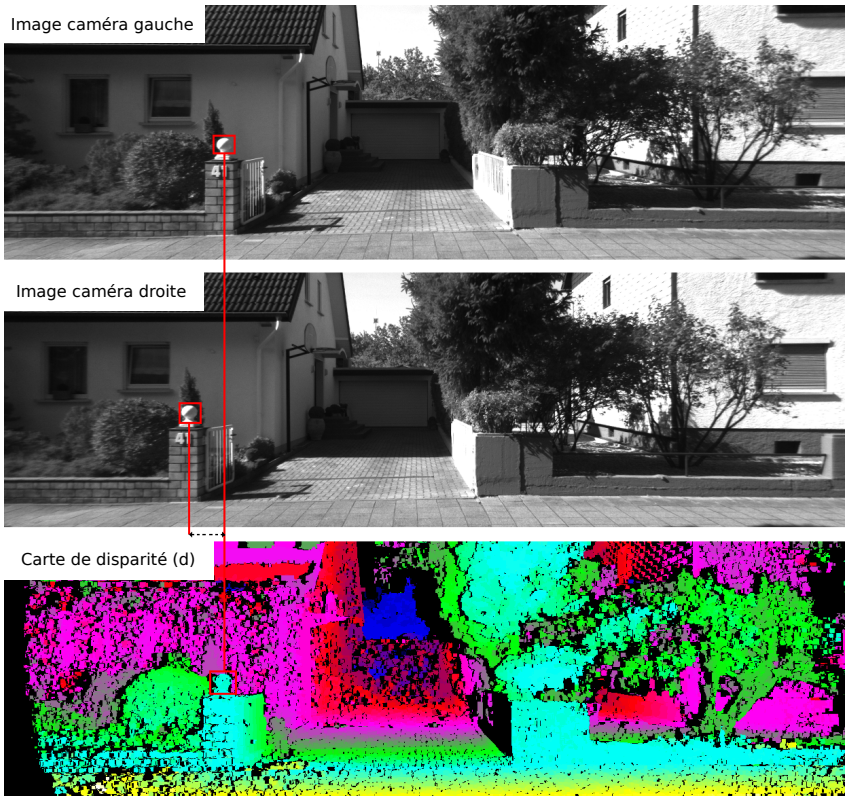

FIGURE 2.7 – Illustration d'une carte de disparité estimée avec l'algorithme stéréo ACTF [Sizintsev and Wildes, 2010] à partir d'une paire d'images stéréo rectifiée

2.2.1 Approches locales

Dans cette sous-section nous décrivons les algorithmes d'appariement stéréo dense appartenant aux approches locales qui estiment la disparité d indépendamment pour chaque pixel (x, y) de l'image gauche, en parcourant l'image droite le long de la même ligne $(., y)$.

L'approche locale traditionnellement utilisée correspond aux méthodes stéréo par fenêtres locales qui peuvent considérer uniquement I_G et I_D, ou bien s'inscrivent dans un schéma *Coarse-To-Fine* en estimant successivement la disparité sur des pyramides d'images construites à partir de I_G et I_D.

Méthodes stéréo par fenêtres locales

Principe. Une approche naïve pour effectuer l'appariement stéréo, consisterait à comparer les intensités $I_G(x,y)$ et $I_D(x+d,y)$ en prenant par exemple la valeur absolue de la différence, et en gardant le candidat d qui minimise cette différence. Une telle mesure de similarité n'est ni suffisamment discriminante ni robuste aux changement d'illumination entre I_G et I_D. C'est pourquoi ces méthodes locales (aussi appelées *Block Matching*), comparent les intensités non pas entre deux pixels mais entre deux régions de l'image au voisinage de (x,y) dans l'image gauche, et au voisinage de $(x+d,y)$ dans l'image droite. Notons $\mathcal{N}(x,y)$ l'ensemble des pixels qui définissent le voisinage de (x,y) (typiquement $\mathcal{N}(x,y)$ est une fenêtre carrée centrée en (x,y)). On utilise aussi les acronymes anglais SSD (*Sum of Squared Differences*) et SAD (*Sum of Absolute Differences*). Les algorithmes de *Block Matching* cherchent généralement à minimiser la somme des carrées (ou des valeurs absolues) des différences d'intensité sur la région $\mathcal{N}(x,y)$:

$$d(x,y) = \underset{d_{\min}\leq d\leq 0}{\arg\min} \sum_{(x',y')\in\mathcal{N}(x,y)} \big(I_G(x',y') - I_D(x'+d,y')\big)^2 . \tag{2.14}$$

Robustesse aux changement d'illumination. Dans l'équation (2.14), la somme des carrés des différences d'intensité est considérée. Mais d'autres métriques peuvent être utilisées, en particulier pour tenir compte du changement d'illumination entre les images gauche et droite. Certaines mesures de similarité compensent par exemple la moyenne et la variance des intensités sur les régions, en estimant localement l'intensité moyenne :

$$\bar{I}(x,y) = \sum_{(x',y')\in\mathcal{N}(x,y)} I(x',y'), \tag{2.15}$$

pour minimiser la somme renormalisée des carrés des différences :

$$\sum_{(x',y')\in\mathcal{N}(x,y)} \left(\frac{\big(I_G(x',y') - \bar{I_G}(x,y)\big)}{\sqrt{\sum_{(\tilde{x},\tilde{y})} \big(I_G(\tilde{x},\tilde{y}) - \bar{I_G}(\tilde{x},\tilde{y})\big)^2}} - \frac{\big(I_D(x'+d,y') - \bar{I_D}(x+d,y)\big)}{\sqrt{\sum_{(\tilde{x},\tilde{y})} \big(I_D(\tilde{x}+d,\tilde{y}) - \bar{I_D}(\tilde{x}+d,\tilde{y})\big)^2}} \right)^2 \tag{2.16}$$

ou encore la corrélation croisée normalisée centrée (ZNCC) :

$$\frac{\sum_{(x',y')\in\mathcal{N}(x,y)} \big(I_G(x',y') - \bar{I_G}(x,y)\big)\big(I_D(x'+d,y') - \bar{I_D}(x+d,y)\big)}{\sqrt{\sum_{(\tilde{x},\tilde{y})} \big(I_G(\tilde{x},\tilde{y}) - \bar{I_G}(\tilde{x},\tilde{y})\big)^2 \big(I_D(\tilde{x}+d,\tilde{y}) - \bar{I_D}(\tilde{x}+d,\tilde{y})\big)^2}}, \tag{2.17}$$

qui est une variante de la corrélation croisée normalisée (NCC) :

$$\frac{\sum_{(x',y')\in\mathcal{N}(x,y)} I_G(x',y') I_D(x'+d,y')}{\sqrt{\sum_{(\tilde{x},\tilde{y})} I_G(\tilde{x},\tilde{y})^2 I_D(\tilde{x}+d,\tilde{y})^2}}; \tag{2.18}$$

Une autre solution pour être invariant aux distorsions radiométriques entre images gauche et droite, consiste à appliquer un pré-traitement sur les images. Par exemple

en prenant l'image des gradients, ou bien en appliquant un détecteur de bords (e.g. filtre de Sobel), ou en considérant la transformée de Rang ou de Census des images [Zabih and Woodfill, 1994] (cf. illustration en Figure 3.3 au chapitre suivant).

Tailles de fenêtres. La taille de fenêtre $\mathcal{N}(x, y)$ est un paramètre important car une fenêtre trop petite ne permet pas de capturer suffisamment d'information dans l'image pour effectuer une mise en correspondance robuste et sans ambiguïté. Ce type de réglage est notamment mis en défaut sur les régions homogènes / peu texturées de l'image. Une fenêtre trop grande quant à elle, non seulement augmente le temps de calcul, mais également accentue le défaut principal des méthodes de Block Matching, à savoir : l'effet d'adhérence aux bords des objets. En effet, l'hypothèse implicite dans l'équation (2.14) est que la disparité est constante localement sur le voisinage $\mathcal{N}(x, y)$, ce qui erroné au niveau des discontinuités des objets dans l'image. En pratique, on observe que la disparité de l'objet le plus proche tend à s'étendre au-delà du support de l'objet. Cet effet est d'autant plus important que la fenêtre est plus grande. Cela donne l'impression que la fenêtre "colle" à l'objet au premier plan, raison pour laquelle certains auteurs ont parlé d'effet d'adhérence pour ce phénomène (terme "Adhesion effect" dans [Delon and Rougé, 2007]), terme que nous réemploierons dans la suite. Notons par ailleurs que l'hypothèse de disparité constante locolament est souvent abusive car elle n'est valable que dans le cas très particulier d'un plan fronto-parallèle à la caméra.

Pondérations variables dans la fenêtre. Plusieurs travaux ont été effectués pour remédier à ce problème en considérant des fenêtres adaptatives dont les dimensions varient afin de couvrir une zone susceptible de contenir des pixels de disparités similaires [Okutomi and Kanade, 1992], ou en utilisant plusieurs fenêtres de formes asymétriques [Fusiello et al., 1997]. Plutôt que de modifier la forme de l'ensemble $\mathcal{N}(x, y)$ d'autres auteurs ont suggéré d'utiliser des pondérations variables sur $\mathcal{N}(x, y)$ en utilisant une pondération de la forme

$$w(x', y') = \exp\left(- \left(\frac{\Delta_c((I(x, y), I(x', y'))}{\gamma_c} + \frac{\sqrt{((x' - x)^2 + (y' - y)^2}}{\gamma_s} \right) \right), \quad (2.19)$$

avec $\Delta_c(.,.)$ la distance dans l'espace de couleur CIELab, pour privilégier la contribution des pixels d'aspect similaire à (x, y). L'idée étant que localement, les pixels de même couleur ou niveau de gris, sont plus susceptibles d'appartenir à la même surface et donc d'avoir la même disparité.

Malheureusement, ces pondérations et fenêtres adaptatives entraînent un surcoût important en termes de temps de calcul. Or l'avantage principal des méthodes locales est leur vitesse d'exécution. Une autre piste intéressante selon nous, est celle adoptée par *Adaptive Coarse-To-Fine Stereo* (ACTF) [Sizintsev and Wildes, 2010]. Leur méthode repose sur un schéma classique *Coarse-To-Fine* (CTF) qui consiste à chercher les appariements stéréo successivement dans une pyramide d'images, que nous présentons dans la sous-section suivante.

Estimation pyramidale

Schéma CTF. Les algorithmes basés sur la méthode pyramidale — une stratégie souvent appelée *Coarse-to-fine* (CTF)— reposent sur l'estimation successive des cartes de disparité sur des versions sous-échantillonnées des images gauche et droite obtenues en construisant une pyramide d'image, en divisant par exemple par 2 (pyramide dyadique) la résolution d'image entre chaque niveau (cf. figure 2.8). Dans le schéma CTF, la carte de disparité estimée à une échelle donnée est ensuite propagée pour l'initialisation du calcul de disparité à l'échelle inférieure. Pour cela, on extrapole la carte de disparité estimée à l'échelle précédente, et on la multiplie par le facteur d'échelle (égal à 2 dans le cas d'un pyramide dyadique), comme illustré figure 2.9. Le pseudo code 1 ci-dessous résume les grandes lignes de l'algorithme stéréo CTF, on utilise les notations suivantes :

- $\rho(.)$, une fonction de distance entre les intensités (e.g. carré des différences)
- $G_\sigma *$, l'opérateur de convolution par une gaussienne de variance σ^2
- $(.)\uparrow_2$, l'opérateur de sur-échantillonnage par un facteur 2
- $(.)\downarrow_2$, l'opérateur de sous-échantillonnage par un facteur 2

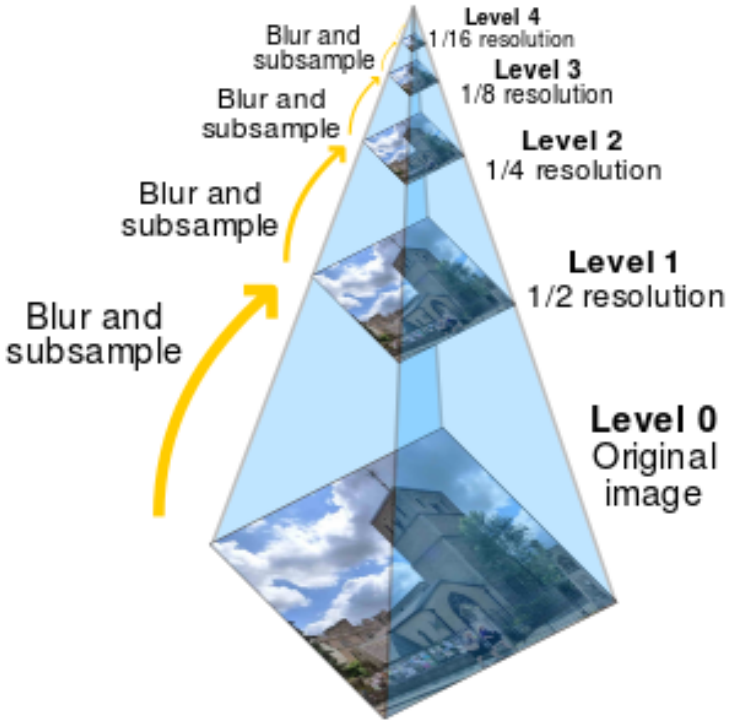

FIGURE 2.8 – Pyramide d'image construite par sous-échantillonnages successifs de l'image initiale à pleine résolution pour obtenir des images de plus en plus basse résolution. Pour éviter le phénomène bien connu de repliement spectral (ou d'*aliasing*), un filtrage passe-bas (e.g. gaussien) est appliqué avant chaque sous-échantillonnage.

Data: Paire d'images rectifiées : I_G et I_D
Result: Carte de disparité stéréo $d = disp^0$
% initialization;
$I_1^0 = I_G$, $I_2^0 = I_D$;
$\forall 1 \leq i \leq 2,\ \forall 1 \leq k \leq K,\ I_i^k = \left(G_\sigma * I_i^{k-1}\right) \downarrow_2$;
$\forall (x, y),\ disp^K(x, y) = 0$;
for k=K :0 **do**
  $disp^k(x, y) = 2 \times disp^{k+1}(x, y) \uparrow_2 +$
  $\arg\min_{-\Delta \leq \delta \leq \Delta} \sum_{(x', y') \in \mathcal{N}(x,y)} \rho\left(I_1^k(x', y'), I_2^k(x' + 2 \times disp^{k+1}(x, y) \uparrow_2 + d_i, y')\right)$;
end

Algorithm 1: Algorithme stéréo CTF

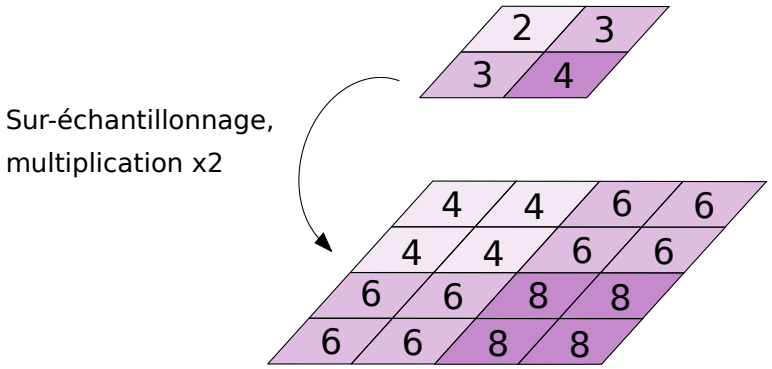

FIGURE 2.9 – Illustration de la propagation des cartes de disparité d'un étage à un autre dans la pyramide.

Cette approche pyramidale permet d'estimer des déplacements plus importants et réduit le problème d'ambiguïté de mise en correspondance liée aux zones homogènes dans l'image, car à basse résolution la plupart des fenêtres $\mathcal{N}(x, y)$ contiennent des régions texturées. Lorsque l'estimation de la disparité échoue à un certain niveau de la pyramide (e.g. à cause du manque de texture), une estimation grossière est quand même obtenue grâce à la propagation de la disparité estimée à l'échelle précédente. On note que cette approche est très efficace en termes de temps de calcul car l'initialisation aux niveaux supérieurs se fait sur des images de dimensions inférieures et n'est donc pas coûteuse, et parce qu'elle permet de limiter la zone de recherche aux niveaux bas de la pyramide. Parmi les défauts de cette approche, on remarque que les erreurs commises lors de l'estimation de d aux échelles supérieures, se propagent et peuvent difficilement être corrigées aux étages inférieurs. De plus, l'effet d'adhérence mentionné précédemment est encore plus important ici avec la propagation de la disparité entre les échelles. C'est pourquoi Sizintsev et Wildes ont proposé une méthode alternative nommée *Adaptive Coarse-to-Fine* (ACTF) qui

consiste à introduire une étape supplémentaire dans le schéma (CTF) pour remettre en question le choix de la valeur de disparité que l'on propage d'une échelle à une autre.

Schéma ACTF. Dans la méthode ACTF [Sizintsev and Wildes, 2010], la disparité retenue en (x, y) lors de la propagation de la carte de disparité entre deux étages de la pyramide, est choisie parmi les disparités $d(x', y')$ au voisinage $\mathcal{N}(x, y)$ en fonction du score de similarité calculé à l'échelle suivante. L'idée sous-jacente est illustrée en Figure 2.10 (voir la légende). La métrique $\rho(.)$ utilisée par les auteurs est NCC (cf. équation (2.18)). le pseudo code 2 décrit l'algorithme ACTF.

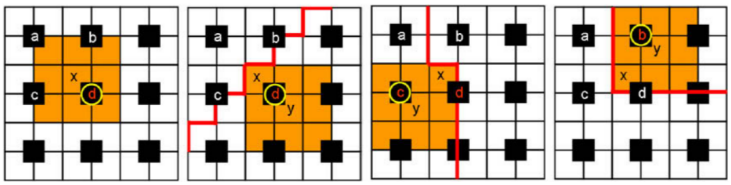

FIGURE 2.10 – Schémas extrait de [Sizintsev and Wildes, 2010]. Les pixels au niveau supérieur sont représentés en noir, les cases blanches ou oranges sont les pixels du niveau inférieur et on considère différents cas de discontinuité dans la disparité. On veut définir la disparité en x. À gauche, sans discontinuité, la solution par défaut est de prendre la disparité du pixel d le plus proche. Lorsqu'il existe des discontinuités de disparités (notées par une frontière rouge) le calcul d'une "confiance" basée sur un score de similarité local conduit à sélectionner une fenêtre décalée, centrée en y, et donc à affecter à x une disparité provenant d'un autre pixel de niveau supérieur (entouré en jaune).

Data: Paire d'images rectifiées : I_G et I_D
Result: Carte de disparité stéréo $d = disp^0$
% initialization;
$I_1^0 = I_G$, $I_2^0 = I_D$;
$\forall 1 \leq i \leq 2,\ \forall 1 \leq k \leq K,\ I_i^k = \left(G_\sigma * I_i^{k-1}\right) \downarrow_2$;
$\forall (x, y),\ disp^K(x, y) = 0$;
for k=K :0 **do**
  $disp_0^k(x, y) = 2 \times disp^{k+1}(x, y) \uparrow_2 +$
  $\arg\min_{-\Delta \leq \delta \leq \Delta} \sum_{(x', y') \in \mathcal{N}(x, y)} \rho\left(I_1^k(x', y'), I_2^k(x' + 2 \times disp^{k+1}(x, y) \uparrow_2 + \delta, y')\right)$;

  $conf_0^k(x, y) = \sum_{(x', y') \in \mathcal{N}(x, y)} \rho\left(I_1^k(x', y'), I_2^k(x' + disp_0^k(x, y), y')\right)$;
  $disp^k(x, y) = disp_0^k\left(arg\min_{(x', y') \in \mathcal{N}(x, y)} conf_0^k(x', y')\right)$;
end

Algorithm 2: Algorithme stéréo ACTF [Sizintsev and Wildes, 2010]. En pratique l'optimisation locale de la confiance peut être effectuée efficacement (notamment sur GPU) et permet de limiter efficacement les erreurs d'appariement aux discontinuités entre les objets, comme nous le verrons dans la suite.

Méthode ELAS avec carte de disparité a priori

ELAS (*Efficient LArge-Scale stereo*) est une méthode locale alternative présentée par Geiger *et al* [Geiger et al., 2010], qui repose sur l'appariement stéréo de points supports $\mathcal{S} = \{\mathbf{s}_n\}_n$ pour construire un modèle a priori $\mathbb{P}\left(d(\mathbf{x})|\mathcal{S}, I_G(\mathbf{x})\right)$ sur la carte de disparité. Avec cet a priori, les auteurs proposent de formuler l'estimation stéréo sous la forme du calcul d'un maximum a posteriori : $d^*(\mathbf{x}) = \arg\max_d \Big(\mathbb{P}\big(d|\mathcal{S}, I_G(\mathbf{x}), I_D\big)\Big)$.

Descripteur utilisé pour les appariements. Les auteurs proposent d'effectuer les appariements stéréo en utilisant un descripteur local $\mathbf{f}(\mathbf{x})$ qui est la concaténation des réponses aux filtres de Sobel horizontal et vertical, sur une fenêtre 9x9 centrée en $\mathbf{x}$. Les auteurs utilisent des filtres de Sobel de taille 3x3. Ce type de descripteur par les gradients de l'image a été introduit précédemment et permet d'améliorer la robustesse de la méthode aux changements d'illumination.

Points supports et modélisation de l'a priori sur la disparité. Les points de supports sont initialement pris sur une grille régulière dans l'image I_G (e.g. tous les 5 pixels). Ces points supports $\mathbf{s}_n$ sont appariés dans l'image droite en cherchant exhaustivement sur la même ligne dans l'image droite le pixel $\mathbf{s}_n+(d,0)$ qui minimise : $\rho(\mathbf{s}_n, d) = |\mathbf{f}_G(\mathbf{s}_n) - \mathbf{f}_D(\mathbf{s}_n + (d,0))|$, la distance L1 entre les descripteurs $\mathbf{f}_G$ et $\mathbf{f}_D$ des pixels dans les images gauche et droite. Ces points de support permettent donc d'obtenir une estimation précise de la disparité même au niveau des objets fins comme par exemple des poteaux de lampadaires qui peuvent parfois poser problème aux méthodes par approche pyramidale.

Par souci de robustesse, un aller-retour est effectué pour vérifier si le pixel $\mathbf{s}_n + (d,0)$ apparié dans l'image droite, est de nouveau apparié au pixel $\mathbf{s}_n$ lorsque l'on effectue l'appariement droite-gauche. Si ce n'est pas le cas, le point support est supprimé. Pour éviter toute ambiguïté, sont également supprimés les points supports $\mathbf{s}_n$ pour lesquels le ratio $\rho(\mathbf{s}_n, d_2)/\rho(\mathbf{s}_n, d)$ est inférieur à 0.9, où d et d_2 sont les disparités correspondant au meilleur et au deuxième meilleur appariement dans I_D.

Une triangulation de Delaunay est faite sur les points de supports restants, et la moyenne $\mu(\mathbf{x})$ de l'a priori $\mathbb{P}\left(d(\mathbf{x})|\mathcal{S}, I_G(\mathbf{x})\right)$ est calculée en tout pixel $\mathbf{x}$ par interpolation linéraire sur le maillage obtenu. L'a priori $\mathbb{P}\left(d(\mathbf{x})|\mathcal{S}, I_G(\mathbf{x})\right)$ est modélisé comme la combinaison d'une loi uniforme et d'une gaussienne tronquée :

$$\mathbb{P}\left(d(\mathbf{x})|\mathcal{S}, I_G(\mathbf{x})\right) \propto \begin{cases} \gamma + \exp\left(-\dfrac{(d(\mathbf{x}) - \mu(\mathbf{x}))^2}{2\sigma^2}\right), \text{si } |d(\mathbf{x}) - \mu(\mathbf{x})| < 3\sigma \text{ ou } d(\mathbf{x}) \in \mathcal{N}_{\mathcal{S}}(\mathbf{x}) \\ 0, \text{ sinon,} \end{cases} \tag{2.20}$$

où $\mathcal{N}_{\mathcal{S}}(\mathbf{x})$ est l'ensemble des disparités des points supports dans un voisinage 20x20 autour de $\mathbf{x}$. On note que cette probabilité est non nulle seulement pour les valeurs de d prises dans l'intervalle $]\mu(\mathbf{x}) - 3\sigma; \mu(\mathbf{x}) + 3\sigma[$, ce qui réduit grandement l'espace de recherche pour estimer le maximum a posteriori de d en minimisant l'énergie présentée à l'équation (2.22).

Estimation du maximum a posteriori. Geiger *et al* modélisent la vraisemblance de $I_D(\mathbf{x}+d)$ de la manière suivante :

$$\mathbb{P}\left(I_D(\mathbf{x}')|d(\mathbf{x}), I_G(\mathbf{x})\right) \propto \begin{cases} \exp\Big(-\beta|\mathbf{f}_G(\mathbf{x}) - \mathbf{f}_D(\mathbf{x}')|\Big), \text{si } \mathbf{x}' = \mathbf{x} + (d,0) \\ 0, \text{ sinon.} \end{cases} \quad (2.21)$$

In fine, l'estimation de la disparité comme maximum a posteriori s'obtient en minimisant l'énergie suivante, qui peut être calculée paralèllement et indépendamment pour chaque pixel $\mathbf{x}$ en cherchant $d(\mathbf{x}) \in]\mu(\mathbf{x}) - 3\sigma; \mu(\mathbf{x}) + 3\sigma[$:

$$\mathcal{E}(d(\mathbf{x})) = \beta\Big|\mathbf{f}_G(\mathbf{x}) - \mathbf{f}_D(\mathbf{x} + (d,0))\Big| - \log\left(\gamma + \exp\left(-\frac{(d(\mathbf{x}) - \mu(\mathbf{x}))^2}{2\sigma^2}\right)\right). \quad (2.22)$$

Cette méthode permet de se ramener à une recherche locale dans l'image : en considérant une distribution de probabilité a priori tronquée aux valeurs assez proches des moyennes locales, et en orientant la recherche par les associations sur les points supports. Ces points de supports jouent un rôle semblable à l'initialisation de la disparité à partir des échelles supérieures dans l'approche *coarse-to-fine*, mais sans les problèmes d'adhérence et de propagation d'erreur dans la pyramide. Nous verrons son efficacité dans la section consacrée aux comparaisons, plus loin dans ce chapitre.

Bilan sur les approches locales

Dans cette sous-section, nous avons vu les méthodes de stéréo dites *locales*. Nous avons mentionné plusieurs problèmes liés aux images, qui peuvent rendre difficile l'estimation des appariements stéréo : les changements d'illumination entre I_G et I_D, ainsi que la présence de régions homogènes ou peu texturées dans l'image.

Pour résoudre le problème de changement d'illumination, des métriques $\rho(.)$ robustes peuvent être considérées, par exemple NCC qui est utilisé dans l'algorithme ACTF. L'algorithme ELAS utilise un descripteurbasé sur les réponses des filtres de Sobel.

Pour gérer la difficulté liée aux régions homogènes dans les images I_G et I_D, nous avons vu que la carte de disparité peut être estimée en utilisant des pyramides d'images. Une autre solution proposée par ELAS, est de s'appuyer sur la disparité de points supports dans l'image, qui permettent de calculer une première estimation de la carte de disparité par interpolation.

Enfin nous avons évoqué le problème d'adhérence qui peut affecter les méthodes locales traditionnelles, et qui est bien géré par l'algorithme ACTF avec ses fenêtres adaptatives, et par l'algorithme ELAS avec ses points supports fournissant des estimation précises même sur les objets fins.

Ces méthodes locales sont de faible complexité algorithmique et sont fortement parallélisables, ce qui les rend très rapides (surtout avec l'approche pyramidale). Cependant elles sont réputées moins précises que les méthodes *gloables* que nous présentons dans la sous-section suivante.

2.2.2 Approches globales

Principe des méthodes globales

Les méthodes *globales* d'appariement stéréo minimisent une énergie définie sur l'ensemble de la carte de disparité d, de la forme :

$$E(d) = E_{obs}(d) + \lambda E_{regu}(d). \tag{2.23}$$

$E_{obs}(d)$ est un terme d'attache aux données caractérisant la cohérence photométrique, avec $\rho(.)$ une fonction de dissimilarité entre pixel du même type que celles utilisées dans les approches locales :

$$E_{obs}(d) = \sum_{(x,y)} \rho\Big(I_G(x,y), I_D(x+d,y)\Big) \tag{2.24}$$

$E_{regu}(d)$ caractérise la régularité spatiale de la carte de disparité d, et modélise les interactions entre pixels voisins dans l'image (on note $\mathcal{V}$ l'ensemble des paires de pixels voisins ou cliques d'ordre 2) :

$$E_{regu}(d) = \sum_{\big((x,y),(x',y')\big)\in\mathcal{V}} c\Big(d(x,y), d(x',y')\Big) \tag{2.25}$$

Enfin, le paramètre λ de l'équation (2.23) est en général choisi par l'utilisateur et permet de régler le compromis entre la cohérence photométrique et spatiale.

Nécessité d'utiliser des hypothèses simplificatrices

D'un point de vue mathématique, l'optimisation d'une énergie du type (2.23), est un problème compliqué. En effet, même en prenant un modèle très simple comme celui de Potts : $c(d,d') = 0$ si $d = d'$, et $c(d,d') = 1$ sinon, l'optimisation de $E(d)$ (cf. équation (2.23)) est un problème NP-complet [Boykov et al., 2001] pour lequel on ne dispose pas de solution rapide et efficace pour trouver l'optimum global. En pratique, des hypothèses simplificatrices sont donc faites pour minimiser $E(d)$.

Différentes techniques pour résoudre le problème d'appariement stéréo

Recuit simulé. La technique de recuit simulé popularisée par Geman et Geman [Geman and Geman, 1984] permet en théorie d'atteindre un optimum global, mais le résultat ne dit rien sur le temps nécessaire pour atteindre cet optimum. En pratique cette méthode est très lente et, dans le contexte de l'appariement stéréo dense où les problèmes sont de grandes dimensions, d'autres méthodes d'optimisation qui ne garantissent que la convergence vers un minimum local sont utilisées.

Technique de *Graph-Cut*. Une autre approche consiste à utiliser la méthode d'optimisation combinatoire *graph-cut*, dont la convergence vers un minimum globale n'est garantie que dans le cas d'une labellisation binaire, mais qui peut servir de méthode approximative d'optimisation pour l'estimation de la disparité correspondant à un problème labélisation multiple (K>2). Comme le montrent Boykov *et al.* [Boykov et al., 2001], si la fonction d'interaction $c(.,.)$ dans $E_{regu}(d)$ est une *métrique*, c'est-à-dire qu'elle vérifie les trois propriétés suivantes :

- $c(\alpha,\beta) = 0 \Leftrightarrow \alpha = \beta$
- $c(\alpha,\beta) = c(\beta,\alpha) \geq 0$
- $c(\alpha,\beta) \leq c(\alpha,\gamma) + c(\gamma,\beta)$

alors la méthode appelée $\alpha-$*expansion* peut être employée itérativement pour trouver une approximation de l'optimum global. Cette méthode consiste à approcher le problème de labellisation multiple par une série de sous-problèmes de labellisation binaire pour lequel le minimum global peut être estimé par une méthode de flot maximum. En effet à chaque itération, pour α correspondant à une valeur de disparité possible, on cherche à segmenter l'image en deux ensembles : les pixels dont la disparité devrait passer à α pour minimiser $E(d)$, et ceux qui doivent rester inchangés (cf. illustration Figure 2.11). Itérativement, pour α parcourant l'ensemble des disparités possibles, on parvient à optimiser approximativement $E(d)$.

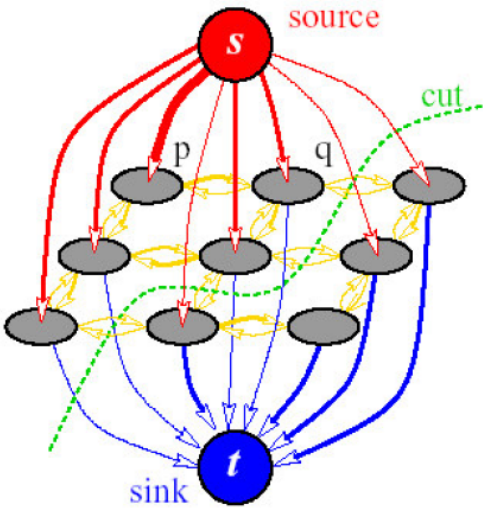

FIGURE 2.11 – Pour chaque $\alpha \in \{d_{\min}, ..., 0\}$, on peut appliquer un algorithme de calcul du flot maximal pour partitionner le graphe en deux et ainsi déterminer quels pixels devraient prendre comme nouvelle valeur de disparité la valeur α.

Algorithmes de "Propagation de Croyance". Une autre approche populaire dans la littérature pour minimiser $E(d)$ même lorsque $c(.,.)$ ne vérifie pas les propriétés d'une métrique, est la stratégie dite de propagation de croyance (*Belief Propagation* (BP) [Sun et al., 2003]). En pratique, même si cette méthode d'inférence

n'est exacte que pour un modèle graphique sans cycles, elle peut quand même être appliquée avec succès sur des graphes cycliques comme le montrent les travaux de Freeman *et al* [Freeman et al., 2000]. La méthode inspirée de BP, présentée par Sun *et al.* est itérative. Elle consiste à effectuer des passages de messages locaux $m^i_{(x',y')->(x,y)}(d(x,y))$ depuis le pixel (x',y') vers (x,y), représentant la croyance de (x',y') que (x,y) devrait prendre pour valeur de disparité $d(x,y)$:

$$m^i_{(x',y')->(x,y)}(d(x,y)) = \min_{d_{\min}\leq d(x',y')\leq 0}\Bigg(\rho\Big(I_G(x',y'), I_D(x'+d(x',y'),y')\Big) + c\Big(d(x,y),d(x',y')\Big) + \sum_{(\tilde{x},\tilde{y})\in\mathcal{N}(x',y')\setminus\{(x,y)\}} m^{i-1}_{(\tilde{x},\tilde{y})->(x',y')}(d(x',y'))\Bigg) \tag{2.26}$$

Après un nombre prédéfini N d'itérations, la disparité finale $d^*(x,y)$ s'écrit :

$$d^*(x,y) = \arg\min_{d_{\min}\leq d(x,y)\leq 0}\Bigg(\rho\Big(I_G(x,y), I_D(x+d(x,y),y)\Big) + \sum_{(x',y')\in\mathcal{N}(x,y)} m^N_{(x',y')->(x,y)}(d(x,y))\Bigg) \tag{2.27}$$

La disparité obtenue est ainsi bien l'optimum d'un critère associant une mesure de similarité locale et d'un terme d'interaction entre pixels voisins.

Programmation dynamique. Pour finir, de nombreux travaux utilisent les méthodes de programmation dynamique pour estimer la carte de disparité. Ces méthodes permettent l'optimisation exacte de $E(d)$ dans le cas particulier où $\mathcal{V}$, qui représente les interactions de régularisation entre pixels de l'image, forme un arbre. Une façon simple de supprimer les cycles dans $\mathcal{V}$, est de retirer toute les interactions verticales entre pixels de l'image en considérant alors uniquement une régularisation ligne par ligne (cf. [Belhumeur and Mumford, 1992, Birchfield and Tomasi, 1999]). Cependant, comme les décisions sont prises indépendamment ligne à ligne, les erreurs d'une ligne à l'autre forment des stries très gênantes sur la carte de disparité estimée. Birchfield et Tomasi [Birchfield and Tomasi, 1999] tentent de pallier ce problème en utilisant la carte de disparité ainsi estimée, comme initialisation pour une minimisation de $E(d)$ par recuit simulé. Dans l'algorithme *Semi Global Matching* (SGM) [Hirschmuller, 2005], très connu en stéréo, chaque pixel (x,y) est à la racine d'un arbre en étoile constitué non seulement de lignes de balayage horizontales (comme dans [Belhumeur and Mumford, 1992, Birchfield and Tomasi, 1999]) mais aussi verticales et diagonales, ce qui évite l'effet de striage. Veskler *et al* [Veksler, 2005] proposent de transformer $\mathcal{V}$ en un arbre, en privilégiant les termes de régularisation entre pixels voisins de même intensité (ou couleur) qui sont plus susceptibles d'avoir la même disparité. Pour cela, ils extraient l'arbre couvrant de poids maximal, en pondérant le graphe de la grille des pixels de sorte à favoriser les connexions entre

pixels d'intensités similaires. La figure 2.12 illustre les différentes configurations précédemment mentionnées du graphe de $\mathcal{V}$, dont les nœuds représentent des pixels et arêtes indiquent l'existence d'un terme de régularisation entre ces deux pixels.

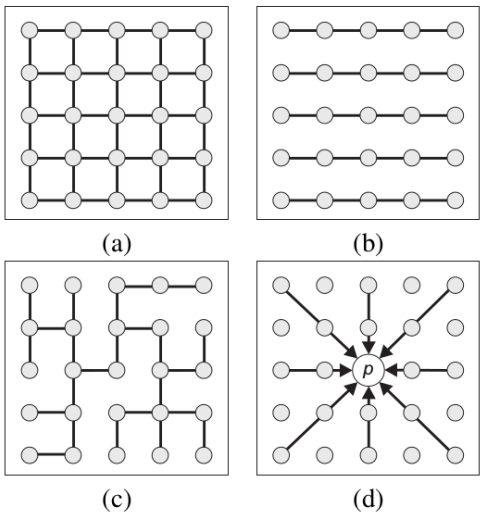

FIGURE 2.12 – Différents graphes de représentation de $\mathcal{V}$. Configurations de : (a) la grille standard de l'image (4-connexité), (b) l'approche par balayage horizontal, (c) l'approche par arbre couvrant de Veksler [Veksler, 2005], et (d) l'approche de Hirschmuller SGM [Hirschmuller, 2005] pour estimer la disparité du pixel (x, y).

Bilan des approches globales

Nous avons décrit dans cette sous-section les approches globales pour l'appariement stéréo dense. L'énergie globale $E(d)$ considérée par ces méthodes autorise une modélisation plus fine que les approches locales, en particulier au niveau des termes de régularisation permettant notamment de tenir compte des occlusions et des discontinuités dans l'image. Cette richesse de modélisation se paie par un problème de minimisation d'énergie difficile qu'il faut simplifier.

Nous avons vu dans cette sous-section différentes façon d'aborder le problème de minimisation d'une énergie globale $E(d)$. L'algorithme SGM, qui utilise une méthode de programmation dynamique pour minimiser $E(d)$, est considéré dans la littérature comme l'état de l'art en stéréo.

Quelle que soit l'approche de minimisation choisie, la complexité algorithmique plus élevée rend ces approches globales toujours plus lentes que les méthodes locales décrites à la sous-section 2.2.1.

2.2.3 Raffinement sous-pixellique des cartes de disparité

Les méthodes d'appariement stéréo locales et globales présentées dans les sous-sections précédentes, retournent une carte de disparité à valeurs entières. Pour aller

en-deçà du pixel et passer outre cette limitation due à la quantification pixellique de l'image, une étape d'affinage est généralement effectuée sur la carte de disparité. Une telle étape permet notamment d'éviter l'effet "marche d'escalier" qui apparaît lors de l'estimation de la disparité au niveau de surfaces inclinées non fronto-parallèles à la caméra (cf. [Scharstein and Szeliski, 2002]).

Raffinement parabolique

Une méthode couramment utilisée dans la littérature pour obtenir une estimation sous-pixellique avec une approche locale, est de faire correspondre une courbe parabolique à la fonction de coût d'appariement stéréo autour du minimum (cf. [Shimizu and Okutomi, 2001]), comme illustré sur la Figure 2.13. Notons $f_{(x,y)}(d)$ la fonction de dissimilarité minimisée, par exemple :

$$f_{(x,y)}(d) = \sum_{(x',y')\in\mathcal{N}(x,y)} \Big(I_G(x',y') - I_D(x'+d,y')\Big)^2 \tag{2.28}$$

Notons $d^- = d-1$ et $d^+ = d+1$. La valeur d^* de l'estimation sous-pixellique correspondant au minimum de la parabole passant par les 3 points $(d^-, f_{(x,y)}(d^-))$, $(d, f_{(x,y)}(d))$ et $(d^+, f_{(x,y)}(d^+))$ s'écrit :

$$d^* = \frac{f_{(x,y)}(d^-) - f_{(x,y)}(d^+)}{2f_{(x,y)}(d^-) - 4f_{(x,y)}(d) + 2f_{(x,y)}(d^+)} \tag{2.29}$$

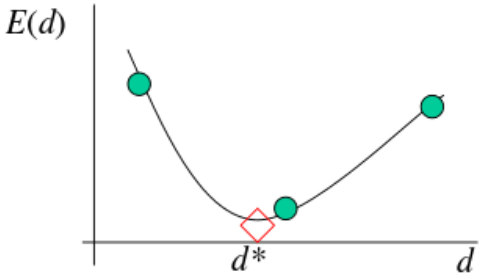

FIGURE 2.13 – Un raffinement sous-pixellique d^* de la disparité d estimée, peut être obtenu en prenant le minimum de la parabole passant par les 3 points d'abscisses $d-1$, d, et $d+1$ et d'ordonnées égales à leur coût (mesure de dissimilarité) associé.

Raffinement par sur-échantillonnage

Une autre manière classique d'obtenir une estimation sous-pixellique est de sur-échantillonner d'un facteur 2 (ou même 4) les images gauche et droite afin d'effectuer l'estimation sur des fractions de disparité : 1/2 (ou 1/4). Cette méthode est applicable aux approches locales et globales, mais multiplie également par 2 (ou 4) le temps de calcul des algorithmes.

Scharstein et Szeliski [Scharstein and Szeliski, 2002] préconisent de combiner cette technique de raffinement avec la précédente pour obtenir la meilleure précision possible : en estimant la disparité sur des demi-pixels avant de calculer le raffinement parabolique (2.29).

Filtrage par *Consensus de Région*

Comme on l'a vu, certains défauts de la stéréo sont liés à l'hypothèse de surfaces fronto-parallèles qui est implicite derrière l'utilisation de fenêtres ou, d'une manière générale, derrière la nature discrète de la disparité. Nombre de méthodes ont cherché à pallier ces limitations en estimant localement des surfaces planes non nécessairement fronto-parallèles, éventuellement comme une étape d'affinage de la disparité. Parmi ces méthodes nous décrivons dans ce paragraphe les travaux de Chakrabarti *et al.* [Chakrabarti et al., 2015], qui prennent en entrée une carte de disparité d_0 pour la filtrer en appliquant localement et à différentes échelles, des modèles de plans inclinés paramétrés par le vecteur tridimensionnel θ :

$$d(x,y) = U_{(x,y)}\theta \tag{2.30}$$

avec

$$U_{(x,y)} = \begin{bmatrix} x & y & 1 \end{bmatrix}^T \tag{2.31}$$

L'idée est de surparamétrer le problème en estimant pour chaque pixel (x,y) plusieurs paramètres θ_p pour tous les plans définis sur des régions p de tailles croissantes contenant (x,y). Plus précisément, on utilise l'ensemble de toutes les régions carrées que l'on peut extraire dans l'image, et dont la largeur est comprise dans les valeurs $\{4, 8, 16, 32, 64\}$ pixels, ensemble que l'on note $\mathcal{P}$. L'objectif est de calculer $d^*(x,y,)$ en combinant des plans définis sur les régions $p \ni (x,y)$ qui sont représentatifs de la géométrie locale. Pour cela on estime aussi une variable binaire $I_p \in \{0;1\}$ indiquant si un plan défini sur $p \in \mathcal{P}$ est cohérent avec les données locales de disparité $\{d_0(\mathbf{x}), \mathbf{x} \in p\}$ — on dit alors qu'il est "inlier". Ainsi, Chakrabarti *et al.* minimisent l'énergie suivante :

$$E\Big(\{I_p, \theta_p\}_{p\in\mathcal{P}}\Big) = \sum_{p:I_p=0} \tau_p + \sum_{p:I_p=1} \sum_{(x,y)\in p} \Big(U_{(x,y)}\theta - d_0(x,y)\Big)^2 + \lambda \sum_{(x,y)} |J_{(x,y)}| Var\Big(\{U_{(x,y)}\theta_p\}_{p\in J_{(x,y)}}\Big) \tag{2.32}$$

où $J_{(x,y)}$ est l'ensemble des régions inliers de $\mathcal{P}$ qui contiennent le pixel (x,y) :

$$J_{(x,y)} = \{p \in \mathcal{P} : p \ni (x,y), I_p = 1\} \tag{2.33}$$

Le premier terme de l'énergie définie par l'équation (2.32) pénalise la création de régions outliers de sorte à assurer que l'ensemble des pixels de l'image soit couverts par au moins une région inlier. Le deuxième terme est un terme de fidélité aux données (à savoir la carte de disparité initiale d_0) s'appliquant sur les régions p

déclarées inliers. Quant au troisième terme, il garantit une faible variabilité entre les valeurs de disparité prédites par chaque hypothèse de plan *inlier* couvrant un même pixel (x, y). Les auteurs proposent ensuite de minimiser itérativement une approximation de cette énergie.

Dans le contexte de la vision embarquée qui nous intéresse, cette optimisation itérative est bien trop coûteuse en temps de calcul, et nous considérons le cas particulier $\lambda = 0$ qui permet d'optimiser l'énergie définie à l'équation (2.32) en une seule itération :

- $\hat{\theta}_p = \left(\sum_{\mathbf{x}\in p} U(\mathbf{x})^T U(\mathbf{x})\right)^{-1} \left(\sum_{\mathbf{x}\in p} U(\mathbf{x})^T d_0(\mathbf{x})\right)$
- $\hat{I}_p = \begin{cases} 0, \text{ si } \sum_{\mathbf{x}\in p} \left(U(\mathbf{x})\theta_p - d_0(\mathbf{x})\right)^2 > \tau_p \\ 1, \text{ sinon.} \end{cases}$

In fine, lorsque les variables $\{I_p, \theta_p\}_{p\in\mathcal{P}}$ ont été estimées, la valeur de la disparité affinée s'écrit :

$$d^*(x, y) = \frac{1}{\sum_{p\ni(x,y)} \hat{I}_p} \sum_{p\ni(x,y)} U_{(x,y)} \hat{\theta}_p \hat{I}_p \tag{2.34}$$

Cette méthode de filtrage que nous appellerons *Consensus de Régions*, donne de très bons résultats en lissant correctement la carte de disparité tout en préservant les discontinuités.

Bilan

Nous avons vu dans cette sous-section trois méthodes de raffinement sous-pixellique de la carte de disparité stéréo. Le minimum de la parabole de la fonction de coût passant par $\{d-1; d; d+1\}$, est utilisable en pratique pour une méthode de stéréo *locale*. L'estimation sur des fractions de pixel peut s'appliquer pour tout type de méthodes stéréo, en sur-échantillonnant au préalable la paire d'image stéréo. Enfin nous avons proposé une méthode par *Consensus de Régions*, comme une version simplifiée des travaux de Chakrabarti *et al.* [Chakrabarti et al., 2015], pour filtrer une carte de disparité.

Le raffinement par sur-échantillonnage multiplie par 2 ou plus le temps de calcul, et n'est donc pas envisageable dans un contexte temps-réel. D'autre part, tout comme le raffinement parabolique, le sur-échantillonnage de la disparité ne permet pas de filtrer les outliers (i.e. pixels pour lesquels la disparité estimée diffère grandement de la vérité terrain), contrairement au filtrage par *Consensus de Régions* qui permet par ailleurs de mieux respecter les discontinuités dans la carte de disparité. De plus, comme on le voit à la section 2.3 suivante, *Consensus de Régions* est particulièrement efficace et son implémentation est très rapide. C'est donc cette dernière technique que nous choisissons pour le post-traitement de la carte de disparité.

2.3 Évaluation des algorithmes de stéréo dense

2.3.1 Données KITTI pour l'évaluation des algorithmes de stéréo

Pour nos évaluations nous avons choisi d'utiliser le jeu de données KITTI 2012 [Geiger et al., 2012] constitué de paires d'images stéréo acquises depuis un banc placé sur le toit d'une voiture se déplaçant en milieu urbain. La voiture KITTI, présentée à la Figure 2.14, embarque deux bancs stéréo dont la baseline est d'environ 54cm : un qui acquiert des images en couleur et l'autre qui acquiert des images en noir et blanc. Les algorithmes sont testés sur les images stéréo en noir et blanc et dont la résolution est approximativement de 370 x 1200 pixels. La vérité terrain pour la carte de profondeur, est obtenue grâce au système de localisation associant un GPS et une centrale inertielle, et aux mesures du laser Velodyne qui fournit la profondeur pour environ 30% des pixels dans l'image.

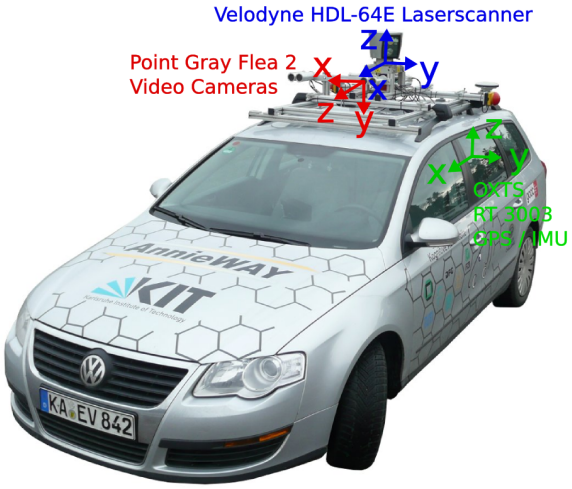

FIGURE 2.14 – Illustration de la voiture et des capteurs utilisés pour la construction du jeu de données KITTI [Geiger et al., 2012].

L'intérêt de ce jeu de données est qu'il est acquis en conditions réelles. Ces conditions correspondent donc à des scènes urbaines observées par un véhicule en déplacement, ce qui implique des mouvements apparents très variables dans l'image, avec notamment des grandes vitesses de défilement en bord de champ. Enfin, ce jeu de données fait aussi apparaître des difficultés liées à la fois à des problèmes d'illumination (e.g. saturation du capteur, réflexions sur les vitres) et des zones homogènes ou peu texturées (e.g. certaines façades, la route...).

2.3.2 Évaluation et comparaison des algorithmes de stéréo

Nous avons choisi de comparer ici plusieurs techniques d'appariement stéréo dense parmi les différentes approches décrites précédemment : *Belief Propagation*, *Semi-Gobal Matching*, pour les méthodes gloables et les méthodes locales : *Efficient*

Large Scale stereo matching, *Coarse-To-Fine* et *Adaptive Coarse-To-Fine*.

L'algorithme de propagation de croyance que nous avons testé est : *Constant Space Belief Propagation* (CSBP) [Yang et al., 2010], qui est disponible dans la bibliothèque open source OpenCV [Bradski, 2000]. L'algorithme original SGM décrit par Hirschmuller [Hirschmuller, 2005] utilise l'information mutuelle comme fonction de coût. Cependant, pour des raisons de coût de calcul, beaucoup de travaux utilisent le même algorithme avec une fonction de coût de Birchfield-Tomasi : c'est cette version, appelée SGBM dans la bibliothèque OpenCV, que nous utilisons ici. Pour ELAS [Geiger et al., 2010], nous avons utilisé la bibliothèque C++ LIBELAS dont le code a été mis en ligne par les auteurs, en prenant les paramètres par défaut et en testant aussi avec l'option dite de "sous-échantillonnage" qui permet d'accélérer le calcul notamment en utilisant moins de points supports, et que nous désignons "$\text{ELAS}_{1/2}$". Enfin nous avons implémenté en C++/CUDA les algorithmes CTF et ACTF ainsi que le post-traitement par *Consensus de Régions* que nous testons aussi avec ACTF (alors désigné "ACTF+CoR").

TABLE 2.1 – Tableau récapitulatif des acronymes faisant référence aux différents algorithmes d'appariement stéréo et de post-traitement utilisés.

Acronyme	Méthode
CSBP	*Constant Space Belief Propagation* [Yang et al., 2010]
SGBM	*Semi Global Bloc Matching* [Bradski, 2000]
ELAS	*Efficient LArge Scale stereo* [Geiger et al., 2010]
$\text{ELAS}_{1/2}$	ELAS avec l'option de sous-échantillonnage [Geiger et al., 2010]
CTF	*Coarse-To-Fine stereo*
ACTF	*Adaptive Coarse-To-Fine stereo* [Sizintsev and Wildes, 2010]
ACTF + CoR	ACTF + *Consensus of Regions* [Chakrabarti et al., 2015]

Résultats qualitatifs

Les Figures 2.15 et 2.16 permettent d'apprécier qualitativement les cartes de disparité estimées par chacun des algorithmes testés. On a choisi deux types de scènes assez différentes, l'une (Fig. 2.15) dans une rue avec des véhicules garés, donc un environnement complètement artificiel, avec beaucoup de surfaces homogènes voire réfléchissantes ; l'autre (Fig. 2.16) comprend des parties de végétations importantes et des effets d'ombre portée sur la route. À partir de ces résultats qualitatifs nous pouvons faire plusieurs remarques.

Tout d'abord CSBP est peu robuste et peine notamment au niveau du sol et des ombres, comme on le voit sur la Figure 2.16. On note aussi que l'algorithme ELAS qui s'appuie sur un ensemble de points supports préalablement appariés, rend un résultat incomplet, avec notamment de grandes zones manquantes sur les bords latéraux de l'image : ce comportement est gênant, sachant que les objets les plus

proches du véhicule sont souvent localisés dans ces régions (obstacles sur les bords de la route, croisement de véhicules). L'algorithme SGBM, même s'il ne parvient pas non plus à couvrir toute l'image, retourne néanmoins une carte de disparité précise sur la quasi-totalité de l'image. Quant aux méthodes par fenêtres locales, on constate que ACTF est effectivement plus précis que CTF qui présente plus d'*outliers* et conduit à un "grignotage" des bords des objets, visible sur la Fig. 2.16.

Enfin le filtrage par Consensus de Régions permet à la fois de lisser la carte de disparité et d'obtenir une estimation sous-pixellique tout en préservant assez bien les bords des objets. On note cependant que Consensus de Régions, qui modélise la carte de disparité comme des combinaisons linéaires de plans inclinés à différentes échelles, rencontre parfois des difficultés notamment au niveau des voitures pour lesquels la carte de disparité peut être erronée à cause de la transparence et des réflexions sur les vitres. Hormis ce cas où Consensus de Régions ne parvient pas à estimer correctement un modèle local de plans inclinés, on constate qu'il permet de supprimer des artefacts comme ceux présents sur la route Figure 2.15.

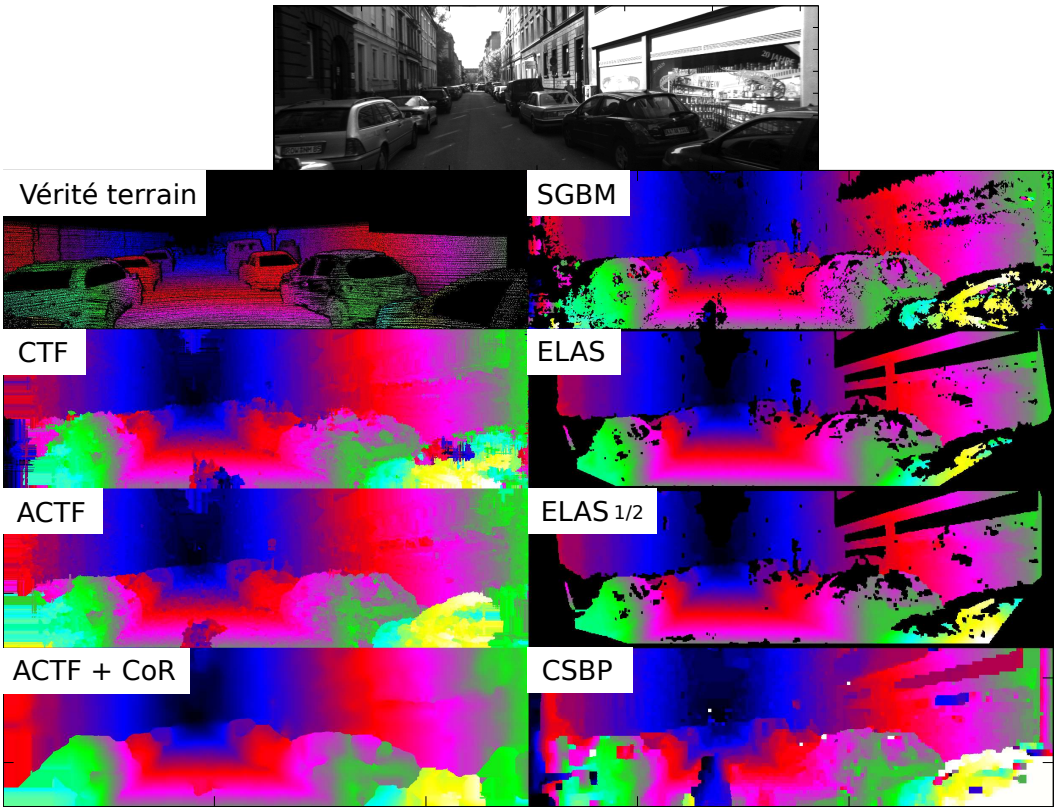

FIGURE 2.15 – Cartes de disparité obtenues avec différents algorithmes. En haut à gauche figure la vérité terrain acquise avec des scans laser.

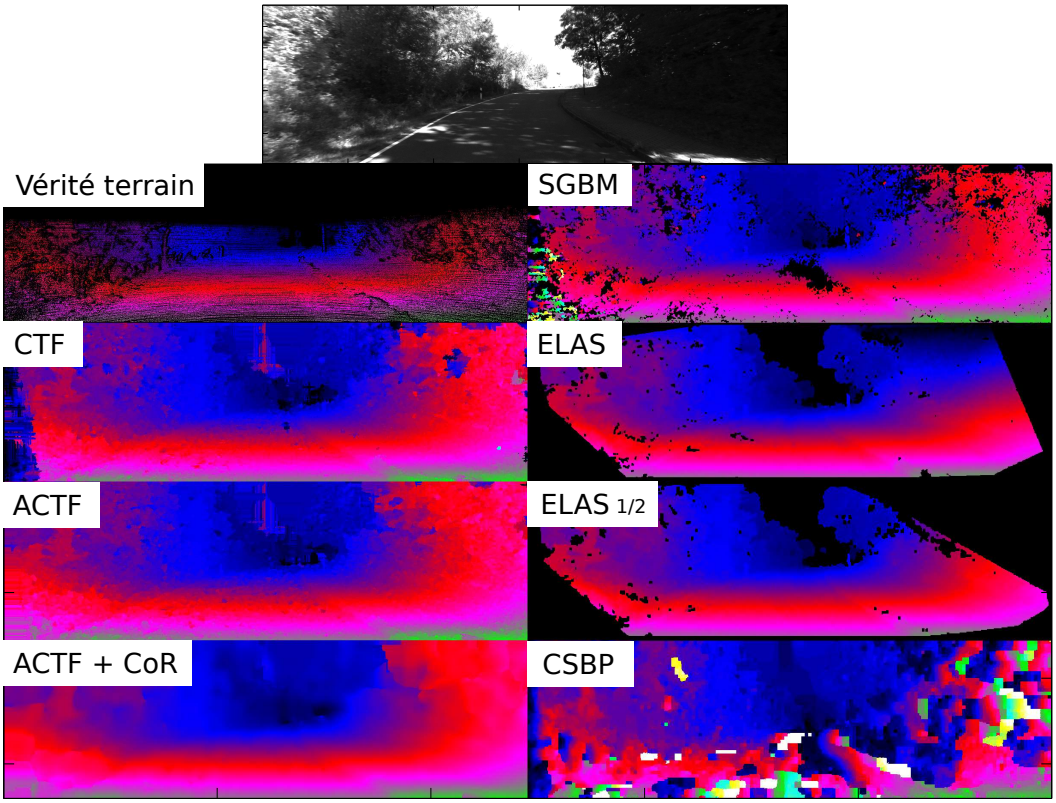

FIGURE 2.16 – Cartes de disparité obtenues avec différents algorithmes. En haut à gauche figure la vérité terrain acquise avec des scans laser.

Evaluation quantitative et choix de l'algorithme stéréo

Les différents algorithmes stéréo mentionnés précédemment ont été testés sur le jeu de données d'entrainement KITTI 2012, sur lequel on dispose de la vérité terrain partielle fournie par le télémètre laser. Deux tableaux résument les résultats : dans le premier, Table 2.2, la densité des résultats n'est pas totale pour les méthodes SGBM, ELAS et $\text{ELAS}_{1/2}$ (comme on l'avait vu dans l'étude qualitative). Or la densité des cartes de disparité est un critère important : d'une part, pour la suite de nos travaux elle est utile à la reconstruction 3D de la scène et notamment des zones situées sur les bords, ainsi qu'à la détection des objets mobiles ; d'autre part le critère de densité est un facteur de modification des performances : il est souvent possible d'améliorer la précision des résultats d'une méthode à densité variable en diminuant la densité. Pour éviter ces biais dans les comparaisons il est préférable de travailler à la densité de référence : 100%. Nous présentons donc en Table 2.3 un comparatif avec des cartes de disparités complétées par interpolation linéaire à partir des données renseignées sur les bords des zones manquantes.

La Table 2.2 montre que SGBM et ELAS obtiennent les résultats les plus précis lorsqu'on conserve leur densité inférieure à 100%. SGBM se distingue par sa densité plus élevée et sa précision meilleure. Dans les résultats "densifiés" de la Table 2.3,

la méthode ACTF+CoR prend l'avantage, notamment par rapport à ELAS dont la performance se dégrade nettement, mais aussi en comparaison avec SGBM. Notons qu'on pourrait rechercher des méthodes d'interpolation plus précises, mais ce serait sans doute des efforts importants pour des améliorations limitées car il est toujours possible de piéger une méthode d'interpolation qui, par nature, fait des hypothèses sur des données non observées, et par ailleurs il faut rester dans des coûts de calculs raisonnables. Avec une densité maximale ACTF + CoR apparaît donc comme l'algorithme le plus précis et il est, par ailleurs, nettement plus rapide que les autres (hormis ACTF et CTF dont les résultats sont nettement moins précis).

Notons que l'implémentation de SGBM est en C++ et non optimisée : il existe une implémentation FPGA [Gehrig et al., 2009] qui peut fournir des résultats à cadence vidéo (à 25Hz sur des images de taille 600x480 avec 128 niveaux de disparité), mais nous n'en disposions pas. Il aurait intéressant d'ajouter à ce comparatif une implémentation GPU de SGBM (ou même une implémentation C++ optimisée), mais c'est un algorithme très complexe et ce n'était pas réalisable dans le cadre de ce travail de thèse.

Pour ces raisons, nous choisissons donc d'utiliser par la suite ACTF ou ACTF+CoR dont la vitesse d'exécution sur GPU est compatible avec les exigences de temps de traitement imposés par une utilisation en ligne sur une plateforme embarquée.

TABLE 2.2 – Comparaison des performances des algorithmes de stéréo sur le jeu de données d'entrainement de KITTI 2012, sur tous les pixels de la vérité terrain, en termes de pourcentage d'outliers (*Out-*) et d'erreur moyenne en pixel (*Avg-*) sur les parties non occultées (*-Noc*) et sur toute l'image (*-All*)

Algorithme Stéréo	Out-Noc	Out-All	Avg-Noc	Avg-All	Densité	Temps de calcul
SGBM	4.12%	4.76%	1.1px	1.3px	81.5%	387ms
ELAS	4.77%	4.77%	0.9px	1.0px	69.3%	147ms
$ELAS_{1/2}$	5.00%	5.01%	1.1px	1.1px	67.1%	54ms
ACTF + CoR	7.03%	8.01%	1.4px	1.5px	100.0%	19ms (GPU)
ACTF	11.07%	12.09%	1.8px	2.0px	100.0%	10ms (GPU)
CTF	13.67%	14.70%	2.4px	2.6px	100.0%	10ms
CSBP	26.53%	28.18%	8.5px	9.3px	100.0%	58ms (GPU)

TABLE 2.3 – Comparaison des performances des algorithmes de stéréo sur le jeu de données d'entrainement de KITTI 2012, sur tous les pixels de la vérité terrain, en termes de pourcentage d'outliers (*Out-*) et d'erreur moyenne en pixel (*Avg-*) sur les parties non occultées (*-Noc*) et sur toute l'image (*-All*)

Algorithme Stéréo	Out-Noc	Out-All	Avg-Noc	Avg-All	Densité	Temps de calcul
ACTF + CoR	7.03%	8.02%	1.4px	1.5px	100.0%	19ms (GPU)
SGBM	8.37%	10.16%	2.0px	2.8px	100.0%	384ms
ELAS	10.83%	12.04%	1.8px	2.0px	100.0%	147ms
ACTF	11.07%	12.09%	1.8px	2.0px	100.0%	10ms (GPU)
$ELAS_{1/2}$	12.68%	13.90%	2.6px	2.8px	100%	54ms
CTF	13.67%	14.70%	2.4px	2.6px	100.0%	10ms
CSBP	26.53%	28.18%	8.5px	9.3px	100.0%	58ms (GPU)

2.4 Conclusion

Comme nous l'avons vu dans ce chapitre, un banc stéréo calibré permet d'obtenir des images gauche et droite rectifiées à partir desquelles une carte de disparité peut être estimée avec l'algorithme stéréo de notre choix, avant d'être raffinée pour atteindre une précision sous-pixellique. Par triangulation cette carte de disparité fournit une carte de profondeur de la scène observée à chaque instant d'acquisition, dans la mesure où la méthode choisie est compatible avec la cadence d'acquisition. Cette contrainte de temps de calcul nous a conduit à sélectionner une méthode que nous avons réalisée à partir de deux approches publiées : ACTF + CoR. Il faut noter que ces deux approches ne sont pas spécialement mises en avant dans les comparatifs ou les revues sur les méthodes de stéréovision. Nous avons montré cependant que la méthode ACTF+CoR permet d'obtenir une information sur l'ensemble de l'image (densité de 100%) avec une bonne précision (Avg-Noc= $1.4px$) et pour un temps de calcul très réduit (19ms sur un GPU GeForce GTX Titan) donc compatible avec une cadence de 10Hz.

À ce stade nous disposons d'un moyen de perception 3D de l'environnement à un instant donné, mais nous n'avons aucune information dynamique, notamment concernant le déplacement de la caméra et le mouvement des objets mobiles potentiellement présents. Le chapitre suivant aborde les différents algorithmes permettant d'estimer le déplacement de la caméra, ainsi que le mouvement apparent dans l'image (Flot Optique), et dans la scène observée (Flot de Scène).

Chapitre 3

Estimations Temporelles : Odométrie Stéréo, Flot Optique, Flot de Scène

Sommaire

Ce chapitre a donné lieu à la publication [2] de la liste fournie en section 6.3.

Introduction

Dans ce chapitre nous intéressons à l'estimation des informations dynamiques, à savoir : le déplacement du banc stéréo, le mouvement apparent des éléments de l'image (ou "flot optique") et le mouvement 3D de la scène entre deux instants d'acquisition.

La perception et la modélisation d'environnements dynamiques nécessitent d'estimer le mouvement des éléments dynamiques présents dans le milieu dans lequel on évolue. Si l'on dispose d'une caméra, on peut calculer le mouvement apparent dans l'image à l'aide d'algorithmes d'estimation du flot optique qui prennent en entrée les images acquises à deux instants, et qui renvoient le champ des déplacements des pixels entre les deux images (section 3.2). Mais on mesure uniquement dans ce cas la projection du mouvement 3D sur le plan image. Si l'on dispose d'un banc stéréo (donc de deux caméras) on peut directement estimer le mouvement 3D des points observés, dans le repère du banc stéréo, à l'aide d'algorithmes d'estimation du flot de scène (section 3.3). De plus, le déplacement du banc stéréo peut être calculé. L'estimation du déplacement du banc stéréo à l'aide d'algorithmes d'odométrie visuelle (section 3.1), est primordiale car ce déplacement contribue à une partie du flot optique, et du flot de scène. Enfin, en compensant le mouvement induit par le déplacement de la caméra, on peut calculer le mouvement propre à chaque point en 3D et ainsi vraiment percevoir l'environnement dynamique.

3.1 Calcul du déplacement de la caméra par odométrie visuelle

Une manière efficace d'estimer le déplacement du banc stéréo est d'utiliser un algorithme d'odométrie visuelle. Les algorithmes d'odométrie visuelle permettent d'estimer la trajectoire parcourue par le robot depuis sa position initiale, en s'appuyant sur les images acquises à partir d'une seule caméra (monoculaire) ou à partir d'un banc stéréo. Ici, nous supposons que nous disposons d'un banc stéréo. Notons que l'odométrie stéréo possède certains avantages par rapport à l'odométrie monoculaire. D'une part la stéréoscopie permet d'obtenir, à chaque instant, des informations 3D avec une précision connue et stable, alors que la vision monoculaire nécessite plusieurs images selon des points de vues séparés d'une translation minimale. D'autre part, la vision monoculaire souffre d'une ambiguïté d'échelle : si on ne connaît pas la distance que la caméra a parcourue entre deux prises d'images, alors on ne peut pas connaître l'échelle absolue des dimensions ou des déplacements 3D estimés à partir de ces deux images. Au contraire, l'échelle est fixée pour un banc stéréoscopique par la connaissance de la baseline entre les caméras. Dans notre cas nous nous intéressons seulement à l'estimation du déplacement du banc entre deux instants consécutifs d'acquisition et non pas à l'estimation à plus long terme de la pose du système par rapport au repère intial.

Le problème qui consiste à chercher la position et l'orientation de la caméra à une ou plusieurs prises de vue, se nomme *Calcul de Pose*. On distingue deux types d'approches en stéréovision pour résoudre ce problème : le calcul de pose utilisant un critère "3D-3D", et le calcul de pose "2D-3D". Nous décrivons chacune de ces approches en sous-sections 3.1.1 et 3.1.2, avant de détailler à la sous-section 3.1.3 l'algorithme de calcul de pose 2D-3D que nous avons choisi d'utiliser par la suite.

3.1.1 Calcul de pose 3D-3D

Le calcul de pose 3D-3D tel qu'il est présenté par Umeyama [Umeyama, 1991] consiste à estimer les paramètres $\{R, T, c\}$ de rotation, translation et facteur d'échelle pour recaler deux ensembles de points 3D appariés, en minimisant le critère des moindres carrés. Dans notre cas, illustré Figure 3.1, nous nous plaçons dans le référentiel de la caméra gauche à l'instant t (noté $\mathcal{R}^t$ dans la figure) et souhaitons calculer $\{R, T\}$ entre la pose à l'instant t et la pose à l'instant $t+1$ (sans modification du facteur d'échelle). Pour ce faire, N points remarquables sont extraits dans l'image gauche à t et suivis temporellement dans l'image gauche à $t+1$, ce qui nous donne après l'appariement et la triangulation stéréo, deux ensembles de points 3D de coordonnées : $\{X_i^t\}_{1 \leq i \leq N}$ et $\{X_i^{t+1}\}_{1 \leq i \leq N}$. L'énergie à minimiser s'écrit :

$$\mathcal{E}(R, T) = \frac{1}{N} \sum_{i=1}^{N} \| X_i^{t+1} - \left(R X_i^t + T \right) \|^2 \tag{3.1}$$

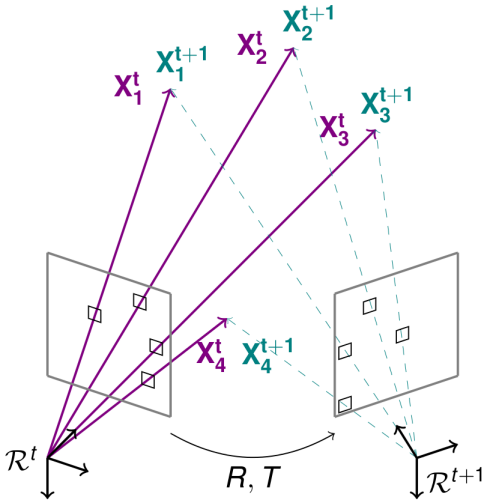

FIGURE 3.1 – Illustration du problème de calcul de pose 3D-3D pour estimer la position et l'orientation relative de la caméra gauche à l'instant $t+1$ par rapport à la caméra gauche à l'instant t.

Umeyama démontre que l'optimum de l'énergie (3.1) peut être calculé par une décomposition de matrice en valeurs singulières (*Singular Value Decomposition* (SVD)),

avec les formules présentées dans l'Algorithme 3.

Data: Ensemble de points 3D appariés : $\{X_i^t\}_{1\leq i\leq N}$ et $\{X_i^{t+1}\}_{1\leq i\leq N}$
Result: Rotation R et translation T entre les poses de la caméra à t et $t+1$
$\mu_Y = \frac{1}{N}\sum_{i=1}^{N} X_i^{t+1}$;
$\mu_X = \frac{1}{N}\sum_{i=1}^{N} X_i^{t}$;
$Y = [X_1^{t+1} - \mu_Y | ... | X_N^{t+1} - \mu_Y]$;
$X = [X_1^{t} - \mu_X | ... | X_N^{t} - \mu_X]$;
$[U, V, D] = SVD(YX^T)$;
$S = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \det(U)\det(V) \end{pmatrix}$;
$\hat{R} = USV^T$;
$\hat{T} = \mu_Y - R\mu_X$;

Algorithm 3: Algorithme de Calcul de Pose 3D-3D [Umeyama, 1991]

L'avantage de cette méthode est sa simplicité et sa rapidité. En effet, on dispose d'une formule explicite pour calculer l'optimum $\{R, T\}$ sans avoir à utiliser un algorithme itératif impliquant par exemple une descente de gradient comme c'est le cas pour le calcul de pose 2D-3D que nous verrons à la sous-section 3.1.2.

Cependant cette méthode minimise les moindres carrés de l'erreur 3D, et repose donc sur l'hypothèse que cette erreur est isotrope et suit une loi de probabilité invariante en fonction du point considéré (e.g. une loi gaussienne de la forme $\mathcal{N}\left(O_3, \sigma^2 Id_3\right)$). Or comme nous le montrons en Section 4.1, l'incertitude de triangulation stéréo est particulièrement anisotrope et variable selon la distance du point triangulé.

3.1.2 Calcul de pose 2D-3D

Les algorithmes de calcul de pose 2D-3D tel que celui proposé par Fischler et Bolles [Fischler and Bolles, 1981], minimisent un critère non pas dans l'espace en 3D mais en 2D dans l'image. Plus précisément, il s'agit d'un critère des moindres carrés sur l'erreur de reprojection dans l'image :

$$\mathcal{E}(R,T) = \frac{1}{N}\sum_{i=1}^{N} \left\| U_i^{t+1} - \Pi\Big(RX_i^t + T\Big) \right\|^2, \tag{3.2}$$

où $\Pi(.)$ est l'opérateur de projection dans l'image défini à l'équation (2.13) ; où $\{X_i^t\}_{1\leq i\leq N}$ désignent les coordonnées 3D des points remarquables extraits dans l'image gauche à t et triangulés ; et $\{U_i^{t+1}\}_{1\leq i\leq N}$ sont les positions dans l'image gauche à $t+1$ de ces mêmes points qui ont été suivis temporellement. La Figure 3.2 illustre cette approche.

Minimiser cette erreur de reprojection 2D revient à faire l'hypothèse que cette erreur suit une loi invariante isotrope dans le plan de l'image (e.g. une loi gaussienne

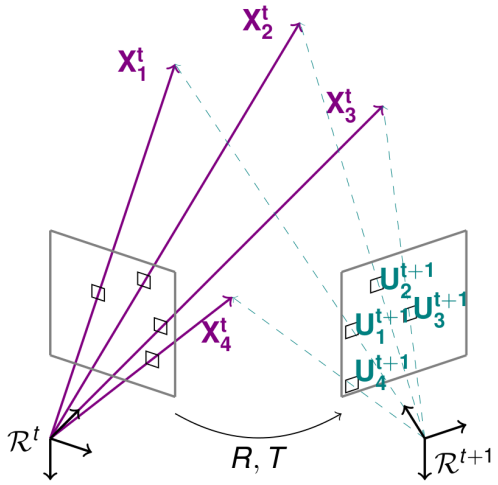

FIGURE 3.2 – Illustration du problème de calcul de pose 2D-3D pour estimer la position et l'orientation relative de la caméra gauche à l'instant $t+1$ par rapport à la caméra gauche à l'instant t.

du type $\mathcal{N}(0_2, Id_2)$). Au regard de la modélisation d'erreur présentée Section 4.1, cette hypothèse est inexacte (car la propagation de l'erreur de triangulation peut engendrer une erreur anisotrope sur le terme $\Pi(RX_i^t+T)$). Cependant cette méthode paraît plus adaptée que le calcul de pose 3D-3D, car les variations de l'erreur de triangulation en fonction de la position du point triangulé en 3D sont généralement beaucoup plus importantes que celles de l'erreur de reprojection en 2D d'un point triangulé (surtout lorsque le déplacement de la caméra est relativement petit).

3.1.3 Algorithme de calcul de pose utilisé

L'algorithme de calcul de pose choisi par la suite est celui utilisé dans eVO [Sanfourche et al., 2013] (*embedded Visual Odometer*), un système d'odométrie stéréo rapide et précis qui figurait 1er du classement du benchmark KITTI [Geiger et al., 2012] pour l'odométrie visuelle en 2013. Il s'agit d'une implémentation du calcul de pose 2D-3D basé sur l'approche de Fischler et Bolles [Fischler and Bolles, 1981], et dont nous décrivons ici brièvement les paramètres et détails d'implémentation.

Estimation des appariements 2D-3D

Puisqu'il s'agit d'une méthode 2D-3D, la première étape de cet algorithme de calcul de pose consiste à estimer ces appariements 2D-3D : $\{U_i^{t+1}, X_i^t\}_i$. Pour cela, les étapes suivantes sont successivement effectuées :

- extraction de N (e.g. 400) points remarquables dans l'image gauche à t, en utilisant le détecteur de coin *FAST* [Rosten and Drummond, 2006]. Ceci en veillant à ne pas avoir deux points FAST trop proches dans l'image (e.g. en utilisant un rayon d'exclusion de 10 pixels) et en forçant le détecteur à extraire

des points sur l'ensemble de l'image. On utilise une technique de "bucketing" consistant à quadriller l'image en plusieurs zones —e.g. 8x4 zones pour les images KITTI de taille 1200x400— et à imposer un nombre minimum de points extraits dans chaque zone.

- appariement stéréo des points remarquables, en cherchant leur pixel correspondant sur la même ligne dans l'image droite. Pour améliorer la robustesse de l'appariement stéréo, on effectue une vérification "aller-retour" dont le principe est décrit à la sous-section 2.2.1 (cf. paragraphe présentant l'algorithme ELAS).
- triangulation des points qui ont été correctement appariés en stéréo. On obtient les coordonnées 3D dans le repère caméra gauche à l'instant t : $\{X_i^t\}_{1\leq i\leq N}$.
- suivis temporel des points remarquables de l'image gauche à l'instant t, dans l'image gauche à $t+1$ en utilisant la méthode "Kanade-Lucas-Tomasi" (KLT) [Shi et al., 1994] avec l'implémentation pyramidale [Bouguet, 2001] disponible dans la bibliothèque OpenCV [Bradski, 2000]. On obtient les coordonnées 2D dans le repère image à l'instant $t+1$: $\{U_i^{t+1}\}_{1\leq i\leq N}$.

Calcul de pose robuste

Pour estimer la pose de manière robuste à partir des appariements $\{U_i^{t+1}, X_i^t\}_{1\leq i\leq N}$, on utilise d'abord une procédure *Random sample consensus* (RANSAC) de Fischler et Bolles [Fischler and Bolles, 1981] qui retourne une solution initiale et un ensemble d'*inliers* (i.e. d'appariements 2D-3D cohérents avec la solution trouvée). La méthode RANSAC consiste ici à effectuer de nombreux tirages aléatoires de triplets d'appariements 2D-3D. Á partir de chacun de ces triplets, on calcule la pose avec un algorithme à 3 points (P3P) [Fischler and Bolles, 1981]. L'algorithme P3P pouvant renvoyer plusieurs solutions, on intègre chaque solution dans la liste des solutions possibles construites avec tous les tirages aléatoires RANSAC. Finalement, la solution retenue est celle qui obtient le meilleur consensus, c'est-à-dire le plus grand nombre d'*inliers*. En sortie de cette procédure, on dispose d'un ensemble d'*inliers* que nous notons $\{U_i^{t+1}, X_i^t\}_{1\leq i\leq K}$, et d'une solution initiale paramétrée par $\Theta = (t_x, t_y, t_z, \theta_x, \theta_y, \theta_z)$: les composantes xyz de la translation T et les angles d'Eulers de la rotation R qui s'écrit suivant l'équation (3.3).

$$R_{(\theta_x,\theta_y,\theta_z)} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_x) & \sin(\theta_x) \\ 0 & -\sin(\theta_x) & \cos(\theta_x) \end{pmatrix} \begin{pmatrix} \cos(\theta_y) & 0 & -\sin(\theta_y) \\ 0 & 1 & 0 \\ \sin(\theta_y) & 0 & \cos(\theta_y) \end{pmatrix} \begin{pmatrix} \cos(\theta_z) & \sin(\theta_z) & 0 \\ -\sin(\theta_z) & \cos(\theta_z) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.3}$$

Cette solution initiale de pose, est ensuite raffinée en utilisant l'algorithme de Levenberg-Marquardt pour résoudre le problème d'optimisation non-linéaire consistant à minimiser les moindres carrés de l'erreur de reprojection dans l'image des *inliers* :

$$\text{minimiser} : \mathcal{E}(\Theta) = \frac{1}{K}\sum_{i=1}^{K} \left\| U_i^{t+1} - \Pi\Big(R_{(\Theta)} X_i^t + T_{(\Theta)}\Big) \right\|^2 \tag{3.4}$$

Temps de calcul

L'algorithme de calcul de pose décrit précedemment, s'exécute en 50ms avec un CPU Intel Core i7 sur les séquences KITTI, en utilisant 400 points FAST. Ce qui est compatible avec une utilisation en ligne sur plate-forme robotique.

3.2 Estimation du mouvement apparent dans l'image : le Flot Optique

Les environnements dynamiques comportent des éléments en mouvement qu'il est nécessaire de pouvoir identifier et caractériser. Pour cela il faut tout d'abord pouvoir estimer leur mouvement en les suivant temporellement d'une image à une autre. Puisque l'on souhaite couvrir toute l'image, c'est le déplacement de chaque pixel de l'image que l'on cherche à estimer entre deux prises de vue consécutives. Ce champ de mouvements 2D apparents est appelé *Flot Optique*.

L'estimation du flot optique dont la formulation générale est donnée sous-section 3.2.1, est un problème ancien en vision par ordinateur. Les travaux pionniers de Lucas et Kanade [Lucas et al., 1981] et de Horn et Schunck [Horn and Schunck, 1981], ont été respectivement à l'origine des deux grandes familles d'algorithmes d'estimation du flot optique : les approches globales et les approches locales que nous présentons respectivement aux sous-sections 3.2.2 et 3.2.3. Comparé à la masse d'articles publiés sur le sujet du flot optique, relativement peu de travaux portent sur l'estimation rapide du flot optique. Dans ce travail nous allons utiliser FOLKI [Plyer et al., 2014], un algorithme rapide fondé sur une approche locale. FOLKI est l'un des rares algorithmes dont la vitesse de calcul est compatible avec une application temps-réel et embarquée. Cependant FOLKI, comme d'autres algorithmes par fenêtres locales, a tendance à dégrader l'estimation du flot optique au niveau des bords des objets. En effet, l'estimation s'effectue en calculant des moyennes sur des fenêtres carrées au voisinage de chaque pixel, donc sur le bord des objets le flot estimé peut être un mélange du flot correspondant au premier plan et à l'arrière plan. Cela génère ce que nous avons appelé un "effet d'adhérence" en sous-section 2.2.1. Dans la sous-section 3.2.4 qui se focalise sur FOLKI, nous explorons plusieurs pistes d'amélioration de cet algorithme notamment en cherchant à atténuer l'effet d'adhérence.

3.2.1 Formulation du problème d'estimation du flot optique

Le flot optique $\mathbf{u} : (x, y) \rightarrow \Big(u(x, y), v(x, y)\Big)$ est un champ vectoriel bidimensionnel qui correspond au déplacement des pixels d'une image à une autre. Notons I_0 et I_1 deux images en niveaux de gris, qui peuvent correspondre à deux images successives d'une scène en mouvement. En supposant la conservation de l'intensité d'un pixel (x, y) de l'image I_0, suivi dans l'image I_1, on obtient une contrainte qui

doit être satisfaite par le flot optique :

$$I_0(x,y) = I_1\Big(x + u(x,y), y + v(x,y)\Big). \tag{3.5}$$

Equation de contrainte du flot optique (OFC)

Afin d'estimer ce champ $\mathbf{u}$, on procède souvent par itération en effectuant une linéarisation du type :

$$I_1\Big(x + u(x,y), y + v(x,y)\Big) \simeq I_1(x,y) + \nabla I_1(x,y)\mathbf{u}(x,y), \tag{3.6}$$

qui suppose donc que $\mathbf{u}(x,y)$ correspond à un "petit" déplacement. Si l'on combine les équations (3.5) et (3.6), en notant $I_t(x,y) = I_1(x,y) - I_0(x,y)$ et en oubliant les indices (x,y) pour alléger les notations, on obtient *l'équation de contrainte du flot optique* (OFC pour *Optical Flow Constraint*) :

$$I_t + \nabla I.\mathbf{u} = 0 \tag{3.7}$$

Problème d'ouverture

On constate que le problème d'estimation du flot optique ainsi formulé, est mal défini car on ne dispose que d'une seule équation OFC (3.7) pour deux inconnues $\mathbf{u} = (u,v)$. C'est ce que l'on appelle dans la littérature, le *problème d'ouverture*. Pour estimer le flot optique il faut donc ajouter une contrainte a priori sur le champ $\mathbf{u}$ qui prend la forme d'un terme de régularisation spatiale pour les méthodes variationnelles gloables (cf. sous-section 3.2.2), et pour les méthodes locales (cf. sous-section 3.2.3) se traduit par l'hypothèse que $\mathbf{u}$ est localement constant sur un voisinage $\mathcal{N}$.

Hypothèses sous-jacentes à l'écriture de l'OFC

Indépendamment de la méthode d'estimation du flot optique, on remarque que deux hypothèses importantes ont été faites pour écrire l'équation OFC (3.7) : la conservation de l'intensité entre I_0 et I_1, et l'hypothèse que $\mathbf{u}(x,y)$ correspond à un déplacement infinitésimal (permettant une linéarisation de la forme (3.6)). Or en pratique, ces hypothèses peuvent être mises en défaut : des changements d'illumination peuvent survenir entre deux prises de vue (e.g. avec des caméras à gain ajustable et diaphragme variable), et le mouvement apparent $\mathbf{u}$ dans l'image peut être très important en présence d'objets mobiles et selon le déplacement de la caméra (e.g. déplacement rapide, ou rotation importante).

Pour être robuste aux changements d'illumination, une technique usuelle est de considérer non pas l'image, mais une transformée de celle-ci par exemple en appliquant un filtre de Sobel ou de Rang [Zabih and Woodfill, 1994], ou encore en considérant comme Vaudrey et al. [Vaudrey et al., 2011] l'image résiduelle TV-L2 (cf. Figure 3.3 illustrant ces différentes transformations robustes aux changements

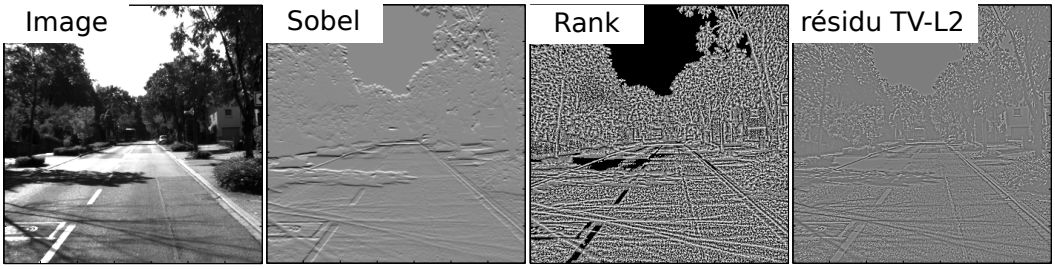

FIGURE 3.3 – Exemples de transformations d'image permettant de rendre les traitements robustes aux changements d'illumination ; de gauche à droite : image initiale, filtrage de Sobel, filtrage de Rang, image résiduelle TV-L2.

d'illumination).

Quant aux grands déplacements, ils peuvent être pris en compte par l'estimation successive du flot optique dans une pyramide d'image (cf. Figure 2.8, sous-section 2.2.1), car aux échelles élevées le mouvement apparent dans l'image devient petit. Le principe des approches pyramidales est le même pour les algorithmes de flot optique et pour les algorithmes de stéréo (cf. sous-section 2.2.1) : on estime successivement le flot optique en partant de l'échelle la plus haute (i.e. image à très faible résolution spatiale) et en propagant le flot estimé (sur-échantillonné et multiplié par le facteur d'échelle) comme initialisation pour l'estimation du flot optique à l'échelle d'en dessous. L'algorithme 4 résume le schéma d'estimation du flot optique par approche pyramidale en utilisant N niveaux d'échelle.

Data: Images : I_0 et I_1
Result: Flot optique $\mathbf{u} = \mathbf{u}_1$
% Construction récursive des pyramides d'images;
$pyrI_0 = \text{pyramide}(I_0)$;
$pyrI_1 = \text{pyramide}(I_1)$;
% Initialization;
$\mathbf{u}_{N+1} = 0$;
for *n=N :-1 :1* **do**
  $\mathbf{u}_n^0 = 2 \times \mathbf{u}_{n+1} \uparrow_2$;
  calculer (itérativement) $\mathbf{u}_n$ à partir de $pyrI_0(n)$, $pyrI_1(n)$ et $\mathbf{u}_n^0$;
end

Algorithm 4: Pseudo-code de l'estimation du flot optique avec pyramides d'images

Depuis les années 90 la très grande majorité des algorithmes d'estimation du flot optique utilisent une approche pyramidale pour estimer les grands mouvements. On peut cependant souligner quelques approches récentes qui évitent ce procédé [Weinzaepfel et al., 2013, Revaud et al., 2015] et les erreurs qu'il entraîne, notam-

ment la perte des mouvements rapides d'objets fins. Par la suite nous adoptons l'approche pyramidale et ce point ne sera plus mentionné, pour nous focaliser davantage sur les modèles et équations sur lesquels reposent les différents algorithmes d'estimation du flot optique.

3.2.2 Méthodes variationnelles globales d'estimation du flot optique

Principe

Les méthodes variationnelles d'estimation du flot optique, minimisent une énergie globale sur l'ensemble des pixels $\mathbf{x} = (x, y) \in \Omega$ de l'image, de la forme :

$$\mathcal{E}(\mathbf{u}) = \int_{\Omega} \mathcal{E}_{obs}(\mathbf{u}, \mathbf{x}) + \alpha \mathcal{E}_{regu}(\mathbf{u}, \mathbf{x}) \mathrm{d}\mathbf{x}, \tag{3.8}$$

où $\mathcal{E}_{obs}(\mathbf{u}, \mathbf{x})$ est le terme d'attache aux données qui assure généralement le respect de la contrainte de conservation de l'intensité (cf. Eq. (3.5)) ou bien de sa forme linéarisée (cf. Eq. (3.6)), et où $\mathcal{E}_{regu}(\mathbf{u}, \mathbf{x})$ est un terme de régularisation spatiale pondéré par un paramètre $\alpha > 0$.

Pour minimiser une telle énergie $\mathcal{E}(\mathbf{u})$, on applique la théorie sur le calcul variationnel en écrivant les équations d'Euler-Lagrange dont la résolution donne le flot optique estimé $\mathbf{u}^* = \arg\min_{\mathbf{u}} \mathcal{E}(\mathbf{u})$.

La méthode de Horn et Schunck

Les premiers travaux sur le sujet remontent à Horn et Schunck [Horn and Schunck, 1981], qui considèrent la somme des normes au carré des gradients des composantes u et v du flot optique comme terme de régularisation spatiale :

$$\begin{cases} \mathcal{E}_{obs}(\mathbf{u}, \mathbf{x}) = \|I_t(\mathbf{x}) + \nabla I_1(\mathbf{x}) \mathbf{u}(\mathbf{x})\|^2 \\ \mathcal{E}_{regu}(\mathbf{u}, \mathbf{x}) = \|\nabla u(\mathbf{x})\|^2 + \|\nabla v(\mathbf{x})\|^2, \end{cases} \tag{3.9}$$

en utilisant l'approximation numérique suivante pour le calcul du gradient : $\nabla u(\mathbf{x}) \approx (\bar{u}(\mathbf{x}) - u(\mathbf{x}))$, avec $\bar{u}(\mathbf{x})$ la moyenne locale de la composante u.

Dans ce cas, les équations d'Euler-Lagrange s'écrivent :

$$\begin{cases} I_x \left(I_t + I_x u + I_y v\right) + \alpha(u - \bar{u}) = 0 \\ I_y \left(I_t + I_x u + I_y v\right) + \alpha(v - \bar{v}) = 0, \end{cases} \tag{3.10}$$

avec les raccourcis : $I_x(x, y) \equiv \frac{\partial I_1}{\partial x}(x, y)$ et $I_y(x, y) \equiv \frac{\partial I_1}{\partial y}(x, y)$, et avec :

$$\begin{aligned} \bar{u}(x, y) = & \frac{1}{6}\Big(u(x-1, y) + u(x+1, y) + u(x, y-1) + u(x, y+1)\Big) + \\ & \frac{1}{12}\Big(u(x-1, y-1) + u(x+1, y-1) + u(x+1, y-1) + u(x+1, y+1)\Big) \end{aligned} \tag{3.11}$$

Le système d'équation (3.10) équivaut au système suivant :

$$\begin{cases} \left(I_x^2 + I_y^2 + \alpha\right) u = \left(\alpha + I_y^2\right) \bar{u} - I_x I_y \bar{v} - I_x I_t \\ \left(I_x^2 + I_y^2 + \alpha\right) v = \left(\alpha + I_x^2\right) \bar{v} - I_x I_y \bar{u} - I_y I_t \end{cases} \tag{3.12}$$

Horn et Schunck proposent de résoudre itérativement le système d'équation résoudre (3.12) en estimant à l'itération $k+1$ le flot optique (u^{k+1}, v^{k+1}), à partir de $(\bar{u}^k, \bar{v}^k)$ calculé par une moyenne locale du flot précédemment estimé à l'itération k.

On remarque que grâce au terme de régularisation spatiale, avec $\alpha \neq 0$, le flot optique peut être estimé même dans les zones homogènes où $I_x = I_y = 0$.

La littérature comporte un grand nombre de travaux proposant des algorithmes variationnels issus de la méthode de Horn et Schunck, et de nombreuses alternatives ont été proposées pour les termes $\mathcal{E}_{obs}(\mathbf{u}, \mathbf{x})$ et $\mathcal{E}_{regu}(\mathbf{u}, \mathbf{x})$ considérés. Parmi les algorithmes les plus connus on peut citer TV-L1 qui minimise non pas la somme des carrés mais la somme des valeurs absolues des termes à minimiser :

$$\mathcal{E}(u, v) = \int_\Omega \lambda |I_1(\mathbf{x} + \mathbf{u}) - I_0(\mathbf{x})| + |\nabla u(\mathbf{x})| + |\nabla v(\mathbf{x})| \mathrm{d}\mathbf{x} \tag{3.13}$$

L'avantage de la norme L1 par rapport à la norme L2 est que l'estimation du flot optique est plus robuste et préserve mieux les discontinuités du flot optique aux bords des objets. Cependant elle complique l'optimisation car elle n'est pas deux fois dérivable : il faut alors utiliser une méthode duale comme celle proposée par Zach et al. [Zach et al., 2007], ou bien l'approximer comme dans l'algorithme *High Accuracy Optical Flow* (HAOF) [Brox et al., 2004] présenté ci-dessous.

L'algorithme *High Accuracy Optical Flow* (HAOF) [Brox et al., 2004] fait partie des algorithmes de référence pour l'estimation du flot optique. Cet algorithme utilise dans le terme $\mathcal{E}_{obs}$ non seulement la contrainte de conservation de l'intensité mais aussi la conservation des gradients de l'image afin d'être moins sensible aux changements d'illumination. Les auteurs utilisent également la fonction $\Phi(s^2) = \sqrt{s^2 + \varepsilon^2}$ (e.g. $\varepsilon = 0.001$) dans leurs termes $\mathcal{E}_{obs}$ et $\mathcal{E}_{regu}$ pour approximer la norme L1. L'énergie qu'ils minimisent s'écrit :

$$\begin{aligned} \mathcal{E}(u, v) = \int_\Omega & \Phi\left(\|I_1(\mathbf{x} + \mathbf{u}) - I_0(\mathbf{x})\|^2 + \gamma \|\nabla I_1(\mathbf{x} + \mathbf{u}) - \nabla I_0(\mathbf{x})\|^2\right) \\ & + \alpha \Phi\left(\|\nabla u(\mathbf{x})\|^2 + \|\nabla v(\mathbf{x})\|^2\right) \mathrm{d}\mathbf{x} \end{aligned} \tag{3.14}$$

Les équations d'Euler-Lagrange associées étant non-linéaires, une méthode d'approximation numérique itérative est appliquée pour se ramener à un système d'équations linéaires en les incréments (du, dv) qui est ensuite résolue par une méthode de Gauss-Seidel ou bien des itérations SOR.

Enfin, une des pistes de recherche importantes en dehors du choix des normes et des techniques de résolution utilisées, est l'intégration d'information additionnelle

apportée par des descripteurs extraits dans I_0 et I_1, et dont on effectue l'appariement. L'algorithme *Large Displacement Optical Flow* (LDOF) [Brox et al., 2009] utilise des appariements de descripteurs *Histogram of Gradient* (HoG) pour aider à estimer le flot optique notamment pour les grands déplacements et les structures fines dont le mouvement est difficilement estimé à cause de l'approche pyramidale d'estimation du flot optique. L'énergie minimisée s'écrit :

$$\mathcal{E}(\mathbf{u}) = \mathcal{E}_{color}(\mathbf{u}) + \gamma\mathcal{E}_{grad}(\mathbf{u}) + \alpha\mathcal{E}_{smooth}(\mathbf{u}) + \beta\mathcal{E}_{match}(\mathbf{u}, \mathbf{u}_1) + \mathcal{E}_{desc}(\mathbf{u}_1) \quad (3.15)$$

Les trois premiers termes semblables à HAOF, correspondent aux contraintes de conservation de l'intensité et des gradients, et au terme de régularisation spatiale :

$$\begin{cases} \mathcal{E}_{color}(\mathbf{u}) = \int_\Omega \Phi\left(\|I_1(\mathbf{x}+\mathbf{u}) - I_0(\mathbf{x})\|^2\right) \mathrm{d}\mathbf{x} \\ \mathcal{E}_{grad}(\mathbf{u}) = \int_\Omega \Phi\left(\|\nabla I_1(\mathbf{x}+\mathbf{u}) - \nabla I_0(\mathbf{x})\|^2\right) \mathrm{d}\mathbf{x} \\ \mathcal{E}_{smooth}(\mathbf{u}) = \int_\Omega \Phi\left(\|\nabla u(\mathbf{x})\|^2 + \|\nabla v(\mathbf{x})\|^2\right) \mathrm{d}\mathbf{x} \end{cases} \quad (3.16)$$

Les deux derniers termes ajoutent l'information additionnelle apportée par la mise en correspondance de descripteurs HoG (*Histogram of Gradient*) par l'intermédiaire d'un flot $\mathbf{u}_1$ dit "épars" défini uniquement en certains pixels :

$$\begin{cases} \mathcal{E}_{match}(\mathbf{u}, \mathbf{u}_1) = \int_\Omega \delta(\mathbf{x})\rho(\mathbf{x})\Phi\left(\|u(\mathbf{x}) - u_1(\mathbf{x})\|^2 + \|v(\mathbf{x}) - v_1(\mathbf{x})\|^2\right) \mathrm{d}\mathbf{x} \\ \mathcal{E}_{desc}(\mathbf{u}_1) = \int_\Omega \Phi\left(\|\mathbf{f}_1(\mathbf{x}+\mathbf{u}_1) - \mathbf{f}_0(\mathbf{x})\|^2\right) \mathrm{d}\mathbf{x} \end{cases} \quad (3.17)$$

où $\mathbf{f}_0(\mathbf{x})$ (respectivement $\mathbf{f}_1(\mathbf{x})$) désigne le vecteur descripteur HoG calculé en un pixel $\mathbf{x}$ de l'image I_0 (resp. I_1), $\delta(\mathbf{x})$ est la fonction valant 1 s'il y a un descripteur en $\mathbf{x}$ dans l'image I_0 et 0 sinon, et $\rho(\mathbf{x})$ est le score de correspondance des descripteurs $\mathbf{x}$ dans I_0 et $(\mathbf{x}+\mathbf{u}_1)$ dans I_1.

Pour optimiser $\mathcal{E}(\mathbf{u})$ les auteurs minimisent séparément $\mathcal{E}_{desc}(\mathbf{u}_1)$ puis la somme des autres termes. A chaque étage k de la pyramide d'images, les descripteurs HoG sont extraits dans I_0^k et I_1^k et mis en correspondance (ce qui minimise $\mathcal{E}_{desc}(\mathbf{u}_1)$), puis l'ensemble des termes restants de $\mathcal{E}(\mathbf{u})$ est minimisé suivant la même approche que HAOF.

Avantages et inconvénients des méthodes variationnelles

Un des avantages de l'approche variationnelle est la grande liberté concernant le choix des termes intégrés dans l'énergie $\mathcal{E}(\mathbf{u})$, qui peuvent prendre n'importe quelle forme tant que l'on peut dériver et résoudre des équations d'Euler-Lagrange associées. Nous avons vu précédemment trois alternatives à l'énergie $\mathcal{E}(\mathbf{u})$ considérée par Horn et Schunck (cf. Eq. (3.8) et (3.9)), qui font partie des références dans la littérature : TV-L1, HAOF et LDOF. Mais il existe de nombreuses autres formulations de $\mathcal{E}(\mathbf{u})$, notament *DeepFlow* [Weinzaepfel et al., 2013] et *Non-Local Total Generalized Variation* (NLTGV) [Ranftl et al., 2014] qui font partie des algorithmes de l'état de l'art pour l'estimation du flot optique.

Si ces méthodes sont précises elles sont en revanche coûteuses en temps de calcul. En effet elles impliquent une optimisation itérative, et pour chaque itération il faut encore une boucle d'itérations internes (voires plusieurs dans le cas de HAOF) pour résoudre les équations aux dérivées partielles. De plus, comme cela est montré dans les travaux de Plyer et al [Plyer et al., 2014] de nombreuses itérations peuvent être nécessaires pour converger.

3.2.3 Méthodes locales d'estimation du flot optique

Principe des méthodes locales

Contrairement aux méthodes variationnelles qui minimisent une énergie globale sur l'ensemble Ω des pixels de l'image, les méthodes locales (aussi appelées "par fenêtres locales") estiment $\mathbf{u}(\mathbf{x})$ pour chaque pixel en tenant compte uniquement du voisinage de $\mathbf{x}$. Les premiers travaux dans ce domaine remontent à Lucas et Kanade [Lucas et al., 1981] et consistent à définir $\mathbf{u}(\mathbf{x})$ comme le minimum d'une énergie définie sur une fenêtre locale $\mathcal{N}(\mathbf{x})$, de taille N x N, autour de $\mathbf{x}$:

$$\mathcal{E}(\mathbf{u}(\mathbf{x})) = \sum_{\mathbf{x}' \in \mathcal{N}(\mathbf{x})} \left(I_1(\mathbf{x}' + \mathbf{u}(\mathbf{x})) - I_0(\mathbf{x}')\right)^2. \tag{3.18}$$

Cette énergie est minimisée itérativement par un algorithme de Gauss-Newton, en utilisant une formule de Taylor pour linéariser le terme $I_1(\mathbf{x}' + \mathbf{u}(\mathbf{x}))$ et se ramener à un problème des moindres carrés. Différentes façons de linéariser et de résoudre $\mathcal{E}(\mathbf{u}(\mathbf{x}))$ existent dans la littérature.

Différents schémas itératifs pour estimer le flot optique

Méthode Lucas-Kanade. Dans la méthode originale, Lucas et Kanade considèrent le dévelopement limité à l'ordre 1 de $I_1(.)$ autour de $\mathbf{x}' + \mathbf{u}(\mathbf{x})$:

$$I_1\left(\mathbf{x}' + \mathbf{u}(\mathbf{x}) + \delta\mathbf{u}(\mathbf{x})\right) \approx I_1\left(\mathbf{x}' + \mathbf{u}(\mathbf{x})\right) + \nabla I_1\left(\mathbf{x}' + \mathbf{u}(\mathbf{x})\right)^T \delta\mathbf{u}(\mathbf{x}) \tag{3.19}$$

En adoptant une approche itérative et en initialisant $\mathbf{u}^0(\mathbf{x}) = 0$, cela conduit à l'itération $k+1$ à estimer $\mathbf{u}^{k+1}(\mathbf{x}) = \mathbf{u}^k(\mathbf{x}) + \delta\mathbf{u}(\mathbf{x})$ par la résolution du problème des moindres carrés suivant :

$$\min_{\mathbf{u}(\mathbf{x})} \sum_{\mathbf{x}' \in \mathcal{N}(\mathbf{x})} \left(I_1\left(\mathbf{x}' + \mathbf{u}^k(\mathbf{x})\right) + \nabla I_1\left(\mathbf{x}' + \mathbf{u}^k(\mathbf{x})\right)^T \left(\mathbf{u}(\mathbf{x}) - \mathbf{u}^k(\mathbf{x})\right) - I_0(\mathbf{x}')\right)^2 \tag{3.20}$$

Ce schéma de résolution est appliqué itérativement et de façon multi-échelle à travers l'utilisation d'une pyramide d'images.

Modèle additif inverse. Dans leur algorithme *additif inverse*, Baker et Matthews [Baker and Matthews, 2004] proposent de remplacer $\nabla I_1(\mathbf{x}' + \mathbf{u}^k(\mathbf{x}))$ par $\nabla I_0(\mathbf{x}')$

ce qui réduit significativement le temps de calcul et conduit à calculer à l'itération $k+1$ le flot optique $\mathbf{u}^{k+1}(\mathbf{x})$ de la façon suivante :

$$\min_{\mathbf{u}(\mathbf{x})} \sum_{\mathbf{x}' \in \mathcal{N}(\mathbf{x})} \left(I_1\left(\mathbf{x}' + \mathbf{u}^k(\mathbf{x})\right) + \nabla I_0\left(\mathbf{x}'\right)^T \left(\mathbf{u}(\mathbf{x}) - \mathbf{u}^k(\mathbf{x})\right) - I_0(\mathbf{x}') \right)^2 \tag{3.21}$$

FOLKI. Allant encore plus loin dans la réduction du nombre d'opérations, Le Besnerais et Champagnat [Le Besnerais and Champagnat, 2005] proposent d'effectuer le développement limité à l'ordre 1 au voisinage de $\mathbf{x}' + \mathbf{u}^k(\mathbf{x}')$ de l'expression :

$$I_1\left(\mathbf{x}' + \mathbf{u}^k(\mathbf{x}) + \delta\mathbf{u}(\mathbf{x})\right) = I_1\left(\mathbf{x}' + \mathbf{u}^k(\mathbf{x}') - \mathbf{u}^k(\mathbf{x}') + \mathbf{u}^k(\mathbf{x}) + \delta\mathbf{u}(\mathbf{x})\right), \tag{3.22}$$

ce qui conduit à écrire, en adoptant le modèle additif inverse de Baker et al :

$$I_1\left(\mathbf{x}' + \mathbf{u}^k(\mathbf{x}) + \delta\mathbf{u}(\mathbf{x})\right) \approx I_1\left(\mathbf{x}' + \mathbf{u}^k(\mathbf{x}'))\right) + \nabla I_0(\mathbf{x}')^T \left(\mathbf{u}^k(\mathbf{x}) + \delta\mathbf{u}(\mathbf{x}) - \mathbf{u}^k(\mathbf{x}')\right) \tag{3.23}$$

Ainsi $\mathbf{u}^{k+1}(\mathbf{x}) = \mathbf{u}^k(\mathbf{x}) + \delta\mathbf{u}(\mathbf{x})$ est calculé en minimisant le critère des moindres carrés suivant :

$$\min_{\mathbf{u}(\mathbf{x})} \sum_{\mathbf{x}' \in \mathcal{N}(\mathbf{x})} \left(I_1\left(\mathbf{x}' + \mathbf{u}^k(\mathbf{x}')\right) + \nabla I_0(\mathbf{x}')^T \left(\mathbf{u}(\mathbf{x}) - \mathbf{u}^k(\mathbf{x}')\right) - I_0(\mathbf{x}') \right)^2 \tag{3.24}$$

Cet algorithme nommé FOLKI (Flot Optique Lucas-Kanade Iteratif) [Plyer et al., 2014] est extrêment rapide car à chaque itération $k+1$, il nécessite seulement de calculer :

$$\forall \mathbf{x} \in \Omega : \; I_1\left(\mathbf{x} + \mathbf{u}^k(\mathbf{x})\right),$$

alors que l'algorithme additif inverse de Baker et al nécessite de calculer :

$$\forall \mathbf{x} \in \Omega, \forall \mathbf{x}' \in \mathcal{N}(\mathbf{x}) : \; I_1\left(\mathbf{x}' + \mathbf{u}^k(\mathbf{x})\right),$$

et l'algorithme de Lucas-Kanade nécessite de calculer :

$$\forall \mathbf{x} \in \Omega, \forall \mathbf{x}' \in \mathcal{N}(\mathbf{x}) : \; I_1\left(\mathbf{x}' + \mathbf{u}^k(\mathbf{x})\right) \text{et} \nabla I_1\left(\mathbf{x}' + \mathbf{u}^k(\mathbf{x})\right).$$

Il est à noter que dans l'extension de FOLKI proposée dans [Plyer et al., 2014] et nommée eFOLKI, plusieurs tailles de fenêtres carrées $(2r+1)\times(2r+1)$ sont utilisées afin d'effectuer l'estimation d'abord sur des grandes fenêtres (e.g. $r = 8$) puis sur des fenêtres de tailles réduites (e.g. $r = 4$). Notons que pour améliorer la robustesse aux changements d'illumination, FOLKI applique aussi une transformée de Rang sur les images avant de réaliser l'estimation du flot. Nous détaillerons ce prétraitement, qui a déjà été présenté en Figure 3.3, dans la sous-section 3.2.5. En pratique

on effectue à chaque niveau d'échelle n, les opérations indiquées dans l'algorithme 5.

Data: initialisation : $\mathbf{u}_n^0 = 2 \times \mathbf{u}_{n+1} \uparrow_2$
Result: Flot optique estimé à l'échelle n : $\mathbf{u}_n = \mathbf{u}_n^K$
for $r \in \{8; 4\}$ **do**
  for *k=0 :K-1* **do**
    calculer $\mathbf{u}_n^{k+1}$ minimisant l'équation (3.24) en prenant $\mathcal{N}$ de taille $(2r+1) \times (2r+1)$;
  end
end
Algorithm 5: Pseudo-code détaillant l'estimation itérative multi-fenêtres

Plyer et al [Plyer et al., 2014] montrent que non seulement cette version robuste et multi-fenêtres de FOLKI est plus rapide que pyramidLK (l'implémentation OpenCV de Bouguet [Bouguet, 2001] du modèle additif inverse), mais également meilleure en termes de précision et de robustesse du flot optique estimé. C'est donc cette méthode itérative FOLKI que nous considérons par la suite.

Pondération des termes dans $\mathcal{E}(\mathbf{u})$

Plutôt que de considérer de la même manière les contributions de chaque terme en $\mathbf{x}' \in \mathcal{N}(\mathbf{x})$ dans l'équation (3.18), on peut considérer sans surcoût de calcul une fonction de pondération séparable $\omega(\mathbf{x}'-\mathbf{x})$ (par exemple une gaussienne de support limité à $\mathcal{N}(\mathbf{x})$) :

$$\mathcal{E}(\mathbf{u}(\mathbf{x})) = \sum_{\mathbf{x}' \in \mathcal{N}(\mathbf{x})} \omega\left(\mathbf{x}' - \mathbf{x}\right) \left(I_1(\mathbf{x}' + \mathbf{u}(\mathbf{x})) - I_0(\mathbf{x}')\right)^2. \tag{3.25}$$

En pratique nous avons constaté que l'utilisation d'une fonction gaussienne pour $\omega(\mathbf{x}' - \mathbf{x})$, à la place d'une pondération uniforme n'apporte pas de gain sur la précision du flot optique estimé.

Un autre type de pondération est proposé par Simoncelli [Simoncelli, 1999] qui formule le problème d'estimation du flot optique comme la recherche d'un maximum a posteriori en modélisant l'incertitude des termes δI_t et $\delta\mathbf{u}$ dans l'équation de contrainte du flot optique, par des lois gaussiennes de moyenne nulle et de covariances valant λ_2 et la matrice scalaire $\lambda_1 Id_2$:

$$I_t + \delta I_t + \nabla I^T (\mathbf{u} + \delta\mathbf{u}) = 0, \tag{3.26}$$

avec

$$\begin{cases} \delta\mathbf{u} \sim \mathcal{N}(0_2, \lambda_1 Id_2) \\ \delta I_t \sim \mathcal{N}(0, \lambda_2) \end{cases} \tag{3.27}$$

Ce qui conduit à la probabilité conditionnelle de I_t :

$$\mathbb{P}(I_t|\nabla I, \mathbf{u}) \propto \exp\left(-\frac{1}{2}\left(I_t + \nabla I^T\mathbf{u}\right)\left(\nabla I^T \lambda_1 Id_2 \nabla I + \lambda_2\right)^{-1}\left(I_t + \nabla I^T\mathbf{u}\right)\right) \tag{3.28}$$

En appliquant la formule de Bayes et en considérant : $\mathbb{P}(\mathbf{u}) \sim \mathcal{N}(0_2, \Lambda_p)$ comme a priori, on peut estimer le maximum a posteriori de $\mathbf{u}$ sachant ∇I et I_t à partir de :

$$\begin{aligned}\mathbb{P}(\mathbf{u}|\nabla I, I_t) &= \frac{\mathbb{P}(I_t|\nabla I, \mathbf{u})\mathbb{P}(\mathbf{u})}{\mathbb{P}(I_t)}, \\ &\propto \mathbb{P}(I_t|\nabla I, \mathbf{u})\mathbb{P}(\mathbf{u}), \\ &\propto \exp\left(-\frac{1}{2}\left(\mu_{\mathbf{u}} - \mathbf{u}\right)^T \Lambda_{\mathbf{u}}^{-1}\left(\mu_{\mathbf{u}} - \mathbf{u}\right)\right)\end{aligned} \tag{3.29}$$

où :

$$\begin{cases} \Lambda_{\mathbf{u}} = \left(\dfrac{\nabla I \nabla I^T}{\lambda_1 \|\nabla I\|^2 + \lambda_2} + \Lambda_p^{-1}\right)^{-1} \\ \mu_{\mathbf{u}} = -\Lambda_{\mathbf{u}} \dfrac{I_t}{\lambda_1 \|\nabla I\|^2 + \lambda_2} \nabla I \end{cases} \tag{3.30}$$

En supposant les incertitudes indépendantes en chaque pixel $\mathbf{x}'$ du voisinage $\mathcal{N}(\mathbf{x})$, Simoncelli écrit la formule :

$$\mathbf{u}_{\text{MAP}} = -\left(\sum_{\mathbf{x}' \in \mathcal{N}(\mathbf{x})} \frac{\nabla I \nabla I^T}{\lambda_1 \|\nabla I\|^2 + \lambda_2} + \Lambda_p^{-1}\right)^{-1} \sum_{\mathbf{x}' \in \mathcal{N}(\mathbf{x})} \frac{I_t}{\lambda_1 \|\nabla I\|^2 + \lambda_2} \nabla I \tag{3.31}$$

En réécrivant cette formule dans le contexte de la méthode itérative FOLKI (cf. formule (3.24)), on obtient :

$$\mathbf{u}^{k+1}(\mathbf{x}) = -\left(\sum_{\mathbf{x}' \in \mathcal{N}(\mathbf{x})} \frac{\nabla I_0(\mathbf{x}') \nabla I_0(\mathbf{x}')^T}{\lambda_1 \|\nabla I_0(\mathbf{x}')\|^2 + \lambda_2} + \Lambda_p^{-1}\right)^{-1} \sum_{\mathbf{x}' \in \mathcal{N}(\mathbf{x})} \frac{I_t\left(\mathbf{x}', \mathbf{u}^k(\mathbf{x}')\right)}{\lambda_1 \|\nabla I_0(\mathbf{x}')\|^2 + \lambda_2} \nabla I_0(\mathbf{x}'), \tag{3.32}$$

où

$$I_t\left(\mathbf{x}', \mathbf{u}^k(\mathbf{x}')\right) = I_1(\mathbf{x}' + \mathbf{u}^k(\mathbf{x}')) - \nabla I_0(\mathbf{x}')^T \mathbf{u}^k(\mathbf{x}') - I_0(\mathbf{x}') \tag{3.33}$$

Ainsi, si l'on considère l'estimateur de flot optique FOLKI et que l'on néglige le terme Λ_p, l'approche de Simoncelli équivaut à appliquer la pondération :

$$\omega(\mathbf{x}') = \frac{1}{\lambda_1 \|\nabla I_0(\mathbf{x}')\|^2 + \lambda_2} \tag{3.34}$$

Dans le modèle de Simoncelli, le terme λ_1 représente l'incertitude sur le flot optique propagé d'une échelle à une autre. L'influence de cette incertitude est plus grande dans les régions de forts gradient car l'erreur $\delta\mathbf{u}$ est multipliée par ∇I^T dans l'équation (3.26). Quant au terme λ_2, il représente l'incertitude sur la conservation d'intensité (niveau de gris) d'un même point, d'une image à une autre, et affecte donc tous les pixels de manière égale. Cela explique le terme en $\lambda_1 \|\nabla I_0(\mathbf{x}')\|^2$ dans l'équation (3.34).

Dans la sous-section 3.2.5 nous testons l'utilisation de cette pondération afin d'étudier le gain en précision qu'elle peut apporter.

Avantages et inconvénients des approches par fenêtres locales

L'avantage principal des méthodes locales d'estimation du flot optique est leur rapidité. D'une part, elles sont très parallélisables et conviennent parfaitement à une implémentation sur GPU qui permet une augmentation très importante de leur vitesse d'exécution. D'autre part, ces méthodes locales qui utilisent l'algorithme de Gauss-Newton, convergent beaucoup plus rapidement que celles utilisant un schéma d'Euler-Lagrange et une seconde boucle d'itérations internes.

Cependant ces méthodes reposent sur l'hypothèse que le flot optique est constant localement sur une fenêtre $\mathcal{N}(\mathbf{x})$ autour du pixel $\mathbf{x}$. Cette hypothèse est souvent fausse d'une part car les surfaces observées ne sont généralement pas fronto-paralèlles à la caméra, et d'autre part car sur les bords des objets, certains pixels de $\mathcal{N}(\mathbf{x})$ peuvent appartenir au fond tandis que d'autres peuvent appartenir à l'objet en premier plan. Ceci induit donc l'effet d'adhérence mentionné au début de la Section 3.2 et rend les méthodes locales moins précises que les méthodes variationnelles, comme le montre le classement de Middlebury [Baker et al., 2011] où les algorithmes pyramidLK et FOLKI figurent aux dernières positions. Dans la sous-section 3.2.5, nous considérons diverses variantes de FOLKI et essayons d'atténuer les effets d'adhérence.

3.2.4 Comparaison des approches gobales et locales

Dans les sous-sections 3.2.2 et 3.2.3 nous avons décrit successivement les principes de l'approche variationnelle et de l'approche par fenêtres locales pour l'estimation du flot optique. Les méthodes locales sont plus rapides que les méthodes variationnelles, mais sont moins précises en tout cas selon le classement de Middlebury [Baker et al., 2011]. Toutefois le jeu de données Middlebury est constitué de scènes en intérieur avec des conditions idéales d'illumination et avec des objets texturés, ce qui n'est pas représentatif du cas d'utilisation que l'on envisage, à savoir : traiter des images stéréo acquises depuis une plate-forme mobile en intérieur comme en extérieur pour permettre une navigation autonome. C'est pourquoi dans la suite nous préférons tester nos algorithmes plutôt sur le dataset KITTI [Geiger et al., 2012] dont les images ont été acquises en milieu urbain en condition réelles.

Evaluation des performances sur le dataset KITTI

Le dataset KITTI [Geiger et al., 2012] est constitué entre autres d'images acquises à une fréquence de 10Hz depuis un banc stéréo placé sur le toit d'une voiture circulant en environnement urbain. Comme pour la stéréo, la vérité terrain pour le flot optique est établie à partir de données acquises avec un scanner laser Velodyne et un système de localisation combinant un GPS et une centrale intertielle. La taille de ces images varie autour de 1230 x 790 pixels. Le critère principal du classement KITTI est le pourcentage d'ouliers (noté : *Out-*), qui est défini comme le pourcentage de pixels pour lesquels la différence entre le flot estimé et la vérité terrain est supérieure à 3 pixels. Ce critère est estimé sur l'ensemble des points de

la vérité terrain (noté : *-All*, utilisé pour le classement), ou seulement sur les points non occultés entre les deux images consécutives (*-Noc*). De même l'erreur moyenne est calculée sur tout ou partie des points de la vérité terrain : *Avg-All* ou *Avg-Noc*.

Les données KITTI sont difficiles à traiter pour les algorithmes d'estimation de flot optique car elles sont acquises en conditions réelles : le capteur de la caméra est parfois saturé en raison de l'illumination, les images présentent souvent des zones homogènes peu texturées (e.g. la route, ou les murs de certaines maisons), il y a de la réflection sur les fenêtres et les vitres des voitures... C'est donc la robustesse des algorithmes qui est essentiellement éprouvée. Sur ce point FOLKI réussit assez bien comme l'indique le Tableau 3.1 qui montre que FOLKI obtient de meilleures performances que des algorithmes variationnels comme HAOF, TV-L1 et même LDOF, en termes de taux d'outliers mais également concernant l'erreur moyenne. Néanmoins, FOLKI reste peu précis avec une erreur moyenne de 5 pixels dans l'estimation du flot optique, contre une erreur de 1.5 pixels pour des algorithmes comme NLTGV et DeepFlow.

TABLE 3.1 – Extrait du classement KITTI 2012 pour le flot optique.

Algorithme	Out-Noc	Avg-Noc	Temps	Environnement
NLTGV-SC	5.93 %	1.6 px	16 s	GPU @ 2.5 Ghz (Matlab + C/C++)
DeepFlow	7.22 %	1.5 px	17 s	1 core @ 3.6Ghz (Python + C/C++)
eFOLKI	19.31 %	5.2 px	0.026 s	GPU @ 700 Mhz (C/C++)
LDOF	21.93 %	5.6 px	1 min	1 core @ 2.5 Ghz (C/C++)
TV-L1	30.87 %	7.9 px	16 s	1 core @ 2.5 Ghz (Matlab)
HAOF	35.87 %	11.1 px	16.2 s	1 core @ 2.5 Ghz (C/C++)

Comparaison des temps de calcul

On voit sur le Tableau 3.1 que FOLKI affiche un temps de calcul beaucoup moins élevé que les autres algorithmes, et s'exécute à presque 40Hz sur les données KITTI. Cependant les temps indiqués sur le classement KITTI correspondent à tout type d'environnement de calcul comme l'indique la dernière colonne, ces temps ne sont donc pas directement comparables. Puisque l'on dispose notamment sur OpenCV [Bradski, 2000] des codes GPU pour HOAF et TV-L1, nous pouvons exécuter ces codes et FOLKI sur le même environnement de calcul sur notre machine équipée d'un GPU GeForce GTX Titan. Nous avons utilisé les paramètres par défaut dans OpenCV pour les algorithmes HAOF et TV-L1, et obtenu les performances et temps de calcul indiqués dans le Tableau 3.2. Ce tableau montre que FOLKI est de loin le plus rapide, et le seul pouvant s'exécuter à plus de 10 Hz (soit la cadence vidéo sur les séquences KITTI). Pourtant la paramétrisation par défaut d'OpenCV pour HAOF, utilise très peu d'itérations (77 itérations dans la boucle extérieur, 10 itéra-

tions dans la boucle intérieure et 10 itérations pour le solveur SOR). Or comme le montrent les travaux de Plyer et al. [Plyer et al., 2014], HAOF nécessite nettement plus d'itérations pour atteindre ses performances optimales (en prenant 300, 10 et 100 itérations pour la boucle extérieur, intérieure, et le solveur) ce qui nécessite alors 6.114 secondes de calcul pour HAOF. En pratique on constate que, contrairement à FOLKI et HAOF dont les temps de calcul sont constants sur l'ensemble des données KITTI (à 1 ou 2 ms près), le temps de TV-L1 fluctue énormément (du simple au double) d'une paire d'images à une autre. En effet, FOLKI et HAOF ont un nombre prédéfini d'itérations contrairement à TV-L1 qui s'exécute jusqu'à satisfaire une condition d'arrêt (critère de convergence).

TABLE 3.2 – Performances des algorithmes FOLKI, TV-L1 et HAOF, sur la base de données d'entraînement KITTI 2012 pour le flot optique.

Algorithme	Out-Noc	Avg-Noc	Temps	Environnement
FOLKI	19.21%	4.6px	0.027s	GeForce GTX Titan (GPU)
TV-L1	37.74%	9.3px	0.292s	GeForce GTX Titan (GPU)
HAOF	41.16%	9.0px	0.127s	GeForce GTX Titan (GPU)

Conclusion

Au vu des considérations précédentes, FOLKI constitue le choix le plus adapté pour servir de base à un système de perception temps-réel embarqué : il est robuste aux conditions difficiles d'illuminations comme le montre ses performances sur les données KITTI, et il s'agit surtout du seul algorithme suffisamment rapide pour être considéré dans une chaîne de traitement capable de tourner à cadence vidéo sur les séquences KITTI (10Hz).

Cependant comme on peut le voir sur le Tableau 3.1, FOLKI reste assez peu précis avec une erreur moyenne de 5px dans l'estimation du flot optique. Dans la sous-section suivante, nous explorons différentes pistes d'amélioration de la précision de FOLKI.

3.2.5 Description détaillée de FOLKI et améliorations testées

Description détaillée de FOLKI

Pyramide d'images et transformée en Rang. Comme la grande majorité des algorithmes, FOLKI utilise une approche pyramidale (cf. 2.2.1 et 3.2.1) pour estimer le flot optique entre deux images I_0 et I_1. Ces pyramides $\{pyrI_0(n)\}_{1\leq n\leq N}$ et $\{pyrI_1(n)\}_{1\leq n\leq N}$ sont construites en prenant $pyrI_i(N) = I_i$ et calculant récursivement $pyrI_i(n-1)$ en sous-échantillonnant $pyrI_i(n)$ d'un facteur 2, après avoir appliqué un filtre passe-bas pour éviter les effets d'*aliasing*. Dans notre cas, le filtre passe-bas est une convolution séparable de l'image par le noyau : $\left[\frac{1}{16}, \frac{1}{4}, \frac{3}{8}, \frac{1}{4}, \frac{1}{16}\right]$.

Comme mentionné précédemment, après avoir construit ces pyramides, on applique à chaque image $\{pyrI_i(n)\}_{i,n}$ la transformation de Rang [Zabih and Woodfill, 1994]. Celle-ci consiste à attribuer à chaque pixel $\mathbf{x}$ de l'image, une valeur égale au rang de son intensité $I(\mathbf{x}$ parmi les intensités $\{I(\mathbf{x}'), \mathbf{x}' \in \mathcal{N}(\mathbf{x})\}$ dans une fenêtre de taille $(2R+1) \times (2R+1)$ autour de $\mathbf{x}$. Cette transformation des images permet d'être particulièrement robuste aux changements d'illumination entre I_0 et I_1.

Pseudo-code de FOLKI. L'algorithme FOLKI appartient aux méthodes locales. A un étage de la pyramide et à un rayon de la fenêtre $\mathcal{N}$ donné, FOLKI résoud à chaque itération un problème des moindres carrés correspondant à l'équation (3.24). Le pseudo code de l'algorithme est donné ci-dessous. Les notations $\otimes$ et $*$ désignent les opérateurs : multiplication pixel par pixel et convolution dans l'image ; $\omega(r)$ désigne une fenêtre carrée de taille (2r+1)x(2r+1) et dont les coefficients sont égaux à 1.

Data: Images : I_0 et I_1
Result: Flot optique $\mathbf{u} = \mathbf{u}_1^K$
$pyrI_0 = \text{pyramideRang}(I_0)$;
$pyrI_1 = \text{pyramideRang}(I_1)$;
$\mathbf{u}_{N+1}^K = 0$;
for *n=N :-1 :1* **do**
  $\mathbf{u}_n^0 = 2 \times \mathbf{u}_{n+1}^K \uparrow_2$;
  $\nabla I = \text{gradient}\,(pyrI_0(n))$;
  for $r \in Radii$ **do**
    $H = \omega(r) * \left(\nabla I \otimes \nabla I^T\right)$;
    for *k=1 :K* **do**
      $\forall \mathbf{x}, I_1^w(\mathbf{x}) = I_1\left(\mathbf{x} + \mathbf{u}_n^{k-1}(\mathbf{x})\right)$;
      $I_t = I_1^w - \nabla I \otimes \mathbf{u}_n^{k-1} - I_0$;
      $b = \omega(r) * (\nabla I \otimes I_t)$;
      $\mathbf{u}_n^k(\mathbf{x}) = H^{-1} b$;
    end
  end
end

Algorithm 6: Algorithme FOLKI

Paramètres. Nous utilisons les paramètres suivants pour FOLKI :

- nombre d'étages des pyramides d'images : $N = 5$
- rayon de la transformée de Rang : 2
- ensemble des rayons considérés pour la fenêtre locale : $Radii = \{8; 4\}$
- nombre d'itérations : $K = 4$

Améliorations de FOLKI testées

Comme nous l'avons mentionné précédemment, FOLKI souffre des mêmes défauts que les autres méthodes locales : le flot optique estimé au niveau des bords des

objets est souvent erroné à cause de la violation de l'hypothèse implicite de flot optique localement constant sur un voisinage de $\mathbf{x}$. Pour ne pas ralentir excessivement FOLKI, les pistes d'améliorations que nous envisageons doivent rester simples et ne doivent pas ajouter trop de calculs et d'itérations supplémentaires. C'est pourquoi nous nous sommes restreint à l'étude de différents pondérations $\omega(\mathbf{x}')$ dans l'algorithme FOLKI, en considérant des itérations de la forme :

$$\mathbf{u}^{k+1} = -\left(\sum_{\mathbf{x}'\in\mathcal{N}(\mathbf{x})} \omega(\mathbf{x}')\nabla I \nabla I^T\right)^{-1} \sum_{\mathbf{x}'\in\mathcal{N}(\mathbf{x})} \omega(\mathbf{x}')\nabla I_0(\mathbf{x}') I_t\left(\mathbf{x}', \mathbf{u}^k(\mathbf{x}')\right) \quad (3.35)$$

où

$$I_t\left(\mathbf{x}', \mathbf{u}^k(\mathbf{x}')\right) = I_1(\mathbf{x}' + \mathbf{u}^k(\mathbf{x}')) - \nabla I_0(\mathbf{x}')^T \mathbf{u}^k(\mathbf{x}') - I_0(\mathbf{x}') \quad (3.36)$$

FOLKI-Celli. S'inspirant de Simoncelli [Simoncelli, 1999], notre première proposition d'amélioration consiste à prendre les pondérations suivantes :

$$\omega(\mathbf{x}') = \frac{1}{\nabla I_0(\mathbf{x}')^T (\lambda_1 I d_2) \nabla I_0(\mathbf{x}') + \lambda_2} = \frac{1}{\lambda_1 \|\nabla I_0(\mathbf{x}')\|^2 + \lambda_2} \quad (3.37)$$

Notons qu'en prenant $\lambda_1 = 0$ on retrouve la version usuelle de FOLKI car la pondération constante $\omega(\mathbf{x}') = 1/\lambda_2$ est équivalente à ne pas prendre de pondération comme on peut le voir en regardant l'équation (3.35). Du fait de cette invariance à un facteur multiplicatif près, on étudie donc la pondération suivante avec un seul paramètre λ :

$$\omega(\mathbf{x}') = \frac{1}{\lambda \|\nabla I_0(\mathbf{x}')\|^2 + 1} \quad (3.38)$$

Nous avons testé plusieurs valeurs pour le paramètre λ, et nous avons obtenu avec notre code Matlab, les performances suivantes indiquées dans le Tableau 3.3. Nous constatons qu'en fait une telle pondération diminue les performances et que les meilleurs résultats sont obtenus pour $\lambda = 0$ c'est-à-dire avec l'algorithme FOLKI classique. Nous reviendrons sur l'interprétation de ce résultat dans la conclusion.

TABLE 3.3 – Comparaison des performances de FOLKI-Celli avec différentes valeurs de λ, sur le jeu de données d'entraînement de KITTI 2012 pour le flot optique.

FOLKI-Celli	Out-Noc	Out-All	Avg-Noc	Avg-All
$\lambda = 0$	19.57%	28.77%	4.8px	10.2px
$\lambda = 0.01$	20.19%	29.45%	5.1px	10.8px
$\lambda = 0.25$	22.61%	31.72%	5.9px	12.0px
$\lambda = 1$	24.66%	33.56%	6.6px	12.9px

SAD-FOLKI. Dans la formulation de Simoncelli, le terme $\lambda_1 Id_2$ dans la pondération $\omega(\mathbf{x}')$ représente la covariance de l'erreur du flot optique actuellement estimé en $\mathbf{x}'$. Plutôt que de modéliser cette incertitude de manière constante et *a priori*, nous choisissons un critère empirique pour caractériser l'incertitude sur $\mathbf{u}(\mathbf{x}')$ à un instant donné. Ainsi nous considérons un critère de la forme : $\rho\left(I_1\left(\mathbf{x}'+\mathbf{u}(\mathbf{x}')\right)-I_0\left(\mathbf{x}'\right)\right)$, pour déterminer la précision du flot optique $\mathbf{u}(\mathbf{x}')$ courant. Ce qui nous amène à considérer la pondération suivante :

$$\omega(\mathbf{x}') = \frac{1}{\lambda C(\mathbf{u},\mathbf{x}')+1} \tag{3.39}$$

avec :

$$C(\mathbf{u},\mathbf{x}') = \rho\left(I_1\left(\mathbf{x}'+\mathbf{u}(\mathbf{x}')\right)-I_0\left(\mathbf{x}'\right)\right) \tag{3.40}$$

On remarque que (3.39) est une généralisation de l'équation (3.38) que l'on retrouve prenant : $C(\mathbf{u},\mathbf{x}') = \|\nabla I_0(\mathbf{x}')\|^2$.

Nous avons considéré plusieurs métriques pour $\rho(.)$: la valeur absolue de la différence, et le carré de la différence. Nous avons également comparé les résultats obtenus en considérant non seulement la différence $I_1\left(\mathbf{x}+\mathbf{u}(\mathbf{x})\right)-I_0\left(\mathbf{x}\right)$ en $\mathbf{x}=\mathbf{x}'$, mais aussi sur tout un voisinage $\mathcal{N}_C(\mathbf{x}')$:

$$C(\mathbf{u},\mathbf{x}') = \sum_{\mathbf{x}\in\mathcal{N}_C(\mathbf{x}')} \rho\left(I_1\left(\mathbf{x}+\mathbf{u}(\mathbf{x})\right)-I_0\left(\mathbf{x}\right)\right) \tag{3.41}$$

Nous avons également considéré différentes localisations dans l'algorithme pour placer la remise à jour des coefficients $C(\mathbf{u},\mathbf{x}')$. Soit on calcule $C(\mathbf{u}_{n,r},\mathbf{x}')$ pour chaque rayon r, après avoir fini les K itérations. Soit on calcule $C(\mathbf{u}_n,\mathbf{x}')$ après avoir effectué les K iterations pour tous les rayons r de fenêtre (cf. le pseudo-code 6). Nous n'avons pas considéré d'estimer $C(\mathbf{u}^k_{n,r},\mathbf{x}')$ après chaque itération car cela ralentirait trop l'algorithme FOLKI.

En pratique nous avons constaté que les meilleurs résultats sont obtenus en estimant $C(\mathbf{u}_{n,r},\mathbf{x}')$ à chaque rayon $r \in \mathit{Radii}$ après avoir effectué les K itérations. Le tableau 3.4 indique les performances de SAD-FOLKI avec différentes valeurs du paramètre λ, et montre que $\lambda = 0.1$ donne les meilleurs résultats.

Le tableau 3.5 justifie le choix de la taille de fenêtre 5x5 pour le calcul de la sommes des valeurs absolues. Plusieurs différentes tailles de fenêtres carrées $\mathcal{N}_C(\mathbf{x}')$ centrées en $\mathbf{x}'$ sont testées : 3x3, 5x5 et 7x7.

Comparaison de FOLKI et de SAD-FOLKI

Grâce aux pondérations $\omega(\mathbf{x}',\mathbf{u})$ dans SAD-FOLKI, on constate parfois une nette amélioration par rapport à FOLKI, dans la qualité du flot estimé. En particulier on note à la Figure 3.4, que ces pondérations ont permis de réduire très fortement l'effet d'adhérence au voisinage du bord des voitures à gauche et à droite de l'image. Cependant, en testant sur toute la base d'entraînement KITTI 2012, on constate que

TABLE 3.4 – Comparaison des performances de SAD-FOLKI avec différentes valeurs de λ, sur le jeu de données d'entraînement de KITTI 2012, pour une fenêtre $\mathcal{N}_C$ de taille 5x5.

SAD-FOLKI	Out-Noc	Out-All	Avg-Noc	Avg-All
$\lambda = 0$	19.57%	28.77%	4.8px	10.2px
$\lambda = 0.01$	17.46%	26.76%	4.5px	9.7px
$\lambda = 0.1$	16.98%	26.31%	4.5px	9.6px
$\lambda = 1$	17.04%	26.36%	4.5px	9.6px
$\lambda = 10$	17.06%	26.37%	4.5px	9.6px

TABLE 3.5 – Comparaison des performances de SAD-FOLKI avec différentes taille de fenêtre $\mathcal{N}_C(\mathbf{x}')$, sur les données d'entraînement de KITTI 2012, pour un paramètre $\lambda = 0.1$.

SAD-FOLKI	Out-Noc	Out-All	Avg-Noc	Avg-All
$\mathcal{N}_C(\mathbf{x}')$ de taille 3x3	17.32%	26.57%	4.6px	9.8px
$\mathcal{N}_C(\mathbf{x}')$ de taille 5x5	16.98%	26.31%	4.5px	9.6px
$\mathcal{N}_C(\mathbf{x}')$ de taille 7x7	17.61%	26.91%	4.7px	9.9px

SAD-FOLKI ne parvient pas toujours à corriger cet effet d'adhérence, et il arrive même parfois que SAD-FOLKI entraîne une légère dégradation du flot optique par rapport à FOLKI.

Toutefois, de manière générale, SAD-FOLKI donne de meilleurs résultats que FOLKI en termes de pourcentage d'outliers et d'erreur moyenne sur le flot estimé, comme le montre le Tableau 3.6 qui indique les performances sur les données d'entraînement KITTI 2012 avec les codes Matlab de FOLKI et SAD-FOLKI.

TABLE 3.6 – Comparaison des performances de FOLKI et SAD-FOLKI sur le jeu de données d'entraînement KITTI 2012.

Algorithme	Out-Noc	Out-All	Avg-Noc	Avg-All	Temps	Environnement
SAD-FOLKI	16.98%	26.31%	4.5px	9.6px	2.6s	CPU (Matlab)
FOLKI	19.57%	28.77%	4.8px	10.2px	2.4s	CPU (Matlab)

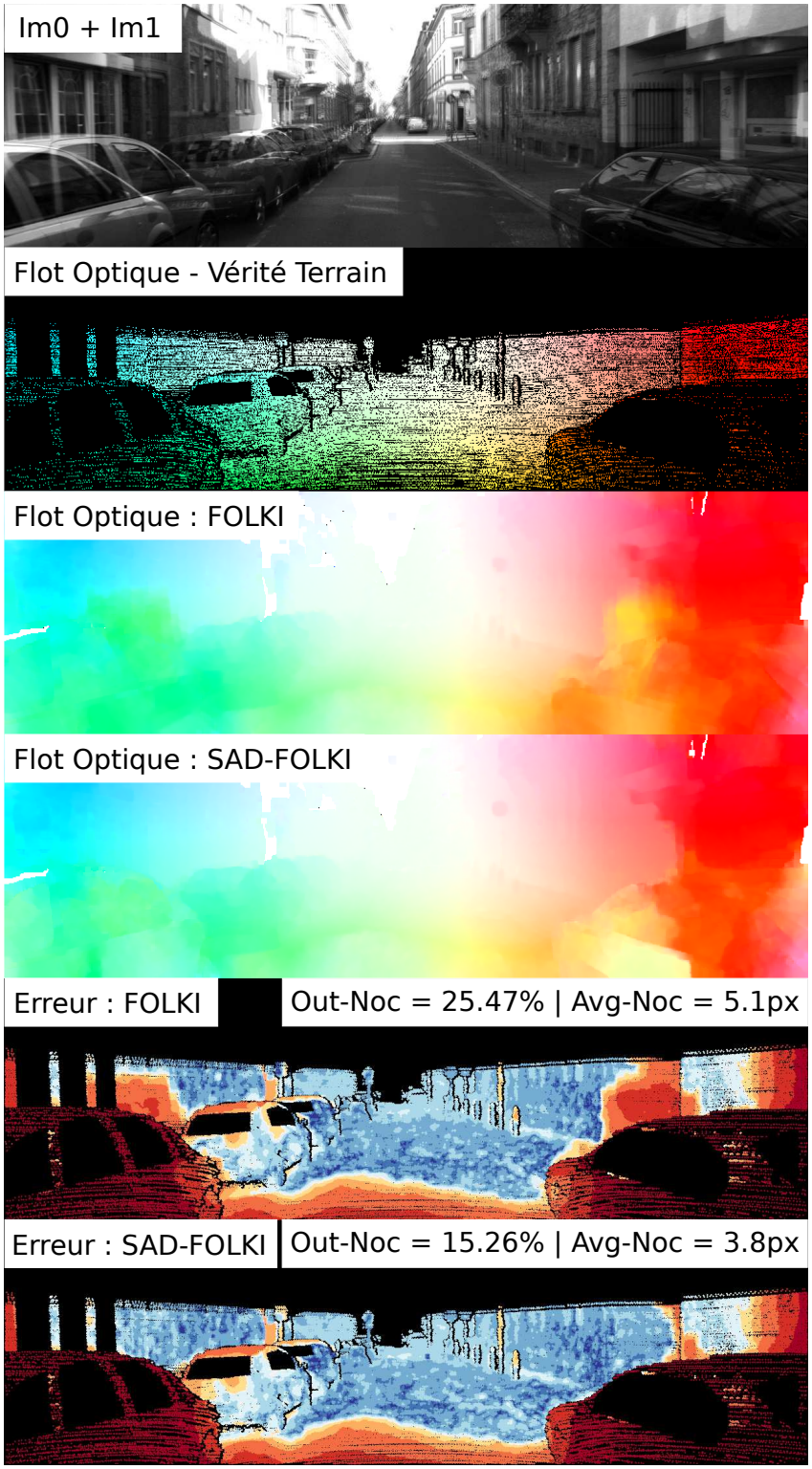

FIGURE 3.4 – De haut en bas : superposition des 2 images consécutives utilisées en entrée des algorithmes de flot optique, flot optique vérité terrain issu des scans laser, de FOLKI et de SAD-FOLKI, et erreur du flot estimé par FOLKI et SAD-FOLKI.

Conclusion

Dans cette sous-section nous avons étudié différentes manières de pondérer la contribution de chaque pixel voisin de $\mathbf{x}$ dans l'estimation du flot optique $\mathbf{u}(\mathbf{x})$ par approche locale. Deux variantes de l'algorithme FOLKI ont été proposées : FOLKI-Celli et SAD-FOLKI. Ces deux variantes correspondent à l'utilisation de pondérations ($\omega(\mathbf{x}')$) dont la forme est donnée à l'équation (3.39). La variante FOLKI-Celli inspirée de [Simoncelli, 1999], correspond à l'utilisation d'un critère $C(\mathbf{u}, \mathbf{x}')$ égal à $\|\nabla I(\mathbf{x}')\|^2$. Comme le montre le tableau 3.3, FOLKI-Celli donne de moins bons résultats que FOLKI, quelle que soit la valeur du paramètre λ choisie. Ceci n'est pas très étonnant, car le critère $C(\mathbf{u}, \mathbf{x}')$ utilisé par FOLKI-Celli réduit la contribution des pixels $\mathbf{x}'$ situés sur les régions de forts gradients alors que ce sont justement ces pixels qui fournissent le plus de renseignements sur le flot optique.

La variante SAD-FOLKI quant à elle, utilise un critère $C(\mathbf{u}, \mathbf{x}')$ correspondant à l'erreur de recalage autour du pixel $\mathbf{x}'$ (cf. Eq. (3.41)), et réduit donc la contribution des pixels $\mathbf{x}'$ pour lesquels l'estimation $\mathbf{u}(\mathbf{x}')$ s'avère mauvaise. L'utilisation de ce critère plus judicieux permet d'obtenir un léger gain en performance, comme on le voit dans le tableau 3.6, avec 2.6% de moins dans le taux d'outlier et une diminution de l'erreur moyenne de 0.3px (Avg-Noc) à 0.6px (Avg-All). Cependant, avec une erreur moyenne entre 4px et 5px, et un taux d'outliers d'environ 17%, les performances de SAD-FOLKI sont encore relativement insatisfaisantes. Dans la section suivante, nous considérons non pas deux images mais deux paires stéréo consécutives, et nous montrons que l'on peut construire, à partir de FOLKI, une méthode d'estimation de mouvement performante exploitant l'information supplémentaire apportée par la stéréo.

Dans la section suivante, nous considérons non pas deux images mais deux paires stéréo consécutives, et nous montrons que l'on peut construire, à partir de FOLKI, une méthode d'estimation de mouvement performante exploitant l'information supplémentaire apportée par la stéréo.

3.3 Estimation dense du mouvement des points 3D : le Flot de Scène

3.3.1 Contexte et notations

Un banc stéréo permet l'acquisition de deux images à chaque instant, ainsi après deux instants de mesure consécutifs notés t et $t+1$, on dispose de 4 images à partir desquelles on peut notamment estimer le flot optique $\mathbf{u}$ entre les images de la caméra gauche, et les cartes de disparité à chaque instant d_t et d_{t+1}. Ces données $\mathbf{u}, d_t, d_{t+1}$ permettent de calculer pour chaque pixel $\mathbf{x}$, le mouvement 3D apparent ΔX du point 3D auquel il correspond. En effet, en appliquant la formule de triangulation donnée à l'équation (2.12), on peut écrire :

$$\Delta X(\mathbf{x}) = X(\mathbf{x} + \mathbf{u}(\mathbf{x}), d_{t+1}) - X(\mathbf{x}, d_t) \tag{3.42}$$

La figure 3.5 récapitule à quoi correspond chacune de ces grandeurs, en prenant les notations :

- $(x, y) = \mathbf{x}$: un pixel de l'image gauche à l'instant t
- $(u, v) = \mathbf{u}(\mathbf{x})$: le flot optique estimé en $\mathbf{x}$
- $d = d_t(\mathbf{x})$: la disparité en $\mathbf{x}$ à t
- $d' = d_{t+1}(\mathbf{x} + \mathbf{u})$: la disparité de $(\mathbf{x} + \mathbf{u})$ à $t + 1$

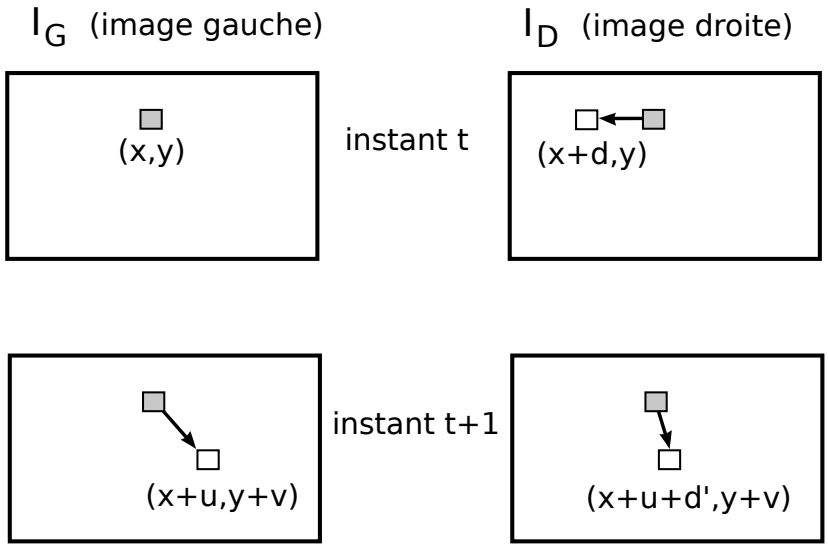

FIGURE 3.5 – Exemple de grandeurs (flot optique et disparités) mesurables à partir de 2 paires stéréo successives. Dans [Alcantarilla et al., 2012], les notations x_D, x', y' et x'_D sont employées pour décrire : $(x + d)$, $(x + u)$, $(y + v)$ et $(x + u + d')$.

Vedula et al [Vedula et al., 2005] ont introduit le terme *Flot de Scène* (*Scene Flow*) pour définir ce déplacement 3D ($\Delta X(\mathbf{x})$), qui est observé entre t et $t + 1$ et exprimé dans le repère de la caméra (gauche) à l'instant t. Les auteurs estiment le flot de scène à partir d'un jeu de 17 caméras fixées dans une salle de capture du mouvement (*motion capture*). De nombreux travaux ont été ensuite effectués sur l'estimation du flot de scène à partir de deux paires d'images stéréo successives. En pratique ces algorithmes estiment des grandeurs similaires à celles présentées à la figure 3.5, dont la forme diffère légèrement d'un algorithme à un autres, par exemple en estimant :

- $(\mathbf{u}, d_t, d_{t+1})$
- $(\mathbf{u}, d, d')$, où $d(\mathbf{x}) = d_t(\mathbf{x})$ et $d'(\mathbf{x}) = d_{t+1}(\mathbf{x} + \mathbf{u})$ (cf. Figure 3.5)
- $(\mathbf{u}, d, p)$, où p est le changement de disparité : $p(\mathbf{x}) = d'(\mathbf{x}) - d(\mathbf{x})$

Ces algorithmes permettent à partir de ces grandeurs, de calculer $X(\mathbf{x}, d_t)$ et $\Delta X(\mathbf{x})$. Si l'on dispose du déplacement (R, T) du banc stéréo entre t et $t + 1$, alors on peut calculer le mouvement propre à chaque point, en écrivant dans le repère de la caméra

gauche à t :

$$M(\mathbf{x}) = X(\mathbf{x}, d_t) + \Delta X(\mathbf{x}) - \Big(RX(\mathbf{x}, d_t) + T\Big) \tag{3.43}$$

Cette grandeur $M(\mathbf{x})$ constitue une information essentielle pour caractériser les éléments dynamique de l'environnement.

Une façon naïve (et très rapide) d'estimer le flot de scène, consiste à utiliser des algorithmes classiques de flot optique et de stéréo pour effectuer indépendamment l'estimation de $\mathbf{u}$ et des cartes de disparité d_t et d_{t+1}. Dans ce cas ces trois estimations ne bénéficient pas mutuellement les unes des autres. Au contraire, les méthodes dédiées à l'estimation du flot de scène sont basées sur l'optimisation d'un critère global qui inclue les interactions et les contraintes reliant ces différents termes $(\mathbf{u}, d_t, d_{t+1})$. Il n'est donc pas étonnant de constater que les quantités (flot optique et disparités) issues d'un algorithme d'estimation du flot de scène sont plus précises que celles issues d'algorithmes d'estimation du flot optique ou de stéréo. C'est ce que montre le tableau 3.7 qui présente le classement du benchmark KITTI 2012 pour le flot optique : les algorithmes qui retournent les estimations les plus précises du flot optique sont ceux qui considèrent non pas deux images, mais deux paires stéréo successives (notés "st" dans la 2ème colonne du tableau 3.7) et relèvent donc de méthodes d'estimation du flot de scène.

TABLE 3.7 – Extrait du classement KITTI 2012 pour le flot optique, avec le rang des algorithmes en septembre 2016. Le top du classement est constitué d'algorithmes utilisant des informations supplémentaires pour estimer le flot optique, tel que : la stéréo (st), plus de 2 images consécutives (mv), ou la géométrie épipolaire (ms).

Rang	Method	Setting	Out-Noc	Avg-Noc	Runtime	Environment
1	PRSM	st mv	2.46 %	0.7 px	300 s	1 core @ 2.5 Ghz
2	VC-SF	st mv	2.72 %	0.8 px	300 s	1 core @ 2.5 Ghz
3	SPS-StFl	st ms	2.82 %	0.8 px	35 s	1 core @ 3.5 Ghz
4	SPS-Fl	ms	3.38 %	0.9 px	11 s	1 core @ 3.5 Ghz
5	OSF	st	3.47 %	1.0 px	50 min	1 core @ 3.0 Ghz
6	PR-Sf+E	st	3.57 %	0.9 px	200 s	4 cores @ 3.0 Ghz
7	PCBP-Flow	ms	3.64 %	0.9 px	3 min	4 cores @ 2.5 Ghz
8	PR-Sceneflow	st	3.76 %	1.2 px	150 sec	4 core @ 3.0 Ghz
...						
24	NLTGV-SC		5.93 %	1.6 px	16 s	GPU @ 2.5 Ghz
31	DeepFlow		7.22 %	1.5 px	17 s	1 core @ 3.6Ghz
54	eFolki		19.31 %	5.2 px	0.026 s	GPU @ 700 Mhz
61	LDOF		21.93 %	5.6 px	1 min	1 core @ 2.5 Ghz
63	TV-L1		30.87 %	7.9 px	16 s	1 core @ 2.5 Ghz
68	HAOF		35.87 %	11.1 px	16.2 s	1 core @ 2.5 Ghz

Dans la sous-section 3.3.2 nous présentons l'état de l'art sur l'estimation du flot de scène. Cependant l'optimisation conjointe sur les différentes variables du

problème a un coût, comme on peut le remarquer sur le Tableau 3.7 (temps de calcul très élevé des solution "st"). Ces algorithmes étant trop coûteux en temps de calcul, nous proposons à la sous-section 3.3.3 une approche alternative en étudiant plusieurs stratégies pour l'estimation temps-réel du flot de scène. Il s'agit là, à notre connaissance, de travaux originaux dans la littérature qui se focalise essentiellement sur des méthodes d'estimation coûteuses pour le flot de scène.

3.3.2 Algorithmes d'estimation du flot de scène : état de l'art

Parmi les algorithmes d'estimation du flot de scène présents de la littérature, on distingue deux types de méthodes : celles qui modélisent le problème avec des termes de régularisation formulés dans l'espace image (en 2D), et celles qui utilisent des termes de régularisation 3D.

Formulation dans l'espace image

Les premiers algorithmes d'estimation du flot de scène comme ceux de Patras *et al.* [Patras et al., 1997], Huguet et Devernay [Huguet and Devernay, 2007], Min et Sohn [Min and Sohn, 2006], peuvent être vus comme des extensions des méthodes de calcul du flot optique. En effet les énergies minimisées dans ces méthodes comportent des termes correspondant à des contraintes de type conservation de l'intensité ainsi que des termes de régularisation spatiale dans l'image, que l'on trouve classiquement dans les algorithmes de flot optique et de stéréo.

Dans les travaux [Patras et al., 1997] et [Min and Sohn, 2006], l'estimation du flot de scène s'effectue en 3 étapes : la disparité d_t est d'abord calculée, puis les flots optiques $\mathbf{u}_G$ et $\mathbf{u}_D$ sont estimés dans les images gauches et dans les images droites, et enfin d_{t+1} est calculé. Dans [Patras et al., 1997], les flots $\mathbf{u}_G$ et $\mathbf{u}_D$ sont calculés sans tenir compte de d_t, et permettent d'obtenir ensuite d_{t+1} qui satisfait :

$$d_{t+1}(\mathbf{x}+\mathbf{u}_G) = v_D(x+d_t, y) - v_G(\mathbf{x}) \tag{3.44}$$

Au contraire dans [Min and Sohn, 2006], après avoir calculé la disparité d_t, les flots $\mathbf{u}_G$ et $\mathbf{u}_D$ sont estimés sachant d_t, en minimisant suivant $(\mathbf{u}_G\mathbf{u}_D)$ l'énergie :

$$\mathcal{E}(\mathbf{u}_G, \mathbf{u}_D|d_t) = \mathcal{E}_G(\mathbf{u}_G) + \mathcal{E}_D(\mathbf{u}_D) + \mathcal{E}_J(\mathbf{u}_G, \mathbf{u}_D|d_t), \tag{3.45}$$

qui contient des termes de contrainte de conservation de l'intensité et de régularisation spatiale :

$$\begin{aligned}
\mathcal{E}_G(\mathbf{u}_G) &= \int_\Omega \left(I_G^{t+1}(\mathbf{x}+\mathbf{u}_G) - I_G^t(\mathbf{x})\right)^2 + \lambda Tr\left(\frac{\nabla\mathbf{u}_G^T\left(\nabla I_G \nabla I_G^T + \sigma^2 I_d\right)\nabla\mathbf{u}_G}{\|\nabla I_G\|^2 + 2\sigma^2}\right)\mathrm{d}\mathbf{x} \\
\mathcal{E}_D(\mathbf{u}_D) &= \int_\Omega \left(I_D^{t+1}(\mathbf{x}+\mathbf{u}_D) - I_D^t(\mathbf{x})\right)^2 + \lambda Tr\left(\frac{\nabla\mathbf{u}_D^T\left(\nabla I_D \nabla I_D^T + \sigma^2 I_d\right)\nabla\mathbf{u}_D}{\|\nabla I_D\|^2 + 2\sigma^2}\right)\mathrm{d}\mathbf{x},
\end{aligned} \tag{3.46}$$

et un terme supplémentaire d'énergie *jointe* qui applique la contrainte liée à d_t :

$$\mathcal{E}_J(\mathbf{u}_G, \mathbf{u}_D | d_t) = \int_\Omega \Big(I_G^{t+1}(\mathbf{x} + \mathbf{u}_G) - I_D^{t+1}(\mathbf{x} + d_t + \mathbf{u}_D) \Big)^2 \mathrm{d}\mathbf{x} \tag{3.47}$$

Min et Sohn utilisent ensuite l'équation (3.44) pour obtenir d_{t+1}^0 qui sert alors d'initialisation pour calculer d_{t+1} en utilisant une méthode stéréo classique.

Huguet et Devernay [Huguet and Devernay, 2007] proposent une autre façon d'estimer le flot de scène, en calculant cette fois conjointement $\mathbf{u}$ (noté précédemment $\mathbf{u}_G$), d et d'. L'approche adoptée s'inspire de la méthode de Brox et al [Brox et al., 2004] (cf. équation (3.14)), et consiste à minimiser une énergie globale similaire :

$$\begin{aligned}
\mathcal{E}(\mathbf{u}, d, d') = \int_\Omega & \Phi\Big(\Delta_{Brox}\Big(I_G^{t+1}(\mathbf{x} + \mathbf{u}), I_G^t(\mathbf{x}) \Big) \Big) \\
& + \Phi\Big(\Delta_{Brox}\Big(I_D^{t+1}(\mathbf{x} + \mathbf{u} + d'), I_D^t(\mathbf{x} + d) \Big) \Big) \\
& + \Phi\Big(\Delta_{Brox}\Big(I_D^{t+1}(\mathbf{x} + \mathbf{u} + d'), I_G^{t+1}(\mathbf{x} + \mathbf{u}) \Big) \Big) \\
& + \Phi\Big(\Delta_{Brox}\Big(I_D^{t}(\mathbf{x} + d), I_G^{t}(\mathbf{x}) \Big) \Big) \\
& + \alpha \Phi\Big(\|\nabla u(\mathbf{x})\|^2 + \|\nabla v(\mathbf{x})\|^2 + \lambda \|\nabla \big(d'(\mathbf{x}) - d(\mathbf{x}) \big)\|^2 + \mu \|\nabla d(\mathbf{x})\|^2 \Big) \mathrm{d}\mathbf{x}
\end{aligned}$$

avec :

$$\Delta_{Brox}(I, I') = \|I - I'\|^2 + \gamma \|\nabla I - \nabla I'\|^2 \tag{3.48}$$

Afin de réduire la probabilité d'obtenir un minimum local de l'énergie $\mathcal{E}$ qui dépend maintenant de 4 champs scalaires différents (u, v, d, d'), l'optimisation par méthode variationnelle des énergies présentées dans [Huguet and Devernay, 2007] et [Min and Sohn, 2006], se fait en initialisant $\mathbf{u}$ et (d, d') avec des estimations indépendantes obtenues en utilisant des algorithmes de flot optique et de stéréo classiques.

Enfin, une autre approche intéressante est celle de [Wedel et al., 2011] qui s'inspire de [Patras et al., 1997] et [Min and Sohn, 2006] en découplant le problème stéréo et l'estimation du flot de scène : en estimant d'abord d, avant d'estimer les autres grandeurs du flot de scène (dans leur cas : $\mathbf{u} = \mathbf{u}_G$ et $p = d' - d$). L'idée derrière cette approche est de distinguer les problèmes d'appariement stéréo et d'appariement temporel : ces deux opérations qui sont souvent considérées comme assez proches, ont en fait des contraintes et des exigences très différentes. En effet, dans le cas des algorithmes stéréo, le déplacement estimé est potentiellement très grand (e.g. plus d'une centaine de pixels), et l'exigence principale n'est pas d'avoir une précision sous-pixellique mais de bien respecter les discontinuités dans la carte de disparité (i.e. bords des objets), discontinuités qui sont parfois d'amplitude très

importante (par exemple entre un objet très proche et le fond). Dans le cas de l'estimation du flot optique $\mathbf{u}$ et de la variation de disparité p, les mouvements mesurés sont beaucoup plus faibles et dans ce cas l'exigence principale est d'obtenir la meilleure précision (sous-pixellique) possible. Les estimations de d et de $(\mathbf{u}, p)$ correspondent donc à des problématiques très différentes pour lesquels il existe des algorithmes distincts. Pour cette raison, les auteurs calculent d'abord d avec l'algorithme stéréo *Semi-Global Matching* [Hirschmuller, 2005] pour ensuite estimer $(\mathbf{u}, p)$ en utilisant une méthode variationnelle inspirée des travaux de Huguet de Devernay [Huguet and Devernay, 2007] présentés précédemment. En utilisant un tel découplage, Wedel *et al.* parviennent à calculer le flot de scène en 50ms sur GPU, soit environ 500 fois plus rapidement que les techniques précédentes pour des images QVGA (320x240).

Formulation dans l'espace 3D

Au lieu de considérer des termes de régularisation spatiale définis dans l'espace image, certains algorithmes de flot de scène utilisent un modèle géométrique pour définir ces termes de régularisation dans l'espace 3D. C'est le cas des algorithmes de Vogel et al. [Vogel et al., 2013, Vogel et al., 2014, Vogel et al., 2015] qui modélisent la scène comme une collection de plans 3D sujets à des mouvements rigides. Le problème de flot de scène est exprimé sous la forme d'un problème d'inférence des plans 3D et de leur mouvement rigide, en utilisant une segmentation de l'image à base de superpixels inspirée de [Veksler et al., 2010]. Dans leur méthode, Vogel et al considèrent des termes de régularisation spatiale qui forcent la géométrie et le mouvement 3D à être lisses par morceaux.

Enfin, Menze et Geiger [Menze and Geiger, 2015] vont encore plus loin que Vogel et al. en modélisant la scène à partir de plans 3D estimés sur des superpixels, mais aussi à partir de modèles CAO 3D de voitures. Dans leur algorithme OSF (*Object Scene Flow*), un flot de scène épars est d'abord calculé sur ensemble de points caractéristiques extraits dans l'image et dont on estime les appariements temporels et stéréo. Ce flot de scène épars sert à segmenter la scène en une collection d'objets dont le mouvement supposé rigide est estimé avec un algorithme RANSAC à 3 points [Fischler and Bolles, 1981]. Cet algorithme OSF est très précis et a été utilisé pour construire, avec des scans laser 3D, la récente base de donnée KITTI 2015 pour l'évaluation des algorithmes de flot de scène.

Conclusion sur les méthodes existantes de flot de scène

Parmi les approches existantes pour l'estimation du flot de scène, les méthodes utilisant une régularisation en 3D et non pas dans l'espace image, font partie de l'état de l'art et figurent dans le top 5 du classement de flot optique KITTI 2012 et du classement de flot de scène KITTI 2015. Ces algorithmes qui utilisent une

représentation 3D pour modéliser les objets de la scène et leurs interactions, s'appuient sur des modèles plus proches de la réalité mais sont beaucoup plus chers en temps de calcul que leurs homologues qui utilisent une formulation dans l'espace image. Pour être clair, les méthodes 3D demandent un temps de calcul de plusieurs minutes, voire plusieurs dizaines de minutes et ne sont donc pas envisageables pour des applications embarquées temps réel, pour lesquels on considérera davantage une formulation dans l'espace image.

Cependant, même l'algorithme de Wedel *et al.* est encore trop lent pour l'application temps réel que nous envisageons. En effet, celui-ci s'exécute à 20Hz sur des images QVGA (320x240), or nous souhaitons travailler sur des images de résolution VGA (640x480) ou plus : comme les images du dataset KITTI (1230x370), pour lesquels la fréquence d'exécution se réduirait à 5Hz ou moins. Or il faut également ajouter à cela le temps requis pour le calcul de la disparité et du déplacement de la caméra par odométrie visuelle, qui sont nécessaires pour estimer le mouvement propre à chaque objet mobile, cf. Eq. (3.43). Un tel système de perception stéréo en environnement dynamique est donc trop lent pour notre application. Dans la sous-section 3.3.3, nous présentons une stratégie alternative pour estimer le flot de scène en temps réel (i.e. à cadence vidéo) sur les séquences KITTI acquises à 10Hz.

3.3.3 Estimation du flot de scène : découplage *structure-mouvement*

Nous avons vu à la sous-section 3.3.2, plusieurs méthodes d'estimation du flot de scène utilisant une régularisation soit dans l'espace image, soit dans l'espace 3D. Jusqu'à présent les algorithmes de flot de scène existants sont trop lents pour être utilisés à cadence vidéo sur les séquence KITTI (10Hz), et le seul algorithme capable de s'exécuter à plusieurs Hz est celui de Wedel et al [Wedel et al., 2011] qui utilise une régularisation dans l'espace image.

La stratégie adoptée par Wedel et al pour atteindre une telle vitesse d'exécution, est de découpler le calcul de la disparité d, du mouvement $(\mathbf{u}, p)$. Nous proposons d'aller encore plus loin dans ce découplage, en estimant d'abord la part du flot de scène induite par la structure de la scène et le déplacement du banc stéréo entre t et $t+1$, avant d'estimer la part restante correspondant au mouvement des objets. L'estimation de la partie *structure* du flot de scène repose sur le calcul de d_t et du changement de pose (R, T) pour pouvoir trianguler les points observés à t (en utilisant l'équation (2.12)) et prédire en supposant la scène statique, leurs coordonnées 3D dans le repère de la caméra à $t+1$:

$$X_{pred}(\mathbf{x}, d_t) = RX(\mathbf{x}, d_t) + T. \tag{3.49}$$

À partir de $X_{pred}(\mathbf{x}, d_t)$ on peut déduire le flot optique $\mathbf{u}_{pred}$ et le changement de disparité p_{pred} prédits en supposant la scène statique, à l'aide de l'opérateur de projection dans le plan image $\Pi(.)$ (cf. Eq. (2.13)) et de l'expression de la disparité donnée à l'équation (2.9) :

$$\mathbf{u}_{pred}(\mathbf{x}, d) = \Pi\Big(RX(\mathbf{x}, d_t) + T\Big) - \mathbf{x}, \tag{3.50}$$

et

$$p_{pred}(\mathbf{x}) = \frac{-bf_x}{\begin{pmatrix}0 & 0 & 1\end{pmatrix}^T \Big(RX(\mathbf{x}, d_t) + T\Big)} - d_t(\mathbf{x}). \tag{3.51}$$

On propose ensuite d'estimer la partie *mouvement* du flot de scène en partant de ces estimations $\mathbf{u}_{pred}$ et p_{pred}, pour estimer les grandeurs résiduelles $\delta\mathbf{u}$ et δp induites par les objets en mouvement. Une fois ces grandeurs résiduelles obtenues, on peut calculer le flot optique final $\mathbf{u}$ ainsi que p, en additionnant la partie *structure* avec la partie *mouvement* (ou en les composant, comme on le verra Eq. (3.62), si l'on utilise la stratégie de prédiciton d'image) :

$$\begin{aligned}\mathbf{u} &= \mathbf{u}_{pred} + \delta\mathbf{u} \\ p &= p_{pred} + \delta p\end{aligned} \tag{3.52}$$

Dans la lignée de Wedel *et al.*, mais avec l'objectif d'obtenir une méthode plus rapide, nous avons tenté de développer un algorithme rapide de type FOLKI pour estimer conjointement $\delta\mathbf{u}$ et δp. Cenpendant nos tentatives se sont révélées infructueuses. Au final, nous proposons d'utiliser FOLKI pour estimer la partie flot optique $\delta\mathbf{u}$ et de définir ensuite simplement p par :

$$p(\mathbf{x}) = d_{t+1}(\mathbf{x} + \mathbf{u}(\mathbf{x})) - d_t(\mathbf{x}) \tag{3.53}$$

Différentes manières peuvent être envisagées pour calculer le flot optique final $\mathbf{u}$ à partir de $\mathbf{u}_{pred}$ et de $\delta\mathbf{u}$. Dans la suite nous avons exploré les pistes suivantes :

- Le flot optique $\mathbf{u}_{pred}$ induit par la structure de la scène et le déplacement de la caméra (mouvement ego), peut servir d'initialisation ou d'a priori dans l'estimation de $\mathbf{u}$.
- Le flot optique prédit $\mathbf{u}_{pred}$ peut également être utilisé pour synthétiser I_t^{pred} : l'image que l'on devrait observer à t si la scène était statique. Notons que la prédiction se fait à rebours, en utilisant $\mathbf{u}_{pred}$ et I_{t+1} pour prédire I_t^{pred}. De cette manière en effet, le flot résiduel $\delta\mathbf{u}$ calculé entre I_t et I_t^{pred} se compose simplement pour former le flot total $\mathbf{u}(\mathbf{x}) = \delta\mathbf{u}(\mathbf{x}) + \mathbf{u}_{pred}(\mathbf{x} + \delta\mathbf{u})$ (cf. Eq. (3.62) et sa démonstration).

La figure 3.6 illustre ces différentes stratégies d'exploitation de d_t et (R, T) pour l'estimation de $\mathbf{u}$.

Dans un contexte de perception stéréo en environnement dynamique, un tel découplage est particulièrement sensé. En effet pour estimer le mouvement propre des objets mobiles, il faut compenser le mouvement induit par le déplacement de la caméra : il faut donc calculer tôt ou tard d_t et (R, T). Puisque ces informations de *structure* peuvent par ailleurs aider à l'estimation de $\mathbf{u}$, autant les estimer en amont pour pouvoir les utiliser, comme nous le proposons.

3.3.3.1 Stratégies basées sur le flot optique prédit

Stratégie d'initilisation du flot optique (PWOF). La manière la plus évidente d'exploiter $\mathbf{u}_{pred}$, est de s'en servir comme initialisation dans un algorithme

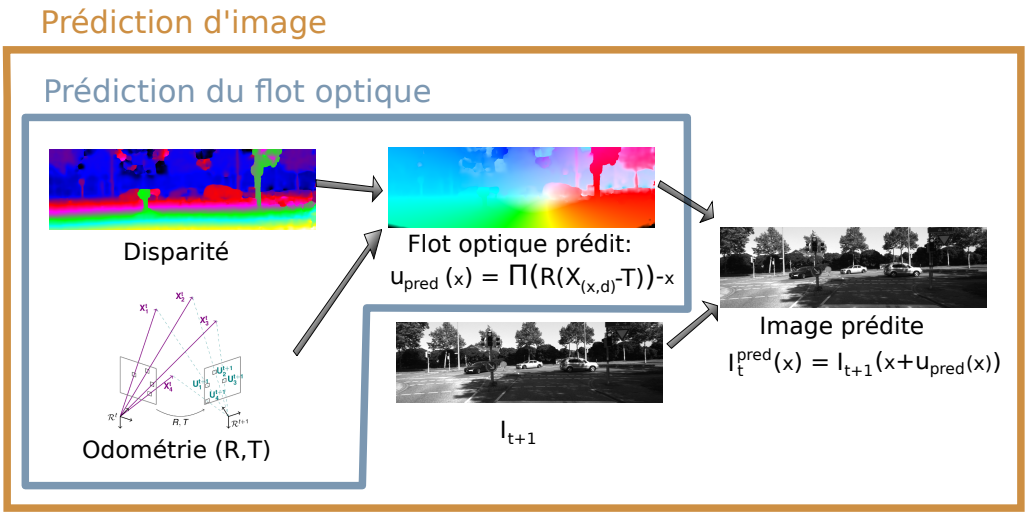

FIGURE 3.6 – Illustrations de différents moyens d'utiliser les informations de structure et de mouvement ego : en prédisant le flot optique $\mathbf{u}_{pred}$, ou en prédisant l'image I_t^{pred}.

d'estimation du flot optique. Cependant se pose la question suivante : comment s'interfacer aux algorithmes de flot optique usuels qui utilisent une approche pyramidale (cf. sous-section 3.2.1) pour estimer successivement le flot optique à différentes échelles ?

On peut construire une pyramide à partir de $\mathbf{u}_{pred}$ et initialiser le flot optique à un étage donné. Mais à un étage k trop élevé dans la pyramide, l'initialisation devient inutile car $\mathbf{u}_{pred}^k \approx 0$. De plus, avec un algorithme de stéréovision bien choisi, les discontinuités entre objets sont bien restituées dans $\mathbf{u}_{pred}$, comme illustré en Figure 3.6 : il est donc particulièrement inadapté de dégrader $\mathbf{u}_{pred}$ en montant dans la pyramide. On préfère donc utiliser $\mathbf{u}_{pred}$ à l'échelle la plus fine.

Nous nommons PWOF (*Prediction Warping Optical Flow*) la stratégie qui consiste à estimer le flot optique entre I_t et I_{t+1} sans utiliser d'approche pyramidale, en initialisant avec $\mathbf{u}_{pred}$.

Cette stratégie est proche de celle adoptée dans *Epic Flow* [Revaud et al., 2015], où un algorithme variationnel est appliqué sans utiliser de pyramides d'images, pour raffiner le flot optique initial obtenu par interpolation du flot estimé sur un ensemble de points caractéristiques. La différence étant qu'ici, l'initialisation par $\mathbf{u}_{pred}$ est incorrecte dans le sens où elle ne tient pas compte des objets mobiles. Ce qui peut engendrer des difficultés pour l'estimation de $\mathbf{u}$ lorsque les objets se déplacent rapidement, puisque l'on n'utilise plus de pyramide d'images.

Stratégie d'interactions d'échelle (PSIOF). Une autre façon de tenir compte de $\mathbf{u}_{pred}$ dans l'estimation du flot optique, est de considérer le modèle d'observation basé sur les interactions d'échelles présenté par Zille et al [Zille et al., 2014]. Dans

ces travaux, les auteurs étudient l'estimation du flot optique dans l'espace-échelle (*scale-space*) dont le principe est proche de l'estimation pyramidale (cf. Algorithme 4 dans la section 3.2) : ils considèrent non pas $2N$ images $\{I_0^n, I_1^n\}_{1\leq n\leq N}$ de tailles décroissantes avec n (pyramides), mais $2N$ images $\{I_0^{\sigma_n}, I_1^{\sigma_n}\}_{1\leq n\leq N}$ de même taille, construites récursivement par convolution avec un noyau gaussien g_σ :

$$I_i^{\sigma_n} = g_\sigma * I_i^{\sigma_{n-1}}, \text{ avec : } I_i^{\sigma_0} = I_i \tag{3.54}$$

D'une manière générale, que l'on décime ou pas, les approches multi-échelles rencontrent des difficultés pour estimer le flot optique au niveau des bords et des structures fines, car les détails des échelles les plus fines sont perdus aux échelles plus grossières. Les auteurs proposent de modéliser cette perte d'information d'une échelle $n-1$ à une échelle plus grossière n, en utilisant un flot optique a priori $\mathbf{u}_{prior}$ par exemple issu d'une première estimation de $\mathbf{u}$ —dans notre cas nous disposons de $\mathbf{u}_{pred}$ pouvant servir d'a priori. Ils représentent cette perte d'information liée à l'application du filtre passe-bas (noté $\overline{(.)} = g_\sigma * .$), en appliquant ce filtre à l'équation de contrainte du flot optique (OFC) :

$$\overline{\frac{\partial I(\mathbf{x})}{\partial t}} + \overline{\nabla I(\mathbf{x})^T \mathbf{u}(\mathbf{x})} = 0, \tag{3.55}$$

ce qui donne[1] :

$$\frac{\partial \overline{I(\mathbf{x})}}{\partial t} + \overline{\nabla I(\mathbf{x})}^T \overline{\mathbf{u}(\mathbf{x})} + \tau_{SI}(I, \mathbf{x}, \mathbf{u}) = 0, \tag{3.56}$$

avec :

$$\tau_{SI}(I, \mathbf{x}, \mathbf{u}) = \overline{\nabla I(\mathbf{x})^T \mathbf{u}(\mathbf{x})} - \overline{\nabla I(\mathbf{x})}^T \overline{\mathbf{u}(\mathbf{x})} \tag{3.57}$$

Etant donné qu'à chaque échelle n, on ne dispose que d'observations filtrées (passe-bas), les auteurs ne cherchent pas à résoudre l'équation standard (OFC), mais plutôt l'équation (3.56) comportant le terme de *Scale Interaction* (τ_{SI}) qui est calculé en utilisant $\mathbf{u}_{prior}$. En pratique l'estimation du flot optique à une échelle n donnée requiert plusieurs itérations, donc les auteurs considèrent pour l'estimation de $\mathbf{u}_n^{k+1}$ le terme $\tau_{SI}(I, \mathbf{x}, \mathbf{u}_{prior} - \mathbf{u}_n^k)$. On note que ce terme $\tau_{SI}(I, \mathbf{x}, \mathbf{u}_{prior} - \mathbf{u}_n^k)$ est élevé notamment lorsque ∇I est grand, alors que $\bar{\nabla} I$ l'est moins, ce qui correspond au cas où le changement d'échelle a effacé des détails de l'image, entraînant donc une perte d'informations qui doit alors être compensée à l'aide de $\mathbf{u}_{prior}$.

Dans notre cas avec FOLKI, l'opérateur de changement d'échelle $\overline{(.)}$ n'est plus seulement une convolution mais aussi un sous-échantillonnage d'un facteur 2 : $\overline{(.)} = (g_\sigma * .)\downarrow_2$, qui est également un opérateur linéraire[2]. Pour adapter à notre cas la méthode de Zilles et al., il faut donc construire la pyramide de notre a priori : $\{\mathbf{u}_n^{pred}\}_{1\leq n\leq N}$ dans notre cas. En appliquant ce modèle d'intéractions d'échelles à

1. Puisque le filtre passe-bas est un noyau gaussien, l'opérateur $\overline{(.)}$ est commutatif avec l'opérateur différentiel

2. Cette linéarité est nécessaire pour écrire l'équation (3.56).

l'algorithme FOLKI, on écrit donc à une échelle n et une itération $k+1$ données :

$$\mathbf{u}_n^{k+1}(\mathbf{x}) = \left(\sum_{\mathbf{x}' \in \mathcal{N}(\mathbf{x})} \nabla I_0^n(\mathbf{x}') \nabla I_0^n(\mathbf{x}')^T \right)^{-1} \left(- \sum_{\mathbf{x}' \in \mathcal{N}(\mathbf{x})} \nabla I_0^n(\mathbf{x}') \left(\frac{\partial I_0^n}{\partial t}(\mathbf{x}') + \tau_{SI} \left(I_0^n, \mathbf{x}', \mathbf{u}_{r,n}^k \right) \right) \right), \tag{3.58}$$

avec :

$$u_{r,n}^k(\mathbf{x}') = \mathbf{u}_{n-1}^{pred}(\mathbf{x}') - \left(\mathbf{u}_n^k \uparrow_2 \right)(\mathbf{x}'), \tag{3.59}$$

où $(.) \uparrow_2$ est l'opérateur de sur-échantillonnage d'un facteur deux, utilisé pour passer d'un étage n à $n-1$ de la pyramide (cf. sous-section 2.2.1). Notons que l'on retrouve l'estimateur usuel du flot optique en prenant $\tau_{SI} = 0$.

Pour ce qui nous concerne, ce modèle a été appliqué dans FOLKI, en prenant le flot optique prédit $\mathbf{u}_{pred}$ comme a priori. C'est donc cette configuration que nous retenons dans cette stratégie que nous appelons PSIOF (*Predicion Scale Interactions Optical Flow*). L'expérience montre que l'utilisation du terme τ_{SI} seulement à l'avant-dernière échelle $n = 2$, et non pas à chaque échelle $n \in \{2, 3, ..., N\}$ (il n'y a pas de terme d'interaction d'échelle à $n = 1$), donne de meilleurs résultats. Nous notons PSIOF-2 cette deuxième configuration de la stratégie PSIOF.

3.3.3.2 Stratégie basée sur l'image prédite

Les stratégies PWOF et PSIOF précédemment mentionnées, qui utilisent le flot optique prédit $\mathbf{u}_{pred}$ comme initialisation ou comme a priori, sont idéales si $\mathbf{u}_{pred}$ correspond approximativement au flot optique réel. Or comme nous l'avons déjà mentionné, ce n'est pas toujours le cas car $\mathbf{u}_{pred}$ n'inclut pas la part de flot optique (potentiellement grande) induite par les objets mobiles dans la scène.

Ceci est assez problématique car concrètement, avec ces stratégies, ce n'est qu'à l'échelle la plus fine que $\mathbf{u}$ peut vraiment différer de $\mathbf{u}_{pred}$. En effet PWOF n'utilise pas de schéma d'estimation pyramidale, et le terme d'interactions d'échelle $\tau_{IS}(I, \mathbf{u}_{prior})$ dans PSIOF influence l'estimation de $\mathbf{u}$ à toutes les échelles de la pyramide (sauf à l'échelle la plus fine) et biaise fortement $\mathbf{u}$ qui a tendance à rester très proche de $\mathbf{u}_{pred}$ alors que la scène n'est pas nécessaire statique.

Pour permettre d'une part d'estimer correctement les mouvements d'objets mobiles rapides, et d'autre part d'être moins dépendant de $\mathbf{u}_{pred}$ qui représente uniquement la partie statique du flot optique, nous proposons une autre stratégie basée sur l'utilisation d'images prédites.

Stratégie PCOF (*Prediction-Correction Optical Flow*). Puisque l'environnement observé peut contenir des objets mobiles dont le mouvement n'est pas nécessairement petit, il est convenable d'adopter un schéma d'estimation pyramidal. Nous avons vu précédemment avec PWOF et PSIOF que l'utilisation d'un a priori $\mathbf{u}_{prior}$ dans un schéma pyramidal, n'est pas trivial. C'est pourquoi nous proposons d'utiliser indirectement $\mathbf{u}_{pred}$ en synthétisant une image I_t^{pred} :

$$I_t^{pred}(\mathbf{x}) = I_{t+1}(\mathbf{x} + \mathbf{u}_{pred}(\mathbf{x})), \tag{3.60}$$

pour ensuite estimer de manière pyramidale, le flot optique résiduel $(\delta u, \delta v)$ entre I_t et I_t^{pred}, qui est induit par le mouvement des objets mobiles. L'équation de conservation de l'intensité nous permet d'écrire :

$$I_t(x, y) = I_t^{pred}(x + \delta u(x, y), y + \delta v(x, y)). \tag{3.61}$$

Ainsi, en combinant les équations (3.60) et (3.61), on estime le flot optique final $\mathbf{u}$, après correction :

$$\mathbf{u}(\mathbf{x}) = \delta\mathbf{u}(\mathbf{x}) + \mathbf{u}_{pred}(\mathbf{x} + \delta\mathbf{u}(\mathbf{x})) \tag{3.62}$$

Nous nommons par la suite PCOF (*Prediction-Correction Optical Flow*) cette stratégie.

3.3.4 Evaluation des différentes stratégies sur les jeux de données KITTI

Comme nous l'avons mentionné à la sous-section 3.3.3, nous proposons d'estimer le flot de scène $(\mathbf{u}, d, d')$ en calculant d'abord la disparité $d = d_t$, et le déplacement (R, T) de la caméra entre t et $t + 1$. Le flot optique $\mathbf{u}$ est alors estimé, sachant la structure de la scène et le mouvement propre, en adoptant une des trois stratégies PWOF, PSIOF et PCOF précédemment présentées.

Cette approche requiert donc l'estimation préalable de d_t, d_{t+1} et (R, T). Nous utilisons l'algorithme de stéréo ACTF [Sizintsev and Wildes, 2010] pour estimer les cartes de disparité, ainsi que le post-traitement CoR tous deux présentés à la section 2.2. Pour l'odométrie visuelle, nous utilisons l'algorithme de calcul de pose 2D-3D utilisé dans le système eVO [Sanfourche et al., 2013] que nous avons présenté à la section 3.1. Enfin pour le flot optique, c'est l'algorithme FOLKI décrit à la section 3.2 que nous choisissons. À titre de comparaison, nous testons également par la suite d'autres algorithmes de stéréo et de flot optique.

Nous évaluons ici les algorithmes sur les deux bases de données KITTI 2012 [Geiger et al., 2012] (pour le flot optique) et KITTI 2015 [Menze and Geiger, 2015] (pour le flot de scène), afin de comparer les différentes stratégies PWOF, PSIOF et PCOF. Nous analysons ensuite les variations de performances obtenues en considérant d'autres algorithmes de flot optique dans les stratégies PWOF et PCOF. Enfin nous discutons des performances générales en termes de robustesse, précision du flot optique et du flot de scène estimé, mais aussi en termes de temps de calcul.

3.3.4.1 Comparaison des stratégies PWOF, PSIOF et PCOF

Nous évaluons les performances des différents algorithmes de flot de scène sur les jeux de données KITTI 2012 (pour le flot optique) et KITTI 2015 (flot optique et flot de scène). Le premier jeu de données comporte des images stéréo acquises en environnement statique, et le critère d'évaluation est le taux d'*outliers* sur les zones non occultées (-Noc) de l'image (i.e. pour les pixels visibles à la fois dans I_0 et dans I_1), et sur toute l'image (-All). Pour les données KITTI 2012, un pixel est

considéré comme outlier si la norme de l'erreur du flot optique estimé est supérieure à 3 pixels. Pour les données KITTI 2015 qui comportent des objets mobiles avec des mouvements importants (e.g. voitures sur autoroute), la définition d'un *outlier* diffère : pour le flot optique $\mathbf{u}$ (respectivement pour les disparités d et d'), un pixel est considéré comme outlier si la norme de l'erreur est supérieure à 3 pixels et si celle-ci est supérieure à 5% du flot optique (respectivement : de d et d') de la vérité terrain (ceci pour être moins sévère quant aux erreurs sur les mouvements des objets très rapides). Dans KITTI 2015, un pixel est considéré comme outlier pour le flot de scène $(\mathbf{u}, d, d')$ s'il correspond à un outlier pour une des trois grandeurs : d, d' ou $\mathbf{u}$.

Les tableaux 3.8 et 3.9 indiquent les performances obtenues sur les données KITTI 2012 et KITTI 2015 pour le flot optique $\mathbf{u}$ obtenu en utilisant les différentes stratégies de prédiction utilisant la structure de la scène et le mouvement propre. Pour l'évaluation KITTI 2015, on distingue les erreurs faites sur l'ensemble de l'image (-all), et les erreurs faites sur les objets mobiles, notés *foreground* (-fg), et sur la partie statique, notée *background* (-bg).

TABLE 3.8 – Comparaison des performances obtenues avec les différentes stratégies de prédiction, avec FOLKI, et avec le flot optique prédit $\mathbf{u}_{pred}$, sur les données d'entraînement de KITTI 2012.

Flot Optique	Out-Noc	Out-All	Avg-Noc	Avg-All
FOLKI	19.57%	28.77%	4.8px	10.2px
Flot Optique Prédit	6.70%	9.85%	1.2px	1.7px
PWOF	8.03%	11.79%	1.6px	2.2px
PSIOF	8.02%	12.05%	1.6px	2.3px
PSIOF-2	7.98%	12.02%	1.6px	2.3px
PCOF	5.43%	8.77%	1.1px	1.6px

On constate sur ces tableaux que, quelle que soit la stratégie adoptée, l'utilisation de l'information supplémentaire apportée par d et (R, T) améliore nettement les performances par rapport à une utilisation classique de FOLKI. Sur KITTI 2012 on divise presque par 4 le pourcentage d'Out-Noc, et sur KITTI 2015 on réduit presque de moitié le pourcentage total d'outliers.

On remarque également que les stratégies PSIOF et PWOF sont moins bonnes que la stratégie PCOF qui est à chaque fois en première position, et qui est la seule dont les performances dépassent celles du flot optique prédit dans le cas statique de KITTI 2012. Cependant la prédiction $\mathbf{u}_{pred}$ ne peut pas capturer le mouvement des objets mobiles et, dans le tableau 3.9 (KITTI 2015), en présence d'objets mobiles PWOF et PSIOF sont évidemment plus adaptés.

Enfin, comme évoqué précédemment, PCOF est la stratégie qui peut le mieux s'affranchir des erreurs commises par $\mathbf{u}_{pred}$ en supposant la scène statique. D'ailleurs même dans le cas statique de KITTI 2012, PCOF donne de meilleures performances

TABLE 3.9 – Comparaison des performances obtenues avec les différentes stratégies de prédiction, avec FOLKI, et avec le flot optique prédit $\mathbf{u}_{pred}$, sur les données d'entraînement de KITTI 2015 : sur la partie statique (-bg), sur les objets mobiles (-fg), et sur toute l'image (-all).

Flot Optique	Out-bg	Out-fg	**Out-all**	Avg-bg	Avg-fg	Avg-all
FOLKI	37.40%	52.62%	**41.83%**	8.7px	21.9px	12.0px
Flot Optique Prédit	17.53%	86.69%	**29.46%**	2.1px	36.9px	10.9px
PWOF	17.90%	81.98%	**29.38%**	2.2px	36.6px	10.9px
PSIOF	17.78%	76.85%	**28.76%**	2.2px	36.2px	10.8px
PSIOF-2	17.24%	76.73%	**28.27%**	2.1px	36.2px	10.8px
PCOF	14.46%	55.55%	**22.62%**	2.0px	28.1px	9.1px

que $\mathbf{u}_{pred}$, car elle parvient à corriger les erreurs de stéréo et d'odométrie qui impactent $\mathbf{u}_{pred}$. Des résultats comparés sur une image particulière sont présentés en Figure 3.7 (en fin de chapitre). On constate d'abord que le gain principal par rapport à FOLKI est obtenu par la prédiction du flot statique. Dans les méthodes d'estimation du flot résiduel, seule PCOF apporte un gain significatif. La Figure 3.7 montre bien la capacité de PCOF à corriger notamment les erreurs d'odométrie visuelle qui ont une influence globale et se propagent sur l'ensemble de l'image.

3.3.4.2 Comparaison de différents algorithmes pour PCOF et PWOF

Nous étudions ici l'influence du choix des algorithmes de flot optique sur les performances de PWOF et de PCOF. Jusqu'à présent nous avons choisi ACTF+CoR pour la stéréo, et FOLKI pour l'estimation du flot optique, essentiellement car ce sont de loin les algorithmes les plus rapides dans leurs domaines respectifs. En quoi un tel choix impacte les performances du flot optique, et plus généralement du flot de scène estimé ? Pour répondre à cette question nous avons testé PWOF et PCOF avec les algorithmes de flot optiques : TV-L1 [Zach et al., 2007], HAOF [Brox et al., 2004] et LDOF [Brox et al., 2009], introduits dans la section 3.2.2. Nous avons aussi remplacé ACTF par SGBM (en appliquant également le filtrage CoR) dans PCOF. Les résultats en pourcentage d'*ouliers* et en erreur moyenne, sont présentés aux tableaux 3.10, 3.11 et 3.12.

Sur le jeu de données d'entraînement KITTI 2012 pour l'évaluation du flot optique, qui présente uniquement des scènes statiques sans objets en mobiles, l'utilisation de PCOF avec FOLKI donne les meilleurs résultats (cf. tableau 3.11). On en déduit que FOLKI permet de mieux raffiner ou corriger le flot optique prédit. Cet effet de correction est illustré Figure 3.8 (en fin de chapitre) où l'erreur d'odométrie entraîne une erreur globale importante sur $\mathbf{u}_{pred}$ (30% d'Out-Noc) que seuls FOLKI et LDOF parviennent à corriger significativement. Plusieurs raisons peuvent expliquer pourquoi l'utilisation d'un algorithme de flot optique local comme FOLKI

TABLE 3.10 – Comparaison des performances du flot optique estimé avec PCOF et PWOF en utilisant FOLKI (par défaut), HAOF, TV-L1, et LDOF, sur les données d'entraînement de KITTI 2012. PCOF avec SGBM (+CoR) à la place d'ACTF (+CoR) est également testé.

Flot Optique	Out-Noc	Out-All	Avg-Noc	Avg-All
PCOF_{SGBM}	5.27%	8.08%	1.1px	1.8px
PCOF	5.43%	8.77%	1.1px	1.6px
PCOF - LDOF	5.51%	8.84%	1.1px	1.6px
PCOF - HAOF	5.92%	9.18%	1.1px	1.6px
PWOF - HAOF	5.96%	9.18%	1.1px	1.6px
PWOF - TV-L1	6.66%	15.59%	1.2px	2.3px
Flot Optique prédit	6.70%	9.85%	1.2px	1.7px
PCOF - TV-L1	7.39%	10.42%	1.3px	1.8px
PWOF - FOLKI	8.03%	11.79%	1.6px	2.2px

corrige mieux les erreurs de $\mathbf{u}_{pred}$:

- l'erreur sur le flot optique prédit ne correspond pas nécessairement à un champ lisse par morceaux ce qui est implicitement supposé par le terme de régularisation spatiale dans les méthodes variationnelles (cf. 3.2.2)
- l'algorithme FOLKI converge très vite et permet un incrément important du flot estimé à chaque itération, contrairement aux méthodes variationnelles qui nécessitent plus d'itérations pour converger. En particulier, nous avons considéré les paramètres par défaut d'OpenCV pour HAOF, ce qui correspond à peu d'itérations et ne permet sans doute pas d'atteindre la convergence dans tous les cas.

L'utilisation de FOLKI améliore donc la robustesse en corrigeant les erreurs de prédiction, ce qui est crucial dans un système en cascade comme PCOF où l'estimation du flot optique repose directement sur $\mathbf{u}_{pred}$.

Sur le jeu de données d'entraînement KITTI 2015, où nous évaluons le flot optique (tableau 3.11) et le flot de scène (tableau 3.12), PCOF reste la meilleure stratégie et les performances obtenues avec FOLKI sont battues uniquement par LDOF. En effet dans les données KITTI 2015, les mouvements induits par les voitures sur la route peuvent parfois être très importants et difficiles à estimer pour les algorithmes de flot optique traditionnels. Cette problématique de mouvements de grandes amplitudes est directement gérée dans *Large Displacement Optical Flow* (LDOF), mais constitue encore un défi pour les algorithmes plus rapides comme FOLKI. Si l'on regarde le pourcentage d'outliers sur les objets mobiles (Out-fg), on s'aperçoit en effet que PCOF (avec FOLKI) donne près de 55% d'erreur sur le flot optique et 60% sur le flot de scène estimé.

TABLE 3.11 – Comparaison des performances des flots optiques estimés avec PCOF et PWOF en utilisant FOLKI (par défaut), HAOF, TV-L1, et LDOF, sur les données d'entraînement de KITTI 2015 : sur la partie statique (-bg), sur les objets mobiles (-fg), et sur toute l'image (-all). PCOF avec SGBM (+CoR) à la place d'ACTF (+CoR) est également testé.

Flot Optique	Out-bg	Out-fg	**Out-all**	Avg-bg	Avg-fg	Avg-all
PCOF - LDOF	13.97%	29.62%	**18.16%**	3.0px	16.9px	7.2p
PCOF_{SGBM}	14.28%	55.77%	**22.52%**	3.1px	28.8px	9.5px
PCOF	14.46%	55.55%	**22.62%**	3.0px	29.0px	9.6px
PCOF - HAOF	15.79%	52.34%	**23.12%**	3.1px	27.7px	9.3px
PCOF - TVL1	14.97%	55.68%	**23.14%**	3.1px	28.9px	9.6px
PWOF - HAOF	16.73%	85.88%	**28.68%**	2.9px	37.7px	11.0px
PWOF	17.90%	81.98%	**29.38%**	3.2px	37.5px	11.2px
Flot Optique Prédit	17.53%	86.69%	**29.46%**	2.1px	36.9px	10.9px
PWOF - TVL1	21.21%	80.15%	**32.09%**	3.7px	36.9px	11.6px

Concernant le choix de l'algorithme stéréo, nous voyons dans ces tableaux que l'utilisation de SGBM (+CoR) à la place de ACTF (+CoR), n'entraîne au mieux qu'une très légère diminution du taux d'outliers et de l'erreur moyenne des performances dans l'estimation du flot. Pour l'estimation du flot de scène, cette tendance est même inversée et PCOF donne de meilleures performances que PCOF_{SGBM} (cf. tableau 3.12). PCOF n'est donc pas particulièrement sensible au choix de la stéréo, ce qui conforte notre choix de prendre ACTF+CoR, en privilégiant la vitesse d'exécution.

Compromis précision/temps de calcul

Précédemment nous avons discuté du choix de stratégie à adopter et des algorithmes à utiliser. Maintenant nous faisons le point sur le temps de calcul de chaque algorithme de flot optique et de stéréo considérés.

Dans le tableau 3.13 nous indiquons les temps de calcul de chacune des étapes s'effectuant sur carte graphique (GPU) : l'appariement et le filtrage stéréo, le calcul de $\mathbf{u}_{pred}$ et I_t^{pred}, et l'estimation du flot optique. La seule étape qui ne bénéficie pas d'une accélération matérielle sur le GPU, est l'odométrie qui s'exécute en 50ms sur CPU (Intel Core i7).

Selon la stratégie adoptée, les algorithmes de flot optique doivent s'exécuter sur tous les étages n des pyramides d'images (e.g. PCOF), ou bien uniquement sur les images initiales, à l'étage $n = 1$ (e.g. PWOF). Ainsi pour les algorithmes dont on dispose des codes CUDA pour une exécution sur GPU, nous affichons dans le tableau 3.13 les temps de calcul avec et sans approche pyramidale. Ne disposant pas du code CUDA pour LDOF (mais seulement de son exécutable) son temps de calul n'est pas

TABLE 3.12 – Comparaison des performances des flots de scène $(\mathbf{u}, d, d')$ obtenus en prenant $\mathbf{u}$ estimé avec PCOF et PWOF en utilisant FOLKI (par défaut), HAOF, TV-L1, et LDOF, sur les données d'entraînement de KITTI 2015 : sur la partie statique (-bg), sur les objets mobiles (-fg), et sur toute l'image (-all). PCOF avec SGBM (+CoR) à la place d'ACTF (+CoR) est également testé.

Flot de Scène	Out-bg	Out-fg	**Out-all**
PCOF - LDOF	18.59%	37.22%	**23.16%**
PCOF	19.09%	60.31%	**27.03%**
PCOF_{SGBM}	19.98%	60.86%	**27.80%**
PCOF - HAOF	20.07%	57.61%	**27.36%**
PCOF - TVL1	19.61%	60.46%	**27.55%**
PWOF - HAOF	21.85%	87.86%	**33.00%**
PWOF	22.82%	84.29%	**33.56%**
PWOF - TVL1	26.07%	82.74%	**36.29%**

indiqué dans le tableau 3.13. LDOF s'exécute en 50s environ sur CPU (Intel Core i7).

TABLE 3.13 – Temps de calcul moyen (en ms) obtenu sur l'ensemble des données KITTI, sur deux GPUs différents, pour l'estimation de la stéréo, de la prédiction, et du flot optique avec et sans (entre parenthèses) approche pyramidale.

	ACTF + filtering	Prediction	FOLKI	HOAF	TV-L1
GTX TITAN (2688 cores)	11 + 7	1	27 (15)	127 (31)	128 (226)
Quadro 2000 (192 cores)	50 + 31	1	150 (113)	843 (283)	703 (1401)

On voit avec ces temps de calcul que la stratégie PWOF qui utilise un calcul de FOLKI à un seul niveau de pyramide, est effectivement moins coûteuse que PCOF (notamment avec HAOF dont le temps de calcul est divisé par 4 avec PWOF). D'après ce tableau, seule l'utilisation de FOLKI, ou bien de HOAF sans pyramides d'images (i.e. avec PWOF), est compatible avec une vitesse d'exécution à cadence vidéo (10Hz) sur KITTI. Etant données les performances nettement supérieures démontrées par PCOF-FOLKI par rapport à PWOF-HAOF et par rapport à l'utilisation de FOLKI avec n'importe quelle autre stratégie (PWOF, PSIOF, PSIOF-2), PCOF-FOLKI est le choix privilégié pour l'estimation du flot optique $\mathbf{u}$ et du flot de scène $(\mathbf{u}, d, d')$.

Notons que l'estimation des appariements stéréo par ACTF+CoR pour calculer d_t et d_{t+1}, peut s'effectuer en parallèle de l'odométrie calculant (R, T). L'estimation de (R, T), d_t et d_{t+1} nécessite donc seulement 50ms en utilisant une Ge-

Force GTX TITAN comme GPU. En ajoutant à cela le temps de calcul de FOLKI (27ms) et les étapes de prédiction et d'interpolation, le temps total nécessaire pour calculer $(\mathbf{u}, d, d') = (\mathbf{u}, d_t, d_{t+1}(\mathbf{x} + \mathbf{u}(\mathbf{x}))$ est inférieur à 80ms. Notre algorithme PCSF (*Prediction-Correction Scene Flow*) basé sur la stratégie PCOF faisant appel à FOLKI, est à notre connaissance le premier algorithme de flot de scène publié capable de tourner à cadence vidéo sur les séquences KITTI acquises à 10Hz.

3.3.5 Classements KITTI pour le flot optique et le flot de scène

Contrairement à la plupart des travaux effectués dans le domaine de l'estimation du flot optique et du flot de scène, nous n'avons pas cherché à obtenir la meilleure précision possible sans se préoccuper du temps de calcul, mais plutôt à obtenir un algorithme qui soit le plus rapide possible, tout en conservant une bonne précision. Nos algorithmes de flot optique PCOF_{SGBM} et PCOF sont actuellement classés 16e et 19e sur 78, sur le classement KITTI 2012.

A l'exception de PCOF_{SGBM} qui s'exécute en 0.8s, tous les algorithmes de flot optique du top-20 du classement KITTI 2012, sont 100 à 10000 fois plus lents que PCOF. PCOF est actuellement un des 6 algorithmes répertoriés sur KITTI capables d'estimer le flot optique à plus de 10Hz, et grâce à l'exploitation des informations stéréo, PCOF est en moyenne 5 à 7 fois plus précis que n'importe lequel de ces algorithmes. Le tableau 3.14 est extrait du classement KITTI 2012 [3]

Notons que la plupart des algorithmes du classement estiment le flot optique seulement à partir de deux images consécutives (et non pas en considérant comme nous, deux paires stéréo consécutives). Toutefois les algorithmes en tête du classement (le top-8) utilisent des informations additionnelles : soit en prenant en entrée deux paires stéréo (st), soit en utilisant la géométrie épipolaire en supposant la scène statique (ms), soit en utilisant plus de 3 images consécutives (mv) pour estimer le flot optique.

3. à la date du 4 octobre 2016

TABLE 3.14 – Extrait du classement KITTI 2012 pour l'estimation du Flot Optique (en octobre 2016).

	Method	Setting	Out-Noc	Out-All	Avg-Noc	Avg-All	Runtime	
1	PRSM	st mv	2.46 %	4.23 %	0.7 px	1.0 px	300 s	
2	VC-SF	st mv	2.72 %	4.84 %	0.8 px	1.3 px	300 s	
3	SPS-StFl	st ms	2.82 %	5.61 %	0.8 px	1.3 px	35 s	
4	SPS-Fl	ms	3.38 %	10.06 %	0.9 px	2.9 px	11 s	
5	OSF	st	3.47 %	6.34 %	1.0 px	1.5 px	50 min	
6	PR-Sf+E	st	3.57 %	7.07 %	0.9 px	1.6 px	200 s	
7	PCBP-Flow	ms	3.64 %	8.28 %	0.9 px	2.2 px	3 min	
8	PR-Sceneflow	st	3.76 %	7.39 %	1.2 px	2.8 px	150 sec	
9	SDF		3.80 %	7.69 %	1.0 px	2.3 px	TBA s	
10	MotionSLIC	ms	3.91 %	10.56 %	0.9 px	2.7 px	11 s	
11	CNNF+PMBP		4.70 %	14.87 %	1.1 px	3.3 px	30 min	
12	PatchBatch_s		4.81 %	13.64 %	1.1 px	3.0 px	60 s	**GPU**
13	CNN-HPM		4.89 %	13.01 %	1.2 px	3.0 px	23 s	**GPU**
14	Patchbatch_i		4.92 %	13.84 %	1.2 px	3.3 px	60 s	**GPU**
15	PatchBatch		5.29 %	14.17 %	1.3 px	3.3 px	50 s	**GPU**
16	PCOF-SGBM	st	5.40 %	8.73 %	1.2 px	2.1 px	0.8 s	**GPU**
17	PGM-G-noK		5.53 %	15.12 %	1.2 px	3.5 px	5.05 s	
18	PGM-G		5.59 %	13.87 %	1.2 px	3.3 px	5.05 s	
19	PCOF	st	5.59 %	9.69 %	1.2 px	1.9 px	0.08 s	**GPU**
20	DDF		5.73 %	14.18 %	1.4 px	3.4 px	1 min	**GPU**
...								
54	eFolki		19.31 %	28.79 %	5.2 px	10.9 px	0.026 s	**GPU**
...								
56	DSTF	st mv	19.96 %	30.58 %	4.0 px	12.4 px	0.07s	**GPU**
...								
67	UnsupFlownet		34.85 %	43.15 %	4.6 px	11.3 px	0.03 s	**GPU**
...								
69	FlowNetS+ft		37.05 %	44.49 %	5.0 px	9.1 px	0.08 s	**GPU**
...								
72	DIS-FAST		38.58 %	46.21 %	7.8 px	14.4 px	0.023	

Sur le classement KITTI 2015 évaluant les algorithmes d'estimation du flot de scène (affiché au tableau 3.15), notre algorithme PCSF qui calcule le flot de scène à partir de PCOF et d'ACTF (+CoR), atteint la 6e place (sur 12) avec LDOF et la 7e place avec FOLKI. Néanmoins si l'on considère le taux d'outlier uniquement sur les objets mobiles (SF-fg), PCSF est dernier avec FOLKI et 8e avec LDOF.

Notons que ce classement est très récent, par conséquent la plupart des algorithmes antérieurs à sa création n'y figurent pas (e.g. [Wedel et al., 2011]). Ce classement comporte à titre de référence, les évaluations du flot de scène estimé en calculant de manière totalement indépendante le flot optique et les cartes de dispa-

rités. *Semi-Global Matching* (SGM), l'algorithme de référence en stéréo est combiné avec d'autres algorithmes de référence en flot optique (e.g. LDOF, C+NL) afin d'obtenir le flot de scène : $(\mathbf{u}(\mathbf{x}), d_t(\mathbf{x}), d_{t+1}(\mathbf{x}+\mathbf{u}(\mathbf{x}))$. Il est intéressant de noter qu'avec notre approche PCOF, qui exploite notamment d_t lors de l'estimation du flot optique, nous obtenons de meilleurs résultats qu'en combinant naïvement des algorithmes plus performants (et plus coûteux) d'estimation du flot optique (LDOF, C+NL) et de stéréo (SGM).

On remarque également que PCSF (noté "PCOF +ACTF") est actuellement le seul algorithme du classement s'exécutant sur GPU, et le seul (à l'exception de GCSF) qui ne nécessite pas un temps de calcul de l'ordre de la minute voire de l'heure.

Enfin on constate que l'estimation du mouvement sur les objets mobiles est très mauvaise sur le jeu de données KITTI 2015 avec près de 70% d'*outliers* pour le flot de scène sur les objets mobiles. Ceci est dû à la spécificité de ce jeu de données qui est acquis à 10Hz en environnement urbain et autoroutier, et qui comporte donc de très grands déplacements dus aux vitesses élevées des objets mobiles (e.g. voitures). Un tel contexte impose une gestion particulière de ces grands déplacement par exemple en s'appuyant sur l'extraction et le suivi de descripteurs ou points caractéristiques dans l'image, comme le font des algorithme tels que LDOF.

TABLE 3.15 – Classement KITTI 2015 pour l'estimation du Flot de Scène ; critère principal : le taux d'outlier sur le flot de scène (SF-all).

	Method	**Fl-bg**	**Fl-fg**	**Fl-all**	**SF-bg**	**SF-fg**	**SF-all**	**Time**	
1	PRSM	5.33 %	17.02 %	7.28 %	6.61 %	23.60 %	**9.44 %**	300 s	
2	OSF	5.62 %	22.17 %	8.37 %	7.01 %	28.76 %	**10.63 %**	50 min	
3	CSF	10.40 %	30.33 %	13.71 %	12.21 %	36.97 %	**16.33 %**	80 s	
4	PR-Sceneflow	11.73 %	27.73 %	14.39 %	13.49 %	33.72 %	**16.85 %**	150 s	
5	SGM+SF	20.91 %	28.90 %	22.24 %	23.09 %	37.12 %	**25.43 %**	45 min	
6	PCOF-LDOF	14.34 %	41.30 %	18.83 %	25.26 %	51.55 %	**29.63 %**	50 s	
7	PCOF + ACTF	14.89 %	62.42 %	22.80 %	25.77 %	69.35 %	**33.02 %**	0.08 s	GPU
8	SGM+C+NL	34.24 %	45.40 %	36.10 %	38.21 %	53.04 %	**40.68 %**	4.5 min	
9	SGM+LDOF	40.81 %	35.42 %	39.91 %	43.99 %	44.79 %	**44.12 %**	86 s	
10	HWBSF	40.74 %	35.53 %	39.87 %	46.42 %	43.99 %	**46.02 %**	7 min	
11	GCSF	47.38 %	45.08 %	47.00 %	52.92 %	59.11 %	**53.95 %**	2.4 s	
12	VSF	50.06 %	47.57 %	49.64 %	67.69 %	64.03 %	**67.08 %**	125 min	

3.3.6 Conclusion

Dans cette section nous avons présenté les algorithmes d'estimation du flot de scène qui permettent de calculer à partir de deux paires stéréo à t et $t+1$, le flot optique $\mathbf{u}$, ainsi que les disparités successives d et d', pour chaque pixel $\mathbf{x}$ de l'image gauche à l'instant t. Aucun algorithme de l'état de l'art n'étant suffisamment rapide pour une exécution temps réel sur des images de tailles réalistes, nous avons proposé une nouvelle approche découplant l'estimation de la structure, du mouvement propre (R,T), et du mouvement des objets dynamiques dans la scène. Nous estimons d'abord d et (R,T) pour ensuite estimer le flot optique $\mathbf{u}$ en exploitant cette information géométrique addtitionnelle. Parmi les stratégies proposées pour exploiter la structure et le mouvement propre, nous avons montré que la méthode PCOF par prédiction d'image est particulièrement précise et robuste, et permet une diminution signification des *outliers* et de l'erreur moyenne.

Nous avons montré qu'une telle approche permet l'estimation à 10Hz du flot de scène sur les données KITTI. Toutefois, n'ayant pas de gestion particulière des grands déplacements , notre méthode est mise en défaut sur les objets rapides (e.g. voitures sur la route) présents dans la base KITTI 2015. Cependant, dans notre cas l'application ciblée est l'exploration autonome d'un robot terrestre ou aérien de taille réduite, et la vitesse des objets mobiles susceptibles d'être rencontrés est a priori beaucoup moins importante que celle d'une voiture sur une autoroute. Donc cette problématique de mouvement très rapide n'est pas un enjeu principal pour nous.

Maintenant que nous disposons d'un algorithme rapide d'estimation du flot de scène, nous pouvons modéliser à la fois la partie statique et la partie dynamique de l'environnement, à cadence vidéo. En effet, en soustrayant au flot de scène, la part induite par le déplacement de la caméra, nous pouvons isoler le mouvement 3D propre de chaque pixel de l'image. Cependant toutes les valeurs que nous estimons sont sujettes à des incertitudes liées aux erreurs de mesures (e.g. stéréo, flot optique, odométrie visuelle,...). Pour pouvoir exploiter et interpréter statistiquement les valeurs que nous calculons, il faut donc modéliser et propager ces erreurs. Dans le chapitre suivant, nous étudions des méthodes de détection dense du mouvement qui reposent sur différents critères géométriques —que nous appellons aussi *valeurs résiduelles*—, et nous modélisons leur incertitude par propagation d'erreur.

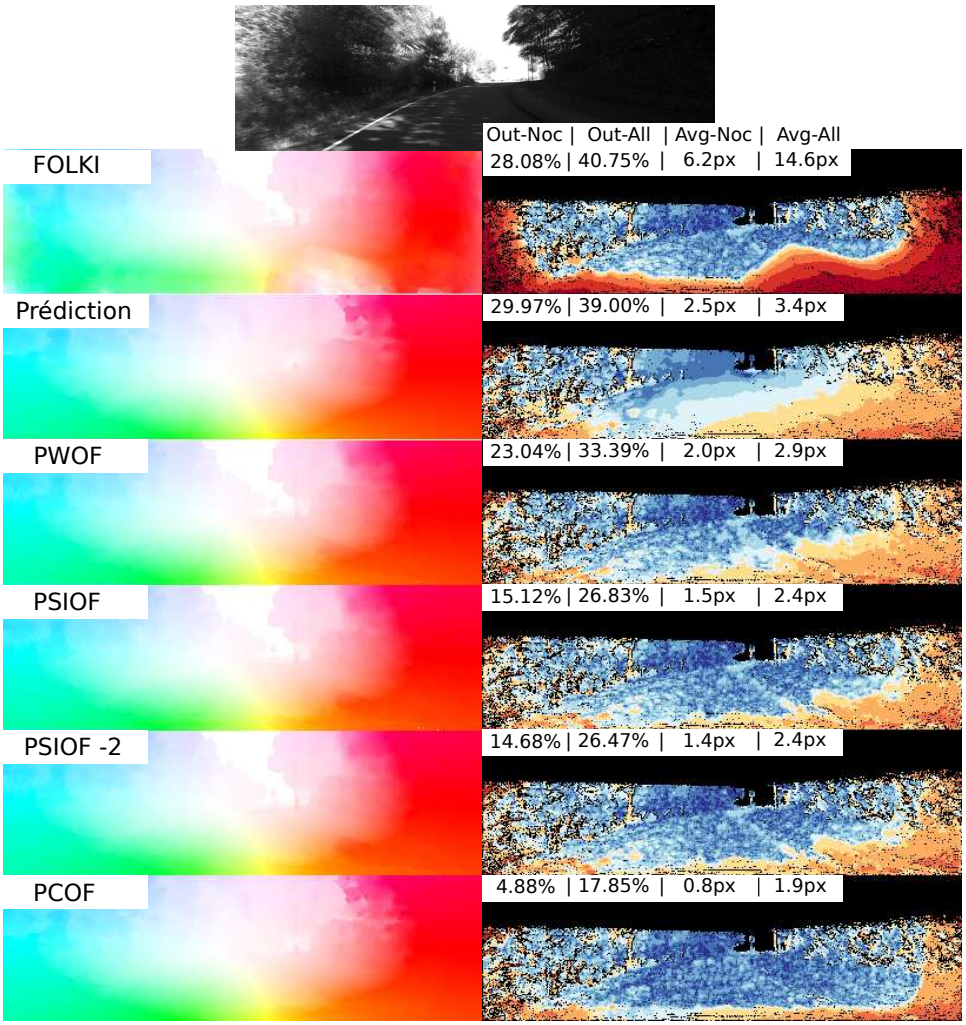

FIGURE 3.7 – Comparaison du flot optique estimé à $t = 10$ sur le jeu de données d'entraînement KITTI 2012, en utilisant FOLKI avec les stratégies PWOF, PSIOF et PCOF. A gauche : le flot optique affiché avec les conventions de couleurs KITTI (la teinte code l'orientation, et la saturation code l'intensité de $\mathbf{u}$) ; à droite : la carte d'erreurs avec le pourcentage d'outliers (Out) et l'erreur moyenne (Avg) sur les parties non-occultée (-Noc) et sur toute l'image (-All).

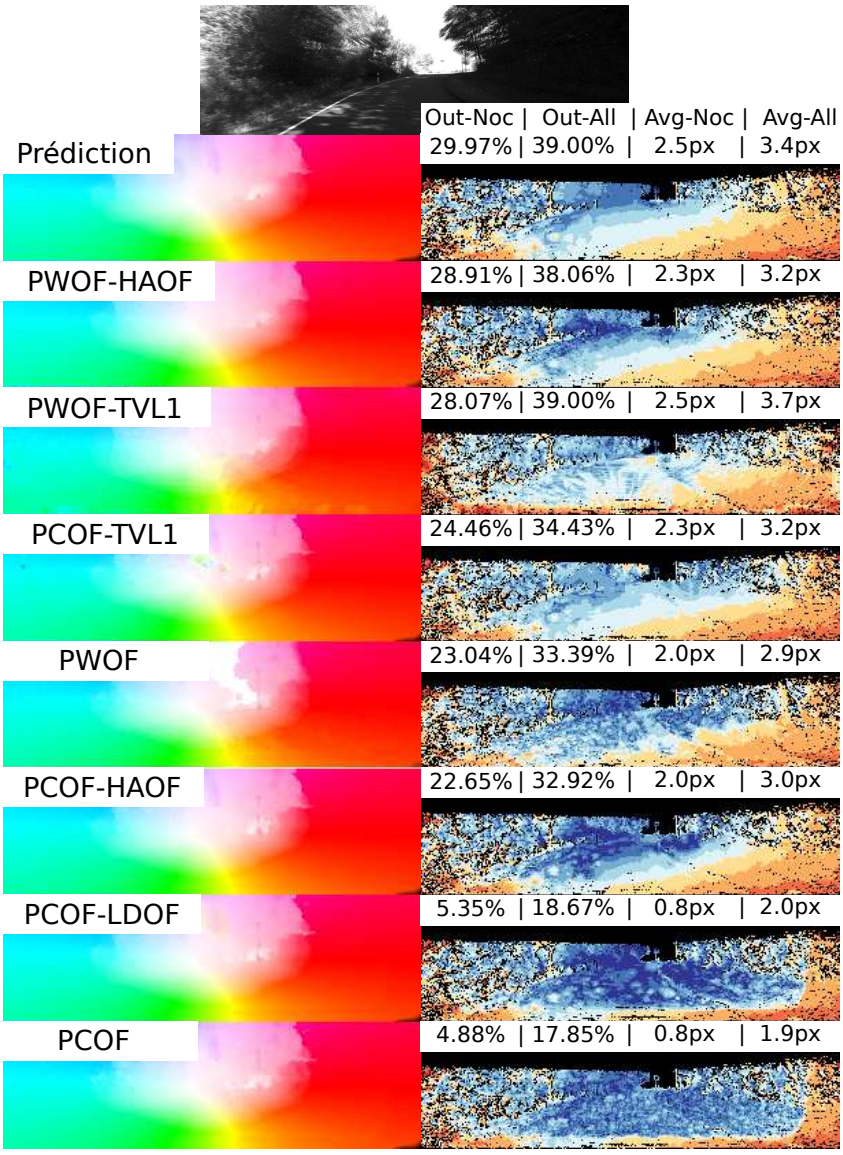

FIGURE 3.8 – Comparaison des flot optique estimés à $t = 10$ sur KITTI 2012, en utilisant HAOF, TVL1, FOLKI, LDOF avec les stratégies PWOF et PCOF. A gauche : flot estimé ; à droite : carte d'erreur.

Chapitre 4

Détection Dense d'Objets Mobiles à Partir d'un Banc Stéréo en Déplacement

Sommaire

Ce chapitre a donné lieu aux publications [1] et [3] de la liste fournie en section 6.3.

Introduction

Au Chapitre 2 nous avons présenté des algorithmes d'appariement stéréo, et au Chapitre 3 des algorithmes pour estimer le mouvement du banc stéréo (par odométrie visuelle) ainsi que des points observés (flot optique, flot de scène). Ces mesures fournissent pour chaque pixel de l'image : sa position 3D par rapport à la caméra (via la carte de profondeurs) et sa vitesse (déduite du flot de scène en compensant le mouvement de la caméra). Nous disposons donc de quoi modéliser à la fois les obstacles statiques et les obstacles mobiles. La perception d'obstacle est un élément indispensable dans un système de navigation robotique autonome car elle est nécessaire pour planifier une trajectoire sans collision. La détection et la caractérisation des objets mobiles en termes de forme et de dynamique, sont particulièrement cruciales puisqu'il faut anticiper le mouvement de ces derniers afin d'éviter leur collision.

Ce chapitre traite de la détection dense —i.e. en s'appuyant sur l'analyse de chaque pixel de l'image— d'objets mobiles, à partir d'un banc stéréo embarqué sur un robot. À partir des mesures mentionnées précédemment, on peut calculer un résidu M qui sert de base à la construction d'un critère géométrique de détection. Plusieurs résidus peuvent être considérés pour la détection d'objets mobiles ; nous comparons en Sec. 4.2 différents types de résidus proposés dans la littérature et différents modes opératoires possibles pour leur calcul. Pour effectuer des tests statistiques (e.g. test du χ^2) sur ces résidus, il convient d'abord de caractériser les erreurs de mesures et de les propager pour modéliser l'incertitude des résidus. C'est l'objet de la section 4.1. Ensuite, à la section 4.3 nous proposons une méthode de segmentation des objets mobiles dans l'image à partir de ces résidus. Enfin nous mettons en place dans la section 4.4 un protocole expérimental pour évaluer quantitativement les performances des algorithmes en termes de détection d'objets mobiles. Ceci nous permet de mettre en évidence et de sélectionner la meilleure approche et le meilleur résidu dans les situations qui nous intéressent pour la détection. Cette évaluation permet aussi d'analyser l'impact sur la performance de détection, des choix algorithmiques (stéréo et flot optique) et des choix de modèle d'incertitude.

4.1 Modélisation et propagation des erreurs de mesures

Dès lors que l'on effectue une mesure, on commet des erreurs : en raison de la résolution limitée de l'instrument de mesure, de sa calibration imparfaite, du bruit et des limites physiques du capteur, ou encore du traitement nécessaire à l'estimation d'une grandeur. La caractérisation des erreurs de mesure est un problème très général dont la portée dépasse largement le domaine de la vision par ordinateur.

Dans le cas de la vision stéréo, le capteur utilisé est un banc stéréoscopique que l'on a préalablement calibré pour fournir des paires d'images rectifiées (cf. sous-section. 2.1.3). Ces images potentiellement bruitées, sont les données d'entrée des algorithmes d'estimation de la carte de disparité, du flot optique et d'odométrie visuelle qui constituent les briques algorithmiques élémentaires du système de per-

ception présenté au Chapitre 3. Dans la suite, nous négligeons l'erreur de calibration et nous nous focalisons sur les erreurs d'estimation uniquement.

4.1.1 Modélisation des erreurs d'estimation : approches courantes

Une façon usuelle de modéliser l'incertitude d'une estimation X, est de la représenter comme la somme d'une vérité terrain $\overline{X}$ et d'un variable aléatoire ε représentant l'erreur d'estimation :

$$X = \overline{X} + \varepsilon \tag{4.1}$$

Si l'estimation est correctement effectuée, X est en moyenne égale à $\overline{X}$: on dit alors que l'estimation est *non-biaisée*. Dans ce cas, la variable aléatoire ε vérifie :

$$\mathbb{E}[\varepsilon] = 0, \tag{4.2}$$

et sa modélisation statistique est généralement définie à partir de son moment d'ordre 2 :

$$\mathbb{E}[\varepsilon\varepsilon^T] = \Sigma_\varepsilon. \tag{4.3}$$

Si la variable aléatoire ε suit une loi gaussienne, alors la connaissance de sa covariance Σ_ε suffit pour la caractériser entièrement. Dans le cas général ce n'est plus vrai : Σ_ε ne suffit pas pour représenter la probabilité de distribution de ε. Neanmoins, puisque nous propageons à l'ordre 1 les erreurs de mesure, en linéarisant le résidu M calculé à partir de celles-ci (voir sous-section 4.1.5), Σ_ε est suffisant pour estimer l'incertitude Σ_M.

Pour pouvoir calculer et analyser l'erreur ε définie à l'équation (4.1), il est nécessaire de disposer de la vérité terrain $\overline{X}$. Pour cela on peut soit utiliser des données de synthèse, soit faire une campagne d'acquisition de données réelles en effectuant des mesures à la fois avec le capteur que l'on évalue et avec un capteur très précis servant de référence pour établir une vérité terrain. Pour la stéréovision, on peut considérer les séquences de synthèse de Vaudrey et al [Vaudrey et al., 2008] qui décrivent des scènes de traffic routier vues depuis une voiture, ou bien les images de synthèse SINTEL [Butler et al., 2012] qui correspondent à un court film d'animation. Ces deux bases de données fournissent des vérités terrain pour évaluer les performances des algorithmes d'estimation de la disparité stéréo, du flot optique et du flot de scène. Neanmoins ces séquences ont une apparence type *cartoon*, et ne sont pas très représentatives du type d'images que l'on acquiert dans le monde réel. Les bases de données KITTI 2012 [Geiger et al., 2012] et KITTI 2015 [Menze and Geiger, 2015] quant à elles, fournissent une vérité terrain pour la stéréo, le flot optique, le flot de scène ainsi que l'odométrie visuelle, sur des données réelles acquises à partir d'un banc stéréo sur le toit d'une voiture circulant sur route et en milieu urbain. Dans ce cas la vérité terrain est établie à l'aide de capteurs très précis, en l'occurrence : un laser (Velodyne HDL-64E) et un système de localisation de pointe (OXTS RT 3003) combinant GPS, GNSS et centrale inertielle. En revanche cette vérité terrain est partielle, n'étant disponible que sur une partie du champ de vue (correspondant

au balayage du lidar Velodyne)

Grâce à la vérité terrain on peut calculer l'erreur ε faite sur chaque estimation, et analyser son comportement. Si l'on suppose que l'erreur ε suit la même densité de probabilité, quelles que soient les circonstances de mesure, alors on peut estimer la covariance Σ_ε de manière empirique à partir d'un ensemble de N données :

$$\Sigma_\varepsilon \approx \frac{1}{N} \sum_{i=1}^{N} \left(X_i - \overline{X}_i\right) \left(X_i - \overline{X}_i\right)^T \tag{4.4}$$

Au contraire si l'on suppose que les mesures ont une erreur dont la distribution de probabilité varie selon le contexte, on peut définir des indices contextuels $\alpha \in \mathcal{A}$ et exprimer Σ_ε en fonction de α : en calculant différentes covariances empiriques sur plusieurs partitions de $\mathcal{A}$, ou bien en cherchant une fonction de la forme :

$$\Sigma_\varepsilon(\alpha) = F(\alpha). \tag{4.5}$$

Par exemple pour l'estimation du flot optique et des appariement stéréo, on peut supposer que la précision de la mesure dépend de la texture et du contraste autour du point que l'on suit, ou bien encore de l'amplitude du mouvement que l'on mesure.

Enfin lorsqu'une estimation résulte d'une minimisation d'énergie, son incertitude provient non seulement du bruit des données utilisées mais aussi de la répartition des valeurs de ces données et du profil de la fonction optimisée. L'odométrie visuelle en particulier, est très sensible à la répartition spatiale des points utilisés pour le calcul de pose : si les points 3D considérés sont alignés ou coplanaires alors il est impossible d'estimer sans ambiguïté la rotation et la translation de la caméra, même si la position de ces points est parfaitement connue. De même pour l'estimation du flot optique par approche locale, si les gradients sont colinéaires (ou de norme trop faible) sur le voisinage d'un pixel $\mathbf{x}$ alors on ne peut résoudre le système (cf. Sec. 3.2.3) :

$$\left(\sum_{\mathbf{x}' \in \mathcal{N}(\mathbf{x})} \nabla I \nabla I^T \right) \mathbf{u} = - \sum_{\mathbf{x}' \in \mathcal{N}(\mathbf{x})} I_t \nabla I \tag{4.6}$$

En pratique puisque les données sont bruitées, ces situations dégénérées sont rarement rencontrées mais on peut être proche d'une situation de dégénérescence et avoir une très grande incertitude sur la mesure. Quoiqu'il en soit, il reste important de considérer la fonction minimisée et la répartition des données pour pouvoir propager leur erreur jusqu'à l'estimation. Dans le cas d'une minimisation des moindres carrés, pour estimer un vecteur X de dimension N à partir d'une matrice A connue et de données b affectées d'un bruit qui suit une loi normale $\mathcal{N}(0_N, \sigma_b^2 Id_N)$, on peut écrire :

$$\begin{aligned} X^* &= \left(A^T A\right)^{-1} A^T b = \arg\min_X \|AX - b\|^2 \\ \Sigma_{X^*} &= \sigma_b^2 \left(A^T A\right)^{-1}, \end{aligned} \tag{4.7}$$

dans le cas où $A^T A$ est régulière. Dans le cas plus général où une grandeur X est estimée à partir de données z, en optimisant le problème suivant :

$$X^* = \arg\min_X \mathcal{E}(X, z), \tag{4.8}$$

on peut appliquer le théorème des fonctions implicites, comme montré dans Faugeras [Faugeras, 1993], en considérant la fonction ϕ :

$$\phi : (X, z) \mapsto \frac{\partial \mathcal{E}}{\partial X}(X, z)^T, \tag{4.9}$$

qui vérifie : $\phi(X^*, z) = 0$. Alors, en notant $H = \frac{\partial^2 \mathcal{E}}{\partial X \partial X}$, le théorème des fonctions implicites donne l'expression de la covariance de X :

$$\Sigma_X = H^{-1} \left(\frac{\partial \phi}{\partial z}\right) \Sigma_z \left(\frac{\partial \phi}{\partial z}\right)^T H^{-T} \tag{4.10}$$

Dans la suite, nous décrivons comment sont modélisées dans la littérature les erreurs d'estimation des appariements stéréo, du flot optique et de l'odométrie visuelle.

4.1.2 Erreur de mesure des appariements stéréo

Les algorithmes de stéréo effectuent la mise en correspondance des pixels de l'image gauche du banc stéréo, avec les pixels de l'image droite. Dans le cas où la paire d'images stéréo est rectifiée, le correspondant d'un pixel $\mathbf{x} = (x, y)$ de l'image gauche se situe sur la même ligne dans l'image droite : en position $(x + d, y)$ (cf. 2.1.3). Cette mise en correspondance se réduit donc à l'estimation d'une quantité scalaire d, appelée disparité.

Dans les applications de la stéréovision à la mesure 3D, on utilise généralement des critères quadratiques, comme le critère des moindres carrés (ou SSD) sur un recalage de fenêtre (*Block-Matching*) présenté en (2.14). Le critère est souvent approximé par une fonction quadratique aux alentours du minimum pour permettre un affinage sous-pixellique de l'estimation. L'incertitude est alors souvent estimée à partir de la courbure du critère (ou de son approximation parabolique). Szeliski [Szeliski, 2010] propose ainsi une expression du type $Var(d) = \sigma^2/\rho$, où ρ est la courbure et σ le niveau de bruit sur les images. Dans le cas où l'on minimise une SSD, on peut montrer que la courbure est bien estimée par le carré du gradient spatial de l'image selon x et que la variance σ^2 peut être approximé par la valeur de la SSD au minimum.

Dans le domaine de la détection dense d'objets mobiles en stéréo, Wedel et al. [Wedel et al., 2011] sont les seuls, à notre connaissance, à avoir attaché un soin particulier à l'étude de l'incertitude de la disparité d. Les auteurs présentent une méthode pour caractériser σ_d en chaque pixel de la carte de disparité calculée avec

l'algorithme SGM [Hirschmuller, 2005], suivi d'un raffinement sous-pixellique dont le principe est illustré Figure 4.1. Ce raffinement consiste à évaluer pour chaque pixel $\mathbf{x}$, la fonction de coût d'appariement stéréo $E(.)$ en $\{d(\mathbf{x})-1, d(\mathbf{x}), d(\mathbf{x})+1\}$ pour déterminer :

$$d_E = \text{argmax}_{d' \in \{d(\mathbf{x})-1, d(\mathbf{x})+1\}} |E(d') - E(d(\mathbf{x}))|. \tag{4.11}$$

On note dans la suite $\Delta E(\mathbf{x}) = |E(d_E) - E(d(\mathbf{x}))|$. Ils considèrent la disparité sous-pixellique $d^*(\mathbf{x})$ comme le minimum de la fonction symétrique formée de deux parties affines de pentes $\pm\Delta E$, et passant par les trois points : $(d-1, E(d-1))$, $(d, E(d))$, et $(d+1, E(d+1))$ (cf. Fig. 4.1). L'inverse de $\Delta E(\mathbf{x})$ est utilisée pour calculer $\sigma_d^2(\mathbf{x})$ que les auteurs modélisent sous la forme d'une fonction affine :

$$\sigma_d^2(\mathbf{x}) = g_d + \frac{h_d}{\Delta E(\mathbf{x})}, \tag{4.12}$$

où g_d et h_d sont des coefficients estimés en étudiant les histogrammes d'erreur obtenus à partir de séquences de synthèse, selon différentes valeurs de disparité d. Dans ce modèle d'incertitude, σ_d dépend donc à la fois de la valeur de d et de la valeur absolue (ΔE) des pentes de la fonction symétrique d'interpolation utilisée lors du raffinement de la carte de disparité.

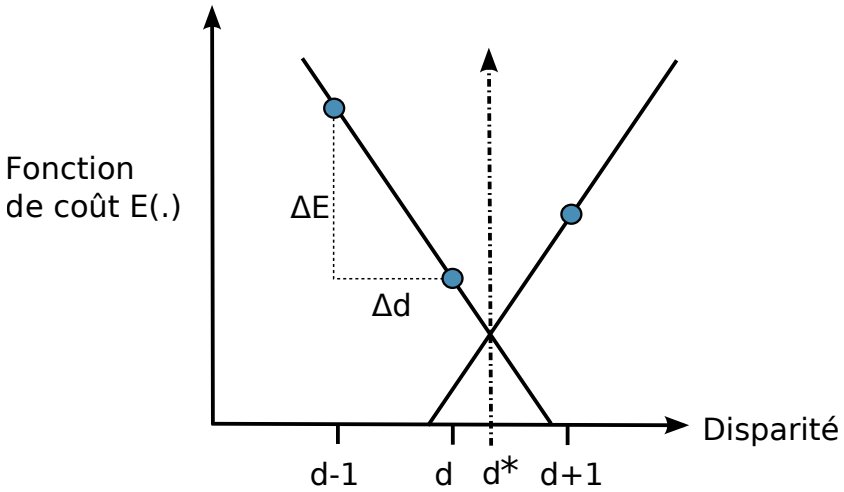

FIGURE 4.1 – Wedel *et al.* [Wedel et al., 2011] estiment le raffinement sous-pixellique d^* en évaluant la fonction de coût $E(.)$ utilisée pour l'estimation stéréo, en $(d-1)$ et $(d+1)$, et en considérant $\Delta E = max(|E(d-1) - E(d)|, |E(d+1) - E(d)|)$. Ils utilisent cette valeur $\Delta E(\mathbf{x})$ pour caractériser l'incertitude de l'appariement stéréo du pixel $\mathbf{x}$.

On le voit, l'estimation de l'incertitude du calcul de disparité dépend fortement de la méthode de raffinement sous-pixellique choisie, donc, pour les deux approches précédentes, de la modélisation du critère de mise en correspondance autour du minimum (parabole ou fonction affine symétrique). Dans la chaîne de stéréo que

nous avons détaillée au Chapitre 2, c'est le post-traitement *Consensus of Region* (CoR) tiré des travaux de Chakrabarti et al [Chakrabarti et al., 2015], qui est appliqué pour le raffinement sous pixellique de la carte de disparité. Les méthodes précédentes d'estimation d'incertitude ne sont donc pas directement transférables ici. Pour cette méthode par consensus de régions, Chakrabarti *et al.* ont suggéré d'évaluer l'erreur commise sur d par le cardinal $|J(\mathbf{x})|$: le cardinal de l'ensemble des régions incluant $\mathbf{x}$ et faisant partie du consensus (cf sous-section 2.2.3). Cependant ce cardinal représente le poids du consensus en $\mathbf{x}$, et est principalement révélateur de la géométrie locale de la surface de la scène en $\mathbf{x}$: en pratique on constate que $|J(\mathbf{x})|$ est grand notamment sur les régions planes ou légèrement courbées, alors que sur les zones de discontinuités (e.g. bords de voiture, barrières, objets fins,...) il est beaucoup plus petit, et ce, quelle que soit la précision réelle de $d(\mathbf{x})$.

Pour les raisons mentionnées précédemment, nous employons donc simplement un modèle d'écart-type σ_d constant pour caractériser l'erreur d'appariement stéréo. Notons que le choix d'une erreur constante est aussi adopté par d'autres auteurs dans le domaine de la détection d'objets mobiles. C'est le cas par exemple dans l'algorithme de détection de Romero-Cano et al. [Romero-Cano and Nieto, 2013]. De manière similaire, Alcantarilla et al. [Alcantarilla et al., 2012] attribuent une variance constante à chaque valeur pixellique $(x, x_D, y, x', x'_D, y')$ qui correspondent au pixel (x, y) dans l'image gauche à l'instant t, à sa position (x', y') à $t+1$, et aux appariements stéréo : (x_D, y) et (x'_D, y') (cf. Figure 3.5). Dans ce cas, ce n'est donc pas exactement l'incertitude de $d(x, y)$, mais celle de $x + d(x, y)$ qui est décrite par un écart-type σ.

4.1.3 Erreur de mesure du flot optique

Le flot optique $\mathbf{u}$ entre une image I_1 et I_2, est un champ vectoriel qui décrit le déplacement apparent des pixels $\mathbf{x}$ de I_1 vers I_2, et qui satisfait l'équation de conservation de l'intensité :

$$I_1(\mathbf{x}) = I_2(\mathbf{x} + \mathbf{u}(\mathbf{x})). \tag{4.13}$$

Le flot optique $\mathbf{u}(\mathbf{x})$ estimé en un pixel est un vecteur 2D, et la matrice de covariance de son erreur —notée $\Sigma_{\mathbf{u}(\mathbf{x})}$— appartient donc à $\mathbb{R}^{2\times 2}$.

Dans le contexte des approches locales d'estimation du flot optique avec un critère des moindres carrés, la matrice de covariance de l'erreur d'estimation peut être obtenue en appliquant la formule donnée à l'équation (4.7) :

$$\Sigma_{\mathbf{u}(\mathbf{x})} \propto (M(\mathbf{x}))^{-1} = \left(\sum_{\mathbf{x}' \sim \mathcal{N}(\mathbf{x})} \omega(\mathbf{x}' - \mathbf{x}) \nabla I(\mathbf{x}') \nabla I(\mathbf{x}')^T \right)^{-1}. \tag{4.14}$$

Une généralisation de cette formule au cas de l'estimation Bayésienne peut être dérivée du modèle de Simoncelli [Simoncelli, 1999] (utilisé par Romero-Cano et Nieto dans leur système de détection de mouvement [Romero-Cano and Nieto, 2013]) :

$$\Sigma_{\mathbf{u}(\mathbf{x})} = -\left(\sum_{\mathbf{x}' \in \mathcal{N}(\mathbf{x})} \frac{\nabla I \nabla I^T}{\lambda_1 \|\nabla I\|^2 + \lambda_2} + \Lambda_p^{-1} \right)^{-1} \tag{4.15}$$

L'intérêt de ces matrices de covariance est de tenir compte des incertitudes liées à la configuration locale des gradients image (notamment lorsque la fenêtre est centrée sur un contour rectiligne). Cependant il s'agit souvent d'une description trop riche de l'incertitude pour les applications de détection et on se contente alors d'une approximation isotrope :

$$\Sigma_{\mathbf{u}(\mathbf{x})} = \sigma^2_{\mathbf{u}}(c(\mathbf{x})) \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{4.16}$$

où l'écart-type $\sigma(c)$ est une fonction décroissante d'une mesure de confiance c dépendant du pixel.

Dans les approches locales, la mesure de confiance peut se déduire de la matrice qu'on inverse. Par exemple, certains considèrent la trace de la matrice $M(\mathbf{x})$ [Simoncelli et al., 1991] :

$$c(\mathbf{x}) = Tr\left(M(\mathbf{x})\right), \tag{4.17}$$

ou bien sa valeur propre minimale [Barron et al., 1994] :

$$c(\mathbf{x}) = \lambda_{\min}(M(\mathbf{x})). \tag{4.18}$$

Dans le même ordre d'idée, Uras et al [Uras et al., 1988] proposent de prendre la valeur du conditionnement $\kappa(H)$ de la Hessienne de $I(\mathbf{x})$:

$$\kappa(H) = \frac{\lambda_{\max}(H)}{\lambda_{\min}(H)} \quad , \text{ avec : } H = \begin{pmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{pmatrix}. \tag{4.19}$$

où la matrice peut être intégrée sur une fenêtre locale.

Pour les algorithmes variationnels présentés à la section 3.2.2, qui minimisent une énergie globale du type :

$$\mathcal{E}(\mathbf{u}) = \int_\Omega \mathcal{E}(\mathbf{u}, \mathbf{x}) \mathrm{d}\mathbf{x}, \tag{4.20}$$

Bruhn et Weickert [Bruhn and Weickert, 2006] proposent d'utiliser la contribution locale à l'énergie ($\mathcal{E}(\mathbf{u}, \mathbf{x})$) comme mesure de confiance, en posant :

$$c(\mathbf{x}) = \frac{1}{\mathcal{E}(\mathbf{u}, \mathbf{x}) + \varepsilon}. \tag{4.21}$$

Dans le même contexte d'estimation globale, Wedel et al. [Wedel et al., 2011], en s'inspirant de Bruhn et Weickert [Bruhn and Weickert, 2006], proposent la matrice diagonale :

$$\Sigma_{\mathbf{u}(\mathbf{x})} = \begin{pmatrix} g_u + h_u \mathcal{E}(\mathbf{u}, \mathbf{x}) & 0 \\ 0 & g_v + h_v \mathcal{E}(\mathbf{u}, \mathbf{x}) \end{pmatrix}, \tag{4.22}$$

où les coefficients h_u, g_u, h_v, g_v sont appris sur des séquences vidéos de synthèse.

Les expériences que nous avons menées sur les données réelles de la base de données KITTI nous ont montré qu'en ce qui concerne la performance de détection au final, les modèles d'erreur utilisant une matrice de covariance ne donnent pas de résultats significativement meilleurs que le modèle isotrope (4.16) tout simplement pris avec une variance $\sigma^2_{\mathbf{u}}$ constante. Nous choisissons donc ce modèle scalaire et constant pour la suite : cependant les dérivations que nous proposons peuvent être aisément généralisées à des matrices de covariance complètes et variables d'un pixel à l'autre.

4.1.4 Erreur de l'odométrie visuelle

Les algorithmes d'odométrie visuelle permettent d'estimer le déplacement (rotation et translation) de la caméra entre deux prises de vue. La translation T est paramétrée par ses 3 composantes (t_x, t_y, t_z) qui sont directement estimées par l'odométrie. Quant à la matrice de rotation R, diverses paramétrisations sont possibles, par exemple en utilisant :

- un quaternion $\mathbf{q} \in \mathcal{S}^3$
- des angles d'Eulers $(\theta_x, \theta_y, \theta_z) \in \mathbb{R}^3$
- un vecteur angulaire $\Omega = (\omega_x, \omega_y, \omega_z) \in \mathbb{R}^3$, pour une rotation infinitésimale :

$$R(\Omega) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix}$$

Il faut tenir compte du choix de cette paramétrisation Θ, car l'incertitude de l'odométrie visuelle est représentée par la matrice de covariance de ses paramètres : Σ_Θ.

Les systèmes stéréo de détection d'objets mobiles présents dans la littérature modélisent l'incertitude de l'odométrie avec une attention très inégale. Talukder et Matthies [Talukder and Matthies, 2004] ne représentent aucune erreur de mesure du déplacement de la caméra. Ils ne détaillent pas non plus explicitement de modèle d'erreur pour l'estimation de la disparité et du flot optique, mais ils prennent comme critère la norme $L2$ de la différence entre la mesure et la prédiction de (u, v, δ) : le flot optique et la variation de disparité. Ce qui revient à négliger la propagation d'erreur dans la prédiction et à ne considérer que l'erreur de mesure dont la covariance serait de la forme :

$$\Sigma_{(u,v,\delta} = \sigma^2 \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{4.23}$$

Dans le système présenté par Wedel et al [Wedel et al., 2011], les erreurs d'estimation stéréo et de flot sont modélisées de manière élaborée (cf. équations 4.12 et 4.22) mais la rotation est considérée nulle (matrice R égale à l'identité) et d'incertitude négligeable, et l'incertitude Σ_T de la translation est seulement mentionnée sans indication pour l'estimer. De même, l'algorithme de détection dense développé par Romero-Cano et Nieto [Romero-Cano and Nieto, 2013] modélise explicitement l'erreur du flot optique (cf. Eq. 4.15), mais tient uniquement compte de l'erreur d'odométrie avec des constantes scalaires σ_R^2 et σ_T^2 dont le calcul n'est pas précisé.

Une caractérisation plus fine de l'incertitude d'odométrie est faite dans le système de Bak et al. [Bak et al., 2014] qui supposent les rotations faibles en les paramétrisant avec le vecteur angulaire Ω. Les auteurs font l'hypothèse que l'erreur d'estimation des paramètres (Ω, T) peut être représentée par une matrice diagonale constante : $\Sigma_{(\Omega,T)} = diag(\omega_x, \omega_y, \omega_z, t_x, t_y, t_z)$ qu'ils estiment à partir d'une séquence vidéo de synthèse.

Lors de nos tests sur la base KITTI, avec les algorithmes de détection présentés dans les sections suivantes, nous avons notamment constaté que l'erreur d'odométrie pouvait devenir élevée dans le cas de fortes rotations de la caméra. Une modélisation de l'incertitude d'odométrie visuelle par une matrice de covariance constante est donc discutable. À notre connaissance, Alcantarilla et al [Alcantarilla et al., 2012] sont les seuls à avoir modélisé cette erreur par une matrice de covariance non constante. Ils paramétrisent leur rotation avec un quaternion $\mathbf{q}$, et estiment le déplacement de la caméra en résolvant un problème des moindres carrés non-linéaire de la forme :

$$\text{minimiser} : \mathcal{E}(q,T) = \sum_i \|F_i(q,T)\|^2, \tag{4.24}$$

et utilisent le modèle classique semblable à celui présenté Eq. 4.7 (en linéarisant la fonction $F(.)$) :

$$\Sigma_{\widehat{q},\widehat{T}} \propto H^{-1} \approx \sum_i J^T_{F_i(\widehat{q},\widehat{T})} J_{F_i(\widehat{q},\widehat{T})}, \tag{4.25}$$

où $J_{F_i(\widehat{q},\widehat{T})}$ est la matrice jacobienne de $F_i(.)$ évaluée en $(\widehat{q},\widehat{T})$: l'optimum estimé pour 4.24. Ce modèle tient compte de l'énergie minimisée en considérant sa matrice Hessienne en l'optimum $(\widehat{q},\widehat{T})$ estimé. Néanmois cette énergie dépend aussi de données d'entrée, en l'occurrence des points remarquables extraits dans l'image, et dont on calcule les appariements temporels et stéréoscopiques. Le modèle (4.25) ne prend pas en compte les erreurs affectant ces données d'entrée. À la sous-section suivante nous présentons un modèle plus fin qui permet d'exprimer l'incertitude d'odométrie visuelle en propageant plus rigoureusement les erreurs de mesures sur lesquelles elle s'appuie.

4.1.5 Propagation d'erreurs dans la chaîne de détection

Comme expliqué en Sec. 4.2, diverses grandeurs résiduelles peuvent être considérées pour la détection de mouvement. Cependant, quel que soit le résidu choisi, on doit calculer le déplacement (R,T) de la caméra entre deux instants et trianguler

les points $X(\mathbf{x}, d)$ pour pouvoir prédire —en faisant l'hypothèse qu'ils sont fixes— leur coordonnées 3D à l'instant suivant :

$$X_{\text{pred}} = RX + T \tag{4.26}$$

Cette prédiction "statique" (i.e. effectuée pour une hypothèse de scène rigide et statique) est utilisée pour le calcul de toute grandeur résiduelle. Pour déterminer l'incertitude des résidus, on doit donc estimer l'incertitude de cette prédiction, en propageant les erreurs de mesures (appariements temporels et stéréo) à travers (R, T) et X.

Incertitude de triangulation stéréo

La triangulation stéréo dont on rappelle ici la formule déjà donnée en sous-section 2.1.3, est à la base de la perception 3D stéréoscopique :

$$\mathbf{X}(x, y, d) = -\frac{b}{d}\begin{pmatrix} x - x_0 \\ \frac{f_x}{f_y}(y - y_0) \\ f_x \end{pmatrix} \tag{4.27}$$

On prendra temporairement la notation $\mathbf{X} = (X, Y, Z)$ pour désigner un point 3D et chacune de ses coordonnées.

De manière classique, on modélise l'incertitude sur $\mathbf{X}(x, y, d)$ en propageant l'erreur sur d (notée δd) par linéarisation de l'expression 4.27 :

$$\begin{aligned} \delta\mathbf{X} &= \frac{\partial \mathbf{X}}{\partial d}\delta d \\ &= \frac{b}{d^2}\begin{pmatrix} x - x_0 \\ (y - y_0)\frac{f_x}{f_y} \\ f_x \end{pmatrix}\delta d. \end{aligned} \tag{4.28}$$

En allant plus loin on peut modéliser l'incertitude liée à la résolution spatiale limitée de l'image, avec $\delta x \sim \delta y \sim \mathcal{N}(0, \sigma_x^2)$, et en utilisant la matrice jacobienne $J_{\mathbf{X}(x,y,d)}$:

$$\begin{aligned} \delta\mathbf{X} &= J_{\mathbf{X}(x,y,d)}\begin{pmatrix}\delta x \\ \delta y \\ \delta d\end{pmatrix} \\ &= \left(\begin{array}{cc|c} -\frac{b}{d} & 0 & \\ 0 & -\frac{b}{d} & \frac{b}{d^2}\begin{pmatrix} x - x_0 \\ (y - y_0)\frac{f_x}{f_y} \\ f_x \end{pmatrix} \\ 0 & 0 & \end{array}\right)\begin{pmatrix}\delta x \\ \delta y \\ \delta d\end{pmatrix}, \end{aligned} \tag{4.29}$$

ce qui permet d'écrire Σ_X sous la forme :

$$\Sigma_{\mathbf{X}} = J_{\mathbf{X}(x,y,d)}\begin{pmatrix}\sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_d^2\end{pmatrix} J_{\mathbf{X}(x,y,d)}^T. \tag{4.30}$$

On constate avec les équations 4.28 et 4.29 que l'erreur de triangulation stéréo est très anisotrope. Au premier ordre, il est facile de montrer l'équations suivante :

$$\delta Z = \frac{Z^2}{b f_x} \delta d \tag{4.31}$$

Cette équation indique que l'erreur de triangulation croît en Z^2 et qu'elle est fortement anisotrope, en proportion du rapport de la distance à la baseline. Il est donc essentiel de tenir compte de cette incertitude de triangulation stéréo partout où elle intervient, notamment dans l'odométrie visuelle comme nous allons le voir.

Incertitude de l'odométrie visuelle

Les modèles d'incertitude d'odométrie présentés en sous-section 4.1.4, qui sont utilisés dans les systèmes de détection présents dans la littérature, ne tiennent pas comptent des erreurs affectant les appariements stéréo ou temporel. En particulier, ils ne modélisent pas l'influence de l'erreur de triangulation des amers sur lesquels s'appuie l'algorithme d'odométrie visuelle.

Dans le cas des algorithmes d'odométrie visuelle qui effectuent un calcul de pose 2D-3D (cf. 3.1.2), les amers sont triangulés dans l'image courante (X_i) et suivis dans l'image suivante (U'_i) pour estimer une paramétrisation Θ de (R, T) en minimisant un critère des moindres carrés sur l'erreur de reprojection :

$$\mathcal{E}(\Theta) = \frac{1}{N} \sum_{i=1}^{N} \left\| U'_i - \Pi\Big(R_{(\Theta)} X_i + T_{(\Theta)}\Big) \right\|^2, \tag{4.32}$$

L'erreur d'estimation des paramètres Θ peut donc être exprimée en fonction des erreurs affectant les données d'entrée $\{X_i, U'_i\}$, en appliquant le théorème des fonctions implicites [Faugeras, 1993] sur la fonction suivante :

$$\begin{gathered} \phi : (\Theta, \mu) \mapsto \frac{\partial \mathcal{E}}{\partial \Theta}(\Theta, \mu)^T, \\ \text{avec : } \mu = \{U'_i, X_i\}_{1 \leq i \leq N}. \end{gathered} \tag{4.33}$$

On obtient alors l'expression de la covariance de Θ avec l'équation 4.10, qui se réécrit :

$$\Sigma_\Theta = \left(\frac{\partial^2 \mathcal{E}}{\partial \Theta \partial \Theta}\right)^{-1} \left(\sum_{i=1}^{N} \left(\frac{\partial \phi}{\partial \mu_i}\right) \Sigma_{\mu_i} \left(\frac{\partial \phi}{\partial \mu_i}\right)^T \right) \left(\frac{\partial^2 \mathcal{E}}{\partial \Theta \partial \Theta}\right)^{-T}, \tag{4.34}$$

avec :

$$\Sigma_{\mu_i} = \left(\begin{array}{c|c} \Sigma_{U'_i} & 0_{2\times 3} \\ \hline 0_{3\times 2} & \Sigma_{X_i} \end{array}\right) \tag{4.35}$$

Nous choisissons de modéliser l'incertitude de suivi d'amer $\Sigma_{U'_i}$ de la même façon que le flot optique (cf. sous-section 4.1.3) en posant :

$$\Sigma_{U'_i} = \sigma_u^2 \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{4.36}$$

$\Sigma_{X_i(\mathbf{x},d)}$ est calculée suivant la formule donnée à l'équation (4.30) et dépend donc des coordonnées 3D triangulées $X_i(\mathbf{x}, d)$.

Propagation d'erreur dans X_{pred} et jusqu'au résidu M

Les grandeurs résiduelles M présentées en Sec. 4.2 dépendent de la prédiction statique X_{pred} donnée à l'équation 4.26, et de la mesure de grandeurs Y_{mes} qui incluent le déplacement des objets dans la scène. Ces résidus résultent de la comparaison de X_{pred} et de Y_{mes} pour permettre de détecter les objets mobiles. Ils s'écrivent sous la forme :

$$M = F\Big(X_{\text{pred}}\Big) - G\Big(Y_{\text{mes}}\Big) \tag{4.37}$$

Typiquement dans la litterature : X_{pred} est calculé pour chaque pixel $\mathbf{x}$ de l'image I_t à partir de la disparité d_t et du déplacement (R,T) de la caméra entre t et $t+1$, et Y_{mes} correspond à la disparité d_{t+1} et au flot optique entre t_t et I_{t+1}. L'incertitude de Y_{mes} peut être estimée en utilisant les modèles présentés aux sous-sections 4.1.2 et 4.1.3. Quant à l'incertitude de $X_{\text{pred}}(\mathbf{x}, d, \Theta)$, elle peut être estimée à partir des incertitudes de triangulation de d'odométrie visuelle modélisées précédemment, par linéarisation :

$$\Sigma_{X_{\text{pred}}} = R\Sigma_{X(\mathbf{x},d)}R^T + J_{X_{\text{pred}}(\Theta)}\Sigma_\Theta J^T_{X_{\text{pred}}(\Theta)}, \tag{4.38}$$

avec :

$$J_{X_{pred}(\Theta)} = \Big(\frac{\partial R}{\partial \theta_x}X(\mathbf{x},d) \;\Big|\; \frac{\partial R}{\partial \theta_y}X(\mathbf{x},d) \;\Big|\; \frac{\partial R}{\partial \theta_z}X(\mathbf{x},d) \;\Big|\; Id_3 \Big). \tag{4.39}$$

Finalement on peut estimer l'incertitude du résidu M en linéarisant de la même façon :

$$\Sigma_{M(\mathbf{x})} = \frac{\partial F}{\partial X}\Sigma_{X_{\text{pred}}(\mathbf{x})}\frac{\partial F}{\partial X}^T + \frac{\partial G}{\partial Y}\Sigma_{Y_{\text{mes}}(\mathbf{x})}\frac{\partial G}{\partial Y}^T \tag{4.40}$$

4.2 Choix de la méthode et de la gandeur résiduelle M

Divers approches géométriques ont été proposées dans la littérature pour la détection dense d'objets mobiles. Ces approches suivent le même principe : elles calculent une grandeur résiduelle $M(\mathbf{x}) \in \mathbb{R}^N$ indicée par le pixel $\mathbf{x}$ de l'image de référence (en général l'image gauche à l'instant t) et égale à l'écart entre des grandeurs observées incluant le mouvement des objets dans la scène, et des grandeurs prédites en supposant la scène statique. Ensuite, connaissant un modèle de l'incertitude sur le résidu $M(\mathbf{x})$, sa norme $\chi^2(M(\mathbf{x}))$ est calculée et utilisée comme critère de détection d'objet mobile en tout pixel $\mathbf{x}$. En pratique, le résidu M est calculé à partir du flot de scène $(\mathbf{u}, d, d')$ et du déplacement de la caméra (R,T) entre les deux instants, dont les méthodes d'estimation ont été présentées aux sections 3.3 et 3.1. En propageant les erreurs de mesure, comme montré à la section 4.1, on peut estimer la matrice de covariance d'erreur Σ_M, et calculer :

$$\chi^2(M) = M^T\Sigma_M^{-1}M$$

Le seuillage de cette norme χ^2 détermine les zones de l'image qui ne satisfont pas l'hypothèse de scène statique, c'est-à-dire les objets mobiles.

Plusieurs grandeurs résiduelles M et plusieurs méthodes pour calculer ces résidus, ont été proposées dans la littérature. Dans cette section 4.2 nous présentons ces différentes méthodes et critères de détection que nous comparons qualitativement à partir des cartes $\chi^2(M)$. Il est important de noter que les résultats des différentes méthodes varient non seulement en fonction du choix des grandeurs considérées (disparité, flot, image résiduelle) mais aussi selon les algorithmes sélectionnés pour estimer ces grandeurs, parmi ceux présentés aux Chapitres 2 et 3.

4.2.1 Méthodes de détection et résidus utilisés dans la littérature

Plusieurs résidus M ont été proposées dans la littérature, pour la détection dense d'objets mobiles en stéréo. Certains auteurs comme Wedel et al [Wedel et al., 2011], et Alcantarilla et al [Alcantarilla et al., 2012] considèrent la différence entre la position 3D d'un point suivi à $t+1$ et sa position prédite en le supposant fixe :

$$\begin{aligned} M(\mathbf{x}) &= X_{t+1}(\mathbf{x}) - X_{t+1}^{pred}(\mathbf{x}) \\ &= X_{t+1}(\mathbf{x}+\mathbf{u}, d_{t+1}) - \big(RX(\mathbf{x}, d_t) + T\big), \end{aligned} \tag{4.41}$$

où la notation $X_t(\mathbf{x}, d_t)$ désigne les coordonnées 3D dans le repère caméra gauche à l'instant t, d'un point triangulé à t à partir d'un pixel $\mathbf{x}$ de l'image gauche et de son correspondant $\mathbf{x} + (d_t, 0)$ dans l'image droite.

Il s'agit donc ici d'un résidu de dimension 3 : $M(\mathbf{x}) \in \mathbb{R}^3$. D'autres travaillent directement en géométrie image, en considérant les écarts entre prédiction et observation pour des grandeurs telles que la position d'un pixel dans l'image, la disparité, le flot optique, ou la variation de disparité $p(\mathbf{x}) = d_{t+1}(\mathbf{x}+\mathbf{u}) - d_t(\mathbf{x})$. Talukder et Matthies [Talukder and Matthies, 2004] par exemple, prennent le critère :

$$M(\mathbf{x}) = \big(u(\mathbf{x}), v(\mathbf{x}), p(\mathbf{x})\big)^T - \big(u_{pred}(\mathbf{x}), v_{pred}(\mathbf{x}), p_{pred}(\mathbf{x})\big)^T \tag{4.42}$$

Les résidus tridimensionnels donnés aux équations (4.41) et (4.42) sont relativement similaires car leur calcul repose sur les mêmes paramètres : $\{\mathbf{u}, d_t, d_{t+1}\}$ et le déplacement (R, T) de la caméra. La seule différence est la manière de combiner ces paramètres, qui modifie la façon dont sont propagées les erreurs de mesures dans le modèle d'incertitude de M. Puisque les grandeurs directement mesurées sont le flot optique et les cartes de disparité, il est plus logique d'utiliser telles quelles ces estimations dont nous savons modéliser l'incertitude (cf. section 4.1), plutôt que de les combiner (e.g. pour calculer $X(\mathbf{x}+\mathbf{u}, d_{t+1})$) et devoir ensuite propager (approximativement) les erreurs de mesures à l'ordre 1. Dans la suite, en ce qui concerne l'utilisation d'un résidu 3D, nous choisissons donc l'écart entre le flot optique et la disparité directement observés et leur prédiction statique :

$$M(\mathbf{x}) = (u(\mathbf{x}), v(\mathbf{x}), d_{t+1}(\mathbf{x}))^T - (u_{pred}(\mathbf{x}), v_{pred}(\mathbf{x}), d_{t+1}^{pred}(\mathbf{x}))^T, \tag{4.43}$$

une forme très proche de (4.42) [Talukder and Matthies, 2004], mais dans laquelle on utilise directement la disparité d plutôt que sa variation.

Nous considérons également l'utilisation d'un résidu M bidimensionnel, comme le font Romero-Cano et Nieto [Romero-Cano and Nieto, 2013] en considérant uniquement l'écart entre le flot optique observé et le flot optique prédit :

$$M(\mathbf{x}) = \mathbf{u}(\mathbf{x}) - \mathbf{u}_{pred}(\mathbf{x}) \tag{4.44}$$

Dans les résidus précédents apparaît le flot optique résiduel $\delta\mathbf{u} = \mathbf{u} - \mathbf{u}_{pred}$ qui peut être calculé de deux manières différentes. Les méthodes dites *directes* prédisent un mouvement entre t et $t+1$ en utilisant la disparité et le mouvement de la caméra, et définissent le flot résiduel comme la différence entre le flot estimé et le flot prédit. Une autre approche, proposée notamment par Bak et al. [Bak et al., 2014], consiste à prédire une image I_{t+1}^{pred} et à définir le flot résiduel comme le flot optique calculé entre I_{t+1} et I_{t+1}^{pred}. Cette approche dite par prédiction d'image, est détaillée dans la sous-section 4.2.3.

Ainsi, pour la détection dense de mouvement dans l'image, nous avons le choix de considérer un résidu M tri- ou bi-dimensionnel, et d'utiliser une méthode directe ou par prédiction d'image. Dans les sous-sections suivantes, nous décrivons les détails du calcul des résidus M selon chacune de ces méthodes.

Avant de rentrer dans le détail des méthodes, il convient de clarifier un point concernant le sens (direct ou bien rétrograde) du calcul du résidu. Dans la littérature, la détection s'effectue dans le référentiel de l'image t, à partir des paires stéréo à t et $t+1$, ce qui correspond à un calcul en sens *rétrograde*. Afin de limiter la latence, nous choisissons d'effectuer la détection en sens *direct*, c'est-à-dire dans l'image t à partir des paires stéréo t et $t-1$. Nous inversons donc la direction temporelle dans l'écriture des résidus M, en remplaçant les indices $t+1$ par $t-1$. Ainsi, le flot optique dans les *méthodes directes* est estimé de I_t vers I_{t-1}, et le déplacement de la caméra utilisé est le déplacement rétrograde de I_t vers I_{t-1} qui s'écrit :

$$X_{t-1} = R^T(X_t - T) \tag{4.45}$$

4.2.2 Méthodes directes : expressions des résidus

Dans les *méthodes directes*, le flot optique $\mathbf{u}$ estimé de I_t à I_{t-1} et la disparité $d_{t-1}(\mathbf{x}+\mathbf{u})$ sont comparés à leur prédiction :

$$\mathbf{u}_{pred}(\mathbf{x}) = \Pi\big(X_{t-1}^{pred}(\mathbf{x}, d_t)\big) - \mathbf{x}, \tag{4.46}$$

$$d_{t-1}^{pred}(\mathbf{x}) = \frac{-bf_x}{\begin{pmatrix}0 & 0 & 1\end{pmatrix}^T X_{t-1}^{pred}(\mathbf{x}, d_t)}, \tag{4.47}$$

où

$$X_{t-1}^{pred}(\mathbf{x}, d_t) = R^T\big(X(\mathbf{x}, d_t) - T\big), \tag{4.48}$$

Les résidus 2D et 3D considérés sont :

- $M(\mathbf{x}) = \{\mathbf{u}(\mathbf{x}) - \mathbf{u}_{pred}(\mathbf{x})\}$
- $M(\mathbf{x}) = \{\mathbf{u}(\mathbf{x}) - \mathbf{u}_{pred}(\mathbf{x}), d_{t-1}(\mathbf{x}+\mathbf{u}) - d_{t-1}^{pred}(\mathbf{x})\}$

Les prédictions $\mathbf{u}_{pred}$ et d_{t-1}^{pred} étant fonctions de X_{t-1}^{pred}, on peut calculer leur matrice d'incertitude $\Sigma_{\mathbf{u}_{pred},d_{t-1}^{pred}}$, comme expliqué à la sous-section 4.1.5, à partir de $\Sigma_{X_{t-1}^{pred}}$ dont l'expression est donnée à l'équation 4.38 :

$$\Sigma_{\{\mathbf{u}_{pred},d_{t-1}^{pred}\}} = \frac{\partial(\mathbf{u}_{pred},d_{pred})}{\partial X_{pred}} \Sigma_{X_{pred}} \frac{\partial(\mathbf{u}_{pred},d_{pred})}{\partial X_{pred}}^T, \tag{4.49}$$

où

$$\frac{\partial(\mathbf{u}_{pred},d_{pred})}{\partial X_{pred}} = \frac{1}{\begin{pmatrix}0 & 0 & 1\end{pmatrix}^T X_{pred}} \left(\begin{array}{cc|c} f_x & 0 & \begin{pmatrix}x_0\\y_0\end{pmatrix} - \Pi(X_{pred}) \\ 0 & f_y & \\ \hline 0 & 0 & -d_{pred}\end{array}\right) \tag{4.50}$$

Quant aux valeurs mesurées $\mathbf{u}(\mathbf{x})$ et $d_{t-1}(\mathbf{x}+\mathbf{u})$, leur erreurs sont modélisées par des probabilités de distribution gaussiennes : $\mathcal{N}\left(0,\sigma_u^2 Id_2\right)$ et $\mathcal{N}\left(0,\sigma_d^2\right)$ (cf. section 4.1). On peut souligner que cela revient à modéliser l'incertitude de $d_{t-1}(\mathbf{x}+\mathbf{u})$ en considérant uniquement l'erreur de mesure sur d_{t-1}, sans tenir compte de l'incertitude sur $\mathbf{x}+\mathbf{u}(\mathbf{x})$. Ainsi on écrit :

$$\Sigma_{\{\mathbf{u},d_{t-1}\}} = \begin{pmatrix}\sigma_u^2 & 0 & 0\\ 0 & \sigma_u^2 & 0\\ 0 & 0 & \sigma_d^2\end{pmatrix} \tag{4.51}$$

Dans la pratique, on choisit $\sigma_{\mathbf{u}} = 1\text{px}$ et $\sigma_d = 1\text{px}$.

Puisque les valeurs mesurées $\mathbf{u}(\mathbf{x})$ et $d_{t-1}(\mathbf{x}+\mathbf{u})$ sont obtenues indépendamment des valeurs prédites, la matrice de covariance d'erreur de M est égale à la somme des matrices de covariance d'erreurs des mesures et des prédictions, soit :

$$\Sigma_M = \Sigma_{\{\mathbf{u},d_{t-1}\}} + \Sigma_{\{\mathbf{u}_{pred},d_{t-1}^{pred}\}} \tag{4.52}$$

En appliquant le même raisonnement on peut écrire l'incertitude du résidu M en 2D. Disposant de M et de Σ_M, la norme $\chi^2(M)$ peut être calculée.

4.2.3 Méthode par prédiction d'image : détails des critères M utilisés

Les *méthodes par prédiction d'image* utilisent une image prédite comme intermédiaire de calcul pour estimer le flot optique résiduel $\delta\mathbf{u}$ entre celle-ci et l'image observée. Ce résidu $\delta\mathbf{u}$ peut ensuite être pris comme critère M, ou bien on peut ajouter une troisième dimension en calculant $(d_{t-1} - d_{t-1}^{pred})$ comme présenté à la sous-section 4.2.2.

Prédiction d'image : méthode de [Bak et al., 2014] et limitations

Dans les travaux de Bak et al [Bak et al., 2014], la détection de mouvement entre deux instants notés t_0 et t_1, s'effectue en estimant le flot optique résiduel entre

l'image prédite $I_{t_0}^{pred}$ et I_{t_1}. Dans notre cas, si nous appliquons la stratégie de Bak *et al.*, il faut calculer l'image prédite I_{t-1}^{pred} selon :

$$I_{t-1}^{pred}\Big(\mathbf{x}_{t-1}^{pred}(\mathbf{x})\Big) \leftarrow I_t(\mathbf{x}), \tag{4.53}$$

avec

$$\mathbf{x}_{t-1}^{pred}(\mathbf{x}) = \Pi\Big(X_{t-1}^{pred}(\mathbf{x}, d_t)\Big). \tag{4.54}$$

Lors de cette prédiction, la grille régulière des pixels $\{\mathbf{x}\}$ dans I_t est transformée en un ensemble de coordonnées $\{\mathbf{x}_{t-1}^{pred}(\mathbf{x})\}$ dans le référentiel $t-1$, qui ne sont plus des valeurs entières. Pour synthétiser une image complète I_{t-1}^{pred} il faut donc interpoler ces valeurs irrégulièrement espacées, ce qui est une opération coûteuse. De plus, Bak *et al.* définissent le flot optique résiduel $\delta\mathbf{u}$ comme le flot estimé entre I_{t-1}^{pred} et I_{t-1}, qui vérifie l'équation :

$$I_{t-1}^{pred}(\mathbf{x}') = I_{t-1}(\mathbf{x}' + \delta\mathbf{u}(\mathbf{x}')) \tag{4.55}$$

En réfléchissant à l'équation (4.55) on remarque deux choses :

1) la géométrie de référence du flot résiduel est celle de I_{t-1}^{pred}, qui n'est pas forcément parfaitement alignée avec celle de I_{t-1} : une erreur est susceptible de s'ajouter ici ;

2) le flot résiduel estimé ne peut plus être relié aux coordonnées prédites ($\mathbf{x}_{t-1}^{pred}$) à cause de l'opération d'interpolation des intensités qui fait que I_{t-1}^{pred} dépend en fait de plusieurs coordonnées $\mathbf{x}_{t-1}^{pred}$ pour des indices différents au voisinage de la position interpolée.

Ces changements de géométrie ne sont pas sans conséquence, car elles imposent d'interpoler non seulement les valeurs de coordonnées $\mathbf{x}_{t-1}^{pred}$, mais aussi les matrices de covariance $\Sigma_{\mathbf{x}_{t-1}^{pred}}$ pour pouvoir définir le résidu M et sa norme $\chi^2(M)$. Au final, il va se produire une accumulation d'imprécisions issues de ces interpolations multiples qui sera dommageable à la validité de la quantité $\chi^2(M)$ effectivement seuillée en chaque pixel.

Pour pouvoir modéliser l'incertitude sur M il est donc nettement préférable de pouvoir écrire M directement comme une fonction de X_{t-1}^{pred} (cf. 4.1.5). C'est pourquoi nous proposons en section suivante, une autre approche de prédiction d'image.

Prédiction d'image : méthode proposée

La méthode que nous proposons pour la prédiction d'image est calquée sur celle que nous avons présentée à la sous-section 3.3.3 pour décrire la stratégie PCOF d'estimation du flot optique. Elle consiste à prédire I_t :

$$I_t^{pred}(\mathbf{x}) \leftarrow I_{t-1}\Big(\Pi\big(X_{t-1}^{pred}(\mathbf{x}, d_t)\big)\Big) = I_{t-1}\Big(\mathbf{x}_{t-1}^{pred}(\mathbf{x})\Big) \tag{4.56}$$

De même que dans [Bak et al., 2014], lorsque $I_t^{pred}(\mathbf{x})$ ne peut pas être définie suivant l'équation 4.56 (i.e. parce que $\mathbf{x}_{pred}(\mathbf{x})$ sort des frontières de l'image I_{t-1}),

c'est l'intensité $I_t(\mathbf{x})$ qui est prise comme valeur. Nous proposons d'estimer le flot optique résiduel $\delta\mathbf{u}$ entre I_t et I_t^{pred} et de l'utiliser ensuite comme résidu M pour la détection. On note que cette fois ci $\delta\mathbf{u}(\mathbf{x})$ est bien défini sur une grille régulière de pixels dans la géométrie de I_t car il vérifie :

$$I_t(\mathbf{x}) = I_t^{pred}(\mathbf{x} + \delta\mathbf{u}(\mathbf{x})) \tag{4.57}$$

Si l'on souhaite prendre un résidu M à trois dimensions, il faut donc calculer la disparité résiduelle :

$$\delta d(\mathbf{x}) = d_{t-1}(\mathbf{x} + \mathbf{u}(\mathbf{x})) - d_{t-1}^{pred}(\mathbf{x}) \tag{4.58}$$

On remarque que cette expression fait apparaître le flot optique "total" $\mathbf{u}$ (et non le flot résiduel $\delta\mathbf{u}$). Il faut donc calculer ce flot optique, ce que l'on fait avec la stratégie PCOF présentée en sous-section 3.3.3 (cf. Eq. (3.62)) :

$$\mathbf{u}(\mathbf{x}) = \delta\mathbf{u}(\mathbf{x}) + \mathbf{u}_{pred}\big(\mathbf{x} + \delta\mathbf{u}(\mathbf{x})\big) \tag{4.59}$$

Pour un pixel $\mathbf{x}$ correspondant à un point statique, le flot optique résiduel $\delta\mathbf{u}$ entre I_t et I_t^{pred} est sensé être nul, sauf erreurs d'estimation et de prédiction que nous modélisons par des variables gaussiennes de moyennes nulles et de covariances $\Sigma_{\mathbf{u}}$ et $\Sigma_{\mathbf{u}_{pred}}$, de la même manière qu'à la sous-section 4.2.2. Pour calculer $\chi^2(M)$, on applique donc aux résidus $M = \delta\mathbf{u}$ et $M = \{\delta\mathbf{u}, \delta d\}$ le même modèle d'incertitude Σ_M que sur les résidus M obtenus par méthode directe.

4.2.4 Comparaison des différents critères et méthodes de détection

Parmi les méthodes et les résidus M détaillées aux sous-sections 4.2.2 et 4.2.3, nous testons ici chaque possibilité avec les algorithmes de vision stéréo temps-réel présentés aux Chapitres 2 et 3. En particulier nous considérons FOLKI (cf. 3.2.5) et PCOF (cf. 3.3.3) pour l'estimation temps-réel du flot optique, et ACTF (cf. 2.2.1) pour l'estimation de la carte de disparité. [1] L'objectif est d'établir quel critère M (2D ou 3D) et quelle méthode (directe, ou par prédiction d'image) conviennent le mieux avec les algorithmes que nous utilisons.

La figure 4.2 montre sur un cas particulier issu de la base d'apprentissage de KITTI, les cartes $\chi^2(M)$ de probabilité de mouvement obtenues avec les différentes approches. Notons que dans le cas de la méthode directe, les cartes $\chi 2(M)$ calculées avec PCOF ou avec FOLKI sont très différentes car le flot optique "final" $\mathbf{u}$ utilisé pour calculer M est alors très différent. Au contraire, pour la méthode par prédiction d'image, le résidu M est calculé à partir du flot résiduel $\delta\mathbf{u}$ qui est strictement

1. En pratique nous avons remarqué que les cartes $\chi^2(M)$ de probabilité de mouvement sont particulièrement sensibles aux valeurs aberrantes dans l'estimation de la disparité. Ainsi il vaut mieux appliquer un masque pour soustraire les pixels dont la disparité est incertaine (e.g. suivant un critère aller-retour : comme dans ELAS, cf. 2.2.1), que de chercher à "densifier" puis raffiner la carte de disparité par des interpolations et post-traitements. C'est pourquoi nous employons ici uniquement ACTF avec masque *aller-retour*, sans appliquer de post-traitement par *Consensus de Régions*.

identique avec ces deux algorithmes car PCOF utilise FOLKI pour estimer le flot entre I_t et I_tpred. C'est pourquoi dans la figure 4.2, on distingue l'utilisation de PCOF et de FOLKI seulement dans le cas de la méthode directe. Les résultats montrés à la figure 4.2 nous permettent de faire plusieurs constats :

- L'ajout de δd pour obtenir un résidu M tridimensionnel entraîne une très forte augmentation du bruit dans la carte $\chi^2(M)$: dans les cartes présentées dans la colonne de droite, en effet, de nombreuses régions statiques présentent à tort une très forte probabilité de mouvement. En effet, δd est particulièrement sensible à la qualité de la disparité, notamment dans les zones où la disparité n'est pas constante (e.g. discontinuité, feuillage). De plus, cet ajout de δd dans le résidu M n'entraîne aucun gain apparent, en particulier il ne conduit pas vraiment à une augmentation du $\chi^2(M)$ au niveau des objets mobiles.
- La méthode directe est très sensible à la précision du flot optique utilisé, et notamment aux erreurs au niveau des discontinuités des objets. Ceci est plutôt logique puisque le flot optique est directement comparé au flot optique prédit calculé à partir de (R, T) et de la carte de disparité d qui, elle, respecte bien les discontinuités.
- L'utilisation de la méthode directe en utilisant PCOF pour le flot optique, donne des résultats très similaires (mais pas exactement identiques) à ceux de la méthode par prédiction d'image. Ceci est normal car dans la méthode directe M est calculé à partir de $(\mathbf{u} - \mathbf{u}_{pred})$ qui est différent de $\delta\mathbf{u}$, étant donné la formule de $\mathbf{u}$ rappelée Eq. (4.59). En effet, les résidus M calculés respectivement par la méthode par prédiction d'image, et par la méthode directe avec PCOF sont :

$$\begin{cases} M(\mathbf{x}) = \delta\mathbf{u}(\mathbf{x}) \\ M(\mathbf{x}) = \delta\mathbf{u}(\mathbf{x}) + \mathbf{u}_{pred}(\mathbf{x} + \delta\mathbf{u}) - \mathbf{u}_{pred}(\mathbf{x}) \end{cases} \tag{4.60}$$

Une comparaison plus approfondie des différentes méthodes et algorithmes à utiliser pour l'estimation du flot optique et de la disparité stéreo est présentée à la section 4.4, en évaluant quantitativement les performances en termes de détection d'objets mobiles. Pour effectuer une telle évaluation il faut d'abord disposer d'une méthode de détection des objets mobiles dans l'image. C'est l'objet de la section suivante où nous étudions la segmentation d'objets mobiles à partir de la carte $\chi^2(M)$.

4.3 Segmentation d'objets mobiles à partir de $\chi^2(M)$

4.3.1 Segmentation d'objets mobiles dans la littérature

Dans la littérature, la plupart des auteurs se contentent de seuiller la carte $\chi^2(M)$ qu'ils ont calculée, pour déterminer les pixels appartenant à l'image d'objets en mouvement dans la scène [Alcantarilla et al., 2012, Romero-Cano and Nieto, 2013]. D'un point de vue statistique cela revient à effectuer indépendamment pour chaque

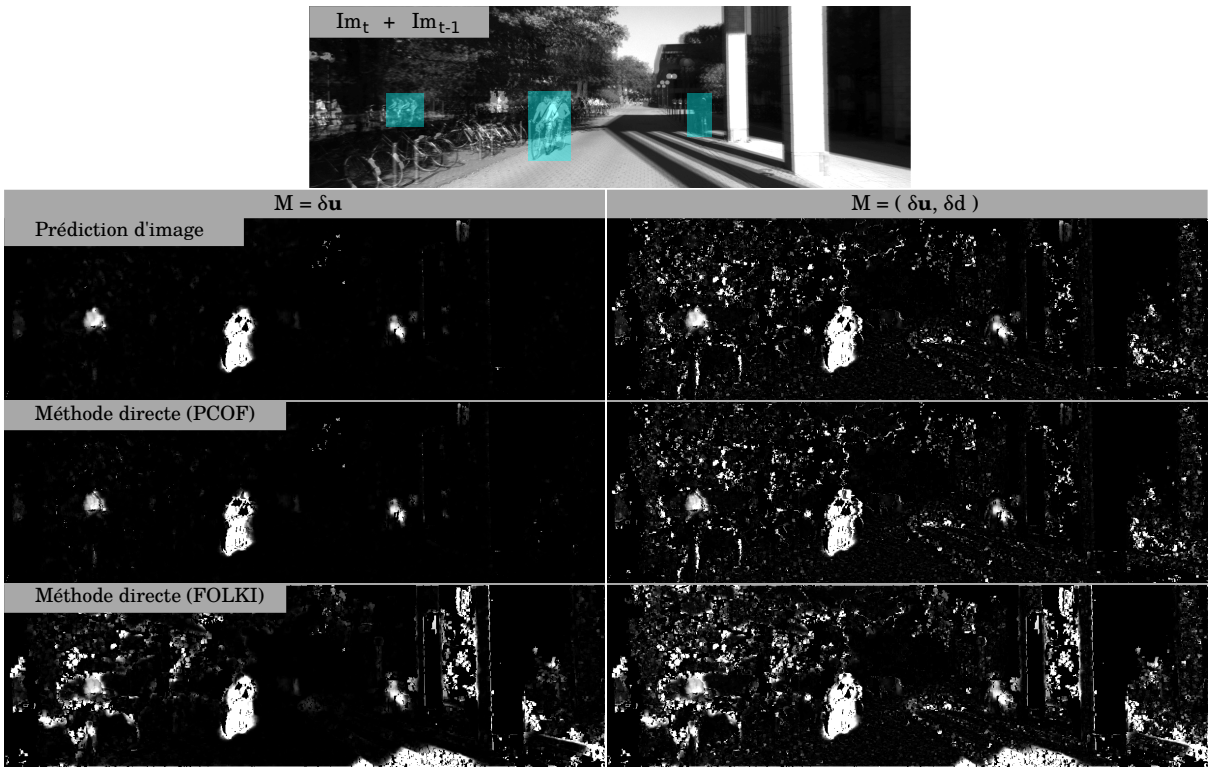

FIGURE 4.2 – Cartes $\chi^2(M)$ sur une échelle de 0 (noir) à 20 (blanc), calculées avec un critère 2D (colonne gauche) et 3D (colonne droite), en appliquant la méthode par prédiction d'image et la méthode directe avec FOLKI et PCOF. Dans le cas de la méthode par prédiction d'image l'utilisation de PCOF est équivalente à celle de FOLKI.

pixel un test du χ^2 permettant d'invalider ou non l'hypothèse que le pixel est statique. Cependant comme nous avons vu à la section 4.2, la carte $\chi^2(M)$ estimée peut être relativement bruitée (e.g. au niveau des discontinuités), ce qui nous incite à employer plus qu'un simple seuillage.

En pratique les objets en mouvement que l'on détecte ne se réduisent pas à un pixel mais à ensemble connexe de pixels. C'est pourquoi une technique de *Graph-Cut* est utilisé par Wedel et al [Wedel et al., 2011, Wedel et al., 2009] pour effectuer une segmentation binaire de l'image (statique ou en mouvement), en tenant compte de la valeur $\chi^2(M)$ en chaque pixel mais également en incorporant un terme de régularisation spatiale qui tend à associer le même label aux pixels voisins d'apparence similaire.

Cependant, dans un contexte de vision embarquée en robotique, la contrainte sur le temps de calcul est prépondérante. Ce sont donc des méthodes plus simples qui doivent être envisagées. Talukder et Matthies [Talukder and Matthies, 2004] proposent d'extraire les composantes connexes dans l'image $\chi^2(M)$ seuillée, afin de fusionner celles qui sont voisines et de supprimer celles dont les surfaces sont trop

réduites. Pour ce faire, les auteurs classent d'abord les pixels de la carte binaire de détection (e.g. $\chi^2(M)$ seuillée) selon leur profondeur, en leur attribuant un plan de profondeur Z_{PLAN} parmi un ensemble de plans répartis par pas de 5m. Ils raisonnent ensuite en 2D, en fusionnant les composantes connexes de même profondeur Z_{PLAN}, dont les centres (x_0^c, y_0^c) et (x_1^c, y_1^c) sont voisins, avec l'approximation :

$$dist(\mathbf{x_0^c}, \mathbf{x_1^c})^2 = \left(\frac{Z_{PLAN}}{f_x}(x_0^c - x_1^c)\right)^2 + \left(\frac{Z_{PLAN}}{f_y}(y_0^c - y_1^c)\right)^2 \tag{4.61}$$

4.3.2 Méthode proposée pour la segmentation d'objets mobiles

La méthode que nous proposons pour effectuer la segmentation d'objets mobiles dans l'image est proche de celle de [Talukder and Matthies, 2004]. En voici les différentes étapes :

1. binarisation de la carte $\chi 2(M)$ seuillée pour une valeur de seuil τ.
2. extraction des composantes connexes (CC) à partir de la carte binaire du $\chi^2(M)$ seuillé
3. à chaque CC est attribuée une profondeur Z_{CC} calculée à partir de la disparité médiane de la CC
4. fusion des CC voisines en considérant que chaque CC est une surface rectangulaire définie par sa boîte englobante et sa pronfondeur Z_{CC}
5. chaque groupe de CC fusionnées représente un objet mobile auquel on attribue une profondeur Z_{Obj} estimée à partir de la disparité médiane
6. supression de chaque groupe de CC (i.e. "objet mobile") dont la surface $\mathcal{S}_{Obj}$, calculée à partir de la profondeur Z_{Obj} et du nombre de pixels N_{px} qui le constituent, est trop petite :

$$\mathcal{S}_{Obj} = N_{px}\left(\frac{Z_{Obj}^2}{f_x f_y}\right) \tag{4.62}$$

En sortie de cette segmentation nous disposons donc de l'image binaire de détection des pixels en mouvement, dans laquelle une partie des fausses détections a été supprimée (par la suppression des surfaces détectées trop petites). Nous avons également regroupé les pixels détectés comme mobiles en plusieurs objets, ce qui nous donne une segmentation des objets mobiles dans l'image et nous permet de définir des boîtes englobantes autour de ces derniers.

4.3.3 Modèle de région d'intérêt utilisé pour la détection

Dans le cas d'un banc stéréo monté sur une plate-forme terrestre (comme une voiture ou bien le robot que l'on utilise au Chapitre 6), si l'on suppose le sol plat alors on peut restreindre la perception aux régions situées entre le sol et une certaine hauteur H_{max}.

C'est ce que proposent Romero-Cano et al. [Romero-Cano and Nieto, 2013] qui suppriment les éléments de plus de 2m de haut dans la scène. Pour ce faire, ils estiment le plan correspondant au sol par une méthode des moindres carrés, pour calculer la hauteur de chaque point 3D observé. Ils projettent ensuite l'ensemble des points observés dans une grille polaire, et suppriment les points appartenant aux cellules dont la hauteur dépasse 2m. En faisant cela, ils limitent les fausses détections qui sont fréquentes au niveau des structures fines commes les poteaux et lampadaires en milieu urbain.

Dans notre cas, nous souhaitons considérer comme régions d'intérêt l'ensemble des points situés entre deux hauteurs H_{min} et H_{max} pour réduire le nombre de fausses détections et éviter de détecter les ombres des objets mobiles sur le sol. Pour cela nous utilisons une méthode beaucoup plus rapide, en considérant que la caméra est positionnée à une hauteur H_{cam}, avec une inclinaison d'angle ϕ par rapport à l'horizontale comme illustré à la figure 4.3. Connaissant H_{cam} et ϕ, on peut alors déterminer pour chaque pixel (x, y) de disparité d, s'il correspond à un point situé entre $H_{min}(< H_{cam})$ et $H_{max}(> H_{cam})$:

$$H_{min} < h \iff d < \cos(\phi)\frac{\alpha(y - y_0) + \beta}{H_{cam} - H_{min}} \tag{4.63}$$

$$h < H_{max} \iff d < \cos(\phi)\frac{\alpha(y - y_0) - \beta}{H_{max} - H_{cam}}, \tag{4.64}$$

où

$$\begin{aligned} \alpha &= -\frac{bf_x}{f_y} \\ \beta &= -bf_x \tan(\phi) \end{aligned} \tag{4.65}$$

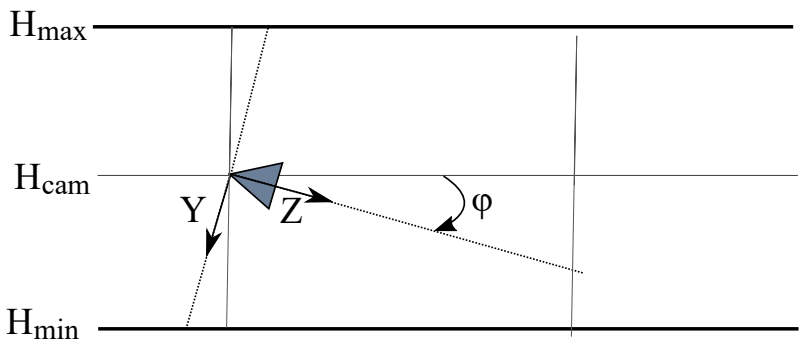

FIGURE 4.3 – Illustration du modèle considéré pour les régions d'intérêts : la caméra située à hauteur H_{cam} est inclinée d'un angle ϕ par rapport à l'horizontale.

L'évaluation de telles régions d'intérêt ($H_{min} < h < H_{max}$) s'effectue indépendament pour chaque pixel au moment du calcul $\chi^2(M)$ —qui prend la valeur 0 en dehors de ces régions. Cette évaluation ne demande donc en pratique aucun temps de calcul supplémentaire et convient parfaitement pour une détection temps-réel embarquée.

Pour les séquences KITTI que nous considérons pour l'évaluation à la section 4.4, nous prenons : $H_{cam} = 1.65m$, $H_{min} = 0.10m$, $H_{max} = 2.50m$, $\tan(\phi) = 0.02$ et $\cos(\phi) \approx 1$.

4.4 Évaluation des algorithmes en termes de performance de détection d'objets mobiles

Maintenant que nous avons vu comment détecter les objets mobiles dans l'image en segmentant la carte de probabilité de mouvement $\chi^2(M)$, nous pouvons comparer les différentes méthodes de calcul de M présentées à la section 4.2, en étudiant leur performance en termes de détection.

4.4.1 Protocole d'évaluation

Pour l'évaluation, nous avons manuellement annoté les objets mobiles sur deux séquences de la base de données KITTI [Geiger et al., 2012] : 28/09-drive0037 (cf. Fig. 4.6) et 29/09-drive0071 (cf. Fig. 4.7). La première dure environ 9s et comporte un carrefour, avec un fond végétal et inclut un mouvement de rotation à 90 degrés de l'observateur. La seconde est acquise en zone urbaine dense avec des véhicules et des piétons sur le trottoir et traversant la chaussée, le trajet décrit est en ligne droite et nous nous sommes limité à l'annotation des 200 premières images (soit 20s puisque la fréquence d'acquisition est à 10Hz). Cette annotation manuelle a été faite en utilisant l'outil VATIC [Vondrick et al., 2013]. Ainsi pour chaque objet en déplacement, nous disposons d'une vérité terrain sous la forme d'une boîte englobante autour de celui-ci.

Les boîtes englobantes (BE) des objets mobiles détectés avec l'algorithme et les régions d'intérêt présentés à la section 4.3, sont comparées aux boîtes englobantes de la vérité terrain ($\mathrm{BE_{VT}}$) en considérant comme critère le taux de recouvrement défini par :

$$\tau_R\big(\mathrm{BE}, \mathrm{BE_{VT}}\big) = \frac{\mathcal{A}_{\mathrm{BE}\cap\mathrm{BE_{VT}}}}{\mathcal{A}_{\mathrm{BE}\cup\mathrm{BE_{VT}}}}, \tag{4.66}$$

On considère qu'une BE estimée par l'algorithme de détection est bonne lorsqu'il existe une $\mathrm{BE_{VT}}$ telle que : $\tau_R\big(\mathrm{BE}, \mathrm{BE_{VT}}\big) > 20\%$. Si ce n'est pas le cas, cette BE est considérée comme erronée et est comptabilisée dans les *Faux Positifs* (FP). Parfois notre algorithme de détection détecte plusieurs BE sur le même objet mobile. Dans ce cas, même si plusieurs BE sont bonnes (i.e. avec $\tau_R > 20\%$), on ne compte qu'une seule bonne détection dans les *Vrais Positifs* (VP). Enfin, chaque $\mathrm{BE_{VT}}$ qui n'a pas été détectée, est comptée dans les *Faux Négatifs* (FN). Les critères usuels de *Précision* et de *Rappel* peuvent alors être calculés suivant les formules :

$$\mathit{Précision} = \frac{VP}{VP + FP} \tag{4.67}$$

$$\mathit{Rappel} = \frac{VP}{VP + FN} \tag{4.68}$$

On calcule ces valeurs *Précision* et *Rappel* en utilisant différents seuils τ pour la segmentation des objets mobiles dans $\chi^2(M)$. Ces différents couples de valeurs nous permettent de tracer une courbe Précision-Rappel avec les valeurs de rappel sur l'axe X et les valeurs de précision sur l'axe Y.

Cette courbe représente les points de fonctionnement correspondant à différentes valeurs de seuil τ de la carte $\chi^2(M)$ (cf. sous-section 4.3.2). Un seuil bas donne un bon Rappel mais une mauvaise Précision, et au contraire un seuil élevé est sensé augmenter la Précision mais donner une valeur plus faible de Rappel. Habituellement une telle courbe est donc strictement décroissante. Cependant dans notre cas, l'emploi du critère τ_R de taux de recouvrement et les règles choisies pour le calcul des VP/TP/FN font que l'augmentation de τ n'entraîne pas forcément un gain en Précision, et la diminution de τ n'entraîne pas non plus nécessairement un gain en Rappel. Pour donner un exemple, si τ est trop bas, l'aire de la BE détectée peut devenir tellement grande que l'aire de $\mathrm{BE}_{\mathrm{VT}}$ ne représente pas 20% de celle-ci, et la détection n'est donc pas comptée comme un VP. Ceci explique l'aspect parfois en "dents de scie" des courbes présentées dans les sous-sections qui suivent.

4.4.2 Comparaison des méthodes directes et par prédiction d'image

Nous avons vu à la section 4.2 que l'on peut utiliser un résidu 2D ou 3D pour la détection de mouvement, et que l'on peut appliquer une méthode directe ou bien par prédiction d'image pour le calculer. Les résidus 3D étant trop bruités comme on le voit à la figure 4.2, on considère ici uniquement les résidus 2D obtenus avec les différentes méthodes. La figure 4.4 présente les courbes de Précision-Rappel obtenues par méthode directe et par prédiction d'image avec FOLKI et PCOF pour l'estimation du flot optique, et ACTF pour l'appariement stéréo. On rappelle qu'employer FOLKI ou PCOF dans la méthode par prédiction d'image revient au même (cf. section 4.2).

Les courbes de la figure 4.4 montrent que, comme on pouvait s'y attendre, la méthode directe est particulièrement sensible au choix du flot optique utilisé : avec FOLKI, la précision est réduite de manière très significative, alors qu'avec PCOF les performances sont presques identiques à celles de la méthode par prédiction d'image. Par la suite c'est la méthode par prédiction d'image que nous choisissons, qui présente l'avantage d'utiliser uniquement le flot optique résiduel $\delta\mathbf{u}$ et qui ne nécessite pas de calcul et d'interpolation supplémentaires comme c'est le cas avec PCOF et la méthode directe (cf. équation 4.60).

4.4.3 Étude de l'influence du choix des algorithmes sur les performances de détection

Maintenant que nous avons choisi la méthode par prédiction d'image pour calculer le résidu M (donc $= \delta\mathbf{u}$, le flot résiduel) nous pouvons nous demander dans quelle mesure les performances de détection dépendent des choix algorithmiques

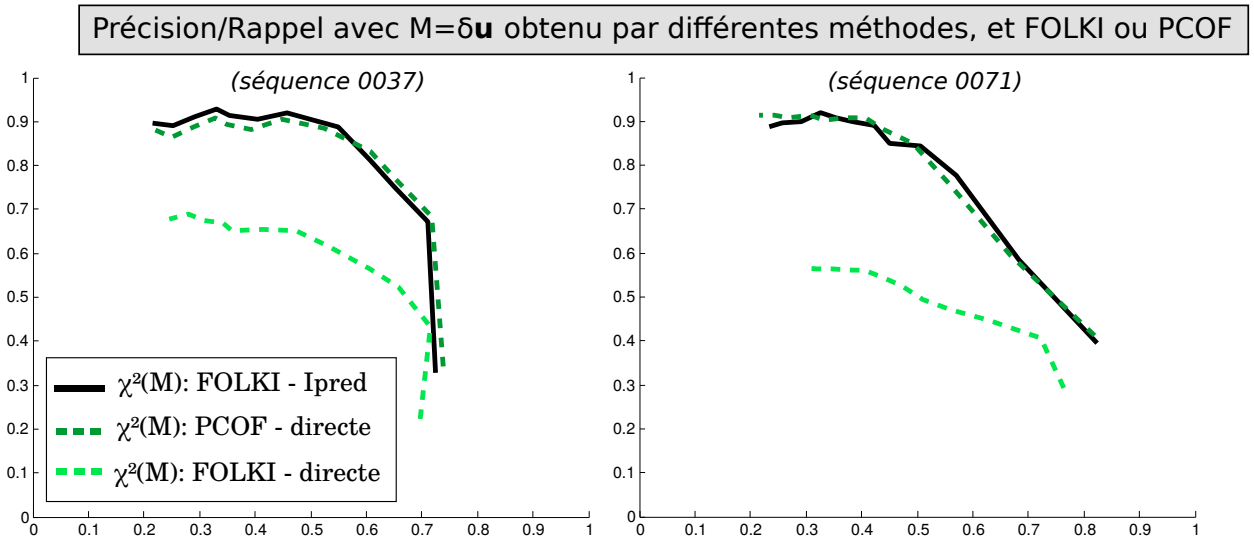

FIGURE 4.4 – Courbes de Précision-Rappel des algorithmes de détection d'objets mobiles par méthode directe et par prédiction d'image (I_{pred}) en estimant le flot optique avec FOLKI ou PCOF.

pour l'estimation du flot optique et de la disparité stéréo.

Sans pour autant considérer les algorithmes du top des classements KITTI qui sont beaucoup trop coûteux en temps de calcul, nous pouvons néamoins envisager l'utilisation d'algorithmes comme HAOF et ELAS qui sont certes 4 à 5 fois plus lents que FOLKI et ACTF, mais qui peuvent quand même s'exécuter à une cadence relativement élevée. En effet, avec les paramètres par défault d'OpenCV, HAOF tourne en 127 ms sur un GPU (GeForce GTX TITAN). Quant à ELAS, il s'exécute en 147 ms sur CPU (Intel Core i7) avec les paramètres par défaut, et en 54ms avec l'option de sous-échantillonnage (version notée $\text{ELAS}_{1/2}$).

La figure 4.5 ci-dessous, montre les courbes de Précision-Rappel obtenues avec $M = \delta\mathbf{u}$ par prédiction d'image, en utilisant : FOLKI et ACTF (configuration de base), HOAF et FOLKI, et FOLKI et $\text{ELAS}_{1/2}$.

On remarque à la figure 4.5, que l'algorithme de détection est relativement peu sensible au choix de l'algorithme stéréo. En pratique la différence entres les courbes (FOLKI+ACTF) et (FOLKI+$\text{ELAS}_{1/2}$) est en bonne partie due au fait que la carte de disparité d'$\text{ELAS}_{1/2}$ est moins dense que celle d'ACTF.

L'emploi de HAOF à la place de FOLKI entraîne une variation de performance plus importante dans la séquence 28/09-drive0037, HOAF donne de moins bonnes performances pour la détection, alors que dans la séquence 29/09-drive0071 HAOF permet un meilleur *Rappel*, à *Précision* égale. Si l'on regarde en détail les cartes $\chi^2(M)$ présentées aux figures 4.6 et 4.7, obtenues avec HAOF et FOLKI, on constate plusieurs choses. Tout d'abord, HAOF, qui est une méthode variationnelle, donne un résidu $\delta\mathbf{u}$ plus lisse et moins bruité que la méthode FOLKI, par fenêtre locale.

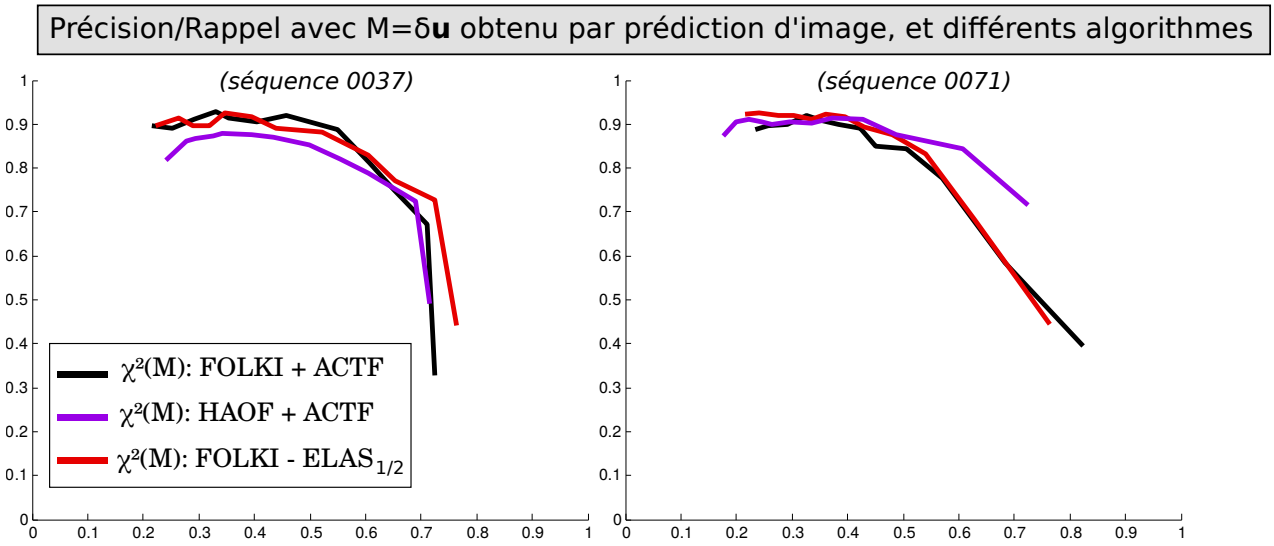

FIGURE 4.5 – Courbes de Précision-Rappel obtenues par prédiction d'image, avec FOLKI ou HAOF pour le flot optique, et ACTF ou $\text{ELAS}_{1/2}$ pour la stéréo.

Ainsi, avec HOAF, de nombreuses fausses détections sont évitées notamment aux seuils les plus bas (ce qui explique un tel écart en termes de précision sur la partie droite des courbes dans la séquence 0071, figure 4.5).

Cependant, on remarque également que HAOF ne parvient pas toujours à détecter des objets mobiles, pourtant proches et rapides comme le vélo dans la figure 4.6, que FOLKI parvient effectivement à détecter. D'après nous, cela s'explique par la convergence lente des algorithmes de flot optique variationnel comme HAOF qui nécessite parfois de nombreuses itérations —du moins plus que le nombre d'itérations par défaut dans OpenCV, qui permet de faire tourner HAOF en 127ms—. Et ce, contrairement à FOLKI qui converge en quelques itérations.

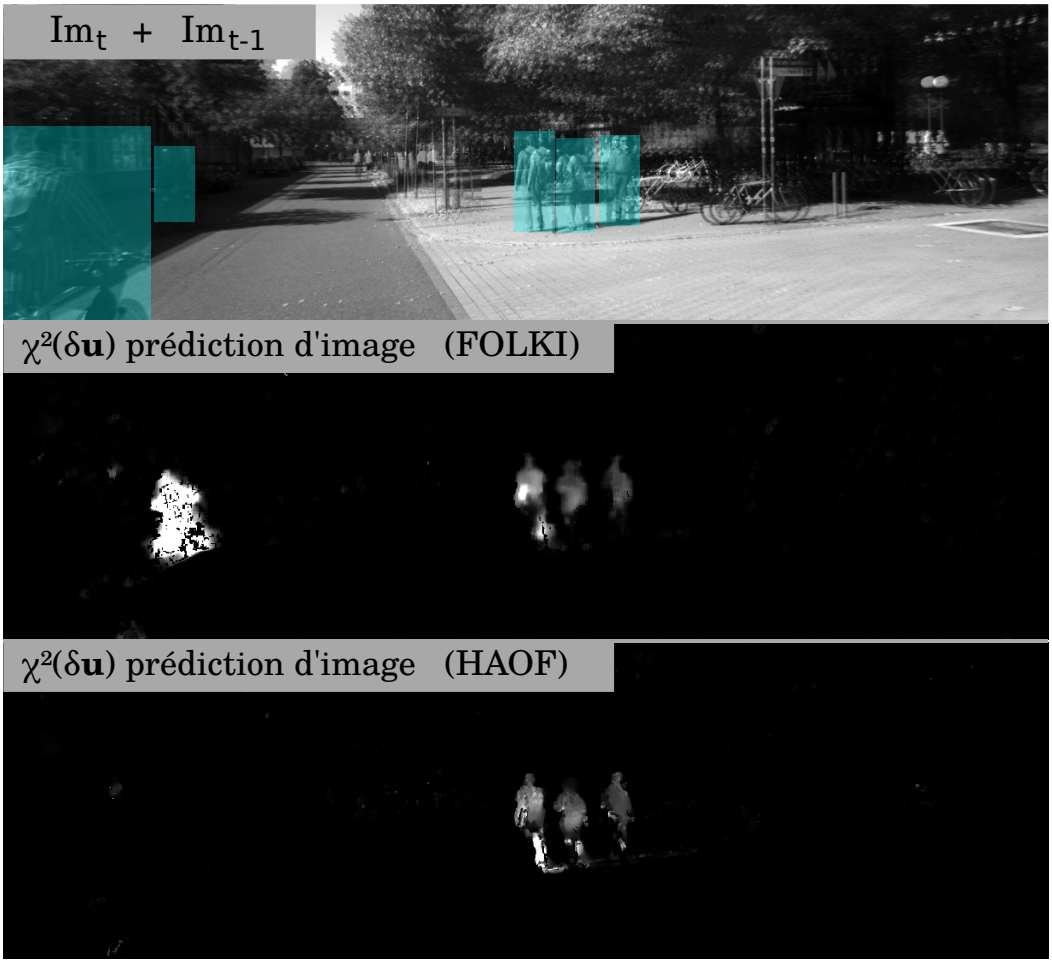

FIGURE 4.6 – Cartes $\chi^2(M)$ obtenues avec FOLKI et HAOF (séquence 0037).

4.4.4 Étude de l'influence du modèle d'incertitude sur les performances de détection

À la Section 4.1 nous avons vu comment modéliser et propager les erreurs de mesures, et notamment l'erreur d'odométrie visuelle. Les courbes Précision-Rappel dans la figure 4.8 montrent l'intérêt d'une telle modélisation de l'erreur de mesure.

On constate un gain relativement important en termes de précision pour la détection d'objets mobiles dans la séquence 37, lorsque l'on utilise le modèle complet $\Sigma_{M(\mathbf{x},d,\mathbf{u},\Theta)}$ pour l'incertitude de M (sauf pour les seuils très bas qui ne sont pas ceux que nous utilisons en pratique pour la détection). D'ailleurs l'incertitude d'odométrie visuelle est ici prépondérante, et la courbe obtenue en considérant $\Sigma_{M(\mathbf{x},d,\mathbf{u})}$ est très proche de celle avec $\Sigma_M = Id_2$, contrairement à celle avec $\Sigma_{M(\mathbf{x},d,\mathbf{u},\Theta)}$. Dans cette séquence, le véhicule effectue un virage à 90 degrés entraînant des rotations importantes sur plusieurs paires stéréo consécutives. Le flot optique prédit a donc une amplitude importante, et est d'autant plus sensible aux erreur d'odométrie visuelle.

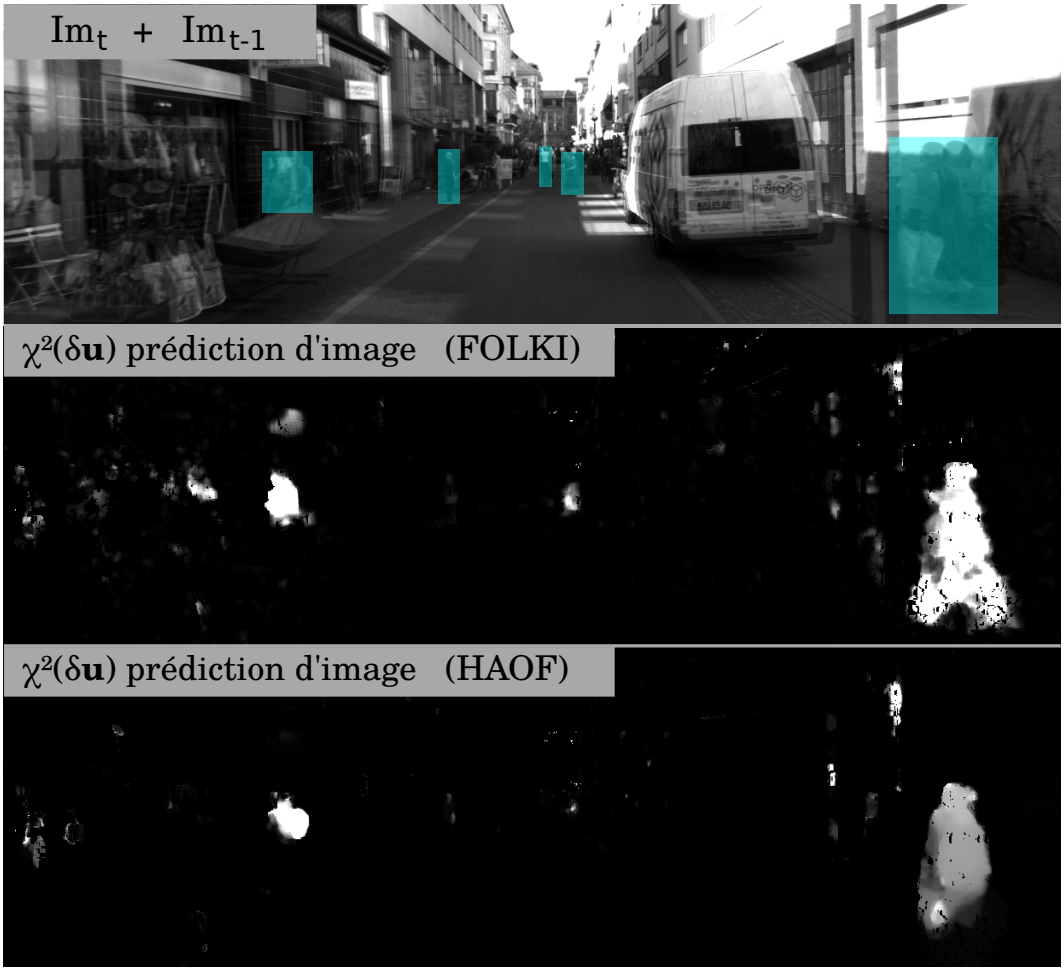

FIGURE 4.7 – Cartes $\chi^2(M)$ obtenues avec FOLKI et HAOF (séquence 0071).

Au contraire, dans la séquence 71 on ne constate pas de gain avec ce modèle complet de Σ_M. En effet, le véhicule se déplace alors en ligne droite, à vitesse relativement faible, donc les erreurs d'odométrie visuelle ont beaucoup moins d'impact qu'à la séquence 37. Par ailleurs dans cette séquence en milieu urbain, les images présentent des points remarquables bien répartis sur l'ensemble du champ visuel et en profondeur donc l'odométrie visuelle peut s'appuyer sur des points remarquables proches (utiles pour l'estimation de la translation) et distants (utiles pour l'estimation de la rotation). Dans ce cas l'odométrie est très précise et son incertitude devient négligeable. Considérer $\Sigma_{M(\mathbf{x},d,\mathbf{u},\Theta)}$ ou $\Sigma_{M(\mathbf{x},d,\mathbf{u},\Theta)}$ pour le calcul de $\chi^2(M)$ revient alors au même.

On a donc pu mettre en évidence l'intérêt d'une propagation d'erreur plus fine de l'odométrie visuelle dans le processus de détection, en fonction du type de séquences utilisées. Cette analyse, que nous n'avons pas pu poursuivre sur des exemples plus nombreux, est, à notre connaissance, originale dans le domaine de la détection d'ob-

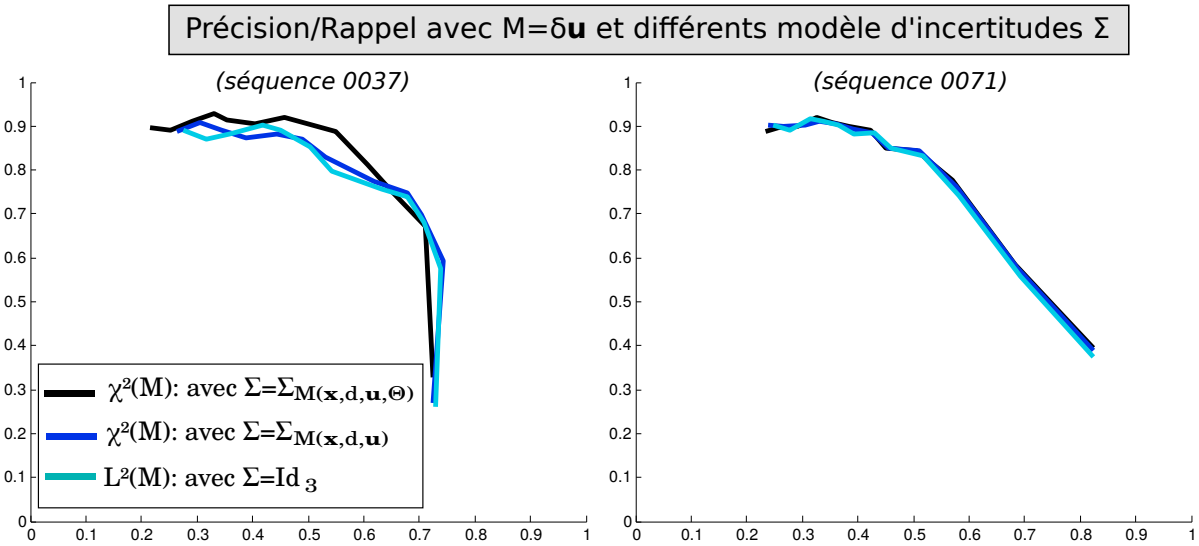

FIGURE 4.8 – Courbes de Précision-Rappel obtenues avec différents modèle d'incertitude Σ_M.

jets mobiles.

4.5 Conclusion

Dans ce chapitre, nous avons étudié comment effectuer une détection dense des objets mobiles à partir d'une carte $\chi^2(M)$ de vraisemblance de mouvement pour chaque pixel de l'image, carte obtenue en exploitant un système stéréo. Nous avons passé en revue différents résidus M possibles, que nous avons calculés en utilisant la méthode usuelle (*directe*) et proposant une nouvelle méthode par prédiction d'image que nous avions déjà présentée à la sous-section 3.3.3 pour l'estimation du flot de scène. Nous avons pris soin de formuler cette prédiction de sorte à faciliter la propagation des erreurs de mesure et permettre l'estimation de la covariance du résidu.

Une évaluation qualitative et quantitative nous a permis de choisir le résidu M et la méthode la plus adaptée pour les algorithmes temps réel que nous avons sélectionnés au Chapitres 2 et 3 (FOLKI et ACTF). Cette étude des performances en terme de détection nous a d'ailleurs conforté dans nos choix algorithmes (FOLKI et ACTF) et a souligné l'importance de la modélisation d'incertitude plus complète par le calcul au premier ordre de la matrice de covariance $\Sigma_{M(\mathbf{x},d,\mathbf{u},\Theta)}$, tenant compte notamment des erreurs d'odométrie visuelle.

Pour finir, il convient de souligner que la solution retenue pour la détection (à savoir : calculer $M = \delta\mathbf{u}$ par prédiction d'image) n'implique pas de surcoût en temps de calcul puisque la synthèse de l'image prédite I_t^{pred} et le calcul du flot résiduel $\delta\mathbf{u}$ avec FOLKI font déjà partie du pipeline proposé au Chapitre 3 pour l'estimation du

flot de scène par l'algorithme PCOF. Les seules opérations additionnelles à effectuer sont la renormalisation $\chi^2(M)$ du résidu (en tenant compte des régions d'intérêt), et la segmentation des objets mobiles à partir du seuillage de la carte $\chi^2(M)$ (cf. sous-section 4.3.2). Ces opérations se font en seulement quelques millisecondes sur GPU (GeForce GTX TITAN) et CPU.

Ainsi le système global de perception stéréo en milieu dynamique (incluant estimation du flot de scène et détection des objets en mouvement) que nous proposons s'exécute à plus de 10Hz —soit une cadence compatible avec certains capteurs embarqués— sur les séquences de la base de donnée KITTI (images 370 par 1200 pixels acquises à 10 images par seconde). Á notre connaissance il s'agit du premier système de détection dense de mouvement en stéréo qui est temps réel pour un système embarqué réaliste comme celui de la base KITTI.

Maintenant que nous avons présenté un système de perception stéréo en environnement dynamique, nous étudions au chapitre suivant son application à la navigation robotique autonome en présence d'objets mobiles. Pour cela nous expliquons comment obtenir des codes suffisamment performants en profitant d'une accélération matérielle des calculs sur carte graphique (GPU), et comment intégrer notre code dans un système robotique en utilisant ROS (Robot Operating System).

Chapitre 5

Programmation sur GPU d'Algorithmes Temps-Réel et Application à la Navigation Autonome en Robotique

Sommaire

Ce chapitre a donné lieu à la publication [4] de la liste fournie en section 6.3.

Introduction

Dans les chapitres précédents nous nous sommes efforcés d'utiliser et de concevoir des algorithmes de perception très rapides pour pouvoir envisager une exécution en ligne sur une plateforme embarquée telle que le robot Summit XL dont nous disposons au laboratoire ATEXPA (cf. Figure 5.1).

Nous employons maintenant ces algorithmes pour permettre à un robot de naviguer de façon autonome dans un environnement comportant des objets en mouvement. L'hypothèse est que le robot utilise seulement comme capteur un banc stéréo. L'objectif est d'aboutir à une preuve de concept en montrant expérimentalement dans un cas simple, que nos algorithmes sont suffisamment rapides et précis pour permettre une telle navigation robotique autonome en environnement dynamique. On souhaite notamment vérifier que le robot est suffisamment réactif pour adapter en temps-réel sa trajectoire afin d'éviter d'entrer en collision avec les autres objets en mouvement.

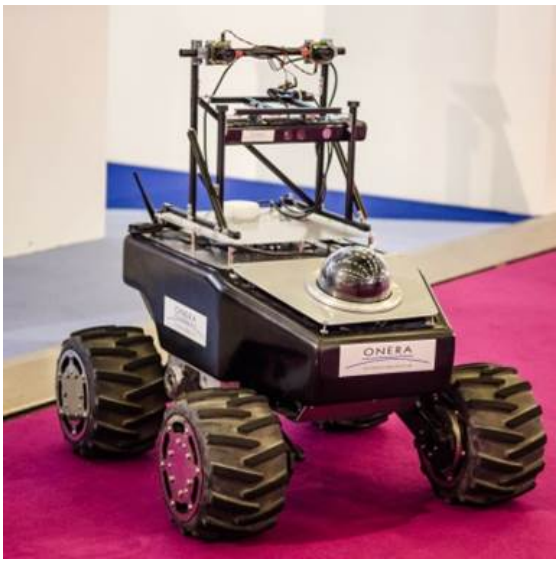

FIGURE 5.1 – Nous utilisons la plateforme Summit XL ©Robotnik, qui intègre une carte graphique Nividia GT640 et un processeur Intel quad-core i7. Le banc stéréo doté d'une baseline de 18cm, est configuré pour enregistrer des images VGA à 20Hz.

Ce chapitre est organisé de la manière suivante : en section 5.1 nous expliquons le principe de la programmation sur carte graphique (GPU) qui nous a permis d'obtenir des algorithmes temps-réel ; en section 5.2 nous détaillons l'architecture du système robotique que nous avons intégré sous ROS (Robot Operating System) [Quigley et al., 2009], ainsi que les modules de perception et de commande utilisés. La partie commande proprement dite a été développée par Hélène Roggeman, doctorante à l'ONERA/DCPS. Enfin en section 5.3 nous présentons la preuve de

concept expérimentale que nous avons effectuée, et discutons des résultats obtenus.

5.1 Programmation sur carte graphique (GPU) pour des algorithmes temps-réel

Afin d'expliquer pourquoi nous utilisons un GPU (*Graphic Processor Unit*) et comment nous l'exploitons pour accélérer la vitesse d'exécution de nos algorithmes (sous-section 5.1.3), il nous faut d'abord décrire l'architecture d'un GPU (sous-section 5.1.1) et le principe du langage CUDA dont nous nous servons pour la programmation sur GPU (sous-section 5.1.2).

5.1.1 Description de l'architecture GPU

La station de travail sur laquelle nos algorithmes tournent, est équipée d'un processeur Intel Core i7, et d'une carte graphique GeForce GTX TITAN dont les spécifications sont données au tableau 5.1, à titre de comparaison.

TABLE 5.1 – Spécifications du CPU Intel Core i7 et du GPU GeForce GTX TITAN.

	Intel Core i7-3930k	GeForce GTX TITAN
Nombre de coeurs	6	2688
Fréquence	3.20GHz	837 MHz
Nombre d'opérations flottantes par secondes (Flops)	$\approx$ 100 GFlops	$\approx$ 10 TFlops

On constate que les GPUs, qui sont dotés de plusieurs centaines voire milliers de coeurs, permettent un nombre d'opérations par seconde (FLOPS : "FLoating-point Operations Per Second") beaucoup plus élevé que les CPUs : dans notre cas, on gagne un facteur 100 en puissance brute de calcul (cf. tableau 5.1).

Cependant seuls des algorithmes massivement parallélisables qui permettent d'occuper simultanément l'ensemble des coeurs, profitent d'une accélération matérielle sur GPU. De plus, l'architecture des GPUs, illustrée Figure 5.3, favorise les algorithmes dont la parallélisation repose sur des processus qui s'exécutent de façon indépendante, ou bien sur des processus qui communiquent uniquement de façon locale entre eux. En effet, l'accès à la mémoire globale est très coûteux et nécessite plusieurs centaines de cycles GPU, alors que l'accès à la mémoire partagée au sein d'un multiprocesseur est très rapide.

Afin d'optimiser le temps d'exécution d'un code sur carte graphique, il faut donc non seulement paralléliser le plus possible les opérations à effectuer, mais aussi limiter au maximum les échanges de données entre processus. La gestion de la mémoire est un point clé pour l'exploitation optimale des GPUs qui sont conçus pour accéder efficacement à des blocs de mémoire contigus.

Enfin, les GPUs (qui sont initialement conçus pour des applications de rendu graphique) disposent d'un autre type de mémoire cache, en lecture seule, dédiée à

l'accès et l'interpolation rapide d'images de textures. Ces fonctions d'interpolation sont gravées dans le silicium et permettent l'accès quasi-immédiat au résultat de l'interpolation d'une image en un point.

5.1.2 Quelques éléments sur la programmation GPU en CUDA

Le langage CUDA, pour la programmation sur cartes graphiques, est conçu pour exploiter leurs spécificités architecturelles évoquées précédemment. L'architecture logique sur laquelle se base CUDA est la suivante : un *kernel* est une fonction codant les opérations qui s'exécutent en parallèle dans de nombreux processus, ces processus sont regroupés dans des blocs, eux-même répartis dans une grille (cf. illustration Figure 5.2).

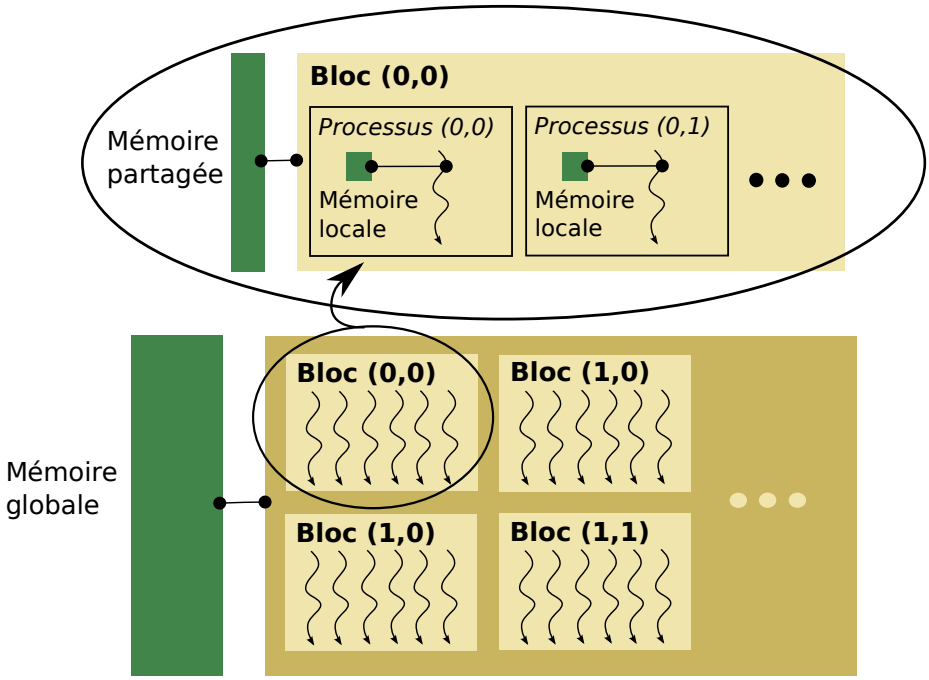

FIGURE 5.2 – Organisation de l'exécution des processus en grille de blocs, et structure des différentes mémoire du GPU.

Les blocs sont lancés de manière asynchrone, et disposent d'une mémoire partagée entre les processus d'un même bloc. Les processus d'un bloc peuvent être synchronisés en utilisant la fonction : `__syncthreads()`. Les dimensions (1D, 2D ou 3D) de la grille et des blocs, sont définis avec les signes <<< .,. >>> au lancement du *kernel* dans le programme C/C++ :

```
monKernel<<<Nblocs,Nprocessus>>>(...).
```

Cette architecture logique, illustrée Figure 5.2, correspond plus ou moins à l'architecture matérielle illustrée Figure 5.3 puisque chaque bloc est assigné à un multiprocesseur (mais d'autres blocs peuvent également tourner sur ce même multipro-

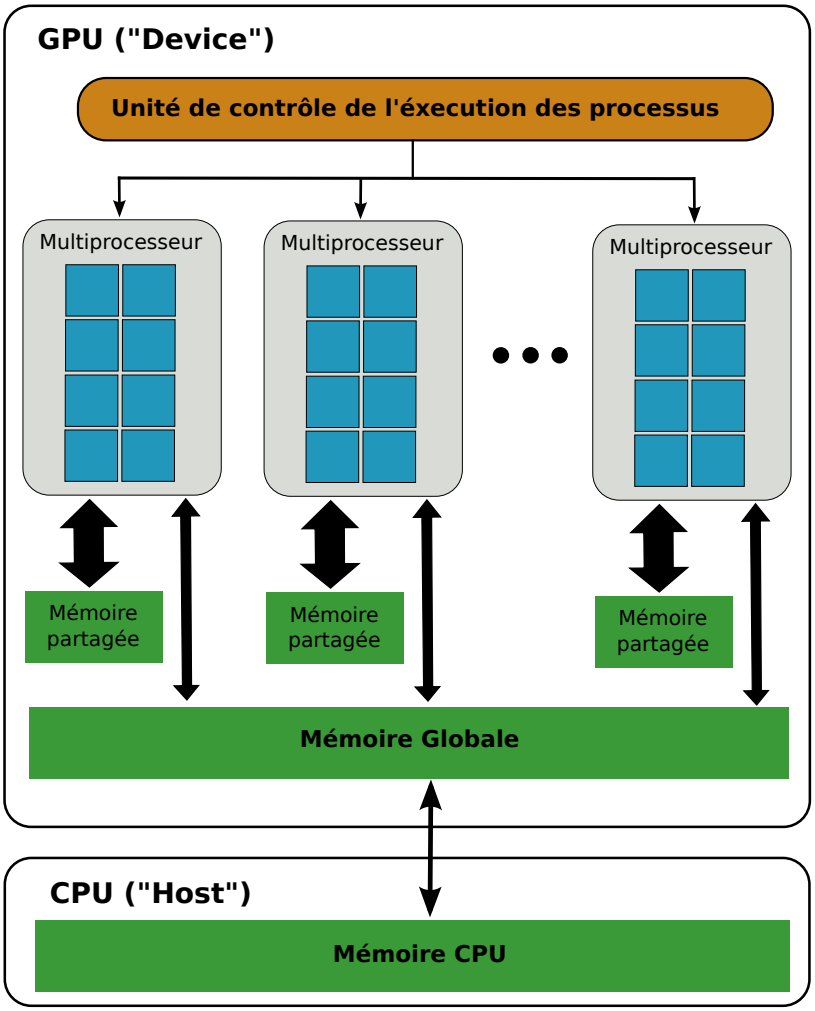

FIGURE 5.3 – Architecture matérielle d'un GPU : chaque multi-processeur dispose de plusieurs coeurs et d'une mémoire partagée à accès rapide, et communique avec la mémoire globale (d'accès plus lent) qui sert d'interface entre le GPU et le CPU.

cesseur par *multithreading*).

La mémoire cache dédiée aux textures, quant à elle, peut être utilisée au sein d'un kernel en utilisant par exemple l'instruction :

```
val = tex2D(texImg,x,y),
```

qui retourne la valeur de l'image de texture 'texImg', interpolée en la position ('x','y'). Pour cela, il faut préalablement définir et instancier la variable de texture, par exemple avec :

```
static texture<float, 2> texImg,
```

et à l'aide des fonctions :

`cudaBindTexture2D(...)` ou `cudaBindTextureToArray(...)`.

5.1.3 Utilisation du GPU pour accélérer nos algorithmes

La plupart des algorithmes que nous utilisons pour la perception stéréo en environnement dynamique, estiment des valeurs en chaque pixel de l'image : FOLKI pour le flot optique, ACTF et CoR pour l'estimation et le post-traitement des cartes de disparité stéréo, et le calcul de la carte de probabilité de mouvement $\chi^2(M)$.

Puisque les images que nous traitons comportent des centaines de miliers de pixels, nos algorithmes denses sont capables a priori d'occuper la totalité des centaines/milliers de coeurs présents dans un GPU.

Comme nous l'avons vu à la sous-section 5.1.1, occuper l'ensemble des coeurs du GPU ne suffit pas à garantir un temps d'exécution minimal : il faut également exploiter judicieusement la mémoire. Nous expliquons ici en quoi nos algorithmes satisfont ces critères et tirent profit de l'architecture des GPU.

Communications locales pour l'exploitation de la mémoire partagée

Hormis le calcul de la carte $\chi^2(M)$ qui implique des opérations totalement indépendantes en chaque pixel de l'image, les algorithmes que nous utilisons font appel à des régularisations spatiales nécessitant une certaine communication entre processus :

- L'algorithme FOLKI (cf. 3.2.5) repose sur la résolution, à chaque niveau n de la pyramide d'image, et à chaque itération $k+1$, de l'équation suivante :

$$\mathbf{u}(\mathbf{x})_n^{k+1} = \left(\sum_{\mathbf{x}' \sim \mathbf{x}} \nabla I_0 \nabla I_0^T(\mathbf{x}') \right)^{-1} \left(\sum_{\mathbf{x}' \sim \mathbf{x}} \nabla I_0 I_t(\mathbf{x}') \right) \tag{5.1}$$

  Dans la référence [Plyer et al., 2014], les auteurs montrent que cette opération réquiert des convolutions avec des noyaux correspondant aux tailles des fenêtres de corrélation locale, soit dans notre cas $(2r+1)$x1 (on utilise des noyaux séparables), avec $r \in \{8; 4\}$.

- L'algorithme ACTF nécessite d'aggréger localement des données en calculant des sommes sur des voisinages de tailles 5x5 autour de chaque pixel, afin de calculer et minimiser une mesure de similarité (NCC) de la forme :

$$\mu(\mathbf{x}, d) = \sum_{\mathbf{x}' \sim \mathbf{x}} \rho \left(I_G(\mathbf{x}'), I_D(\mathbf{x}' + d) \right) \tag{5.2}$$

- Le filtrage de la carte de disparité d par Consensus de Régions (cf. 2.2.3) effectue des sommes sur des voisinages p —qui sont des fenêtres carrées d'arêtes

$N \in \{16; 32\}$— pour calculer :

$$\theta_p = \left(\sum_{\mathbf{x} \in p} U(\mathbf{x})^T U(\mathbf{x}) \right)^{-1} \left(\sum_{\mathbf{x} \in p} U(\mathbf{x})^T d_0(\mathbf{x}) \right) \tag{5.3}$$

L'implémentation sur GPU des algorithmes FOLKI, ACTF et CoR ne fait donc pas intervenir uniquement des processus indépendants pour chaque pixel $\mathbf{x}$, puisque les résultats calculés sur les pixels $\mathbf{x}'$ voisins sont à prendre en compte (cf. Eq. (5.1), (5.2) et (5.3)). Néanmoins les sommes qui interviennent sont calculées sur un voisinage du pixel courant $\mathbf{x}$ et impliquent des interactions locales entre processus voisins : elles peuvent donc être calculées efficacement sur GPU en exploitant la mémoire partagée entre processus d'un même bloc (cf. sous-section 5.1.2).

Transferts limités avec la mémoire globale du GPU

Les algorithmes FOLKI et ACTF font tous les deux appel à des pyramides d'images pour estimer respectivement le flot optique et la disparité : les valeurs estimées sur les images de plus basse résolution (en haut de la pyramide) sont propagées aux niveaux inférieurs pour servir d'initialisation. Cette propagation implique seulement un sur-échantillonnage et une multiplication de chaque valeur, par un facteur 2 (cf. Figure 2.9), et peut être menée efficacement sur GPU.

En dehors de ces propagations induites par l'approche pyramidale, et des sommations locales évoquées précédemment, les transferts de données restants ont lieu entre le CPU et le GPU via la mémoire globale du GPU. Nous avons vu que l'accès à cette mémoire globale est très coûteux en temps. Cependant, dans le cas de nos algorithmes (FOLKI, ACTF et CoR), les transferts entre le CPU et le GPU ont lieu uniquement en début et en fin d'algorithme pour transmettre les données d'entrées (les images gauche consécutives, la paire d'images stéréo, et la carte de dispartié) et pour récupérer les sorties (les champs u et v du flot optique, la carte de disparité d, et la carte de disparité filtrée d^*).

Interpolation sur GPU

Enfin, la dernière raison pour laquelle des algorithmes tel que FOLKI profitent d'une accélération sur carte graphique, est l'utilisation de fonction d'interpolation d'image : dans l'algorithme FOLKI, à chaque niveau n, rayon r et itération k, on doit effectuer en chaque pixel $\mathbf{x}$ l'interpolation suivante :

$$I_1^w(\mathbf{x}) = I_1\Big(\mathbf{x} + \mathbf{u}_n^k(\mathbf{x})\Big). \tag{5.4}$$

On obtient donc un gain de temps significatif en effectuant toutes ces interpolations sur GPU (qui est spécialement conçu pour cela).

Il est à noter que l'interpolation codée en dur dans la mémoire graphique d'un GPU est une interpolation assez frustre, une interpolation linéaire quantifiée sur

256 niveaux. Ce type d'interpolation est bien souvent suffisant pour le calcul du flot optique. Pour des situations où une meilleure précision est utile, il est nécessaire de coder des interpolations d'ordre supérieur, par exemple en utilisant des fonctions B-spline cubique. On peut cependant trouver des façons très rapides de réaliser ces interpolations sur GPU comme expliqué par Champagnat et Le Sant [Champagnat and Le Sant, 2012]

Temps de calcul

Comme on l'a vu, l'utilisation du GPU semble particulièrement judicieuse pour l'implémentation de nos algorithmes. L'implémentation de FOLKI a été réalisée par Aurélien Plyer en 2009 et 2012. Pour ma part, j'ai effectué le codage des algorithmes ACTF, CoR et des opérations nécessaires au calcul de la carte $\chi^2(M)$. Ces codes ont ensuite été évalués en termes de coût en temps de calcul sur un GPU GeForce GTX TITAN. Les résultats en temps moyens sont présentés dans le Tableau 5.2. A titre de comparaison, la version Matlab de FOLKI s'exécute en 2.5s sur CPU.

TABLE 5.2 – Temps d'exécution sur les images KITTI avec un GPU GeForce GTX TITAN.

	ACTF	CoR	FOLKI	$\chi^2(M)$
Temps de calcul	11ms	8ms	27ms	0.6ms

L'utilisation de ces algorithmes massivement parallélisables et particulièrement adaptés à l'architecture des GPUs, nous a permis d'atteindre une vitesse d'exécution sans précédent pour notre système de perception stéréo dense. Les algorithmes d'estimation de flot de scène et de détection d'objets mobiles présentés aux Chapitres 3 et 4 sont à notre connaissance les plus rapides qui existent dans leurs domaines, et sont compatibles avec une exécution à cadence vidéo sur les séquences KITTI (10Hz) avec un GPU GeForce GTX TITAN.

5.2 Un système de navigation robotique autonome intégrant nos algorithmes de perception

Grâce à l'optimisation des calculs sur GPU (cf. section 5.1), nous disposons d'algorithmes rapides potentiellement capables de tourner en ligne sur un robot de taille réduite comme Summit XL (cf. Figure 5.1).

Dans cette section nous décrivons le système robotique intégrant nos codes dans le framework *Robot Operating System* (sous-section 5.2.1). À la sous-section 5.2.5, nous présentons le module de commande prédictive développé par Hèlene Roggeman, doctorante au DCPS [1], qui permet au robot d'adapter à chaque instant sa trajectoire en fonction des objets, fixes et mobiles, qui l'environnent. Ce module a besoin d'une représentation spatiale des objets mobiles détectés (sous-section 5.2.2) ainsi que

1. Département de Conception et évaluation des Performances de Systèmes, ONERA

d'une modélisation de leur dynamique (sous-section 5.2.3). En amont du module de commande prédictive, il est également nécessaire de gérer les pistes de détection en effectuant l'association temporelle des objets mobiles détectés, et en supprimant d'éventuelles fausses alarmes (sous-section 5.2.4).

5.2.1 Intégration dans un système robotique en utilisant ROS

Pour fonctionner, un robot doit accomplir plusieurs tâches en parallèle, par exemple : percevoir l'environnement, planifier sa trajectoire, calculer les entrées adéquates à envoyer au module de contrôle des moteurs, etc. Ces fonctions peuvent s'opérer simultanément et à différentes cadences : typiquement les fonctions de commande du robot s'effectuent à une cadence plus élevée que les fonctions de perception et de modélisation de l'environnement.

Robot Operating System (ROS) [Quigley et al., 2009], est une plateforme de développement C++/Python de systèmes robotiques. ROS se comporte comme un système d'exploitation et permet de faire fonctionner sur un robot, plusieurs modules qui peuvent communiquer entre eux et accéder aux périphériques (capteurs, moteurs,...). ROS repose sur un modèle de fonctionnement fondé sur des noeuds qui accomplissent chacun une tâche spécifique, et qui communiquent entre eux par des messages, via un mécanisme d'écriture et d'abonnement à des *topics* ou via l'utilisation de *services*.

Principe des *topics*. Après avoir effectué ses opérations, un noeud peut publier ses résultats dans un *topic* prédéfini. Tous les noeuds ayant souscrit un abonnement à ce *topic* sont alors notifiés du nouveau résultat et reçoivent ce dernier dans un message daté de l'instant d'écriture.

Principe des *services*. Certains noeuds peuvent proposer des services qui permettent d'effectuer une fonction à la demande, lorsqu'un autre noeud en a besoin. Le noeud qui envoie une requête de service reçoit ensuite un message incluant les données voulues, datées de leur instant de production, une fois que celles-ci ont été calculées par le noeud fournisseur du service.

Pour l'expérience de navigation autonome en présence d'objets mobiles décrite à la section 5.3, le système robotique utilisé sur la plateforme Robotnik est illustré Figure 5.4. Il comprend un noeud ROS pour chacune des quatres fonctions suivantes :

- l'odométrie visuelle effectuée avec l'algorithme eVO
- la perception et la modélisation d'environnement dynamique
- la commande
- le contrôle des moteurs

Une description plus détaillée de ce système est donnée à la Figure 5.5 qui présente distinctement chacune des fonctions effectuées au sein des noeuds, en les regroupant en 4 parties :

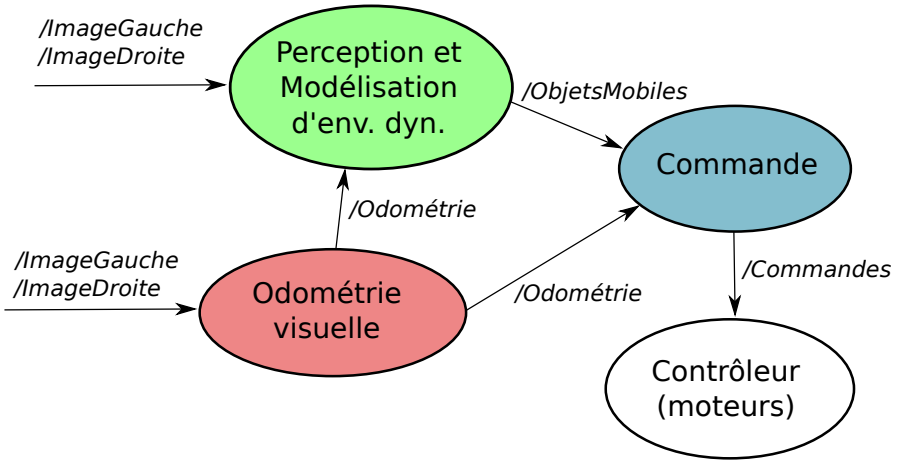

FIGURE 5.4 – Schéma du système utilisé avec les noeuds (en ovale) et les échanges messages via des topics (en */Italic*).

- la partie *Localisation*
- la partie *Perception 3D dynamique*
- la partie *Modélisation d'environnement dynamique*
- la partie *Contrôle*

Le noeud de perception et de modélisation d'environnement dynamique comprend l'algorithme d'estimation du flot de scène décrit au Chapitre 3, qui utilise ACTF et PCOF, et l'algorithme de détection d'objets en mouvement décrit au chapitre 4. Ce noeud comprend également une fonction de modélisation des objets mobiles, et de gestion temporelle des détections (ces fonctions sont présentées aux sous-sections 5.2.2-5.2.4), pour fournir au module de commande prédictive toutes les informations requises pour la planification de la trajectoire.

Le noeud de commande, quant à lui, se divise en deux : la fonction de calcul des cartes d'obstacles, et la fonction de commande prédictive à proprement parler. Ces fonctions sont toutes deux décrites à la sous-section 5.2.5.

5.2.2 Représentation spatiale des objets mobiles

Pour pouvoir planifier une trajectoire sans collision en présence d'obstacles mobiles, il est nécessaire de disposer d'une représentation spatiale de ces derniers. Une méthode classique pour modéliser l'environnement en 3D à partir d'un capteur stéréo ou lidar, est de construire une grille de *Voxels* (i.e. des cubes de dimensions prédéfinies), en utilisant par exemple le logiciel OctoMap [Hornung et al., 2013]. Le principe est le suivant : chaque point 3D observé se trouve sur un rayon partant du centre optique d'une caméra. Chaque rayon s'arrête au premier obstacle rencontré. Le voxel correspondant à cet obstacle est marqué comme occupé. Tous les voxels intermédiaires, situés entre la caméra et l'obstacle, sont marqués comme libres puisque la lumière les a traversés, et les voxels restant sont marqués comme inconnus.

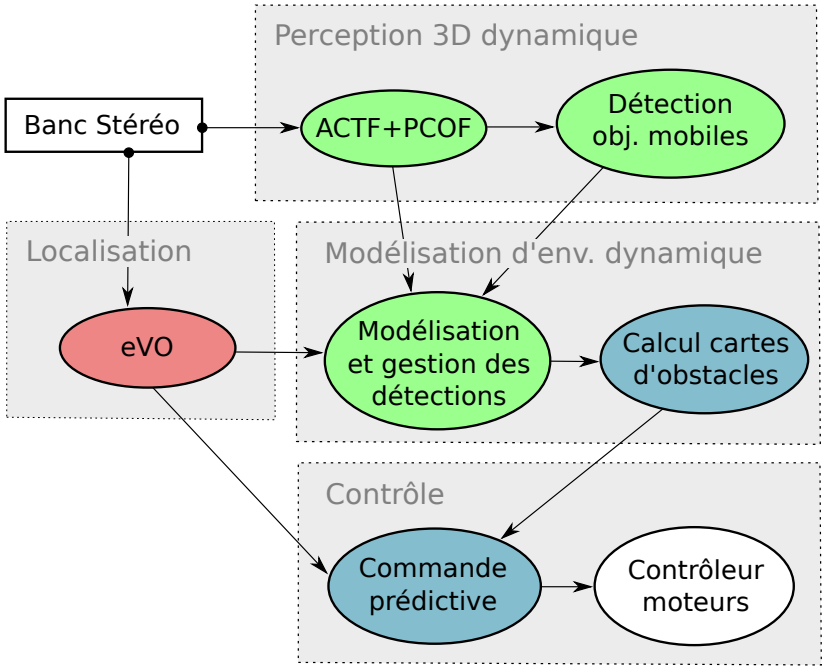

FIGURE 5.5 – Représentation détaillée du système, décomposant les fonctions dans chaque noeud (indiqué par une couleur) du schéma de la Figure 5.4 .

Ce type de représentation 3D est par exemple utilisée dans le système 3DSCAN [Sanfourche et al., 2014] développé au DTIM, qui permet à un robot naviguant en environnement statique, de se localiser et de reconstruire en temps-réel une carte de son environnement. Une telle représentation est également employée par Bajracharya et al [Bajracharya et al., 2012] qui ont conçu un système de perception d'environnement complexe capable de s'exécuter en temps-réel sur une plateforme embarquée. Leur système accumule temporellement dans chaque voxel des informations radiométriques et géométriques (la normale à la surface observée) utilisées pour segmenter les obstacles dans la grille de voxel. Leur algorithme permet d'identifier les régions libres au sol, pour indiquer où marcher à leur robot quadripède. Ils appliquent également leur algorithme en milieu urbain en segmentant avec succès tout type d'obstacle, y compris les piétons. Cependant l'utilisation de voxels n'est pas adaptée en présence d'objets mobiles, car ces derniers créent des artefacts en laissant derrière eux des voxels marqués à tort comme occupés.

En plus de ces voxels "fantômes" induits par les objets en mouvement, la représentation en voxels met en évidence un autre problème —cette fois inhérent au capteur—, à savoir : seule la surface apparente des objets est visible par le capteur, et peut être mesurée. On ne peut pas estimer avec un banc stéréo la profondeur d'un objet et son volume à un instant donné sans utiliser d'a priori géométrique sur la forme de l'objet. Pour pallier ce problème, on peut utiliser des templates d'objets (e.g. voiture, piétons, cyclistes...) comme le font Menze et Geiger [Menze and Geiger, 2015]

qui mettent en correspondance des modèles CAO de voitures avec les points 3D mesurés. Dans leur cas, le but est d'affiner l'estimation du flot de scène pour obtenir la meilleure précision possible, et leur méthode d'inférence est coûteuse en temps de calcul (leur algorithme s'exécute en 50min sur deux paires stéréo consécutives). Dans notre contexte de navigation autonome, une représentation spatiale plus grossière suffit et le temps de calcul est une ressource limitée. Une telle approche serait envisageable avec des modèles 3D a priori et des méthodes d'inférences plus simples.

Pour notre preuve de concept nous avons choisi une solution beaucoup plus simple en modélisant les objets mobiles par des cylindres. La hauteur et le diamètre de chaque objet sont déduits à partir de la surface apparente de l'objet mobile qui est segmenté avec l'algorithme présenté au Chapitre 4. Cette modélisation présente deux avantages : d'une part elle est simple et rapide, d'autre part elle est cohérente avec la démarche purement géométrique adoptée jusqu'à présent puisqu'elle n'est pas spécifique à des classes d'objets prédéfinies et ne nécessite pas de connaissance préalable des objets mobiles à modéliser (hormis l'hypothèse de forme cylindrique qui reste très générale).

5.2.3 Modélisation de la dynamique des objets mobiles

Dans un scénario sans objet mobile, seules la localisation du robot et la cartographie statique de l'environnement (e.g. avec des voxels) sont nécessaires pour la navigation autonome. En présence d'objets en mouvement, connaître les informations statiques (position et de forme des objets mobiles) n'est plus suffisant. Pour être capable de prédire leur trajectoire il faut au moins disposer de leur vitesse.

Dans notre cas les algorithmes présentés Chapitre 3 fournissent une estimation de la position et de la vitesse pour chaque pixel de l'image. D'autre part on sait segmenter les objets mobiles dans l'image (cf. Chap. 4). On peut donc estimer la position et la vitesse d'un objet en mouvement en calculant une moyenne sur l'ensemble des pixels qui le constituent. Toutefois la segmentation qui s'appuie sur l'estimation du flot optique est parfois erronée et peut attribuer à un objet mobile des pixels appartenant en fait au fond de la scène. Ces pixels mal segmentés ont une position et une vitesse estimées qui ne sont alors pas cohérentes avec l'objet mobile qui leur est attribué. Pour être robuste à ce genre d'erreurs, nous estimons la position d'un objet mobile à partir de la valeur médiane de la disparité (d_{med}) et de chacune des coordonnées 2D (x_{med}, y_{med}), sur l'ensemble de ses pixels :

$$X_{obj} = -\frac{b}{d_{med}} \begin{pmatrix} x_{med} - x_0 \\ y_{med} - y_0 \\ f \end{pmatrix} \tag{5.5}$$

La vitesse de l'objet mobile (V_{obj}) quant à elle, est obtenue en prenant la valeur médiane de chaque composante x,y,z, de la vitesse propre de chacun de ses pixels. La vitesse de chaque pixel est estimée à partir du flot de scène et de l'odométrie

(cf. Chap. 3), sachant l'écart de temps Δt entre les deux paires d'images stéréo considérées.

$$V_{obj} = \begin{pmatrix} \text{median}\{V_x^i\}_i \\ \text{median}\{V_y^i\}_i \\ \text{median}\{V_z^i\}_i \end{pmatrix} \tag{5.6}$$

avec :

$$\begin{pmatrix} V_x^i \\ V_y^i \\ V_z^i \end{pmatrix} = \frac{X_{t+1}^i - \left(RX_t^i + T\right)}{\Delta t} \tag{5.7}$$

En utilisant les modèles d'erreur et de propagation d'incertitudes vus à la Section 4.1, on dispose des matrices de covariances d'erreur : $\{\Sigma_{x,y,d}^{(i)}; \Sigma_V^{(i)}\}_{i \in obj}$. Malheureusement puique l'on utilise les valeurs médianes de $\{x^{(i)}, y^{(i)}, d^{(i)}, V_x^{(i)}, V_y^{(i)}, V_z^{(i)}\}$, il devient difficile d'exprimer $\Sigma_{X_{obj}}$ et $\Sigma_{V_{obj}}$ directement en fonction de ces matrices de covariance. Nous avons donc choisi comme heuristique de modéliser l'erreur sur X_{obj} en lui attribuant la même covariance que celle de la triangulation du pixel (x_{med}, y_{med}) de disparité d_{med}, qui s'écrit en appliquant l'équation 4.30 :

$$\Sigma_{X_{obj}} \approx J_X(x_{med}, y_{med}, d_{med}) \begin{pmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_d^2 \end{pmatrix} J_X^T(x_{med}, y_{med}, d_{med}). \tag{5.8}$$

Le calcul de $V^{(i)}$ nécessite deux triangulations : pour calculer $X_t^{(i)}$ et $X_{t+1}^{(i)}$. En conséquence, on adopte l'heuristique suivante pour $\Sigma_{V_{obj}}$ (en négligeant l'incertitude de R et T) :

$$\Sigma_{V_{obj}} \approx 2\Sigma_{X_{obj}}. \tag{5.9}$$

5.2.4 Gestion temporelle des détections

L'algorithme de détection (Chap. 4) retourne une liste d'objets détectés comme étant en mouvement à un instant t. Pour pouvoir chaîner ses détections dans le temps et obtenir des séries temporelles de détections (des "pistes") pour chaque objet en mouvement, il faut utiliser un algorithme de suivi de piste. L'association des pistes aux détections faites à l'instant courant, peut être effectué de plusieurs façons : par des méthodes de type *plus proches voisins* (e.g. *Global Nearest Neighbours* (GNN)), par une association probabiliste avec des *Probabilistic Data Association Filters* (PDAF), ou encore par du pistage à hypothèses multiples (*Multiple Hypothesis Tracking*) [Bar-Shalom et al., 2011].

Pour notre preuve de concept nous avons choisi une méthode des plus proches voisins, en privilégiant la simplicité et la vitesse d'exécution. Dans notre cas, chaque piste comprend la position et la vitesse d'un objet en mouvement, avec leurs incertitudes représentées par leurs matrices de covariance (cf. sous section 5.2.3). On peut donc définir la distance suivante entre une piste existant déjà à $t-1$, et un objet en

mouvement détecté à l'instant courant t :

$$d(\text{obj}, \text{piste}, t) = \sqrt{\chi^2\Big(X^{t-1}_{piste} + \Delta t V^{t-1}_{piste} - X^{t}_{obj}\Big)}, \tag{5.10}$$

où :

$$\chi^2(M) = M^T \Sigma_M M \tag{5.11}$$

avec ici :

$$\Sigma_M = \Sigma_{X^{t-1}_{piste}} + \Delta t^2 \Sigma_{V^{t-1}_{piste}} + \Sigma_{X^{t}_{obj}} \tag{5.12}$$

L'approche *Globale Nearest Neighbours* (GNN) consiste à trouver les appariements "$i \sim j$" entre une observation i et une piste j, minimisant :

$$\min \sum_{i=1}^{N} \sum_{j=1}^{M} d(i,j)\delta(i \sim j), \tag{5.13}$$

avec $\delta(i \sim j) = 1$, si i et j sont appariés et 0 sinon. Les objets qui n'ont pas été appariés, ou bien pour lesquels $d(i,j)\delta(i \sim j)$ dépasse un certain seuil, donnent lieu à la création de nouvelles pistes. Pour notre expérience, nous nous sommes contentés d'une méthode locale en associant une piste à chaque observation (i.e. objet mobile détecté) si :

$$\Big(d(\text{obj}, \text{piste}, t) < d_{\max}\Big) \text{ et } \Big(d(\text{obj}, \text{piste}, "t-1") < d_{\max}\Big), \tag{5.14}$$

avec $d(\text{obj}, \text{piste}, t)$ défini à l'équation 5.10 et $d(\text{obj}, \text{piste}, "t-1")$ défini par :

$$d(\text{obj}, \text{piste}, "t-1") = \sqrt{\chi^2\Big(X^{t}_{obj} - \Delta t V^{t}_{obj} - X^{t-1}_{piste}\Big)}. \tag{5.15}$$

Si aucune piste ne correspond à une observation, une nouvelle piste est créée.

Pour supprimer les fausses détections, relativement fréquentes mais ponctuelles dans notre cas, nous envoyons au module de commande prédictive uniquement les objets ayant déjà été détectés à l'instant précédent et pour lesquels le critère de cohérence temporelle a été validé (Eq. 5.14). Pour chacun de ces objets en mouvement détectés et validés, nous transmettons à l'algorithme de commande prédictive : son association temporelle (i.e. son numéro de piste), ses attributs de position et de vitesse avec leur matrice de covariance d'erreur, ainsi que le rayon et la hauteur du cylindre qui le représente spatialement (cf. sous-section 5.2.2).

5.2.5 Description du module de commande prédictive utilisé

Principe de la commande prédictive

L'algorithme développé par Hélène Roggeman de l'ONERA/DCPS relève de la commande prédictive. L'objectif de la commande prédictive est de trouver sur un horizon de temps H_{pred} la séquence optimale de commandes $\{U^*_k\}_{0 \leq k \leq H_{pred}}$ à fournir au contrôleur bas-niveau qui gère les moteurs. Pour cela, on simule des

trajectoires $\{\xi_k\}_k = \{x_k, y_k, \theta_k\}_k$ (position et angle de lacet) à partir de séquences de commande $\{U_k\}_k = \{v_k, \omega_k\}_k$ (vitesse linéaire et vitesse angulaire) en utilisant le modèle dynamique suivant :

$$\begin{cases} x_k = x_{k-1} + t_e v_{k-1} \cos(\theta_{k-1}) \\ y_k = y_{k-1} + t_e v_{k-1} \sin(\theta_{k-1}) \\ \theta_k = \theta_{k-1} + t_e \omega_{k-1} \end{cases} \tag{5.16}$$

Notons que les notations x et y désignent désormais les coordonnées 2D dans le référentiel du robot défini ci-après, et non plus les coordonnées en pixels dans l'image. Nous prenons comme convention le repère local centré sur le robot, avec l'axe x vers l'avant du robot et l'axe y vers sa gauche, et avec θ l'angle de lacet (cf. illustration Figure 5.6).

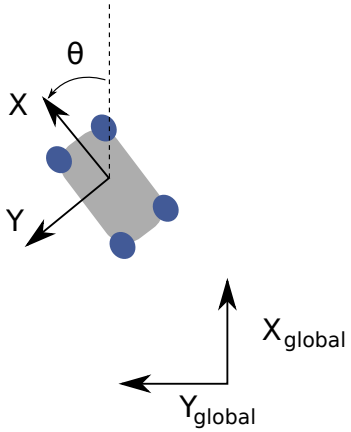

FIGURE 5.6 – Représentation du repère local centré sur le robot avec les axes x et y correspondant à l'avant et la gauche du robot, et l'angle de lacet θ.

Prédiction de la position des obstacles mobiles

Le module de perception transmet, pour chaque objet détecté : sa position, sa vitesse (X_{obj}, V_{obj}) et la matrice de covariance d'erreur de ces paramètres. Étant donné que le robot dispose d'une puissance de calcul réduite et n'embarque pas un GPU aussi puissant que la carte GeForce GTX TITAN utilisée dans les précédents chapitres, il faut compenser le retard dû à l'exécution plus lente des algorithmes de perception. Connaissant l'instant t_d de la dernière paire d'images stéréo ayant servi pour la détection, on peut calculer le retard Δt entre l'instant courant t et t_d, et simplement ajouter $\Delta X = \Delta t V_{obj}$ à la position de l'objet détecté.

Ce même type de compensation est effectué pour prédire les positions des objets mobiles, sur un horizon de temps H_{pred}. La connaissance de la vitesse instantanée V_{obj} est donc essentielle. Cependant la mesure de V_{obj} peut être assez bruitée en pratique. Un filtre de Kalman est donc appliqué, en tenant compte des incertitude de mesures $\{\Sigma_{X_{\text{obj}}}, \Sigma_{V_{\text{obj}}}\}_{\text{obj}}$. On utilise un modèle de vitesse constante pour l'équation

d'état du filtre qui fournit les valeurs : $(\hat{X}_{\text{obj}}, \hat{V}_{\text{obj}})$. Après compensation du retard induit par les algorithmes de perception, on obtient :

$$\begin{pmatrix} \tilde{X} \\ \tilde{V} \end{pmatrix}_{\text{obj}} = \begin{pmatrix} \hat{X} + \Delta t \hat{V} \\ \hat{V} \end{pmatrix}_{\text{obj}} \tag{5.17}$$

La prédiction de la position des obstacles en mouvement s'effectue sur un horizon de temps H_{pred} en considérant un pas de temps t_e :

$$\tilde{X}_{\text{obj}}^{(k+1)} = \tilde{X}_{\text{obj}}^{(k)} + t_e \tilde{V}_{\text{obj}}^{(k)}, \quad \forall\ 0 \leq k \leq H_{pred} \tag{5.18}$$

Pour chacun de ces objets mobiles modélisés, comme on l'a vu, par un cylindre d'axe vertical, on dispose d'un rayon r_{obj} et d'une hauteur h. Nous faisons ici l'hypothèse d'un sol plat et horizontal sur lequel ces cylindres sont placés avec un axe vertical. On projette donc pour chaque objet détecté sa surface occupée au sol ce qui conduit à un disque de rayon r_{obj}. En faisant cela pour tous les objets mobiles détectés, et à chaque temps de prédiction $k \in [0; H_{pred}]$, on peut estimer des cartes d'occupation binaires $\{\Xi_k\}_k$ à partir desquelles on déduit les cartes de distances à l'obstacle le plus proche : $\{dist_{\text{obst}}^{(k)}(., \Xi_k)\}_k$.

Estimation de la commande optimale

Pour chaque trajectoire prédite, on calcule un coût J qui représente les critères de la mission : évitement d'obstacles et ralliement de points. Ce coût s'exprime sous la forme de l'expression donnée à l'équation 5.19. Cette expression fait apparaître une somme de coûts, chacun associé à un objectif particulier (coût de navigation, coût lié à la proximité d'obstacles sur la trajectoire, etc.), et multiplié par une pondération permettant de régler l'importance relative de chaque objectif. Dans notre cas, le coût associé à une trajectoire s'écrit :

$$J = \omega_{\text{Nav}} J_{\text{Nav}} + \omega_{\text{Obst}} J_{\text{Obst}}. \tag{5.19}$$

J_{Nav} est le coût de ralliement de point, il favorise les trajectoires qui se rapprochent du point de ralliement $P_{\text{ralliement}}$ fixé au début de l'expérience, et s'écrit :

$$J_{\text{Nav}} = \frac{1}{H_{pred} v_{\max} t_e} \sum_{k=0}^{H_{pred}} \|P_{\text{ralliement}} - P_k\|^2, \tag{5.20}$$

où $v_{\max}$ est la vitesse maximale du robot autorisée, et $\{P_k\}_k$ les positions $\{x_k, y_k\}_k$ prédites du robot dans l'horizon de temps H_{pred}.

J_{Obst} pénalise les trajectoires qui rentrent en collision ou qui se rapprochent trop près des obstacles :

$$J_{\text{Obst}} = \frac{1}{H_{pred}} \sum_{k=0}^{H_{pred}} f\Big(dist_{\text{obst}}^{(k)}(X_k, \Xi_k)\Big), \tag{5.21}$$

où $f(dist)$ est une fonction égale à 1 si $dist < d_{\text{danger}}$ et qui décroît progressivement pour valoir 0 lorsque $dist > d_{\text{securite}}$ (cf. figure 5.7). Les obstacles ne sont donc plus pris en compte si $dist$ est supérieure à la distance de sécurité, et la pénalité est maximale lorsque $dist$ est inférieure à la distance de danger. On estimer cette distance $dist$ pour chaque position X_k prédite du robot à l'instant k, en considérant la carte d'obstacle Ξ_k correspondante.

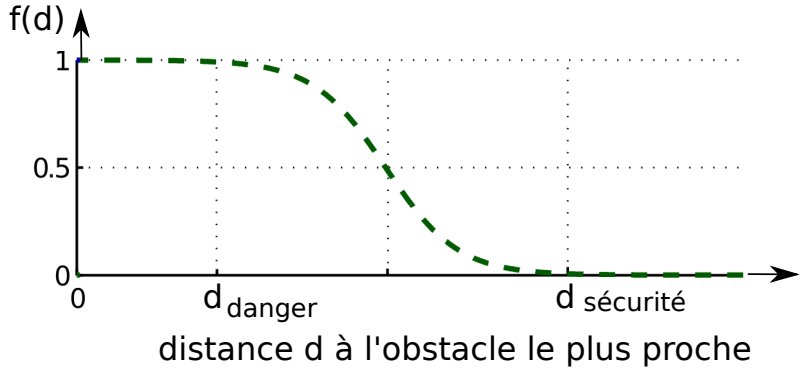

FIGURE 5.7 – Profil de la fonction $f(dist)$ qui pénalise les positions du robot dont la distance à l'obstacle le plus proche est trop faible.

La minimisation de cette fonction nous donne la séquence de commandes optimales $\{U_k^*\}_k$ à donner au robot. On applique ensuite uniquement le premier élément (v_0^*, ω_0^*) de la séquence optimale $\{(u_0^*, \omega_0^*), ...(u_{H_{pred}}^*, \omega_{H_{pred}}^*)\}$. Puis on réitère l'opération au pas de temps suivant, en utilisant les nouvelles informations sur les obstacles qui ont pu être acquises entre temps. La Figure 5.8 illustre le fonctionnement de l'algorithme de commande prédictive présenté dans cette sous-section. L'algorithme construit un ensemble fini de séquences de commande possibles en échantillonnant les paramètres (vitesse et vitesse angulaire). Chaque séquence se traduit par un trajet qui est valué par le critère J présenté précédemment. Le trajet optimal (en violet) est un compromis entre l'évitement de l'obstacle en mouvement qui se rapproche du robot pour couper sa trajectoire, et l'arrivée rapide sur le point de ralliement prédéfini.

5.3 Navigation robotique autonome en présence d'objets mobiles à l'aide de la stéréo : preuve de concept

Dans cette section nous présentons une validation expérimentale du système robotique détaillé en section. 5.2 qui repose sur les algorithmes de perception stéréo présentés Chap. 2, 3 et 4. L'objectif est double : d'une part on souhaite vérifier que nos algorithmes qui ont été éprouvés jusqu'à présent sur le dataset KITTI, fonctionnent également de manière efficace sur notre robot Summit XL équipé d'un banc stéréo de plus petite baseline (18cm au lieu de 54cm) et avec une résolution plus basse (480x640 au lieu de 370x1200 environ). D'autre part on prend cette ex-

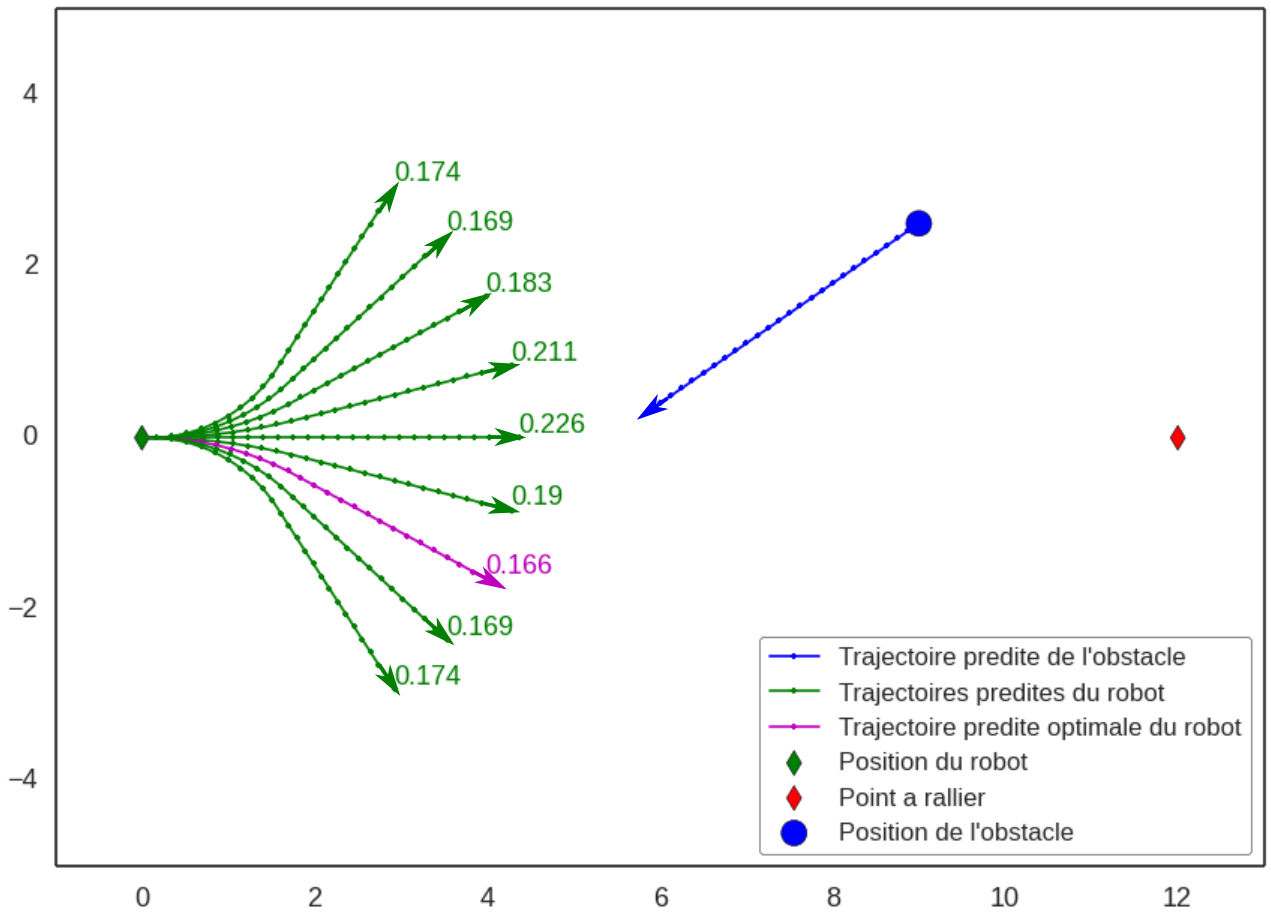

FIGURE 5.8 – Illustration du fonctionnement de l'algorithme de commande prédictive qui détermine la trajectoire de coût J minimum. La trajectoire optimale (en violet) est calculée à la fois pour éviter l'obstacle mobile avec suffisamment de marge, et à la fois pour se rapprocher au plus vite du point de ralliement.

périence comme une preuve de concept d'un système de navigation robotique basé sur la vision stéréo, permettant d'adapter en temps-réel sa trajectoire pour éviter d'entrer en collision avec des objets mobiles.

Cette section est organisée de la manière suivante : la preuve de concept ainsi que ses hypothèses simplificatrices sont décrites sous-section 5.3.1, les résultats expérimentaux sont présentés en 5.3.2, et les conclusions et remarques finales sont formulées en sous-section 5.3.3.

5.3.1 Description de la preuve de concept

Pour notre expérience, nous utilisons la plateforme robotique à roues Summit XL présentée figure 5.1 équipée d'un PC embarqué doté d'un GPU Nvidia GT 640.

La mission de navigation est la suivante : le robot part d'un point initial A et doit rallier le point B situé 15m devant, en évitant toute collision avec les objets mouvement grâce au module de commande prédictive présenté en 5.2.5. L'expérience est faite sur une route dégagée et suffisamment large pour que le robot puisse manoeuvrer pour éviter les objets en mouvement, tout en restant sur celle-ci. Les photos de cet environnement sont données à la figure 5.9, avec les positions A et B de la mission. Les vitesses maximales linéaire et angulaire autorisées sont $v_{max} = 0.7m/s$ et $\omega_{max} = 0.5rad/s$. La distance de sécurité d_{scurit} (cf. Fig. 5.7) est fixée à 2m.

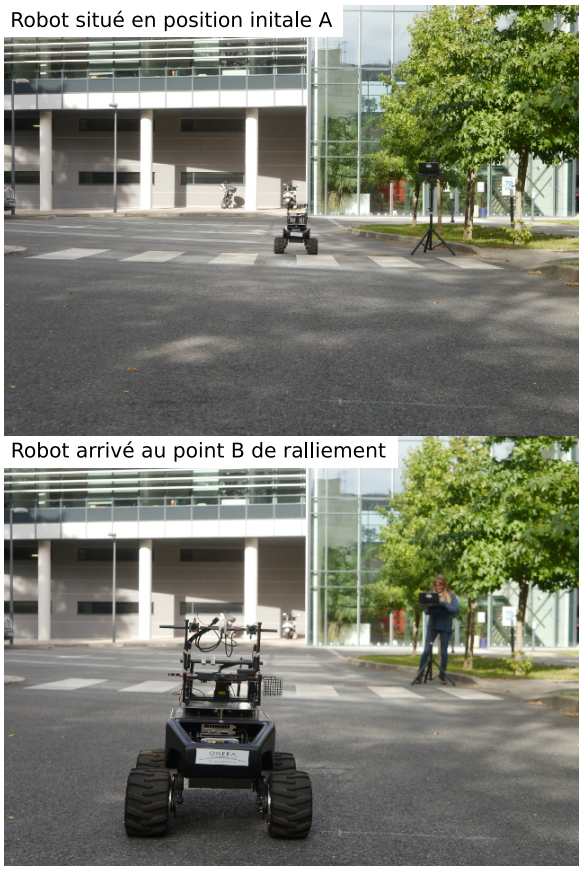

FIGURE 5.9 – Position initiale et point de ralliement pour notre expérience.

Pour cette preuve de concept, plusieurs hypothèses simplificatrices sont faites sur l'environnement :

- **Monde ouvert** : on suppose que l'environnement dans lequel évolue le robot est vide et ne contient aucun obstacle en dehors des objets en mouvement.
- **Monde plat** : on suppose que le robot et les objets en mouvement se déplacent au sol, et que le sol peut être représenté par un plan.
- **Vitesse et orientation constantes des objets mobiles** : dans notre scénario, les piétons qui font office d'objets mobiles se déplacent avec une vitesse

plus ou moins constante et en gardant la même direction.

L'hypothèse de monde ouvert est faite pour simplifier l'analyse en n'étudiant que les obstacles en mouvement. Mais les obstacles statiques pourraient être simplement ajoutés dans l'algorithme de commande prédictive en modifiant les cartes d'occupation binaires $\{\Xi_k\}_k$ (cf. sous-section 5.2.5).

L'hypothèse de monde plat quant à elle, permet de calculer les trajectoires sur une carte 2D et simplifie grandement l'algorithme de commande prédictive qui s'appuie sur des cartes d'occupation $\{\Xi_k\}_k$ en 2D. Elle simplifie aussi l'analyse de la scène dans la partie perceptive. Le cadre des routes plates n'est pas sans intérêt, c'est une situation fréquente, mais bien sûr il serait utile de considérer des routes présentant des dénivelés. On pourrait par exemple considérer un sol localement plan en réestimant régulièrement le plan 3D correspondant par stéréovision. On se ramènerait ainsi dans le cas 2D, en projetant sur ce plan les positions et vitesses des objets mobiles et du robot.

L'hypothèse que les objets en mouvement ont une vitesse et une direction constante, sert pour la prédiction de leur trajectoire faite par le module de commande prédictive. D'autres modèles moins restrictifs pour la dynamique des objets peuvent être considérés, par exemple : une vitesse linéaire et angulaire constante, ou bien une accéleration constante,... Cependant cela nécessite de pouvoir mesurer l'accelération ou la vitesse angulaire d'un objet, ce qui s'avère difficile et peu précis à partir d'un banc stéréo. Au contraire, la vitesse quant à elle est directement déduite des algorithmes d'odométrie et de flot de scène. Cette hypothèse est utilisée par le filtre de Kalman de la commande prédictive, qui offre cependant une certaine souplesse en permettant une certaine variation de la vitesse d'un objet via l'influence de la matrice de bruit.

5.3.2 Présentations des résultats expérimentaux

Nous avons effectué de nombreux essais de l'expérience décrite en sous-section 5.3.1, avec des objets en mouvement décrivant différentes trajectoires. Nous présentons ici deux cas assez représentatifs qui illustrent bien le fonctionnement de notre système et certaines difficultés rencontrées. L'objectif de cette expérience est aussi de vérifier que nos algorithmes s'exécutent rapidement avec l'ordinateur embarqué sur robot. Nous indiquons donc également les fréquences auxquelles tourne notre noeud de perception dans le système ROS.

Rapport de l'expérience numéro 1

Pour la première expérience, nous avons choisi un cas simple avec un seul piéton qui marche à environ 1.5 m/s et qui coupe la trajectoire du robot avec un angle d'environ 45°.

Durant l'expérience, les estimations des trajectoires effectuées/prédites, des positions et vitesses du robot et des objets mobiles (ainsi que leur rayon), sont enregistrés dans un "rosbag" (ce terme désigne la structure de sauvegarde de ROS). La figure 5.10 qui a été faite à partir de ces données enregistrées au cours de l'expérience, présente les différents positions et trajectoires des éléments de la scène à 6 instants clés :

- à t_1 : le robot quitte sa position initale et –puisqu'aucun objet mobile n'est détecté– il se dirige droit vers le point de ralliement situé 15m plus loin.
- à t_2 : un objet en mouvement est détecté à proximité du robot et sa trajectoire prédite (indiquée en pointillés verts) coupe la route du robot. Ce dernier prévoit donc d'adapter sa future trajectoire (indiquée en pointillés rouges).
- à t_3 : le robot effectue la manoeuvre de contournement de l'obstacle mobile et prévoit de réajuster sa trajectoire pour se rediriger vers le point de ralliement ensuite.
- à t_4 : l'obstacle a été évité et le robot va droit sur le point de ralliement. Notez que l'obstacle mobile ne figure plus sur la carte car il est sorti du champ de vision du robot. Lorsqu'un objet mobile sors du champ de vision, il est encore modélisé quelques secondes à l'aide de sa dernière position et vitesse filtrée, avant d'être "oublié" par le robot et supprimé de la carte d'obstacles.
- à t_5 : le robot a atteint le point de ralliement, et l'on peut constater que sa trajectoire diffère de la simple ligne droite horizontale qu'il aurait eu en ce point sans la présence de l'obstacle en mouvement.

Dans cette expérience, le robot accomplit sa mission avec succès puisqu'il détecte suffisamment à l'avance l'obstacle mobile pour adapter sa propre trajectoire et maintenir une distance de sécurité avec celui-ci, tout en se rapprochant du point de ralliement.

Rapport de l'expérience numéro 2

Dans la deuxième expérience, deux piétons sont impliqués et leur trajectoires ont un angle plus faible avec l'axe x (i.e. l'axe avant du robot) que dans l'expérience 1. Le premier piéton se déplace de façon parallèle à l'axe x et légèrement décalé par rapport au robot, de manière à le frôler si celui-ci continue d'avancer droit vers le point de ralliement. Le deuxième piéton coupe la route du robot avec un angle d'environ 30°. Les résultats de cette deuxième expérience sont illustrés figure 5.11 où sont indiquées les positions et trajectoires du robot et des objets en mouvement détectés, à 6 instants clés :

- à t_1 : le 1er piéton qui se déplace parallèlement à l'axe x est tardivement détecté et le robot planifie une trajectoire pour le contourner.
- à t_2 : le robot s'est légèrement dévié de sa trajectoire après avoir détecté le piéton, mais n'a pas réagit suffisamment tôt pour respecter la distance de sécurité.

- à t_3 : une fausse détection est faite sur la gauche du robot (au niveau des arbres) mais la position de l'objet détecté en mouvement est trop loin (et même plus loin que le point de ralliement) et n'influence pas la trajectoire du robot.
- à t_4 : le 2e piéton qui coupe l'axe x avec un angle de 30°, est détecté correctement et suffisamment à l'avance pour permettre au robot de calculer une trajectoire prédite adéquate.
- à t_5 : le robot a bien contourné l'objet en gardant sa distance de sécurité.
- à t_6 : en absence d'objet mobile détecté, le robot finit par arriver au point de ralliement en se dirigeant droit sur celui-ci.

Dans cette deuxième expérience, le robot n'a pas correctement accompli sa mission puisqu'il a détecté tardivement le premier piéton et n'a pas su maintenir une distance de sécurité avec lui. En dehors de cela, le robot réussit à bien détecter le second piéton et à effectuer une bonne trajectoire jusqu'au point de ralliement, malgré une fausse détection.

Temps de calcul mesurés pour les algorithmes de perception

Pour obtenir une estimation du temps de calcul de nos algorithmes de perception stéréo, nous avons enregistré tous les messages du topic */ObjetsMobiles* envoyés par le noeud de perception au noeud de commande prédictive durant l'expérience (cf. la figure 5.4 du schéma ROS utilisé). En regardant l'écart entre les dates de publication des messages */ObjetMobiles* successifs, on obtient des durées Δt qui constituent une approximation du temps de calcul nécessaire pour détecter et modéliser les objets en mouvement. L'histogramme de la figure 5.12 présente ces durées Δt mesurées lors d'une des expériences. Notons que ce temps Δt peut être légèrement supérieur au temps de calcul du noeud de perception, car il inclue aussi le délai induit par la synchronisation des données d'entrée du noeud, à savoir : une paire stéréo et le résultat de l'odométrie visuelle. Cette synchronisation explique le fait que le graphe de la Figure 5.12 présente des paliers toutes les 50ms (étant donnée la fréquence de 20Hz d'acquisition d'image)

On voit sur la Figure 5.12 que le noeud de perception s'exécute à une cadence de 4Hz à 20Hz. En pratique, on constate que le temps d'exécution du noeud augmente en fonction du nombre d'objets mobiles détectés : dans le cas où il n'y a pas d'objet en mouvement la fréquence d'exécution peut atteindre 20Hz, et lorsque des objets mobiles sont observés elle peut descendre jusqu'à 4Hz. Cette fréquence de 4Hz est nettement plus faible que celle de 10Hz obtenue sur notre station de travail, car le robot dispose d'un GPU beaucoup moins puissant que la GeForce GTX Titan utilisée aux Chapitres 3 et 4. Cependant cette cadence permet au robot d'être suffisamment réactif pour adapter sa trajectoire en temps-réel et éviter les objets en mouvement, comme le montrent les résultats des expériences 1 et 2 (cf. figures 5.10 et 5.11),

sauf dans le cas difficile où l'axe de progression de l'objet est quasi aligné avec la trajectoire du robot.

5.3.3 Discussion des résultats et conclusion

Dans les nombreux essais réalisés nous avons constaté que notre module de perception rencontre plusieurs difficultés. Comme mentionné à la sous-section 5.3.2, des fausses détections apparaissent de temps en temps, et notre système peine à détecter les objets qui se déplacent près de l'axe x du robot et qui coupent la route du robot avec un angle faible. Enfin le troisième problème que nous avons le plus souvent observé, concerne les détections intermittentes des objets en mouvement qui ne sont pas toujours détectés en continu sur plusieurs instants consécutifs, ce qui perturbe la perception.

Fausses détections

Malgré la gestion temporelle des détections présentée sous-section 5.2.4 qui impose une certaine cohérence temporelle (cf. critère 5.14), quelques fausses détections apparaissent encore au cours de nos expériences. Ces fausses détections sont ponctuelles, et généralement les objets détectés à tort sont distants et n'influencent donc pas la trajectoire calculée par le robot. Cependant au cours de nos expériences nous avons observé plusieurs fois le robot dévier sa trajectoire à cause d'une fausse détection.

Ces fausses détections sont dues à des erreurs d'estimation, notamment à des outliers dans le calcul des cartes de disparités stéréo qui peuvent causer d'importants changements de profondeur entre deux instants pouvant être perçus comme un objet se rapprochant à vitesse élevée. Notons que le critère (5.14) de cohérence temporelle est d'autant moins sévère que les incertitudes Σ_X et Σ_V sur la position et la vitesse des points sont grandes. Or l'incertitude de triangulation est d'autant plus élevée que les points sont distants (cf. Eq. 4.29). Donc pour des outliers stéréo affectant des points distants, Σ_X et Σ_V sont grands et le critère 5.14 n'est plus assez contraignant et efficace pour assurer une cohérence temporelle. Même si l'objet est distant, ce type de fausse détection peut influer la trajectoire si la vitesse estimée de l'objet est très élevée.

Une solution naïve pour éviter ce problème est d'imposer une contrainte sur la vitesse des objets à détecter : en supposant a priori que celle-ci est inférieure à une valeur v_{max}, et en considérant qu'un objet de vitesse excédant v_{max} est un outlier. Une autre solution plus élégante est d'imposer une cohérence temporelle non pas entre deux, mais entre trois détections consécutives ou plus avant de valider une piste.

Détection difficile dans l'axe x du robot

Lors de nos expériences nous avons constaté des difficultés pour détecter des objets qui se déplacent dans la même direction que le robot, en sens opposé. C'est

à dire vers le robot, sur l'axe x du repère local de naviguation défini Figure 5.6, qui correspond à l'axe z du repère caméra défini Figure 2.2.

Ces non-détections d'objets mobiles (aussi appelées "Faux Négatifs" à la section 4.4) sont dues aux valeurs $\chi^2(M)$ qui sont trop faibles et en dessous du seuil de détection, pour les pixels correspondant à l'objet mobile. En effet il s'agit là du cas le plus défavorable pour la détection : le mouvement apparent est minimal pour un objet se déplaçant suivant l'axe de visée de la caméra. Au contraire, le mouvement apparent d'un objet est maximal lorsque celui-ci se déplace perpendiculairement à l'axe de visée de la caméra. D'autre part, ce cas est également défavorable car si l'objet qui se rapproche est au centre de l'image, alors il induit un mouvement apparent divergent. Ce mouvement est en contradiction avec l'hypothèse faite par l'algorithme de flot optique FOLKI, qui suppose que le mouvement apparent dans l'image peut être localement considéré comme une translation 2D.

Pour pallier ce problème, une solution consisterait à augmenter la baseline temporelle : non pas en augmentant le temps entre les deux paires stéréo utilisées pour la détection (ce qui nuirait à la réactivité du système), mais plutôt en accumulant dans le temps les estimations du flot de scène. Pour ce faire, on pourrait adopter la stratégie de Rabe et al. [Rabe et al., 2010] qui appliquent un filtre de Kalman étendu sur chaque point de l'image observée pour estimer la position 3D (qui se résume à la distance à la caméra puisque l'objet est situé sur le rayon passant par le pixel considéré) et la vitesse 3D à partir des mesures suivantes : coordonnées 2D du pixel considéré dans l'image, disparité et changement de disparité entre deux instants.

Détections intermittentes

Le plus gros problème rencontré lors des essais expérimentaux est l'intermittence des détections : les piétons en déplacement sont rarement détectés en continu tout au long de leur passage dans le champ de vision de la caméra stéréo. Ce phénomène est illustré Figure 5.13, où les cartes $\chi^2(M)$ sont présentées avec des fenêtres carrées indiquant les détections : ces carrés sont de taille 30x30cm et sont centrés sur le point central de l'objet, avec une couleur blanche pour les détections validées par le critère de cohérence temporelle, et grise sinon.

Plusieurs raisons expliquent ces détections intermittentes. D'une part, les piétons ne sont pas vraiment en déplacement à vitesse constante : leur marche, qui est observée à 4Hz-10Hz par le module de perception, peut comporter entre deux enjambées des phases de déplacement plus faible. Dans ces phases, le piéton peut échapper à la détection (e.g. l'instant t_1 Figure 5.13). Par ailleurs ces variations de vitesse apparente sont plus marquées sur les jambes : dans l'exemple présenté il arrive parfois que seules les jambes soient détectées. Ceci a pour effet de décaler le centre de l'objet mobile qui est estimé en prenant la position médiane des pixels sélectionnés par seuillage de l'image $\chi^2(M)$. En conséquence la détection ne satis-

fait pas le critère de cohérence temporelle (5.14) : la détection est alors invalidée et le module de commande prédictive ne "voit" donc pas l'objet en mouvement (cf. l'instant t_3 Figure 5.13).

De manière générale, l'intermittence des détections est symptomatique de notre algorithme de perception qui ne détecte pas des "objets mobiles", mais des "objets en mouvement". En effet, puisque notre détection s'effectue de manière instantanée à partir d'estimations (position, mouvement 3D des points...) faites uniquement sur deux paires stéréo consécutives, elle ne tient pas compte –hormis pour la validation– des précédentes détections. On pourrait imaginer de relier des détections d'un instant à l'autre en utilisant le flot optique que nous estimons par ailleurs. Ainsi, même si un piéton s'arrête, on pourrait continuer de le considérer (du moins pour un certain temps) dans notre modélisation d'objet mobile. La détection se ferait ainsi bien sur l'ensemble du piéton (et pas seulement ses jambes) même lorsque celui-ci ralentit, et le module de perception serait moins sensible aux variations de vitesse au cours de la marche du piéton, puis sa vitesse serait filtrée par le filtre de Kalman. Une autre approche d'intégration temporelle, qu'on peut relier au "track-before-detect" en RADAR, pourrait s'inspirer de la méthode de Rabe et al [Rabe et al., 2010] en attribuant à chaque pixel un filtre de Kalman, pour obtenir une estimation du signal (position et mouvement 3D) plus stable dans le temps et effectuer la détection après ce filtrage.

Conclusion sur les expérimentations

Le système de perception que nous avons testé ici utilise une gestion temporelle minimaliste des détections (gestion naïve des pistes et contrainte de cohérence temporelle sur seulement deux instants successifs), et une détection "d'objets en mouvement" plutôt que "d'objets mobiles". Il en résulte quelques fausses détections, et surtout des détections souvent intermittentes.

Malgré ces problèmes pour lesquels nous avons mentionné des pistes de solution (notamment en suivant les pixels détectés, ou avec des méthodes d'intégration temporelle du signal), dans la plupart des expériences le robot effectue sa mission avec succès : il parvient à atteindre le point de ralliement tout en adaptant dynamiquement sa trajectoire pour éviter les objets en mouvement. Ces expériences ont montré que nos algorithmes de perception stéréo d'environnement dynamique sont assez rapides pour s'exécuter à la volée sur une plateforme embarquée, et pour permettre de construire un système robotique suffisamment réactif pour éviter les collisions avec les objets en mouvement.

Parmi les pistes d'amélioration autres que celles mentionnées précédemment, nous considérons l'utilisation de méthodes plus haut niveau en complément de notre approche purement géométrique. Par exemple en employant des méthodes de reconnaissance d'objets (piétons, voiture,...), ou bien des algorithmes capables d'ap-

prendre l'apparence d'un objet quelconque que l'on aurait préalablement détecté comme étant en mouvement (par exemple en s'inspirant de l'approche *Tracking-Learning-Detection* proposée par Kalal *et al.* [Kalal et al., 2012]). La combinaison de notre système stéréo avec d'autres types de capteurs devrait aussi être envisagée notamment pour pallier les situations où la vision est mise en défaut, telles que les aveuglements momentanés (fumée, suréclairement) ou les mouvements d'objets dans l'axe comme celui observé dans l'expérimentation 2. Par exemple on pourrait envisager l'utilisation de capteurs à ultrasons pour la perception des objets proches (situés à quelques mètres du robot), ou encore de RADAR Doppler qui sont, eux, au contraire, particulièrement adaptés pour détecter des objets dont le mouvement est aligné avec l'axe du robot.

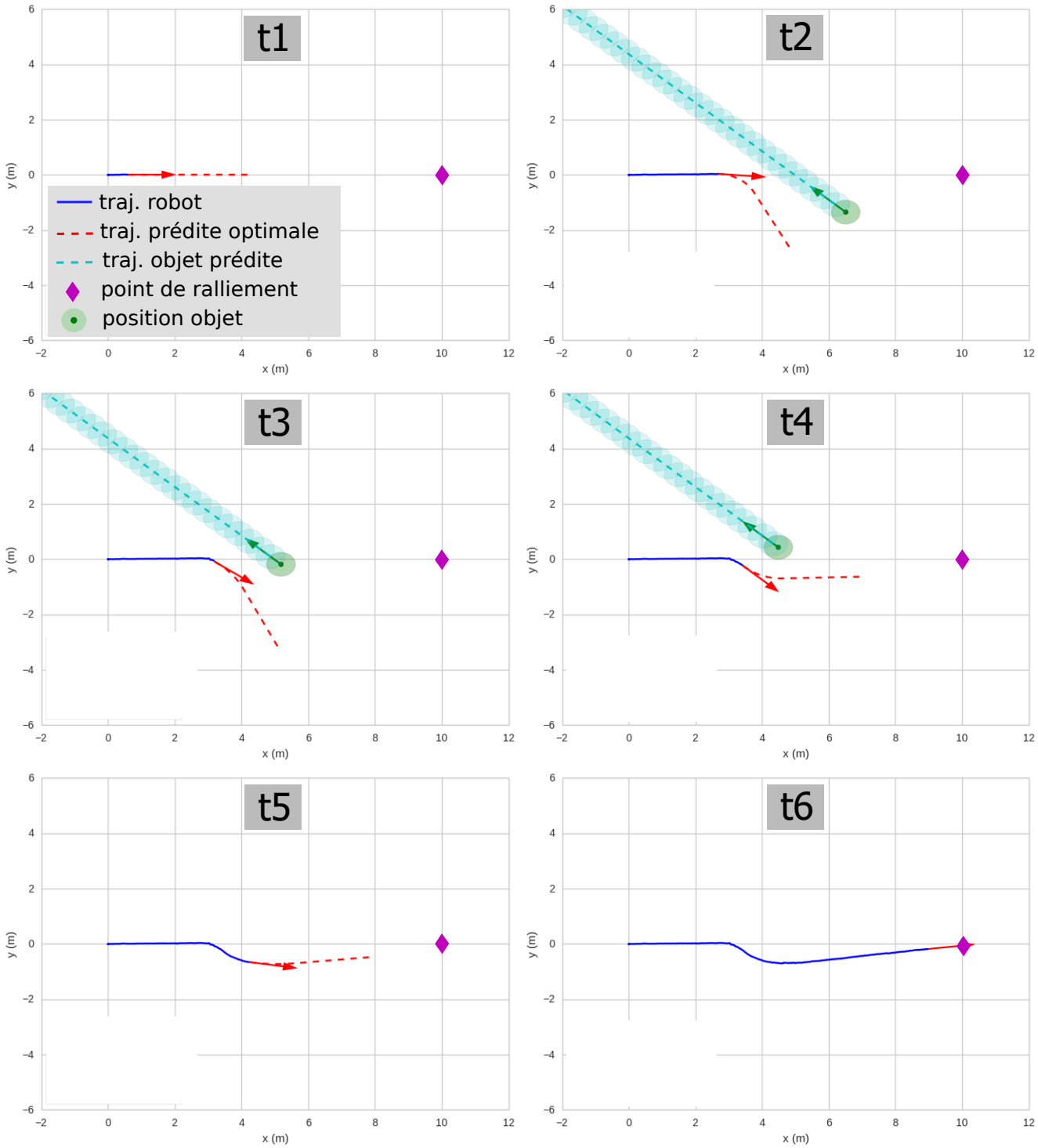

FIGURE 5.10 – Vue d'oiseau des positions et trajectoires du robot et de l'objet mobile dans l'expérience 1. À t5 l'objet sort du champ de vision et n'est plus représenté.

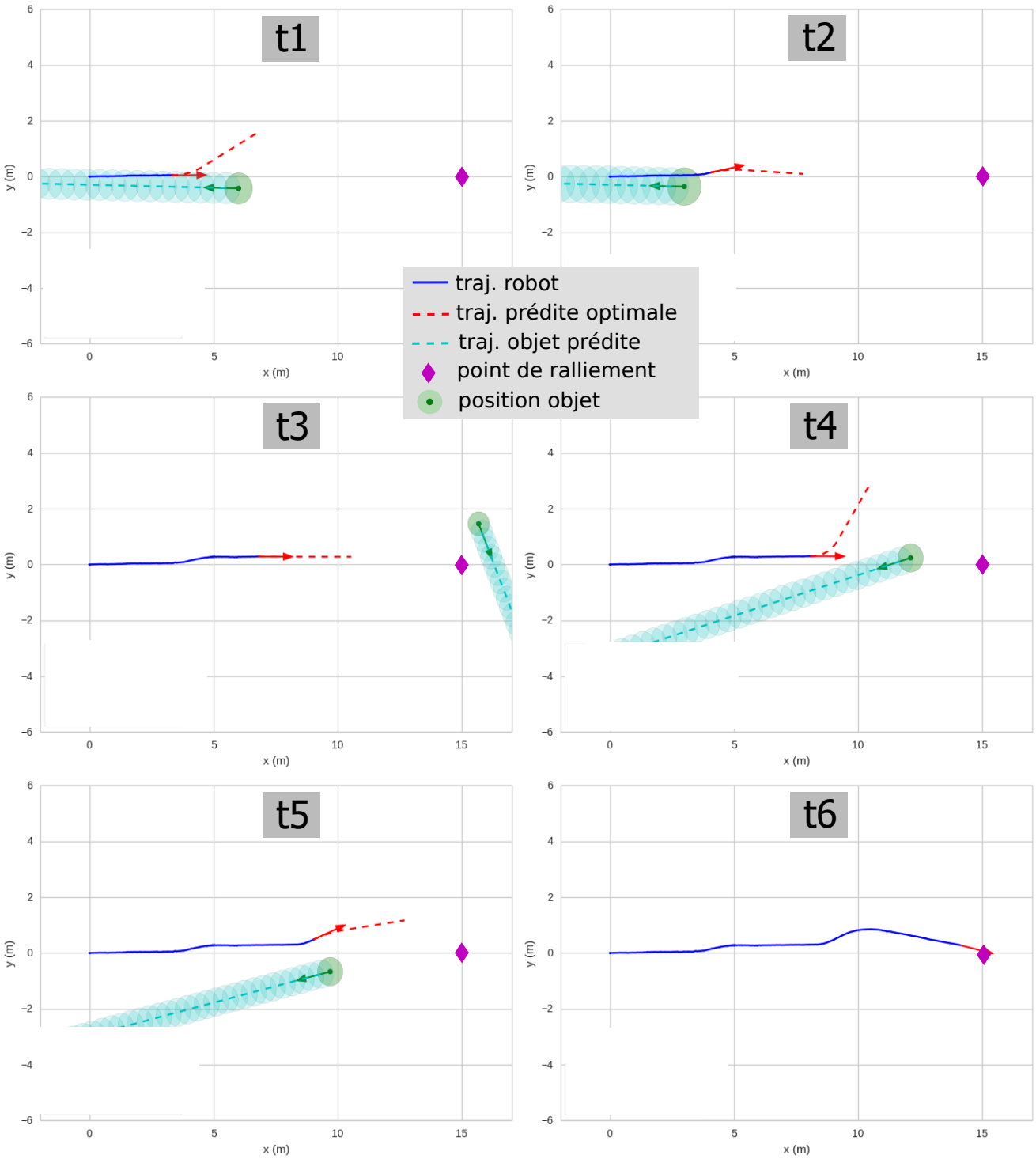

FIGURE 5.11 – Vue d'oiseau des positions et trajectoires du robot et de l'objet mobile dans l'expérience 2.

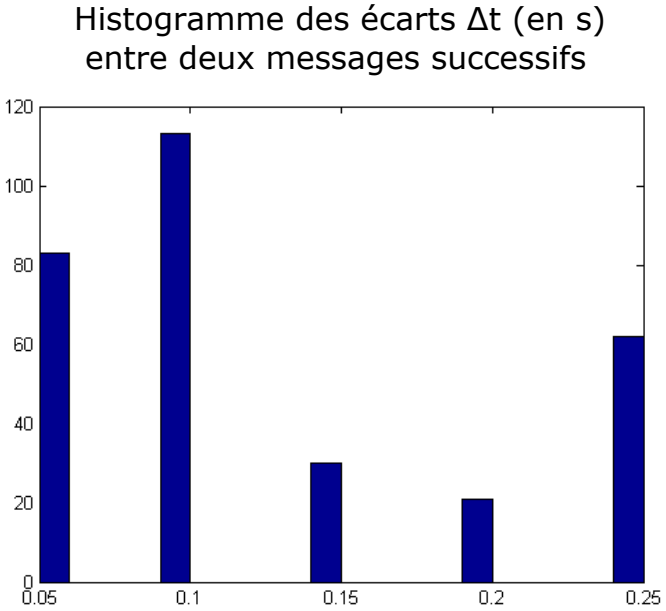

FIGURE 5.12 – Histogramme des écart Δt (s) entre les messages "*/ObjetsMobiles*" successifs sortant du noeud de perception. On note qu'ils sont multiples de 50ms.

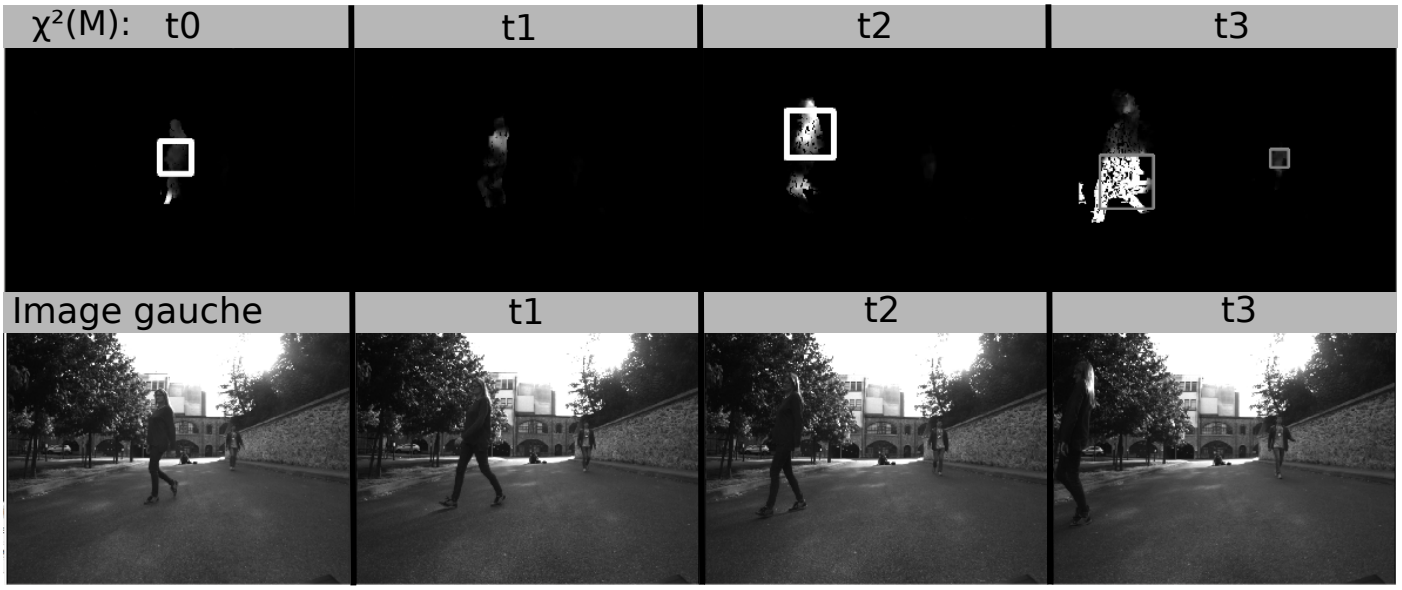

FIGURE 5.13 – Illustration de détections intermittentes avec carte $\chi^2(M)$ (en haut) et l'image de la caméra gauche (en bas) à 4 instants successifs. Les objets détectés sont indiqués par un carré de 30x30cm, centré sur le point central de l'objet et de couleur blanche si la détection est validée, et grise sinon.

Chapitre 6

Conclusion

Sommaire

6.1 Rappel de la démarche de thèse & Contributions

L'objectif de ces travaux de thèse était de concevoir un système de perception stéréoscopique embarqué, permettant une navigation robotique autonome en environnement dynamique (i.e. comportant des objets mobiles). Pour cela, nous nous sommes imposé plusieurs contraintes :

- Puisque l'on souhaite pouvoir naviguer en terrain inconnu et en présence de tout type d'objets mobiles, nous avons adopté une approche purement géométrique.
- Pour assurer une couverture maximale du champ visuel nous avons choisi d'employer des méthodes d'estimation denses qui traitent chaque pixel de l'image.
- Puisque les algorithmes utilisés doivent pouvoir s'exécuter en embarqué sur un robot, nous avons attaché le plus grand soin à sélectionner ou concevoir des algorithmes particulièrement rapides, pour nuire au minimum à la réactivité du système.

Nous rappelons ici la démarche qui a été présentée dans ce manuscrit de thèse et les contributions qui ont été faites pour remplir cet objectif.

Tout d'abord au **Chapitre 2**, nous avons étudié plusieurs algorithme de stéréo qui permettent d'estimer une carte de disparité dont on peut déduire une carte de profondeur (par triangulation). Grâce à cette évaluation nous avons mis en évidence l'algorithme ACTF [Sizintsev and Wildes, 2010] qui ne figure pas sur les benchmarks KITTI 2012 et KITTI 2015, mais qui offre un excellent compromis précision/temps de calcul. Nous avons également proposé une méthode par Consensus de Régions (CoR), comme une version simplifiée des travaux de Chakrabarti et al [Chakrabarti et al., 2015], pour filter une carte de disparité. Nous avons codé les

algorithmes ACTF et CoR en C/CUDA pour profiter de l'accélération des calculs sur carte graphique (GPU), ce qui nous a permis de montrer que la combinaison ACTF+CoR donne une bonne précision (1.4 px d'erreur moyenne) et qu'elle s'exécute très rapidement (en 19ms sur GPU GeForce GTX Titan).

Afin de percevoir les objets mobiles et estimer leur mouvement, nous avons ensuite décrit au **Chapitre 3** des méthodes de calcul du déplacement du banc stéréo (Odométrie Visuelle), et des méthodes d'estimation du mouvement apparent 2D (Flot Optique) et 3D (Flot de Scène). Partant du constat que seul l'algorithme d'estimation du flot optique FOLKI (appartenant aux approches locales) permet un calcul en temps-réel, nous avons proposé et testé plusieurs modifications de celui-ci qui améliorent légèrement ses performances au prix d'une augmentation de son temps de calcul. Concernant le flot de scène, aucun algorithme existant ne permet d'atteindre la vitesse d'exécution souhaitée et nous avons proposé une nouvelle approche découplant structure et mouvement pour estimer rapidement le flot de scène. Trois stratégies ont été proposées pour exploiter cette décomposition structure-mouvement. Parmi ces stratégies, nous avons montré que PCOF permet d'estimer très rapidement du flot de scène avec une précision relativement bonne. A notre connaissance, il s'agit du seul algorithme publié de calcul du flot de scène capable de s'exécuter à cadence vidéo sur les données KITTI (10Hz).

Pour détecter les objets en mouvement et les segmenter dans l'image, nous avons présenté au **Chapitre 4** différents modèles statistiques et différents résidus sur lesquels fonder une détection par seuillage d'un critère χ^2. Nous avons proposé une modélisation statistique rigoureuse qui tient compte de toutes les incertitudes d'estimation, notamment celles de l'odométrie visuelle, ce qui n'avait pas été fait à notre connaissance dans le contexte de la détection d'objets mobiles. Nous avons aussi proposé un nouveau résidu pour la détection, en utilisant la méthode par prédiction d'image que nous avions présentée au Chapitre 3, ce qui permet de faciliter la propagation des incertitudes et l'obtention du critère χ^2. Nous avons ensuite proposé une méthode simple et rapide pour segmenter les objets mobiles dans l'image. Enfin nous avons mis en place un protocole expérimental pour évaluer les algorithmes de détection, et montré, sur des exemples tirés de la base KITTI, le gain apporté par le résidu et le modèle d'erreur que nous avons proposés.

Afin de valider expérimentalement notre système de perception en embarqué sur une plateforme robotique, nous avons implémenté nos codes sous ROS et codé certains algorithmes (ACTF, CoR, et le calcul des cartes $\chi^2(M)$) en CUDA pour une accélération sur GPU. Le **Chapitre 5** décrit les principes de base de ROS et de la programmation sur GPU, ainsi que le système de perception et de navigation [1] utilisé pour la preuve de concept que nous avons effectué et qui montre que notre systeme de perception, convient à une application embarquée. Nous avons aussi souligné ses difficultés et des pistes d'améliorations.

1. Nous rappelons que la partie commande de ce système a été développée par Hélène Roggeman, doctorante à l'ONERA/DCPS

6.2 Perspectives

Parmi les pistes d'amélioration possibles pour les travaux que nous avons présentés dans ce manuscrit, nous identifions deux axes principaux : le filtrage temporel, et l'exploitation d'algorithmes de reconnaissance et d'apprentissage.

Les données considérées sont des flux vidéo issus des caméras embarquées qui présentent une certaine cohérence temporelle : les objets perçus par le système comme comme étant en mouvement à un instant donné, sont susceptibles de l'être encore (et avec une vitesse comparable) à l'instant suivant. Ainsi nous pourrions filtrer temporellement les positions et vitesses que l'on estime en chaque pixel, en utilisant une approche semblable à celle de Rabe *et al.* [Rabe et al., 2010] qui attribue à chaque pixel un filtre de Kalman pour filtrer le flot de scène estimé. Une alternative plus économique en temps de calcul est d'attribuer un filtre de Kalman non pas à chaque pixel, mais à chaque objet mobile détecté et segmenté dans l'image, une approche très souvent proposée dans la littérature.

Une autre manière d'exploiter la cohérence temporelle serait de suivre dans l'image courante les pixels des objets mobiles précédemment détectés, afin d'améliorer le critère de détection χ^2 et pouvoir maintenir la détection d'un objet mobile même lorsque celui-ci ralentit ou s'arrête (comme dans le cas présenté Fig. 5.13).

L'approche *structure-mouvement* et l'algorithme PCOF présentés à la sous-section 3.3.3 pourraient aussi être modifiés pour bénéficier de l'accumulation temporelle des informations. En effet, plutôt que de calculer la partie statique du flot de scène (à partir de la carte de profondeur et du déplacement de la caméra) pour prédire l'image I_t^{pred} en supposant la scène statique, on pourrait également tenir compte du mouvement propre estimé pour chaque pixel à l'instant précédent, pour l'inclure dans la prédiction qui tiendrait alors aussi compte des objets mobiles précédemment détectés. Si l'on implémentait des filtres de Kalman en chaque pixel de l'image, cette prédiction tenant compte du mouvement estimé jusqu'à présent consisterait alors simplement à utiliser le flot de scène prédit par l'ensemble de ces filtres de Kalman, pour prédire le flot $\mathbf{u}_{pred}$ et l'image I_t^{pred}. La phase de *correction* de l'algorithme PCOF se réduirait alors à l'estimation du flot optique qui serait induit par la variation de vitesse des objets mobiles. Elle pourrait être mise en oeuvre, comme dans le filtre de Kalman, en tenant compte de la propagation d'incertitude au premier ordre par la covariance.

Enfin, en complément de notre approche purement géométrique, nous pourrions utiliser des algorithmes de traitement d'image plus haut niveau de reconnaissance (e.g. de piétons, voitures, etc.) et d'apprentissage afin d'exploiter au mieux les informations sémantiques et d'apparence présentes dans l'image dont nous n'avons pas tenu compte jusqu'à présent. Nous pourrions en particulier exploiter une méthode de type *Tracking-Learning-Detection* [Kalal et al., 2012], qui permettrait d'apprendre l'apparence d'un objet quelconque que l'on aurait préalablement détecté. Les propositions dans ce domaine se sont multipliées ces derniers temps avec le développement des approches utilisant des réseaux de neurones convolutionnels. Un travail

cherchant à combiner ces techniques avec des informations géométriques telles que celles que nous fournissont serait certainement intéressant.

6.3 Publications

Communications dans des conférences internationales

[1] Derome, M., Plyer, A., Sanfourche, M., & Le Besnerais, G. (2014, December). Real-time mobile object detection using stereo. In Control Automation Robotics & Vision (ICARCV), 2014 13th International Conference on (pp. 1021-1026). IEEE.

[2] Derome, M., Plyer, A., Sanfourche, M., & Le Besnerais, G. (2016, September). A Prediction-Correction Approach for Real-Time Optical Flow Computation Using Stereo. In German Conference on Pattern Recognition (pp. 365-376). Springer International Publishing.

Articles de revue internationale

[3] Derome, M., Plyer, A., Sanfourche, M., & Besnerais, G. L. (2015). Moving object detection in real-time using stereo from a mobile platform. *Unmanned Systems*, 3(04), 253-266.

[4] Roggeman, H., Marzat, J., Derome, M., Sanfourche, M., Eudes, A., Besnerais, G. L. (2017, February). Detection, estimation and avoidance of mobile objects using stereo-vision and model predictive control. *Soumis à IEEE Robotics and Automation Letters (RA-L) ainsi qu'à la conférence IEEE IROS (soumission jointe)*

Bibliographie

[Alcantarilla et al., 2012] Alcantarilla, P. F., Yebes, J. J., Almazán, J., and Bergasa, L. M. (2012). On combining visual slam and dense scene flow to increase the robustness of localization and mapping in dynamic environments. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1290–1297. IEEE. (Cité en pages 60, 89, 92, 96 et 101.)

[Bajracharya et al., 2012] Bajracharya, M., Ma, J., Howard, A., and Matthies, L. (2012). Real-time 3d stereo mapping in complex dynamic environments. In *International Conference on Robotics and Automation-Semantic Mapping, Perception, and Exploration (SPME) Workshop*, volume 15. (Cité en page 123.)

[Bak et al., 2014] Bak, A., Bouchafa, S., and Aubert, D. (2014). Dynamic objects detection through visual odometry and stereo-vision : a study of inaccuracy and improvement sources. *Machine vision and applications*, pages 1–17. (Cité en pages 92, 97, 98 et 99.)

[Baker and Matthews, 2004] Baker, S. and Matthews, I. (2004). Lucas-kanade 20 years on : A unifying framework. *International journal of computer vision*, 56(3) :221–255. (Cité en page 47.)

[Baker et al., 2011] Baker, S., Scharstein, D., Lewis, J., Roth, S., Black, M. J., and Szeliski, R. (2011). A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, 92(1) :1–31. (Cité en page 51.)

[Bar-Shalom et al., 2011] Bar-Shalom, Y., Willett, P. K., and Tian, X. (2011). *Tracking and data fusion*. YBS publishing. (Cité en page 125.)

[Barron et al., 1994] Barron, J. L., Fleet, D. J., and Beauchemin, S. S. (1994). Performance of optical flow techniques. *International journal of computer vision*, 12(1) :43–77. (Cité en page 90.)

[Belhumeur and Mumford, 1992] Belhumeur, P. N. and Mumford, D. (1992). A bayesian treatment of the stereo correspondence problem using half-occluded regions. In *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR'92., 1992 IEEE Computer Society Conference on*, pages 506–512. IEEE. (Cité en page 24.)

[Birchfield and Tomasi, 1999] Birchfield, S. and Tomasi, C. (1999). Depth discontinuities by pixel-to-pixel stereo. *International Journal of Computer Vision*, 35(3) :269–293. (Cité en page 24.)

[Bouguet, 2001] Bouguet, J.-Y. (2001). Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. *Intel Corporation*, 5(1-10) :4. (Cité en pages 40 et 49.)

[Boykov et al., 2001] Boykov, Y., Veksler, O., and Zabih, R. (2001). Fast approximate energy minimization via graph cuts. *IEEE Transactions on pattern analysis and machine intelligence*, 23(11) :1222–1239. (Cité en pages 22 et 23.)

[Bradski, 2000] Bradski, G. (2000). *Dr. Dobb's Journal of Software Tools*. (Cité en pages 30, 40 et 52.)

[Brown, 1966] Brown, D. C. (1966). Decentering distortion of lenses. *Photometric Engineering*, 32(3) :444–462. (Cité en page 9.)

[Brox et al., 2009] Brox, T., Bregler, C., and Malik, J. (2009). Large displacement optical flow. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 41–48. IEEE. (Cité en pages 46 et 72.)

[Brox et al., 2004] Brox, T., Bruhn, A., Papenberg, N., and Weickert, J. (2004). High accuracy optical flow estimation based on a theory for warping. In *European conference on computer vision*, pages 25–36. Springer. (Cité en pages 45, 63 et 72.)

[Bruhn and Weickert, 2006] Bruhn, A. and Weickert, J. (2006). A confidence measure for variational optic flow methods. In *Geometric Properties for Incomplete Data*, pages 283–298. Springer. (Cité en pages 90 et 91.)

[Butler et al., 2012] Butler, D. J., Wulff, J., Stanley, G. B., and Black, M. J. (2012). A naturalistic open source movie for optical flow evaluation. In *European Conference on Computer Vision*, pages 611–625. Springer. (Cité en page 85.)

[Chakrabarti et al., 2015] Chakrabarti, A., Xiong, Y., Gortler, S. J., and Zickler, T. (2015). Low-level vision by consensus in a spatial hierarchy of regions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4009–4017. (Cité en pages 27, 28, 30, 89 et 143.)

[Champagnat and Le Sant, 2012] Champagnat, F. and Le Sant, Y. (2012). Efficient cubic b-spline image interpolation on a gpu. *Journal of Graphics Tools*, 16(4) :218–232. (Cité en page 120.)

[Delon and Rougé, 2007] Delon, J. and Rougé, B. (2007). Small baseline stereovision. *Journal of Mathematical Imaging and Vision*, 28(3) :209–223. (Cité en page 16.)

[Duran, 2015] Duran, M. (2015). Marcel hirscher news : Crashing drone narrowly misses champion skier during race. http ://www.enstarz.com/articles/129346/20151223/marcel-hirscher-news-crashing-drone-narrowly-misses-champion-skier-during-race.htm. (Cité en page 2.)

[Faugeras, 1993] Faugeras, O. (1993). *Three-dimensional computer vision : a geometric viewpoint*. MIT press. (Cité en pages 87 et 94.)

[Fischler and Bolles, 1981] Fischler, M. A. and Bolles, R. C. (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. (Cité en pages 38, 39, 40 et 64.)

[Freeman et al., 2000] Freeman, W. T., Pasztor, E. C., and Carmichael, O. T. (2000). Learning low-level vision. *International journal of computer vision*, 40(1) :25–47. (Cité en page 24.)

[Fusiello and Irsara, 2008] Fusiello, A. and Irsara, L. (2008). Quasi-euclidean uncalibrated epipolar rectification. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4. IEEE. (Cité en page 11.)

[Fusiello et al., 1997] Fusiello, A., Roberto, V., and Trucco, E. (1997). Efficient stereo with multiple windowing. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 858–863. IEEE. (Cité en page 16.)

[Gehrig et al., 2009] Gehrig, S. K., Eberli, F., and Meyer, T. (2009). A Real-Time Low-Power Stereo Vision Engine Using Semi-Global Matching. In Fritz, M., Schiele, B., and Piater, J., editors, *Proceedings of the 7th International Conference on Computer Vision Systems : Computer Vision Systems*, volume 5815 of *Lecture Notes in Computer Science*, pages 134–143. Springer. (Cité en page 33.)

[Geiger et al., 2012] Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving ? the kitti vision benchmark suite. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3354–3361. IEEE. (Cité en pages 4, 29, 39, 51, 70, 85 et 105.)

[Geiger et al., 2010] Geiger, A., Roser, M., and Urtasun, R. (2010). Efficient large-scale stereo matching. In *Asian conference on computer vision*, pages 25–38. Springer. (Cité en pages 13, 20 et 30.)

[Geman and Geman, 1984] Geman, S. and Geman, D. (1984). Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6) :721–741. (Cité en page 22.)

[Hirschmuller, 2005] Hirschmuller, H. (2005). Accurate and efficient stereo processing by semi-global matching and mutual information. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 807–814. IEEE. (Cité en pages 24, 25, 30, 64 et 88.)

[Horn and Schunck, 1981] Horn, B. K. and Schunck, B. G. (1981). Determining optical flow. *Artificial intelligence*, 17(1-3) :185–203. (Cité en pages 41 et 44.)

[Hornung et al., 2013] Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W. (2013). Octomap : An efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34(3) :189–206. (Cité en page 122.)

[Huguet and Devernay, 2007] Huguet, F. and Devernay, F. (2007). A variational method for scene flow estimation from stereo sequences. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–7. IEEE. (Cité en pages 62, 63 et 64.)

[Juhan, 2013] Juhan, V. (2013). La voiture sans chauffeur de google : comment ça marche ?. http ://www.journaldunet.com/solutions/saas-logiciel/google-car-une-voiture-sans-chauffeur.shtml. (Cité en page 3.)

[Kalal et al., 2012] Kalal, Z., Mikolajczyk, K., and Matas, J. (2012). Tracking-learning-detection. *IEEE transactions on pattern analysis and machine intelligence*, 34(7) :1409–1422. (Cité en pages 138 et 145.)

[Le Besnerais and Champagnat, 2005] Le Besnerais, G. and Champagnat, F. (2005). Dense optical flow by iterative local window registration. In *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, volume 1, pages I–137. IEEE. (Cité en page 48.)

[Lucas et al., 1981] Lucas, B. D., Kanade, T., et al. (1981). An iterative image registration technique with an application to stereo vision. In *International Joint Conference on Artificial Intelligence*, pages 674–679. (Cité en pages 41 et 47.)

[Menze and Geiger, 2015] Menze, M. and Geiger, A. (2015). Object scene flow for autonomous vehicles. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3061–3070. (Cité en pages 64, 70, 85 et 123.)

[Min and Sohn, 2006] Min, D. and Sohn, K. (2006). Edge-preserving simultaneous joint motion-disparity estimation. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 2, pages 74–77. IEEE. (Cité en pages 62 et 63.)

[Okutomi and Kanade, 1992] Okutomi, M. and Kanade, T. (1992). A locally adaptive window for signal matching. *International Journal of Computer Vision*, 7(2) :143–162. (Cité en page 16.)

[Patras et al., 1997] Patras, I., Hendriks, E., and Tziritas, G. (1997). A joint motion/disparity estimation method for the construction of stereo interpolated images in stereoscopic image sequences. In *Proc. 3rd Annual Conference of the Advanced School for Computing and Imaging, Heijen, The Netherlands*. (Cité en pages 62 et 63.)

[Plyer et al., 2014] Plyer, A., Le Besnerais, G., and Champagnat, F. (2014). Massively parallel lucas kanade optical flow for real-time video processing applications. *Journal of Real-Time Image Processing*, 11(4) :713–730. (Cité en pages 41, 47, 48, 49, 53 et 118.)

[Quigley et al., 2009] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros : an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe. (Cité en pages 114 et 121.)

[Rabe et al., 2010] Rabe, C., Müller, T., Wedel, A., and Franke, U. (2010). Dense, robust, and accurate motion field estimation from stereo image sequences in real-time. In *European conference on computer vision*, pages 582–595. Springer. (Cité en pages 136, 137 et 145.)

[Ranftl et al., 2014] Ranftl, R., Bredies, K., and Pock, T. (2014). Non-local total generalized variation for optical flow estimation. In *European Conference on Computer Vision*, pages 439–454. Springer. (Cité en page 46.)

[Revaud et al., 2015] Revaud, J., Weinzaepfel, P., Harchaoui, Z., and Schmid, C. (2015). Epicflow : Edge-preserving interpolation of correspondences for optical flow. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1164–1172. (Cité en pages 43 et 67.)

[Romero-Cano and Nieto, 2013] Romero-Cano, V. and Nieto, J. I. (2013). Stereo-based motion detection and tracking from a moving platform. In *Intelligent Vehicles Symposium (IV), 2013 IEEE*, pages 499–504. IEEE. (Cité en pages 89, 90, 92, 97, 101 et 104.)

[Rosten and Drummond, 2006] Rosten, E. and Drummond, T. (2006). Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer. (Cité en page 39.)

[Sanfourche et al., 2014] Sanfourche, M., Plyer, A., Bernard-Brunel, A., and Le Besnerais, G. (2014). 3dscan : Online ego-localization and environment mapping for micro aerial vehicles. *AerospaceLab*, (8) :1–17. (Cité en page 123.)

[Sanfourche et al., 2013] Sanfourche, M., Vittori, V., and Le Besnerais, G. (2013). evo : A realtime embedded stereo odometry for mav applications. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 2107–2114. IEEE. (Cité en pages 39 et 70.)

[Scharstein and Szeliski, 2002] Scharstein, D. and Szeliski, R. (2002). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, 47(1-3) :7–42. (Cité en pages 4, 13, 26 et 27.)

[Shi et al., 1994] Shi, J. et al. (1994). Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 593–600. IEEE. (Cité en page 40.)

[Shimizu and Okutomi, 2001] Shimizu, M. and Okutomi, M. (2001). Precise sub-pixel estimation on area-based matching. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 1, pages 90–97. IEEE. (Cité en page 26.)

[Simoncelli, 1999] Simoncelli, E. P. (1999). 14 bayesian multi-scale differential optical flow. (Cité en pages 49, 55, 59 et 90.)

[Simoncelli et al., 1991] Simoncelli, E. P., Adelson, E. H., and Heeger, D. J. (1991). Probability distributions of optical flow. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on*, pages 310–315. IEEE. (Cité en page 90.)

[Sizintsev and Wildes, 2010] Sizintsev, M. and Wildes, R. P. (2010). Coarse-to-fine stereo vision with accurate 3d boundaries. *Image and Vision Computing*, 28(3) :352–366. (Cité en pages 14, 16, 19, 30, 70 et 143.)

[Sun et al., 2003] Sun, J., Zheng, N.-N., and Shum, H.-Y. (2003). Stereo matching using belief propagation. *IEEE Transactions on pattern analysis and machine intelligence*, 25(7) :787–800. (Cité en page 23.)

[Szeliski, 2010] Szeliski, R. (2010). *Computer vision : algorithms and applications*. Springer Science & Business Media. (Cité en page 87.)

[Talukder and Matthies, 2004] Talukder, A. and Matthies, L. (2004). Real-time detection of moving objects from moving vehicles using dense stereo and optical flow. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004*

IEEE/RSJ International Conference on, volume 4, pages 3718–3725. IEEE. (Cité en pages 91, 96, 102 et 103.)

[Tsai, 1986] Tsai, R. Y. (1986). An efficient and accurate camera calibration technique for 3d machine vision. pages 364–374. (Cité en page 10.)

[Umeyama, 1991] Umeyama, S. (1991). Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on pattern analysis and machine intelligence*, 13(4) :376–380. (Cité en pages 37 et 38.)

[Uras et al., 1988] Uras, S., Girosi, F., Verri, A., and Torre, V. (1988). A computational approach to motion perception. *Biological Cybernetics*, 60(2) :79–87. (Cité en page 90.)

[Vaudrey et al., 2011] Vaudrey, T., Morales, S., Wedel, A., and Klette, R. (2011). Generalised residual images' effect on illumination artifact removal for correspondence algorithms. *Pattern Recognition*, 44(9) :2034–2046. (Cité en page 42.)

[Vaudrey et al., 2008] Vaudrey, T., Rabe, C., Klette, R., and Milburn, J. (2008). Differences between stereo and motion behaviour on synthetic and real-world stereo sequences. In *Image and Vision Computing New Zealand, 2008. IVCNZ 2008. 23rd International Conference*, pages 1–6. IEEE. (Cité en page 85.)

[Vedula et al., 2005] Vedula, S., Rander, P., Collins, R., and Kanade, T. (2005). Three-dimensional scene flow. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(3) :475–480. (Cité en page 60.)

[Veksler, 2005] Veksler, O. (2005). Stereo correspondence by dynamic programming on a tree. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 384–390. IEEE. (Cité en pages 24 et 25.)

[Veksler et al., 2010] Veksler, O., Boykov, Y., and Mehrani, P. (2010). Superpixels and supervoxels in an energy optimization framework. In *European conference on Computer vision*, pages 211–224. Springer. (Cité en page 64.)

[Vogel et al., 2014] Vogel, C., Roth, S., and Schindler, K. (2014). View-consistent 3d scene flow estimation over multiple frames. In *European Conference on Computer Vision*, pages 263–278. Springer. (Cité en page 64.)

[Vogel et al., 2013] Vogel, C., Schindler, K., and Roth, S. (2013). Piecewise rigid scene flow. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1377–1384. (Cité en page 64.)

[Vogel et al., 2015] Vogel, C., Schindler, K., and Roth, S. (2015). 3d scene flow estimation with a piecewise rigid scene model. *International Journal of Computer Vision*, 115(1) :1–28. (Cité en page 64.)

[Vondrick et al., 2013] Vondrick, C., Patterson, D., and Ramanan, D. (2013). Efficiently scaling up crowdsourced video annotation. *International Journal of Computer Vision*, 101(1) :184–204. (Cité en page 105.)

[Wedel et al., 2011] Wedel, A., Brox, T., Vaudrey, T., Rabe, C., Franke, U., and Cremers, D. (2011). Stereoscopic scene flow computation for 3d motion understan-

ding. *International Journal of Computer Vision*, 95(1) :29–51. (Cité en pages 63, 65, 77, 87, 88, 91, 92, 96 et 102.)

[Wedel et al., 2009] Wedel, A., Meißner, A., Rabe, C., Franke, U., and Cremers, D. (2009). Detection and segmentation of independently moving objects from dense scene flow. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 14–27. Springer. (Cité en page 102.)

[Weinzaepfel et al., 2013] Weinzaepfel, P., Revaud, J., Harchaoui, Z., and Schmid, C. (2013). Deepflow : Large displacement optical flow with deep matching. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1385–1392. (Cité en pages 43 et 46.)

[www.dronotec.com, 2015] www.dronotec.com (2015). Dronotec. (Cité en page 1.)

[www.leparisien.fr, 2014] www.leparisien.fr (2014). Parrot présente son "bebop drone", piloté à longue distance. http ://www.leparisien.fr/high-tech/parrot-presente-son-bebop-drone-pilote-a-longue-distance-07-11-2014-4276151.php. (Cité en page 1.)

[www.theguardian.com, 2016] www.theguardian.com (2016). Tesla driver dies in first fatal crash while using autopilot mode. https ://www.theguardian.com/technology/2016/jun/30/tesla-autopilot-death-self-driving-car-elon-musk. (Cité en page 3.)

[Yacou, 2013] Yacou, Y. (2013). Le drone, nouvel outil de maintenance d'edf guadeloupe. http ://la1ere.francetvinfo.fr/guadeloupe/2013/05/24/le-drone-nouvel-outil-de-maintenance-d-edf-guadeloupe-37092.html. (Cité en page 1.)

[Yang et al., 2010] Yang, Q., Wang, L., and Ahuja, N. (2010). A constant-space belief propagation algorithm for stereo matching. In *Computer vision and pattern recognition (CVPR), 2010 IEEE Conference on*, pages 1458–1465. IEEE. (Cité en page 30.)

[Zabih and Woodfill, 1994] Zabih, R. and Woodfill, J. (1994). Non-parametric local transforms for computing visual correspondence. In *European conference on computer vision*, pages 151–158. Springer. (Cité en pages 16, 42 et 54.)

[Zach et al., 2007] Zach, C., Pock, T., and Bischof, H. (2007). A duality based approach for realtime tv-l 1 optical flow. In *Joint Pattern Recognition Symposium*, pages 214–223. Springer. (Cité en pages 45 et 72.)

[Zille et al., 2014] Zille, P., Corpetti, T., Shao, L., and Chen, X. (2014). Observation model based on scale interactions for optical flow estimation. *IEEE Transactions on Image Processing*, 23(8) :3281–3293. (Cité en page 67.)