Université de Montréal

Méthodes d'analyse de mouvement en vision 3D : Invariance aux délais temporels entre des caméras non synchronisées et flux optique par isocontours

par

Rania Benrhaiem

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures
en vue de l'obtention du grade de
Philosophiae Doctor (Ph.D.)
en informatique

août, 2016

RÉSUMÉ

Cette thèse porte sur deux sujets de vision par ordinateur axés sur l'analyse de mouvement dans une scène dynamique vue par une ou plusieurs caméras.

En premier lieu, nous avons travaillé sur le problème de la capture de mouvement avec des caméras non synchronisées. Ceci entraîne généralement des erreurs de mise en correspondance 2D et par la suite des erreurs de reconstruction 3D. En contraste avec les solutions matérielles déjà existantes qui essaient de minimiser voire annuler le délai temporel entre les caméras, nous avons proposé une solution qui assure une invariance aux délais. En d'autres termes, nous avons développé une méthode qui permet de trouver la bonne mise en correspondance entre les points à reconstruire indépendamment du délai temporel.

En second lieu, nous nous sommes intéressés au problème du flux optique avec une approche différente des méthodes proposées dans l'état de l'art. Le flux optique est utilisé pour l'analyse de mouvement en temps réel. Il est donc important qu'il soit calculé rapidement. Généralement, les méthodes existantes de flux optique sont classées en deux principales catégories : ou bien à la fois denses et précises mais très exigeantes en calcul, ou bien rapides mais moins denses et moins précises. Nous avons proposé une alternative qui tient compte à la fois du temps de calcul et de la précision du résultat. Nous avons proposé d'utiliser les isocontours d'intensité et de les mettre en correspondance afin de retrouver le flux optique en question.

Ces travaux ont amené à deux principales contributions intégrées dans les chapitres de la thèse.

Mots clés: synchronisation de caméras, délai temporel, géométrie épipolaire, reconstruction 3D, invariance aux délais temporels, flux optique, isocontours.

ABSTRACT

In this thesis we focused on two computer vision subjects. Both of them concern motion analysis in a dynamic scene seen by one or more cameras.

The first subject concerns motion capture using unsynchronised cameras. This causes many correspondence errors and 3D reconstruction errors. In contrast with existing material solutions trying to minimize the temporal delay between the cameras, we propose a software solution ensuring an invariance to the existing temporal delay. We developed a method that finds the good correspondence between points regardless of the temporal delay. It solves the resulting spatial shift and finds the correct position of the shifted points.

In the second subject, we focused on the optical flow problem using a different approach than the ones in the state of the art. In most applications, optical flow is used for real-time motion analysis. It is then important to be performed in a reduced time. In general, existing optical flow methods are classified into two main categories : either precise and dense but computationally intensive, or fast but less precise and less dense. In this work, we propose an alternative solution being at the same time, fast and precise. To do this, we propose extracting intensity isocontours to find corresponding points representing the related optical flow.

By addressing these problems we made two major contributions.

Keywords : camera synchronisation, temporal delay, epipolar geometry, 3D reconstruction, temporal delay invariance, optical flow, isocontours.

TABLE DES MATIÈRES

LISTE DES FIGURES

REMERCIEMENTS

Je tiens à remercier mon co-directeur Sébastien Roy pour m'avoir permis de rejoindre le laboratoire de vision 3D et aussi pour tout ce qu'il m'a apporté durant ma thèse. Je le remercie pour son énergie, son enthousiasme et aussi pour toutes les discussions qu'on a eu depuis le début. Ceci m'a donné plus de confiance et m'a permis d'explorer plusieurs chemins.

Je tiens à remercier également mon directeur de recherche Jean Meunier pour m'avoir donné l'opprtunité de faire un doctorat, pour m'avoir laissé la liberté de travailler sur le sujet de thèse de mon choix et pour tout le temps mis pour la correction de mes articles et de ma thèse.

Je remercie particulièrement mon mari pour son soutien continu, ses conseils et ses encouragements.

Je remercie finalement ma famille et particulièrement mes parents pour leur soutien qui, malgré la distance, m'a toujours donné la force de continuer.

INTRODUCTION

Dans cette thèse nous avons travaillé sur deux sujets de vision par ordinateur qui traitent l'analyse de mouvement dans une scène vue par une ou plusieurs caméras. Le premier sujet porte sur la reconstruction 3D d'une scène dynamique à partir de deux (ou plusieurs) caméras. Dans ce travail, la problématique posée est : comment corriger les erreurs de reconstruction causées par deux (ou plusieurs) vues temporellement décalées capturées par des caméras non synchronisées ? La grande majorité des solutions pour les caméras non synchronisées est matérielle. Les applications se servant de plusieurs caméras utilisent souvent des déclencheurs (*triggers*) qui envoient simultanément un signal à toutes les caméras pour commencer la capture. Le problème de ce genre de solution est que le coût de l'équipement augmente proportionnellement avec la précision voulue. Plus le délai visé entre les caméras est petit, plus le coût du matériel est élevé. Ceci complique l'utilisation et l'installation des solutions efficaces pour certains projets à petit budget. En contre partie, il existe très peu de solutions logicielles efficaces qui traitent ce problème. D'ailleurs, celles qui existent s'intéressent à aligner des séquences déjà prises. Elles ne sont pas adaptées à des applications temps réel. Nous proposons une solution logicielle qui corrige cette erreur en temps réel en se basant sur la géométrie épipolaire des caméras et sur la trajectoire du mouvement des points suivis.

Le deuxième sujet concerne l'analyse de mouvement dans des images 2D par l'étude du flux optique. Le flux optique, qui représente le mouvement apparent des objets dans une image, est dans certains cas obtenu suite au mouvement de la caméra qui capture la scène. Dans ce cas, il peut fournir plusieurs informations sur le mouvement de la caméra. Certains algorithmes utilisent aussi le flux optique pour détecter des obstacles dans la scène. Le début de ce domaine est marqué par l'utilisation de méthodes différentielles qui sont

basées sur certaines contraintes (constance de la luminosité, contrainte de régularité, etc.) limitant l'efficacité de la performance dans certaines situations (performent mieux pour des petits mouvements, fonctionnent moins bien dans les discontinuités). Depuis, plusieurs méthodes ont été introduites. La performance de ces méthodes est souvent proportionnelle à leur complexité. Souvent, les algorithmes qui fonctionnent bien sont très exigents en terme de calcul et *vice-versa*. Notre méthode détermine le flux optique en se basant sur l'extraction et la mise en correspondance des isocontours d'intensité dans les deux images. Le choix d'utiliser les isocontours part du principe qu'ils décrivent tous les pixels ayant la même intensité et se plaçant sur une rampe d'intensités par rapport à leurs voisins (c'est souvent le cas sauf si l'isocontour est un minimum local ou un maximum local). Contrairement aux contours normaux qui décrivent un changement brusque d'intensité où échoue la majorité des algorithmes du flux optique, les isocontours se placent souvent dans des régions ayant des variations d'intensité plus faibles ce qui rend la recherche du flux plus facile. L'avantage de notre méthode est de trouver un flux optique robuste dans un temps de calcul considérablement plus court.

Organisation de la thèse

Notre thèse est présentée en deux parties détaillant nos contributions dans le domaine de la vision par ordinateur. Chaque partie est composée de deux chapitres.

La première partie concerne la synchronisation de deux caméras pour obtenir une reconstruction 3D avec un minimum d'erreurs (on parle d'un modèle à deux caméras pour des raisons de simplicité mais ceci peut être généralisé à plusieurs caméras). Le premier chapitre est une introduction aux notions préliminaires du calibrage des caméras, la reconstruction 3D, etc. qui seront utilisées dans le deuxième chapitre. Celui-ci inclut l'article qui présente ces travaux publiés dans le journal *Machine Vision and Applications*.

La deuxième partie porte sur l'aspect d'analyse de la scène qui consiste à calculer le flux optique. Elle est composée de trois chapitres. Le troisième chapitre présente les notions

de bases du flux optique. Il explique ses origines en explorant les premiers travaux. Il présente aussi un survol sur l'état de l'art de ce domaine. Il introduit également le quatrième chapitre qui présente notre contribution sous forme d'un papier en préparation. Ce chapitre est conclu par une discussion qui explique les modifications apportées à la version initiale et appuyées par de nouveaux résultats.

En guise de conclusion, nous présentons les perspectives de nos travaux ainsi que les travaux qui sont présentement en cours de développement portant encore sur le flux optique.

Première partie

Vers une invariance au délai temporel entre les caméras

Chapitre 1

CAPTURE MULTI-CAMÉRAS

1.1 Géométrie de la caméra

Une image est la représentation d'une scène du monde 3D projetée vers un monde 2D par une caméra. La projection obtenue suit principalement un modèle mathématique dont l'estimation est un aspect important de la vision par ordinateur. Les modèles mathématiques représentant les caméras ont des degrés de complexité variables. Les modèles de caméras les plus communément utilisés sont basés sur une géométrie projective. Ceux-ci sont constitués d'un ensemble de matrices ayant des propriétés algébriques spécifiques qui définissent la transformation de la scène 3D vers l'image 2D. Parmi les caméras ayant une géométrie projective, on distingue les caméras avec un centre optique fini (caméra perspective), et les caméras avec un centre à l'infini (exemple : caméra orthographique). Les travaux de notre thèse sont axés sur le modèle de caméra perspective. Dans ce cas, le modèle le plus simple à étudier est la caméra sténopé. Ce modèle est illustré à la figure 1.1. Il est principalement composé d'un plan image (qui représente le senseur CCD ou CMOS dans une caméra réelle) et d'un centre de la caméra c (ou centre optique de l'objectif).

La projection d'un point 3D du monde p_w sur le plan image n'est que l'intersection de ce plan avec la ligne 3D qui relie le point du monde p_w avec le centre de la caméra c. c est placé à l'origine du système de coordonnées de monde. Le plan image est parallèle au plan $x_w y_w$, et placé devant le centre de la caméra à une distance f. Le point $p_w = (x_w, y_w, z_w)^T$ vu par une caméra sténopé, est projeté au point $p = (fx_w/z_w, fy_w/z_w, f)^T$. Si on ne considère pas la profondeur f du plan de l'image, la projection de $p_w = (x_w, y_w, z_w)^T$ est $p = (fx_w/z_w, fy_w/z_w)^T$.

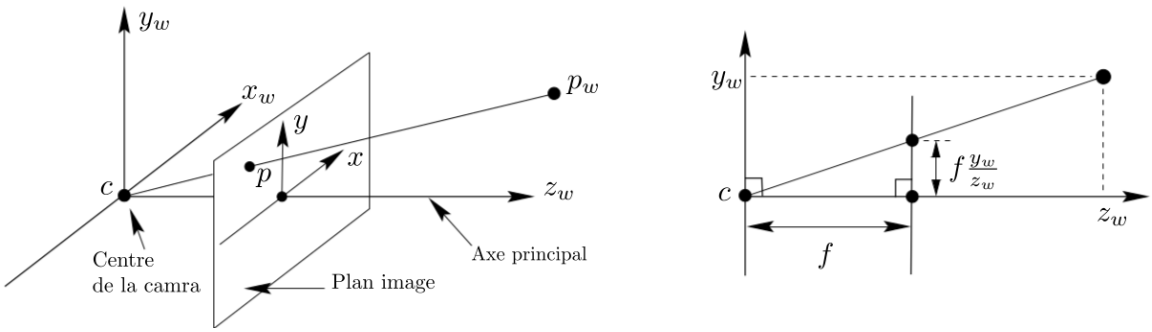


FIGURE **1.1. Modèle de caméra sténopé. Tiré de [38] (notations modifiées).**

Coordonnées homogènes

Le système euclidien représente un point dans $\mathbb{R}^n$ par un vecteur de longueur n. Dans le système de coordonnées homogènes on utilise pour le même point un vecteur de longueur $n+1$. Ainsi, les coordonnées homogènes "sur-dimensionnent" l'espace et représentent le point dans l'espace projectif. Le passage du système homogène vers le système euclidien (et vice-versa) est donné par

$$\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \\ w \end{pmatrix} \leftrightarrow \begin{pmatrix} x_1/w \\ x_2/w \\ \dots \\ x_n/w \end{pmatrix}. \tag{1.1}$$

Un point p dans l'espace euclidien est noté $\tilde{p}$ dans l'espace de coordonnées homogènes. Ce qui nous intéresse le plus dans les coordonnées homogènes est qu'elles simplifient certaines opérations telles que la translation. Les coordonnées homogènes offrent également une représentation de l'infini plus simple (les points et les lignes à l'infini). Dans la représentation homogène, les transformations linéaires sont représentées sous la forme de multiplications matricielles. Si on réécrit la projection du point X dans la caméra sténopé sous la forme

matricielle, on obtient :

$$\begin{pmatrix} fx_w \\ fy_w \\ z_w \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_w \\ y_w \\ z_w \\ 1 \end{pmatrix}. \tag{1.2}$$

La matrice introduite dans l'équation précédente est la matrice de projection de la caméra. Elle est homogène et de dimensions 3×4. L'équation 1.2 pourrait être réécrite

$$\tilde{p} = P\tilde{p}_w \tag{1.3}$$

où $\tilde{p}$ est le point projeté sur le plan image dans le système de coordonnées de la caméra (en coordonnées homogènes), $\tilde{p}_w$ est le point dans le monde et P est la matrice de la caméra sténopé (ou la matrice de projection). Parmi les transformations appliquées à un point (ou à une ligne) on trouve la translation, la rotation, la mise en échelle, la transformation projective (ou homographie), etc. Toutes ces informations sont linéaires dans l'espace projectif. Chaque transformation a un nombre de degrés de liberté qui définit les propriétés géométriques qui vont changer. Ces transformations sont détaillées à la figure 1.2 et au tableau 1.1 [67].

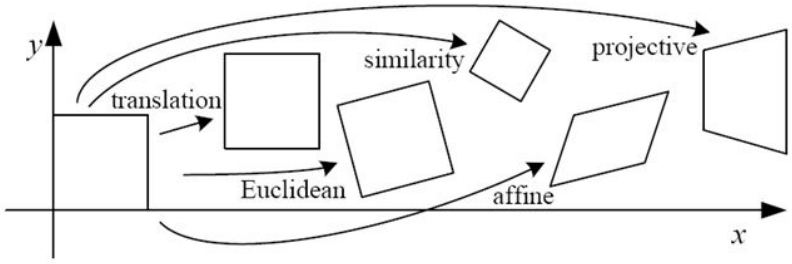


FIGURE **1.2. L'ensemble des transformations 2D. Tiré de [67].**

Transformation	Matrice	# D. de. L.	Préserve	Icone
translation	$[I\|t]_{2\times3}$	2	orientation + ...	
rigide (euclidien)	$[R\|t]_{2\times3}$	3	longueurs + ...	
similarité	$[sR\|t]_{2\times3}$	4	angles + ...	
affine	$[A]_{2\times3}$	6	parallélisme + ...	
projective	$[\tilde{H}]_{2\times3}$	8	lignes droites + ...	

TABLE **1.1. Hiérarchie des transformations 2D. Chaque transformation préserve les propriétés listées au dessus. La transformation ayant le plus grand degré de liberté est la transformation projective (homographie). Tiré de [67] (traduction libre).**

Modèle général de la caméra projective

À partir du modèle simplifié, il est important d'intégrer la pose de la caméra au modèle. La pose de la caméra définit sa rotation et sa translation par rapport au système de coordonnées du monde. Ces informations permettent de déterminer la matrice de projection de la caméra. On définit ainsi R comme étant une matrice 3×3 de rotation de la caméra et t un vecteur de translation de la caméra. On définit également p_{cam} comme étant le point p_w après avoir subi la rotation et la translation. p_{cam} est obtenu comme suit :

$$\tilde{p}_{cam} = \left(\begin{array}{ccc|c} & R & & Rt \\ 0 & 0 & 0 & 1 \end{array}\right) \tilde{p}_w. \tag{1.4}$$

On peut donc généraliser l'équation 1.2 en intégrant la pose de la caméra. La matrice P peut alors être définie comme suit :

$$P = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \left(\begin{array}{c|c} R & Rt \end{array}\right). \tag{1.5}$$

Soit K la matrice gauche dans l'équation précédente. Dans un modèle généralisé de la caméra projective, il faut prendre en considération d'autres paramètres dans cette matrice.

La distance f étant définie comme la distance par rapport au centre optique ou au centre de la caméra (appelée aussi la distance focale), elle n'est pas toujours la même en terme de nombre de pixels (autrement dit dans le système de coordonnées de l'image) par rapport aux axes x et y. Il faut plutôt définir f_x et f_y comme les distances focales selon les axes x et y respectivement. Dans la très grande majorité des cas f_x et f_y sont égaux mais il existe des caméras ayant des pixels non carrés. Dans le modèle simplifié, on suppose que le centre du plan image est son origine. C'est ainsi qu'on définit l'équation 1.2 sans introduire les coordonnées du centre du plan image (on suppose que $c = (c_x, c_y)^T = (0,0)^T$). Ceci n'est pas toujours le cas dans le modèle généralisé. Il est donc pratique d'introduire ces coordonnées dans la matrice K. Finalement, il est possible mais très rare que les axes x et y de l'image ne soient pas perpendiculaires entre eux. Dans cette situation, il faut introduire la variable s dans la matrice K (pour des axes perpendiculaires entre eux, $s = 0$). On obtient finalement

$$K = \begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}. \tag{1.6}$$

On définit K comme étant la matrice interne de la caméra. La matrice P, appelée la matrice de projection dans le modèle généralisé de la caméra projective peut donc être réécrite comme

$$P = K \left[\; R \;\middle|\; Rt \; \right]. \tag{1.7}$$

En coordonnées homogènes, P prend la forme

$$P = \begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 1 & 1 \end{pmatrix} \tag{1.8}$$

avec $r_{11}, ..., r_{33}$ sont les paramètres de la matrice de rotation R et $t = (t_x, t_y \text{ et } t_z)^T$ sont les paramètres du vecteur de translation t. Nous utiliserons le modèle de caméra projective dans la suite de nos travaux.

1.2 Calibrage, estimation de la pose et reconstruction 3D

Le calibrage est l'opération permettant de retrouver les paramètres internes de la caméra (la matrice K et les coefficients de la distorsion radiale). La distorsion radiale est une aberration géométrique causée par la position du diaphragme par rapport à la lentille. La distorsion de *Seidel* est la plus facile à reconnaître car elle déforme une grande partie de l'image. La déformation est visible surtout sur les lignes droites. Il existe plusieurs types de distorsions radiales comme la distorsion en barillet (*barrel*) et la distorsion en coussinet (*pincushion*) (voir la figure 1.3(a)). Elle est causée par la position du diaphragme qui est placé parfois en avant ou en arrière du centre optique (voir la figure 1.3(b)) [1].

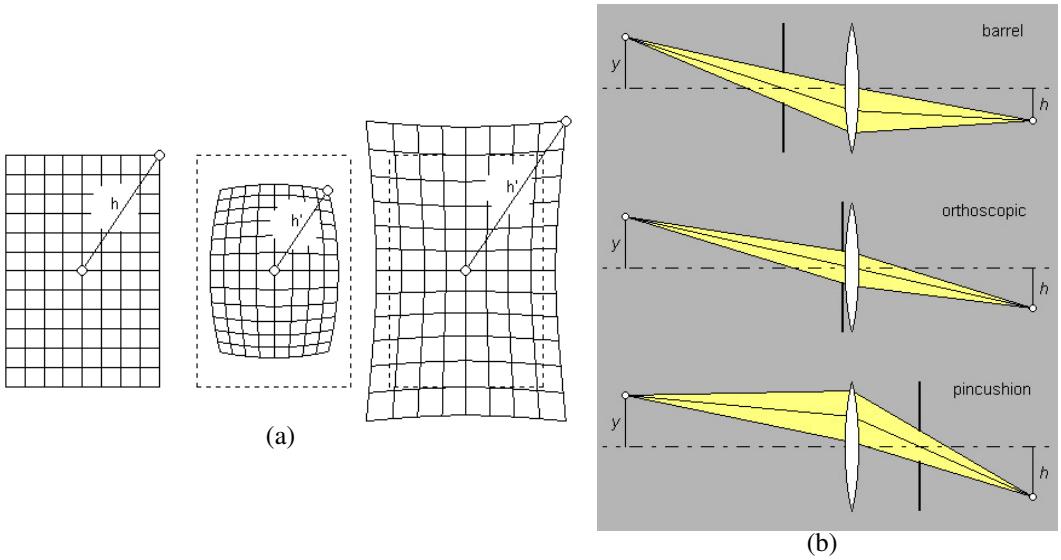


FIGURE **1.3. (a) Distorsion d'une grille rectangulaire. À gauche : grille non déformée. Au milieu : distorsion en barillet. À droite : distorsion en coussinet. (b) Origines de la distorsion radiale : position du diaphragme par rapport à la lentille [1].**

L'estimation de la pose quant à elle permet de déterminer l'orientation de la caméra définie par la matrice R et la translation t. Soit p_w un point dans le monde et p_{cam} son image dans le système de coordonnées de la caméra. La relation entre ces deux points est

donnée par

$$p_{cam} = R(p_w - c) \tag{1.9}$$

où c est le centre de la caméra dans le monde. L'équation 1.9 peut être réécrite en coordonnées homogènes comme suit

$$\tilde{p}_{cam} = \begin{bmatrix} R & -Rc \\ 0 & 1 \end{bmatrix} \begin{pmatrix} x_w \\ y_w \\ z_w \\ 1 \end{pmatrix} = \begin{bmatrix} R & -Rc \\ 0 & 1 \end{bmatrix} \tilde{p}_w. \tag{1.10}$$

Dans cette équation, la translation t n'est pas explicitement exprimée. En réalité t est une fonction de R et c définie à partir de l'équation 1.10 comme suit

$$p_{cam} = Rp_w - Rc = Rp_w + t \tag{1.11}$$

ainsi $t = -Rc$. R et t sont appelés les paramètres externes de la caméra. Pour la plupart des caméras, la technique de calibrage la plus souvent utilisée est le calibrage planaire introduit par Zhang [81].

Pour retrouver la position 3D de chaque point dans le monde vu par les deux caméras, il faut d'abord calibrer les caméras et procéder par la suite à la triangulation. Le calibrage sert à retrouver les paramètres internes et externes des caméras. Il existe plusieurs types de calibrage : le calibrage planaire, le calibrage avec un objet 3D, le calibrage par rotation pure, etc. Dans notre cas, nous utilisons un calibrage planaire. Il consiste à utiliser une grille de calibrage (un damier). À la première étape, il faut attribuer des coordonnées aux points saillants (les coins des carreaux du damier) et ce, pour chaque damier. Généralement, on fixe l'origine du monde sur l'un des coins et le plan du damier à $z_w = 0$. La projection du

plan en $z_w = 0$ dans l'image de la caméra est donnée par l'équation 1.12

$$KR[I|-c]_{3\times 3}\begin{pmatrix} x_w \\ y_w \\ 0 \\ 1 \end{pmatrix} = KR\begin{bmatrix} 1 & 0 & \\ 0 & 1 & -c \\ 0 & 0 & \end{bmatrix}\begin{pmatrix} x_w \\ y_w \\ 1 \end{pmatrix} = H\begin{pmatrix} x_w \\ y_w \\ 1 \end{pmatrix}. \tag{1.12}$$

où x_w et y_w sont les coordonnées dans le monde d'un point saillant sur le damier (sachant que $z_w = 0$). Ainsi, chaque damier est muni d'une homographie H. L'homographie, définie par la matrice H, s'applique seulement dans le cas d'une rotation pure de la caméra ou lorsque la scène est planaire. Chaque homographie contient, entre autres, la matrice des paramètres internes K. En exploitant quelques propriétés algébriques (comme l'orthogonalité de la matrice de rotation R ainsi que l'image de la conique absolue (*IAC*)), nous arrivons à décomposer chaque homographie H sous la forme indiquée par l'équation 1.13

$$H = KR(I|-c), \tag{1.13}$$

où I est la matrice identité et c désigne la position de la caméra. Une fois tous ces paramètres récupérés, il faut vérifier l'exactitude de ces mesures. En fait, l'étape de vérification est très importante car plus le calibrage est précis, plus les opérations qui viennent après le seront. Dans notre cas, la vérification de la précision du calibrage consiste à trouver, pour chaque point du damier vu par les deux caméras, sa position 3D dans le monde, le reprojeter sur chaque image et calculer l'erreur en pixels.

La reconstruction d'un point 3D à partir de deux vues se réduit à une résolution linéaire d'un système d'équations. Pour trouver les coordonnées (x_w, y_w, z_w) d'un point p_w dans le monde, il faut au minimum deux matrices de projection représentant deux caméras et donc, deux vues (une matrice de projection par caméra). Une matrice de projection est le produit des matrices des paramètres internes et externes d'une caméra. Pour un point 2D donné p , de coordonnées (x, y) (en pixels), et une matrice de projection P_i où, i est l'indice de la

caméra, nous obtenons un système de deux équations à trois inconnues (comme indiqué dans l'équation 1.15), dérivé de l'équation 1.14

$$\tilde{p} \propto P_i \tilde{p}_w \tag{1.14}$$

où $\tilde{p} = (x, y, w)$ et $\tilde{p}_w = (x_w, y_w, z_w, w)$ représentent respectivement p et p_w en coordonnées homogènes et w représente un facteur d'échelle dans l'espace projectif.

$$\begin{aligned} x_j &= \frac{p_{j00}x_w + p_{j01}y_w + p_{j02}z_w + p_{j03}w}{p_{j20}x_w + p_{j21}y_w + p_{j22}z_w + p_{j23}} \\ y_j &= \frac{p_{j10}x_w + p_{j11}y_w + p_{j12}z_w + p_{j13}w}{p_{j20}x_w + p_{j21}y_w + p_{j22}z_w + p_{j23}}. \end{aligned} \tag{1.15}$$

C'est pourquoi il nous faudra au minimum un deuxième système de deux équations pour résoudre le système d'équations et trouver les coordonnées du point 3D.

1.3 Géométrie épipolaire

La géométrie épipolaire est une géométrie projective interne qui décrit la relation entre deux vues (voir la figure 1.4). Elle dépend essentiellement des paramètres internes et des poses des deux caméras. La géométrie épipolaire s'adresse principalement au problème de la correspondance des points dans deux vues différentes. Pour p_w un point dans le monde et p_1 son image dans la caméra 1, quel est le correspondant de p_1 dans la caméra 2 ? Le problème de la mise en correspondance a plusieurs utilisations en vision comme le stéréo, la reconstruction 3D, etc. Algébriquement, la transformation qui projette un point p_1 vers p_2 est une homographie. Elle décrit la transformation qu'ont subi les points du damier lorsqu'ils sont projetés dans l'image. La matrice H est une matrice 3×3 donnée comme

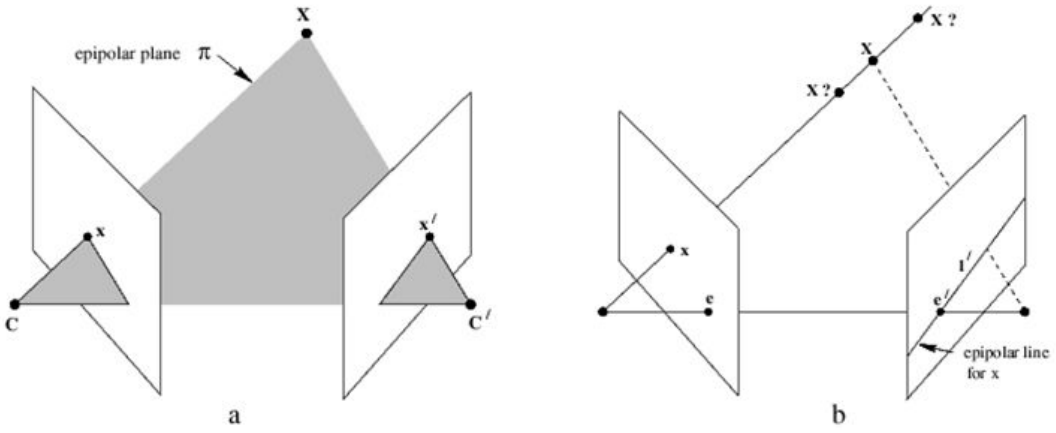


FIGURE **1.4. La géométrie épipolaire d'un point 3D X dans la scène. x_1 et x_2 sont les images respectives de X dans la caméra 1 et la caméra 2. C et C' sont les centres des deux caméras. Le *plan épipolaire* π est formé par les trois points (X,C,C'). π intersecte la caméra 2 en une ligne l' appelée *la ligne épipolaire*. La ligne qui relie les deux centres de caméras C et C' intersecte les plans images dans deux points respectifs e et e' appelés les *épipoles*. La géométrie épipolaire permet de déterminer la ligne épipolaire sur laquelle le point x' doit être. X, C, C', x, x', e' et l' correspondent respectivement à p_w, c_1, c_2, p_1, p_2, e_2 et l_2 dans la notation que nous avons définie dans notre document. Tiré de [67] (traduction libre).**

suit

$$H = \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{pmatrix}. \tag{1.16}$$

La relation entre deux points 2D $\tilde{p}_1$ et $\tilde{p}_2$ est

$$H\tilde{p}_1 \propto \tilde{p}_2, \tag{1.17}$$

où $\propto$ exprime le facteur d'échelle dans les coordonnées homogènes. Dans le cas d'une scène arbitraire (non planaire et la caméra n'a pas fait une rotation pure), l'homographie

ne s'applique plus. Dans ce cas, la géométrie épipolaire définit mieux la relation entre les points 2D à travers la matrice fondamentale F. En vision stéréo, la matrice fondamentale F établit une relation entre des points correspondants. F associe à chaque point p_1 dans la première image, une ligne épipolaire l_2 dans la deuxième image. p_2 qui correspond à p_1 dans la deuxième image est automatiquement sur l_2. Puisque p_w est vu par les deux caméras, il existe une homographie H_π (définie à partir des points qui sont dans le plan π) qui transforme p_1 dans la caméra 1 vers p_2 dans la caméra 2. La relation est donnée par $\tilde{p}_2 = H_\pi \tilde{p}_1$. La ligne épipolaire l_2 peut être définie par $l_2 = e_2 \times p_2 = [e_2]_\times p_2$ où e_2 est l'épipole qui représente l'image du centre de la caméra 1 c_1 dans l'image 2 (voir la figure 1.4). $[e']_\times$ est définie dans l'annexe A (pour plus de détails voir aussi [38]). Les deux relations précédentes impliquent

$$l' = \left[e'\right]_\times H_\pi x = Fx \tag{1.18}$$

où $F = [e']_\times H_\pi$ est la matrice fondamentale. Ainsi, F satisfait la relation

$$x'^T F x = 0 \tag{1.19}$$

pour chaque couple de points correspondants x et x'. Comme on l'a déjà mentionné, F dépend des paramètres internes et externes des deux caméras. On pourrait alors l'obtenir si les deux caméras sont calibrées. Il est aussi possible d'estimer F autrement si on ne connaît pas les paramètres internes et externes des caméras (elles ne sont pas calibrées). L'idée part de l'équation 1.19 qui représente une des propriétés principales de F. Si on a suffisamment de paires de points correspondants x_i et x'_i respectivement dans l'image 1 et 2, on pourrait retrouver F (voir annexe A).

1.4 Captation d'une scène dynamique et synchronisation des caméras

La captation stéréo d'une scène dynamique exige que les caméras soient parfaitement synchronisées. Dans le cas contraire, les points seraient capturés à des instants différents

et donc à des positions différentes. L'erreur de la reconstruction 3D serait plus grande. La synchronisation assure que les points suivis soient reconstruits au même instant.

La synchronisation des caméras pourrait se faire sur différents niveaux du traitement des vidéos. On peut distinguer deux principales catégories de synchronisation, à savoir : la synchronisation matérielle qui concerne les caméras et le réseau qui connecte les caméras et sur lequel circulent les vidéos, et la synchronisation logicielle au niveau du traitement des vidéos.

1.4.1 Synchronisation matérielle

FIGURE **1.5. Partie d'un ensemble de** 120 **caméras** *Canon EOS 30D* **contrôlées par cinq ordinateurs à l'aide de** *DSLR Remote Pro Multi-Camera* **[2].**

La synchronisation au moment de la capture est typiquement assurée par des outils matériels appelés déclencheurs (*triggers*). Un déclencheur est un micro-contrôleur qui envoie un signal pour synchroniser toutes les caméras connectées dessus. Les déclencheurs représentent une solution très efficace, mais qui exige que les caméras soient adaptées et équipées pour recevoir ce genre de signal. Cette solution ajoute une complexité d'installation et un coût supplémentaire au système d'acquisition. Parmi ces solutions matérielles, il y a ceux qui sont contrôlés à distance à travers des réseaux sans fils. Ceci les rend trop

dépendants des variations du réseau et donc moins fiables. À titre d'exemple, *Point Grey* a développé une unité de synchronisation appelée *Sync Unit* [5]. Elle est destinée à la synchronisation des caméras *Point Grey* connectées à un ou plusieurs ordinateurs. *Breeze Systems* a développé un système qui contrôle plusieurs caméras *Canon DSLR* (jusqu'à 25 caméras controlées par un seul ordinateur) (voir la figure 1.5) [2]. La synchronisation est assurée soit en les connectant au même déclencheur (un délai de 1 ms à 5 ms entre elles) ou en utilisant le logiciel fourni pour lancer la capture simultanément chez toutes les caméras (jusqu'à 3 secondes approximativement).

En général, ces solutions n'assurent que la synchronisation de l'instant initial de la capture. Si les caméras ont des fréquences d'acquisition variables, le délai entre les caméras va apparaître.

1.4.2 Synchronisation logicielle

La deuxième approche de la synchronisation ne se fait pas au niveau de la capture mais plutôt après l'acquisition. Lorsqu'on dispose de deux séquences vidéo (ou plus), il existe plusieurs façons de les réaligner temporellement pour synchroniser la vidéo. Les solutions les plus simples utilisent un signal lumineux visible partout pour marquer l'instant de début de capture dans toutes les séquences. Le nombre d'images d'écart entre les séquences détermine le délai entre elles. Certaines méthodes ont utilisé la géométrie épipolaire pour résoudre ce problème. Comme mentionné dans la section 1.3, la géométrie épipolaire définit une correspondance entre deux vues. Un point dans la première image correspond à une ligne dans la deuxième image et vice-versa. Comme cette information ne dépend pas de la synchronisation temporelle entre les images, il est possible de l'utiliser pour déduire de potentielles correspondances et corriger les positions des points décalés à cause du délai temporel. Dans ce contexte, Sinha *et al.* [59] ont proposé une approche qui synchronise plusieurs séquences vidéo provenant de caméras non synchronisées en calculant la géométrie épipolaire à partir des silhouettes dynamiques.

La méthode que nous présentons au chapitre 2 se situe dans cette catégorie d'approches. Nous utilisons la géométrie épipolaire pour corriger les positions des points décalés à cause d'un délai temporel.

Chapitre 2

ACHIEVING INVARIANCE TO THE TEMPORAL OFFSET OF UNSYNCHRONIZED CAMERAS THROUGH EPIPOLAR POINT-LINE TRIANGULATION

Ce chapitre présente l'article [19] publié tel qu'indiqué dans la référence bibliographique ci-dessous : *Rania Benrhaiem, Sébastien Roy et Jean Meunier. Achieving invariance to the temporal offset of unsynchronized cameras through epipolar point-line triangulation. Machine Vision and Applications, 27(4) : p. 545-557, 2016. ISSN 1432-1769. URL* `http://dx.doi.org/10.1007/s00138-016-0765-7`. Il a été élaboré suite à la publication dans la conférence *IEEE International Conference on Image Processing (ICIP) 2015* [18].

Dans cet article, nous présentons une méthode qui s'adresse au problème de synchronisation des caméras stéréo pour une scène dynamique. Le délai temporel entre deux caméras non synchronisées peut engendrer un décalage spatial pour un même point suivi par les deux caméras. Ce décalage spatial introduit des erreurs dans la reconstruction 3D du point. Notre méthode propose une triangulation invariante au délai temporel entre les caméras. Elle estime les positions correctes des points sans nécessairement estimer le délai temporel entre les caméras. Notre méthode est basée sur la géométrie épipolaire. Elle part du principe qu'un point décalé sort probablement de sa ligne épipolaire. L'intersection de la trajectoire que fait le point (supposé être en mouvement vu que la scène est dynamique) avec la ligne épipolaire donne un bon estimé de sa vraie position sans délai temporel. L'avantage majeur de notre méthode par rapport aux autres existantes dans l'état de l'art est qu'elle est invariante aussi aux variations du délai temporel durant la capture de la séquence.

Nous avons testé notre méthode sur des données synthétiques et réelles et elle est com-

parée à d'autres méthodes.

L'article est présenté dans sa version originale.

Abstract

This paper addresses the stereo camera synchronization problem for dynamic scenes by proposing a new triangulation method which is invariant to the temporal offset of the cameras. Contrary to spatio-temporal alignment approaches, our method estimates the correct positions of the tracked points without explicitly estimating the temporal offset of the cameras. The method relies on epipolar geometry. In the presence of a temporal delay, a tracked point does not lie on its corresponding epipolar line, thereby inducing triangulation errors. We propose to solve this problem by intersecting its motion trajectory with the corresponding epipolar line. The method we propose does not require calibrated cameras, since it can rely on the fundamental matrix. One major advantage of our approach is that the temporal offset can change throughout the sequence. Evaluated with synthetic and real data, our method proves to be stable and robust to varying temporal offset as well as complex motions.

2.1 Introduction

Traditionally, a 3D reconstruction of a scene is usually achieved by triangulation [31] from at least two perspective views that are simultaneously captured. It is assumed that the cameras are calibrated or at least the fundamental matrix F is known. The triangulation itself is usually solved as a linear algebraic problem [38]. When reconstructing a dynamic scene and capturing moving points, it is important to make sure that the cameras are synchronized. Otherwise, each scene point would be captured at different instants by each camera. This would induce triangulation errors because the two image rays would either intersect at an incorrect location, or simply not intersect at all. This will be explained in detail in section 2.2. By synchronizing the cameras, we ensure that the feature points are

seen at exactly the same time. Most of commercial motion cameras are synchronized using hardware solutions. Unfortunately, this increases the installation complexity and the cost of the system. Also, some of these hardware solutions are controlled by remote or wireless signals. Wireless triggers are highly dependent on network variations and then, the synchronization can become less reliable. Sivrikaya and Yener [61] reviewed the time synchronization problem and presented in detail the basic synchronization methods explicitly designed and proposed for sensor networks. Likewise, some software-based solutions reset camera clocks at the same time [6]. They control camera time-stamps with considerably smaller network dependencies. In this context, many alternative synchronization solutions based on the captured images are proposed to synchronize the cameras. They generally deal with image-to-image (or sequence to sequence) matching. Methods for multiview correspondences are mainly inspired from [36, 38, 77, 20, 68, 82, 85]. In this work, we focus on tracking a dynamic scene with unsynchronized cameras. The temporal delay between the cameras is assumed unknown or possibly changing over time. Our method makes an accurate estimation of the 2D image point coordinates without requiring knowledge of the temporal delay between the cameras. It combines the basic principles of epipolar geometry with the motion trajectory.

2.1.1 Related Work

Several works have already looked into the problem of video sequence synchronization. As mentioned above, Sivrikaya and Yener [61] published a survey reviewing the time synchronization problem in sensor networks. They explain the need for synchronization and present the basic synchronization methods for sensor networks. Tuytelaars and Van Gool [72] present a method dealing with arbitrarily moving cameras. Their algorithm needs a configuration of 5 moving points, matched and tracked throughout both sequences.

The previous references as well as others (for example [11]), treat the synchronization problem from the camera (or sensor) side. However, many other works have focused

on the temporal synchronization of already-recorded video sequences. Indeed, there is a large variety of protocols for solving this problem. In content-based temporal alignment approaches, Caspi and Irani [26] propose a method of spatio-temporal alignment of two sequences having no spatial overlap between their fields of view using two cameras attached closely together and moving jointly in space. Likewise, they propose a method of matching space-time trajectories of moving objects using the epipolar geometry [28]. This approach relates the uncalibrated camera case. It approximates the homography H (for the 2D scene case) and the fundamental matrix F (for the 3D scene case) using standard methods (Hartley and Zisserman, 2000, Chapters 3 and 10). It uses point-to-point distances for the 2D case and point-to-epipolar line distances for the 3D case, as an error function for choosing the best trajectory matching. It defines the error function as the minimization of the total of the distances. In our experiments, we compare our results to this approach.

Velipasalar and Wolf [73] propose a content-based method of temporal alignment of video sequences by using projective invariants. Many solutions for spatio-temporal alignment of unsynchronized cameras are proposed in the literature [66, 54, 59, 75, 55]. In addition to Caspi *et al.*'s work [28], several other temporal alignment approaches are dealing with both uncalibrated and unsynchronized cameras. For instance, Wolf and Zomet [76] present an algorithm for reconstructing a dynamic scene from sequences acquired by two uncalibrated and unsynchronized affine cameras. They choose affine models probably because they are easier to solve. They use linear algorithms for synchronizing the two sequences and reconstructing the 3D points tracked in both views. One particular work was proposed by Reid and Zisserman [57] in order to solve the 1966's world cup goal problem. They estimate the time difference by computing the ground plane homography between each couple of images and measuring the error when the homography is applied to moving objects. Stein [62] shows that given a set of feature correspondences, it is possible to determine the camera geometry. He shows that enforcing geometric constraints enables the alignment of the tracking data in time. A method of 3D reconstruction for uncalibrated and unsynchronized cameras is proposed by Sinha and Pollefeys [60]. They compute the temporal offset and

epipolar geometry using a RANSAC-based algorithm to search for the epipoles. The calibration and synchronization for the complete camera network is recovered and then refined through maximum likelihood estimation.

Zhou and Tao [83] estimate the temporal offset between the cameras in the second view based on the cross ratio of four 3D points. They synthesize sub-frame images in order to synchronize the sequences. They estimate the optical flow fields between consecutive frames and then warp images to make an in-between image. Several synchronization methods are adapted for human motion. For instance, Tresadern and Reid [70] present a method of synchronizing video sequences of human motion. They estimate synchronization parameters to sub-frame accuracy for sequences of unknown and differing frame rates. Besides sequences alignment, several other works propose solutions for 3D points reconstruction from unsynchronized stereo cameras. For markerless motion capture, Hasler *et al.* [39] track people with off-the-shelf handheld video cameras. Camera synchronization is achieved via the audio streams recorded by the cameras in parallel. Kakumu *et al.* [41] propose a method that does not recover the temporal delay between the cameras. They show that by using a set of unsynchronized cameras instead of synchronized ones, they can obtain much more information on 3D motions and thus, they can reconstruct higher frequency 3D motions than that with the standard synchronized cameras. In addition to stereo camera synchronization, Lei and Yang [42] propose a synchronization approach adapted for three view vision systems.

In contrast with all time offset recovery solutions presented above, we develop a temporal offset invariant 3D reconstruction method. Our approach is based on the epipolar geometry of the cameras. It assumes a prior knowledge of the fundamental matrix F, which can be obtained from a set of corresponding points between two uncalibrated cameras. With this solution, we do not need to make any temporal alignment of the sequences. It approximates the correct 3D positions of the tracked points without knowing or even estimating the time shift between the cameras. One major advantage of our approach is that the temporal offset can change throughout the sequence. The major contribution of this work

is to provide a temporal offset invariant triangulation solution.

2.1.2 Paper Outline

The remainder of the paper is organized as follows. In Section 2.2, we explain in detail the problem we intend to resolve. In Section 2.3, we present the theory and method for solving the 3D reconstruction errors caused by the temporal offset of the cameras. In Section 2.4, we compare our method to the method introduced by Caspi *et al.* [28]. Section 2.5 shows experimental results and comparisons of all the tested methods. Section 2.6 discusses the approach and the results.

2.2 Problem Formulation

In this section, we introduce two use cases : 2D and 3D scenes. We explain how to recover the temporal delay between the cameras in each case.

2.2.1 Homography Model : 2D Scenes

The homography model corresponding to the 2D scene case has been mainly studied in the spatio-temporal alignment methods of Caspi, Simakov and Irani [27, 26, 28]. Homography model can be used in two cases :

1. The cameras observe points moving independently on a plane.
2. The distance between the camera centers is too small or even negligible compared to the distance between the cameras and the scene.

Theoretically, in the two-view geometry, the second case is formulated as : when a transformation between the two cameras is only a pure rotation (no translation and the same optic center). The relation between the two sequences of corresponding feature points p_1

and p_2, is given by the 3×3 homograpy H,

$$H = \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{pmatrix}. \tag{2.1}$$

It is defined as,

$$Hp_1 \propto p_2, \tag{2.2}$$

where $\propto$ expresses that the equality of homogeneous coordinates is subject to the scale factor.

As we treat the case of an existing temporal delay between the cameras, the delayed point p'_2 coordinates are not equal to p_2 coordinates. The distance between p_2 and p'_2 is the geometric error due to the temporal offset. This means that p_2 is the point corresponding to zero time shift. Knowing H, it is possible to recover p'_2 coordinates.

2.2.2 Perspective Model : 3D Dynamic Scenes

The perspective projection model addresses a more realistic but complex case : when the 3D scene contains independently moving points. More precisely, we examine here the case where two cameras with unknown frame rates and possibly being unsynchronized (having an unknown temporal offset), are capturing a dynamic 3D scene. Consider the case of two video sequences $Cam_1 = I_1^1 ... I_i^1 ... I_n^1$ and $Cam_2 = I_1^2 ... I_i^2 ... I_n^2$ captured with two cameras having the previously described experimental conditions. In such conditions, the i^{th} frame in each camera (I_i^1 and I_i^2) may not really correspond. This may induce a wrong 3D reconstruction (see Fig. 2.1).

The temporal delay only matters when dealing with a dynamic scene. In fact, when capturing a static scene, even with a big temporal delay, points are always captured at the same spatial positions.

In order to understand the relationship between the temporal delay and the 3D reconstruction errors, let us consider a moving 3D point parallel to the baseline direction. If the

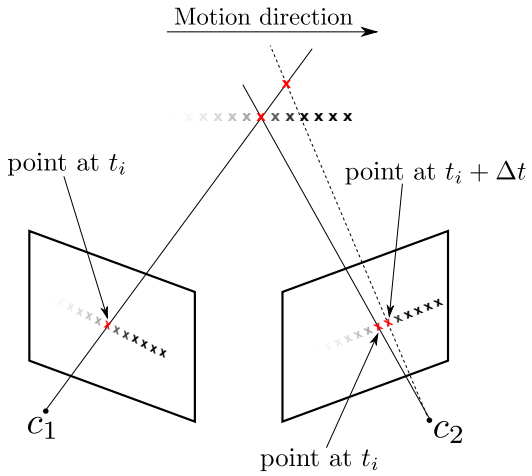
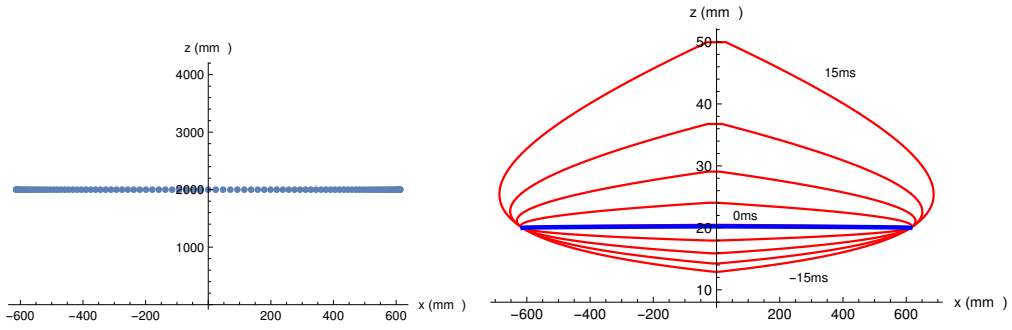


FIGURE **2.1. Camera C_2 is delayed. A captured point at time t_i by the camera C_1 corresponds to a point captured at time $t_i + \Delta t$. The points do not match and then, the reconstructed 3D point position is not correct. In this case, a reconstructed 3D linear motion becomes parabolic.**

second camera is not synchronized to the first, the selected point in the second image does not represent the real point at time t_i. It represents the projection of the 3D point at time $t_i + \Delta t$. For non-zero Δt, the reconstructed 3D point would not be at its real position. Moreover, the reconstructed trajectory of a linear motion would not be linear anymore. In order to demonstrate this, we simulated various delays between two perspective cameras observing a point undergoing a linear motion parallel to the baseline. The reconstructed trajectories are illustrated in Fig. 2.2. They are reconstructed closer or further to the real trajectory depending on which camera is delayed. All the trajectories intersect at the endpoints because the motion velocity is zero. Notice that closer trajectories show less displacement than further trajectories. This is caused by the fact that z is inversely proportional to the disparity. As the disparity is linearly affected by the temporal delay, z is exponentially affected.



(a) (b)

FIGURE 2.2. **(a) Top view of an example of a linear motion with a variable velocity. (b) Top view of 3D reconstruction of a linear motion along the baseline direction with different temporal delays between the cameras. Red trajectories represent the delayed reconstructions from $-15ms$ to $+15ms$. Blue line represents the optimal reconstruction (a linear trajectory) and corresponds to the $0ms$ temporal delay.**

2.3 Temporal Offset Recovery

Our goal is to recover the temporal offset by using information embedded in the epipolar geometry. Since it restricts the correspondence of a point in the first image to a line in the second image, the search for correspondence is constrained from 2D to only 1D. In the presence of an unknown delay between the cameras, a corresponding point in the second image could lie outside its epipolar line. We then need to recover the initially established correspondences. We can take advantage of this fact to get information about the delay and possibly recover the true trajectory.

Technically, epipolar lines in two-view geometry are computed from the fundamental matrix F [38]. Given a pair of images, each point p_1 from the first image has a corresponding epipolar line l_2 in the second image which contains the corresponding point p_2 (see Fig. 2.3). This is represented as

$$p_2^T F p_1 \simeq 0.$$

As a result, for a point p_1 from the first image, the corresponding epipolar line is $l_2 \simeq F p_1$.

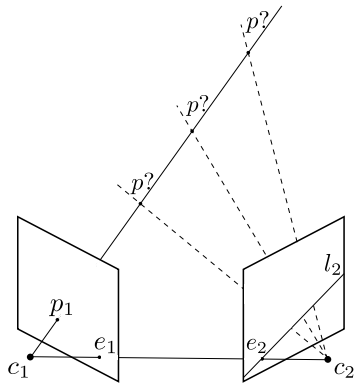


FIGURE **2.3. The epipolar line represents the intersection of the plane (c_1c_2p) with the camera image. The epipoles e_1 and e_2 represent the intersection of the epipolar line with cameras center line c_1c_2.**

2.3.1 Intersection

In the presence of a temporal delay, the observed point p'_2 from the second image corresponding to p_1 from the first image will probably not lie on the epipolar line l_2 corresponding to p_1. At frame i, the epipolar line l_{2_i} will separate points p'_{2_i} and $p'_{2_{i+1}}$ under the right circumstances. Therefore, the intersection of the segment $[p'_{2_i}, p'_{2_{i+1}}]$ with the epipolar line l_{2_i} will be the best estimation of the real position $p^*_{2_i}$ which is theoretically p_{2_i} (see Fig. 2.4). This is true under the assumption that the inter-frame motion is small enough to be considered linear. In practice, for a frame rate of 60 fps, typical "real world" motions will satisfy this assumption.

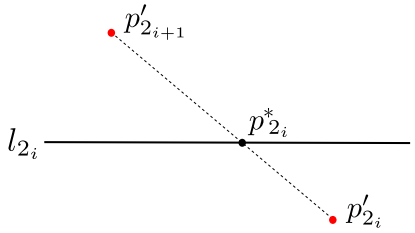


FIGURE **2.4. Intersection of the epipolar line l_{2_i} and the segment of the trajectory $[p'_{2_i}, p'_{2_{i+1}}]$. The black point represents the best approximation $p^*_{2_i}$ of the true position p_{2_i}.**

For each point p_{1_i} in the first image, we compute its epipolar line equation. We also compute the line equation for the segment $[p'_{2_i}\ ,\ p'_{2_{i+1}}]$ representing the motion trajectory. We then compute the intersection of these lines, which represents the best estimation of the true value p_{2_i}.

This approximation resolves the correspondence problem without explicitly estimating the temporal delay between the cameras. Moreover, in the case where the temporal delay is changing, this approach still applies. This method does not work under all circumstances. The next section will introduce a generalization which will cover all the cases.

2.3.2 Generalized Approach

The approach we propose can be affected by the direction of the motion. For example, if the observed motion is parallel to the epipolar line, then no intersection can be computed. The same is true for stationary points because the segment $[p'_{2_i},p'_{2_{i+1}}]$ becomes a single point. In practice, observation noise will make the intersection unstable for these two cases.

Ill-defined Intersection Problem

When we have stationary points or motion parallel to the epipolar line, we cannot guarantee that the segment $[p'_{2_i},p'_{2_{i+1}}]$ intersects the epipolar line. Obviously, points p'_{2_i} and $p'_{2_{i+1}}$ will be on the same side of the epipolar line. In this case, intersecting the trajectory line with the epipolar line will yield a solution outside the segment which is known to be wrong (point x in Fig. 2.5). For the case of the stationary point, the motion segment will be random because of noise.

Because of acquisition noise, a stationary point captured n times will be uniformly distributed. This could induce random motion trajectories, as shown in Fig. 2.6.

In order to recover from these ill-defined cases, we propose an approach to estimate more robustly the best point position on the epipolar line. It relies on the orthogonal projection of the points p'_{2_i} and $p'_{2_{i+1}}$ on the epipolar line. A weighted middle point is computed

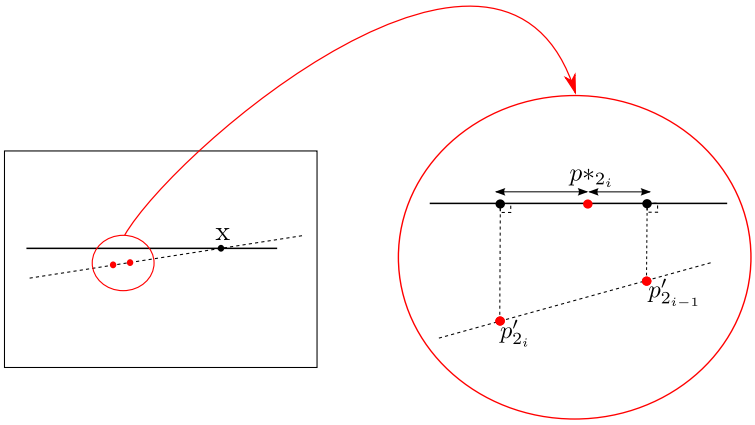


FIGURE **2.5. Point projection solution : an alternative to line intersection. We project p'_{2_i} and $p'_{2_{i+1}}$ (i is the frame index) on the epipolar line l_{2_i}. We approximate the final point by weighting with the point distances to the line.**

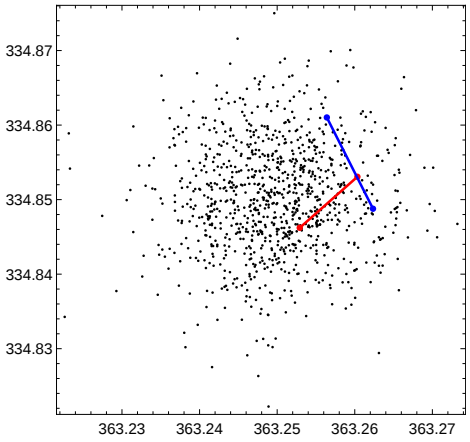


FIGURE **2.6. 2D Distribution of the positions of a marker centroid captured 1000 times. As an example, red and blue points represent two successive points showing random trajectory lines.**

from the two projected points. It represents the recovered position of p'_{2_i}. The weights are defined as the distance from the point to the epipolar line. As illustrated in Fig. 2.5, we denote $p^*_{2_i}$ as the final point and $proj_{2_i}$ and $proj_{2_{i+1}}$ as the projections of p'_{2_i} and $p'_{2_{i+1}}$. The

final point $p^*_{2_i}$ is recovered from the relation,

$$\frac{\|\overrightarrow{p^*_{2_i} proj_{2_i}}\|}{\|\overrightarrow{p^*_{2_i} proj_{2_{i+1}}}\|} = \frac{\|\overrightarrow{p'_{2_i} proj_{2_i}}\|}{\|\overrightarrow{p'_{2_{i+1}} proj_{2_{i+1}}}\|}. \tag{2.3}$$

More than One Frame Delay

Until now, we only addressed the case of a temporal delay of a single frame. As mentioned before, our method searches the best point between p'_{2_i} and $p'_{2_{i+1}}$. But what happens when the temporal delay is longer than one frame ? In that case, we need to use a larger temporal window to search the best point. As an example, Fig. 2.7 illustrates a case with a temporal delay of four frames. The epipolar line l_{2_i} intersects the motion trajectory at the segment $[p'_{2_{i+4}}, p'_{2_{i+5}}]$. Estimating the solution from p'_{2_i} and $p'_{2_{i+1}}$ would give a bad solution since l_{2_i} does not intersect the motion trajectory of that segment.

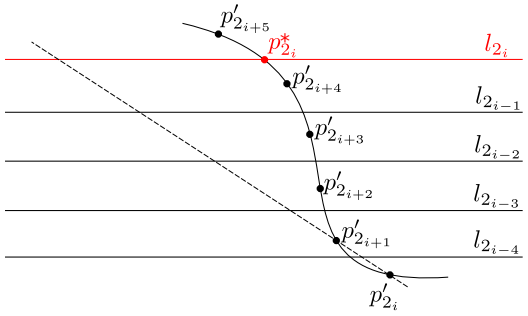


FIGURE **2.7. Four frames of temporal delay. l_{2_i} intersects the motion trajectory at the segment $[p'_{2_{i+4}}, p'_{2_{i+5}}]$.**

The problem is then reduced to finding the part of the trajectory which intersects l_{2_i}. In other words, we need to find consecutive points that lie on both sides of l_{2_i} in order to guarantee a valid intersection point. This approach to large time delay is still compatible with varying temporal delays during capture.

Multiple-point Reconstruction

For the case of multiple moving points in the scene, we do not have to treat each point individually. Obviously, they all share the same temporal delay if they are taken from a global shutter camera. In that case, all the points will be displaced according to a single temporal delay. At a given frame i of the second camera, we need to find a point j such that the segment $[p'_{2_{i,j}}, p'_{2_{i+1,j}}]$ will intersect the epipolar line. Clearly the true point $p^*_{2_{i,j}}$ is likely to be inside this segment. At this step, we define two initial constraints on the true point $p^*_{2_{i,j}}$: (1) It should be as close as possible to the epipolar line. (2) It should be inside the segment $[p'_{2_{i,j}}, p'_{2_{i+1,j}}]$, that is

$$p^*_{2_{i,j}} = (1-t)p'_{2_{i+1,j}} + tp'_{2_{i,j}}. \tag{2.4}$$

Generalizing for all tracked points in the frame, the objective is to find the best t that minimizes the sum of distances of all points to their epipolar lines. It can be formulated as

$$\min \sum_j d_j^2 = \min \sum_j \left((p_j - r_j).\frac{(a,b)_j}{\|(a,b)\|_j} \right)^2. \tag{2.5}$$

where, at frame i, p_j is the j^{th} point to be recovered on the frame, r_j is a point on the epipolar line l_{2_j} and $(a,b)_j$ are the coordinates of the normal vector of the j^{th} epipolar line. By substituting Eq. 2.4 in Eq. 2.5 we obtain the cost function

$$\min \sum_j \left((((1-t)p'_{2_{i+1,j}} + tp'_{2_{i,j}}) - r_j).\frac{(a,b)_j}{\|(a,b)\|_j} \right)^2. \tag{2.6}$$

Finding t which minimizes this cost function can be easily achieved. We then use Eq. 2.4 to recover all points $p^*_{2_{i,j}}$ for frame i.

Rolling Shutter Cameras

Unlike global shutter cameras, rolling shutter cameras do not expose the entire sensor at the same time. They expose it line by line. This introduces a temporal delay between the tracked points within the same frame. As a result, some motion distortion effects can

appear in the final image. In the case of multi-camera tracking, this capture technique can cause several matching problems even if there is no delay between the cameras. However, it is possible to compensate the spatial displacement of each point by interpolating its motion trajectory based on its vertical position [43, 12].

2.3.3 Algorithm Formulation

Our approach is summarized in Algorithm 1. It performs the intersection of the motion trajectory and the epipolar line while achieving a 3D reconstruction which is invariant to temporal offset, for all the cases presented above.

2.4 Comparison with the Method of Caspi et al.

Caspi *et al.* [28] also address the problem of matching two unsynchronized video sequences. Their method relies on feature-based sequence matching. They use trajectories of corresponding points over a long sequence of frames. Like our method, they use the fundamental matrix as basis for estimating the temporal delay.

In order to minimize the reconstruction error, they propose a *trajectory-based sequence matching algorithm*. This algorithm consists of minimizing an error function defined as the sum of the pixel distances between each point and its corresponding epipolar line. The sum is over all the trajectories and over all frames. The minimum of the error function represents the best temporal delay.

To summarize, this approach minimizes the total reconstruction error. Consequently, it assumes that the temporal delay between the cameras is constant during the whole sequence. We consider that it is less efficient in the case of a variable temporal delay. Such case could occur when using low-cost devices such as webcams. In the experimentation section, we evaluated the performance of this algorithm in such conditions.

Furthermore, this method is not designed for real-time applications, but rather for pre-recorded video sequences. Note that in this comparison, we only focus on the 3D case.

Algorithm 1 Generalized epipolar point-line triangulation algorithm.

for each frame **do**
  if one point **then**
    if stationary **then**
      Project p'_{2_i} and $p'_{2_{i+1}}$ on l_{2_i}
      Find the weighted middle point $p^*_{2_i}$
    else
      Find segment $[p'_{2_k}, p'_{2_{k+1}}]$ that intersects l_{2_i}
      Set the intersection as the solution $p^*_{2_i}$
    end if
    Triangulate
  else if many points **then**
    if all points are stationary **then**
      for each point **do**
        Project $p'_{2_{i,j}}$ and $p'_{2_{i+1,j}}$ on $l_{2_{ij}}$
        Find the weighted middle point $p^*_{2_{ij}}$
      end for
    else
      Find t minimizing the global projection error
      Recover all the points coordinates using t
    end if
    Triangulate
  end if
end for

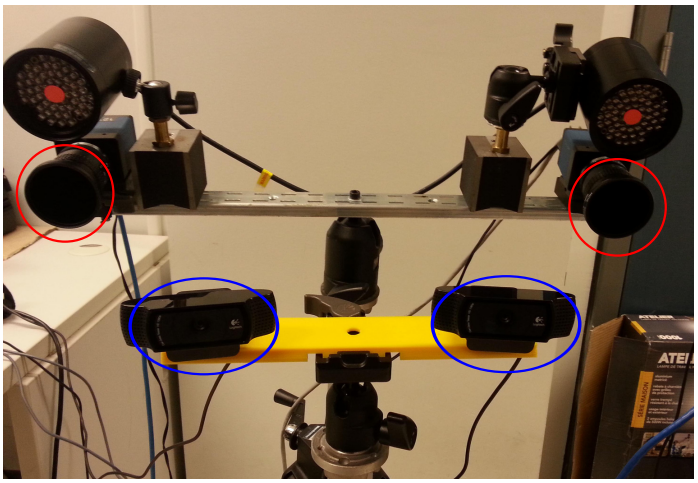


FIGURE **2.8. Experimental setup : Red circles are our ground truth cameras (Prosilica GE680). The blue circles are the webcams (HD Logitech C920).**

2.5 Experiments

In this section, we evaluate our proposed method and we also compare its performance to Caspi *et al.*'s method [28]. As shown in Fig. 2.8, our experimental setup is composed of a system of stereo cameras used as reference (two Prosilica *GE*650 gige cameras [6]). They feature a resolution of 659×493 and a baseline distance of $45cm$. They are temporally synchronized in order to provide a ground truth. The second camera system is composed of two webcams (HD Logitech *C*920 USB cameras [7]). They are used to test the variable temporal delay case as they feature an unstable temporal delay (see Fig. 2.14). In these experiments, we used infrared light to track infrared markers. The reason for this is to ensure perfect point correspondences, thereby making sure that we only test our method and not the quality of the tracking algorithm. The use of infrared light is not a constraint because our method still works under any other acquisition conditions using any features-tracking and corresponding algorithms.

We tested our method by performing various experiments on linear and nonlinear marker motion. The first set of experiments concerns the reconstruction of the linear trajectory

of a marker. In the presence of a temporal delay, a linear motion parallel to the baseline with variable motion velocity is always reconstructed as a curve (instead of a straight line) depending on which camera is delayed (see Fig. 2.2).

We evaluated the linearity of the reconstructed trajectories using standard triangulation, our method and Caspi *et al.*'s method [28]. The second set of experiments consists of tracking a more complex motion : human gait. The gait was captured using markers on the heels while walking on a treadmill placed in front of the cameras and spanning a distance of 2 to 3 meters (see Fig. 2.13). The ground truth cameras were synchronized within $0.1ms$. We captured the motion without adding a temporal offset and we introduced virtual temporal delay to compare our approach to the standard triangulation. We also tested our algorithm as well as Caspi's algorithm with a gradually increasing temporal delay.

2.5.1 Linear Motion

In this experiment, we tried to reconstruct a linear motion in the presence of a temporal offset and then compared the obtained results using three methods. Triangulation was used to reconstruct the 3D position of the marker tracked at subpixel accuracy. In order to make a linear motion, we manually moved the IR marker along horizontal linear slide. The motion direction is in the x-z plane of the camera system, oriented at an angle of 30 degrees of the baseline. The frame rate was set to 30 *fps*. We varied the temporal delay from $1ms$ to $14ms$ with a step of $1ms$. We then applied standard triangulation, our epipolar point-line triangulation as well as Caspi *et al.*'s triangulation method. Fig. 2.9 shows the 3D reconstruction of the marker using (a) the epipolar point-line triangulation and (b) the standard triangulation. In this figure, the trajectories of all temporal offsets are displayed. Clearly, temporal delay has an impact on the precision of the reconstruction. The standard triangulation is sensitive to temporal offset variations while our method keeps the linearity of the trajectory. In order to quantify the reconstruction errors, we estimated the linearity of each trajectory using the points collinearity. It can be factorized using *Singular Value*

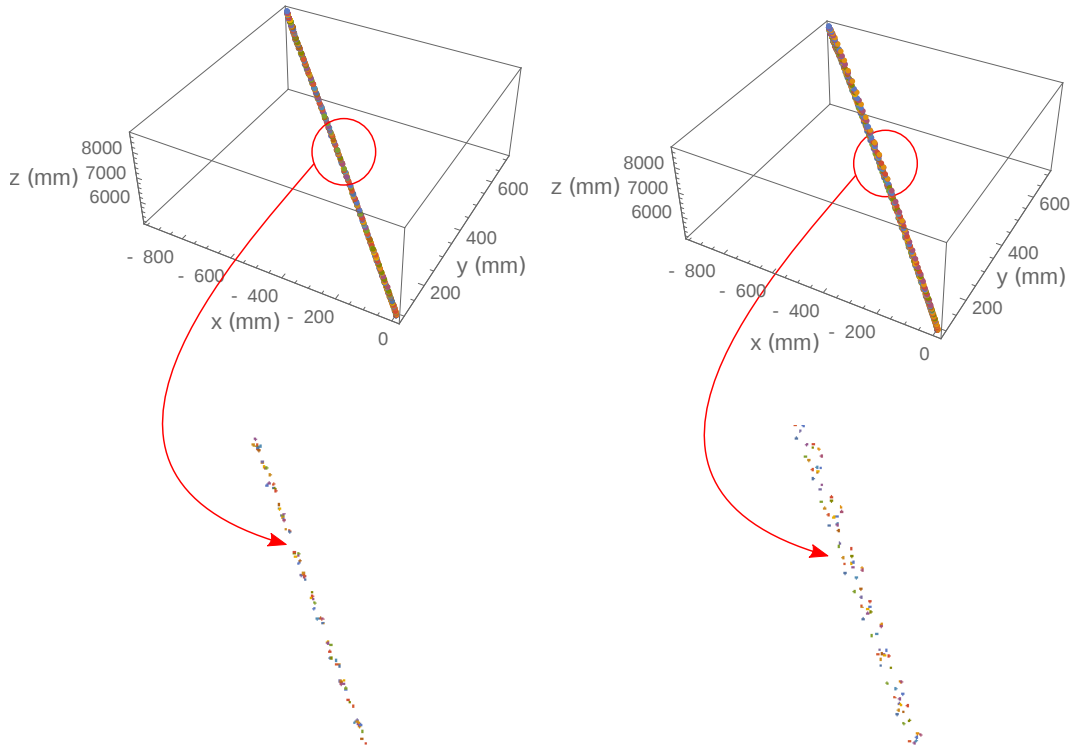


FIGURE **2.9. 3D reconstruction of a linear trajectory with (a) epipolar point-line triangulation and (b) standard triangulation for different temporal offsets (from** $0ms$ **to** $14ms$ **with a step of** $1ms$**). Experimental frame rate is** $30\ fps$**.**

Decomposition (SVD) to the form

$$A = UDV^T$$

where U and V are orthonormal matrices, and D is a $3 \times n$ matrix containing the singular values of matrix A. The vector in V associated to the maximum singular value defines the direction of the best-fit-line l of the set of points. With this direction we can calculate the distance of each point to the line. The mean distance of the points to the best-fit-line is our measure of the linearity of the reconstructed trajectories.

As illustrated in Fig. 2.10, the standard triangulation reconstruction error increases with

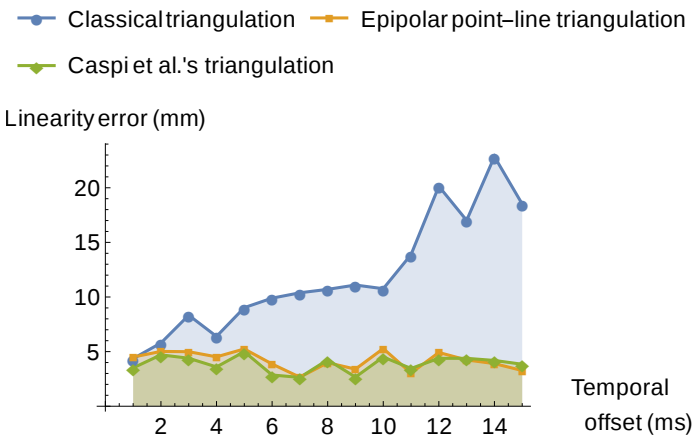


FIGURE **2.10. Comparison of the mean of distances (mean error) for standard triangulation algorithm, epipolar point-line triangulation algorithm and Caspi *et al.*'s algorithm. The error depends on the temporal offset.**

increasing temporal offset between the cameras. In contrast, our method and Caspi *et al.*'s were invariant to the temporal offset variations : the reconstruction error is almost stable. For the three methods, we obtained mean errors of $11.9mm$, $4.2mm$ and $3.9mm$ respectively. To solve the temporal delay with Caspi *et al.*'s method, we estimated the cost function for various Δt varying from $-15ms$ to $+15ms$ (half the period of a frame) with a step of $1ms$. The cost function is the sum of the distances of each point to its corresponding epipolar line. Fig. 2.11 illustrates the error function corresponding to an added temporal delay of $5ms$ between the cameras. The algorithm finds that the interpolated Δt minimizing the error function is $-7ms$.

2.5.2 Complex Motion : Human Gait

In this experiment, we tested our method on a nonlinear motion of the feet during normal human gait. We tracked two markers placed on the heels of the subject (see Fig. 2.13). We captured this motion using the same infrared vision system as in the first experiment. We compared the results of our algorithm to the normal triangulation as well as to Caspi *et al.*'s method. In order to have two sequences with temporal delays of $0ms$ and $\frac{1}{60}s$, we

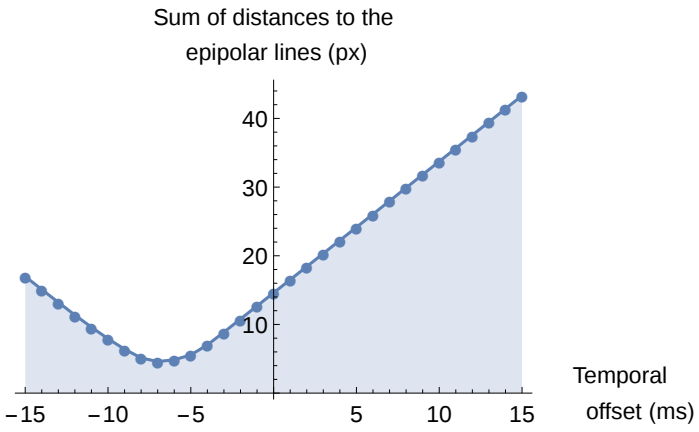


FIGURE **2.11. Error function (sum of distances to the epipolar lines) minimization depending on the temporal offset variation.**

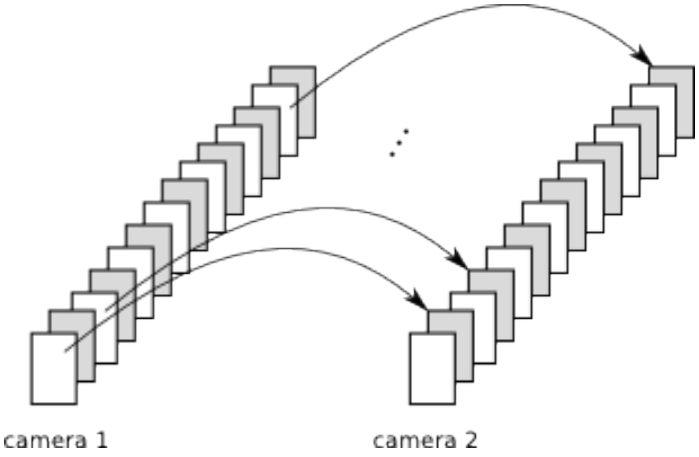


FIGURE **2.12. To simulate a temporal delay, we match each odd number frame from the first camera to an even number frame from the second camera.**

captured the motion at 60 *fps*. As illustrated in Fig. 2.12, we kept the odd frames in both cameras to obtain a synchronized sequence. We kept the even frames of the second camera to obtain $\frac{1}{60}s$ $(16,7ms)$ temporal delay. Note that initially both cameras are temporally synchronized. Results are shown in Table 2.1.

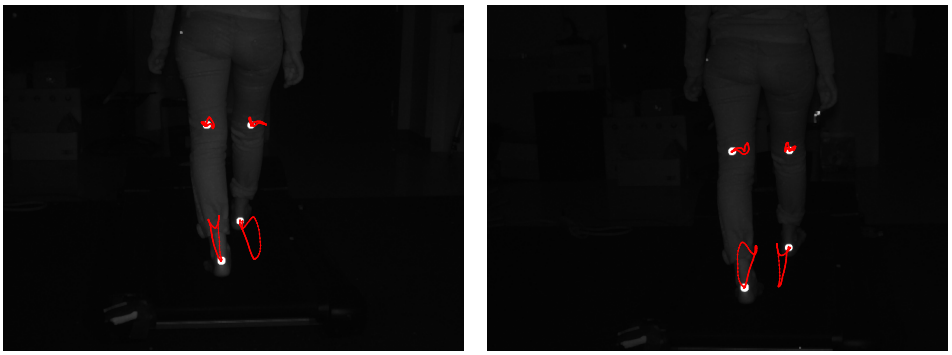
TABLE **2.1. Mean of the distances between the reconstructed trajectories without and with temporal delay. Reconstruction is performed using : normal triangulation, Caspi's algorithm and our algorithm.**

	Normal triangulation	Caspi *et al.*'s algorithm	Epipolar point-line triangulation
Mean of the distances	25.611*mm*	12.280*mm*	7.610*mm*

2.5.3 Multiple-points Tracking

Until now, our algorithm has been applied to a single tracked point. However, it is often possible to track multiple points. As we know, at a given frame, all the captured points in the scene share a single temporal delay. It is then possible to estimate this temporal delay by minimizing the total projection error of those points. This has been explained in Eq. 2.6. In this experiment, we evaluate our algorithm for the multiple-points case. We tracked four independently moving points. As illustrated in Fig. 2.13, the motion represents a human gait. As in the previous experiment, we relied on a one-frame skip in order to induce a temporal offset. The frame rate of the capture was 90 *fps* and the reconstruction frame rate was 45 *fps* (see Fig. 2.12). We compared three algorithms : normal triangulation, epipolar point-line triangulation and Caspi *et al.*'s algorithm. The error function is the mean distance between the reconstructed trajectories with and without temporal delay, over a sequence of 500 frames. The results are shown in Table 2.2 and indicate that our approach gives the minimum reconstruction error. We also observed that it is more efficient to consider all the points in order to perform a good reconstruction instead of treating them separately.

In order to show that our method can be used in real-time, we measured the processing time required by each method. Results, presented in Table 2.3, show that our method is as fast as the standard triangulation. However, we registered 1.847 seconds of processing time



(a) (b)

FIGURE **2.13. An example of the tracking of the human gait. An infrared marker is installed on the heel and knee joints of the walking subject. The intensity of the markers is very high so it is easy to isolate it and estimate its centroid subpixel coordinates. Red lines represent the trajectory during one cycle of human gait.**

TABLE **2.2. Mean of the distances between the reconstructed trajectories without and with temporal delay. Reconstruction is performed using : normal triangulation, Caspi *et al.*'s algorithm and our algorithm.**

	Normal triangulation	Caspi *et al.*'s algorithm	Epipolar point-line triangulation
Trajectory 1	39.111*mm*	26.404*mm*	8.740*mm*
Trajectory 2	32.536*mm*	22.878*mm*	6.998*mm*
Trajectory 3	37.090*mm*	24.584*mm*	6.856*mm*
Trajectory 4	26.088*mm*	17.969*mm*	7.048*mm*

using Caspi *et al.*'s algorithm.

2.5.4 Variable Temporal Delay Case

Until now, all our experiments assumed that the temporal delay between the cameras remained stable for all sequences. Let us consider the case where the temporal delay is

TABLE **2.3. Processing time per frame for each method in seconds.**

	Normal triangulation	Caspi *et al.*'s algorithm	Epipolar point-line triangulation
Processing time (in seconds)	0.002	1.847	0.003

varying during the capture. Variable delay mostly occurs when using low precision cameras such as webcams. We captured a random motion with two HD Logitech webcams [7]. The baseline separating the webcams was 18.8*cm*. As a reference, we used two high precision Ethernet cameras (Prosilica) [6] without significant temporal delay. The baseline of those cameras was 45*cm* which is slightly larger than the webcams baseline to ensure that the triangulation accuracy is higher. Both systems captured images at a frame rate of 30 *fps*.

We captured the cyclic motion of a marker with the two vision systems. During the motion capture, we recorded and compared the time-stamps of each image from the two webcams. We obtained a mean delay $\mu = 9.391ms$ and a standard deviation $\sigma = 2.752ms$, which strongly suggests that the temporal delay is varying. Fig. 2.14 shows the distribution of the temporal delay between the webcams during the motion capture. Although the time-stamps provide a good estimate of the variation of image capture process, they are not accurate enough to be considered as the true temporal offset. This is why it is advantageous to estimate the temporal delay at each frame.

We performed the three triangulation algorithms (standard triangulation, Caspi *et al.*'s algorithm and our algorithm) on the data provided by the webcams. We performed a standard triangulation on data provided from the reference system to obtain a ground truth 3D reconstruction. Fig. 2.15 shows the distances between the reference and the webcam trajectories reconstructed with the three methods. Results presented in Table 2.4 show that our method is superior in the case of a varying temporal offset between the cameras. Caspi *et al.*'s algorithm performed poorly basically because the algorithm assumes that the time

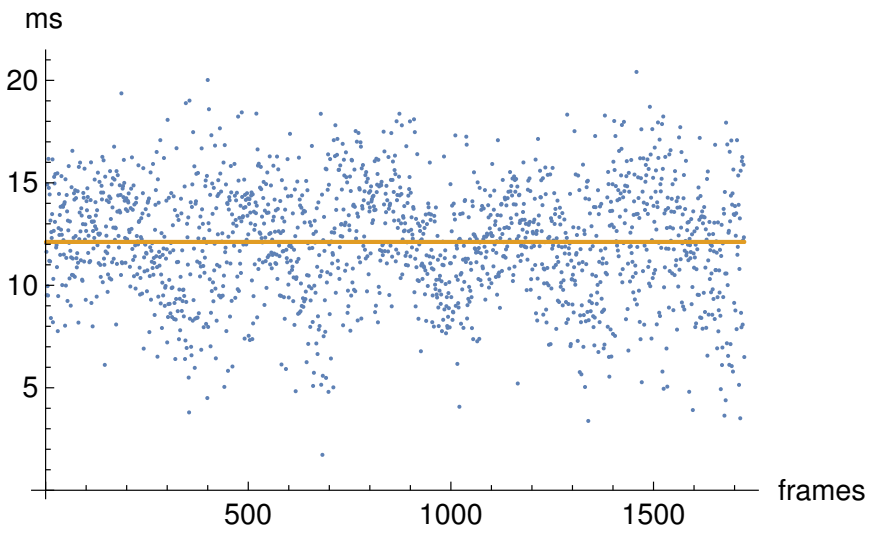


FIGURE **2.14. Distribution of the temporal delay between the webcams. The orange line represents the mean of the temporal offset.**

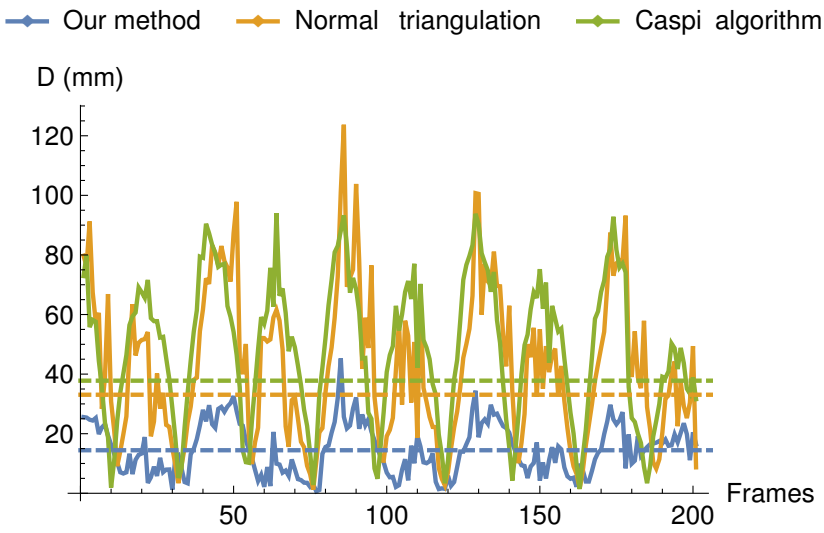


FIGURE **2.15. Distances between each reconstructed trajectory from the webcams and the reference.**

shift is invariant during the whole sequence, which is not necessarily true when using low cost cameras such as webcams. In this experimentation, we demonstrated that our method is well suited to tackle the problem of capturing with low cost cameras which can suffer

TABLE **2.4. The average distance of each trajectory from the reference.**

	Normal triangulation	Caspi *et al.*'s algorithm	Epipolar point-line triangulation
First sequence	42.5*mm*	48.4*mm*	14.8*mm*
Longer sequence (of 1700 frames)	33.1*mm*	37.8*mm*	14.4*mm*

from varying temporal delay.

TABLE **2.5. Processing time per frame for each method in seconds.**

	Normal triangulation	Caspi *et al.*'s algorithm	Epipolar point-line triangulation
Processing time (in seconds)	0.002	0.030	0.003

Table 2.5 shows the average processing time for each method. It demonstrates that our method is as fast as the standard triangulation, so it is suitable to run in real-time. Caspi's method is slightly slower and less adapted to real-time applications because it requires a large number of images.

2.6 Discussion

Through this set of experiments, we first demonstrated that our reconstruction method is invariant to temporal offset. Second, we showed that it is also robust to temporal delay variations within the same sequence. It relies on the trajectory information to perform the point position recovery. Since it treats each frame separately, it is fast and well adapted to all capture conditions (variable frame rate, variable temporal delay, etc.) as well as to real-

time applications. This has been demonstrated in the preceding experiments by measuring processing time per frame for each method.

The results show that the reconstruction error is small compared to the normal triangulation (which is sensitive to temporal delay) and as good as Caspi *et al.*'s method when tracking only one point in the case of a linear motion and a complex motion (human gait). In the case of multiple-point tracking, the reconstruction error of our algorithm is much smaller than both normal triangulation and Caspi *et al.*'s algorithm. We also showed that it is robust to the potential temporal offset variations when capturing with low-cost cameras such as webcams.

Caspi *et al.*'s algorithm performs a minimization of an error function along the whole video sequence and for all tracked trajectories to find the optimal Δt. This algorithm is well adapted to pre-recorded sequences. It is less efficient in the case of real-time applications since it requires a sufficient motion history to establish trajectories as well as a considerable computational time for each triangulation. This algorithm is also less efficient in the case of variable temporal offset.

As this method relies on the epipolar geometry, a good estimation of the fundamental matrix F is important to ensure accurate epipolar lines. In order to achieve this, it is recommended to use a bundle adjustment algorithm [71, 38, 8].

2.7 Conclusion

In this paper, we addressed the problem of dynamic scene reconstruction errors due to a temporal offset between the cameras. We proposed a solution which resolves this problem by estimating the correct tracked point positions. A correct point position is found by intersecting its motion trajectory with its corresponding epipolar line. The method reconstructs the 3D point accurately without requiring knowledge of the temporal offset. It is robust to temporal offset variations. The major contribution of this work is the development of a simple real-time solution to triangulation with unsynchronized cameras. It ensures a

temporal offset invariant 3D reconstruction.

Addendum 1

La synchronisation des séquences vidéo à l'aide de la géométrie épipolaire est une approche qui a été utilisée dans plusieurs travaux notamment [74]. Dans cet article, les auteurs comparent des séquences vidéo non synchronisées capturées à partir d'au moins trois caméras stationnaires qui pourraient avoir différentes fréquences d'images. Ils calculent la géométrie des caméras pour déterminer les matrices fondamentales et les tenseurs trifocaux. La correction des délais entre les séquences est principalement basée sur les trajectoires des points suivis. Si ces trajectoires ont assez de points d'inflection, ceci donne une bonne approximation du délai entre les séquences et aide ainsi à les synchroniser. Pour raffiner cette approximation, il est possible de se servir de la géométrie épipolaire pour corriger les positions des points à une précision sous-image (sub-frame). En l'absence des points d'inflection, cette étape est utilisée directement (sans approximation). Pour un point donné dans la première image, la ligne épipolaire correspondante dans la deuxième image peut intersecter plusieurs points appartenant à plusieurs trajectoires donnant ainsi plusieurs candidats possibles. Chaque candidat subit une transformation tensorielle pour trouver son correspondant dans la troisième image. On lui associe le point de trajectoire le plus proche. À partir des trois points sélectionnés, une étape de validation est performée. Elle consiste à extraire les points des trajectoires correspondantes en positions d'image entière (full frame pas sous-image). À l'aide de fenêtres de corrélation, ces trois points sont comparés pour s'assurer qu'ils sont correspondants. Une autre étape de validation met en correspondance les points saillants des images en question à l'aide de la géométrie déjà calculée (matrices fondamentales et tenseurs focaux). Ce modèle prend en considération la présence d'erreurs dans le calcul de la géométrie des caméras. Cette approche a été appuyée par des expérimentations sur des séquences synthétiques et réelles.

Cette approche est destinée à un système d'au moins trois caméras. Elle suppose également qu'il n'y a aucune mise en correspondance préalable entre les trajectoires (ou les points suivis), ceci ramenant cette tâche à une étape ultérieure de la résolution du problème (une fois les trois points sélectionnés, on passe à une étape de validation).

Dans notre cas, nous supposons un système à deux ou plusieurs caméras en prenant deux caméras à la fois (la géométrie des caméras est basée seulement sur la matrice fondamentale). Du côté de l'algorithme de synchronisation, nous abordons le problème autrement. Nous supposons que les points suivis sont déjà mis en correspondance (tâche faite au préalable). On sait ainsi quelle trajectoire utiliser pour l'intersecter avec la ligne épipolaire en question. Ceci simplifie le modèle parce qu'on ne regarde pas toutes les trajectoires qui intersectent la ligne épipolaire à un instant donné puisqu'on connaît quelle trajectoire nous intéresse. Ainsi, il n'est pas nécessaire de procéder à une étape de validation des candidats à la fin.

Le rapport de causalité entre les caméras n'est pas trop présent surtout dans le cas des caméras à vitesses (frame rate) variables. Admettons qu'on a deux caméras 1 *et* 2*. Si la caméra* 1 *à l'instant* i *est en avance par rapport à la caméra* 2 *d'un délai* t*, il est possible qu'à l'instant* $i+1$ *(ou plus généralement à un instant* $i+n$*) la vitesse de la caméra* 2 *augmente et que la vitesse de la caméra* 1 *baisse (elle expose plus longtemps) ce qui implique que la caméra* 2 *soit en avance. Cette situation nous a laissés supposer dans notre travail que pour chaque image* i*, on cherche le délai dans une intervalle de* $\pm n$ *images. Évidemment cette situation n'arrive pas si les caméras ont des vitesses stables mais plutôt elle risque de se produire dans le cas de caméras à vitesses variables (exemple : les webcams).*

Deuxième partie

Flux optique par isocontours

Chapitre 3

INTRODUCTION AU FLUX OPTIQUE

Dans cette partie de la thèse, nous nous intéressons à un deuxième sujet de vision par ordinateur qui est le flux optique. Le flux optique représente le mouvement apparent dans le plan image d'un point dans le monde. Ce mouvement est représenté dans l'image par le déplacement du pixel qui représente la projection d'un point 3D. Le défi est donc de trouver ce déplacement 2D. Comme les algorithmes de stéréo, les algorithmes qui déterminent le flux optique sont classés en deux principales catégories : ou bien à la fois denses et précises mais assez lentes, ou bien rapides mais moins denses et moins précises. Dans ce chapitre, nous expliquons les domaines d'utilisation du flux optique, nous présentons les premières formulations du flux optique, la comparaison des performances des principaux algorithmes qui existent dans l'état de l'art et à la fin, nous présentons notre contribution.

3.1 Domaines d'application

Le flux optique est utilisé par des applications comme la segmentation basée sur le mouvement puisqu'il est facile de séparer les objets en mouvement de ceux qui ne le sont pas. La structure à partir du mouvement (*structure from motion*) consiste à déterminer la structure 3D à partir des images 2D d'une scène en mouvement vue par une caméra possiblement aussi en mouvement. La structure à partir du mouvement est généralement obtenue à partir de la mise en correspondance des points saillants détectés. Comme le flux optique permet de mettre en correspondance les pixels entre les images, il est utilisé dans la structure à partir du mouvement. Il est aussi utilisé pour l'estimation du mouvement de la caméra. Il est également très utile pour des applications de traitement et d'analyse de vidéo. À titre d'exemple, certains standards de compression comme MPEG se servent de

l'estimation du mouvement pour calculer des images intermédiaires. Il est aussi utilisé dans les effets spéciaux au cinéma. Certains algorithmes de restauration de vieilles séquences vidéos utilisent le flux optique pour améliorer le rendu final. Il est également utilisé dans la vidéo-surveillance pour détecter des objets en mouvement.

Une autre application très importante du flux optique est en robotique pour aider le déplacement automatique des véhicules et aider à détecter et à éviter les obstacles [29, 34].

3.2 Méthodes du flux optique

Les premiers algorithmes d'estimation du flux optique ont été introduits par Horn and Shunck [40]. À partir de deux images d'une vidéo, ils ont défini une fonction à trois variables (représentant trois dimensions) $I(x,y,t)$, où x et y sont les coordonnées d'un pixel dans la première image et t est l'indice temporel de cette image. Cette fonction représente l'intensité du pixel à la position (x,y) à l'instant t. Étant donné que le temps écoulé entre la capture de deux images est très faible (pour une fréquence d'acquisition de 30 fps, on a $\frac{1}{30}$ sec qui s'écoule entre deux images), alors l'intensité du pixel déplacé selon le vecteur (u,v) à l'instant $t+1$ (à l'image suivante) devrait être semblable. Cette hypothèse est appelée la "contrainte de la conservation d'intensité" (traduction de *brightness constancy assumption*). Elle est donnée par l'équation :

$$I(x,y,t) = I(x+u, y+v, t+1). \tag{3.1}$$

Étant donné que le temps entre deux images est petit, le déplacement d'un pixel est par conséquent, petit aussi. On peut ainsi considérer $I(x,y,t)$ comme une fonction continue et la développer en *série de Taylor* :

$$I(x,y,t) = I(x,y,t) + u\frac{\partial I}{\partial x} + v\frac{\partial I}{\partial y} + 1\frac{\partial I}{\partial t}. \tag{3.2}$$

La simplification de l'approximation de l'équation 3.2 donne :

$$uI_x + vI_y + I_t = 0 \tag{3.3}$$

où $I_x = \frac{\partial I}{\partial x}$ et $I_y = \frac{\partial I}{\partial y}$ et $I_t = \frac{\partial I}{\partial t}$. On rappelle que u et v sont les coordonnées du vecteur de déplacement du pixel qu'on cherche. C'est le vecteur du flux optique. L'équation 3.3 est une équation linéaire à deux inconnues. L'ensemble des solutions réalisables est représenté par la ligne pointillée à la figure 3.1.

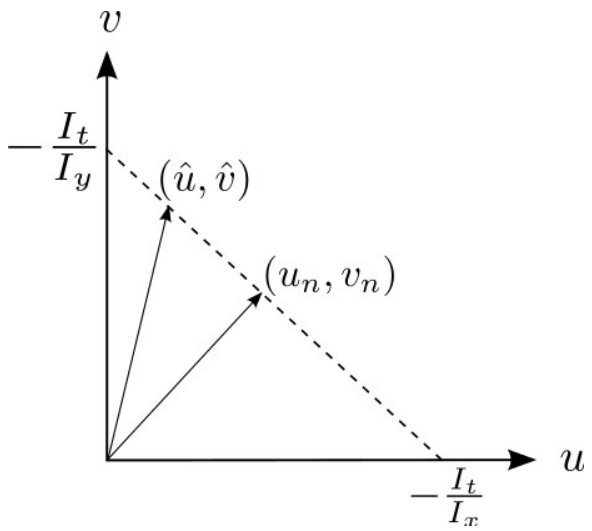


FIGURE **3.1. Le problème d'ouverture. La ligne interrompue représente l'espace de solutions du flux optique.** $(\hat{u}, \hat{v})$ **est la vraie solution.** (u_n, v_n) **est le flux normal.**

Comme cette équation a deux inconnues, la solution est ambiguë. Ce problème est connu sous le nom du problème d'ouverture (*aperture problem*). L'appellation fait référence à l'ouverture de la fenêtre de la recherche du flux optique. Si on cherche seulement à l'aide du pixel en question, l'ouverture est donc petite et la solution est ambiguë. Si on se sert par exemple des pixels voisins, l'ouverture est plus large et la solution est moins ambiguë. Il est alors important d'ajouter d'autres contraintes pour trouver le bon flux optique.

Horn and Shunck [40] ont introduit une deuxième contrainte appelée la "contrainte de régularité" (traduction de *smoothness constraint*). Cette contrainte suppose que le gradient spatial du flux est petit. Les pixels voisins ont tendance à avoir des flux similaires. Le flux varie donc spatialement d'une manière "lisse". Cette contrainte est définie comme étant des dérivées spatiales du flux pondérées par une certaine constante λ. Elle est formulée comme

suit :

$$\sum_{x,y}\left(\left(\frac{\partial u}{\partial x}\right)^2+\left(\frac{\partial u}{\partial y}\right)^2+\left(\frac{\partial v}{\partial x}\right)^2+\left(\frac{\partial v}{\partial y}\right)^2\right).$$

Elle peut être réécrite comme suit :

$$\lambda\sum_{x,y}\left(u_x^2+u_y^2+v_x^2+v_y^2\right) \tag{3.4}$$

où $u_x^2=\left(\frac{\partial u}{\partial x}\right)^2$ et ainsi de suite. La combinaison des deux termes (le terme de l'erreur 3.3 et le terme de la régularité 3.4) forme une fonction à minimiser globalement (sur tous les pixels de l'image). L'équation globale est comme suit :

$$E_{HS}(u,v)=\int_\Omega((I_xu+I_yv+I_t)^2+\lambda(|\nabla u|^2+|\nabla v|^2))dxdy \tag{3.5}$$

où Ω est le domaine de l'image, ∇u est le gradient de u et ∇v est le gradient de v. L'équation 3.5 est par la suite minimisée pour résoudre u et v. Nous expliquons avec plus de détails les étapes de la résolution de u et v dans l'annexe B. u et v sont déterminées itérativement.

Lucas et Kanade [47] ont proposé aussi un algorithme de calcul de flux optique qui part du même principe : la conservation de l'intensité. La première contrainte de l'algorithme est celle donnée par l'équation 4.1. Face au problème d'ouverture, ils supposent aussi que les pixels voisins ont le même flux. Ils utilisent alors une fenêtre de 3×3 pixels pour résoudre l'équation 4.1. Ceci donne un système à neuf équations pour trouver deux inconnues u et v. Comme ce système est surdéterminé, Lucas et Kanade utilisent la méthode des moindres carrés pour le résoudre. Nous expliquons avec plus de détails la démarche de l'algorithme de Lucas et Kanade dans l'annexe B.

Les deux méthodes présentées ici représentent les approches les plus fondamentales du flux optique. Elles sont dites différentielles car elles sont basées sur la dérivation spatio-temporelle de l'image. Dans le même contexte, il existe d'autres algorithmes dans la littérature basés sur la même fonction d'énergie [25, 23, 50, 84]. Les travaux suivants aussi ont proposé des méthodes intéressantes pour résoudre le problème de la détermination du flux optique. Comme mentionné par Barron *et al.* [15], les approches qui divisent l'image

en régions représentent une alternative aux limitations des méthodes différentielles (optimisent globalement, moins bons sur les discontinuités, etc.). Elles consistent à chercher le flux en décomposant l'image en régions. Les régions sont obtenues à l'aide des algorithmes de segmentation [21, 78]. Un autre type d'approches consiste à l'utilisation des techniques de l'apprentissage machine [64, 58]. D'autres méthodes non denses utilisent la correspondance des points saillants (*feature corresponding algorithms*) afin de détecter les grands mouvements. Pour densifier le flux résultant, ils utilisent des méthodes de minimisation de fonctions d'énergie [24, 44], qui sont généralement lentes. Toujours dans le cas de la détection des grands mouvements tout en gardant un résultat dense, Brox et Malik [24] ont proposé d'utiliser des descripteurs et de les intégrer avec des méthodes de flux optique variationnelles. Une récente taxonomie des méthodes du flux optique est présentée par Baker *et al.* [14]. Pour des revues qui couvrent des travaux plus anciens, il est recommandé de voir ces travaux [15, 48, 53, 63].

3.3 Problème des grands mouvements dans le flux optique

La majorité des algorithmes présentés ici performent bien dans le cas des petits mouvements. Cependant ils performent moins bien, ou même échouent, dans le cas de mouvements plus grands. La solution est donc d'ajouter une étape supplémentaire qui consiste à construire une pyramide à partir de l'image originale en réduisant ses dimensions à chaque fois. Lorsqu'on réduit les dimensions d'une image, la vitesse du mouvement est par conséquent réduite du même facteur. Le sous-échantillonnage réduit les hautes fréquences ce qui élimine les minimums locaux. D'abord, l'algorithme est appliqué sur les plus petites images. Par la suite, le résultat du plus haut niveau est utilisé pour initialiser le niveau qui suit et ainsi de suite jusqu'à arriver au plus bas niveau qui représente l'image originale. Ceci aide à détecter les plus grands mouvements (voir la figure 3.2).

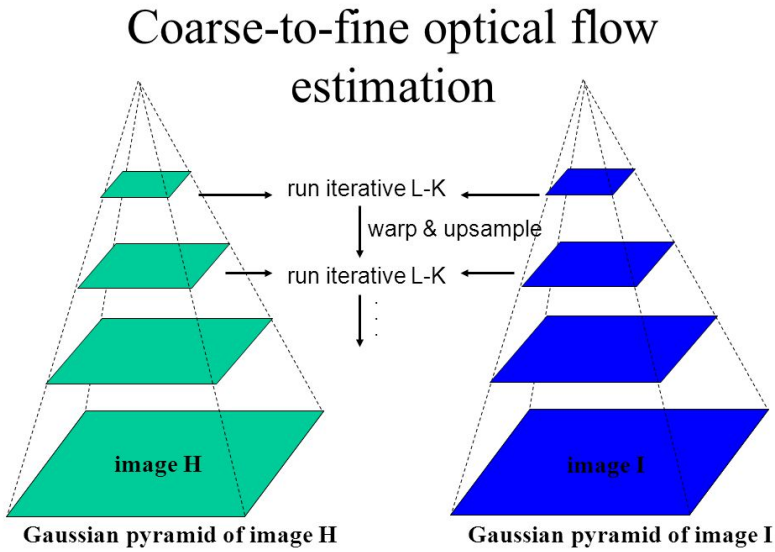


FIGURE 3.2. **Approche pyramidale pour trouver les grands mouvements. Figure tirée de [32].**

3.4 Évaluation de la performance des méthodes du flux optique

Le tableau de Middlebury illustré à la figure 3.3 liste l'ensemble de méthodes de flux optique testées sur l'ensemble des images fournies sur le site [3]. Les fonctions d'erreur utilisées sont : l'erreur du point final (*Endpoint Error EE*), l'erreur angulaire (*Angular Error*), l'erreur d'interpolation (*Interpolation Error*) et l'erreur d'interpolation normalisée (*Normalised Interpolation Error*). Ces mesures sont données par les équations suivantes :

$$AE = cos^{-1}\Big(\frac{1.0 + u \times u_{GT} + v \times v_{GT}}{\sqrt{1.0+u^2+v^2}\sqrt{1.0+u_{GT}^2+v_{GT}^2}}\Big) \tag{3.6}$$

$$EE = \sqrt{(u-u_{GT})^2 + (v-v_{GT})^2} \tag{3.7}$$

$$IE = \left[\frac{1}{N}\sum_{(x,y)}\Big(I(x,y) - I_{GT}(x,y)\Big)^2\right]^{\frac{1}{2}} \tag{3.8}$$

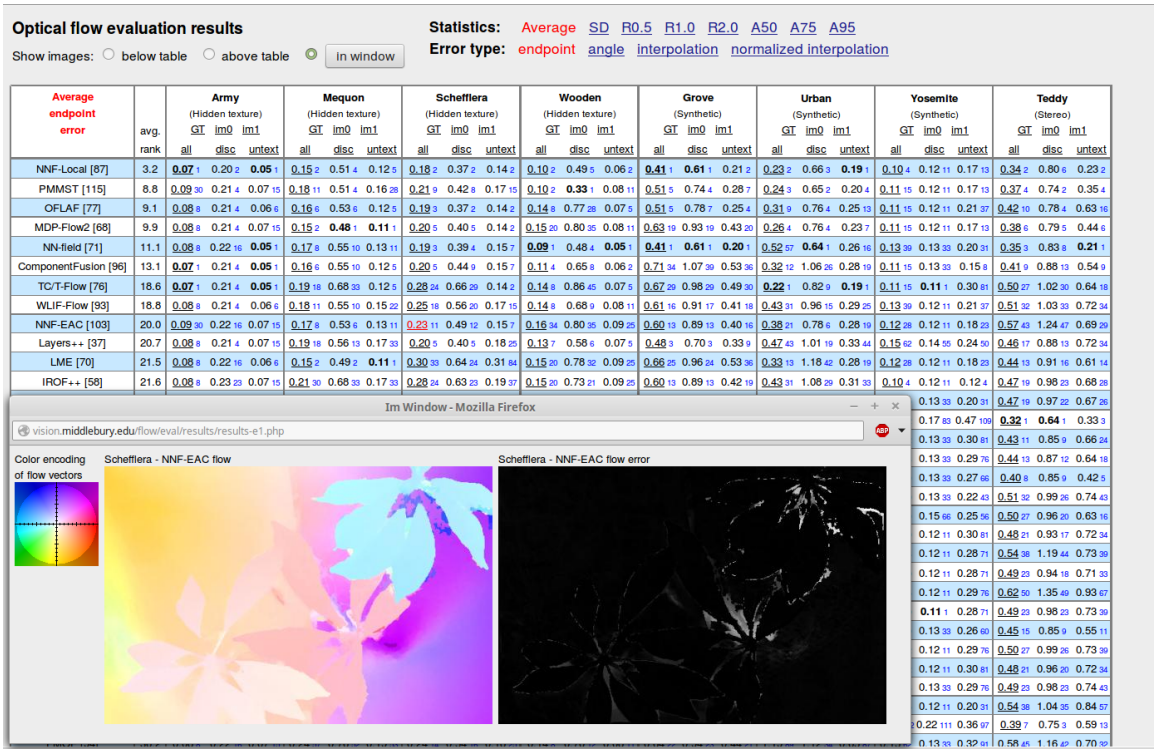


FIGURE **3.3. Une capture d'écran du tableau de Middlebury. On y trouve un ensemble d'images d'évaluation et un ensemble d'algorithmes de flux optique. Les critères d'évaluation sont présentés en haut à gauche. Le critère sélectionné est le** *Endpoint Error*. **En déplaçant la souris sur les résultats, des figures s'affichent montrant le flux optique obtenu ainsi qu'une carte d'erreur [3].**

$$NE = \left[\frac{1}{N} \sum_{(x,y)} \frac{(I(x,y) - I_{GT}(x,y))^2}{\| \bigtriangledown I_{GT}(x,y) \|^2 + \varepsilon} \right]^{\frac{1}{2}}. \tag{3.9}$$

D'autres statistiques sont aussi utilisées comme la moyenne (*Average*) et la déviation standard (*SD*) ainsi que des mesures comme le *RX* qui exprime le pourcentage des mesures dont l'erreur est supérieure à X. Pour l'erreur angulaire, on utilise $R2.5$, $R5.0$, $R10.0$ (en degré). Pour l'erreur du point final, on utilise $R0.5$, $R1.0$, $R2.0$ (en pixels), etc.

En se basant sur le tableau de Middlebury qui classe la performance des algorithmes du flux optique, des algorithmes comme NNF-Local [30], PMMST [80], MDP-Flow2 [79],

etc. sont les mieux classés en terme de performance. Cependant, la plupart de ces algorithmes sont itératifs et requièrent beaucoup de temps de calcul. Inversement, les algorithmes les plus rapides comme Rannacher [56], complOF-FED-GPU [37], Bartels [16], PGAM+LK [9], etc., sont généralement implémentés en GPU. Par contre, la plupart de ces algorithmes performent moins bien et sont moins robustes. Généralement, les algorithmes présentés dans le tableau de Middlebury sont ou bien rapides et performent moins bien, ou bien robustes mais requièrent beaucoup plus de temps de calcul.

3.5 Peut-on faire autrement ?

La méthode que nous présentons au chapitre 4, propose de déterminer le flux optique à l'aide de l'extraction des isocontours pour chaque pixel dans les deux images à comparer. Si la mise en correspondance de deux isocontours est réussie, il y a de bonnes chances de déterminer le flux optique entre les deux pixels en question.

Un iosocontour peut être considéré comme un descripteur intéressant pour les pixels qu'il contient. La particularité d'un isocontour est qu'il est souvent placé sur une rampe d'intensités dans l'image à l'inverse d'un contour normal qui décrit un changement brusque d'intensité. Il décrit donc une intensité de couleur bien spécifique. Généralement, le gradient spatial est non nul et il est perpendiculaire à l'isocontour. En utilisant les isocontours, on assure de garder seulement les pixels placés dans des régions ayant des variations d'intensité non négligeables et donc fournissant des informations utiles. On propose également dans notre méthode une solution qui résout chaque pixel séparément sans avoir recours à une optimisation globale ou locale sur un ensemble de pixels voisins, contrairement aux méthodes classiques du flux optique. Ceci garantit un temps de calcul considérablement réduit par rapport aux méthodes présentées dans le tableau de Middlebury [3]. Cela dit, pour densifier le résultat final, on se sert des valeurs du flux des pixels voisins pour remplir les pixels ambigus.

Chapitre 4

ROBUST NON-ITERATIVE OPTICAL FLOW BASED ON ISOCONTOUR MATCHING AND MONOTONIC INTENSITY MODEL

Dans cet article nous présentons une méthode qui détermine le flux optique en se basant sur l'extraction et la mise en correspondance des isocontours dans deux images. Le choix d'utiliser les isocontours part du principe qu'il décrit tous les pixels ayant la même intensité et se plaçant sur une rampe d'intensités par rapport à leurs voisins (c'est souvent le cas sauf si l'isocontour est un minimum local ou un maximum local). Contrairement aux contours normaux qui décrivent un changement brusque d'intensité causant l'échec de nombreux algorithmes du flux optique, les isocontours se placent souvent dans des régions ayant des variations d'intensité plus lisses ce qui rend la recherche du flux plus facile. La détermination du flux commence par l'extraction des isocontours dans les deux images. Par la suite, en partant de la même position dans la deuxième image, on suit la direction de l'intensité cible jusqu'à arriver au prochain isocontour correspondant à la même intensité. Pour s'assurer que les deux isocontours correspondent l'un à l'autre, on procède à une étape de mise en correspondance. Si les isocontours sont semblables, le flux est donc retrouvé. Cette méthode est adaptée à des flux optiques pas trop grands. La méthode permet aussi d'obtenir un flux optique éparse. Il existe plusieurs algorithmes de flux optique éparse comme par exemple [13, 46, 24].

Basé sur le tableau de Middlebury [3], les algorithmes du flux optique qui se placent premiers sont considérablement lents. De même les algorithmes les plus rapides sont généralement moins robustes. L'avantage de notre méthode est de trouver un flux optique robuste dans un temps de calcul court.

Abstract

Most of optical flow algorithms rely on sophisticated optimization approaches to compute a dense optical flow field with a relatively long processing time. Most of these algorithms have been registered in the Middlebury Flow benchmark [3, 14]. In this paper we propose a new approach which robustly estimates a non-dense optical flow. It achieves good performance by not relying on any global optimization. First, the assumption that intensity profiles are locally linear is replaced by locally monotonic, which results in a highly accurate normal flow. In a second step, local isocontours are extracted and matched to provide a final flow. The structure contained in the isocontours can be sufficient to resolve the aperture problem directly, without having to resort to global optimization. This proposed method is thus non-iterative and parallelizable, as it resolves each pixel independently. This makes it fast and well adapted for real-time applications. As our method is non-dense, it can not be compared directly to the Middlebury benchmark. However, while it is one of the fastest methods, its robustness and accuracy compare well with other algorithms.

4.1 Introduction

Optical flow is a fundamental problem in computer vision and robotics. It consists in the estimation of an image motion field that represents the projection of real 3D velocities onto 2D images. It is used in several computer vision applications : motion detection, obstacle detection and tracking, egomotion and visual odometry, etc. Its original gradient-based formulation was initially proposed by Horn and Schunck [40] as well as Lucas and Kanade [47] in the early eighties.

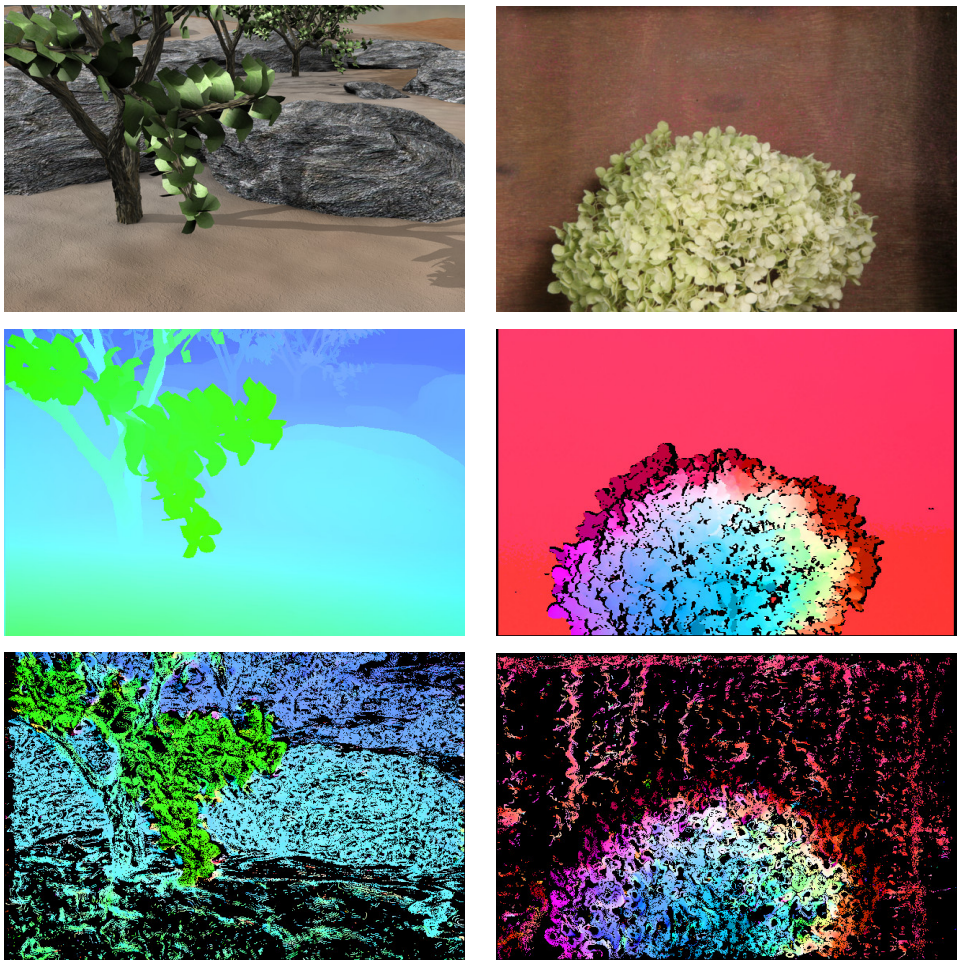


FIGURE 4.1. **Left : "Grove2" image from the Middlebury dataset (top), color-coded groundtruth (middle), and our results (bottom). Right : "Hydrangea" image from the Middlebury dataset (top), colorcoded groundtruth (middle), and our results (bottom).**

Performance evaluations have been proposed by Barron *et al.* [15] and more recently by Baker *et al.* [14] at the Middlebury dataset [3]. Most of the Middlebury benchmark algorithms use heavy optimization methods in order to perform dense flow estimation with a minimal average error making them slow and less adapted to real-time applications. Indeed, as illustrated in the benchmark, runtimes can be over an hour for one pair of images [65]. Very few algorithms run in a few seconds or less, and most of them are implemented

over GPU [16]. As many of the recent applications of optical flow require real-time performance (egomotion, obstacle detection, visual odometry, etc.), it is important to develop a fast and robust optical flow algorithm. Thus, we present in this paper a non-iterative, optimization-free and robust optical flow algorithm. The flow fields it produces are non-dense, as it focuses on reliability and accuracy rather than density. Since we consider that a fast high-quality non-dense flow can be very useful by itself, we leave the optimization to make it dense to future work.

The remainder of this paper is organized as follows. In Section 4.2, we review the existing optical flow estimation algorithms and their related approaches. In Section 4.3, we present our proposed method. In Section 4.4, we propose some changement to the initial approach, we test our method and we discuss the results.

4.2 Related Work

4.2.1 Optical Flow Constraints

In most variational frameworks, optical flow is computed from spatial and temporal derivatives. The two principal characteristics of most optical flow estimation methods is the *brightness constancy assumption* (the "data term"), and some form of smoothness assumption (the "prior term"). The brightness constancy assumption relates the intensity $I(x,y,t)$ of a pixel (x,y) at time t with the intensity of the moved pixel along the flow vector (u,v) at time $t+1$. It is given by :

$$I(x,y,t) = I(x+u, y+v, t+1). \tag{4.1}$$

Assuming that the motion is small, we can model image intensities locally using the gradient. This gives :

$$u\frac{\partial I}{\partial x} + v\frac{\partial I}{\partial y} + \frac{\partial I}{\partial t} = 0. \tag{4.2}$$

This model implies that the intensity varies linearly with the motion which is not always true, especially in the case of large motions. In this work, we propose to replace this linear

model with a monotonic model. Once the normal flow is obtained using this model, we are going to solve the *aperture problem* without regularization by relying on local isocontour extraction and matching.

4.2.2 Most Common Optical Flow Approaches

Horn and Schunck's algorithm [40] was the first to address the problem of optical flow. It combines the brightness constraint term and the smoothness term into a global energy function E_G that must be minimized. The method is considered global because the calculation of any flow vector is based on the entire image. The functional they defined can be minimized by solving the Euler-Lagrange equations. One second popular differential approach is the algorithm proposed by Lucas and Kanade [47]. They assume that the motion between two frames is small and that there is a local spatial coherence. Based on Equation 4.2, they solve the ambiguity related to the aperture problem by relying on the neighbours. The linear system is then solved using the least squares method. In contrast with Horn and Schunck's global method, Lucas and Kanade's algorithm is a local technique. Most differential methods use a coarse-to-fine multiscale approach by building pyramidal images based on downsampling or blurring [10, 17, 33, 22]. It helps handling large displacement motions and avoid selecting local minimums.

According to the Middlebury benchmark [3], algorithms like NNF-Local [30], PMMST [80], MDP-Flow2 [79] and more, are at the top of the ranking in most of measurements. Most of these algorithms are iterative and optimization-based, requiring a lot of processing time (673, 362 and 182 seconds respectively). Fast algorithms like Rannacher [56], complOF-FED-GPU [37], Bartels [16], PGAM+LK [9], and others, are generally performing in less than one second (0.12, 0.97, 0.15 and 0.37 respectively). Most of them are implemented over GPU. They are well adapted for real-time applications, if the required hardware is available. However, in most registered error measurements, they are less precise and less robust. In the end, optical flow algorithms are facing a difficult compromise

between accuracy and speed, and the Middlebury dataset tends to emphasize accuracy over speed.

In contrast, we propose in this paper an optical flow estimation method that is fast and accurate. Its accuracy is high because it only considers a pixel with a motion that satisfies the monotonic intensity model, and presents sufficient local structure. While this makes the flow field non-dense, the computation is non-iterative and easily parallelizable so it can run very fast. It will be discussed in detail in Section 4.3. For instance, a non-iterative method has been proposed by Tao *et al.* [69]. It computes only a sparse set of samples in regions with uniform motion. The method is implemented on a parallel architecture.

4.3 Proposed Approach

We propose an algorithm that estimates a robust and non-dense optical flow based on three principal steps : (1) estimating the normal flow, (2) calculating the intensity isocontours (*local structure*) related to both the pixel at time t and the pixel at time $t+1$, (3) matching the isocontours and determining the final optical flow.

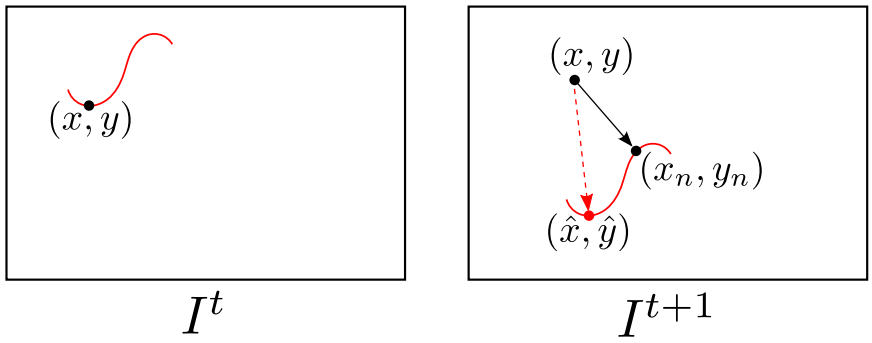


FIGURE **4.2. Overview of the optical flow method.**

Figure 4.2 illustrates the principle of our approach. For a given pixel (x,y) at image I^t, we calculate the normal flow at image I^{t+1} (represented by pixel (x_n,y_n)). We determine the isocontours (red lines in Figure 4.2) for both (x,y) and (x_n,y_n). Isocontours matching

yields the final optical flow $(\hat{x}-x, \hat{y}-y)$. In the following sections, we explain in detail each step of the algorithm. We redefine Equation 4.2 as :

$$f_x u + f_y v + f_t = 0 \tag{4.3}$$

where $f_x = \frac{\partial I}{\partial x}$ is the image derivative along x, $f_y = \frac{\partial I}{\partial y}$ is the image derivative along y. Normal flow vector represents the motion component along the image gradient direction and is defined as

$$\overrightarrow{n} = (f_x, f_y)\frac{f_t}{\sqrt{f_x^2 + f_y^2}}. \tag{4.4}$$

From Equation 4.3, the normal flow constrains the pixel motion on a line, represented as a dashed line in Figure 4.3. The true optical flow $\hat{f}$ is somewhere on this line.

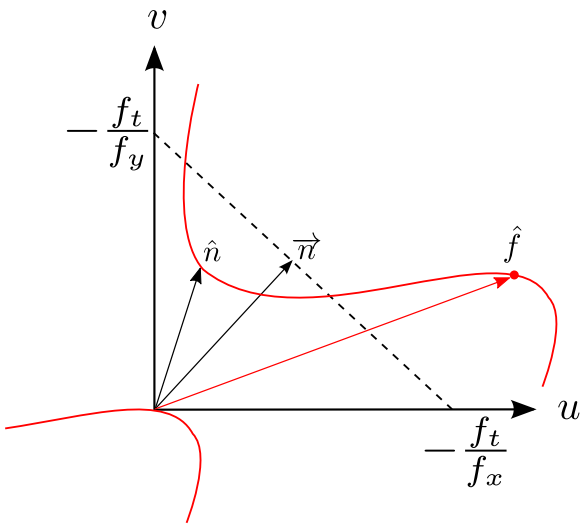


FIGURE **4.3. Real motion range vs. classic optical flow representation of motion range. $\vec{n}$ is the normal flow obtained using the linear gradient (Equation 4.4). $\hat{n}$ is the monotonic normal flow (given by our method). $\hat{f}$ is the optical flow found by our method. The shorter red curve represents the isocontour at I^t. The longer red curve represents the isocontour at I^{t+1}.**

This model has two problems. First, the normal flow $\overrightarrow{n}$ is only accurate if the intensity profile is linear. In practice, this is rarely the case. Second, the motion constraint (dashed

line in Figure 4.3) should have a single intensity. This is also rarely the case in practice. These problems will be solved by using a monotonic intensity model, and by using isocontours, respectively.

4.3.1 Monotonic Intensity Model

We propose an alternative method for finding the normal flow by getting rid of the classical calculation of both spatial and temporal gradients. Instead of assuming a linear intensity profile, we assume a monotonically increasing intensity profile above the isocontour going through the pixel, and a decreasing intensity profile below the isocontour. Essentially, a local intensity profile around a pixel is considered valid as long as the local intensities change with the same sign as the spatial gradient.

Figure 4.4 shows a comparison between the linear gradient approach for normal flow and our monotonic approach, for the case of a translating sinusoid. The red segment represents the *zone of convergence*, where the monotonicity property holds. The solution (small black point on the red curve) is obtained by climbing down along the intensity surface. The linear gradient-based approach provides a wrong solution (black point on dashed line).

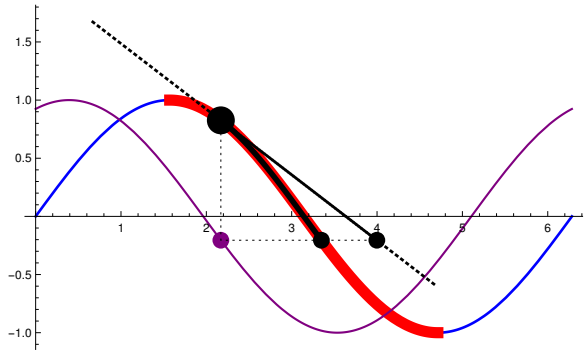


FIGURE **4.4. Monotonic approach vs. linear gradient approach. The starting point is the large black point.**

In our approach, we propose an alternative method for finding the normal flow by getting rid of the classical calculation of both spatial and temporal gradients. It rather follows

the monotonic gradient until finding a pixel greater than or equal to our target (or the inverse depending on the gradient direction we choose to take). Let us consider a pixel $I^t(x,y)$ in image I at time t. Depending on the sign s, defined as $s = \text{sign}(I^{t+1}(x,y) - I^t(x,y))$, we decide we take either the decreasing direction of the gradient or the increasing direction in image I^{t+1}. Starting from pixel $I^{t+1}(x,y)$, we continue following the chosen gradient direction until finding the pixel $I^{t+1}(x',y')$ such as $I^{t+1}(x',y') \geq I^t(x,y)$ (or the inverse, depending on the gradient direction). The vector $(x'-x, y'-y)$ is the normal flow. The next pixel to visit is always taken in the 3×3 neighbourhood, and any pixel with an adequate intensity (moving closer to the target intensity) is a candidate. The selection of one pixel from the candidates can be random, or follow the steepest direction. It does not matter. What matters is that we reach the target intensity, thereby ensuring that we reached a valid normal flow. If the target intensity can not be reached then no optical flow will be computed for that pixel.

$$\begin{pmatrix} 90 & 110 & 130 \\ 90 & 110 & 130 \\ 90 & 110 & 130 \end{pmatrix} \qquad \begin{pmatrix} 110 & 130 & 150 \\ 110 & 130 & 150 \\ 110 & 130 & 150 \end{pmatrix}$$

FIGURE **4.5. Computation of the normal flow.**

Figure 4.5 illustrates a 3×3 kernel of an image being translated. The sign s of the difference between the central pixel is positive ($I^{t+1}(x,y) = 130$ is larger than $I^t(x,y) = 110$). The direction to take is the decreasing gradient. In the decreasing gradient direction of matrix 2, we have three choices. In our method, we use random selection.

Subpixel Accuracy

Since the endpoint intensity of the normal flow overshoots the target and the previous pixel intensity undershoots the target, it is easy to use linear interpolation to get a more precise subpixel position for the normal flow.

4.3.2 Isocontours Extraction

In a more general context, isocontours are also known as *level sets*. They represent the set of variables having the same value provided by a given function f. Level set methods were first proposed by Osher and Sethian [52] for tracking moving 2D and 3D surfaces. They were also used as a tool for analysis of curves, surfaces and shapes as well as for image segmentation [51]. In our context, isocontours represent pixels having the same intensity in the image. Isocontours are used for principally two reasons : First, they replace the motion line of the optical flow model to a more realistic model of motion range. They accurately model the actual motion. As shown in Figure 4.3, the real motion range could be a curve and the classic optical flow representation limits the final solution. Using our method, the normal flow (represented by the small black arrow) ends up at the isocontour instead of at the initial motion range. The second use of the isocontours is to help finding the final solution by matching them. Isocontours are always detected in regions having gradient variations. Thus, a given pixel (x,y) is on an isocontour if it is at a gradient variation. More precisely, let us consider :

$$\begin{aligned} L_c^- &= \{(x,y)|I(x+i,y+j) < c\} \\ L_c^+ &= \{(x,y)|I(x+i,y+j) \geq c\} \end{aligned}$$

where L_c^- is a sublevel of c, L_c^+ is a superlevel of c, and $c = I(x,y)$ is a pixel intensity. A pixel is at a gradient variation if it has a non empty L_c^- and L_c^+. In other terms, a part of its 3×3 neighbours has intensities higher (or equal) than c, and the other part is lower than c. Pixels at the border between L_c^- and L_c^+ are considered at the isocontour of the central pixel. Based on this definition, we determine for each pixel its "isocontour-neighbours". We repeat the same research in both clockwise direction and counter-clockwise direction. Figure 4.6 shows an example of isocontour extraction. The central red ellipse indicates the starting pixels. CW means the clockwise direction of search and CCW means counter-clockwise direction.

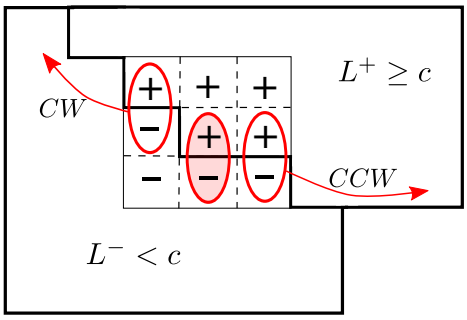


FIGURE **4.6. Example of an isocontour extraction.**

For simplicity and robustness to noise, we only consider pixels on a "gradient slope" where they always have only two groups of neighbours : neighbours having intensities higher or equal to the central pixel, and neighbours having intensities lower than the central pixel. Pixels on more complex and irregular gradient variations are not considered by our isocontour computation method because they are generally unstable and unreliable. As a feasible example, let us consider a 3×3 window. The isocontour related to the central pixel is determined by checking the intensity variations of the neighbours. It is calculated by finding the signs of intensity differences between the central pixel and the neighbours. In Figures 4.7 and 4.8, we show two different situations : a feasible and an unfeasible cases. In the first example, we have a feasible case where the pixel is situated in a region of a "gradient slope". The sub-pixel isocontour path is exactly where signs change. In the second case, we have a more variable gradient around the central pixel. This situation is not handled by our method because of its instability.

$$\begin{pmatrix} 8 & 9 & 9 \\ 8 & 8 & 8 \\ 7 & 1 & 5 \end{pmatrix} \longrightarrow \begin{pmatrix} + & + & + \\ + & 8 & + \\ - & - & - \end{pmatrix}$$

FIGURE **4.7. A feasible case of isocontour extraction.**

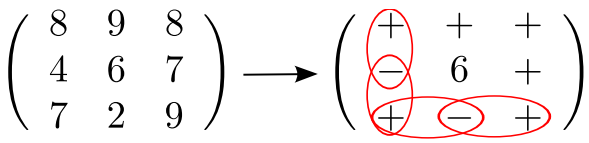


FIGURE **4.8. An unfeasible case of isocontour extraction. We are not on a "gradient slope".**

In order to keep processing time of the algorithm small, we do not extract the entire isocontour for each pixel. We limit the length of the isocontour to a predetermined number of pixels. Isocontours do not have the same length. The first isocontour is shorter than the second. This helps making a more accurate matching by moving toward the second isocontour.

Subpixel Accuracy

As illustrated in Figure 4.6, the initially extracted isocontour is represented at each step by two pixels $+$ and $-$ respectively from superlevel L^+ and sublevel L^-. In order to have a more precise isocontour, we need to make a linear interpolation at each step and extract the final subpixel isocontour. In Figure 4.9, the images of the top row show the isocontours detected in images I^t and I^{t+1} with pixel accuracy. The images from the middle row show the same isocontours with a subpixel accuracy. They are smoother and more precise.

Contour Resampling

The quality of isocontours matching highly depends on their resolution. The smoother they are, the better the matching is. It is then fundamental to perform an isocontour resampling. Figure 4.9 shows in the images from the bottom row a resampling of the subpixel extracted isocontours (the middle row). In this case, the resampling is of the order of half a pixel.

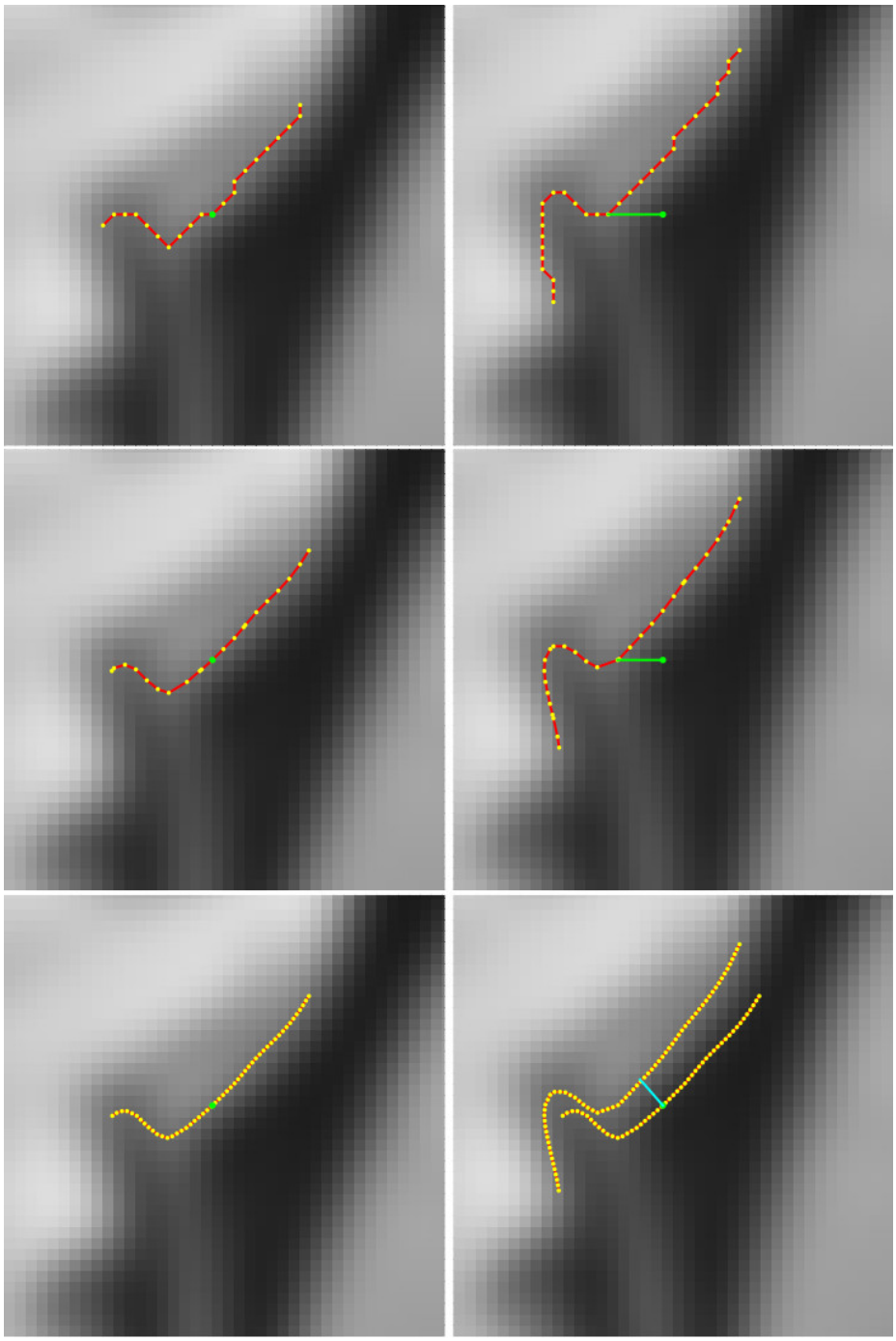


FIGURE 4.9. **Top row :** detected isocontours in images I^t and I^{t+1} with pixel accuracy. **Middle row :** same isocontours with a subpixel accuracy. **Bottom row :** resampled isocontours (from middle row).

4.3.3 Matching the Isocontours

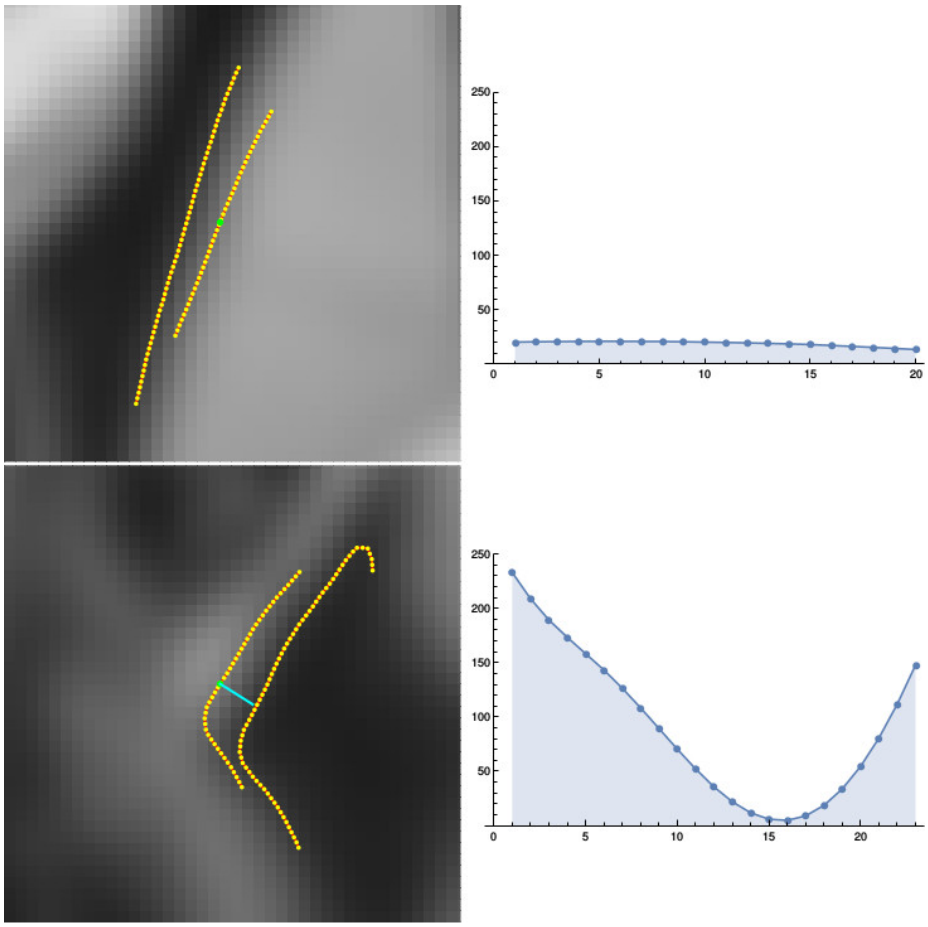


FIGURE **4.10. Two cases of resampled isocontours matching. First case : No match. The cost function is flat. Second case : The cost function gives an optimal solution. There is a match.**

Matching isocontours is equivalent to the problem of sequence alignment. Sequence alignment is a common problem in many fields like bioinformatics, natural language processing, etc. One popular global alignment technique based on dynamic programming is the Needleman–Wunsch algorithm [49]. In our case, the sequences we want to match are the same with an unknown added translation due to the fact that local motion is considered as a pure translation. Resolving such a problem is a simple task. It does not really need

a dynamic programming-based resolution. Instead, we use a simple matching algorithm whose cost function to minimize is the squared euclidean distance. In our case, the second isocontour is always longer than the first. Before making the alignment, we recenter each isocontour by subtracting its mean. Let L_1 and L_2 be the isocontours of the corresponding pixels at respectively I^t and I^{t+1}. Their respective lengths in pixels are $n1$ and $n2$ such that $n1 < n2$. Let $\bar{L_1}$ and $\bar{L_2}$ be the respective means of L_1 and $L_2[i..n1+i]$ at step i. $\hat{L_1} = L_1 - \bar{L_1}$ and $\hat{L_2} = L_2 - \bar{L_2}$ are the recentered isocontours. The cost function C is then obtained at each step i by calculating a distance function between the isocontours. Recentering the isocontours helps having a lower cost function. In order to make our matching robust to ambiguities and degenerated cases, we define some constraints when selecting the best match. The first constraint is that the minimum of the cost function should be a global minimum and should be always lower than a given threshold. The second constraint is that the second minimum of the cost function (if it exists) should be larger enough than the first minimum to avoid ambiguities. Figure 4.10 shows two common cases of matching. In the first, the isocontours are similar. Matching is impossible because the cost function is flat. There is no solution. In the second, the isocontours are easy to match. The cost function has a minimum. A solution has been detected.

4.4 Discussion

Dans le travail présenté dans ce chapitre, nous avons proposé une méthode qui trouve le flux optique en se basant sur l'extraction des isocontours. Dans cette approche nous supposons que l'isocontour est sur une pente d'intensités passant des intensités plus élevées que l'intensité cible vers des intensités plus basses (ou l'inverse). En se basant sur cette hypothèse, nous supposons que le pixel recherché se trouve lui aussi sur une pente d'intensités. L'idée consiste donc à suivre le gradient jusqu'à trouver un pixel d'intensité supérieure ou égale à la cible (ou l'inverse dépendamment de la direction du gradient à prendre). Si l'intensité recherchée n'est pas trouvée ou franchie, nous assumons qu'il n'y a pas de flux optique pour le pixel en question. Si l'intensité recherchée est trouvée, il faut extraire l'isocontour à partir du pixel d'arrivée. Ce qui reste à faire est de mettre en correspondance les deux isocontours et voir s'ils sont semblables ou pas.

4.4.1 Observations

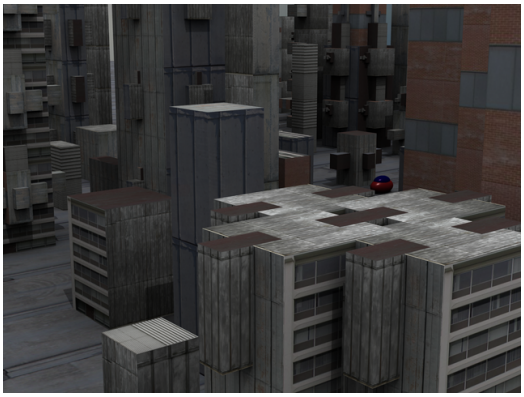


FIGURE **4.11. Example d'image de test de Middleburry : Urban 2.**

Parmi les exemples d'images utilisées pour tester l'algorithme et fournies par Middleburry, il existe de l'ambiguïté spatiale dans certaines régions des images. Par exemple dans les images de *Urban 2* (voir la figure 4.11), au coin droit en bas de l'image il y a un bloc

d'immeuble là où la même structure se répète dans toute la région. Le vrai mouvement de cette région de l'image dépasse 20 pixels. Il s'agit d'un énorme mouvement là où un algorithme classique de flux optique va avoir du mal à fonctionner correctement vu que l'hypothèse principale dans le flux optique suppose que le mouvement est petit.

Notre algorithme de flux optique est basé sur la monotonie spatiale de l'intensité. C'est à dire que si on est à la recherche d'une intensité plus basse, on doit descendre en suivant le gradient jusqu'à atteindre l'intensité cible (ou interpoler si elle n'est pas trouvée) ou l'inverse en cas de montée. Basé sur ce principe, dans l'exemple d'image mentionné précédemment (figure 4.11), vue l'ambiguïté de la région de l'image et les hautes fréquences des intensités spatiales, l'algorithme va s'arrêter à la première intensité cible retrouvée, ce qui ne représente pas forcément la bonne cible. Ceci est supposé être éliminé à l'étape de mise en correspondance des isocontours mais vu que les structures se répètent dans cette région de l'image et aussi vu que la forme des isocontours extraits n'est pas très riche en informations (structure linéaire), la mise en correspondance peut donner une fausse réponse. Il serait très difficile de trouver de tels grands mouvements.

Le problème des grands mouvements concerne presque tous les algorithmes de flux optique. Dans la littérature, la solution proposée à ce problème est d'utiliser les pyramides (*coarse-to-fine approach*). Elle a été déjà expliquée au chapitre 3.

4.4.2 Améliorations apportées à l'algorithme

Initialement, le flux optique est obtenu en partant d'un point dans la première image et en extrayant son isocontour. En suivant le gradient (la direction dépend de la cible), on extrait l'isocontour correspondant dans la deuxième image. Ceci est appliqué sur tous les points. Afin d'optimiser la recherche du flux, nous transformons l'approche de recherche par point, par une approche par contour. Le principe est donc de pré-extraire tous les isocontours des deux images par niveaux d'intensité et de les mettre par la suite en correspondance. L'extraction des isocontours est basée sur l'algorithme *marching squares* [45, 4].

Le principe de l'algorithme *marching squares* est simple. L'image est partionnée en des groupes de quatre pixels voisins à la fois. Pour une intensité I, un groupe de quatre pixels donné représente un des 18 cas présentés à la figure 4.12. En se basant sur ce principe, on extrait pour chaque intensité et pour chaque cas, le segment correspondant. On applique une interpolation linéaire basée sur les intensités des pixels pour trouver la position en sous-pixels du segment du contour. L'isocontour final est obtenu en suivant les segment de contours déjà extraits et en les connectant. Cette étape est appliquée pour chaque composante connexe. En second lieu, la descente de gradient (ou la montée du gradient

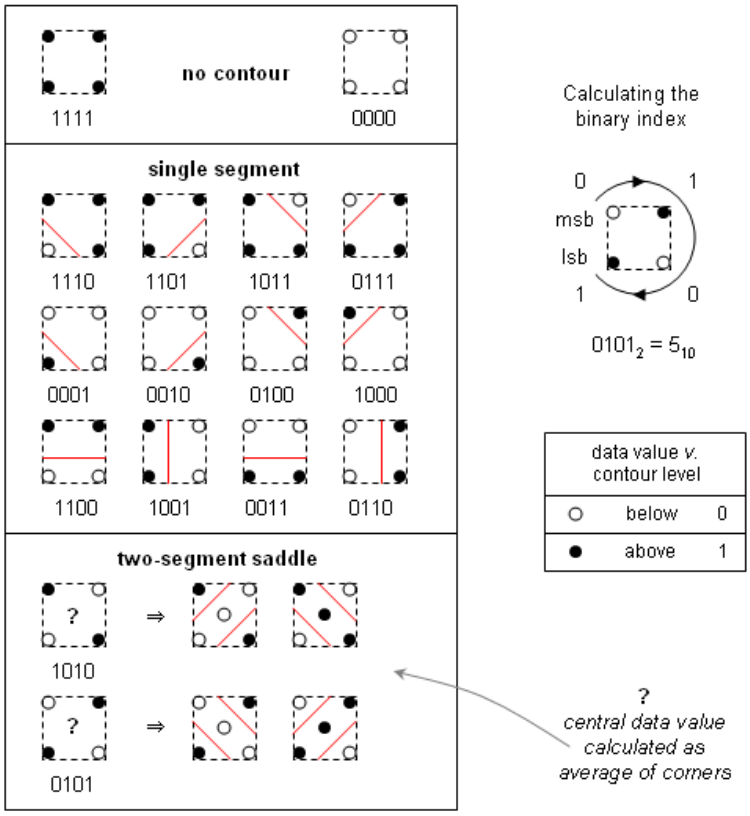


FIGURE **4.12. Les cas possibles des segments de contours dans l'algorithme** *marching squares* **(Tiré de [4]).**

dépendamment de l'intensité ciblée) n'est pas toujours efficace en cas d'image avec de très

hautes fréquences ou aussi en cas de grands mouvements. À la place, nous proposons de comparer l'isocontour de la première image avec ceux de la deuxième image ayant la même intensité et qui sont à une distance pas trop grande (nous avons fixé la distance maximale à 30 pixels dans le cas du *dataset* de Middlebury). Ceci réduit les ambiguïtés relatives à l'approche du suivi du gradient.

La mise en correspondance des isocontours est assurée à l'aide de l'algorithme *dymanic time warping* qui est utilisé pour résoudre les problèmes d'alignement de séquences en bioinformatique, le traitement naturel de la langue, etc. Il est basé sur la programmation dynamique. L'avantage de cet algorithme est qu'il prend en compte les occultations entre deux isocontours. Il est possible de l'adapter (en y ajoutant des contraintes pour qu'il détecte des bonnes correspondances même s'il y a de l'occultation). La figure 4.13 montre un exemple synthétique là où il y a des carrés qui se déplacent dans des directions différentes. Le carré gris se déplace vers le bas et le carré blanc se déplace vers le haut. Les lignes d'isocontours en rouge et en bleu représentent les contours des carrés (en sous-pixels). Les isoncontours affichés varient de l'intensité 0 à 1 avec un pas de 0.1. La figure 4.14 montre deux exemples d'isocontours des deux images qui correspondent l'un à l'autre. Dans le premier exemple, les deux isocontours représentent un isocontour interne du carré blanc. Les deux isocontours ont la même forme et il n'y a pas de problème d'occultation. Le deuxième exemple représente des isocontours qui correspondent à un contour externe représentant la fusion des deux carrés. Comme les deux carrés se déplacent avec des vitesses différentes et dans des directions différentes, la forme de l'isocontour de la deuxième image a changé par rapport au premier. Ceci est un exemple d'occultation.

L'ambiguïté reliée à l'existence d'occultation lors de la mise en correspondance de certains isocontours témoigne de l'instabilité de ceux-ci. Les contours changent parfois de forme et se brisent parfois en des morceaux plus petits (voir la figure 4.15) à cause de la variation du niveau d'intensité d'une image à l'autre. Il est important de détecter ces contours et de les éliminer pour enlever les ambiguïtés de la mise en correspondance.

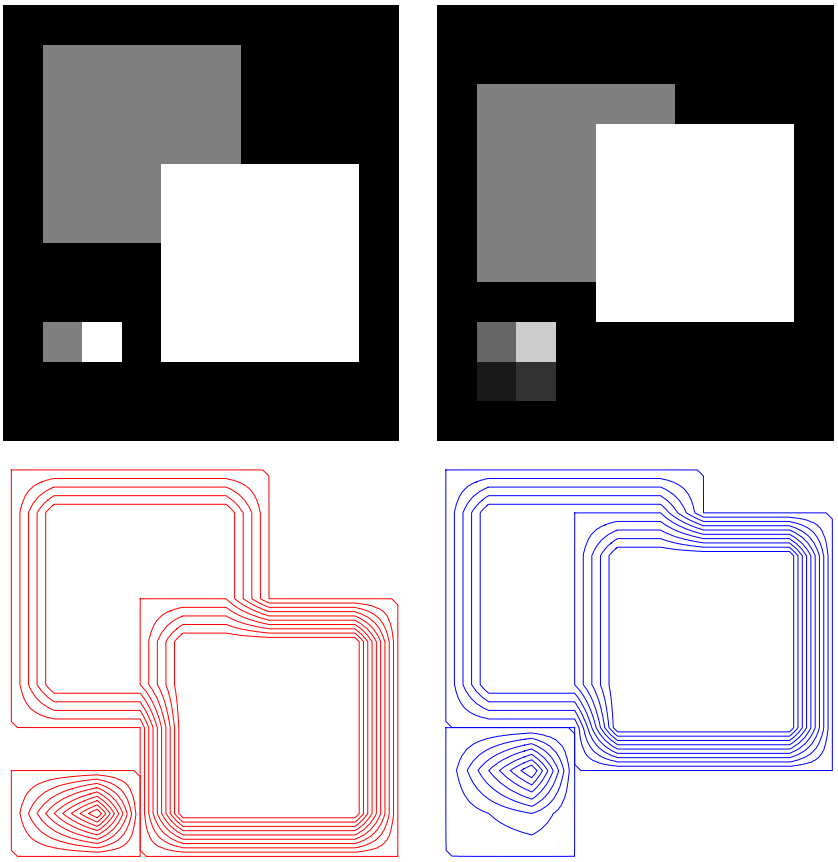


FIGURE **4.13. Exemple synthétique de carrés en mouvement avec la carte des isocontours pour chaque image. Les deux images sont de petite taille (30×30 pixels). Les isocontours bleus et rouges représentent les niveaux de gris entre 0 et 1 avec un pas de 0.1. La distance entre chaque deux isocontours est de l'ordre de 0.1 pixel.**

4.4.3 Expérimentations et résultats

Nous avons testé notre méthode sur les données de Middlebury [3]. Les fonctions de mesure d'erreur sont toutes expliquées dans Baker *et al.* [14]. Nous n'avons pas parallélisé l'implémentation de notre algorithme. Nous n'avons pas implémenté une approche pyramidale pour chercher les grandes vitesses. Nous avons comparé nos résultats avec la méthode de Farneback [35]. Pour chaque exemple, nous calculons pour les deux méthodes l'erreur du point final (*Endpoint Error*) et l'erreur angulaire (*Angular Error*). Dans cet exemple,

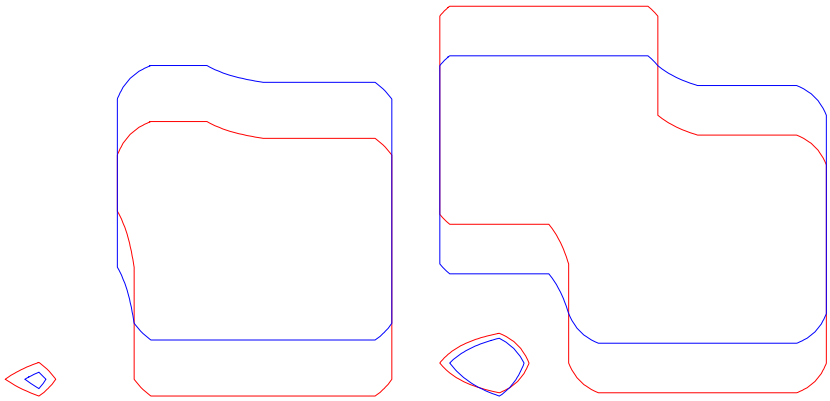


FIGURE **4.14. Deux exemples d'isocontours : Le premier correspond à un isocontour interne du carré blanc. Le deuxième correspond à un isocontour externe qui regroupe les deux carrés.**

FIGURE **4.15. Exemple d'un isocontour correspondant à l'intensité** 0.6 **qui change de forme d'une image à l'autre.**

nous utilisons le flux optique non dense. Dans les exemples illustrés dans le tableau 4.1, les erreurs de notre méthode sont généralement plus petites que celles de la méthode de Farneback. Dans certains cas, la méthode de Farneback fonctionne légèrement mieux (en sous-pixels). En terme de temps d'exécution, la méthode de Farneback est implémentée en GPU. Le temps d'exécution est dans l'ordre de 0.1 sec. Notre méthode n'est pas implémentée en parallèle, elle trouve le flux optique en un temps moyen de 100 sec. Malgré la différence de temps liée au parallélisme, il est sûr que notre méthode peut être accélérée vu qu'elle n'utilise aucune optimisation itérative avec propagation de lissage.

		Erreur du point final	Erreur angulaire
	Farneback Isocontours	0.935 0.826	0.406 0.263
	Farneback Isocontours	0.584 0.773	0.146 0.184
	Farneback Isocontours	1.349 1.241	0.219 0.241
	Farneback Isocontours	0.357 0.605	0.212 0.338
	Farneback Isocontours	1.430 2.290	0.182 0.342
	Farneback Isocontours	2.974 2.234	0.393 0.306
	Farneback Isocontours	1.440 0.834	0.383 0.189

TABLE **4.1. Comparaison de notre méthode avec la méthode de Farneback [35].**

CONCLUSION

Les travaux présentés dans cette thèse visent à faire avancer le domaine de la vision 3D dans l'analyse de mouvement dans une scène dynamique. Nous y avons participé avec deux contributions principales.

En premier lieu, nous nous sommes intéressés au problème du décalage temporel entre les caméras filmant simultanément une scène. Plutôt que de dépendre des solutions matérielles nous avons proposé une solution logicielle pour ce problème.

Nous avons développé en ce sens une méthode basée sur la géométrie épipolaire des caméras et la trajectoire des points en mouvement pour corriger les erreurs dues au décalage temporel entre les caméras. Ceci a garanti une qualité de reconstruction très bonne. Nous avons testé une des principales méthodes d'alignement des séquences vidéo temporellement décalées afin de comparer nos résultats. Les résultats obtenus ont prouvé que la méthode que nous avons développée assure une invariance au décalage temporel entre les caméras et une robustesse par rapport aux variations des fréquences d'acquisition ainsi que pour des délais variables. Comme perspective, il est possible d'utiliser cette méthode pour aligner des séquences vidéo non synchronisées avec un nombre de points suivis plus élevé. Il est également possible de traiter l'ambiguïté reliée au cas d'un mouvement parallèle aux lignes épipolaires. Ce que nous avons proposé dans notre méthode concerne des caméras fixes mais il est possible d'adapter ce travail à des caméras en mouvement. Dans ce cas il est possible de trouver la matrice fondamentale à chaque image en utilisant des points fixes dans la scène. Il est aussi possible d'adapter cette méthode à des systèmes à plus que deux caméras.

En second lieu, tout en restant dans le thème de l'analyse de mouvement, nous nous sommes intéressés au problème du flux optique. En flux optique, il est important de fournir des solutions précises, efficaces et qui fonctionnent en temps réel. Nous avons proposé en

un premier temps un algorithme basé sur l'hypothèse de la monotonie de l'intensité. Par la suite, nous avons modifié notre algorithme pour trouver le flux optique plus facilement dans des situations plus difficiles. Nous avons comparé notre algorithme avec d'autres algorithmes de flux optique notamment avec l'algorithme de Farneback [35]. Les résultats de notre méthode sont très comparables à ceux de Farneback. En terme de temps d'exécution, nous avons expliqué que notre méthode peut être accélérée vu qu'elle n'utilise aucune optimisation itérative avec propagation de lissage. Dans les travaux futurs, nous envisageons d'augmenter la robustesse de notre algorithme en ne sélectionnant que les isocontours stables. De même, nous allons densifier notre algorithme pour trouver le flux optique dans toutes les régions de l'image d'une manière efficace. Trouver le flux optique en utilisant les isocontours a été jugé comme une approche nouvelle et intéressante. Comme toute méthode de flux optique, les perspectives de notre travail sont énormes. Il est possible de combiner notre méthode avec une méthode de reconnaissance de formes pour assurer une mise en correspondance des isocontours plus efficace. De même, il est possible de densifier notre méthode en se servant du flux optique détecté dans les zones des isocontours.

RÉFÉRENCES

[1] http://toothwalker.org/optics/distortion.html.

[2] http://breezesys.com/MultiCamera.

[3] http://vision.middlebury.edu/flow/.

[4] https://en.wikipedia.org/wiki/Marching_squares.

[5] http://www.cs.unc.edu/Research/stc/FAQs/Cameras_Lenses/PtGrey/SyncUnitDocumentation-V1.0.pdf.

[6] https://www.alliedvision.com/en/products/cameras/detail/680.html.

[7] http://www.logitech.com/en-ca/product/hd-pro-webcam-c920.

[8] http://ceres-solver.org/index.html.

[9] Alfonso Alba, Edgar Arce-Santana, et Mariano Rivera. Optical flow estimation with prior models obtained from phase correlation. Dans *Advances in Visual Computing*, volume 6453 de *Lecture Notes in Computer Science*, pages 417–426. Springer Berlin Heidelberg, 2010.

[10] P. Anandan. A computational framework and an algorithm for the measurement of visual motion. *International Journal of Computer Vision*, 2(3) :283–310, 1989.

[11] Alex Arenas, Albert Díaz-Guilera, Jürgen Kurths, Yamir Moreno, et Changsong Zhou. Synchronization in complex networks. *Phys. Rep.*, 469 :93–153, 2008.

[12] Simon Baker, Eric Bennett, Sing Bing Kang, et Richard Szeliski. Removing rolling shutter wobble. Dans *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, June 2010.

[13] Simon Baker et Iain Matthews. Lucas-kanade 20 years on : A unifying framework. *International Journal of Computer Vision*, 56(3) :221–255, 2004.

[14] Simon Baker, Daniel Scharstein, J.P. Lewis, Stefan Roth, MichaelJ. Black, et Richard Szeliski. A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, 92(1) :1–31, 2011.

[15] J.L. Barron, D.J. Fleet, et S.S. Beauchemin. Performance of optical flow techniques. *International Journal of Computer Vision*, 12(1) :43–77, 1994.

[16] C. Bartels et G. de Haan. Smoothness constraints in recursive search motion estimation for picture rate conversion. *Circuits and Systems for Video Technology, IEEE Transactions on*, 20(10) :1310–1319, Oct 2010.

[17] Roberto Battiti, Edoardo Amaldi, et Christof Koch. Computing optical flow across multiple scales : An adaptive coarse-to-fine strategy. *International Journal of Computer Vision*, 6(2) :133–145, 1991.

[18] R. Benrhaiem, S. Roy, et J. Meunier. Real-time software synchronisation of webcams for live 3d tracking. Dans *Image Processing (ICIP), 2015 IEEE International Conference on*, pages 3891–3895, Sept 2015.

[19] Rania Benrhaiem, Sébastien Roy, et Jean Meunier. Achieving invariance to the temporal offset of unsynchronized cameras through epipolar point-line triangulation. *Machine Vision and Applications*, 27(4) :545–557, 2016.

[20] James R. Bergen, P. Anandan, Keith J. Hanna, et Rajesh Hingorani. Hierarchical model-based motion estimation. Dans *Proceedings of the Second European Conference on Computer Vision*, ECCV '92, pages 237–252, London, UK, 1992. Springer-Verlag.

[21] M.J. Black et A.D. Jepson. Estimating optical flow in segmented images using variable-order parametric models with local deformations. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 18(10) :972–986, Oct 1996.

[22] Jean-Yves Bouguet. Pyramidal implementation of the Lucas Kanade feature tracker. *Intel Corporation, Microprocessor Research Labs*, 2000.

[23] Thomas Brox, Andrés Bruhn, Nils Papenberg, et Joachim Weickert. High accuracy optical flow estimation based on a theory for warping. Dans Tomás Pajdla et Jiří Matas, editeurs, *Computer Vision - ECCV 2004*, volume 3024 de *Lecture Notes in Computer Science*, pages 25–36. Springer Berlin Heidelberg, 2004.

[24] Thomas Brox et Jitendra Malik. Large displacement optical flow : Descriptor matching in variational motion estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(3) :500–513, 2011.

[25] Andrés Bruhn, Joachim Weickert, et Christoph Schnörr. Lucas/Kanade meets Horn/Schunck : Combining local and global optic flow methods. *International Journal of Computer Vision*, 61 :211–231, 2005.

[26] Yaron Caspi et Michal Irani. Alignment of non-overlapping sequences. Dans *IEEE International Conference on Computer Vision, July 7-14, 2001, Vancouver, British Columbia, Canada*, pages 76–83, 2001.

[27] Yaron Caspi et Michal Irani. Spatio-temporal alignment of sequences. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24 :1409–1424, 2002.

[28] Yaron Caspi, Denis Simakov, et Michal Irani. Feature-based sequence-to-sequence matching. *Int. J. Comput. Vision*, 68(1) :53–64, 2006.

[29] Haiyang Chao, Yu Gu, et Marcello Napolitano. A survey of optical flow techniques for robotics navigation applications. *Journal of Intelligent & Robotic Systems*, 73(1) :361–372, 2014.

[30] Zhuoyuan Chen, Hailin Jin, Zhe Lin, S. Cohen, et Ying Wu. Large displacement optical flow from nearest neighbor fields. Dans *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 2443–2450, June 2013.

[31] U.R. Dhond et J.K. Aggarwal. Structure from stereo - a review. *Systems, Man and Cybernetics, IEEE Transactions on*, 19(6) :1489–1510, Nov 1989.

[32] Ramani Duraiswami. http://www.umiacs.umd.edu/~ramani/cmsc426/Lecture18_19_motion.pdf.

[33] Wilfried Enkelmann. Investigations of multigrid algorithms for the estimation of optical flow fieldsin image sequences. *Comput. Vision Graph. Image Process.*, 43(2) :150–177, Août 1988.

[34] Wilfried Enkelmann. Obstacle detection by evaluation of optical flow fields from image sequences. *Image and Vision Computing*, 9(3) :160 – 168, 1991.

[35] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. Dans *Proceedings of the 13th Scandinavian Conference on Image Analysis*, SCIA'03, pages 363–370, Berlin, Heidelberg, 2003. Springer-Verlag.

[36] Olivier Faugeras, Quang-Tuan Luong, et T. Papadopoulou. *The Geometry of Multiple Images : The Laws That Govern The Formation of Images of A Scene and Some of Their Applications*. MIT Press, Cambridge, MA, USA, 2001.

[37] Pascal Gwosdek, Henning Zimmer, Sven Grewenig, Andrés Bruhn, et Joachim Weickert. A highly efficient gpu implementation for variational optic flow based on the euler-lagrange framework. Dans KiriakosN. Kutulakos, editeur, *Trends and Topics in Computer Vision*, volume 6554 de *Lecture Notes in Computer Science*, pages 372–383. Springer Berlin Heidelberg, 2012.

[38] Richard Hartley et Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, New York, NY, USA, 2 édition, 2003.

[39] N. Hasler, B. Rosenhahn, T. Thormahlen, M. Wand, J. Gall, et H.-P. Seidel. Markerless motion capture with unsynchronized moving cameras. Dans *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 224–231, June 2009.

[40] Berthold K. P. Horn et Brian G. Schunck. Determining optical flow. *Artificial Intelligence*, 17 :185–203, 1981.

[41] Yumi Kakumu, Fumihiko Sakaue, Jun Sato, Kazuhisa Ishimaru, et Masayuki Imanishi. High frequency 3d reconstruction from unsynchronized multiple cameras. Dans *Proceedings of the British Machine Vision Conference*. BMVA Press, 2013.

[42] Cheng Lei et Yee-Hong Yang. Tri-focal tensor-based multiple video synchronization with subframe optimization. *Image Processing, IEEE Transactions on*, 15(9) :2473–2480, Sept 2006.

[43] Chia-Kai Liang, Li-Wen Chang, et H.H. Chen. Analysis and compensation of rolling shutter effect. *Image Processing, IEEE Transactions on*, 17(8) :1323–1330, Aug 2008.

[44] Ce Liu, Jenny Yuen, Antonio Torralba, Josef Sivic, et WilliamT. Freeman. Sift flow : Dense correspondence across different scenes. Dans David Forsyth, Philip Torr, et Andrew Zisserman, editeurs, *Computer Vision - ECCV 2008*, volume 5304 de *Lecture Notes in Computer Science*, pages 28–42. Springer Berlin Heidelberg, 2008.

[45] William E. Lorensen et Harvey E. Cline. Marching cubes : A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4) :163–169, 1987.

[46] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2) :91–110, 2004.

[47] Bruce D. Lucas et Takeo Kanade. An iterative image registration technique with an application to stereo vision. Dans *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, pages 674–679, 1981.

[48] Amar Mitiche et Patrick Bouthemy. Computation and analysis of image motion : A synopsis of current problems and methods. *International Journal of Computer Vision*, 19(1) :29–55, 1996.

[49] Saul B. Needleman et Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3) :443 – 453, 1970.

[50] Tal Nir, Alfred M. Bruckstein, et Ron Kimmel. Over-parameterized variational optical flow. *International Journal of Computer Vision*, 76(2) :205–216, 2008.

[51] Stanley Osher et Ronald P. Fedkiw. Level set methods : An overview and some recent results. *J. Comput. Phys.*, 169(2) :463–502, Mai 2001.

[52] Stanley Osher et James A Sethian. Fronts propagating with curvature-dependent speed : Algorithms based on hamilton-jacobi formulations. *Journal of Computational Physics*, 79(1) :12 – 49, 1988.

[53] M. Otte et H. H. Nagel. *Optical flow estimation : Advances and comparisons*, pages 49–60. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994.

[54] F.L.C. Padua, R.L. Carceroni, G.A.M.R. Santos, et K.N. Kutulakos. Linear sequence-to-sequence alignment. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(2) :304–320, Feb 2010.

[55] Ying Piao et J. Sato. Computing epipolar geometry from unsynchronized cameras. Dans *Image Analysis and Processing, 2007. ICIAP 2007. 14th International Conference on*, pages 475–480, Sept 2007.

[56] J. Rannacher. Realtime 3D motion estimation on graphics hardware. *Undergraduate Thesis, Heidelberg University*, 2009.

[57] Ian D. Reid et Andrew Zisserman. Goal-directed video metrology. Dans *Proceedings of the 4th European Conference on Computer Vision-Volume II - Volume II*, ECCV '96, pages 647–658, London, UK, 1996. Springer-Verlag.

[58] Stefan Roth et MichaelJ. Black. On the spatial statistics of optical flow. *International Journal of Computer Vision*, 74(1) :33–50, 2007.

[59] S.N. Sinha et M. Pollefeys. Synchronization and calibration of camera networks from silhouettes. Dans *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 1, pages 116–119, Aug 2004.

[60] Sudipta N. Sinha et Marc Pollefeys. Visual-hull reconstruction from uncalibrated and unsynchronized video streams. Dans *Proceedings of the 3D Data Processing,*

Visualization, and Transmission, 2Nd International Symposium, 3DPVT '04, pages 349–356, Washington, DC, USA, 2004. IEEE Computer Society.

[61] F. Sivrikaya et B. Yener. Time synchronization in sensor networks : a survey. *Network, IEEE*, 18(4) :45–50, July 2004.

[62] G.P. Stein. Tracking from multiple view points : Self-calibration of space and time. Dans *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 1, page 527, 1999.

[63] Christoph Stiller, Janusz Konrad, et Robert Bosch. Estimating motion in image sequences - a tutorial on modeling and computation of 2d motion. *IEEE Signal Processing Magazine*, 16 :70–91, 1999.

[64] Deqing Sun, Stefan Roth, J.P. Lewis, et MichaelJ. Black. Learning optical flow. Dans David Forsyth, Philip Torr, et Andrew Zisserman, editeurs, *Computer Vision - ECCV 2008*, volume 5304 de *Lecture Notes in Computer Science*, pages 83–97. Springer Berlin Heidelberg, 2008.

[65] Deqing Sun, Erik B. Sudderth, et Michael J. Black. Layered segmentation and optical flow estimation over time. Dans *In IEEE International Conference on Computer Vision and Pattern Recognition*, pages 1768–1775, 2012.

[66] M. Svedman, L. Goncalves, N. Karlsson, M. Munich, et P. Pirjanian. Structure from stereo vision using unsynchronized cameras for simultaneous localization and mapping. Dans *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 3069–3074, Aug 2005.

[67] Richard Szeliski. *Computer Vision : Algorithms and Applications*. Springer-Verlag New York, Inc., New York, NY, USA, 2010.

[68] Richard Szeliski et Heung-Yeung Shum. Creating full view panoramic image mosaics and environment maps. Dans *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 251–258, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[69] Michael Tao, Jiamin Bai, Pushmeet Kohli, et Sylvain Paris. Simpleflow : A non-iterative, sublinear optical flow algorithm. *Comput. Graph. Forum*, 31 :345–353, 2012.

[70] Philip A. Tresadern et Ian D. Reid. Video synchronization from human motion using rank constraints. *Comput. Vis. Image Underst.*, 113(8) :891–906, Août 2009.

[71] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, et Andrew W. Fitzgibbon. Bundle adjustment - a modern synthesis. Dans *Proceedings of the International Workshop on Vision Algorithms : Theory and Practice*, ICCV '99, pages 298–372, London, UK, 2000. Springer-Verlag.

[72] T. Tuytelaars et L. Van Gool. Synchronizing video sequences. Dans *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 1, pages I–762–I–768, June 2004.

[73] S. Velipasalar et W. Wolf. Frame-level temporal calibration of video sequences from unsynchronized cameras by using projective invariants. Dans *Advanced Video and Signal Based Surveillance, 2005. AVSS 2005. IEEE Conference on*, pages 462–467, Sept 2005.

[74] Anthony Whitehead, Robert Laganiere, et Prosenjit Bose. Temporal synchronization of video sequences in theory and in practice. Dans *Proceedings of the IEEE Workshop on Motion and Video Computing (WACV/MOTION'05) - Volume 2*, WACV-MOTION '05, pages 132–137, Washington, DC, USA, 2005. IEEE Computer Society.

[75] Lior Wolf et Assaf Zomet. Sequence-to-sequence self calibration. Dans *European Conference on Computer Vision*, pages 370–382, 2002.

[76] Lior Wolf et Assaf Zomet. Wide baseline matching between unsynchronized video sequences. *Int. J. Comput. Vision*, 68(1) :43–52, Juin 2006.

[77] Gang Xu et Zhengyou Zhang. *Epipolar Geometry in Stereo, Motion, and Object Recognition : A Unified Approach*. Kluwer Academic Publishers, Norwell, MA, USA, 1996.

[78] Li Xu, Jianing Chen, et Jiaya Jia. A segmentation based variational model for accurate optical flow estimation. Dans *Proceedings of the 10th European Conference on Computer Vision : Part I*, ECCV '08, pages 671–684, Berlin, Heidelberg, 2008. Springer-Verlag.

[79] Li Xu, Jiaya Jia, et Y. Matsushita. Motion detail preserving optical flow estimation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(9) :1744–1757, Sept 2012.

[80] F. Zhang, S. Xu, et X. Zhang. High accuracy correspondence field estimation via MST based patch matching. Dans *Submitted to TIP*, 2015.

[81] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11) :1330–1334, Nov 2000.

[82] Zhengyou Zhang, Rachid Deriche, Olivier Faugeras, et Quang-Tuan Luong. A robust technique for matching two uncalibrated images through the recovery of the unknown epipolar geometry. *Artif. Intell.*, 78(1-2) :87–119, Octobre 1995.

[83] Chunxiao Zhou et Hai Tao. Dynamic depth recovery from unsynchronized video streams. Dans *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 2, pages 351–358, June 2003.

[84] Henning Zimmer, Andrés Bruhn, Joachim Weickert, Levi Valgaerts, Agustin Salgado, Bodo Rosenhahn, et Hans-Peter Seidel. Complementary optic flow. Dans Daniel Cremers, Yuri Boykov, Andrew Blake, et Frank R. Schmidt, editeurs, *Energy Minimization Methods in Computer Vision and Pattern Recognition*, volume 5681 de *Lecture Notes in Computer Science*, pages 207–220. Springer Berlin Heidelberg, 2009.

[85] I. Zoghlami, O. Faugeras, et R. Deriche. Using geometric corners to build a 2d mosaic from a set of image. Dans *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, CVPR '97, pages 420–, Washington, DC, USA, 1997. IEEE Computer Society.

Annexe A

CALCUL ALGÉBRIQUE ET GÉOMÉTRIE ÉPIPOLAIRE

Produit vectoriel (tiré de [38], traduction libre)

À partir du vecteur $a = (a_1, a_2, a_3)^T$ de taille 3, on définit la matrice antisymétrique de dimensions 3×3 du vecteur a comme étant :

$$[a]_\times = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \tag{A.1}$$

$[a]_\times$ est donc la notation d'une matrice antisymétrique issue d'un vecteur a. La matrice $[a]_\times$ est singulière et a est son *vecteur nul*. Le produit vectoriel de deux vecteurs a et b de taille 3, $a \times b$, peut être écrit sous la forme de matrices antisymétriques comme suit :

$$a \times b = [a]_\times b = \left(a^T [b]_\times\right)^T. \tag{A.2}$$

Matrice fondamentale F (tiré de [38], traduction libre)

À partir d'un ensemble de points correspondants $x_i \leftrightarrow x_i'$ des deux images, la matrice F peut être retrouvée en se basant sur

$$x'^T F x = 0. \tag{A.3}$$

Soient $x = (x, y, 1)^T$ et $x' = (x', y', 1)^T$, l'équation A.3 peut être réécrite pour cette paire de points comme suit

$$x'x f_{11} + x'y f_{12} + x' f_{13} + y'x f_{21} + y'y f_{22} + y' f_{23} + x f_{31} + y f_{32} + f_{33} = 0 \tag{A.4}$$

où f est le vecteur à 9 éléments correspondant à F. Ainsi l'équation A.4 peut être réécrite comme suit

$$(x'x, x'y, x', y'x, y'y, y', x, y, 1)f = 0. \tag{A.5}$$

À partir de n points correspondants dans les deux images, on obtient un système d'équations linéaires comme suit

$$Af = \begin{pmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_nx_n & x'_ny_n & x'_n & y'_nx_n & y'_ny_n & y'_n & x_n & y_n & 1 \end{pmatrix}. \tag{A.6}$$

Annexe B

CALCUL DU FLUX OPTIQUE

Flux optique : Méthode de Horn et Shunck [40]

La combinaison du terme de l'erreur 3.3 et du terme de la régularité 3.4 (dans le chapitre 3) forment une fonction à minimiser globalement (sur tous les pixels de l'image). L'équation globale est formulée comme suit :

$$E_{HS}(u,v) = \int_{\Omega} ((I_x u + I_y v + I_t)^2 + \lambda(|\nabla u|^2 + |\nabla v|^2))dxdy \tag{B.1}$$

où Ω est le domaine de l'image, ∇u est le gradient de u et ∇v est le gradient de v. L'équation B.1 est par la suite minimisée en la dérivant par rapport à u et v. Ceci donne deux équations :

$$\begin{aligned} (I_x u + I_y v + I_t)I_x + \lambda(\nabla^2 u) = 0 \\ (I_x u + I_y v + I_t)I_y + \lambda(\nabla^2 v) = 0. \end{aligned} \tag{B.2}$$

où $\nabla^2 u$ est le laplacien de u ($\nabla^2 u = u_{xx} + u_{yy}$) et $\nabla^2 v$ est le laplacien de v ($\nabla^2 v = v_{xx} + v_{yy}$). La forme discrète est par la suite obtenue à partir de la matrice laplacienne. On obtient ainsi $\nabla^2 u = u - \bar{u}$ où $\bar{u}$ est la moyenne locale obtenue à partir de la matrice laplacienne. La version discrète de l'équation B.2 est donc de la forme :

$$\begin{aligned} (I_x u + I_y v + I_t)I_x + \lambda(u - \bar{u}) = 0 \\ (I_x u + I_y v + I_t)I_y + \lambda(v - \bar{v}) = 0. \end{aligned} \tag{B.3}$$

À partir de l'équation B.3, u et v sont de la forme :

$$\begin{aligned} u = \bar{u} - I_x \frac{P}{D} \\ v = \bar{v} - I_y \frac{P}{D} \end{aligned} \tag{B.4}$$

où $P = I_x\bar{u} + I_y\bar{v} + It$ et $D = \lambda + I_x^2 + I_y^2$.

À partir de cette étape, les coordonnées du flux u et v sont déterminées itérativement. Ils sont au début tous initialisés à 0. L'équation B.4 peut être réécrite comme suit :

$$\begin{aligned} u^{k+1} &= \bar{u}^k - I_x \frac{P^k}{D} \\ v^{k+1} &= \bar{v}^k - I_y \frac{P^k}{D} \end{aligned} \tag{B.5}$$

où $P^k = I_x \bar{u}^k + I_y \bar{v}^k + It$.

Flux optique : Méthode de Lucas et Kanade [47]

À partir de l'équation du flux optique :

$$uI_x + vI_y + I_t = 0 \tag{B.6}$$

on forme un système à neuf équations comme suit :

$$\begin{aligned} I_{x1}u + I_{y1}v &= -I_{t1} \\ &\vdots \\ I_{x9}u + I_{y9}v &= -I_{t9}. \end{aligned} \tag{B.7}$$

On peut réécrire cette équation sous une forme matricielle comme suit :

$$\underbrace{\begin{bmatrix} I_{x1} & I_{y1} \\ \vdots & \vdots \\ I_{x9} & I_{y9} \end{bmatrix}}_{\mathrm{A}} \underbrace{\begin{bmatrix} u \\ v \end{bmatrix}}_{\mathrm{w}} = \underbrace{\begin{bmatrix} -I_{t1} \\ \vdots \\ -I_{t9} \end{bmatrix}}_{I_t}. \tag{B.8}$$

Pour trouver $(u,v)^T$, il faut résoudre l'équation B.7 en minimisant la somme des erreurs au carré de toutes les équations. Cette méthode est appelée la méthode des moindres carrés. L'équation à minimiser est comme suit :

$$min \sum_i (I_{xi}u + I_{yi}v + I_{ti})^2. \tag{B.9}$$

Pour résoudre cette équation à deux paramètres, il faut la dériver par rapport à u et v. Ceci donne les deux équations suivantes :

$$\begin{aligned} \sum_i (I_{xi}u + I_{yi}v + I_{ti})I_{xi} = 0 \\ \sum_i (I_{xi}u + I_{yi}v + I_{ti})I_{yi} = 0. \end{aligned} \tag{B.10}$$

Si on décompose les deux équations précédentes, on obtient :

$$\begin{aligned} \sum I_{xi}^2 u + \sum I_{xi}I_{yi}v = -\sum I_{xi}I_{ti} \\ \sum I_{xi}I_{yi}u + \sum I_{yi}^2 v = -\sum I_{yi}I_{ti}. \end{aligned} \tag{B.11}$$

La forme matricielle de l'équation B.11 donne :

$$\underbrace{\begin{bmatrix} \sum I_{xi}^2 & \sum I_{xi}I_{yi} \\ \sum I_{xi}I_{yi} & \sum I_{yi}^2 \end{bmatrix}}_{\text{A}} \begin{bmatrix} u \\ v \end{bmatrix} \begin{bmatrix} -\sum I_{xi}I_{ti} \\ -\sum I_{yi}I_{ti} \end{bmatrix}. \tag{B.12}$$

On inverse la matrice A et on obtient finalement les équations de u et v comme suit :

$$\begin{aligned} u = \frac{-\sum I_{yi}^2 + \sum I_{xi}I_{ti} + \sum I_{xi}I_{yi} + \sum I_{yi}I_{ti}}{\sum I_{xi}^2 \sum I_{yi}^2 - (\sum I_{xi}I_{yi})^2} \\ v = \frac{\sum I_{xi}I_{ti} \sum I_{xi}I_{yi} - \sum I_{xi}^2 \sum I_{yi}I_{ti}}{\sum I_{xi}^2 \sum \sum I_{yi}^2 - (\sum I_{xi}I_{yi})^2}. \end{aligned} \tag{B.13}$$