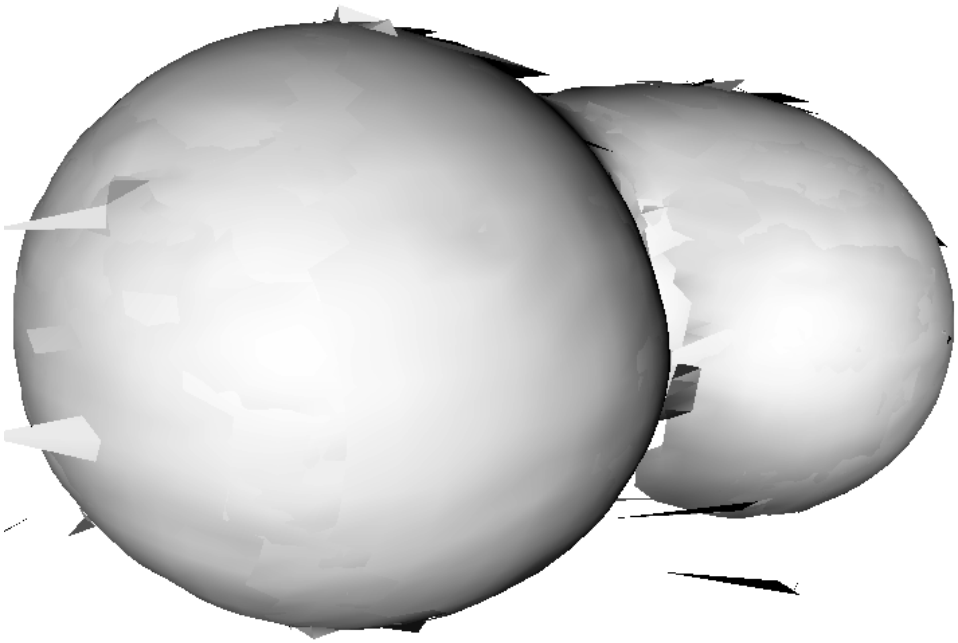
Thèse présentée pour obtenir le grade de
DOCTEUR DE L'ÉCOLE POLYTECHNIQUE

spécialité
ALGORITHMIQUE

par
Frédéric DEVERNAY

Vision stéréoscopique et propriétés différentielles des surfaces



Soutenue le 10 Février 1997 devant le jury composé de :

M. Stéphane MALLAT	Président
M. Olivier FAUGERAS	Directeur de thèse
M. Michael BRADY	Rapporteurs
M. Pascal FUA	
M. Nicholas AYACHE	Examinateurs
M. Rachid DERICHE	
M. Laurent RENOUARD	

Seconde édition, du 3 mai 2006.

Remerciements

Puisqu'on a traditionellement droit à une page de remerciements, autant la remplir, non ?
Je remercie :

Olivier Faugeras, qui a réussi à me guider dans mes recherches, malgré ma forte entropie, et à me donner toute la motivation nécessaire pour arriver à la fin de cette épreuve.

Gaëlle, pour ces derniers mois et les prochaines années.

Les membres du jury et rapporteurs de cette thèse, pour leur présence lors de la soutenance et leurs encouragements.

La DRET, pour avoir financé en grande partie cette thèse. Après tout, n'est-ce pas une manière élégante d'utiliser le budget de la Défense Nationale ?

L'INRIA, formidable structure d'accueil pour un doctorant, ses équipes techniques ultra-compétentes et disponibles (plus particulièrement le SEMIR et la documentation), et ceux qui sont passés en même temps que moi dans le projet Robotvis.

ISTAR, et particulièrement Laurent Renouard, qui m'a accordé sa confiance en acceptant de m'offrir un emploi à plusieurs mois de la fin de cette thèse.

Steak, mes parents, et toute ma famille.

Mes amis : Karim (et la bande des joyeux fêtards), Kristion et Lætitia (Baden-Baden, on y retournera !), Sylv' (pote) et Clem' (potière), Stéphane et Angélique, Jean-No « ça calme » et Drine, Mitch, Odile, Cyril et Valérie, Théo, Stéphane et Gilou (je les mets ensemble juste pour ne pas qu'on confonde les deux Stéphanes, n'allez pas y chercher autre chose), et j'enoublie-un-paquet-c'est-sûr-tant-pis, pour toute leur aide et les bons moments passés ensemble.

Mes colocataires successifs pour avoir supporté mes humeurs et ma musique (forte).

Lewis Trondheim, pour les formidables aventures de Lapinot, et pour l'autorisation d'utiliser une case du *Crabar de mammouth*.

Pour la musique, Leonard Cohen, Pixies, Monochrome Set, They Might Be Giants, Beach Boys, the Sneetches, House Of Love, My Bloody Valentine, Cardinal, Massive Attack, Murat, Love, Papas Fritas, Baby Bird, The Auteurs, ¡Esquivel!, Katerine, Jean Bart, Combustible Edison, Ween,the Walkabouts, Tindersticks, Ron Sexsmith, Sonic Youth, ...

Pour la BD, Joe Matt, Lewis Trondheim, Peter Bagge, Jodorowski, Nicolas de Crecy, Jean-Christophe Menu, Robert Crumb, Art Spiegelman.

Table des matières

Introduction

La vision par ordinateur est un domaine de recherche qui a au plus une vingtaine d'années, et pourtant les applications sont déjà nombreuses et ne cessent de se renouveler. Des réalisations utilisant la vision sont apparues très tôt dans le domaine de la robotique et utilisaient le plus souvent des fonctions de bas niveau proches du traitement d'images (extraction de primitives simples de l'image de type contours ou segments, détection de défauts) et de la reconnaissance de formes. Les capteurs ont ensuite gagné en précision, et la recherche s'est progressivement intéressée au formalisme géométrique lié à la structure des caméras et à leur calibrage, cet intérêt pour la géométrie culminant lors de la (re)découverte de l'auto-calibrage. Les premières applications pouvant rivaliser avec les résultats qu'obtiennent depuis déjà près d'un siècle les photogrammètres à partir de photo-restituteurs sont alors apparues.

Un nouveau champ d'applications fait aujourd'hui son apparition avec le développement explosif du graphisme par ordinateur ou de l'infographie *(computer graphics)* et de la réalité virtuelle *(virtual reality)*, poussés par les industriels de l'informatique qui y voient le meilleur moyen de valoriser leurs progrès en termes de puissance de processeurs et de moteurs graphiques. Cependant, pour pouvoir synthétiser de manière réaliste des objets tels que terrains, villes, bâtiments, objets de la vie quotidienne, ou même des êtres en mouvement, il faut bien évidemment passer par des phases de modélisation et de numérisation qui, lorsqu'elles sont effectuées manuellement (à partir de plans, de cartes, ou avec un modeleur), restent longues et fastidieuses. La modélisation consiste en la conception d'un modèle mathématique permettant de représenter le plus fidèlement possible un objet et son comportement mécanique. Un modèle peut être très simple dans le cas d'une tasse (l'union d'un cylindre et d'un tore) mais devient vite compliqué dans le cas d'êtres vivants (on devra modéliser des surfaces d'aspect complexe, la souplesse des tissus, et les multiples degrés de liberté des articulations) ou de scènes complètes (dans lesquelles les relations entre les objets seront aussi à prendre en compte). La numérisation, quant à elle, consiste à trouver les paramètres d'un modèle générique qui permettent de représenter une instance particulière de l'objet (une tasse à café ou un visage par ex.). Une solution satisfaisante au problème de la numérisation, qu'elle

soit complètement automatique ou seulement assistée par ordinateur, passe évidemment par la vision par ordinateur, et la vision stéréoscopique est sans doute le moyen le plus rapide et le plus fiable de construire un modèle de surface tridimensionnelle correspondant à un objet quelconque, uniquement à partir d'images.

La vision stéréoscopique n'est cependant pas la panacée, et quelques précautions sont encore nécessaires pour pouvoir obtenir des modèles tridimensionnels qui puissent être utilisables dans des contextes exigeants comme l'infographie. En effet, l'insertion par un infographiste d'un objet réel dans une scène synthétique n'exige pas les mêmes qualités d'un modèle tridimensionnel que l'évitement d'obstacles par un robot : le graphiste aura besoin d'une représentation dense, visuellement satisfaisante, et cependant minimale, alors que le robot pourra se satisfaire de quelques points répartis dans le champ visuel, pourvu qu'ils ne comportent pas d'erreur grossière ou d'obstacle oublié, l'aspect de l'environnement ainsi reconstruit important peu.

Dans ce document, on s'intéresse essentiellement aux méthodes permettant de résoudre en partie le problème du graphiste : construire une représentation tridimensionnelle la plus précise possible. Dans cette optique, toutes les étapes permettant à partir d'une ou plusieurs paires d'images stéréoscopiques d'obtenir un modèle tridimensionnel ont été abordées :

- dans le chapitre 1, on commence par aborder le douloureux problème du calibrage, qui consiste à calculer les paramètres des caméras (focale, centre optique, etc.) ainsi que leur position, soit à partir d'images d'objets de géométrie et de position connue, soit de manière automatique (on parle alors d'auto-calibrage). Quel modèle de caméra utiliser ? Est-il impératif d'utiliser une mire de calibrage ? De quelles informations sur un système de caméras a-t-on besoin pour faire de la vision stéréoscopique ? Des résultats nouveaux sont présentés sur l'auto-calibrage de la distorsion optique d'une caméra et sur l'auto-calibrage d'une paire de caméras rigidement liées à partir de plusieurs paires d'images ;
- c'est dans le chapitre 2 qu'on explique la phase de rectification des images. Cette transformation des paires d'images permet de simplifier énormément le processus de mise en correspondance stéréoscopique par corrélation. On essayera surtout de mieux comprendre la rectification en l'examinant selon différents points de vue, géométriques ou algébriques ;
- au cours du chapitre 3, on entre dans le cœur du problème, avec une description détaillée de la stéréoscopie par corrélation. Un algorithme optimisé classique est d'abord expliqué dans les détails, et on propose quelques applications dérivées (images en couleurs, exécution sur une machine parallèle). Ensuite est décrite une méthode de stéréoscopie par corrélation permettant d'obtenir des résultats de loin meilleurs, tout en calculant à partir des images les orientations et courbures d'une

surface. Quelques résultats permettent d'apprécier les performances de cette nouvelle méthode ;
- le chapitre 4 contient les formules permettant, à partir du résultat de la stéréo par corrélation, de reconstruire en trois dimensions une surface ou une scène entière, avec éventuellement la normale et les courbures en chaque point ;
- enfin, quelques applications réalisées au cours de ces recherches sont présentées au chapitre 5.

En annexe, on trouvera quelques uns des articles publiés au cours de cette thèse, ne faisant pas double emploi avec le corps de celle-ci, ainsi que le code source C de quelques routines clés.

Bien que de nombreux domaines aient été abordés au cours de cette thèse, l'objectif qui aurait consisté a construire des modèles utilisables directement en infographie n'est pas encore atteint : il reste à explorer les méthodes permettant de réduire la quantité d'information présente dans les reconstructions présentées au chapitre 4, pour les rendre effectivement utilisables pour ce type d'application, et cette étape est au moins autant du ressort de la géométrie algorithmique que de la vision par ordinateur.

Connaissances requises. Pour une meilleure compréhension il est nécessaire d'avoir acquis quelques notions de géométrie projective. Une bonne introduction à la géométrie projective appliquée à la vision par ordinateur peut être trouvée dans [MZ92, Fau93]. La lecture du second de ces ouvrages est d'autant plus conseillée qu'il traite également des concepts de base développés au cours de cette thèse (calibrage, stéréoscopie par corrélation, extraction de contours), et que l'auteur est mon honorable directeur de thèse.

Notations. Au cours de ce texte, nous essaierons de respecter les règles suivantes de notations :
- l'ensemble des nombres réels est noté $\Re$, la droite projective $\mathcal{P}^1$, le plan projectif $\mathcal{P}^2$, et l'espace projectif $\mathcal{P}^3$;
- le sigle $\cong$ signifie « est congruent à », c'est-à-dire que si $\mathbf{a}$ et $\mathbf{b}$ sont deux éléments de $\mathcal{P}^1$, $\mathcal{P}^2$ ou $\mathcal{P}^3$, ou des applications d'un de ces ensembles vers l'autre, $\mathbf{a} \cong \mathbf{b}$ si et seulement si il existe $\alpha \in \Re\backslash\{0\}$ tel que $\mathbf{a} = \alpha\mathbf{b}$;
- les points de l'espace tridimensionnel sont notés en *majuscules italiques*, p.ex. M ;
- les points d'un plan image sont notés en *minuscules italiques*, p.ex. m ;
- la notation $\mathbf{P}^T$ signifie « transposée de $\mathbf{P}$ » (notation anglo-saxonne).
- le vecteur colonne de l'espace projectif ou euclidien associé respectivement à un point 3-D ou à un point image est noté *par la même lettre, en gras*, p.ex. $\mathbf{M}$ ou $\mathbf{m}$, et un vecteur ligne s'ecrit en notation transposée, p.ex. $\mathbf{m}^T$;

- $\mathbf{a} \wedge \mathbf{b}$ désigne le produit vectoriel de $\mathbf{a}$ et $\mathbf{b}$;
- une application linéaire est confondue avec la matrice qui lui est canoniquement associée et est notée en *majuscules grasses*, p.ex. $\mathbf{P}_1$;
- les matrices sont notés sous leur forme développée *entre crochets*, p.ex. $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$, et un vecteur colonne peut indifféremment être noté sous sa forme développée (x,y) ou $\begin{pmatrix} x \\ y \end{pmatrix}$;

Avertissement. Pendant l'élaboration et la rédaction de cette thèse, aucun animal n'a eu à subir d'expérimentation. Seuls une plante (figure A.2, page 117), un buste (figure 3.11, page 69), une cyclide (figure B.2, page 126), et un Hervé (figure 3.12, page 70) ont été objets d'expériences plus ou moins maléfiques (équation B.4, page 125). Aux dernières nouvelles, tous sont encore en état de marche (sauf le buste, à qui il manque une paire de... jambes), et aucun n'a eu à se plaindre de mauvais traitements.

Chapitre 1

Calibrage du système de caméras

I can't see anything at all, all I see is me
that's clear enough
and that's what's important, to see me

my eyes can focus
my brain is talking
looks pretty good to me

Sonic Youth, Eric's Trip

Calibrer une caméra consiste à déterminer de manière analytique la fonction qui associe à un point de l'espace tridimensionnel sa projection[1] dans l'image donnée par la caméra. Il s'agit bien entendu d'un problème essentiel de la vision artificielle dès qu'on s'intéresse à la géométrie *dans l'espace* de la scène observée à travers une ou plusieurs caméras, puisque dans ce cas on cherche la position dans l'espace des points dont on connaît uniquement l'image par la projection sur la ou les caméras. Cette fonction de projection n'est en général pas quelconque et appartient à une famille de fonctions liée au modèle de caméra choisi. Il est donc d'abord nécessaire pour pouvoir calibrer correctement que ce modèle de caméra corresponde effectivement le plus précisément possible aux appareils utilisés dans le système de vision. Si les appareils présentent des particularités ou des défauts dont le modèle ne peut rendre compte, par exemple si l'on utilise un modèle linéaire de type sténopé pour une caméra munie d'un objectif très grand angle de type « fish-eye » [BL93] comportant de nombreuses distorsions non linéaires, alors les algorithmes reposant fortement sur le modèle de caméra tels que ceux de stéréoscopie ne pourront pas être appliqués, ou bien les ré-

1. La projection est considérée comme ponctuelle, et on ne tient pas compte des problèmes de flou d'origines diverses [Pér91].

sultats comporteront de très nombreuses erreurs et le système risque d'avoir un comportement semblable à celui décrit en tête de chapitre...

Le cas qui nous intéresse plus particulièrement lors de cette étude est celui du calibrage[2] d'un *système stéréoscopique*, c'est-à-dire d'un ensemble de deux caméras rigidement liées observant la même scène, à partir de paires d'images stéréoscopiques. Nous n'explorerons pas les méthodes utilisant le mouvement, c'est-à-dire utilisant des images prises avec la même caméra à des instants très proches, ou des séquences d'images, et pour des références concernant des méthodes s'appliquant à ces situations, on se reportera aux thèses récentes de Zeller [Zel96] et Laveau [Lav96]. Un système stéréoscopique peut *a fortiori* être considéré comme un ensemble de deux caméras indépendantes, et on peut donc pour effectuer le calibrage du système simplement appliquer pour chacune des caméras une méthode classique de calibrage monoculaire utilisant une ou plusieurs images d'une mire ou d'un objet de calibrage de géométrie connue, ce qu'on appelle calibrage *fort*, par opposition au calibrage *faible* que nous verrons plus loin. Cependant, ce type de méthode utilisant un objet de calibrage, il ne peut s'appliquer au cas où l'on ne dispose d'aucune image d'objet de géométrie connue. Dans cette dernière situation en effet, n'ayant aucune connaissance sur la scène observée à travers le système, il devient impossible de calibrer chaque caméra indépendamment de l'autre à partir d'une seule image (chaque image considérée séparément, sans information a priori sur son contenu, ne peut nous renseigner sur la la fonction de projection liée à une caméra), et il faut donc se résoudre à utiliser des méthodes tenant compte du fait qu'on doit calibrer non pas deux caméras complètement indépendantes mais deux caméras *rigidement liées* et observant la *même* scène. On peut alors, à partir de points mis en correspondance entre les deux vues, calibrer le système.

L'une de ces méthodes, appelée calibrage *faible*, permet de déterminer les paramètres ayant uniquement trait à la structure *stéréoscopique* du système, appelée *géométrie épipolaire*. Dans le contexte du modèle de caméra de type sténopé, elle est aussi appelée calibrage projectif, et a été introduite par Longuet-Higgins [LH81]. Comme nous le verrons, cette méthode possède l'avantage énorme sur le calibrage fort des deux caméras *indépendamment* l'une de l'autre de permettre une détermination beaucoup plus précise de la géométrie épipolaire. La précision de la géométrie épipolaire est souvent importante, et elle est même le point crucial conditionnant le succès et la précision des algorithmes de stéréoscopie reposant sur la connaissance *a priori* de cette géométrie, comme par exemple ceux présentés dans les chapitres suivants.

2. On a souvent tendance à utiliser en français le terme de *calibration* au lieu de *calibrage*. Pourtant un seul est dans le dictionnaire, je vous laisse deviner lequel.

1.1 Calibrage fort

Le principe du calibrage dit *fort* est d'utiliser une grille ou tout autre objet de calibrage, pour lequel les positions de points marqués sur l'objet, appelés points de contrôle, sont connues. Ces points de contrôles peuvent être des coins [Tos87], des points [Wil94], des intersections de droites [BMB93] ou toutes autres primitives pouvant être facilement extraites à partir d'images numériques.

Typiquement, le problème du calibrage devient alors : étant donné un ensemble de points de contrôle de coordonnées 3-D connues (X_i,Y_i,Z_i), déterminer les paramètres de la fonction de projection associée au capteur pour que leur projection correspondent au mieux aux mêmes points extraits des images (x_i,y_i). Nous allons dans la suite de cette section faire un bref tour d'horizon des principales méthodes de calibrage fort existant. Pour une synthèse plus détaillée des résultats obtenus dans ce domaine extrêmement actif, on se reportera par exemple à [Sla80, LT88, Zel96].

La fonction de projection peut être décomposée en deux transformations distinctes : la première est la transformation entre l'origine de l'espace 3-D et le système de coordonnées de la caméra – ce déplacement 3-D (rotation et translation) forme les paramètres *extrinsèques* de calibrage – et la seconde est la transformation entre les coordonnées des points 3-D dans le repère attaché à la caméra et les points 2-D dans le plan image de la caméra – transformation dont les paramètres (facteur d'aspect de l'image, distance focale, et d'autres) sont appelés *intrinsèques*.

Les paramètres intrinsèques dépendent du modèle de caméra utilisé. Dans le cas d'un modèle orthographique ou affine, les rayons optiques sont orthogonaux au plan image et il n'y a que 3 paramètres intrinsèques correspondant à un échantillonage spatial du plan image. Le modèle de caméra le plus couramment utilisé est le modèle perspectif linéaire, aussi appelé modèle projectif ou modèle sténopé. Il inclus deux paramètres intrinsèques supplémentaires correspondant à la position du point principal (intersection de l'axe optique avec le plan image) dans l'image [FT87]. Cependant, pour beaucoup d'applications qui exigent une haute précision, ou dans le cas où des optiques bon marché ou de type « grand angle » sont utilisées, le modèle linéaire n'est pas suffisant. Il devient alors nécessaire de prendre en compte les phénomènes de distorsion optique non linéaire et d'introduire plus de paramètres pour prendre en compte cette distorsion [Tsa87, LT88, Bey92]. L'utilisation d'une méthode de calibrage basée sur un modèle sténopé avec des optiques comportant une distorsion non négligeable peut résulter en des erreurs de calibrage importantes.

Les paramètres de distorsion sont le plus souvent couplés avec les autres paramètres intrinsèques de la caméra (p.ex. le centre optique peut être le même que le centre de distorsion), mais il est possible d'utiliser un modèle de caméra dans lequel ceux-ci sont complètement découplés, permettant

alors un calibrage indépendant ou en plusieurs étapes. Le découplage des paramètres de distorsion et des autres paramètres intrinsèques peut être équivalent à ajouter des degrés de liberté au modèle de caméra (par ex. le fait de distinguer le centre optique du centre de distorsion est équivalent, comme nous le montrons annexe A, à ajouter des paramètres de distorsion de décentrage).

Le problème principal commun aux méthodes utilisant des points de contrôle est dû au fait qu'il existe un couplage entre les paramètres intrinsèques et extrinsèques, qui a pour conséquence des erreurs importantes sur les paramètres intrinsèques de la caméra [WCH92].

Nous allons passer en revue quelques méthodes de calibrage utilisant un modèle de caméra perspectif. Une caméra perspective effectue une projection *centrale* du monde sur une surface (une variété à 2 dimensions de l'espace munie d'une carte) appelée *surface rétinienne*. D'autres types de projection sont pris en compte par les modèles orthographique, para-perspectif [Lav96], ou push-broom [HG94]. On montre facilement que la projection centrale sur une surface quelconque est équivalente à une projection sur un plan suivie d'une transformation (ou changement de coordonnées) dans ce plan (ce changement de coordonnées pouvant être une fonction quelconque). Le modèle de caméra perspective le plus simple est donc une projection centrale sur un plan 3-D appelé *plan rétinien* suivie d'un changement de coordonnées linéaire dans le plan 2-D associé : c'est le modèle projectif linéaire aussi appelé sténopé (l'appareil sténopé est un appareil photographique à trou d'épingle). L'application d'une transformation non linéaire au plan image est équivalent à inclure dans le modèle une distorsion non linéaire, et tous les autres modèles perspectifs tombent donc dans cette catégorie.

1.1.1 Modèle projectif linéaire ou sténopé

La plupart des méthodes de calibrage reposent sur le modèle projectif linéaire. En effet, non seulement il permet de modéliser fidèlement la plupart des capteurs projectifs, mais en plus il permet de simplifier les mathématiques mises en jeu pour l'estimation des paramètres du modèle.

Descriptif du modèle sténopé

La figure 1.1 montre les différents éléments du modèle : La caméra est représentée par un plan rétinien Π^r aussi appelé plan image et un centre optique (ou centre de projection) C qui n'appartient pas à Π^r. l'image d'un point M de l'espace est l'intersection de la droite (CM) avec le plan rétinien Π^r. Le plan Π^f parallèle à Π^r passant par C est appelé plan focal. La projection orthogonale de C sur Π^r, c, est appelée point principal , la droite (Cc) est appelée axe optique, et la distance $f = Cc$ est appelée distance focale.

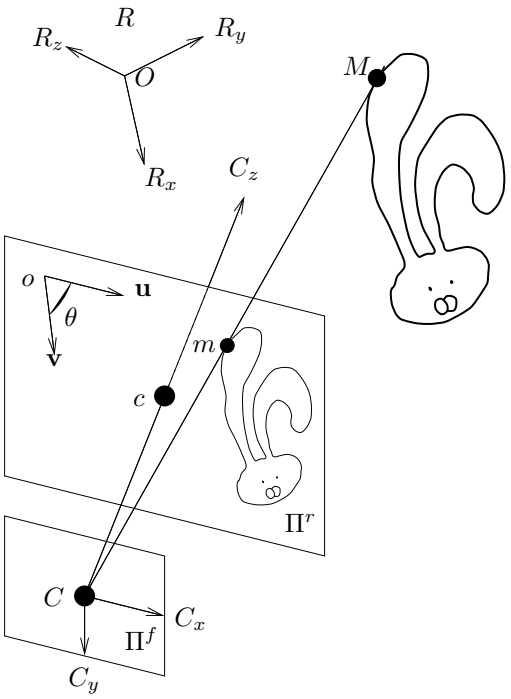


FIG. 1.1 – *Le modèle de caméra sténopé ou projectif linéaire.*

Soit $\mathcal{R}(O,R_x,R_y,R_z)$ un repère orthonormé de l'espace euclidien tridimensionnel $\Re^3$ et $(o,\mathbf{u},\mathbf{v})$ un repère affine du plan rétinien Π^r, la fonction de projection d'un point de l'espace M se décompose alors en trois parties : un changement de repère de l'espace, une projection, et un changement de repère dans le plan image.

Les paramètres extrinsèques de la caméra, qui expriment la position et l'orientation de la caméra dans l'espace lors de la prise de vue, se décomposent en une rotation (représentée par une matrice 3×3, $\mathbf{R}$) et une translation (représentée par un vecteur de dimension 3, $\mathbf{t}$) permettant de passer du repère $\mathcal{R}$ au repère $\mathcal{R}_c$ associé à la caméra. Le repère $\mathcal{R}_c = (C,C_x,C_y,C_z)$ a pour origine le centre optique C, pour axe des z l'axe optique $C_z = (Cc)$. Le plan $\Pi^f = (O,C_x,C_y)$ est parallèle au plan rétinien Π^r, et l'axe (CC_x) est parallèle au vecteur $\mathbf{u}$ du plan rétinien. Soient $\mathbf{M}_R$ les coordonnées de M dans $\mathcal{R}$ et $\mathbf{M}_c$ ses coordonnées dans $\mathcal{R}_c$, on a alors :

$$\mathbf{M}_c = (X_c,Y_c,Z_c) = \mathbf{R}\mathbf{M}_R + \mathbf{t} \tag{1.1}$$

La projection de M sur Π^r a pour coordonnées dans $\mathcal{R}_c$:

$$\left(f\frac{X_c}{Z_c},f\frac{Y_c}{Z_c},f\right) = (x,y,f) \tag{1.2}$$

Le passage en coordonnées pixels se fait par le changement de repère affine :

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{bmatrix} \alpha_u & -\alpha_u \cot\theta \\ 0 & \alpha_v \sin\theta \end{bmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} u_0 \\ v_0 \end{pmatrix} \tag{1.3}$$

θ est l'angle que forment $\mathbf{u}$ et $\mathbf{v}$ (en pratique toujours proche de $\frac{\pi}{2}$), α_u et α_v sont des facteurs multiplicatifs selon chaque axe et leur valeur est la distance focale exprimée en pixels selon chacun des axes, et (u_0,v_0) sont les coordonnées du point principal c dans le repère $(o,\mathbf{u},\mathbf{v})$. Les valeurs de α_u et α_v sont en général données par les dimensions des pixels sur le capteur (données par le constructeur de la caméra) et la distance focale (qui dépend de l'objectif). Donc, en pratique, α_u et α_v varient linéairement avec la focale et le rapport $\frac{\alpha_u}{\alpha_v}$ ne varie pratiquement pas quand on change la focale [Fau93].

L'écriture de la fonction de projection est grandement simplifiée par l'utilisation de la géométrie projective. En effet, soit $\mathbf{M} \cong (x,y,z,1) \in \mathcal{P}^3$ la représentation en coordonnées homogènes[3] du point $M \notin \Pi^f$ et $\mathbf{m} \cong (u,v,1) \in \mathcal{P}^2$ celle du point image associé, alors l'expression de la fonction de projection est simplement une multiplication par la matrice 3×4 $\mathbf{P}$:

$$\mathbf{m} \cong \mathbf{PM} \tag{1.4}$$

avec

$$\mathbf{P} \cong \mathbf{A}\mathbf{P}_0\mathbf{K} \tag{1.5}$$

$$\mathbf{A} \cong \begin{bmatrix} \alpha_u & -\alpha_u \cot\theta & u_0 \\ 0 & \alpha_v \sin\theta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.6}$$

$$\mathbf{P}_0 \cong \begin{bmatrix} \mathbf{I}_3 & \mathbf{0} \end{bmatrix} \tag{1.7}$$

$$\mathbf{K} \cong \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{1.8}$$

La matrice $\mathbf{A}$ est appelée matrice des paramètres intrinsèques, puisqu'elle ne dépend que de ces derniers. De plus, on montre facilement que le centre optique C est le seul point vérifiant $\mathbf{PC} = \mathbf{0}$ [Fau93].

Calibrage fort avec le modèle sténopé

Les méthode de calibrage utilisant uniquement le modèle sténopé sont soit de type directe, le calcul de la matrice de projection se faisant en une

3. Les points de Π^f se projettent à l'infini dans le plan rétinien Π^r, la troisième coordonnée de leur projection est donc nulle et ils ne peuvent pas être représentés par un élément de $\mathcal{P}^2$ du type $(u,v,1)$.

seule étape, soit de type itérative, c'est-à-dire que calcul réclame un nombre indéterminé d'étapes pour converger vers la solution.

Les méthodes de type directe (dont l'État de l'Art est représenté par la méthode décrite dans [FT86]) exploitent la simplicité mathématique du modèle sténopé pour calculer les paramètres intrinsèques et extrinsèques de la caméra à partir d'une collection de points non coplanaires, dont on connaît à la fois les projections dans l'image et les coordonnées 3-D. Ces méthodes ont l'avantage d'être extrêmement rapides et de n'exiger aucune connaissance a priori sur les paramètres de la caméra.

Les méthodes itératives sont le plus souvent une simplification des méthodes de type « ajustement de faisceaux » *(bundle adjustment)* utilisées en photogrammétrie [Sla80], ce type de méthode étant utilisable en théorie pour tout modèle de caméra. Ces méthodes utilisent, comme les méthodes directes, un ensemble de points ou de primitives extraits de l'image, donc les coordonnées 3-D sont connues. L'extraction de ces primitives dans les images étant elle-même une étape délicate [BMB93, Wil94], il est également possible d'utiliser directement les données d'intensité ou de gradient d'intensité de l'image, comme dans [Rob95, Rob93], avec pour conséquence une meilleure précision du calibrage, puisque cette méthode utilise une étape en moins dans le processus de calibrage. Le problème des méthodes itératives est que la solution dépend de l'initialisation et il est donc nécessaire de disposer d'une bonne initialisation. Dans le cas de [Rob95], l'initialisation se fait manuellement par la saisie interactive de quelques coins, mais en général l'initialisation peut se faire automatiquement, par [FT86] par exemple. Dans le cas du modèle sténopé, les méthodes itératives ne réclament pas une initialisation très minutieuse. En effet, le nombre de paramètres du modèle étant faible, la mesure d'erreur de calibrage comporte peu de minima locaux et ces méthodes convergent presque toujours vers le minimum global qui est la solution, même si l'initialisation est grossière.

D'autres méthodes de calibrage fort utilisent des invariants géométriques 3-D de primitives présentes dans les images, plutôt que leurs coordonnées dans un repère donné, comme le parallélisme [CT90, BMZ92, FLR$^+$95] ou l'image d'une sphère [Pen91, DDL94]. Ces méthodes permettent d'obtenir tout ou partie des paramètres intrinsèques de la caméra indépendamment de sa position et donc de découpler lors du calibrage les paramètres intrinsèques des paramètres extrinsèques.

1.1.2 Modèle avec distorsion non-linéaire

Pour obtenir une meilleure précision ou lorsque la qualité des optiques fait qu'elles ne respectent plus le modèle sténopé, il est nécessaire de calibrer et de prendre en compte l'écart par rapport au modèle projectif linéaire, et

donc les distorsions non-linéaires[4]. Les types les plus courants et les plus visibles de distorsion non-linéaires sont les effets dits de « barillet » ou de « coussinet » (figure 1.2) [Sla80] qui peuvent être observés le plus souvent à l'œil nu avec des objectifs très grand angle ou « fish-eye », mais d'autres types de distorsions non-linéaires plus subtiles peuvent apparaître avec ce type d'objectif et il est donc nécessaire de savoir les modéliser pour pouvoir les calibrer. Il est à noter qu'en général les objectifs à grande distance focale possèdent peu ou pas du tout de distorsion radiale (voir [WCH92, DF95b] et annexe A), et c'est d'ailleurs le moyen le plus simple que nous ayons trouvé pour éviter la distorsion non-linéaire lors des expérimentations (les objectifs grand angle ou zoom sans distorsion existent [DLDL96] mais sont encore très onéreux).

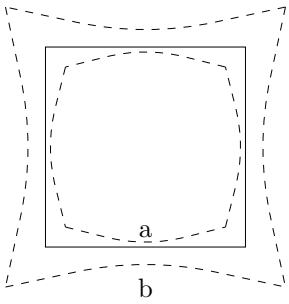


FIG. 1.2 – *Image d'un carré centré face à la caméra par une caméra linéaire (trait plein), image avec une distorsion de type* barillet *(a) et de type* coussinet *(b).*

Modèles de distorsion non-linéaire

L'expression de ces distorsions se fait en ajoutant un terme correctif à chacune des coordonnées x et y correspondant au modèle sténopé et données par l'équation 1.1.1. Les coordonnées x_d et y_d effectivement mesurées sont donc de la forme :

$$x_d = x + \delta_x(x,y)$$
$$y_d = y + \delta_y(x,y)$$

Dans ces équations, $\delta_x(x,y)$ et $\delta_y(x,y)$ sont eux-mêmes la somme de plusieurs termes correctifs correspondant aux différents types de distorsion et au degré d'approximation du modèle de distorsion. Ceux-ci s'expriment, en général, plus naturellement dans un système de coordonnées polaires dont l'origine est le *centre de distorsion.* Le centre de distorsion est le plus souvent

4. Ce qui est communément appelé *distorsion linéaire* correspond en réalité aux effets de la projection perspective, et on confond souvent distorsion non-linéaire et distorsion.

confondu avec le point principal c, et il a été montré [Ste93] que le fait de choisir un centre de distorsion différent du point principal est équivalent à l'ajout d'un terme de distorsion de décentrage. Chaque point m de l'image est donc représenté par sa distance ρ à c et l'angle orienté θ entre l'axe des abscisses et $\overrightarrow{cm}$, de sorte que nous avons :

$$\begin{aligned} x &= \rho \cos\theta \\ y &= \rho \sin\theta \end{aligned}$$

Plus précisément, chaque terme correctif s'exprime comme la combinaison d'une distorsion *radiale* $\delta_r(\rho,\theta)$ et d'une distorsion *tangentielle* $\delta_t(\rho,\theta)$ telles que

$$\begin{pmatrix} \delta_x(x,y) \\ \delta_y(x,y) \end{pmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{pmatrix} \delta_r(\rho,\theta) \\ \delta_t(\rho,\theta) \end{pmatrix} \tag{1.9}$$

Trois types de distorsion sont généralement considérées [Sla80] :

– la *distorsion radiale* due le plus souvent à des défauts dans la courbure radiale des lentilles constituant la caméra, qui peuvent être rendu nécessaires pour avoir une illumination constante sur l'image (c'est-à-dire pour éviter l'effet de *vignettage*) ou pour une mise au point correcte,

$$\begin{aligned} \delta_r(\rho,\theta) &= k_1\rho^3 + k_2\rho^5 + k_3\rho^7 + \cdots \\ \delta_t(\rho,\theta) &= 0 \end{aligned} \tag{1.10}$$

qui s'écrit donc aussi, en négligeant les termes [5] de degré supérieur à 3,

$$\begin{aligned} \delta_x(x,y) &= k_1 x(x^2+y^2) \\ \delta_y(x,y) &= k_1 y(x^2+y^2) \end{aligned} \tag{1.11}$$

– la *distorsion de décentrage*, qui peut être due à un mauvais alignement des centres optiques des lentilles constituant la caméra dans l'espace,

$$\begin{aligned} \delta_r(\rho,\theta) &= 3(j_1\rho^2 + j_2\rho^4 + j_3\rho^6 + \cdots)\sin(\theta-\theta_0) \\ \delta_t(\rho,\theta) &= (j_1\rho^2 + j_2\rho^4 + j_3\rho^6 + \cdots)\cos(\theta-\theta_0) \end{aligned} \tag{1.12}$$

qui s'écrit donc aussi, en négligeant les termes de degré supérieur à 3,

$$\begin{aligned} \delta_x(x,y) &= p_1(3x^2+y^2) + 2p_2xy \\ \delta_y(x,y) &= 2p_1xy + p_2(x^2+3y^2) \end{aligned} \tag{1.13}$$

avec $p_1 = -j_1\sin(\theta_0)$ et $p_2 = j_1\cos(\theta_0)$,

5. On remarque que $k_1 > 0$ correspond à une distorsion de type *coussinet* et qu'à l'inverse $k_1 < 0$ correspond à une distorsion de type *barillet*.

– la *distorsion prismatique* due à une inclinaison des lentilles les unes par rapport aux autres et qui revient à l'adjonction au système optique d'un prisme mince,

$$\begin{aligned}\delta_r(\rho,\theta) &= (i_1\rho^2 + i_2\rho^4 + i_3\rho^6 + \cdots)\sin(\theta - \theta_1)\\ \delta_t(\rho,\theta) &= (i_1\rho^2 + i_2\rho^4 + i_3\rho^6 + \cdots)\cos(\theta - \theta_1)\end{aligned} \tag{1.14}$$

qui s'écrit donc aussi, en négligeant les termes de degré supérieur à 3,

$$\begin{aligned}\delta_x(x,y) &= s_1(x^2 + y^2)\\ \delta_y(x,y) &= s_2(x^2 + y^2)\end{aligned} \tag{1.15}$$

avec $s_1 = -i_1\sin(\theta_1)$ et $s_2 = i_1\cos(\theta_1)$.

Distorsion totale

La distorsion totale associé au modèle de caméra s'obtient en faisant la somme des termes des équations 1.11, 1.13, et 1.15 :

$$\delta_x(x,y) = k_1x(x^2 + y^2) + p_1(3x^2 + y^2) + 2p_2xy + s_1(x^2 + y^2) \tag{1.16}$$

$$\delta_y(x,y) = k_1y(x^2 + y^2) + 2p_1xy + p_2(x^2 + 3y^2) + s_2(x^2 + y^2) \tag{1.17}$$

On remarque que les expressions polaires (équations 1.12 et 1.14) des distorsions de décentrage et de prisme fin ont des formes similaires, cependant elles représentent des effets différents et peuvent avoir chacune un axe de distorsion tangentielle maximum différent.

Calibrage fort avec distorsion non-linéaire

Il n'existe pas de méthode directe efficace pour le calibrage d'une caméra utilisant un modèle avec distorsion non-linéaire (la seule méthode directe proposée qui tienne compte de la distorsion [AAK71] est malheureusement inexacte en présence de bruit dans les données, même faible [WCH92]). Il reste alors deux possibilités pour calibrer en suivant ce modèle : la minimisation non linéaire, ou une méthode en deux étapes : un calibrage par une méthode directe en utilisant le modèle sténopé, puis un calibrage plus fin par minimisation non linéaire qui prenne en compte les paramètres de distorsion.

Le calibrage par minimisation non-linéaire d'un critère étant donné des points de contrôle, c'est-à-dire des points des images dont on connaît les coordonnées 3-D, est une méthode classiquement utilisée en photogrammétrie [Sla80, Bey92]. Un avantage évident de ce type de méthode est que le modèle du capteur peut être quelconque, tant au niveau de la géométrie (caméra projective, capteur push-broom aérien ou satellite, optiques fish-eye, etc.) que de la complexité du modèle (distorsion non-linéaire, perturbations de la trajectoire, etc.). Un autre avantage est que la calibration peut être très

précise, si la minimisation a bien convergé et que le modèle est correct. Évidemment, le nombre de points de contrôle utilisés doit être en rapport avec le nombre de degrés de liberté du modèle, et leur répartition dans l'espace doit être la plus homogène possible[6].

En résumé, l'avantage de ce type de méthode est que le modèle de caméra peut être quelconque et comporter autant de paramètres qu'il est nécessaire. Tous les types de distorsion non-linéaire peuvent être pris en compte par une telle méthode, et les résultats du calibrage seront donc très précis. Cependant, l'inconvénient principal des méthodes par minimisation non-linéaire est que, puisqu'elles sont itératives, elles requièrent une bonne estimation des paramètres de prise de vue (intrinsèques et extrinsèques), et le résultat dépend de cette estimée initiale. Si le nombre de paramètres du modèle est petit et que le critère de convergence est bien choisi, l'estimée initiale peut être grossière, mais plus le nombre de paramètres augmente, plus le nombre de minima locaux du critère de convergence augmente, et moins on a de chances d'obtenir les paramètres de la caméra.

Cette estimation n'étant pas toujours disponible, il faudra éventuellement recourir à une méthode directe pour une première estimation des paramètres. C'est l'approche adoptée par [Tsa87, LT88], qui proposent de calibrer d'abord en utilisant un modèle sténopé par une méthode directe. On obtient ainsi les paramètres extrinsèques et quelques paramètres intrinsèques, puis on réalise une optimisation des paramètres de distorsion en gardant fixes les paramètres déjà calculés lors de la première phase. Le modèle de distorsion qu'ils proposent comporte uniquement la distorsion radiale du premier et du second ordre (paramètres k_1 et k_2), et ne tient pas compte de la distorsion de décentrage (dont l'effet est tangentiel).

Un modèle plus précis est proposé par [WCH92], qui comporte en plus les distorsions de décentrage et prisme fin (on obtient un modèle du troisième ordre, comportant 5 paramètres de distorsion, identique à celui présenté plus haut). De plus, ils ont noté qu'il était nécessaire de faire varier aussi les paramètres autres que ceux de distorsion lors de la seconde optimisation, pour plus de précision. Et lors de la procédure d'optimisation, les paramètres de distorsion sont découplés des autres paramètres pour éviter des interactions néfastes, ce qui rend le résultat de l'optimisation plus stable.

1.2 Calibrage faible

Le calibrage dit faible consiste à calibrer deux caméras (ou plus [Lav96]) sans aucune connaissance a priori sur les paramètres des caméras ou sur la scène observée. En effet, on peut montrer qu'à partir seulement de correspondances de points entre les images des deux caméras, on arrive à calibrer complètement ce qu'on appelle la *géométrie épipolaire* du système stéréo-

6. Ceci est également vrai pour les autres méthodes de calibrage, fort ou faible.

scopique. Lorsque les paramètres intrinsèques des caméras sont connus, la géométrie épipolaire est décrite par la *matrice essentielle* [LH81], qui agit sur les points des images en coordonnées normalisées, et le calcul de cette matrice correspond à la détermination du mouvement et de la structure à partir de correspondances, problème qui a été largement traité en vision par ordinateur (voir [HN94] pour une revue).

Le cas qui nous intéresse ici est celui où l'on ne dispose pas de ces paramètres intrinsèques : la géométrie épipolaire est alors décrite par la *matrice fondamentale* [LF96b], qui agit sur les coordonnées des images en coordonnées pixels, et qui contient des informations permettant de simplifier la mise en correspondance de points entre les images et de retrouver la structure 3-D projective de la scène [Fau92], c'est-à-dire la structure à une transformation projective[7] de l'espace près. Malgré cette incertitude sur la géométrie de la scène, la structure projective conserve certaines propriétés de la structure euclidienne (coplanarité, birapports, et autres invariants projectifs) qui permettent d'effectuer des tâches robotiques simples [ZF94a, BZM94], et elle peut également être une première étape de l'auto-calibrage (§ 1.4).

1.2.1 La géométrie épipolaire

La géométrie épipolaire est intrinsèque à tout système de deux caméras quel que soit le modèle utilisé pour ces caméras. Considérons par exemple le système de deux caméras projectives de la figure 1.3, et soit m_1 un point de la première image. Le point de l'espace M, dont la projection dans la première image est m_1, se situe nécessairement sur la droite $(C_1 m_1)$ passant par le centre optique C_1, et sa projection m_2 dans la deuxième image est donc obligatoirement sur la projection $\mathbf{l}_{\mathbf{m}_2}$ de $(C_1 m_1)$ dans la deuxième image, appelée *droite épipolaire* . Plus généralement, et quel que soit le modèle de caméra, à un point d'une image correspond une droite de l'espace, qui se projète dans l'autre image en une *courbe épipolaire*, et la géométrie de cette courbe dépend du modèle de caméra utilisé (par exemple une hyperbole dans le cas d'une caméra push-broom linéaire [HG94]).

Revenons au cas simple de caméras projetives linéaires, la droite épipolaire $\mathbf{l}_{\mathbf{m}_2}$ est également l'intersection du plan Π qui passe par m_1, C_1, et C_2, appelé *plan épipolaire*, avec le plan rétinien de la deuxième caméra. On remarque ainsi que toutes les droites épipolaires passent dans la deuxième image par le point e_2, appelé *épipôle*, intersection de la droite $(C_1 C_2)$ avec le plan rétinien, puisque la droite $(C_1 C_2)$ est nécessairement incluse dans tous les plans épipolaires. De plus, la géométrie épipolaire est symétrique : étant donné un point m_2 dans la seconde image, son correspondant m_1 se trouve nécessairement sur la droite épipolaire $\mathbf{l}_{\mathbf{m}_1}$, et toutes les droites épipolaires de la première images passent par l'épipôle e_1, intersection de $(C_1 C_2)$ avec le

7. Une transformation projective est représentée par une matrice 4×4 agissant sur les points 3-D en coordonnées homogènes.

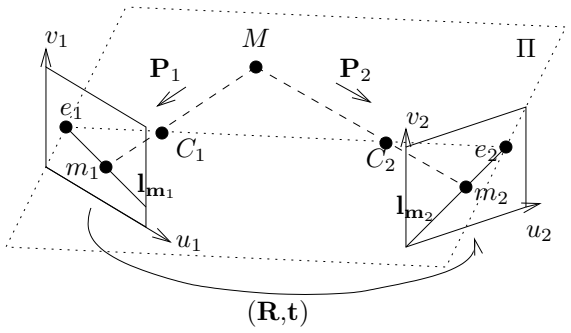


FIG. 1.3 – *La géométrie épipolaire de deux caméras.*

plan image de la première caméra. Les points (m_1,m_2,C_1,C_2) sont donc dans un même plan, c'est la contrainte de coplanarité, utilisée notamment pour calculer le mouvement et la structure 3-D lorsque les paramètres intrinsèques sont connus [LH81].

1.2.2 La matrice fondamentale

La matrice fondamentale est l'expression algébrique de la contrainte de coplanarité des points (m_1,m_2,C_1,C_2) exprimée à partir des points images $\mathbf{m}_1$ et $\mathbf{m}_2$, cette contrainte étant appelée *contrainte épipolaire*.

Pour calculer l'expression algébrique, on commence par écrire les matrices de projection $\mathbf{P}_1$ et $\mathbf{P}_2$, en supposant que le repère du système de coordonnée de référence coincide avec celui de la caméra 2 (on comprend facilement que la matrice fondamentale est indépendante du choix du système de coordonnées euclidien) :

$$\mathbf{P}_1 = \mathbf{A}_1 \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \quad \text{et} \quad \mathbf{P}_2 = \mathbf{A}_2 \begin{bmatrix} \mathbf{I}_3 & \mathbf{0} \end{bmatrix} \tag{1.18}$$

et les coordonnées des projections de M dans les deux images sont :

$$s_1\mathbf{m}_1 = \mathbf{P}_1\mathbf{M} \quad \text{et} \quad s_2\mathbf{m}_2 = \mathbf{P}_2\mathbf{M} \tag{1.19}$$

En éliminant $\mathbf{M}$, s_1 et s_2 dans les équations ci-dessus, on obtient :

$$\mathbf{m}_1^T\mathbf{F}_{12}\mathbf{m}_2 = 0 \quad \text{avec} \quad \mathbf{F}_{12} = \mathbf{A}_1^{-T} \left[\mathbf{t}\right]_\times \mathbf{R}\mathbf{A}_2^{-1} \tag{1.20}$$

où $\left[\mathbf{t}\right]_\times$ est la matrice antisymétrique définie par $\left[\mathbf{t}\right]_\times \mathbf{x} = \mathbf{t} \wedge \mathbf{x}$ quel que soit le vecteur $\mathbf{x}$.

$\mathbf{F}_{12}$ est appelée *matrice fondamentale*. Cette matrice possède quelques propriétés intéressantes : puisque $\det[\mathbf{t}]_\times\mathbf{x} = 0$, alors $\det \mathbf{F}_{12} = 0$ et $\mathbf{F}_{12}$ est de rang 2. De plus, $\mathbf{F}_{12}$ est définie à un facteur d'échelle près puisque l'équation 1.2.2 est toujours valable quand on multiplie $\mathbf{F}_{12}$ par un facteur quelconque. On peut également montrer que si on échange le rôle de chacune des caméras, c'est-à-dire que le système de coordonnées de référence est celui de la caméra 1, l'équation 1.2.2 devient $\mathbf{m}_2^T\mathbf{F}_{21}\mathbf{m}_1 = 0$ et que $\mathbf{F}_{21} \cong \mathbf{F}_{12}^T$.

1.2.3 Calibrage avec un modèle de caméra projectif linéaire

La situation canonique où les deux caméras utilisées suivent un modèle projectif linéaire a été largement traitée dans la revue des algorithmes de calibrage faible par Zhang [Zha96a], et nous ne nous étendrons donc pas sur le sujet. Mentionnons simplement que pour nos expérimentations nous avons utilisé une méthode robuste, de type « moindre médiane des carrés », décrite dans [ZDFL95], et qui donne des résultats satisfaisants.

Il est également possible, en plus du calcul de la matrice fondamentale à partir de correspondances, d'évaluer l'incertitude sur cette matrice [CZZF95, Csu96], ce qui permettrait par exemple d'avoir pour chaque point de la première image non pas une droite, mais une bande épipolaire dans laquelle un algorithme de stéréoscopie devrait chercher son correspondant. Notre algorithme de stéréoscopie (chapitre 3) utilisant la contrainte épipolaire comme une contrainte stricte, il ne peut malheureusement pas utiliser ces bandes épipolaires.

1.2.4 Calibrage utilisant un modèle avec distorsion non linéaire

De la même manière que pour le calibrage fort le modèle projectif linéaire peut se révéler insuffisant, à cause d'optiques générant une forte distorsion, on peut avoir besoin d'utiliser un modèle de caméra comportant de la distorsion non linéaire pour le calibrage faible. Malheureusement, la simplicité de la contrainte épipolaire, qui se réduisait à la matrice fondamentale dans le cas linéaire, se dégrade largement. Pour faire face aux problèmes liés à la géométrie complexe des courbes épipolaires, il existe deux solutions : soit on découple les problèmes, et on cherche à déterminer la fonction de distorsion de chaque caméra, ce qui permet de se ramener au cas projectif linéaire précédemment traité, soit on traite les problèmes simultanément, et on calcule en même temps les fonctions de distorsions des deux caméras et la matrice fondamentale.

Calibrage monoculaire de la distorsion avant calibrage faible

Une méthode de calibrage monoculaire de la distorsion avant même la détermination de la géométrie de la scène nécessite d'avoir des informations a priori sur la scène, permettant au minimum la détermination de la fonction de distorsion.

La méthode de Brand *et al.* [BMB93] consiste à présenter devant la caméra l'image d'une grille plane régulière quelconque (c'est-à-dire que la seule contrainte est qu'elle soit régulière). Les points des intersections de la grille sont ensuite détectés à une précision sous-pixelique, puis on cherche pour chacun de ces points le déplacement minimum de ces points permettant de « redresser » la grille, c'est-à-dire qu'après ces déplacements, l'ensemble de

ces points forment l'image d'une grille par une caméra projective linéaire. On en déduit alors un champ de distorsion par interpolation de ces déplacements entre les points consécutifs de la grille. Cette méthode utilise donc un modèle de distorsion complètement générique, et définit simplement la fonction de distorsion comme une fonction permettant de passer d'une image par une caméra projective linéaire idéale à l'image réelle. Cette méthode possède tout de même des inconvénients de taille. Tout d'abord, l'utilisation d'une grille régulière signifie qu'on aurait aussi bien pu utiliser une méthode de calibrage fort. Ensuite, le champ de distorsion n'est défini qu'entre les points détectés sur la grille, et n'est donc pas défini aux bords de l'image. Une méthode similaire, dite du « fil à plomb », permet de calibrer la distorsion à partir d'images d'une ligne droite [Bro71].

L'idée à retenir de ces méthodes est la suivante : la caméra projective *linéaire* est la seule qui projette toute droite de l'espace en une droite du plan. Partant de là et de la constatation que les constructions humaines comportent essentiellement des droites, nous avons pu développer une méthode de calibrage de la distorsion (voir annexe A et [DF95a]). Le principe de la méthode est d'extraire les contours d'une série d'images prises avec la même caméra à une précision sous-pixélique (en utilisant le détecteur de contours de l'annexe C), puis d'effectuer une première approximation polygonale avec une tolérance de quelques pixels sur ces contours. On a donc un ensemble de « segments potentiels » (on ne sait pas s'il s'agit réellement de segments 3-D) et les morceaux de chaînes de contours qui y correspondent. On cherche alors les paramètres de la fonction de distorsion (nous avons utilisé une distorsion radiale d'ordre 1) permettant de « détordre » le mieux possible ces chaînes de contours correspondant aux segments, et de les rendre rectilignes. Utilisant ces paramètres, on détord alors *l'ensemble* des chaînes de contours et on refait une approximation polygônale, ce qui a pour effet d'eliminer certaines courbes qui avaient été détectées comme des segments et de détecter des segments plus longs, puis on recommence l'optimisation des paramètres de distorsion. Les détails de la méthode et les résultats sont présentés en annexe A.

N'oublions pas de remarquer que ces méthodes de calibrage monoculaire de la distorsion ont été présentées dans le cadre du calibrage faible, mais fonctionnent tout aussi bien avec les méthodes de type auto-calibrage, et la seconde méthode a d'ailleurs été utilisée avec succès dans ce cadre [Lav96], ou de calibrage fort, ce qui permet de découpler facilement les paramètres de distorsion des autres paramètres intrinsèques lors du calibrage.

Calibrage simultané de la distorsion

Il est également possible de calibrer *simultanément* les paramètres de la fonction de distorsion et la géométrie épipolaire. C'est l'objet de la tentative de Zhang [Zha96b], mais les résultats montrent hélas que ce calibrage si-

multané est difficile et instable, puisque les paramètres sont très sensibles au bruit sur les ponts extraits et mis en correspondance. Cependant, il semble que ces résultats n'aient pas été estimés à leur juste valeur : en effet, pour mesurer la précision des paramètres de distorsion trouvés par l'algorithme, il les compare à ceux obtenus par calibrage fort (ou les paramètres utilisés pour genérer l'image dans le cas de données de synthèse), alors qu'il faut simplement mesurer si la caméra obtenue après application de la fonction inverse de distorsion est projective linéaire, c'est-à-dire identique *à une homographie de l'image près* à l'originale. De plus, on peut augmenter le nombre de données en utilisant plusieurs paires d'images prises avec le même système stéréoscopique. Il serait donc nécessaire de réévaluer cette méthode en utilisant des mesures plus justes avant de la juger inutilisable.

1.3 Calibrage hybride

Le but final de la stéréoscopie est d'obtenir une représentation *euclidienne* de la scène observée, et le meilleur moyen d'obtenir la structure euclidienne est d'utiliser une méthode de calibrage fort. Cependant, les méthodes de calibrage fort sont essentiellement monoculaires, et les expérimentations que nous avons menées ont montré que la géométrie épipolaire calculée à partir des paramètres des deux caméras (par la formule 1.2.2) était beaucoup moins précise que celle obtenue directement par une méthode de calibrage faible (§ 1.2). En conséquence, les méthodes de stéréoscopie par corrélation, et notamment la méthode améliorée décrite § 3.2.2 page 59, fonctionnaient beaucoup moins bien avec le calibrage fort qu'avec le calibrage faible. Nous avons donc eu l'idée d'utiliser une méthode de calibrage hybride, qui nous donnerait la géométrie euclidienne de la scène tout en possédant une géométrie épipolaire très précise.

1.3.1 Par recalage projectif des reconstructions

Notre première idée (l'idée et les résultats ont été repris à son compte par [ZFD95]) est la suivante : On calcule d'abors, grâce à la matrice fondamentale, une reconstruction projective (voir chapitre 4) de points de la scène dont on connaît les coordonnées 3-D. Ensuite, grâce à une méthode de moindres carrés linéaires, on calcule la matrice 4×4 dite de « distorsion projective », qui est une collinéation de l'espace $\mathcal{P}^3$, permettant de passer de la reconstruction projective à la reconstruction euclidienne (cette dernière est connue).

En composant les fonctions de projection correspondant à la reconstruction projective par la distorsion projective, on obtient alors les fonctions de projection (soit les matrices de projection dans le cas du modèle projectif linéaire) correspondant à la reconstruction euclidienne. On a donc calculé

une paire de matrices de projection *euclidennes* en se basant sur une matrice fondamentale donnée a priori.

1.3.2 Une méthode de calibrage pour chaque étape

Une autre solution au calibrage hybride est simplement de calculer $\mathbf{F}$ par une méthode de calibrage faible (§ 1.2), de calculer $\mathbf{P}_1$ et $\mathbf{P}_2$ indépendamment, par une méthode de calibrage fort (§ 1.1) puis d'utiliser soit l'un soit l'autre de ces résultats suivant que le calcul réclame une géométrie épipolaire précise ou un moyen de reconstruire en euclidien.

Plus précisément, dans notre cas, nous avons utilisé $\mathbf{F}$ pour l'étape de calcul des matrices rectification, qui dépend fortement de la qualité de la géométrie épipolaire, et nous avons utilisé les matrices $\mathbf{P}_1$ et $\mathbf{P}_2$ pour le calcul de la matrice de reconstruction (§ 4.2.1 page 82). Les matrices de rectification étant compatibles (§ 2.2.1 avec $\mathbf{F}$ et non avec $\mathbf{P}_1$ et $\mathbf{P}_2$, le calcul de la matrice de reconstruction ne sera pas exact et on pourra calculer, en plus de la reconstruction de chaque point, l'incertitude associé à ce point.

1.4 Auto-calibrage

La dernière sorte de techniques de calibrage sont celles qui ne nécessitent aucun point de coordonnés 3-D connues. Elles sont également nommées méthodes *d'auto-calibrage*, et permettent de retrouver la structure euclidienne de l'espace à un facteur d'echelle global près, puisqu'il n'y a a priori aucun étalon de distance pour avoir une notion d'échelle. Les méthodes monoculaires exploitent le fait que les paramètres intrinsèques de la caméra restent constants entre les différentes prises de vue. Celles-ci donnant des résultats plutôt instables, on a commencé par imposer des contraintes sur le type de mouvement imposé à la caméra, mais il n'est pas toujours facile d'obtenir un un mouvement précis (de type « rotation pure », par exemple), et dans de noumbreux cas on n'est même pas maître du mouvement. L'autre solution a été d'imposer des contraintes sur la structure plutôt que sur le mouvement, par exemple en utilisant le fait que les deux caméras d'un système stéréoscopique sont rigidement liées. Ces contraintes structurelles permettent d'obtenir des résultats beaucoup plus stables d'auto-calibrage (annexe B).

1.4.1 Auto-calibrage monoculaire

Les premiers résultats d'auto-calibrage présentaient une partie théorique très attrayante [MF92, FLM92], mais jusqu'à récemment les résultats, qui étaient obtenus par la resolution des équation polynomiales dites de Kruppa, se sont avérés trop instables pour être utilisables dans des situations réelles [Luo92].

On a donc préféré se limiter à un mouvement d'un type donné entre des vues donnée pour calculer tout ou partie des paramètres de la caméra. Ainsi, en se limitant aux rotations pures de la caméra on peut calibrer les paramètres intrinsèques de la caméra, que le modèle soit projectif [Har94a] ou comporte de la distorsion non linéaire [DB93b, Ste93, Ste95]. En se limitant aux mouvements de type translation, on peut retrouver l'image du plan à l'infini dans les deux images et donc retrouver la structure affine de l'espace [ZF94a, Zel96]. Les résultats de toutes ces méthodes sont satisfaisants, mais il est parfois difficile d'obtenir précisément un mouvement donné de la caméra, comme une rotation pure, et il arrive même qu'on ne puisse pas du tout choisir le mouvement de la caméra.

Dernièrement, d'excellents résultats ont été obtenus par Zeller, sans aucune contrainte sur les paramètres des caméras, par l'adaptation à l'auto-calibrage de méthodes de type ajustement de faisceaux *(bundle adjustment)*, utilisées auparavant en photogrammétrie. Ces résultats, et ceux de Laveau qui en sont une extension à un nombre quelconque de caméras, permettent enfin d'espérer pouvoir utiliser l'auto-calibrage dans de nombreuses situations réelles.

1.4.2 Auto-calibrage stéréoscopique

Si on utilise un système stéréoscopique rigide pour l'acquisition des images, on peut utiliser cette contrainte structurelle supplémentaire pour auto-calibrer les deux caméras. Dans [ZBR95, DF95b, DF96] (voir aussi annexe B), l'approche consiste a effectuer des reconstructions projectives de la scène en utilisant la même paire de matrices de projection pour plusieurs paires d'images, puis à remarquer que les paramètres intrinsèques et extrinsèques du système stéréoscopique peuvent être déduits de la distorsion projective entre ces deux reconstructions, représentée par une matrice 4×4.

Plusieurs problèmes apparaissent alors. D'abord, on s'apperçoit que deux paires stéréoscopiques ne suffisent pas à résoudre le problème. Les solutions proposées ont été soit de fixer un des paramètres des caméras (l'angle que font les deux axes, en l'occurence) pour obtenir autant de paramètre que de contraintes, soit d'utiliser plus de deux paires stéréoscopique (notre solution). L'utilisation d'un nombre quelconque (au moins deux) de paires stéréoscopiques permet en plus, comme montré en annexe B, de rendre le résultat plus stable.

Ensuite, la matrice de distorsion projective possède certaines propriétés (sur ses valeurs propres, voir [ZBR95, DF95b]) qu'il est difficile d'exprimer comme des contraintes lors du calcul de cette matrice. La solution proposée par [ZBR95] a été de ne pas utilser ces contraintes lors de son calcul, puis à verifier que les contraintes sont quasiment vérifiées et à reprojeter sur la variété des matrices de distorsion autorisées. Notre solution, plus satisfaisante sur le plan algébrique a été de trouver une paramétrisation algébrique de la

variété des matrices de distorsion projectives autorisées.

Dans [ZLF96], l'approche est plus calculatoire qu'analytique. La méthode consiste d'abord à prendre deux paires stéréoscopiques avec le même système stéréo, et à calculer à partir de correspondances de points les matrices fondamentales liant les deux images droites entre elles, les deux images gauches entre elles, et les images droites aux images gauches (ce qui fait trois matrices fondamentales, $\mathbf{F}_{rr}$, $\mathbf{F}_{ll}$ et $\mathbf{F}_{rl}$). Ensuite, une estimée initiale des paramètres extrinsèques et intrinsèques (sauf les coordonnées des points principaux de chacune des caméras, qui sont supposées connues) est calculée à partir de ces matrices fondamentales, en résolvant les équations dites de Kruppa [FLM92]. Enfin ces paramètres sont optimisés par une méthode de moindres carrés non-linéaires appliquée aux distances de chaque point image aux droites épipolaires issues des autres caméras pour enfin trouver les matrices de projection de chacune des caméras et le mouvement entre les deux prises de vue. Cette méthode utilise beaucoup de chemins détournés pour résoudre un problème simple, et passe notamment par la résolution des équations de Kruppa qui est réputée instable [Luo92].

Lorsque le mouvement du système entre les deux prises de vue stéréoscopiques est un vissage, Beardsley *et al.* ont montré qu'on pouvait uniquement retrouver la structure affine de l'espace 3-D, et ont présenté une application de ce résultat en robotique mobile pour la navigation [BZM94, BZ95].

1.5 Méthodes utilisées au cours des expériences

Lors de nos expérimentations sur des lentilles à distorsion faible (chapitre 4 et annexe B) nous avons utilisé les deux méthodes de calibrage hybride décrites § 1.3.1, à partir du calibrage fort de [Rob95] couplé au calibrage faible de [ZDFL95]. Le calibrage fort utilise la mire de la figure A.3 page 117. Pour corriger la distorsion lorsque c'était nécessaire nous avons utilise [LT88] lorsque des données terrain (l'image d'une mire de calibrage p.ex.) étaient disponibles, dans tous les autres cas nous avons corrigé la distorsion à l'aide de [DF95a] sinon.

Chapitre 2

Rectification des paires d'images

Lewis Trondheim, Le Crabar de Mammouth

La rectification est une étape importante de la stéréoscopie par corrélation puisqu'elle permet en effet de se ramener à une géométrie épipolaire simple, dans laquelle les droites épipolaires sont parallèles aux lignes des images. Après rectification des deux images, les points se correspondant ont nécessairement la même ordonnée dans les deux images, et la recherche du point de la deuxième image correspondant à un point donné de la première image se limite donc à une recherche monodimensionelle le long d'une droite horizontale de la seconde image située à la même ordonnée, plutôt qu'une recherche bidimensionnelle dans une région de la seconde image. La rectification est une transformation des paires d'images stéréoscopiques qui dépend uniquement de la *géométrie* du système stéréoscopique, c'est-à-dire du modèle et des paramètres des caméras ainsi que de leur position relative, et elle permet sans information *a priori* sur la scène observée de simplifier le processus de mise en correspondance.

Nous nous limitons dans ce chapitre au cas d'un système stéréoscopique de caméras perspectives. Dans le cas ou les caméras comportent des distor-

sions non linéaires, on se ramène au modèle projectif linéaire (ou sténopé) par l'application de l'inverse de la fonction de distorsion de l'image (voir § 1.2.4). Dans le cas d'autres types de capteurs, notamment des capteurs « pushbroom » [HG94], les mathématiques et les transformations des images mises en jeu sont complètement différentes.

Nous allons montrer que dans le cas d'un modèle de caméra perspectif linéaire, une manière simple de rectifier une paire d'images consiste à appliquer une certaine transformation homographique à chacune des deux images. Nous verrons que l'ensemble des transformations homographiques permettant de rectifier les deux images d'une paire stéréoscopique est une famille à neuf paramètres, et que le problème de la rectification se résume donc à choisir la paire d'homographies qui convient le mieux pour l'application considérée (en ce qui nous concerne, la stéréoscopie par corrélation), en fonction de la géométrie des caméras, et nous tenterons d'en déduire une stratégie de choix de ces homographies de rectification.

2.1 Le point de vue tridimensionnel

La technique la plus simple de rectification dans le cas d'une paire stéréoscopique prise avec des caméras projectives est de reprojeter les deux images des plans rétiniens Π^r et $\Pi^{r\prime}$ sur un même plan Π, dit « plan de rectification », parallèle à la droite (CC') joignant les deux centres optiques (figure 2.1) [Fau93, AH88]. Les centres optiques de chacune des images restant les mêmes, cette opération revient à appliquer une transformation bidimensionnelle à chacune des images et ne requiert aucune connaissance sur la géométrie de la scène observée. Puisque l'opération de reprojection du plan image est une projection linéaire, la transformation à effectuer sur chacune des images est une homographie.

Par cette transformation, on voit que les épipôles de chacune des images sont reprojetés à l'infini dans les images rectifiées, dans la direction de la droite (CC'). On choisit cette direction comme droite des abscisses, et les droites épipolaires sont donc horizontales.

Soit une paire d'images prises avec un système stéréoscopique de caméras sténopé, et soit la matrice fondamentale $\mathbf{F}$ associée. Une condition nécessaire pour que deux points m et m' se correspondent est qu'il satisfassent la contrainte épipolaire, $\mathbf{m'}^T\mathbf{F}\mathbf{m} = 0$. Lorsque la paire d'image est rectifiée, les droites épipolaires sont horizontales et deux points qui se correspondent ont nécessairement la même ordonnée, pour préparer a la stéréoscopie par corrélation. La matrice fondamentale de la paire d'images rectifiées, aussi appelée matrice fondamentale *rectifiée*, a alors la forme:

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \tag{2.1}$$

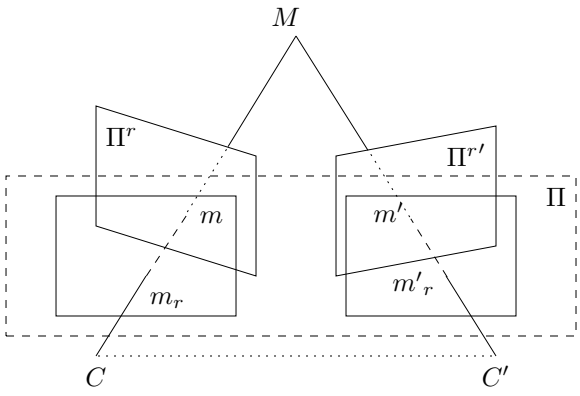


FIG. 2.1 – *La rectification consiste à reprojeter les images des plans rétinaux* Π^r *et* $\Pi^{r'}$ *sur un même plan* Π*, dit « plan de rectification », parallèle à la droite* (CC') *joignant les deux centres optiques. La reprojection revient à appliquer une homographie à chacune des images.*

Les degrés de liberté de la rectification vue comme une reprojection sont assez simples à identifier, gardons cependant en mémoire qu'après rectification un point d'une des deux images et la droite épipolaire qui lui correspond doivent avoir la même ordonnée dans les deux images.

Le premier degré de liberté, et le seul en rapport avec la configuration 3-D de la reprojection, est l'angle Φ que forme le plan de rectification Π avec, par exemple, le plan horizontal d'équation $z = 0$. Les degrés de liberté restants sont uniquement en rapport avec les paramètres intrinsèques des deux caméras virtuelles sur lesquelles se fait la reprojection : (u_0,v_0), qui sont les coordonnées du point principal de la première caméra; l'abscisse u'_0 du point principal dans la seconde caméra, son ordonnée étant nécessairement v_0; les facteurs d'échelle selon les deux axes dans la première caméra, α_u et α_v; le facteur d'échelle selon l'axe des abscisses dans la seconde caméra, α'_u; enfin, les angles que forment les axes dans les deux caméras, θ et θ'.

La difficulté liée à cette approche est qu'on comprend bien ce qui se passe en 3-D, mais le lien des paramètres avec les déformations que la rectification fait subir aux images originales est beaucoup moins clair. De plus il est nécessaire pour utiliser cette approche de connaître les matrices de projection et donc de calibrer fortement ou d'auto-calibrer les caméras, ce qui peut être évité, comme nous allons le voir.

2.2 À partir de la géométrie épipolaire

Nous allons montrer que pour rectifier une paire d'images stéréoscopique, il n'est pas nécessaire de connaître la géométrie tridimensionnelle du système

de caméras. En effet, la connaissance de la géométrie épipolaire, c'est-à-dire de la matrice fondamentale associée au couple de caméras, suffit pour calculer les matrices de rectification, et nous allons voir qu'en considérant la rectification d'un point de vue algébrique on peut facilement identifier l'ensemble des paires de matrices de rectification autorisées, qui forment une variété de dimension 9. De plus, nous allons expliquer simplement de quelle manière chacun de ces neuf paramètres commande la déformation des images par la paire de transformations homographiques. La manière dont nous traitons ici le problème de la rectification à partir de la seule géométrie épipolaire possède l'avantage de traiter les deux images de manière symétrique, contrairement aux approches précédemment proposées [ZF94a, Har97].

2.2.1 Le point de vue algébrique

Considérons le problème de la rectification d'un point de vue purement algébrique. Le problème s'énonce alors ainsi :

Étant donnée la matrice fondamentale $\mathbf{F}$ d'un système stéréoscopique, trouver deux matrices $\mathbf{R}$ et $\mathbf{R}'$, appelées *matrices de rectification*, telles que

$$\mathbf{F} \cong \mathbf{R}'^T \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \mathbf{R} \tag{2.2}$$

Soient deux points m et m' appartenant respectivement à la première et à la deuxième image, alors les images $\mathbf{r}$ et $\mathbf{r}'$ de ces points par les matrices $\mathbf{R}$ et $\mathbf{R}'$ satisfont la contrainte épipolaire simplifiée:

$$\mathbf{r}'^T \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \mathbf{r} = \mathbf{m}'^T \mathbf{R}'^T \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \mathbf{R}\mathbf{m} = \mathbf{m}'^T \mathbf{F}\mathbf{m} = 0 \tag{2.3}$$

soit

$$y_r z_{r'} - y_{r'} z_r = 0. \tag{2.4}$$

C'est-à-dire qu'une condition nécessaire pour que deux points des images rectifiées se correspondent est qu'ils aient la même ordonnée dans ces images rectifiées. On peut donc définir la notion de compatibilité entre une paire de matrices de rectification et une matrice fondamentale.

Définition 1 *Une paire* ($\mathbf{R}$,$\mathbf{R}'$) *de matrices de rectification est dite* compatible *avec la matrice fondamentale* $\mathbf{F}$ *si et seulement si:*

$$\mathbf{F} \cong \mathbf{R}'^T \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \mathbf{R} \tag{2.5}$$

Nous allons d'abord calculer par une méthode purement algébrique une paire *canonique* de matrices de rectification compatible avec une matrice fondamentale $\mathbf{F}$, puis nous allons montrer que l'ensemble de toutes les matrices de rectification s'en déduit facilement et est une variété de dimension 9, c'est-à-dire qu'il est nécessaire de fixer 9 paramètres pour obtenir une paire de matrices de rectification. De plus, la signification précise de chacun de ces 9 paramètres sera clairement expliquée.

Calcul d'un jeu de Matrices de rectification

Soit $\mathbf{F}$ la matrice fondamentale associée à un système stéréoscopique, nous avons vu § 1.2 que cette matrice est de rang 2, donc une décomposition en valeurs singulières [ABB+94] permet d'écrire la matrice fondamentale sous la forme :

$$\mathbf{F} = \begin{bmatrix}\mathbf{e}' & \mathbf{u_1} & \mathbf{u_2}\end{bmatrix} \begin{bmatrix}0 & 0 & 0\\ 0 & \sigma_1 & 0\\ 0 & 0 & \sigma_2\end{bmatrix} \begin{bmatrix}\mathbf{e}^T\\ \mathbf{v_1}^T\\ \mathbf{v_2}^T\end{bmatrix} = \mathbf{Q}'\Sigma\mathbf{Q}^T \tag{2.6}$$

où σ_1 et σ_2 sont deux nombres réels positifs appelés *valeurs singulières* de $\mathbf{F}$ (puisque $\mathbf{F}$ est de rang 2, la troisième valeur singulière est nécessairement nulle), Σ est la matrice diagonale de diagonale $(0,\sigma_1,\sigma_2)$, et les matrices $\mathbf{Q}$ et $\mathbf{Q}'$ sont orthogonales. On remarque que les vecteurs $\mathbf{e}$ et $\mathbf{e}'$ des matrices $\mathbf{Q}$ et $\mathbf{Q}'$ résultats de la décomposition en valeurs singulières sont les épipôles associés à chacune des images, puisqu'en effet ces vecteurs vérifient $\mathbf{e}'^T\mathbf{F} = 0$ et $\mathbf{Fe} = 0$. Dans la suite, $\mathbf{F}$ étant définie à un facteur d'échelle près, on peut prendre pour valeurs propres $\sigma_1 = 1$ et $\sigma_2 = \sigma > 0$.

Une simple manipulation sur les trois matrices permet de se ramener à l'équation 2.2.1, où σ a été réparti de manière à symétriser l'expression :

$$\mathbf{F} = \begin{bmatrix}\mathbf{e}' & \mathbf{u_1} & \sqrt{\sigma}\mathbf{u_2}\end{bmatrix} \begin{bmatrix}0 & 0 & 0\\ 0 & 0 & -1\\ 0 & 1 & 0\end{bmatrix} \begin{bmatrix}\mathbf{e}^T\\ \sqrt{\sigma}\mathbf{v_2}^T\\ -\mathbf{v_1}^T\end{bmatrix} \tag{2.7}$$

Nous avons donc réussi à obtenir une paire de matrices de rectification compatible avec $\mathbf{F}$ à partir d'une simple décomposition en valeurs singulières de $\mathbf{F}$:

$$\mathbf{R}_0 \cong \begin{bmatrix}\mathbf{e}^T\\ \sqrt{\sigma}\mathbf{v_2}^T\\ -\mathbf{v_1}^T\end{bmatrix} \tag{2.8}$$

$$\mathbf{R}'_0 \cong \begin{bmatrix}\mathbf{e}'^T\\ \mathbf{u_1}^T\\ \sqrt{\sigma}\mathbf{u_2}^T\end{bmatrix} \tag{2.9}$$

Variété des rectifications autorisées

Nous allons démontrer ici qu'on peut déduire d'une paire de matrices de rectification compatibles avec $\mathbf{F}$ *l'ensemble* des matrices de rectification compatibles avec $\mathbf{F}$, c'est-à-dire que :

Théorème 1 *L'ensemble des des paires de matrices de rectifications* $(\mathbf{R},\mathbf{R}') \in L(3)^2$ *compatibles avec une matrice fondamentale* $\mathbf{F}$ *est une variété de dimension 9.*

Soient $(\mathbf{R}_0,\mathbf{R}'_0)$ et $(\mathbf{R}_1,\mathbf{R}'_1)$ deux paires de matrices de rectification compatibles avec $\mathbf{F}$, alors

$$\mathbf{F} \cong {\mathbf{R}'_0}^T \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \mathbf{R}_0 \cong {\mathbf{R}'_1}^T \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \mathbf{R}_1 \tag{2.10}$$

Donc, les matrices de rectification étant de rang plein,

$$\left(\mathbf{R}'_1 {\mathbf{R}'_0}^{-1}\right)^T \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \mathbf{R}_1 \mathbf{R}_0^{-1} \cong \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}. \tag{2.11}$$

Il suffit alors de déterminer l'ensemble des couples de matrices 3×3 de rang plein $(\mathbf{M},\mathbf{M}')$ vérifiant

$$\mathbf{F}' \cong {\mathbf{M}'}^T \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \mathbf{M} \cong \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}, \tag{2.12}$$

et l'ensemble des paires de matrices de rectification $(\mathbf{R},\mathbf{R}')$ compatibles avec $\mathbf{F}$ se déduit alors d'une paire de matrices de rectification $(\mathbf{R}_0,\mathbf{R}'_0)$ compatible avec $\mathbf{F}$ par

$$\mathbf{R} \cong \mathbf{M}\mathbf{R}_0 \qquad \text{et} \qquad \mathbf{R}' \cong \mathbf{M}'\mathbf{R}'_0 \tag{2.13}$$

Pour résoudre ce système, posons :

$$\mathbf{M} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \qquad \text{et} \qquad \mathbf{M}' = \begin{bmatrix} a' & b' & c' \\ d' & e' & f' \\ g' & h' & i' \end{bmatrix}. \tag{2.14}$$

L'équation 2.2.1 devient alors :

$$\mathbf{F}' = \begin{bmatrix} g'd - d'g & g'e - d'h & g'f - d'i \\ h'd - e'g & h'e - e'h & h'f - e'i \\ i'd - f'g & i'e - f'h & i'f - f'i \end{bmatrix} \cong \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}. \tag{2.15}$$

Or les matrices M et M' sont de rang plein; leurs vecteurs lignes respectifs sont donc linéairement indépendants, et l'examen de la première colonne et de la première ligne de $\mathbf{F}'$ permet de déduire que

$$d = g = 0 \qquad \text{et} \qquad d' = g' = 0 \tag{2.16}$$

De l'équation 2.2.1 on obtient que $\mathbf{F'}_{22} = 0$ et $\mathbf{F'}_{33} = 0$, donc

$$\exists(\alpha,\beta) \in \Re^2, \qquad \begin{pmatrix} h \\ e \end{pmatrix} = \begin{pmatrix} h' \\ e' \end{pmatrix} \qquad \text{et} \qquad \begin{pmatrix} f \\ i \end{pmatrix} = \begin{pmatrix} f' \\ i' \end{pmatrix} \tag{2.17}$$

De plus, $\mathbf{F'}_{23} = -\mathbf{F'}_{32}$ donne $\alpha = \beta$, et $\mathbf{F}'$ étant défini à un facteur d'échelle près, on peut fixer $\alpha = \beta = 1$, ce qui donne :

$$\mathbf{F}' = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & hf - ei \\ 0 & ie - fh & 0 \end{bmatrix}, \tag{2.18}$$

soit

$$\mathbf{M} = \begin{bmatrix} a & b & c \\ 0 & e & f \\ 0 & h & i \end{bmatrix} \qquad \text{et} \qquad \mathbf{M'} = \begin{bmatrix} a' & b' & c' \\ 0 & e & f \\ 0 & h & i \end{bmatrix}, \tag{2.19}$$

avec

$$\begin{vmatrix} e & f \\ h & i \end{vmatrix} \neq 0. \tag{2.20}$$

Les matrices $\mathbf{M}$ et $\mathbf{M}'$ étant définies à un facteur d'échelle près, on peut fixer par exemple [1] :

$$i = 1 \tag{2.21}$$

La contrainte 2.2.1 impose que l'image par les rectifications $\mathbf{M}$ ou $\mathbf{M}'$ de l'origine (0,0,1) des images rectifiées par ($\mathbf{R}_0$,$\mathbf{R}'_0$) ne peut être envoyée à l'infini par ($\mathbf{M}$,$\mathbf{M}'$) (puisque la 3$^{\text{e}}$ coordonnée de son image ne peut être nulle), ce qui paraît raisonnable.

L'ensemble des solutions ($\mathbf{M}$,$\mathbf{M}'$) de l'équation 2.2.1 est donc une variété de dimension 9 (on a 9 variables qui peuvent être par exemple a, b, c, e, f, h, a', b', c' si on utilise la contrainte $i = 1$). L'ensemble des paires de matrices de rectification compatibles avec $\mathbf{F}$ est donc également une variété de dimension 9, qui s'en déduit par la relation 2.2.1.

1. On ne peut pas appliquer une contrainte du type $\begin{vmatrix} e & f \\ h & i \end{vmatrix} = 1$, puisque que la multiplication de $\mathbf{M}$ ou $\mathbf{M}'$ par un facteur réel ne peut changer le signe de ce déterminant, ce qui signifie qu'on se limiterait aux matrices $\mathbf{M}$ et $\mathbf{M}'$ telles que $\begin{vmatrix} e & f \\ h & i \end{vmatrix} > 0$.

Interprétation des paramètres de la variété

On peut interpréter simplement l'effet de chacun des paramètres des matrices $\mathbf{M}$ et $\mathbf{M}'$ sur les rectifications résultantes (figure 2.2) :

- a et a' sont les facteurs de dilatation selon l'axe des x dans chacune des images rectifiées ;
- b et b' permettent de régler l'inclinaison des images rectifiées ;
- c et c' sont les abscisses de l'origine de chacune des images rectifiées ;
- e est le facteur d'échelle selon l'axe des y (commun aux deux images à cause de la contrainte épipolaire) ;
- f est l'ordonnée de l'origine ;
- h correspond à une distorsion « perspective » sur l'image rectifiée ;
- i est un facteur d'échelle global.

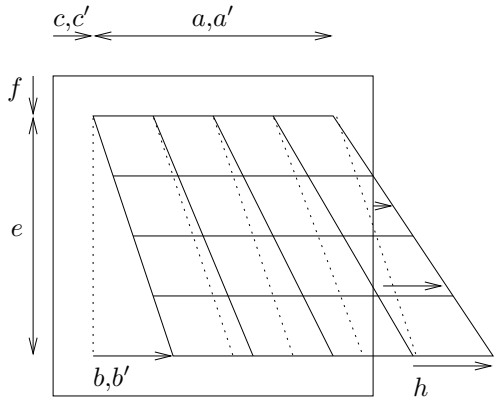


FIG. 2.2 – *Interprétation de chacun des paramètres et leur effet sur les rectifications.*

Cette fois ci, contrairement au point de vue tridimensionnel, tous les paramètres peuvent être interprétés directement comme des déformations des images. Bien que l'approche soit complètement différente, on peut établir un parallèle entre ces paramètres et ceux de l'approche tridimensionnelle, selon leurs effets sur les images (table 2.1).

3-D	α_u	θ	u_0	α'_u	θ'	u'_0	α_v	v_0	Φ
algébrique	a	b	c	a'	b'	c'	e	f	h

TAB. 2.1 – *Correspondance entre les paramètres de l'approche 3-D et de l'approche algébrique.*

2.2.2 Le point de vue géométrique

Une approche plus géométrique de la rectification permet de comprendre différemment ce qu'est la rectification et quelle est la variété de matrices de rectification autorisées.

En effet, les matrices de rectification étant des homographies, autrement dit des transformations projectives des images, elles peuvent être aussi vues comme des birapports de points – ou de droites, ce qui revient au même (figure 2.3). Une paire de rectifications compatibles est donc alors définie complètement par (on note entre parenthèses le nombre de degrés de liberté de chaque élément) :

- les épipôles e et e' dans les deux images (0 D.L.) ;
- deux points f et f', reliés par la contrainte épipolaire, et appelés *points de fuite*, qui correspondent à la direction verticale dans les images rectifiées ($4 - 1 = 3$ D.L.) ;
- deux points g et g', reliés par la contrainte épipolaire, qui correspondent aux origines des images rectifiées ($4 - 1 = 3$ D.L.) ;
- deux points h et h', reliés par la contrainte épipolaire, qui correspondent aux coins inférieurs droits des images rectifiées ($4 - 1 = 3$ D.L.).

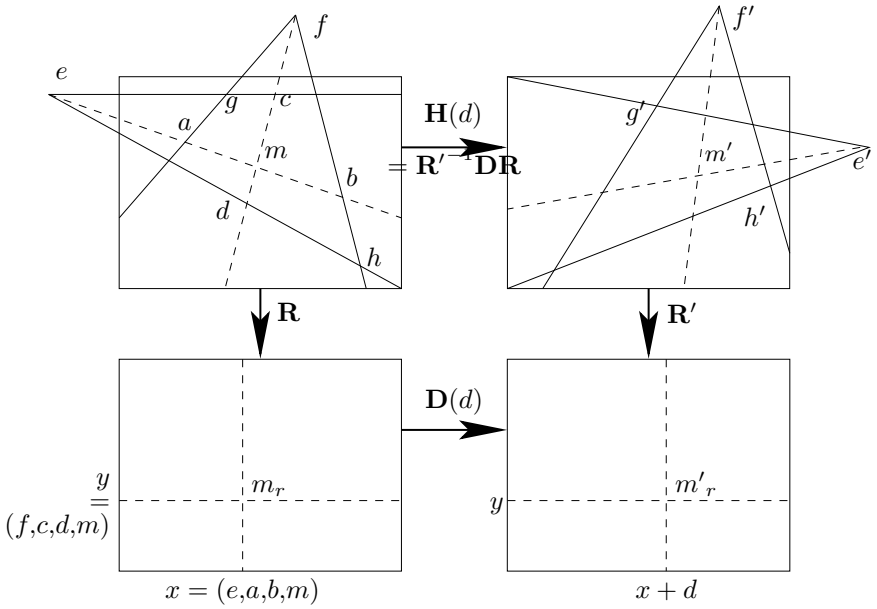


FIG. 2.3 – *Rectification à partir des épipôles* (e,e')*, de points de fuite* (f,f')*, et de points origines* (g,g')*, chacune de ces paires de points satisfaisant la contrainte épipolaire : les coordonnées dans les images rectifiées correspondent à des birapports dans les images originales.*

Les coordonnées rectifiées d'un point peuvent être déduites de ses coor-

données dans les images originales par des birapports, comme montré figure 2.3. Cette approche possède l'avantage de pouvoir visualiser directement dans les images originales la transformation effectuée par la rectification. On identifie notamment facilement les points de fuite f et f', qui sont les points qui seront envoyés à l'infini en y par les rectifications, et qui correspondent donc aux directions verticales des images rectifiées. De plus, on retrouve bien le même nombre de degrés de liberté pour le choix des rectifications : $3 + 3 + 3 = 9$ D.L.

2.3 Applications

2.3.1 Rectification pour la corrélation

Pour la stéréoscopie par corrélation, si on n'a aucune connaissance a priori sur la scène, il est préférable de choisir une paire de rectifications qui distordent le moins possible les images [Har97, ZF94b], ce qui correspond à l'hypothèse que les surfaces observées sont en moyenne fronto-parallèles. Hartley [Har97] choisit une des deux matrices de rectification telle qu'au centre de l'image correspondante la transformation soit localement euclidienne (rotation et translation), et choisit l'autre rectification de manière à minimiser la somme des carrés des disparités d'un ensemble d'appariements. Cette approche présente l'inconvénient d'être asymétrique, en ce sens qu'elle traite différemment chacune des deux images. Cependant, elle peut être facilement symétrisée : il suffit de calculer aux moindres carrés les *deux* matrices de rectification de manière à minimiser à la fois la distorsion des deux images et les disparités associées à un ensemble d'appariements.

2.3.2 Rectification par rapport à un plan

Les raisons qui peuvent mener à l'utilisation de la rectification par rapport à un plan sont diverses :

- les surfaces observées dans la scène sont en moyenne inclinées, l'hypothèse fronto-parallèle n'est donc plus valable et il est préférable pour avoir de meilleurs résultats de rectifier par rapport à cette inclinaison moyenne, par exemple l'inclinaison du sol dans des prises de vue aériennes ;
- un plan de la scène joue un rôle particulier, par exemple le plan du sol dans le contexte de la navigation d'un robot [Fau93]. La rectification par rapport à ce plan permettra de mieux détecter les obstacles : le sol aura une disparité nulle et les points situés respectivement au-dessous ou au-dessus du niveau du sol auront une disparité respectivement respectivement positive ou négative (ou le contraire, si l'image de référence est l'image droite) ;

– on veut se déplacer en évitant les obstacles situés devant un plan donné [ZF94b]. La rectification par rapport à ce plan permet de déduire la position des objets par rapport à ce plan de leur disparité : p.ex. si l'image de référence est l'image gauche, les disparités négatives correspondent aux points situés *devant* ce plan, et les disparités positives aux points derrière ce plan.

Représentation d'un plan 3-D dans la paire d'images

Un plan de l'espace peut être représenté de plusieurs manières dans un système de vision stéréoscopique faiblement calibré. Les deux représentations qui nous intéressent sont l'homographie d'une image à l'autre correspondant à ce plan et les images de trois points génériques du plan dans les deux caméras.

Soit un plan Π de $\mathcal{P}^3$, et soit un système de coordonnées projectif sur ce plan. La projection de ce plan dans la première caméra correspond à une homographie $\mathbf{H}$ de ce plan, et sa projection dans la deuxième caméra correspond à une autre homographie $\mathbf{H}'$ de ce plan. L'homographie de ce plan entre les deux images est alors $\mathbf{H}_\Pi = \mathbf{H}'\mathbf{H}^{-1}$. Rectifier par rapport au plan Π signifie que tous les points de ce plan auront une disparité nulle entre les deux images rectifiées, c'est-à-dire que, si $\mathbf{R}$ et $\mathbf{R}'$ sont les matrices de rectification correspondant à chacune des caméras,

$$\mathbf{H}_\Pi = \mathbf{R}'^{-1}\mathbf{R}. \tag{2.22}$$

La description d'un plan Π par l'homographie $\mathbf{H}_\Pi$ de ce plan entre les deux images est complètement équivalente à la donnée de trois correspondances de points du plan dans une configuration générique. En effet, on peut ajouter aux images de trois points du plan une quatrième paire de points, qui sont les images de l'intersection du plan 3-D avec la droite joignant les deux centres optiques : les épipôles. Une homographie étant définie de manière unique par quatre correspondances de points dans une configuration générique (c'est-à-dire que leurs vecteurs représentatifs de $\mathcal{P}^3$ sont linéairement indépendants), trois correspondances de points et les deux épipôles déterminent de manière unique l'homographie du plan 3-D correspondant. Réciproquement, une homographie liée à un plan 3-D permet évidemment de déterminer trois correspondances de points, ce qui termine la preuve de l'équivalence des deux descriptions. La paramétrisation de l'homographie par 3 correspondances de points permet de plus de montrer simplement que l'ensemble des homographies de plans entre les deux images est de dimension 3.

Homographie rectifiée

Définition 2 *Soit* $(\mathbf{R}_0,\mathbf{R'}_0)$ *une paire de matrices de rectification compatibles avec la géométrie épipolaire du système stéréoscopique, et soit* $\mathbf{H}_\Pi$ *l'homographie d'un plan entre les deux images non rectifiées. L'homographie de ce même plan entre les images rectifiées, appelée* homographie rectifiée, *notée* $\mathbf{H}^r_\Pi$, *vaut* $\mathbf{H}^r_\Pi = \mathbf{R}'_0\mathbf{H}_\Pi\mathbf{R}_0^{-1}$ *(figure 2.4).*

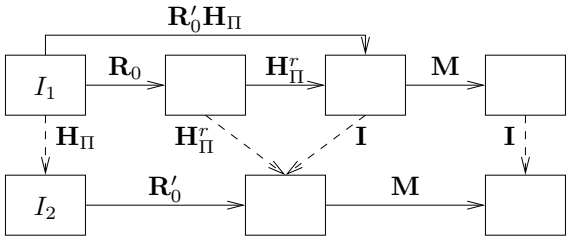


FIG. 2.4 – *Rectification par rapport à un plan : homographie du plan* $\mathbf{H}_\Pi$, *homographie rectifiée* $\mathbf{H}^r_\Pi$, *rectification par rapport au plan* $(\mathbf{R}'_0\mathbf{H}_\Pi,\mathbf{R}'_0)$, *et variété des rectifications.*

Étant donné que $\mathbf{H}^r_\Pi$ est une homographie entre deux images rectifiées, elle doit bien sûr conserver la contrainte épipolaire, c'est-à-dire que seule l'abscisse d'un point peut être modifiée par $\mathbf{H}^r_\Pi$. Autrement dit, elle s'écrit nécessairement sous la forme :

$$\mathbf{H}^r_\Pi = \begin{bmatrix} a' & b' & c' \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.23}$$

Rectification

On voit qu'une paire de matrices de rectification par rapport au plan Π s'en déduit facilement, par exemple $(\mathbf{H}^r_\Pi\mathbf{R}_0,\mathbf{R}'_0)$, ou bien en remplaçant $\mathbf{H}^r_\Pi$ par son expression (figure 2.4) :

$$(\mathbf{R}'_0\mathbf{H}_\Pi,\mathbf{R}'_0) \tag{2.24}$$

Variété des rectifications par rapport à un plan

De la même manière que dans le § 2.2.1, nous allons déterminer la variété des rectifications par rapport à un plan Π dont l'homographie entre les deux images est $\mathbf{H}_\Pi$. Soit une paire de matrices de rectifications par rapport à Π

($\mathbf{R}_0$,$\mathbf{R}'_0$), où $\mathbf{R}'_0$ peut être calculé par une méthode similaire au § 2.2.1, et $\mathbf{R}_0$ s'en déduit par l'identité $\mathbf{R}_0 = \mathbf{R}'_0\mathbf{H}_\Pi$.

L'ensemble des rectifications compatibles avec la géométrie épipolaire du système s'en déduit par l'équation 2.2.1, où la paire de matrices ($\mathbf{M}$,$\mathbf{M}'$) appartient a une variété de dimension 9 décrite par l'identité 2.19. Pour que ces rectifications soient en plus des rectifications par rapport à Π, il est nécessaire que l'homographie du plan entre les deux images rectifiées $\mathbf{H}^r_\Pi$ reste l'identité, c'est-à-dire que $\mathbf{M} = \mathbf{M}'$.

L'ensemble des rectifications par rapport à un plan est donc une variété de dimension 6 qui se déduit d'un couple ($\mathbf{R}_0$,$\mathbf{R}'_0$) de rectifications par rapport à ce plan par les équations :

$$\mathbf{R} \cong \mathbf{M}\mathbf{R}_0 \qquad \text{et} \qquad \mathbf{R}' \cong \mathbf{M}\mathbf{R}'_0 \tag{2.25}$$

où

$$\mathbf{M} = \begin{bmatrix} a & b & c \\ 0 & e & f \\ 0 & h & i \end{bmatrix} \qquad \text{et} \qquad \begin{vmatrix} e & f \\ h & i \end{vmatrix} \neq 0. \tag{2.26}$$

Ces six paramètres peuvent être déterminés de la même manière que dans le § 2.3.1, par exemple de manière à minimiser la distorsion des deux images.

2.3.3 Rectification locale

Lorsque l'on rectifie par rapport à un plan, la contrainte fronto-parallèle devient : « la surface observée doit être parallèle au plan de rectification [2] ». Pourquoi alors ne pas rectifier localement l'image en fonction de l'orientation locale de la surface ? Si on fait cette rectification locale point par point, cela correspond à la méthode de stéréoscopie par corrélation améliorée, qui sera présentée § 3.2.2. Mais cette rectification locale peut également se faire pour des régions de l'image.

Considérons le scénario suivant : pour une paire d'images donnée, on a déterminé la matrice fondamentale, et on a mis en correspondance un certain nombre de points épars dans les images. On triangule l'image de référence, par exemple par une triangulation de Delaunay, et à chaque triangle ainsi créé correspond une homographie de plan, calculée à partir des trois sommets du triangle mis en correspondance [3] (figure 2.5). Bien sûr, il est nécessaire

2. En réalité, il ne s'agit pas vraiment de parallélisme puisqu'on ne connaît a priori pas la structure affine de la scène. Cependant, dans un voisinage du plan de rectification, la contrainte correspond à un plan quasi-parallèle au plan de rectification

3. Il est utile de rappeler que les homographies rectifiées correspondantes sont des transformations affines et sont donc plus simples à calculer que les homographies dans les images originales.

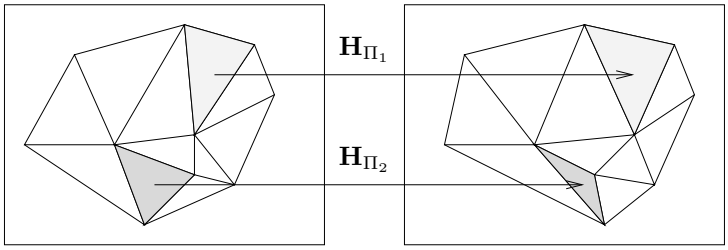


FIG. 2.5 – *Rectification locale d'une paire d'images à partir d'une triangulation dans l'image de référence des points mis en correspondance.*

que la contrainte d'ordre soit respectée pour qu'il n'y ait pas de « chevauchement » dans la triangulation de l'autre image, qui pourraient avoir des effets indésirables. Au pire, on peut éliminer par une méthode de relaxation les paires de points qui violent la contrainte d'ordre.

Après rectification locale, la disparité en chacun des sommets de la triangulation est nulle puisque l'homographie rectifiée associée au triangle est l'identité, et cette méthode de rectification permet donc à la fois de s'affranchir partiellement de la contrainte fronto-parallèle, et de réduire l'intervalle de disparité à utiliser pour la corrélation, puisque celle-ci sera nulle aux sommets de la triangulation et varie lentement. La carte de disparité alors obtenue correspond en fait aux écarts de la surface réelle par rapport au polyèdre formé par les triangles 3-D correspondant à la triangulation des images.

2.4 Conclusion

La rectification était a priori considéré comme une étape simple qui consiste seulement à transformer les images de manière à ce que chaque point de l'image de référence rectifiée ait sont correspondant dans l'autre image rectifiée à la même ordonnée. Ceci simplifie le processus de mise en correspondance, de telle sorte que, alors que la recherche de correspondances entre deux images devrait être bidimensionnelle, elle devient monodimensionelle, et elle se fait le long des lignes de l'image rectifiée. En réalité, nous avons pu voir qu'il y a beaucoup de degrés de liberté dans le choix de la méthode de rectification (globale, par rapport à un plan, ou locale) et de la rectification elle-même (9 D.L. pour la rectification globale, 6 D.L. pour la rectification par rapport à un plan). Si le but de la rectification est de préparer les images pour un algorithme de stéréoscopie par corrélation, le choix de la méthode et des paramètres de la rectification a des conséquences importantes à la fois sur la *qualité* des résultats de la corrélation (selon l'éloignement des images rectifiées de la contrainte fronto-parallèle) et sur le *temps d'exécution* de la stéréoscopie, puisqu'elle permet de réduire avan-

tageusement l'intervalle de disparité à utiliser et donc de diminuer le temps d'exécution d'autant. Cette phase de la stéréoscopie n'est donc surtout pas à négliger.

Chapitre 3

Stéréoscopie par corrélation

I left a woman waiting
I met her sometime later.
She said, *I see your eyes are dead.*
What happened to you, lover?
What happened to you, my lover?

And since she spoke the truth to me
I tried to answer truthfully:
Whatever happened to my eyes
Happened to your beauty
What happened to your beauty
Happened to me.

Leonard Cohen, I left a woman waiting

La vision stéréoscopique consiste à calculer la position de points matériels dans l'espace à partir de leurs images dans deux caméras ou plus. Le problème essentiel à résoudre alors est le problème dit de *mise en correspondance*, qui consiste à trouver les points qui se correspondent entre les images droite et gauche. Une fois les points mis en correspondance, l'information de profondeur se calcule par triangulation (c'est l'objet du chapitre suivant, sur la reconstruction). La résolution du problème de mise en correspondance a été l'objet de nombreuses recherches dans le monde entier, et les différentes approches se distinguent essentiellement par les contraintes imposées sur les objets présents dans la scène (segments, courbes, objets plans ou lisses, contours d'occultation...) et par les applications proposées (robotique mobile, photogrammétrie, réalité virtuelle, etc.).

Deux excellentes études des techniques de vision stéréoscopique permettent de faire un tour d'horizon des méthodes publiées jusqy'à récemment. Il s'agit de la synthèse de Dhond et Aggarwal [DA89], qui couvre les techniques de stéréoscopie développées jusqu'en 1989, et du rapport de Koschan [Kos93], couvrant celles publiées de 1989 à 1993. Ces deux études contiennent

une classification pertinente et minutieuse des différents algorithmes et sont vivement conseillées à tout chercheur en vision stéréoscopique. Plutôt que de paraphraser ces deux documents, nous allons nous borner à citer les éléments nouveaux dans le domaine qui nous intéresse. D'autres études comparatives sont également à noter sur le sujet, plus ancienne [BF82] ou plus récentes [FFH+92, BBH93].

Les méthodes de stéréoscopie peuvent pour la plupart être séparées en deux classes : celles à base de primitives *(feature-based)* et les méthodes surfaciques *(area-based)*.

Méthodes à base de primitives. Beaucoup de domaines de la vision par ordinateur, qu'il s'agisse de la stéréoscopie, de la reconnaissance d'objets, ou du suivi d'objets *(tracking)*, proposent des approches à base de primitives. La définition d'une primitive est arbitraire et la seule généralisation qu'on puisse en faire est que c'est en quelque sorte une représentation fidèle et utile d'une image. Les primitives ont en général les propriétés suivantes : unicité, répétabilité, et signification physique. Dans le contexte de la stéréoscopie, le but des approches par primitives est d'obtenir des correspondances fiables, même en présence d'une certaine quantité de bruit dans les images.

Étant donné que le but de la stéréoscopie est de reconstruire une scène en trois dimensions, les primitives utiles en stéréo sont celles comportant une information sur la structure 3-D de la scène. En général, et surtout dans les environnements construits par l'homme, la structure 3-D peut être décrite par des contours et des intersections de contours (donc des coins). C'est pourquoi la plupart des approches par primitives se sont intéressées aux contours et aux coins. Les contours sont obtenus par un détecteur de contours de type Canny [Can86] (cf. annexe C), puis chaînés pour obtenir des chaînes de contours, et parfois des primitives de plus haut niveau sont extraites de ces chaînes de contours (segments, polygones, courbes splines). Certaines méthodes utilisent d'autres types de primitives, comme des régions [CVSG89, LCK93] ou des structures topologiques [Fle91]. L'avantage de ces méthodes est qu'à la fois elles permettent plus de flexibilité dans le choix des algorithmes (relaxation [PMF85], programmation dynamique [BB81, OK85, SB94], prédiction-vérification [MN85, AF87, FR96], ...), elles sont beaucoup plus rapides grâce à la réduction de l'information contenue dans les images, et elles permettent d'obtenir une précision égale à celle des primitives extraites des images. Cependant, elles sont fortement limitées par le nombre de ces primitives présentes dans les images, et leur résultat sont des données 3-D éparses. En conséquence, on assiste depuis quelques années à un retour vers les méthodes surfaciques, qui donnent des résultats denses, beaucoup plus intéressants dans de nombreuses applications.

Méthodes surfaciques Les méthodes surfaciques sont nées de la constatation que deux images sont *localement* semblables autour des points se correspondant, puisque localement la projection de la surface 3-D associée aux voisinages de ces points est similaire, d'où la dénomination *surfacique*. Ces méthodes consistent à fixer d'abord un point d'intérêt dans une image, puis, à l'aide d'une mesure de corrélation de type distance euclidienne ou produit scalaire, à chercher un point dans l'autre image dont le voisinage soit semblable à celui du point de référence.

La stéréoscopie surfacique a l'avantage sur les méthodes à base de primitives de fournir une représentation dense de la scène observée : en effet, on peut chercher un correspondant pour tous les points de l'image de référence, même si on ne les trouve pas nécessairement. Par contre, pour que les voisinages des points se correspondant aient des fonctions d'intensité similaires, il est théoriquement nécessaire que les surfaces observées satisfassent les contraintes photométriques et géométriques suivantes :

- **contrainte lambertienne :** les surfaces doivent en chaque point être *lambertiennes*, c'est-à-dire que l'intensité de la projection dans chaque image d'un point 3-D doit être indépendante du point de vue (en particulier, il ne doit pas y avoir de réflections spéculaires sur les surfaces observées) ;
- **contrainte fronto-parallèle :** les surfaces doivent être *fronto-parallèles*, c'est-à-dire parallèles aux plans rétiniens des deux caméras.
- **contrainte de continuité :** la plupart des méthodes font l'hypothèse que les surfaces sont au moins localement continues. Aucune méthode de stéréoscopie surfacique à ce jour n'est capable de mettre en correspondance de manière satisfaisante et de reconstruire des objets comportant une texture 3-D complexe comme des buissons ou des arbres.

Dans la pratique, ces contraintes sont plutôt lâches. En effet, la contrainte lambertienne peut devenir « localement lambertienne » si on normalise localement les intensités, comme on le verra plus bas, ou si plutôt que d'appliquer la méthode aux images brutes on l'applique à des images filtrées, par exemple par un laplacien de gaussienne (LOG) [Nis84]. De cette manière, l'algorithme cherchera à mettre en correspondance les *variations locales* de l'intensité, et non plus l'intensité elle-même. Une meilleure solution, cependant, serait d'inclure une fonction de réflectance [KvDS96, ON94] dans la fonction de coût utilisée pour la mise en correspondance ; hélas, de telles méthodes nécessiteraient également de connaître la position des sources lumineuses.

De même, la contrainte fronto-parallèle n'a pas besoin d'être strictement vérifiée, et même des surfaces légèrement inclinées par rapport aux plans rétiniens des caméras pourront être mises en correspondance. Certaines méthodes permettent même de s'affranchir complètement de cette contrainte en tenant compte de la déformation locale de la surface lors de sa projection

dans les deux images, comme celle que nous présentons plus loin (§ 3.2).

Les méthodes surfaciques ont peu évolué depuis l'état des lieux dressé par KOSCHAN en 1993. Globalement, les méthodes se sont stabilisées et, à quelques exceptions près, les nouveautés ont été plutôt techniques (parallélisation, réalisations et implantations matérielles, temps réel) qu'algorithmiques. Les recherches se sont concentrées essentiellement sur les problèmes fort intéressants que sont le traitement des occlusion, des discontinuités, et des surfaces non fronto-parallèles. Pour ce qui est du traitement des discontinuités de profondeur, les algorithmes de type « fenêtres adaptatives » ont largement gagné leurs lettres de noblesse [KO94, Lot96]. Cependant ils ne permettent de traiter qu'une partie des discontinuités de profondeur, puisqu'ils conservent la contrainte fronto-parallèle : les contours d'occultation d'une surface, pour lesquels la normale à la surface devient orthogonale au rayon optique, peuvent générer des discontinuités de profondeur qui ne pourront pas être traitées correctement par les fenêtres adaptatives, l'inclinaison de la surface étant trop importante. De plus, ils considèrent que tous les contours sont des images de discontinuités de profondeur, ce qui est souvent loin d'être vrai.

Une méthode permettant de traiter à la fois les discontinuités, les occlusions, et les surfaces inclinées est l'approche bayesienne telle qu'elle a été proposée par exemple par BELHUMEUR [Bel96, Bel93, BM92], par GEIGER *et al.* [GLY92], par CHANG et CHATTERJEE [CC90, CCK91]. ou par STEWART *et al.* [SFB96]. Une autre solution a été proposée pour résoudre le problème des occlusions et des discontinuités avec une approche par régularisation, en introduisant les contours de discontinuité [RD96], ou bien dans une approche par des champs de Markov, en introduisant des processus de ligne binaires pour modéliser ces discontinuités. Certaines méthodes de stéréoscopie multi-caméras (c'est-à-dire à au moins 3 caméras) prennent également en compte *explicitement* la présence d'occlusions [SD88, NMSO96], soit la possibilité qu'un élément de surface ne soit pas vu sur *toutes* les images.

Dans ce chapitre, nous allons commencer par la présentation en détails d'une implémentation d'une méthode classique de stéréoscopie par corrélation, qui donne déjà d'excellents résultats [Fua91, HZF93]. Ensuite, nous donnerons la description d'une méthode appelé *corrélation fine* permettant non seulement la prise en compte lors de la corrélation de l'inclinaison ou des courbures de la surface, mais aussi leur calcul, directement à partir des données d'intensité des images brutes.

3.1 Méthode classique

La méthode présentée ici, basée sur les travaux de FUA, a déjà subi quelques implantations avec succès [Fua91, HZF93, FHM$^+$93]. C'est une méthode de stéréoscopie par corrélation classique, augmentée d'une phase

de validation permettant de préserver les discontinuités de profondeur et les occlusions [Fua91].

Le problème de la mise en correspondance est simplifié par la rectification de la paire d'images stereoscopiques (chapitre 2), ce qui permet a la fois de limiter la recherche des correspondances a une recherche monodimensionnelle le long des droites epipolaires et de limiter la classe des transformations locales de la surface d'une image a l'autre (en eliminant par exemple les rotations du plan). Les hypothèses sur la surface observée sont les hypothèses classiques de la stéréoscopie surfacique : on suppose que les surfaces sont localement lambertiennes et quasi-fronto-parallèles. Le principe du calcul de la carte de disparité est alors, pour chaque point de la première image :

1° calculer la corrélation entre une fenêtre $l \times h$ de l'image d'intensité centrée en ce point et une fenêtre de même taille centrée en chaque point de l'autre image susceptible de lui correspondre, c'est-à-dire qui a la même ordonnée, puisque les images sont rectifiées ;

2° choisir comme correspondant de ce point celui qui maximise la mesure de corrélation, et en déduire la disparité.

On en déduit alors une carte dense de disparité pour toute l'image, qui peut s'écrire sous la forme d'une fonction :

$$\begin{aligned} d : [0,L] \times [0,H] &\rightarrow \Re \\ (u_1,v_1) &\longmapsto d(u_1,v_1). \end{aligned} \tag{3.1}$$

Cette fonction est définie partout sauf en quelques régions éparses où aucune correspondance n'a été trouvée, parce qu'il y a une occlusion, une discontinuité, ou que la surface ne satisfait pas aux contraintes liées à la méthode. Avant de décrire l'algorithme proprement dit, il est nécessaire de savoir comment on choisit l'intervalle des disparités admissibles (c'est-à-dire quels points de la deuxième image sont susceptibles de correspondre à un point de la première image), et quelle mesure ou critère de corrélation utiliser pour mesurer la ressemblance de deux fenêtres des images.

3.1.1 Choix de l'intervalle de disparité

Il est important de savoir restreindre l'espace de recherche des correspondances dans la deuxième image, et ce pour deux raisons. La première est qu'évidemment, plus le nombre de « candidats à la correspondance » est grand, plus le temps de calcul sera long. La seconde est que plus il y a de candidats, plus on a de chances d'effectuer un faux-appariement, c'est-à-dire d'apparier deux points des images qui ne correspondent pas à un point 3-D physique. On réduit donc cet espace de recherche par le choix d'un intervalle de disparité, correspondant à un segment de la droite épipolaire où chercher les correspondances. Dans une approche hiérarchique ou multi-échelle [Fua91], on peut utiliser les disparités calculées à une résolution grossière

pour déterminer l'intervalle de disparité à utiliser à une résolution plus fine. Éventuellement, cet intervalle peut être calculé localement, en découpant l'image en tuiles et en calculant l'intervalle de variation de la disparité sur chaque tuile. Cependant, l'algorithme présenté ici étant optimisé pour un intervalle de disparité global, il sera alors nécessaire de l'appliquer indépendamment à chaque tuile de l'image, puis de recombiner les cartes de disparité ainsi obtenues.

Le choix de l'intervalle de disparité peut se faire en fonction de la distance des objets à la caméra, grâce à la formule :

$$\delta = \frac{b \times f}{d} + u_1 - u_2 \tag{3.2}$$

où δ est la disparité, b est la ligne de base (distance entre les deux caméras), f est la distance focale de la caméra, et d est la distance de l'objet à la caméra. u_1 et u_2 sont les abscisses d'un point vu respectivement dans la caméra 1 (référence) et dans la caméra 2. Il suffit alors de calculer le minimum et le maximum de cette expression pour toutes les valeurs extrêmales de u_1, u_2 et d afin obtenir l'intervalle de disparité à utiliser.

En pratique, on a souvent préféré déterminer cet intervalle en mesurant la disparité entre certains points d'intérêt des images, détectés automatiquement (par exemple lors du calibrage faible, par une méthode de type [ZDFL95]) ou choisis manuellement, ce qui permet de connaître très précisément l'intervalle à utiliser pour une paire d'images donnée.

3.1.2 Critères de corrélation

Pour obtenir de bons résultats il est nécessaire de choisir un critère de corrélation, c'est-à-dire une mesure de la corrélation entre deux fenêtres des images, qui permette d'obtenir les bonnes correspondances quelle que soit la situation. Pour notre application, les qualités requises pour le critère de corrélation sont :

- permettre de discriminer nettement un bon appariement parmi les points candidats ;
- être robuste aux différentes formes de bruit présentes dans les images (photométrique ou géométrique[1]) ;
- répondre positivement même si les réglages des deux caméras sont légèrement différents : gain et offset (soit la pente et l'ordonnée à l'origine de la courbe de réponse), mise au point (ou toute forme de flou dans les images), ou focale ;

1. Un exemple de bruit géométrique est le « pixel jitter » des caméras analogiques, qui correspond à un bruit dans la synchronisation des lignes de la caméra et se manifeste par un décalage de l'origine de chaque ligne, de l'ordre du pixel.

– permettre la mise en correspondance de surfaces qui ne satisfont que vaguement les contraintes lambertienne et fronto-parallèle, c'est-à-dire des surfaces légèrement inclinées ou quasi-lambertiennes.

La plupart des critères de corrélation dérivent de deux mesures de base : la distance euclidienne entre les deux vecteurs formés par les intensités des images (notée SSD, comme *Sum of Squared Differences*), et le produit scalaire de ces deux vecteurs (noté CC, comme *Cross-Correlation* ou corrélation croisée). Il ressort d'une publication comparant une vingtaine de critères de corrélation [AG92] et de tests que nous avons effectués sur nos paires d'images, que les trois critères de corrélation donnant les meilleurs résultats quelles que soient les conditions de prise de vue sont les critères ZSSD, ZNSSD, et ZNCC, que nous détaillons ci-après, et que le critère classique de distance euclidienne (SSD) reste très performant sous certaines contraintes de prises de vue. Pour chacun de ces critères c,

$$\begin{aligned} c : [0;L] \times [0,H] \times [d_{\min};d_{\max}] &\longrightarrow \Re \\ (x,y,d) &\longmapsto c_{x,y}(d) \end{aligned} \tag{3.3}$$

la disparité attribuée au point (x,y) est :

$$d(x,y) = \arg\max_{d \in]d_{\min};d_{\max}[} c_{x,y}(d) \tag{3.4}$$

Si le maximum est atteint en une des bornes (ce qui signifie que l'intervalle de disparité a sans doute été mal choisi), aucune valeur n'est affectée à la disparité.

Le critère SSD

Le critère SSD *(sum of squared differences)* correspond à minimiser la somme des différences des intensités sur l'ensemble de la fenêtre de corrélation :

$$c_{x,y}(d) = - \sum_{-l \le i \le l; -h \le j \le h} (I_1(x+i,y+j) - I_2(x+d+i,y+j)) \tag{3.5}$$

où I_1 et I_2 sont les fonctions d'intensité dans chacune des deux images. Ce critère est très sensible aux différences d'illumination entre les deux images.

Le critère ZSSD

Le critère ZSSD *(zero-mean sum of squared differences)* consiste à minimiser la somme des différences de l'écart des intensités à leur moyenne calculée sur l'ensemble de la fenêtre de corrélation :

$$\begin{aligned} c_{x,y}(d) = - \sum_{i,j} &((I_1(x+i,y+j) - \overline{I_1}(x,y)) \\ &- (I_2(x+d+i,y+j) - \overline{I_2}(x,y)))^2 \end{aligned} \tag{3.6}$$

où I_1 et I_2 sont les fonctions d'intensité dans chacune des deux images, et où $\overline{I_1}$ et $\overline{I_2}$ sont le moyennes de ces intensités sur chaque fenêtre de corrélation centrée en (x,y). Ce critère permet donc de mettre en correspondance des surfaces dont l'illumination est légèrement différente, et d'être insensible aux différences d'offset dans les fonctions de transfert de chacune des caméras.

Dans la pratique, on considère que les fonctions $\overline{I_1}$ et $\overline{I_2}$ varient peu sur la fenêtre de corrélation, et on utilise le critère suivant :

$$\begin{aligned} c_{x,y}(d) = &- \sum_{i,j}((I_1(x+i,y+j) - \overline{I_1}(x+i,y+j)) \\ &- (I_2(x+d+i,y+j) - \overline{I_2}(x+d+i,y+j)))^2 \\ = &- \sum_{i,j}(I'_1(x+i,y+j) - I'_2(x+d+i,y+j))^2 \end{aligned} \tag{3.7}$$

On voit que c'est équivalent à filtrer d'abord les images pour obtenir les images I'_1 et I'_2 puis à appliquer la méthode SSD.

Le critère ZNSSD

Le critère ZNSSD *(zero-mean sum of squared differences)* consiste à minimiser la somme des différences des intensités filtées par la moyenne sur l'ensemble de la fenêtre de corrélation, normalisée par la variance locale des intensités :

$$\begin{aligned} c_{x,y}(d) = &- \sum_{i,j}((I_1(x+i,y+j) - \overline{I_1}(x,y)) \\ &\quad - (I_2(x+d+i,y+j) - \overline{I_2}(x,y)))^2 \\ &/ \sqrt{\sum_{i,j}(I_1(x+i,y+j) - \overline{I_1}(x,y))^2} \\ &\quad \sqrt{\sum_{i,j}(I_2(x+d+i,y+j) - \overline{I_2}(x,y))^2} \end{aligned} \tag{3.8}$$

On effectue la même simplification que précédemment, et on remarque au plus qu'un des deux termes du dénominateur est constant lorsque d varie, pour obtenir le critère simplifié :

$$c_{x,y}(d) = -\frac{\sum_{i,j}(I'_1(x+i,y+j) - I'_2(x+d+i,y+j))^2}{\sqrt{\sum_{i,j}\overline{I_2}(x+d+i,y+j)^2}} \tag{3.9}$$

Ce critère moyenné et normalisé permet de s'affranchir complètement des différences de gain et d'offset des deux caméras.

Le critère CC

Le critère CC (*cross-correlation*) est en fait le produit scalaire des deux vecteurs d'intensité :

$$c_{x,y}(d) = \sum_{i,j} I_1(x+i,y+j) I_2(x+d+i,y+j) \tag{3.10}$$

Dans la pratique, ce critère est inutilisable puisqu'il favorise les zones d'intensité élevée, par forcément similaires à la fenêtre de corrélation de l'image de référence. On lui préfère largement le critère normalisé ZNCC.

Le critère ZNCC

Le critère ZNCC *(zero-mean normalized cross-correlation)* est en fait le cosinus des vecteurs d'intensité. Plus il est proche de 1, plus les vecteurs sont semblables, indépendamment des problèmes de gain et d'offset des caméras :

$$\begin{aligned} c_{x,y}(d) = \sum_{i,j} & (I_1(x+i,y+j) - \overline{I_1}(x,y)) \\ & (I_2(x+d+i,y+j) - \overline{I_2}(x,y)) \\ & \Big/ \sqrt{\sum_{i,j} (I_1(x+i,y+j) - \overline{I_1}(x,y))^2} \\ & \sqrt{\sum_{i,j} (I_2(x+d+i,y+j) - \overline{I_2}(x,y))^2} \end{aligned} \tag{3.11}$$

Soit en version simplifiée :

$$c_{x,y}(d) = \frac{\sum_{i,j} I'_1(x+i,y+j) I'_2(x+d+i,y+j)}{\sqrt{\sum_{i,j} \overline{I_2}(x+d+i,y+j)^2}} \tag{3.12}$$

Comparaison

Dans la pratique, on a pu noter que les critères qui donnent les meilleurs résultats sont ZSSD, ZNSSD et ZNCC. ZSSD et ZNSSD donnent des résultats comparables, mais ZNSSD requiert en plus le calcul des variances d'une des images, et on pourra donc lui préférer le premier si la rapidité d'exécution est un critère déterminant. ZNCC donne des résultats de qualité similaire, mais on a pu noter qu'il met plus facilement en correspondance des zones d'intensités très différentes (du blanc avec du noir), et génère donc un peu plus de faux-appariements.

3.1.3 Algorithme

Plutôt que de donner les détails de l'algorithme, nous nous contenterons d'en donner le fil directeur, et le code source commenté de la fonction C correspondante est disponible en annexe E.1.

L'idée de l'algorithme est que, pour une valeur de la disparité fixée, calculer le critère de corrélation en chacun des points revient à décaler les deux images de la valeur de la disparité, à en calculer la différence au carré (critères de type SSD) ou le produit point à point (critères de type CC), et à effectuer la convolution de l'image résultat par un masque de la taille de la fenêtre de corrélation.

De plus, il est connu que la convolution par un masque rectangulaire peut être calculée en un temps proportionnel à la taille de l'image, et indépendant de la taille du masque, en utilisant l'optimisation dans les deux directions expliquée graphiquement par la figure 3.1.

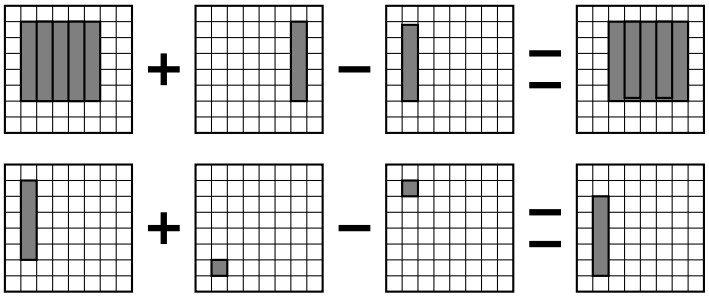


FIG. 3.1 – *Optimisations de la stéréoscopie par corrélation dans les deux directions : à disparité constante, le calcul du score est simplement une convolution.*

Puisqu'on doit faire cette convolution pour chacune des valeurs de la disparité, le temps de calcul sera également proportionnel à la taille de l'intervalle de disparité. Le calcul de la carte de disparité pour des images de taille $L \times H$, pour un intervalle de disparité de taille D, est donc proportionnel au produit $L \times H \times D$, et est indépendant de la taille de la fenêtre de corrélation. Si on a le choix de la résolution de l'image, ou bien dans le cas des méthodes de corrélation hiérarchiques [Fua91, KRS96], l'intervalle de disparité étant lui-même proportionnel à la dimension de l'image (p.ex. la taille d'un de ses côtés), le temps d'exécution est donc proportionnel à *la dimension de l'image au cube*. Le choix de la résolution de travail est donc crucial pour le temps d'exécution du calcul de la carte de disparité.

Nous avons pu remarquer au cours des expérimentations sur différents types de processeurs que le temps de calcul n'était pas proportionel aux spécifications de vitesse du processeur (bancs d'essais de type MIPS ou SPEC). En effet, la méthode de convolution optimisée utilisée par cette méthode est optimale dans le cas où le processeur exécute les instructions de ma-

nière séquentielle, mais elle est très mal adaptée aux architectures de type « pipeline » (permettant d'exécuter plusieurs instructions en même temps), puisque localement elle n'est pas parallélisable : les résultats au point suivant dépendent complètement des résultats au point courant. En conséquence, sur un processeur MIPS R8000 à 75MHz (SGI Power Indigo2) côté à près de 300 SPECfp92, la corrélation n'est pas beaucoup plus rapide que sur un MIPS R4400 à 150MHz (SGI Indy) côté à environ 100 SPECfp92, et ce parce que le gain de vitesse en calcul flottant du R8000 est essentiellement dû à un pipeline d'instructions machines plus large, la cadence d'exécution de chaque instruction étant sensiblement la même.

3.1.4 Affinage de la disparité

La méthode que nous avons décrite donne comme résultat des valeurs entières, exprimées en pixels, correpondant à la valeur de la disparité qui maximise le critère. Or nous avons besoin d'une précision supérieure au pixel pour éviter un effet « marches d'escalier » et avoir des surfaces lisses. La méthode que nous utilisons pour calculer la disparité à une précision inférieure au pixel est de prendre comme valeur de la disparité non pas celle maximisant [2] le critère de corrélation $c(d)$ (appelons-la d_0, c'est un entier), mais le maximum d'une fonction passant par les points $(d_0 - 1, c(d_0 - 1))$, $(d_0, c(d_0))$, et $(d_0 + 1, c(d_0 + 1))$.

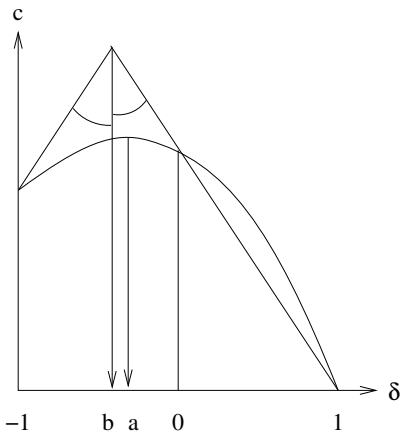


FIG. 3.2 – *Approximation de la disparité à une précision inférieure au pixel comme étant de maximum de la parabole passant par trois valeurs (a) ou par la méthode dite « du toît » (b).*

Nous avons testé deux types de fonction d'interpolation sous-pixélique

2. On défini un maximum local d'une fonction discrète comme étant supérieur strictement à la valeur précédente et supérieur ou égal à la valeur suivante.

du maximum du score (figure 3.2), la fonction « parabole »[3] :

$$d = d_0 + \frac{1}{2}\frac{c(d_0+1) - c(d_0-1)}{(c(d_0) - c(d_0+1)) + (c(d_0) - c(d_0-1))} \tag{3.13}$$

et la fonction « toît » :

$$d = d_0 + \begin{cases} \frac{1}{2}\frac{c(d_0+1)-c(d_0-1)}{c(d_0)-c(d_0-1)} & \text{si } c(d_0+1) > c(d_0-1), \\ \frac{1}{2}\frac{c(d_0+1)-c(d_0-1)}{c(d_0+1)-c(d_0)} & \text{si } c(d_0+1) < c(d_0-1). \end{cases} \tag{3.14}$$

Pour choisir entre ces deux fonctions, nous avons utlisé des tests d'auto-corrélation, c'est-à-dire que nous avons calculé des cartes de disparité en prenant deux fois la même image, puis nous avons mesuré l'écart type des disparités sous-pixéliques ainsi calculées, pour les deux méthodes d'interpolation présentées ci-dessus et pour les différents critères de corrélation.

critère \ image	Hervé	buste	mars
SSD	0.0623	0.0399	0.0812
ZSSD	0.0594	0.0382	0.0724
ZNSSD	0.0531	0.0378	0.0611
ZNCC	0.0569	0.0389	0.0648

TAB. 3.1 – *Écart type de la disparité dans les cartes d'auto-corrélation calculées avec différentes images et différents critères, pour la méthode de la parabole.*

critère \ image	Hervé	buste	mars
SSD	0.0964	0.0672	0.1181
ZSSD	0.0922	0.0660	0.1094
ZNSSD	0.0873	0.0665	0.0982
ZNCC	0.0907	0.0676	0.1024

TAB. 3.2 – *Écart type de la disparité dans les cartes d'auto-corrélation calculées avec différentes images et différents critères, pour la méthode du toît.*

Les précisions ainsi obtenues (tables 3.1 et 3.2) ne doivent pas être interprétées comme étant les précisions de chacun des critères pur la stéréoscopie, puisqu'il s'agit là uniquement d'auto-corrélation, par contre il en ressort clairement que la méthode de la parabole donne des résultats plus précis que

3. Il est nécessaire de ne pas simplifier le dénominateur pour éviter les erreurs d'arrondi. En effet, le calcul de $2c(d_0) - c(d_0+1) - c(d_0-1)$ avec $c(d_0-1) < c(d_0) \leq c(d_0+1)$ en virgule flottante peut donner 0 dans certains cas délicats, alors que celui proposé ne s'annule jamais.

la méthode du toît. De plus, on remarque que plus l'image est texturée, meilleure est la précision de l'approximation sous-pixélique.

Il est difficile de juger si cette interpolation apporte des informations qui ont une *réalité physique* : il faudrait pour celà utiliser une surface dont on connaît précisément la géométrie et la position. Au vu des cartes de disparité montrées en exemple plus loin, on peut tout de même déjà remarquer que la disparité « affinée » a un aspect plus lisse que la disparité « entière », ce qui laisse penser qu'on obtient effectivement une information à une précision meilleure que le pixel.

3.1.5 Validation

L'étape de validation permet d'éliminer le plus possible de points erronés dans la carte de disparité et de tenir compte ainsi de la présence d'occlusions ou de discontinuités de profondeur [Fua91]. Elle consiste à calculer la carte de disparité $d_1(u_1,v_1)$ de la deuxième image par rapport à la première, ainsi que la carte de disparité $d_2(u_2,v_2)$ de la première image par rapport à la seconde, puis à comparer la cohérence des résultats, c'est-à-dire qu'on doit avoir théoriquement [4] :

$$d_1(u_1,v_1) = -d_2(u_1 + d_1(u_1,v_1),v_1)$$

Or dans notre cas les cartes denses de disparité sont discrètes, et nous n'avons la valeur de la disparité que pour les points de coordonnées entières. Le test de validation des cartes de disparité devient donc :

$$|d_1(u_1,v_1) + d_2(u_1 + [d_1(u_1,v_1)],v_1)| \leq s \tag{3.15}$$

où $[\cdot]$ est un opérateur d'arrondi à l'entier le plus proche et s est un seuil à fixer (en général on prend $s \approx 1$).

3.1.6 Images en couleurs

Peu de chercheurs se sont intéressées au problème de la couleur en vision par ordinateur [Fau79, Luo91] – je ne parle pas de traîtement d'images – et encore moins à son utilisation en vision stéréoscopique [Kos94, KRS96, JB92, OYT92]. Mais la vulgarisation des caméras couleurs (appareils photo numériques, PhotoCD, ...) et du matériel permettant l'acquisition des images en couleurs (notamment sur les PC) peut laisser penser que l'intérêt pour la couleur reviendra sous peu.

Motivations

L'utilisation d'images en couleurs permet d'avoir en chaque pixel trois informations (par exemple rouge, vert, bleu) au lieu d'une (la luminance) pour

4. Il est utile de rappeler que comme on se trouve en coordonnées rectifiées, on a $v_1 = v_2$

les images noir et blanc. Mais il est connu que les trois canaux d'une image couleurs sont fortements corrélés [WS67], et l'information apportée par l'utilisation des images couleur n'est donc pas proportionnelle à la quantité de données supplémentaire à traiter. Cependant, on peut espérer que la couleur permettra d'améliorer les résultats de la stéréoscopie par corrélation de deux façons :

1° les informations de couleur permettront de lever des ambiguites qui pourraient exister dans le cas d'images monochromatiques, et d'eliminer des faux appariements (par exemple lorsque seule l'information chromatique permettra de distinguer deux appariements qui auraient un score de corrélation égal si on n'utilisait que l'information de luminance) ;

2° la courbe de score de corrélation $c_{x,y(d)}$ devrait présenter un pic plus net du fait de l'utilisation de données supplémentaires redondantes, et le maximum sera donc plus net et sa position plus précise.

Éventuellement, dans le cas où la scène peut être éclairée avec une lumière structurée synthétique [KRS96], on pourra décorréler totalement les informations des trois canaux RGB, et donc effectivement obtenir trois fois plus d'informations qu'avec une image de luminance.

Le peu de recherches réalisées sur l'utilisation de la couleur en stéréoscopie se sont intéressées à différents types d'algorithmes : le recuit simulé, qui est trop lent et inutilisable pour des applications temps-réel [JB92], une méthode itérative dont la vitesse de convergence est a priori inconnue [OYT92], et une méthode de stéréoscopie par corrélation similaire à la notre [KRS96]. Dans cette dernière, cependant, l'auteur a oublié un problème important, celui du choix de l'espace des couleurs dans lequel travailler.

Choix de l'espace de couleurs

Avant de traiter des images en couleurs, il est nécessaire de choisir une représentation de ces images, c'est-à-dire de choisir un espace de couleurs, espace à trois dimensions permettant de paramétrer la valeur de chaque pixel d'une image couleur. De nombreux espaces de couleurs ont été élaborés, dans des buts différents, et chacun a ses propriétés, ses avantages et ses inconvénients [WS67, Poy]. En un mot, aucun de ces espaces n'est parfait. On peut être tenté par l'utilisation d'un espace « perceptuellement uniforme » , c'est-à-dire pour lequel la distance euclidienne entre deux points de l'espace est proportionelle à la différence de perception entre ces deux couleurs par l'œil humain. Dans la pratique, la conversion entre l'espace RGB et de tels espaces est compliquée, et le jeu n'en vaut peut-être pas la chandelle...

Pour tout espace de couleurs, on peut définir son « uniformité perceptuelle » comme étant le rapport entre la plus grande différence de perception provoquée par un petit déplacement dans l'espace, et la plus petite différence de perception provoquée par un déplacement de la même amplitude. Des

espaces de couleurs ont été élaborés, qui possèdent une grande uniformité perceptuelle, comme les espaces $L*a*b$ ou $L*u*v$, définis par la CIE (Comission Internationale de l'Éclairage), dont l'uniformité est d'environ 6:1. Par contre, l'uniformité de l'espace RGB linéaire est très mauvaise (de l'ordre de 80:1), mais une simple correction gamma de chacun des canaux donne l'espace R'G'B' qui est relativement uniforme [Poy]. La correction gamma consiste à appliquer à chacune des composantes la fonction :

$$\begin{aligned} \Gamma : [0;1] &\longrightarrow [0;1] \\ x &\longmapsto x^{0.45} \end{aligned} \tag{3.16}$$

L'uniformité perceptuelle de l'espace R'G'B' en fait donc un bon candidat pour notre choix de l'espace de couleurs, mais le plus grand avantage de l'espace R'G'B' est que la conversion dans cet espace ne nécessite en pratique aucun calcul : la plupart des caméras vidéo proposent une correction gamma intégrée, et dans le standard international de transmission vidéo, tous les canaux sont corrigés gamma (en particulier dans le signal S-VHS YCbCr, le signal RGB subit une correction gamma avant d'être transformé en YCbCr). À l'opposé, les espaces L*a*b et L*u*v requièrent de nombreux calculs [WS67] et nécessiteraient une charge supplémentaire de traitement importante pour chaque paire d'images.

L'utilisation d'espaces non linéaires pose cependant un problème : la normalisation et le filtrage par la moyenne utilisés dans les critères de corrélation que nous avons vu précédemment supposent que la fonction de transfert de la caméra est linéaire, ce qui n'est pas le cas lorsqu'on utilise la correction gamma. La littérature du traitement d'images fait rarement le discernement entre un codage linéaire et un codage non-linéaire. Par exemple il n'est fait aucune mention de la fonction de transfert dans les standards MPEG ou JPEG, mais celle-ci est implicite : on obtient de mauvais résultats si on utilise un codage linéaire. Les standards de graphisme par ordinateur, comme PHIGS, CGM ou OpenGL ne font pas non plus mention de la fonction de transfert à utiliser. Ce type d'ambiguïté rend difficile l'échange de données entre les systèmes, d'où l'instauration du format d'images PNG sur le World Wide Web, qui fait mention explicite de la correction gamma utilisée pour chaque image. Si vous demandez à un ingénieur vidéo si son système est linéaire, il pensera « bien sûr ! » en pensant à une tension linéaire (système RGB), et un ingénieur opticien vous répondra la même chose en pensant à une intensité linéaire (système R'G'B'). Mais puisque la transformation pour passer de l'un à l'autre est non-linéaire, une transformation linéaire effectuée dans un des systèmes sera non-linéaire dans l'autre.

Il a été difficile de juger lors des expériences de l'impact que pouvait avoir l'utilisation de l'espace non linéaire R'G'B' par rapport à l'espace linéaire RGB, et nous n'avons donc pas de réponse nette à apporter au problème du choix de l'espace des couleurs, bien que nous ayons utilisé R'G'B' lors de nos expériences.

Réalisation

Les critères de corrélation et l'algorithme de stéréo par corrélation décrits précédemment s'adaptent très facilement à la corrélation couleur, ou plus généralement multi-canaux. À titre d'exemple, voici les critères ZSSD et ZNCC simplifiés (équations 3.1.2 et 3.1.2) adaptés à la corrélation multi-canaux. $I_1(x,y,c)$ est l'intensité du canal c du pixel (x,y), éventuellement corrigée. Le critère ZNSSD simplifié couleur s'écrit :

$$c_{x,y}(d) = -\frac{\sum_{i,j,c} \left(I'_1(x+i,y+j,c) - I'_2(x+d+i,y+j,c)\right)^2}{\sqrt{\sum_{i,j,c} \overline{I_2}(x+d+i,y+j,c)^2}} \tag{3.17}$$

Et le critère ZNCC simplifié couleur :

$$c_{x,y}(d) = \frac{\sum_{i,j,c} I'_1(x+i,y+j,c) I'_2(x+d+i,y+j,c)}{\sqrt{\sum_{i,j} \overline{I_2}(x+d+i,y+j,c)^2}} \tag{3.18}$$

L'algorithme optimisé est, lui aussi, sensiblement le même : il suffit de faire la convolution sur tous les plans de l'image en même temps.

Résultats

Les résultats comparés de la stéréoscopie par corrélation sur une paire d'images de scène d'extérieur de luminance et sur la même paire en couleurs ne montrent que peu de différences tant au niveau du nombre de faux-appariements que de la précision. En effet, il existe peu de cas dans lesquels la couleur permettra de lever une ambiguïté, principalement à cause de la forte corrélation existant entre les canaux de chrominance et de luminance. La couleur est donc sans doute plus utile pour des tâches d'interprétation d'une scène que pour une tâche de bas-niveau comme la stéréoscopie par corrélation.

Par contre, dans un système utilisant de la lumière structurée ou de l'illumination active pour favoriser la mise en correspondance [KRS96], par exemple en projetant une image de texture aléatoire sur la scène, on peut faire en sorte que les trois canaux R, G, B, de la texture projetée soient tous décorrélés, et l'image en couleur apportera donc beaucoup plus d'information que la même image en luminance seulement (théoriquement, jusqu'à trois fois plus). Dans ce cas précis, donc, la stéréoscopie couleur devrait donner de bien meilleurs résultats que la stéréoscopie N&B. Des expériences sont en cours pour déterminer une « bonne » texture couleur à utiliser pour la stéréoscopie.

3.1.7 Parallélisation de la stéréoscopie par corrélation

La simplicité de la stéréoscopie par corrélation permet de réaliser très simplement sa parallélisation, voire même son implantation matérielle sur

FIG. 3.3 – *Une paire d'images en couleurs (sur support noir et blanc, seule la luminance est visible).*

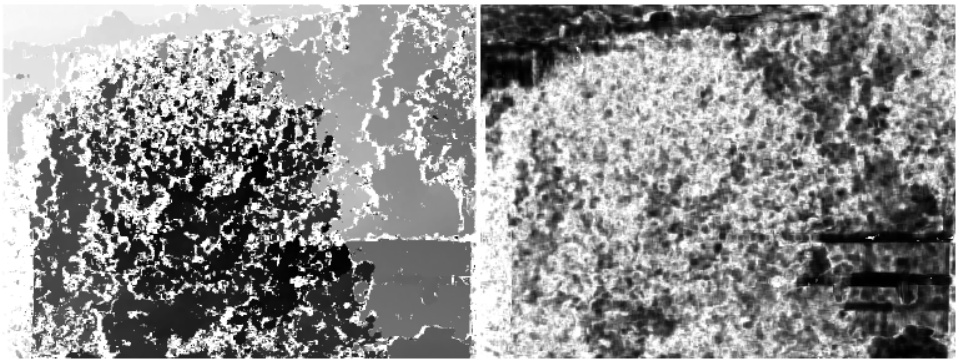


FIG. 3.4 – *la carte de disparité (g) et la carte de score (d) obtenues à partir des images de luminance seules avec le critère ZNCC (les meilleurs scores sont en noir).*

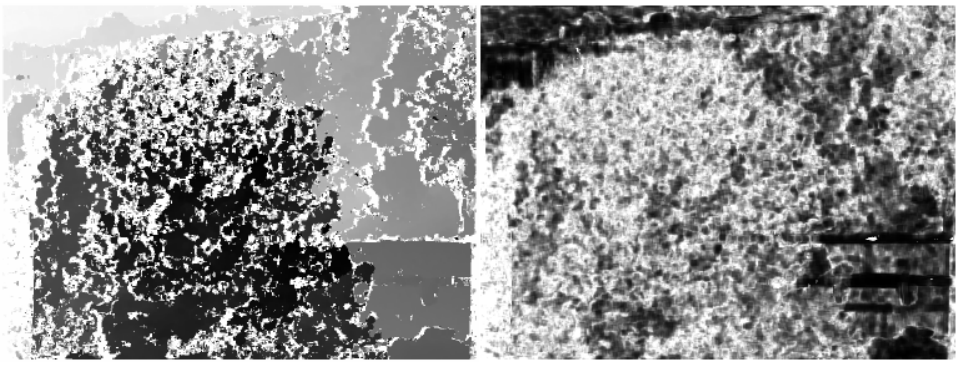


FIG. 3.5 – *la carte de disparité (g) et la carte de score (d) obtenues à partir des images en couleurs avec le critère ZNCC (les meilleurs scores sont en noir).*

une carte dédiée [Mat93, FHM+93, KR95]. Pour l'implémentation présentée ici, nous avons choisi d'utiliser un système portable, permettant d'exécuter le même code sur une machine massivement parallèle, une machine multiprocesseur, ou un groupe de machines de type stations de travail ou PCs en réseau local : PVM[5] *(Parallel Virtual Machine)*. La parallélisation de l'algorithme de corrélation a été réalisée selon un modèle maître-esclaves :

- le maître se charge de créer les esclaves, de rectifier les images originales par bandes, d'envoyer les bandes d'images rectifiées à chacun des esclaves, et de collecter les résultats ;
- chaque esclave reste en attente jusqu'à recevoir des bandes d'images à corréler, effectue la corrélation, renvoie au maître la carte de disparité, et se remet en attente.

L'algorithme s'adapte de lui-même à la vitesse de chacune des tâches esclaves : une tâche esclave tournant sur un processeur plus lent aura moins de bandes à traîter, et une tâche rapide pourra traîter plusieurs bandes pendant qu'une tâche lente n'en fait qu'une. De plus, pour être moins dépendant de l'engorgement des bus ou du réseau, tous les passages de messages sont asynchrones.

Pour mieux comprendre ce système, il est nécessaire d'examiner en détail le schéma d'exécution de chacune des tâches, maître et esclaves. Un unique programme sert à la fois pour les tâches maître et esclaves, et lors de l'exécution, il sait s'il est maître ou esclave selon la valeur du paramètre `inum` en sortie de `steCorPVMInit` (§ E.3.1) : 0 pour les esclaves et le nombre d'esclaves pour le maître.

Le seul paramètre supplémentaire que l'utilisateur doit fournir par rapport à la corrélation monoprocesseur est en combien de bandes la carte de disparité doît être divisée pour son calcul. Ce nombre est en général au moins égal au nombre d'esclaves, et peut être même beaucoup plus grand, chaque esclave pouvant traîter un nombre indéfini de bandes.

Si les bandes de la carte de disparité sont disjointes, il est cependant nécessaire qu'il y ait un recouvrement entre deux bandes d'images rectifiées consécutives, de la hauteur de la fenêtre de corrélation. Ces fractions d'images communes seront donc envoyées plusieurs fois à des esclaves différents. Pour une hauteur de bande très petite, on risque alors d'envoyer un grand nombre de fois chaque ligne des images rectifiées, et par là-même d'engorger le réseau ou le bus. Pour éviter ce problème, il suffit de prendre une hauteur de bande plus grande (au moins 2 fois) que la hauteur de fenêtre de corrélation.

[5] http://www.epm.ornl.gov/pvm/

Tâche maître

Le rôle de la tâche maître est de rectifier les images originales, par bandes horizontales, d'envoyer les bandes d'images rectifiées aux esclaves, et de récupérer les bandes de cartes de disparité qu'ils ont déjà calculées. Plutôt que de rectifier toute l'image puis de la découper, par souci d'économie de place mémoire et pour pouvoir démarrer plus tôt la corrélation, on rectifie progressivement les deux images.

Les étapes d'exécution de la tâche maître (fonction `steCorPVMMaster()`, § E.3.2) sont :

M.1 calculer les dimensions maximum des bandes d'images à envoyer aux esclaves, en divisant la hauteur des images par le nombre n de bandes souhaitées ;

M.2 envoyer ces dimensions et les autres paramètres constants (intervalle de disparité, critère de corrélation...) aux p esclaves ;

M.3 si on n'a pas encore envoyé n bandes, rectifier les images pour la bande suivante à envoyer ;

M.4 si les p esclaves sont occupés ou qu'on n'a plus de bande à envoyer, attendre une bande de disparité (un esclave au moins est alors libre) ;

M.5 si on n'a pas encore envoyé n bandes, envoyer les images rectifiées à un esclave libre ;

M.6 si on n'a pas encore reçu n bandes, aller en M.3.

Tâches esclaves

Le rôle de chaque tâche esclave est de se mettre en attente de données à corréler (des bandes d'images rectifiées), à envoyer le résultat dès qu'il est prêt, puis à se remettre en attente. Les tâches esclaves sont donc beaucoup plus simples que la tâche maître.

Les étapes de chaque tâche esclave (fonction `steCorPVMSlave()`, § E.3.3) sont :

E.1 attendre la réception des paramètres de corrélation (étape M.2 de la tâche maître) ;

E.2 attendre la réception d'images (étape M.5) ou un ordre de fin ;

E.3 si c'est un ordre de fin, terminer l'exécution de la fonction ;

E.4 calculer la bande de carte de disparité par corrélation ;

E.5 envoyer cette bande de disparité ;

E.6 aller en E.2.

Exemple d'exécution

Les figures 3.6 et 3.7 montrent respectivement un graphe de tâches et un graphe d'utilisation de la machine virtuelle PVM. Ces graphes ont été

obtenus grâce au programme de surveillance *xpvm*, qui permet de visualiser l'exécution des tâches PVM et les passages de messages.

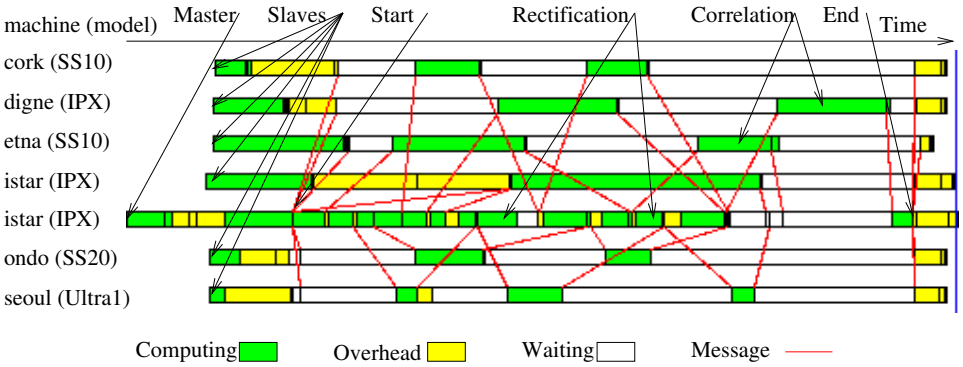


FIG. 3.6 – *Exemple d'exécution de la stéréoscopie par corrélation sous PVM : graphe des tâches commenté.*

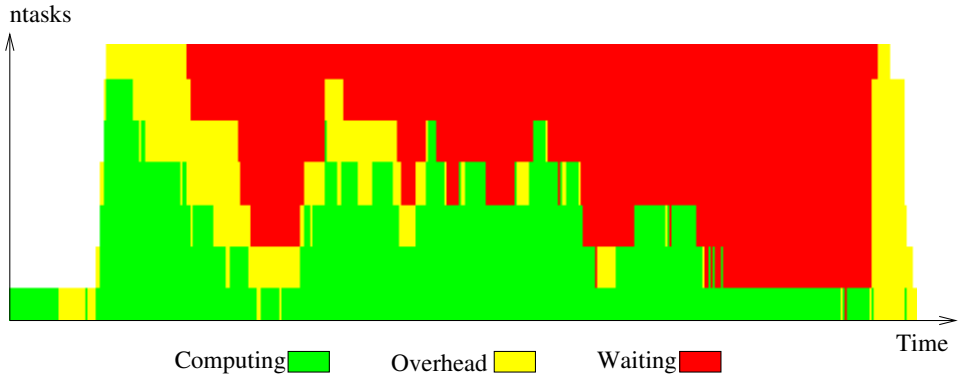


FIG. 3.7 – *Exemple d'exécution de la stéréoscopie par corrélation sous PVM : graphe d'utilisation de la machine virtuelle.*

3.1.8 En résumé

Nous avons utilisé un algorithme de corrélation somme toute classique, mis à part quelques améliorations, respectivement l'optimisation du calcul en le présentant comme une convolution, l'approximation à une précision inférieure au pixel de la disparité et la validation de la carte de disparité. Le choîx du critère de corrélation est important, et ceux que nous avons choisis (ZSSD, ZNSSD et ZNCC) sont réputés donner les meilleurs résultats. Cet algorithme a pu être adapté facilement pour utiliser également des images en couleurs ou multispectrales, et nous avons également présenté une manière simple et efficace de le paralléliser en utilisant le système PVM.

3.2 Dérivées de la disparité

De la même manière que la carte de disparité nous renseigne sur la position des points d'une surface, comme nous le verrons plus précisément au chapitre suivant, les dérivées de la disparité par rapport aux coordonnées image sont étroitement liées aux propriétés différentielles de cette surface. Le lien qui existe entre les dérivées de la disparité stéréoscopique et l'orientation du plan tangent a été clarifié par KOENDERINK et VAN DOORN [KvD76], puis plus tard par WILDES [Wil91]. Le calcul des dérivées de la disparité nous permettra donc d'obtenir après la phase de reconstruction (chapitre 4) des renseignements comme l'orientation du plan tangent ou les courbures de la surface.

Dans les recherches passées, on a presque toujours préféré traîter le problème du calcul de ces propriétés différentielles après celui de la reconstruction, c'est-à-dire qu'on commence par reconstruire les points de la surface, puis on les calcule à partir de cette reconstruction.

C'est notamment le cas des méthodes de calcul par ajustage de surface *(surface fitting)*. Les propriétés différentielles ou les dérivées de la disparité sont alors obtenues par minimisation de fonctions spline [Gri81, Ter86, BZ87], ou à partir des coefficients d'un ajustage de fonction polynomiale [EW87, HA87]. L'inconvénient principal de ces méthodes est qu'elles lissent les surfaces, et donc aussi les discontinuités (des surfaces ou de leur normale). Une solution permettant de retrouver ces discontinuités est d'examiner les résidus de la minimisation fonctionelle [HA87, EW87] : lorsqu'ils sont trop grands c'est que l'ajustement est mauvais, donc il y a une discontinuité. L'autre solution, utilisée dans les approches de type contours actifs *(snakes)*, est d'utiliser une technique de minimisation non-convexe dans laquelle les discontinuités sont autorisées à l'intérieur d'une surface lisse, moyennant une pénalité [Ter86, BZ87]. Une dernière méthode utilisant la reconstruction pour calculer ses propriétés différentielles consiste simplement à dériver point-à-point les données de cette reconstruction [BPYH85, MN84]. Nous présentons ici une méthode simple pour calculer les dérivées d'une carte de de disparité, utilisant la régression bilinéaire (§3.2.1).

Une autre manière de calculer les dérivées de la disparité est de prendre en compte en compte directement dans l'algorithme de stéréoscopie de l'orientation de la surface observée, comme dans certaines approches Bayesiennes [Bel93, SFB96]. Ces méthodes bayesiennes ont l'avantage de tenir compte de la probabilité a priori de chaque orientation possible selon le type de scène observée. STEWART *et al* ont également calculé la distribution théorique des dérivées de la disparité dans le cas où la distribution des orientations des surfaces observées est aléatoire [SFB96].

L'orientation de la surface observée peut également être prise en compte dans le cadre de méthodes de stéréoscopie *par corrélation*. Ces méthodes, apparues récemment, se dispensent de la traditionnelle contrainte fronto-

parallèle en utilisant des fenêtres de corrélation qui peuvent s'allonger et se pencher, d'où leur nom de *stretch-and-shear correlation*. Nous proposons en §3.2.2 une telle méthode, décrite aussi dans [DF94a, DF94b], et on trouvera des techniques plus ou moins similaires dans [LTS94, RH94] et dans une moindre mesure dans [JM92].

3.2.1 Dérivées à partir de la carte de disparité

Un obstacle au calcul des dérivées de la disparité de manière simple est que bien que *dense* la carte de disparité n'en comporte pas moins des trous, c'est-à-dire des endroits où aucun maximum du critère de corrélation n'a été trouvé ou bien où des points qui n'ont pas été validés. À cause de ces trous, donc, nous ne pouvons pas utiliser les magnifiques opérateurs de dérivation gaussiens dont nous disposons [Der93]. Si nous voulons utiliser la carte de disparité obtenue par corrélation il nous faudra donc utiliser des opérateurs *locaux*, qui puissent prendre en compte l'absence d'information en certains points.

Méthode de calcul

La méthode que nous avons trouvée et qui répond à ces critères consiste à effectuer une régression bilinéaire sur l'ensemble des données situées dans un voisinage $l \times h$ d'un point (u_1,v_1) de la carte de disparité, et à prendre comme valeurs pour les dérivées de la disparité les coefficients de cette régression. Plus précisément, ce que nous appelons régression bilinéaire est une approximation aux moindres carrés d'un ensemble de N points (x_i,y_i,z_i) par un plan d'équation :

$$z(x,y) = ax + by + c \tag{3.19}$$

Les calculs permettant d'obtenir les coefficients a, b, et c de cette régression ainsi que leurs incertitudes respectives sont présentés en annexe D.

Application au calcul des dérivées de la disparité

Pour le calcul des dérivées de la disparité utilisant la régression bilinéaire, nous calculons les coefficients a et b de cette régression (la valeur de c ne nous est pas utile) sur l'ensemble des mesures existant dans un voisinage de taille $l \times h$ autour d'un point (u_1,v_1), ainsi que les incertitudes σ_a et σ_b. Ces coefficients a et b correspondent aux dérivées premières du modèle de plan. Nous décidons alors de choisir ces valeurs comme dérivées de la disparité si les trois conditions suivantes sont respectées :

$$\sigma_a < s \tag{3.20}$$
$$\sigma_b < s \tag{3.21}$$
$$a > -1 \tag{3.22}$$

Le seuil s est à fixer par l'utilisateur (nous avons choisi 0,05). L'équation 3.22 correspond à la contrainte d'ordre, qui dit que si deux points A_1 et B_1 de la 1re image placés à la même ordonnée sont dans un certain ordre (par exemple A_1 est à gauche de B_1), alors leurs correspondants A_2 et B_2 dans la 2e image sont dans le même ordre. Soient par exemple $A_1 = (u_1,v_1)$ et $A_2 = (u_1 + d(u_1,v_1),v_1)$, et soit $B_1 = (u_1 + h,v_1)$. On a alors au premier ordre :

$$B_2 = (u_1 + h + d(u_1,v_1) + h\frac{\partial d}{\partial u_1} + o(h),v_1)$$

La contrainte d'ordre s'écrit alors :

$$\frac{\partial d}{\partial u_1} \geq -1$$

ce qui explique l'équation 3.22.

3.2.2 Corrélation fine

Nous savons que la carte de disparité calculée par une méthode classique de stéréoscopie par corrélation a une précision d'à peu près un pixel, malgré l'interpolation décrite §3.1.4. Si on utilise cette carte de disparité pour calculer les dérivées de la disparité, il faudra la filtrer pour en éliminer le bruit, comme nous avons fait précédemment, et ce calcul aura donc une précision limitée par celle de la disparité

Nous présentons ici une nouvelle méthode, que nous appelons *corrélation fine*. Nous avons vu précédemment une méthode de calcul des dérivées de la disparité à partir des résultats de la corrélation classique. Or ces dérivées peuvent être très bruitées, voire fausses, à cause du bruit important de la carte de disparité. L'idée essentielle de cette nouvelle méthode est d'aller chercher ces dérivées de la disparité directement dans les données d'intensité des images. En effet, on observe qu'un élément de surface qui se projette comme un carré de pixels dans l'image de référence est visible dans l'autre image comme un carré distordu, et la forme de ce carré distordu permet de calculer les dérivées de la disparité [DF94a].

Principe

Supposons que l'on ait une paire d'images en coordonnées rectifiées, et que l'on dispose de la carte dense de disparité de cette image qui aura été par exemple calculée grâce à la méthode décrite au chapitre précédent. Si

l'on dérive directement la disparité obtenue de cette façon par rapport aux coordonnées rectifiées u et v de l'image, on obtient des données contenant un bruit important, puisque la disparité n'est connue *exactement* qu'à un pixel près, malgré l'approximation subpixelique qui a été utilisée.

Pourtant les valeurs de ces dérivées doivent être intrinsèquement présentes dans l'image, puisque l'œil humain sait bien apprécier la normale à une surface à partir d'une paire d'images stéréo[6], il doit donc y avoir un moyen de les en extraire par une méthode simple. La méthode que nous présentons ici nous permet de calculer les dérivées premières de la disparité sans utiliser aucun opérateur de dérivation, directement à partir d'une paire d'images stéréoscopique.

Dans la méthode dite classique, pour calculer la disparité entre les deux images, nous avons cherché le maximum de corrélation entre une fenêtre de taille $l \times h$ dans la première image et une fenêtre de même taille sur la droite épipolaire correspondante (c'est-à-dire à la même ordonnée car on est en coordonnées rectifiées) dans la seconde image. Il y a donc un seul paramètre pour la corrélation qui est l'abscisse de la fenêtre correspondante dans la seconde image. Ici, pour calculer les dérivées premières de la disparité, la position du centre de la fenêtre correspond à la disparité telle que nous l'avons vue, et nous faisons varier lors de la recherche de correspondances deux paramètres supplémentaires qui correspondent aux dérivées premières de la disparité par rapport aux coordonnées image.

Appelons $d(u,v)$ la disparité au point (u,v). Au point (u,v) de la première image correspond le point $(u+d(u,v),v)$ de la seconde image, et soient $\alpha(u,v)$ et $\beta(u,v)$ les valeurs des dérivées de la disparité d par rapport à u et v respectivement. Au point $(u + \delta_u,v + \delta_v)$ de la première image correspond alors le point

$$\begin{aligned}(u + d(u,v) + \delta_u + \delta_u \frac{\partial d}{\partial u} + \delta_v \frac{\partial d}{\partial v},v + \delta_v)& \\ = (u + d(u,v) + \delta_u(1 + \alpha) + \delta_v\beta,v + \delta_v)& \quad (3.23)\end{aligned}$$

et à une fenêtre de dimensions $\Delta_u \times \Delta_v$ centrée en (u,v) dans la première image correspond donc dans la deuxième image une fenêtre *penchée*, c'est-à-dire en forme de parallélogramme, de base $(1 + \alpha)\Delta_u$, de hauteur Δ_v, et dont la pente des côtés est β (figure 3.8).

De même, on peut calculer une approximation de degré plus élevé de la transformation qui opère sur une région carrée de l'image gauche. On obtient le point correspondant à $(u + \delta_u,v + \delta_v)$ en calculant le développement de Taylor à l'ordre n de la fonction $(u,v) \rightarrow (u+d(u,v),v)$. Ainsi si l'on prolonge le raisonnement précédent (équation 3.23) en faisant un développement à l'ordre 2 de la fonction qui à un point de la première image associe sont

6. L'équipe de I. Könderink et A.J. van Doorn à l'Université d'Utrecht travaille sur le sujet de l'estimation des normales et courbures d'une surface par l'oeil humain.

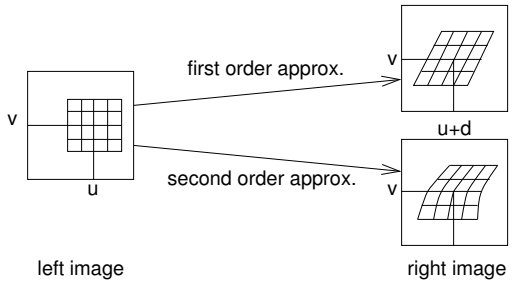


FIG. 3.8 – *Comment une petite région rectangulaire de l'image gauche se transforme dans l'image droite, en utilisant des approximations au premier ordre (en haut) et au second ordre (en bas) de la surface observée et donc de la déformation.*

correspondant, on obtient qu'au point $(u+\delta_u,v+\delta_v)$ de la première image correspond le point :

$$(u + d(u,v) + \delta_u(1 + \frac{\partial d}{\partial u}) + \delta_v \frac{\partial d}{\partial v} + \frac{\delta_u^2}{2}\frac{\partial^2 d}{\partial u^2} + \frac{\delta_u \delta_v}{2}\frac{\partial^2 d}{\partial u \partial v} + \frac{\delta_v^2}{2}\frac{\partial^2 d}{\partial v^2}, v + \delta_v) \quad (3.24)$$

Maintenant que nous savons comment un petit élément de surface de l'image gauche se déforme dans l'image droite étant donné les dérivées de la disparité, nous pouvons inversement essayer de deviner les dérivées de la disparité comme étant les paramètres d'un élément de l'image gauche déformé qui maximisent la corrélation entre les deux région des images. Par exemple, pour calculer les dérivées premières de la disparité (figure 3.8), nous devons simplement calculer les valeurs de d, α, et β qui maximisent la corrélation entre la région carrée de l'image gauche et la région étirée et penchée de l'image droite.

Problèmes pratiques et solutions

Le premier problème qui apparaît alors immédiatement est de calculer une corrélation entre deux fenêtres de tailles et de formes différentes. Ce problème n'existait pas dans la méthode classique puisque les deux fenêtres avaient des formes identiques. Nous avons envisagé les deux solutions suivantes :

1° calculer la corrélation par une intégrale surfacique, en considérant qu'une image est une fonction de deux variables constante par morceaux (c'est-à-dire constante sur la surface de chaque pixel), comme montré figure 3.9 ;

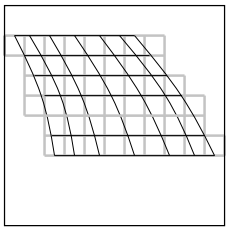


FIG. 3.9 – *Doit-on utiliser une intégrale surfacique ou interpoler les valeurs de l'intensité puis les sommer, afin de calculer le critère de corrélation ?*

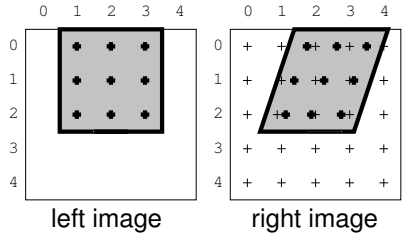


FIG. 3.10 – *Calcul du critère de corrélation*

2° calculer l'intensité des points de la fenêtre B correspondant à des points aux coordonnées entières de la fenêtre A par interpolation linéaire entre les valeurs de l'intensité des points de la 2e image et calculer ainsi la corrélation entre deux fenêtres de tailles égales comme dans les techniques classiques de corrélation.

Ayant rencontré des difficultés lors de l'implantation de la première méthode, nous avons donc choisi la seconde, qui est de loin la plus simple et la plus rapide. De plus, elle est facilement adaptable à toute classe de déformations, ce qui permettra de calculer des dérivées de degré plus élevé avec seulement quelques modifications du code.

Le meilleur moyen pour le comprendre est de prendre un exemple. Nous avons deux images 5×5 et nous voulons calculer le critère de corrélation pour une fenêtre de corrélation 3×3 située en $(u_0,v_0) = (2,1)$ dans l'image gauche, pour une disparité de $d = 0.25$ pixels, une dérivée selon x de la disparité de $\alpha = -0.125$, et une dérivée selon y de $\beta = -0.375$ (figure 3.10). Soient $L(u,v)$ et $R(u,v)$ les valeurs des intensités des pixels des images gauche et droite. La déformation locale de l'image gauche est :

$$(u,v) \longrightarrow \phi(u,v) = (u + d + \alpha(u - u_0) + \beta(v - v_0),v)$$

Nous devons juste calculer les valeurs d'intensité dans l'image droite $R(\phi(u,v))$ pour $u \in \{1,2,3\}$ et $v \in \{0,1,2\}$ par interpolation linéaire entre les deux voisins les plus proches, et ensuite calculer le critère de corrélation comme une

somme finie :

$$C(u_0,v_0,d,\alpha,\beta) = \sum_{u=1}^{3}\sum_{v=0}^{2} \left(R(\phi(u,v)) - L(u,v)\right)^2 \tag{3.25}$$

Le second problème important de l'implantation de cette méthode est de savoir comment faire pour trouver les valeurs de la disparité et de ses dérivées en un point qui maximisent la corrélation (c'est-à-dire qui minimisent le critère de corrélation, dans notre cas). En effet, l'ensemble de variation de ces valeurs n'est plus discret comme dans la méthode classique mais continu.

La solution que nous proposons consiste à estimer tout d'abord la disparité et ses dérivées par une méthode directe utilisant la carte de disparité (§3.2.1). Les paramètres ainsi estimés sont alors optimisés par une méthode de minimisation classique, de manière à obtenir ceux qui maximisent la corrélation. Dans le cas d'un critère de type SSD, on peut utiliser une méthode de moindres carrés non-linéaires, et pour un critère de type CC une méthode de minimisation de fonction objectives (i.e. à valeurs réelles), puisque le critère CC ne s'exprime comme une somme de carrés. On répète ce processus en chaque point de la carte de disparité, pour obtenir enfin une carte de disparité et des cartes de dérivées optimisées (deux pour les dérivées premières et trois pour les dérivées secondes).

Cette méthode peut paraître sous certains côtés extrêmement simple, puisque, grâce à une initialisation par les résultats de la corrélation classique, nous n'avons pas à nous soucier de l'intervalle de variation de la disparité et de ses dérivées [SFB96], mais elle donne en général d'excellents résultats comme nous verrons plus loin.

Implantation

Le code source de notre implantation de cette méthode est inclus en annexe E.2. Nous avons utilisé des routines de moindres carrés non linéaires provenant de diverses bibliothèques mathématiques avec un égal succès : SLATEC, MINPACK (tous deux disponibles sur Netlib[7]), et NaG. Le temps de calcul est évidemment beaucoup plus long qu'avec la méthode classique, mais si on réduit beaucoup le nombre de points 3-D (par exemple par décimation) avant « l'affinage », ce dernier pourra être effectué à une fréquence raisonnable.

3.2.3 Comparaison des deux méthodes

Avant même de voir les résultats on peut prédire l'effet que vont avoir chacun des opérateurs de dérivation et comparer leurs qualités et défauts respectifs.

[7] http://www.netlib.org/

Le gros inconvénient de la dérivation par régression bilinéaire est que cet opérateur effectue en fait un lissage sur les données (puisque c'est une approximation aux moindres carrés), et l'arrête d'un dièdre se verra lissée par cet opérateur, alors que la corrélation « accrochera » un plan ou l'autre du dièdre sans lisser l'angle.

La dérivation par corrélation comprend une partie qui peut s'avérer délicate et qui est le choix de l'intervalle dans lequel les dérivées peuvent varier lors de la recherche du maximum de corrélation. De son côte la dérivation par régression comporte un seul seuil à fixer, et son niveau se fixe assez facilement.

3.2.4 Parallélisation

La parallélisation de ces deux méthodes peut s'avérer nécessaire, surtout de la dérivation par corrélation qui est assez coûteuse en temps de calcul. Cette parallélisation peut s'effectuer exactement de la même manière que pour la stéréoscopie par corrélation « classique » (§3.1.7) : des bandes d'images rectifiées sont distribuées par une tâche maître à des tâches esclaves, qui se chargent de les traîter et de renvoyer le résultat à la tâche maître.

3.3 Corrélation sans rectification

La stéréoscopie par corrélation peut s'effectuer sans rectification de la paire d'image. Cette corrélation sans rectification consiste à construire une famille d'homographies à un paramètre (ce paramètre étant l'équivalent de la disparité dans la corrélation classique), $\{\mathbf{H}(d), d \in [d_1,d_2] \subset \Re\}$ qui respectent la contrainte épipolaire. Pour celà, il est nécessaire et suffisant qu'elles envoient l'épipôle e sur e' et chaque droite de la première image passant par e sur la droite épipolaire correspondante par la matrice fondamentale $\mathbf{F}$ (§1.2.2).

Pour chaque valeur[8] de d, on calcule la transformée de la seconde image par l'homographie $\mathbf{H}^{-1}(d)$. Ensuite on calcule le critère de corrélation sur l'ensemble de l'image, comme dans l'algorithme de corrélation classique (§3.1.3), en convoluant la différence (critère de type SSD) ou le produit (critère de type CC) des deux images résultantes par un masque de la taille de la fenêtre de corrélation. On compare la valeur du critère en chaque point au meilleur score obtenu en ce point pour les autres valeurs de d, tout comme dans la méthode classique, et l'image formée par les valeurs de d maximisant le score est une sorte de carte de disparité.

La première différence avec la méthode classique est qu'on ne transforme qu'une des deux images, on peut donc espérer obtenir de meilleurs

8. On choisit un échantillonage discret de l'intervalle de variation $[d_1,d_2]$ de d.

résultats puisque les intensités d'une image seulement seront interpolées. Évidemment, la contrepartie est qu'on doit calculer cette transformation, qui équivaut à la rectification d'une image en charge de calculs, pour *chaque* valeur de la pseudo-disparité d. La seconde différence essentielle est que la carte de disparité obtenue sera dans le repère de l'image de référence *non rectifiée*, contrairement à la méthode classique. À moins d'un choix judicieux de la famille d'homographies et de sa paramétrisation par d, la formule de reconstruction risque donc de ne pas être aussi simple que pour la méthode classique.

3.3.1 Choix de la famille d'homographies

Le premier choix possible pour la famille d'homographies est de prendre celle qui correspond à la méthode classique par rectification, pour laquelle le paramètre d est effectivement la disparité entre les deux images rectifiées : c'est la famille d'homographies $\mathbf{H}(d)$ de la figure 2.3. C'est en général un bon choix, sauf lorsque les droites (ef) et $(e'f')$ passent dans les images. En effet, la droite (ef) est envoyée à l'infini par $\mathbf{R}$, qui est invariante quel que soit d par $\mathbf{D}(d)$, puis qui est envoyée sur $(e'f')$ par $\mathbf{R}^{-1}$. L'image de tout point de (ef) est donc le même point de $(e'f')$ quel que soit d, et on ne pourra pas mesurer de « disparité » sur les points correspondant aux droites (ef) et $(e'f')$. Ces deux droites sont la projection dans les caméras d'un plan 3-D passant par les deux centres optiques[9]. Cette paramétrisation de l'ensemble des homographies à appliquer pour la *corrélation sans rectification* est donc tout à fait adaptée quand au moins un des deux épipôles est en dehors de l'image, puisqu'on pourra facilement trouver f ou f' tels que la droite (ef) ou $(e'f')$ ne traverse pas l'image. Par contre, dans le cas où les deux épipôles se trouvent à l'intérieur des images (par exemple si le mouvement entre les deux caméras est frontal), il faut trouver une autre famille d'homographies $\{\mathbf{H}(d)\}$.

Une solution dans le cas où les deux épipôles sont proches des images ou même à l'intérieur est d'utiliser pour $\mathbf{H}(d)$ une homothétie de facteur f(d), où f est une fonction monotone de d, de centre soit e soit e', respectivement avant ou après avoir appliqué l'homographie. De cette manière les points dont l'image par $\mathbf{H}(d)$ est indépendante de d, c'est-à-dire pour lesquels la pseudo-disparité d ne pourra pas être mesurée, sont à l'infini respectivement dans la première ou la deuxième image. Évidemment cette solution n'est pas adaptée si les épipôles sont éloignés de l'image, voire à l'infini. Les deux familles d'homographies que nous avons décrites sont donc complémentaires puisqu'elles permettent de couvrir l'ensemble des situations possibles de la géométrie épipolaire.

9. C'est en fait le plan passant par les centres optiques parallèle au plan de rectification, figure 2.1 §2.1

3.3.2 Intervalle de pseudo-disparité

Une fois choisie la famille d'homographies, reste à déterminer l'intervalle $[d_1,d_2]$ de variation de la pseudo-disparité d. Comme cette pseudo-disparité n'a pas forcément une relation directe avec la distance des points à la caméra, le moyen le plus sûr de déterminer cet intervalle est d'utiliser un ensemble de points mis en correspondance par une autre méthode (par exemple lors du calibrage faible, par une méthode de type [ZDFL95]). On calcule l'intervalle de variation de d pour cet ensemble de correspondances, puis on ajoute à chaque borne une marge de sécurité, et on obtient ainsi l'intervalle $[d_1,d_2]$ à utiliser.

3.3.3 Temps d'exécution

Le temps d'exécution de la corrélation sans rectification est, comme pour la méthode classique, proportionnel à la taille de l'intervalle de disparité et à chacune des dimensions de l'image, et indépendant de la taille de la fenêtre de disparité. La différence essentielle est qu'on effectue une rectification par valeur de la disparité, donc la dépendence est plus importante en l'intervalle de disparité (c'est-à-dire que dans le calcul du temps d'exécution, le facteur de proportionalité est plus grand).

3.4 Exemples et comparaison des méthodes

Les cartes de disparité obtenues par corrélation fine sont visiblement moins bruitées que celles obtenues par la méthode classique. Tout d'abord, l'aspect en « marches d'escaliers » des résultats de la méthode classique, dû au fait que l'interpolation sous-pixélique de la valeur de la disparité donne en pratique un résultat proche de la valeur entière, disparaît. Ensuite, les endroits où les variations de la disparité sont élevées (pente et courbures) sont bien mieux traîtés par la corrélation fine, notamment dans les zones proches des contours d'occultation, comme le montrent les résultats suivants. Pour apprécier pleinement la qualité des résultats, il vaut mieux comparer les reconstructions 3-D, et nous continuerons donc la comparaison des méthodes au chapitre suivant.

Tous les tests ont été fait sur des images réelles de sujets qui peuvent se diviser en deux catégories :

1° des images d'objets très texturés, qui sont des moulages de polystirène en forme de bustes et de visages recouverts d'une peinture mouchetée qui permet d'obtenir de bons résultats de corrélation ;

2° des images de visages, pris en gros plan, pour prouver que la corrélation classique ou la corrélation fine peuvent aussi bien fonctionner sur des surfaces peu texturées.

Le système stéréoscopique a été calibré par une méthode de calibrage faible robuste [ZDFL95], qui permet d'obtenir précisément la géométrie épipolaire, et pour la phase de reconstruction, que nous verrons au chapitre suivant, nous avons utilisé une méthode de calibrage hybride 1.3.

3.4.1 Calcul de la disparité par corrélation classique

Nécessité d'une distorsion nulle

Nous nous sommes attachés au cours de ce projet de recherche à utiliser des images les meilleures possibles afin de faire fonctionner les algorithmes au maximum de leurs possibilités. Nos premiers essais de corrélation ont été faits sur des images de taille 512×512, acquises à partir de caméras CCD standard munies d'objectifs de distance focale 8mm.

Avec ce matériel, et malgré le soin apporté aux manipulations et de malgré de multiples essais, nous n'arrivions pas à obtenir de bonnes cartes de disparité. En effet lorsque nous essayions de faire fonctionner la corrélation certaines zones de l'image donnaient systématiquement des résultats médiocres sans raison apparente. Après quelques vérifications nous nous sommes rendu compte que la droite épipolaire correspondant à un point d'une de ces zones passait à plusieurs pixels de son image dans l'autre caméra, et les objets utilisés étant très texturés ceci expliquait complètement nos problèmes. La géométrie épipolaire du reste de l'image étant parfaite, nous en avons déduit que ce problème était sans doute dû à la distorsion optique liée aux optiques utilisées.

Les solutions envisagées

Une première approche à ce problème a été de tenter de corriger cette distorsion (voir §1.1.2, 1.2.4 et annexe A), mais la solution la plus simple était de traiter le problème à sa source, en changeant tout simplement les objectifs. Nous sommes ainsi passés de 6 pixels de distorsion maximale (amplitude de la déformation dans les coins de l'image) pour des objectifs de focale 8mm (le capteur est de diagonale $\frac{1}{3}''$), mesurés grâce à une technique de calibrage fort tenant compte de la distorsion, à moins d'un pixel pour des objectifs de focale 16mm, et les résultats de l'algorithme de corrélation s'en sont nettement ressentis.

Résultats

Pour tester les différentes méthodes décrites précédemment, nous proposons d'utiliser des images de deux types d'objets :

– des moulages de polystirène recouverts d'une peinture mouchetée grise et blanche, très texturée, qui permet de faciliter la corrélation et donc d'obtenir à coup sûr de bons résultats ;

– des objets plus courants, comportant peu de texture et sur lesquels la corrélation sera donc a priori plus difficile.

Nous avons pris une paire d'image de chaque type d'objets pour calculer les résultats que nous allons présenter : un moulage de buste et le visage d'un collègue. Les images ont 256 niveaux de gris pour une taille de 512×512 pixels.

Les résultats de la corrélation classique sur le buste (figure 3.11) sont déjà étonnants, en effet avec une fenêtre de corrélation de taille 7×7 on arrive déjà à de très bons résultats, mais pour une meilleure précision sur la disparité il a fallu utiliser une fenêtre de taille 9×9. La paire d'images ainsi que la carte de disparité obtenue sont présentés figure 3.11. On remarquera que la rectification a été faite de manière à ce que l'objet occupe la totalité de l'image. Pour le buste, les valeurs de la disparité sont comprises entre -12 et 14 pixels, soit un intervalle de 26 pixels.

Autant nous étions sûrs d'obtenir des résultats avec des surfaces très texturées, autant les essais sur des objets réels tels que les visages faisait partie d'un domaine expérimental, et nous ne nous attendions guère à obtenir des résultats satisfaisants sur ce type d'images. Quelle ne fut donc pas notre surprise en voyant apparaître les premiers résultats de corrélation sur une paire d'images de visage !

Ces résultats ont tout de même nécessité de bonnes conditions de prise de vue : nous avons utilisé un éclairage diffus, afin d'éviter les zones d'ombre, et nous avons utilisé la plus grande dimension du capteur CCD pour la hauteur du visage afin d'utiliser au mieux la résolution que nous offrent les caméras et le système d'acquisition. La paire d'image rectifiée et la carte de disparité calculée pour une fenêtre de corrélation de taille 11×11 sont présentés figure 3.12. Les valeurs de la disparité varient entre -55 et -30.

Nous avons également testé sur cette paire une méthode hiérarchique multi-résolutions, censée éliminer des faux-appariements supplémentaires et boucher quelques trous, sans grand succès.

3.4.2 Dérivées de la disparité

Avec le calcul des dérivées de la disparité on commence la série des résultats nouveaux en stéréoscopie par corrélation. Voici donc par ordre d'apparition les dérivées par régression bilinéaire, et les dérivées par corrélation.

Dérivation par régression bilinéaire

La taille de la fenêtre dans laquelle sont pris les points de la carte de disparité utilisés est le seul paramètre délicat à choisir de cette méthode. Nous nous sommes rendu compte qu'il est raisonnable de choisir des dimensions de l'ordre de celles utilisées pour la corrélation. Si l'on choisit des dimensions trop faibles on obtient en effet une variation importante des dérivées

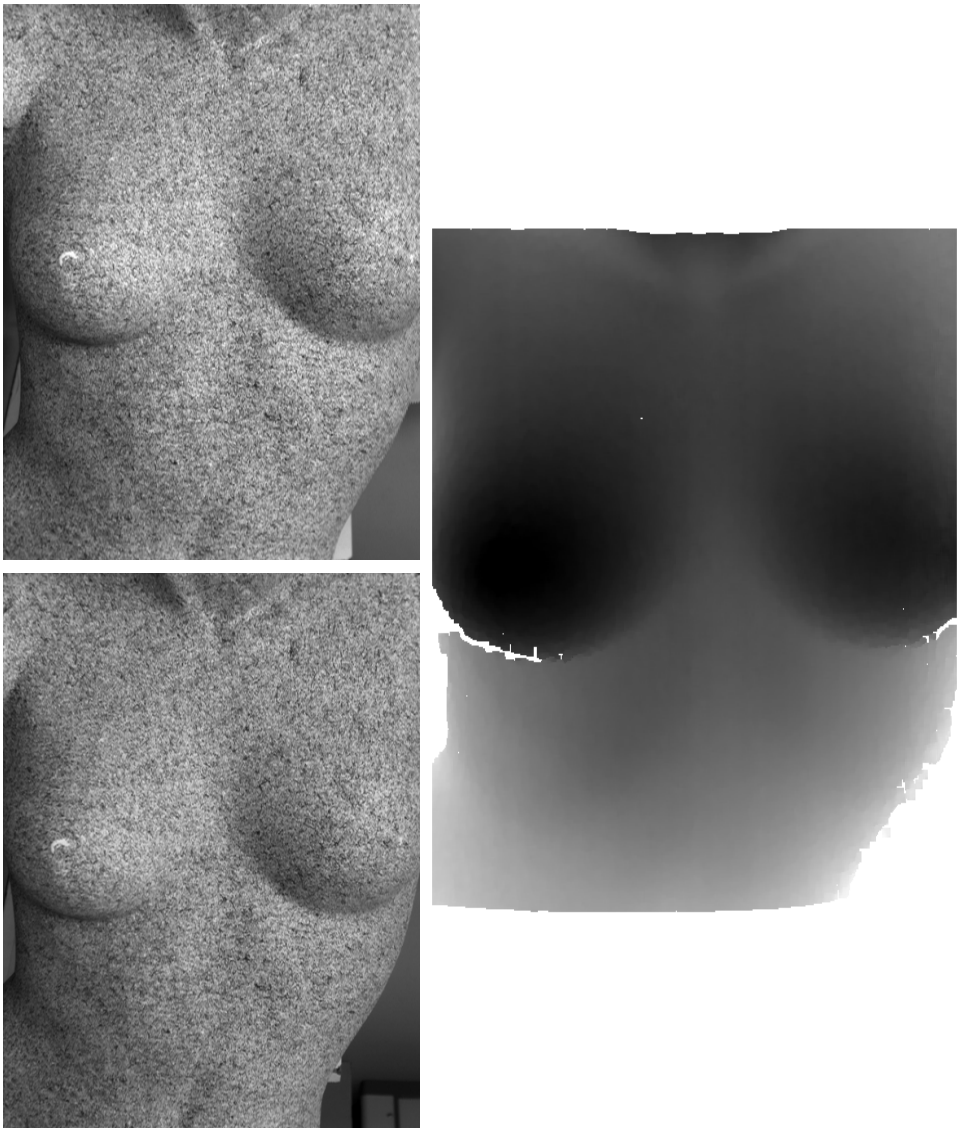


FIG. 3.11 – *Corrélation sur le buste : paire d'images et carte de disparité par la méthode classique.*

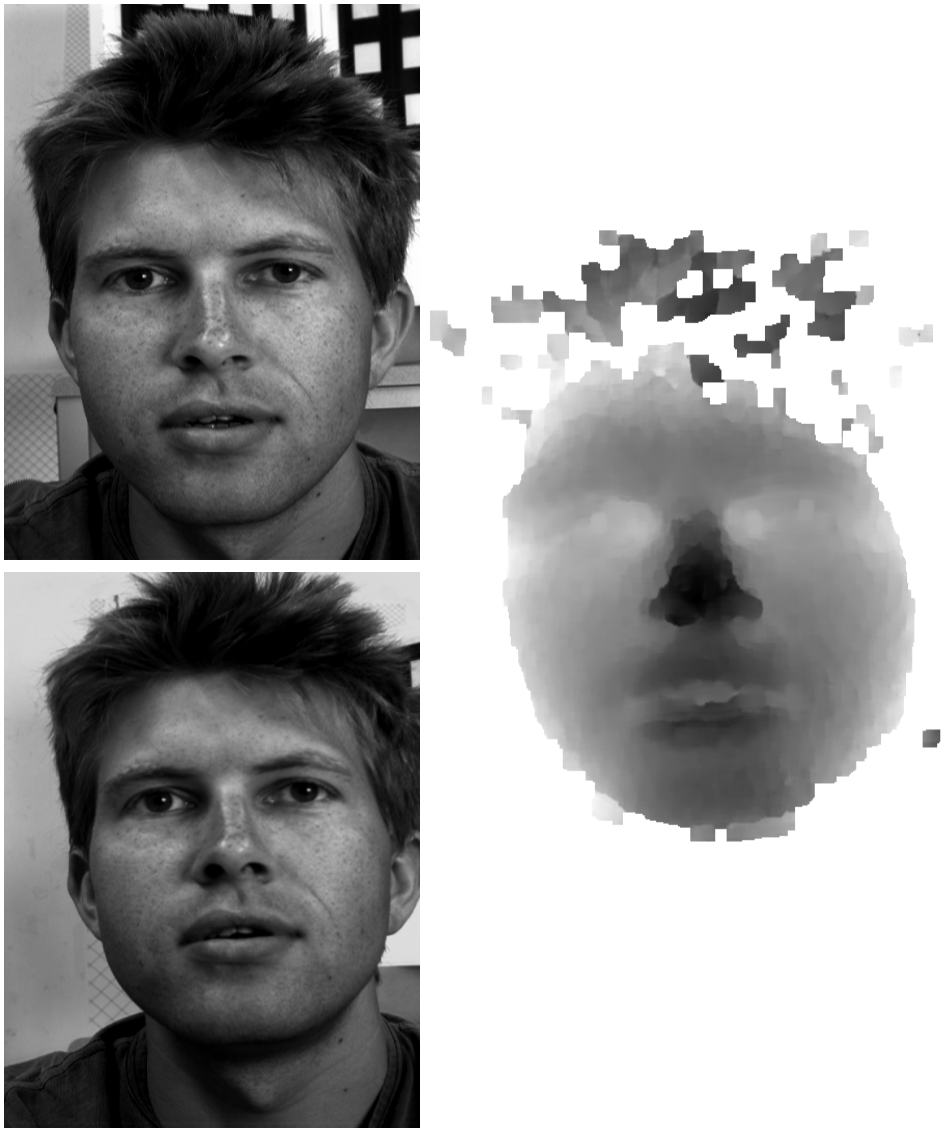


FIG. 3.12 – *Corrélation sur le visage d'Hervé Mathieu : paire d'image (stéréogramme croisé, faire pivoter le document) et carte de disparité par la méthode classique.*

premières qui se traduit par des « bosses » sur les images des dérivées. Au contraire si l'on choisit des dimensions trop importantes on commence à perdre des détails de la surface, et surtout les discontinuités de la normale à la surface commencent à se perdre.

Comme nous pouvons le voir sur les figures 3.13 et 3.15 cette méthode semble bien fonctionner sur les zones où la normale est continue, mais estompe beaucoup les détails intéressants, c'est-à-dire surtout les lieux de forte courbure et où la normale est discontinue. De plus on remarque que des « bosses » semblent apparaître un peu partout sur la surface, sans doute à cause de l'imprécision des données de base utilisées, c'est-à-dire non pas les images, mais la carte de disparité calculée par corrélation classique.

Corrélation fine avec calcul des dérivées

Les résultats de la corrélation fine avec calcul simultané des dérivées sont sans aucun doute les plus spectaculaires, au prix d'un temps de calcul largement plus élevé que pour la corrélation classique (de 10 à 30 fois plus lent). On se rend compte en regardant les résultats des figures 3.14 et 3.16 qu'on arrive à obtenir à partir d'un traitement de bas niveau lié directement aux données d'intensité des images (puisque ce n'est finalement que de la corrélation) des informations qu'on obtenait auparavant presque exclusivement à partir de données de plus haut niveau (comme le résultat d'une méthode de stéréoscopie ou une reconstruction 3-D), tels que l'estimation de la normale à une surface ou les discontinuités de cette normale.

Le résultat le plus remarquable est la nouvelle carte de disparité, qui est beaucoup plus précise à cause du fait qu'on tient compte des déformations locales de l'image dues à l'orientation ou à la courbure des surfaces. Les dérivées premières de la disparité ont l'air également précises (figures 3.14 et 3.16), en particulier comparées à celles obtenues par interpolation bilinéaire (figures 3.13 et 3.15).

En observant les images des dérivées secondes de la disparité calculées par corrélation fine (figure 3.17), on apperçoit un étrange phénomène : en effet, il semble y avoir des bandes horizontales sur toute l'image de la dérivée seconde selon y, et puisque leur amplitude et leur fréquence décroit lorsque la taille de la fenêtre de corrélation diminue, nous en avons déduit qu'elles devaient provenir d'une sorte de bruit. Puisque ceci n'apparaît que dans cette image particulière, nous avons deviné que ce devait être la conséquence de l'erreur de synchronisation au début de chaque ligne vidéo des caméras analogiques (appelée *pixel jitter*). Après vérification, l'amplitude de ces « vagues » correspond en effet au bruit de synchronisation donné par les documents techniques du système d'acquisition (caméra et carte d'acquisition), qui peut aller jusqu'à 0.5 pixel. De plus, après la phase de reconstruction, nous avons pu observer en 3-D l'effet de tôle ondulée du buste, dû à ce défaut des caméras (il n'était pas discernable sur les cartes de disparité,

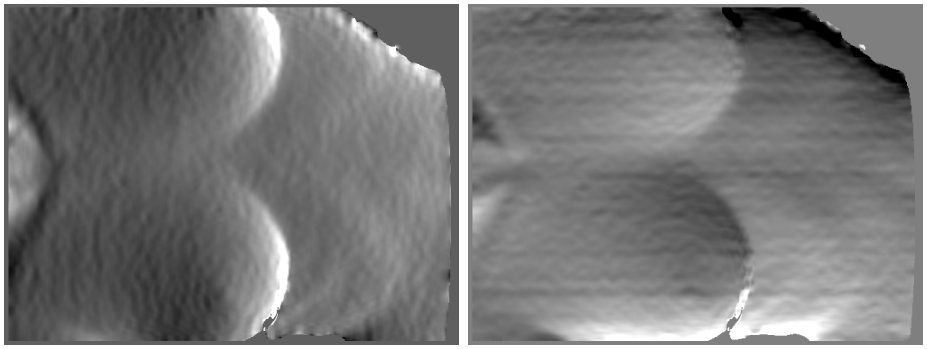


FIG. 3.13 – *Dérivée selon l'horizontale et la verticale calculées par régression bilinéaire sur le buste, avec une fenêtre de taille* 11×11.

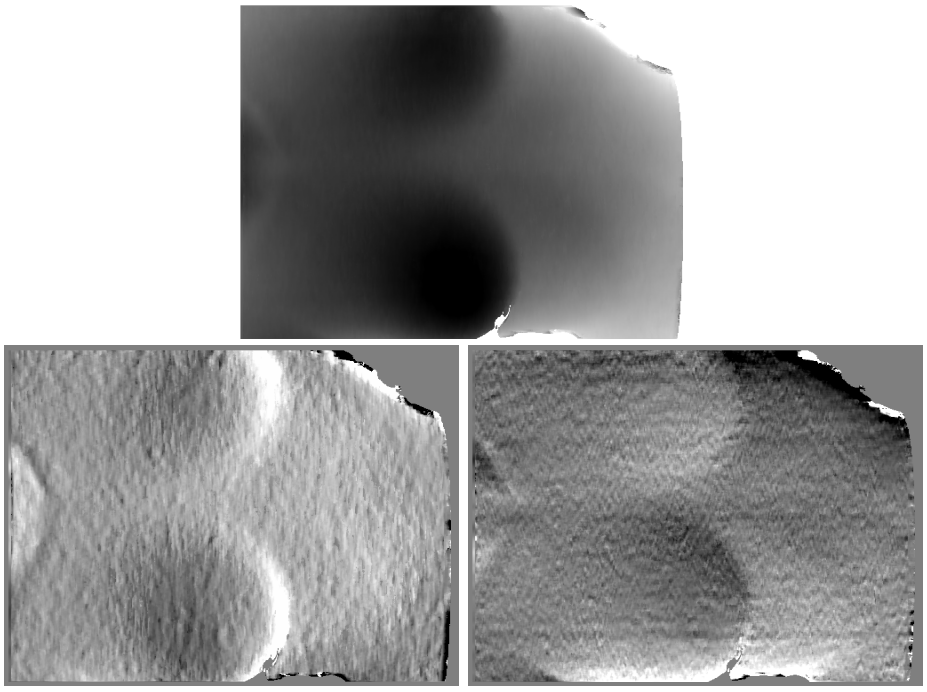


FIG. 3.14 – *Carte de disparité et dérivées selon l'horizontale et la verticale calculées par corrélation fine sur le buste, avec une fenêtre de taille* 15×15.

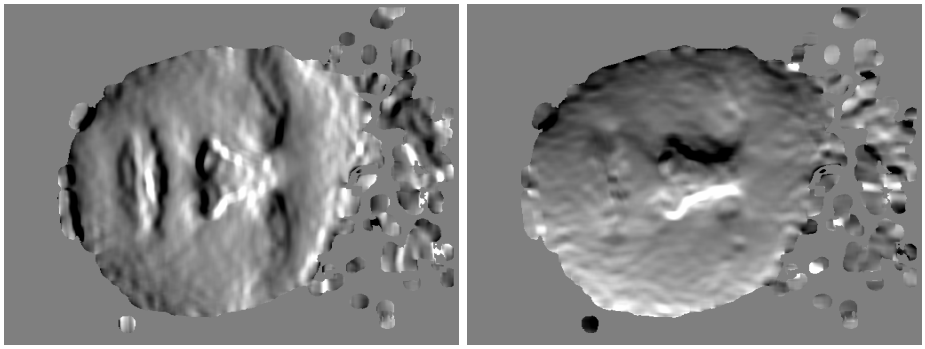


FIG. 3.15 – *Dérivées selon l'horizontale et la verticale calculées par régression bilinéaire sur Hervé, avec une fenêtre de taille* 11×11.

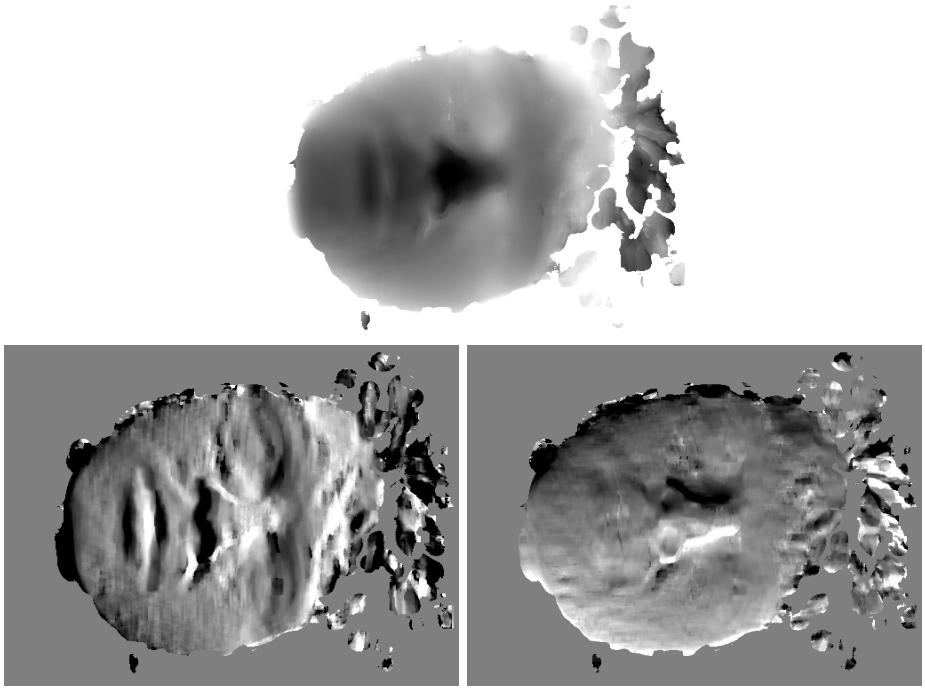


FIG. 3.16 – *Carte de disparité et dérivées selon l'horizontale et la verticale calculées par corrélation fine sur Hervé, avec une fenêtre de taille* 17×21.

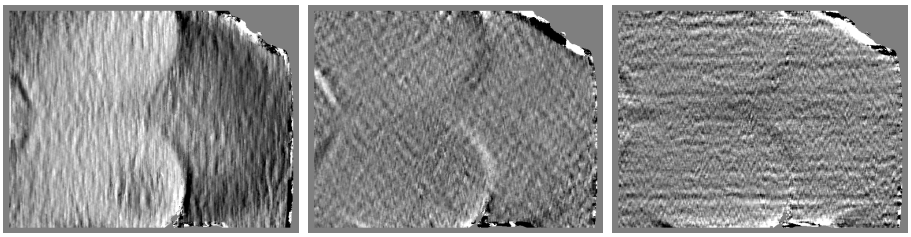


FIG. 3.17 – *Les dérivées secondes de la disparité :* $\frac{\partial f}{\partial x^2}$ *(gauche),* $\frac{\partial f}{\partial x \partial y}$ *(centre), et* $\frac{\partial f}{\partial y^2}$ *(droite), calculées par corrélation fine avec une fenêtre de taille* 21×25*.*

mais on a pu l'observer sur les reconstructions 3-D).

3.4.3 Étude comparative sur un détail

Nous avons extrait d'une des paires d'images utilisées pour les tests (figure 3.12) un détail de petite taille (128×128 pixels, figure 3.18) qui comporte des difficultés classiques pour les méthodes de stéréoscopie par corrélation :

- des réflexions spéculaires (au centre de l'image) ;
- de fortes pentes (en haut et en bas) ;
- des discontinuités de profondeur (à gauche).

Nous présentons pour cette imagette tous les résultats des méthodes présentées dans ce chapitre. Sur la figure 3.18, on peut voir :

- l'imagette rectifiée de référence ;
- la carte de disparité obtenue par la méthode classique, avec une fenêtre de taille 11×11, dans laquelle les trous ont été bouchés par morphologie mathématique;
- la carte de disparité lissée par la méthode de calcul de dérivées par régression bilinéaire (dont l'effet sur la carte de disparité est celui d'un filtre moyen), avec une fenêtre de taille 11×11, qui n'apporte pas plus d'information mais permet d'initialiser la corrélation fine ;
- la carte de disparité obtenue par corrélation fine, en n'optimisant que la disparité et en utilisant pour les dérivées de la disparité celles obtenues par régression bilinéaire, avec une fenêtre de taille[10] 17×21 ;
- la carte de disparité obtenue par corrélation fine, en optimisant la disparité et ses dérivées premières, avec une fenêtre de taille 17×21 ;

10. Les dimensions de la fenêtre de corrélation sont volontairement différentes et ont été choisies en fonction du facteur d'aspect des images, qui sont de taille 512×512, de manière à respecter les proportions des détails du visage.

– la carte de disparité obtenue par corrélation fine avec un modèle de surface du second ordre, en optimisant la disparité et ses dérivées premières et secondes, avec une fenêtre de taille 21×25.

On peut déjà faire quelques remarques sur les cartes de disparité obtenues par corrélation fine. Notons tout d'abord qu'elles sont de qualité largement supérieure à celle obtenue par la méthode classique. Le résultat de la régression bilinéaire n'est qu'une version lissée de cette dernière. On remarque également que si on utilise la méthode de corrélation fine en prenant les dérivées obtenues par régression bilinéaire et n'optimisant que la disparité, le résultat n'est pas satisfaisant aux endroits de forte courbure. Les deux autres cartes de disparité obtenues par corrélation fine sont de très bonne qualité, la différence entre les deux se situe aux endroits de très forte courbure, où le modèle d'ordre 2 donne de meilleurs résultats (il faut également tenir compte du fait que pour le modèle d'ordre 2 on a utilisé une fenêtre de corrélation plus grande, nécessaire à cause du nombre de degrés de liberté du système).

La figure 3.19 montre les dérivées premières de la disparité obtenues par différentes méthodes. Le résultat de la régression bilinéaire montre que les défauts de la carte de disparité originale se retrouvent dans ces images de dérivées, mais ces dernières sont quand même de qualité suffisante pour initialiser la corrélation fine. Les dérivéees calculées par corrélation fine sont très bonnes, et à l'ordre 1 on distingue même nettement les limites des narines dans la dérivée selon x. Les dérivées obtenues avec le modèle d'ordre 2 sont de qualité comparable, bien que légèrement lissées à cause sans doute de la grande taille de la fenêtre de corrélation. Elles comportent également plus de défauts, que l'on retrouve également dans les images de dérivées secondes.

Ces images de dérivées secondes obtenues par corrélation fine avec le modèle d'ordre 2 semblent très bruitées, mais sont qualitativement satisfaisantes : les zones de forte courbure ont bien été détectées et sont localisées au bon endroit. La corrélation fine avec un modèle d'ordre 2 est malgré tout beaucoup moins stable qu'avec un modèle d'ordre 1, et son utilisation est pratiquement déconseillée dans les cas qui risquent de poser problème (nombreuses réflections spéculaires ou manque de texture locale).

3.4.4 Conclusion et perspectives

Nous avons présenté dans ce chapitre différentes méthodes de stéréoscopie par corrélation : d'abord une méthode classique optimisée, puis une méthode améliorée, appelée corrélation fine. La méthode classique a pu être dérivée avec succès en une version qui fonctionne avec des images couleur, et en une version qui fonctionne en parallèle sur une architecture multiprocesseurs. La corrélation fine permet de tenir compte du fait que la surface observée n'est pas fronto-parallèle, et son résultat est à la fois une carte

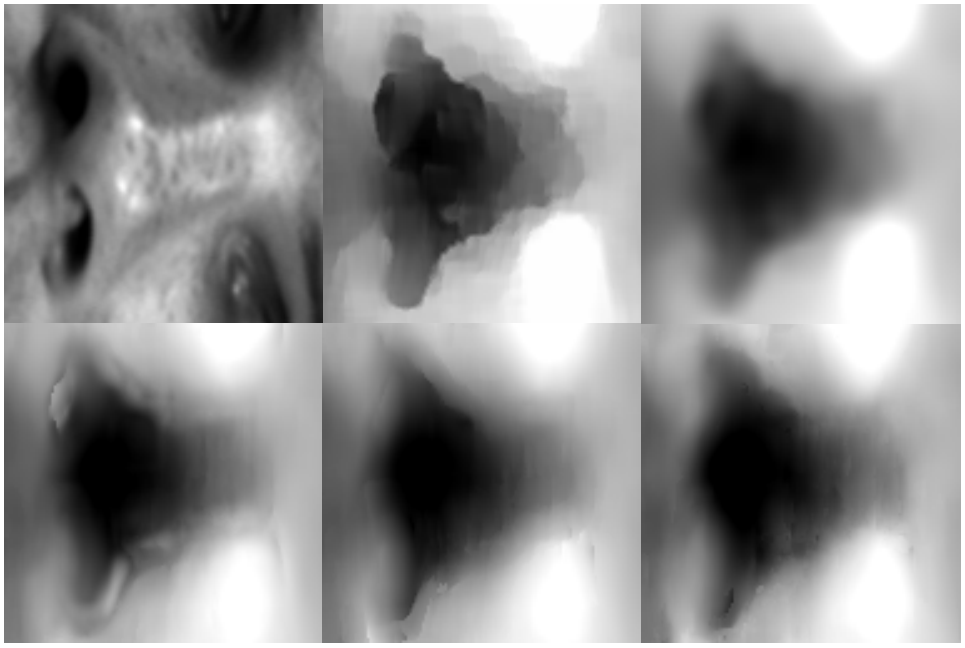


FIG. 3.18 – *Étude d'un détail. De gauche à droite et de haut en bas : imagette de référence, disparité par corrélation classique, lissage par régression bilinéaire, corrélation fine avec optimisation de la disparité seule, corrélation fine avec optimisation de la disparité et des dérivées premières, corrélation fine avec modèle de surface du second ordre et optimisation de toutes les dérivées*

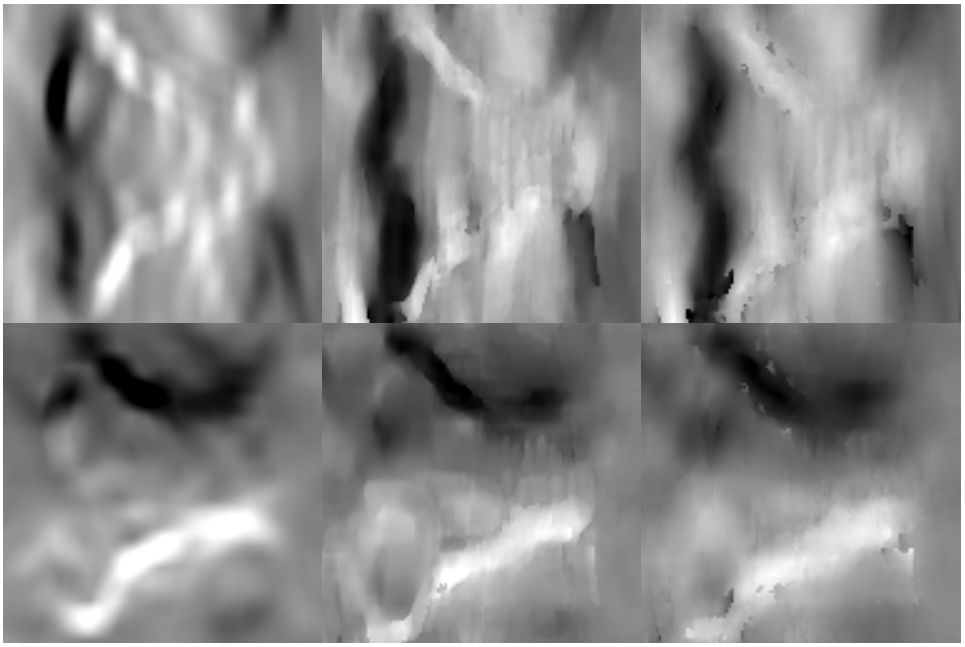


FIG. 3.19 – *Étude d'un détail. En haut les dérivées selon x de la disparité, en bas les dérivés selon y, obtenues par 3 méthodes, de gauche à droite : régression bilinéaire, corrélation fine avec optimisation des dérivées premières, optimisation fine avec modèle du second ordre et optimisation de toutes les dérivées.*

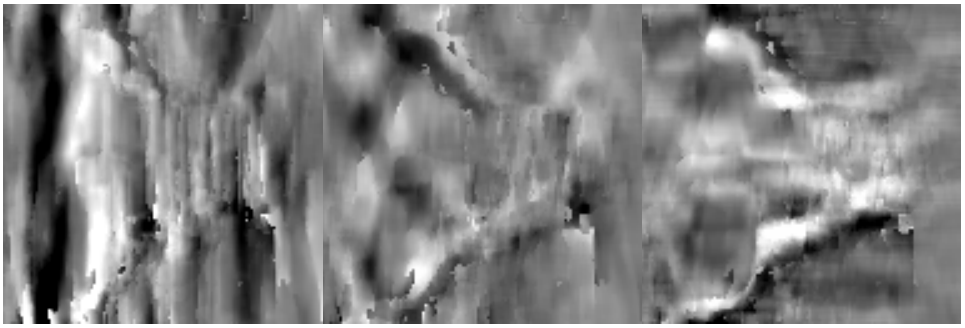


FIG. 3.20 – *Étude d'un détail. Dérivées secondes obtenues par corrélation fine avec un modèle d'ordre 2, de gauche à droite : selon x, croisée, selon y.*

de disparité et des cartes des dérivées premières et secondes de la disparité (nous verrons au chapitre suivant de quelle manière les dérivées premières et secondes de la disparité sont reliées respectivement à la normale et aux courbures de la surface 3-D observée).

Les avantages de cette nouvelle méthode sont multiples. Tout d'abord, elle permet de s'affranchir de la contrainte classique de la stéréoscopie par corrélation, dite *fronto-parallèle*, qui veut que pour avoir de bons résultats il est nécessaire que la surface observée soit presque parallèle au plan des caméras. En effet, dans cette méthode améliorée on utilise un modèle local de la surface qui peut avoir n'importe quelles orientation ou courbures. Mais l'atout le plus intéressant de cette méthode est qu'elle permet de calculer l'orientation et les courbures d'une surface directement à partir des données d'intensité, sans utiliser la carte de disparité ou une reconstruction.

La corrélation fine est plus lente (de 10 à 30 fois) que les méthodes classiques, et il est donc judicieux de ne l'utiliser que localement (par exemple pour une meilleure interprétation des régions d'intérêt, ou aux sommets d'un maillage). Nous avons également pu noter que les dérivées secondes de la disparité obtenues par cette méthode ne sont pas aussi stable qu'on pourrait l'espérer, et une bonne solution à ce problème de stabilité pourrait être une méthode hybride dans laquelle on utilise à la fois les données d'intensité et la reconstruction 3-D. On pourrait par exemple utiliser les résultats au premier ordre pour calculer les dérivées secondes par une autre technique, par exemple une approche par modèles. Ce serait équivalent à calculer l'information de courbure à partir de la reconstruction de la surface et de son champ de normales, et il est très probable qu'on obtienne ainsi de meilleurs résultats qu'en effectuant une approximation de la carte de disparité obtenue par un modèle de surface, par une technique classique.

Pour valider ces techniques, il reste à les appliquer à des objets de géométrie connue, ou plus simplement à des images de synthèse dont on connaît la carte de disparité précisément, pour connaître exacement la précision des résultats et la confiance qu'on peut leur accorder. Du point de vue technique, une amélioration de la méthode consisterait à déduire de la méthode de corrélation fine une méthode « sans rectification », de la même manière qu'on l'a fait pour la méthode classique. Ceci permettrait de s'affranchir de la double interpolation des intensités qu'implique la rectification (une interpolation pour la recification, et une interpolation pour le calcul du critère de corrélation amélioré), et donc sans doute d'obtenir une bien meilleure précision des résultats.

Chapitre 4

Reconstruction tridimensionnelle de la scène

> She's actual size, but she seems much bigger to me.
> Squares may look distant in her rear view mirror
> but they're actual size – as she drives away.
>
> *They Might Be Giants*, She's Actual Size

La phase de reconstruction tridimensionnelle consiste à calculer la géométrie de la scène observée par le système de vision, à partir des résultats des algorithmes de mise en correspondance de primitives image. Dans le cas qui nous intéresse, celui de la stéréoscopie surfacique par corrélation, nous disposons d'une carte de disparité et éventuellement de ses dérivées par rapport aux coordonnées image rectifiées, et nous devons donc pour chaque paire de points mis en correspondance entre les deux images stéréoscopiques calculer la position dans l'espace du point 3-D correspondant. De plus, puisqu'il s'agit d'une méthode de stéréoscopie sur les surfaces nous pouvons éventuellement calculer les propriétés différentielles de la surface au point reconstruit : normale (ou orientation du plan tangent), courbures, etc. Nous présentons dans ce chapitre une méthode utilisant le concept nouveau de *matrice de reconstruction*, spécialement adapté à la stéréoscopie par corrélation, qui permet d'obtenir des expressions tres simples de la position, de la normale, et des courbures en chaque point 3-D à partir de la disparité et de ses dérivées.

4.1 Différents types de reconstruction

Selon le degré de calibrage du système stéréoscopique – faible, affine ou euclidien – on pourra calculer différents types de reconstruction [Fau92] :

Reconstruction projective. Si le système est *faiblement calibré*, on doit se limiter à une reconstruction projective [Har94a, Har94b] de la surface, c'est-à-dire que la géométrie de la scène observée et reconstruite n'est connue qu'à une collinéation (ou homographie) de l'espace près. En particulier, on n'a aucune notion de distance ou de rapport de distances, et on ne sait pas où se situe le plan à l'infini (ensemble des points à l'infini, c'est-à dire des directions de l'espace) dans la scène, bien qu'on ait dans le cas de la stéréoscopie par corrélation, comme on le verra plus tard, des indications sur les lieux *où il ne peut pas être*. L'invariant projectif fondamental est le birapport. On peut également s'intéresser au calcul d'invariants différentiels projectifs de la surface en chaque point, mais dans le cas d'une reconstruction projective, hormis l'orientation du plan tangent, ils sont de degré trop élevé – il y a 6 invariants projectifs, les 4 premiers sont d'ordre 5 et les suivants d'ordre plus élevé [Car92] – pour qu'on puisse prétendre les calculer à partir des données image dont nous disposons.

Reconstruction affine. La connaissance du *plan à l'infini* suffit à obtenir une reconstruction affine de l'espace [Zel96]. L'invariant affine fondamental est le rapport de distances le long d'une droite, et on peut noter que le parallélisme est également invariant par les transformations affines. Cependant le calibrage du plan à l'infini nécessite des connaissances a priori sur le système stéréoscopique ou la scène. Par exemple, si le déplacement entre les deux caméras est une translation pure, alors l'homographie du plan à l'infini, c'est-à-dire la fonction qui fait correspondre les points à l'infini de la première et de la deuxième caméra, est l'identité du plan image, et on connaît donc la géométrie *affine* de la scène. De même, si on arrive à mettre en correspondance un nombre suffisant de points à l'infini (ou de points très éloignés), ou que l'on sait que certains segments extraits des images sont parallèles dans la scène [FLR$^+$95], on peut en déduire l'homographie du plan à l'infini pour le système stéréoscopique, et donc effectuer le calibrage affine de ce système. La connaissance d'un certain nombre de mesures affines (comme des rapports de distances) sur la scène permet également d'effectuer une reconstruction affine. Cependant, dans le cas qui nous intéresse nous n'avons pas supposé avoir ce type de connaissances (homographie du plan à l'infini ou mesures affines) à notre disposition, et nous n'avons donc pas étudié le calibrage et la reconstruction affines.

Reconstruction euclidienne. La reconstruction euclidienne peut être obtenue par auto-calibrage (§ 1.4) ou à partir d'une reconstruction affine par l'utilisation d'un certain nombre de contraintes euclidiennes supplémentaires comme des données d'angles ou de distances [FLR$^+$95, ZF95, Zel96]. Elle peut bien évidemment être aussi obtenue par calibrage fort du système de

caméras (§ 1.1). Si l'on ne dispose d'aucun étalon de distance [1] dans la scène, alors la géométrie de la scène n'est connue qu'à un facteur d'échelle global près. En effet, avec une même caméra ou un même système de caméras, voir un petit objet de près ou le même objet plus gros et de plus loin donnera exactement les mêmes images.

Nous avons ici essentiellement traité des reconstructions *projective* et *euclidienne* à partir des résultats de la stéréoscopie par corrélation, c'est-à-dire de la carte de disparité. De plus, puisque nous savons calculer les dérivées de la disparité (à partir de la carte de disparité ou directement à partir des images, voir chapitre 3), nous allons montrer qu'il est également possible de calculer en chaque point le plan tangent (pour les reconstructions euclidienne et projective) et les courbures euclidiennes (pour la reconstruction euclidienne uniquement, puisque celles-ci ne sont pas des invariants projectifs). Les invariants différentiels projectifs d'une surface sont au minimum d'ordre 5 [Car92], et il est donc inutile d'essayer de les calculer.

4.2 Position

Le calcul de la position d'un point 3-D à partir de ses images dans deux caméras ou plus se fait communément par des méthodes dites de *triangulation* [Gru85], qui consistent à calculer l'intersection dans l'espace des rayons optiques associés aux points des images. Le problème est que ces rayons optiques ne sont en général pas exactement concourants, à cause de petites erreurs sur le calibrage des caméras ou sur la détection des primitives (des points dans le cas qui nous intéresse) et leur mise en correspondance. On prend alors comme reconstruction le point 3-D le plus proche dans l'espace des rayons optiques, la notion de « point le plus proche de deux droites » étant évidemment étroitement liée à la notion de distance dans l'espace, qui varie selon les méthodes de triangulation. Dans le cas de l'utilisation d'une distance euclidienne, le point le plus proche de deux rayons optiques est celui situé sur la perpendiculaire commune de ces deux rayons, au milieu des deux points d'intersection avec ceux-ci. On obtient alors en plus de la reconstruction 3-D une mesure de l'incertitude liée aux données reconstruites [Gru85, DVF92, KKSR95].

Dans notre méthode de stéréoscopie par corrélation, nous utilisons la connaissance a priori de la géométrie épipolaire et plus particulièrement la contrainte épipolaire lors de la recherche de correspondances. Celles-ci satisfont naturellement la contrainte épipolaire et la formule de reconstruction est donc exacte, contrairement aux méthodes n'utilisant pas a priori la contrainte épipolaire dans le processus de détection et de mise en correspondance de primitives (points, segments, courbes [Rob93], etc.), ou aux

1. L'utilisation du calibrage fort implique qu'on a un étalon de distance, puisqu'on dispose des coordonnées vraies d'au moins deux points de l'espace.

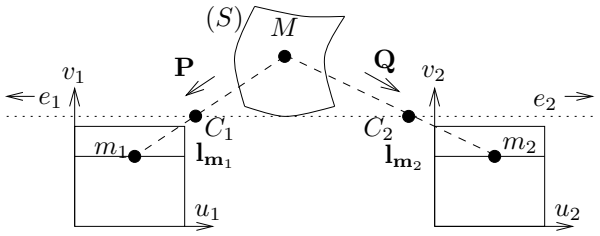


FIG. 4.1 – *En géométrie standard, c'est-à-dire après rectification, les plans rétiniens sont confondus, les distances focales sont égales, et les épipôles sont à l'infini.*

méthodes qui supposent qu'elle n'est pas exacte (stéréo par corrélation avec recherche de correspondances dans une « bande épipolaire » p.ex.). Cependant, dans le cas où les matrices de projection ont été calculées indépendamment de la géométrie épipolaire, p.ex. par la méthode du § 1.3.2, la formule de reconstruction n'est plus exacte, et on peut calculer en plus des points reconstruits une mesure d'incertitude sur chaque point, en utilisant les correspondances données par la corrélation avec les matrices de projection calculées indépendamment pour la triangulation.

4.2.1 La matrice de reconstruction

Dans la situation qui nous intéresse, le résultat de l'algorithme de stéréoscopie par corrélation est une carte de disparité, c'est-à-dire la fonction de disparité $d(x_1,y_1)$ exprimée dans le système de coordonnées rectifié de l'image de référence. Plaçons-nous d'abord dans le cas du calibrage fort. Soient les matrices de projection euclidiennes $\mathbf{P}$ et $\mathbf{Q}$ associées respectivement aux caméras *rectifiées* 1 (caméra de référence) et 2 ; les projections d'un même point de l'espace dans chacune de ces caméras ayant nécessairement la même ordonnée, les matrices de projection ne diffèrent que par la première ligne :

$$\mathbf{P} = \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix} \quad \text{et} \quad \mathbf{Q} = \begin{bmatrix} \mathbf{q}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix} \tag{4.1}$$

Soit $\mathbf{M} = (x,y,z,1)$ un point de l'espace, et $\mathbf{m}_1 = \mathbf{PM} \cong (u_1,v_1,1)$ sa projection dans l'image de référence (figure 4.1), alors sa projection dans l'autre image s'écrit :

$$\mathbf{m}_2 = \mathbf{QM} \cong \begin{pmatrix} u_2 \\ v_2 \\ 1 \end{pmatrix} = \begin{pmatrix} u_1 + d(u_1,v_1) \\ v_1 \\ 1 \end{pmatrix} \tag{4.2}$$

On a donc, pour tout point de l'espace $\mathbf{M} = (x,y,z,1)$:

$$\begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{q}_1 - \mathbf{p}_1 \\ \mathbf{p}_3 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \cong \begin{pmatrix} u_1 \\ v_1 \\ d(u_1,v_1) \\ 1 \end{pmatrix}. \tag{4.3}$$

Soit la matrice $\mathbf{R}$, appelée *matrice de reconstruction*, définie par :

$$\mathbf{R} = \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{q}_1 - \mathbf{p}_1 \\ \mathbf{p}_3 \end{bmatrix}^{-1} \tag{4.4}$$

La fonction de reconstruction euclidienne, qui à un point de la carte de disparité $\mathbf{m}_d = (u_1,v_1,d(u_1,v_1),1)$ associe le point de l'espace euclidien correspondant $\mathbf{M} = (x,y,z,1)$ est donc tout simplement la collinéation de l'espace $\mathcal{P}^3$ associée à la matrice $\mathbf{R}$:

$$\mathbf{M} \cong \mathbf{R}\mathbf{m}_d \tag{4.5}$$

Ce résultat appelle à plusieurs commentaires. On note d'abord évidemment l'extrême simplicité de la fonction de reconstruction, puisque c'est un calcul en une seule étape. Mais la conséquence la plus importante de l'équation 4.2.1 est que *la carte de disparité est une reconstruction projective*, puisqu'elle est reliée à la reconstruction euclidienne par une collinéation. Or, dans le cas du calibrage faible, on connaît la matrice fondamentale et donc la géométrie épipolaire, mais pas les matrices de projection $\mathbf{P}$ et $\mathbf{Q}$, et nous avons vu que dans cette situation on ne peut effectuer qu'une reconstruction projective. La carte de disparité en étant une, il est inutile d'aller plus loin, *la reconstruction projective ne nécessite donc aucune opération.*

La reconstruction projective particulière qu'est la carte de disparité possède en plus quelques propriétés fort intéressantes. En effet, cette reconstruction projective possède l'avantage énorme d'être *bornée dans l'espace euclidien*, en x et y par la taille de l'image, et en z par l'intervalle de disparité, ce qui n'est en général pas le cas des reconstructions projectives, qui peuvent même avoir des points à l'infini. L'autre propriété de cette reconstruction projective est que, puisque les points mis en correspondance se situent nécessairement du même côté[2] du plan focal et du plan à l'infini, *ni le plan focal, ni le plan à l'infini ne peuvent traverser l'enveloppe convexe de cette reconstruction projective.* Nous avons donc déjà, sans avoir effectué aucun calibrage affine, pu éliminer toute une partie de l'espace $\mathcal{P}^3$ que le plan à

2. La notion de « même côté » n'est pas présente dans la géométrie projective, mais existe en géométrie projective orientée [Sto91, Lav96, LF96a].

l'infini Π^∞ ne peut traverser, qui correspond donc par dualité à une partie de l'espace dual (l'espace des plans de $\mathcal{P}^3$) à laquelle Π^∞ ne peut appartenir. Évidemment cette dernière propriété n'est pas nécessairement vraie s'il y a dans la carte de disparité des faux-appariements.

4.3 Normale à la surface ou orientation du plan tangent

La direction de la normale en un point de la surface reconstruite est une propriété différentielle du premier ordre de cette surface. Puisque la reconstruction euclidienne se déduit de la reconstruction projective qu'est la carte de disparité par une homographie de l'espace, nous commençons par calculer le plan tangent en un point de la carte de disparité, puis nous en déduirons son image par la fonction de reconstruction euclidienne.

4.3.1 Plan tangent de la reconstruction projective

L'équation du plan tangent à la reconstruction projective en un point de la reconstruction projective $(x_0,y_0,d(x_0,y_0))$ s'écrit :

$$z = d(x_0,y_0) + (x - x_0)\frac{\partial d}{\partial x}(x_0,y_0) + (y - y_0)\frac{\partial d}{\partial y}(x_0,y_0) \tag{4.6}$$

Ceci peut être réécrit sous la forme :

$$(a,b,c,d)^T \cdot (x,y,z,1) = 0 \tag{4.7}$$

avec

$$a = \frac{\partial d}{\partial x}(x_0,y_0), \quad b = \frac{\partial d}{\partial y}(x_0,y_0) \tag{4.8}$$

$$c = -1, \quad d = d(x_0,y_0) - x_0 a - y_0 b \tag{4.9}$$

Les coordonnées du plan tangent en $\mathbf{m}_d = (x_0,y_0,d(x_0,y_0),1)$ à la reconstruction projective dans l'espace projectif dual s'écrivent donc $\mathbf{p}_d = (a,b,c,d)$.

4.3.2 Plan tangent à la reconstruction euclidienne

La matrice de reconstruction $\mathbf{R}$ étant de rang plein, on peut réécrire l'équation 4.7 sous la forme :

$$\mathbf{p}_d^T \mathbf{R}^{-1} \mathbf{R} \mathbf{m}_d = 0 \tag{4.10}$$

ou encore, puisque $\mathbf{M} = \mathbf{R}\mathbf{m}_d$,

$$\left(\mathbf{R}^{-T}\mathbf{p}_d\right)^T \mathbf{M} = 0 \tag{4.11}$$

Le plan tangent à la surface reconstruite en $\mathbf{M}$ se calcule donc aussi simplement que la reconstruction, puisqu'il est représenté par l'élément de l'espace projectif dual $\mathbf{p}$, avec :

$$\mathbf{p} = \mathbf{R}^{-T}\mathbf{p}_d \tag{4.12}$$

4.4 Courbures

La compréhension de la notion de courbures d'une surface de l'espace euclidien fait appel à quelques notions de géométrie différentielle. On trouvera une introduction brève mais suffisante, reprise en partie ici, en annexe de [Fau93], et pour des renseignements plus complets, on se référera à [BG88, Spi79, Bou90].

4.4.1 Rappels de géométrie différentielle

Un élément de surface (S) est défini comme une application C^2, $\mathbf{P} : (u,v) \rightarrow \mathbf{P}(u,v)$ d'un ouvert de $\Re^2$ dans $\Re^3$. Un tel élément de surface est caractérisé, à un déplacement rigide près, par deux formes quadratiques, appelées les *deux formes fondamentales*, qui sont définies en tout point de l'élément de surface.

La première forme quadratique Φ_1 définit la norme d'un vecteur dans le plan tangent T_P. Plus précisément, les deux vecteurs $\mathbf{P}_u = \frac{\partial \mathbf{P}}{\partial u}$ et $\mathbf{P}_v = \frac{\partial \mathbf{P}}{\partial v}$ sont parallèles à ce plan et définissent donc un système de coordonnées. Tout vecteur du plan tangent peut être représenté comme une combinaison linéaire $\lambda\mathbf{P}_u + \mu\mathbf{P}_v$. Sa norme au carré est donnée par la valeur de la première forme fondamentale Φ_1 :

$$\Phi_1(\lambda\mathbf{P}_u + \mu\mathbf{P}_v) = \lambda^2 E + 2\lambda\mu F + \mu^2 G \tag{4.13}$$

avec les définitions suivantes de E, F, et G :

$$E = \|\mathbf{P}_u\|^2, \quad F = \mathbf{P}_u \cdot \mathbf{P}_v, \quad G = \|\mathbf{P}_v\|^2 \tag{4.14}$$

De plus, la normale $\mathbf{N}_P$ à (S) est parallèle au produit vectoriel $\mathbf{P}_u \wedge \mathbf{P}_v$ dont la norme est la quantité $\sqrt{EG - F^2}$.

La seconde forme quadratique Φ_2 est liée à la courbure. Pour un vecteur $\mathbf{x} = \lambda\mathbf{P}_u + \mu\mathbf{P}_v$ dans le plan tangent, considérons toutes les courbes dessinées sur (S) tangentes à $\mathbf{x}$ en P. Toutes ces courbes ont la même courbure normale, le rapport $\frac{\Phi_2(\mathbf{x})}{\Phi_1(\mathbf{x})}$, avec les définitions suivantes :

$$\Phi_2(\lambda\mathbf{P}_u + \mu\mathbf{P}_v) = \lambda^2 L + 2\lambda\mu M + \mu^2 N \tag{4.15}$$

et

$$L = \frac{\partial^2 \mathbf{P}}{\partial u^2} \cdot \frac{\mathbf{N}_P}{\|\mathbf{N}_P\|}, \quad M = \frac{\partial^2 \mathbf{P}}{\partial u \partial v} \cdot \frac{\mathbf{N}_P}{\|\mathbf{N}_P\|}, \quad N = \frac{\partial^2 \mathbf{P}}{\partial v^2} \cdot \frac{\mathbf{N}_P}{\|\mathbf{N}_P\|} \tag{4.16}$$

Il est important d'étudier les invariants de Φ_2, c'est-à-dire les grandeurs qui ne dépendent pas de la paramétrisation (u,v) de (S). Φ_2 définit une application linéaire $\Psi : T_P \to T_P$ par $\Phi_2(\mathbf{x}) = \Psi(\mathbf{x}) \cdot \mathbf{x}$. Les invariants de Φ_2 sont ceux de Ψ.

Directions principales

Les *directions principales* sont les vecteurs propres de Ψ. Leurs coordonnées (λ,μ) dans le système de coordonnées $(\mathbf{P}_u,\mathbf{P}_v)$ sont les solutions de l'équation suivante :

$$(FL - EM)\lambda^2 + (GL - EN)\lambda\mu + (GM - FN)\mu^2 = 0 \tag{4.17}$$

Ce qui donne les valeurs suivantes pour λ et μ (qui sont définies à un facteur d'échelle près) :

$$\lambda = EN - GL + \epsilon\sqrt{\Delta} \tag{4.18}$$

avec $\epsilon = \pm 1$ et $\Delta = (GL - EN)^2 - 4(FL - EM)(GM - FN)$, et :

$$\mu = 2(FL - EM) \tag{4.19}$$

Courbures principales

Les *courbures principales* sont les valeurs propres de Ψ. Elles sont solutions de l'équation quadratique suivante :

$$(EG - F^2)\rho^2 - (LG + EN - 2FM)\rho + LN - M^2 = 0 \tag{4.20}$$

En particulier, leur produit K et leur demi-somme H sont les courbures gaussienne et moyenne de (S)

$$K = \frac{LN - M^2}{EG - F^2} \tag{4.21}$$

$$H = \frac{1}{2}\frac{LG + EN - 2FM}{EG - F^2} \tag{4.22}$$

Tous les autres invariants de Φ_2 sont des fonctions de ceux-ci.

Cas des surfaces implicites

Soit (S) l'ensemble des zéros de $F : \Re^3 \to \Re$, et soient a, b, c définis par l'identité

$$\begin{vmatrix} \mathbf{F}'' - \lambda \mathbf{I}_3 & \mathbf{F}' \\ \mathbf{F}'^T & 0 \end{vmatrix} = a + b\lambda + c\lambda^2, \tag{4.23}$$

où $\mathbf{F}'$ est le vecteur des dérivées premières de F, $\mathbf{F}''$ la matrice de ses dérivées secondes, et $\mathbf{I}_3$ la matrice identité. Avec cette notation, on a :

$$K = \frac{a/c}{F_x^2 + F_y^2 + F_z^2} \tag{4.24}$$

$$H = -\frac{b/c}{2\sqrt{F_x^2 + F_y^2 + F_z^2}} \tag{4.25}$$

Pour déterminer les directions principales, on prend les bissectrices des directions asymptotiques, qui sont les vecteurs $\mathbf{v}$ satisfaisant simultanément $F'(v) = F''(v,v) = 0$, où $F' \in L(\Re^3; \Re)$ et $F'' \in Bilsym(\Re^3)$ sont les formes dérivées première et seconde de F [Spi79].

4.4.2 Courbures des quadriques

Une quadrique (S) peut être définie dans $\mathcal{P}^3$ comme étant le noyau d'une forme bilinéaire symétrique $\mathbf{Q} \in Bilsym(\mathcal{P}^3)$:

$$(S) : \quad \mathbf{m}^T \mathbf{Q} \mathbf{m} = 0 \tag{4.26}$$

où $\mathbf{m} = (x,y,z,t) \in \mathcal{P}^3$.

On peut alors appliquer le formulaire des surfaces implicites. Soient $\mathbf{U}$, $\mathbf{v}$, et w définis par l'identité :

$$\mathbf{Q} = \begin{bmatrix} \mathbf{U} & \mathbf{v} \\ \mathbf{v}^T & w \end{bmatrix} \tag{4.27}$$

On vérifie simplement, en développant l'équation 4.4.2, que :

$$\mathbf{F}'' = \mathbf{U} \tag{4.28}$$

$$\mathbf{F}' = \begin{bmatrix} \mathbf{U} & \mathbf{v} \end{bmatrix} \mathbf{m} \tag{4.29}$$

L'identité 4.4.1 permettant de définir a, b, c devient alors :

$$\begin{vmatrix} \mathbf{U} - \lambda \mathbf{I}_3 & \begin{bmatrix} \mathbf{U} & \mathbf{v} \end{bmatrix} \mathbf{m} \\ \mathbf{m}^T \begin{bmatrix} \mathbf{U} & \mathbf{v} \end{bmatrix}^T & 0 \end{vmatrix} = a + b\lambda + c\lambda^2. \tag{4.30}$$

En Maple™

```
with(linalg):
U := array(symmetric,1..3,1..3):  v := vector(3):
m := vector([x,y,z]):  M := vector([x,y,z,1]):
Q := concat(stack(U,v),stack(concat(v),vector([w])));
F := concat(stack(U-lambda*array(identity,1..3,1..3),U&*m+v)
         ,stack(concat(U&*m+v),vector([0])));
```

Autre écriture : Prenons un point $m \notin \Pi^\infty$ (c'est-à-dire que $t = 1$), et soit $\mathbf{T}$ la matrice de translation par le vecteur $\mathbf{m}$:

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.31}$$

l'identité 4.4.1 permettant de définir a, b, c devient [3] :

$$\det\left(\mathbf{T}^T\mathbf{Q}\mathbf{T} - \lambda \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}\right) = a + b\lambda + c\lambda^2 \tag{4.32}$$

En Maple™

```
T := concat(array(identity,1..4,1..3),M):
evalm(transpose(T)&*Q&*T);
H := map(t->subs(evalm(transpose(M)&*Q&*M)=0,t),");
# On v{\'e}rifie que les deux {\'e}critures sont {\'e}quivalentes
evalm(F-");
```

Paramètres a,b,c. Le code Maple™ suivant permet de montrer que les trois valeurs a, b, c données par l'identité 4.4.1 ou 4.4.2 peuvent être calculées simplement à partir de $\mathbf{F}'$ et $\mathbf{F}''$. On obtient [4] :

$$a = \det\left(\mathbf{T}^T\mathbf{Q}\mathbf{T}\right) \tag{4.33}$$

$$b = -\left({\mathbf{F}'}^T\mathbf{F}''\mathbf{F}' + \operatorname{trace}\left(\mathbf{F}''\right)\|\mathbf{F}'\|^2\right) \tag{4.34}$$

$$c = -\|\mathbf{F}'\|^2 \tag{4.35}$$

3. l'élément $(4;4)$ de $\mathbf{T}^T\mathbf{Q}\mathbf{T}$ est nécessairement nul puisque $\mathbf{m} \in (S)$

4. Ces résultats sont vrais dans le cas général, pas seulement dans le cas des quadriques, et ils n'étaient pas présents dans [BG88].

En Maple™

```
Fp := vector(3):
G := concat(stack(U-lambda*array(identity,1..3,1..3),Fp)
         ,stack(concat(Fp),vector([0])));
detG := collect(det(G),lambda);
readlib(coeftayl):
a := coeftayl(detG,lambda=0,0):
b := coeftayl(detG,lambda=0,1):
c := coeftayl(detG,lambda=0,2):
simplify(ccoef + evalm(transpose(Fp)&*Fp));
# = 0 => c = -||Fp||^2
simplify(bcoef + evalm(transpose(Fp)&*U&*Fp)+trace(U)*c);
# = 0 => b = -(Fp^T U Fp +trU.||Fp||^2)
K:=concat(stack(U,Fp),stack(concat(Fp),vector([0])));
simplify(a - det(K));
# = 0 => a = det(T^T Q T)
```

Courbures : On utilise les formules 4.24 et 4.25 pour le calcul des courbures gaussienne et moyenne de la quadrique au point m. En remplaçant par les expressions précédentes pour a, b, c, elles deviennent :

$$K = -\frac{\det\left(\mathbf{T}^T\mathbf{Q}\mathbf{T}\right)}{\|\mathbf{F}'\|^4} \tag{4.36}$$

$$H = -\frac{1}{2}\left(\frac{\mathbf{F}'^T\mathbf{F}''\mathbf{F}'}{\|\mathbf{F}'\|^3} + \frac{\text{trace}\left(\mathbf{F}''\right)}{\|\mathbf{F}'\|}\right) \tag{4.37}$$

4.4.3 Courbures de la reconstruction euclidienne

Pour calculer les courbures de la reconstruction euclidienne, on va commencer par écrire l'équation d'une quadrique osculatrice de la reconstruction projective à partir de la formule de Taylor à l'ordre 2 appliquée à la fonction de disparité, puis en déduire l'équation d'une quadrique osculatrice de la reconstruction euclidienne, et enfin les courbures de cette dernière.

Quadrique osculatrice de la reconstruction projective

L'égalité de Taylor à l'ordre 2 appliquée à la fonction de disparité en (x_0,y_0) s'écrit :

$$\begin{aligned} d(x,y) = d &+ (x-x_0)p + (y-y_0)q \\ &+ \frac{(x-x_0)^2}{2}r + (x-x_0)(y-y_0)s + \frac{(y-y_0)^2}{2}t \\ &+ o((x-x_0)^2 + (y-y_0)^2) \end{aligned} \tag{4.38}$$

où on utilise les notations suivantes[5] :

$$d = d(x_0,y_0) \qquad p = \frac{\partial d}{\partial x}(x_0,y_0) \qquad q = \frac{\partial d}{\partial y}(x_0,y_0) \tag{4.39}$$

$$r = \frac{\partial^2 d}{\partial x^2}(x_0,y_0) \qquad s = \frac{\partial^2 d}{\partial x \partial y}(x_0,y_0) \qquad t = \frac{\partial^2 d}{\partial y^2}(x_0,y_0) \tag{4.40}$$

On en déduit l'équation d'une quadrique osculatrice de la carte de disparité en $\mathbf{m}_0 = (x_0,y_0,d,1)$:

$$\mathbf{m}_d^T \mathbf{Q}_d \mathbf{m}_d = 0 \tag{4.41}$$

avec

$$\mathbf{m}_d = (x,y,z,1) \tag{4.42}$$

et

$$\mathbf{Q}_d = \begin{bmatrix} r & s & 0 & p - rx_0 - sy_0 \\ s & t & 0 & q - sx_0 - ty_0 \\ 0 & 0 & 0 & -1 \\ p - rx_0 - sy_0 & q - sx_0 - ty_0 & -1 & \begin{array}{c} 2(d - px_0 - qy_0) \\ +rx_0^2 + 2sx_0y_0 + ty_0^2 \end{array} \end{bmatrix} \tag{4.43}$$

La matrice de reconstruction $\mathbf{R}$ étant de rang plein, on peut réécrire l'équation 4.41 :

$$\mathbf{m}_d^T \mathbf{R}^T \mathbf{R}^{-T} \mathbf{Q}_d \mathbf{R}^{-1} \mathbf{R} \mathbf{m}_d = 0 \tag{4.44}$$

ou encore, puisque $\mathbf{M} = \mathbf{R}\mathbf{m}_d$,

$$\mathbf{M}^T \mathbf{Q} \mathbf{M} = 0 \tag{4.45}$$

avec

$$\mathbf{Q} = \mathbf{R}^{-T} \mathbf{Q}_d \mathbf{R}^{-1} \tag{4.46}$$

L'homographie associée à $\mathbf{R}$ étant C^∞ sur son ensemble de définition, et la quadrique osculatrice définie par l'équation 4.41 étant tangente à l'ordre 2 à la carte de disparité, la quadrique définie par l'équation 4.45 est donc également osculatrice à la reconstruction euclidienne (S).

5. Ces notations classiques des dérivées partielles sont dues à MONGE

Courbures de la reconstruction euclidienne

Pour calculer les courbures en un point de la reconstruction euclidienne, on reprend les résultats du § 4.4.1 sur les quadriques, qu'on applique à la quadrique osculatrice qu'on vient de définir, au point de tangence.

4.5 Exemples

La plupart des résultats présentés ici sont disponibles sur le serveur WWW[6] de l'INRIA, à la fois sous la forme d'images et sous forme de modèles 3-D VRML.

Nous avons comparé la reconstruction 3-D obtenue à partir de cartes de disparité calculées par deux différentes méthodes : l'une par la méthode de corrélation classique (§ 3.1), et l'autre par corrélation fine (§ 3.2.2). Comme on peut le voir figure 4.2, la corrélation fine donne de bien meilleurs résultats que la méthode classique. Cette figure représente un gros plan sur une zone de 100×100 pixels du stéréogramme d'Hervé, où l'amplitude de la disparité est inférieure à 10 pixels. Nous présentons également un sous-échantillonage du champ de normales qui a été obtenu en même temps que la carte de disparité. La totalité de la reconstruction du visage et la reconstruction du buste de la figure 3.11 sont présentées figure 4.3.

Un autre exemple de champ de normales, calculé sur le buste, est montré figure 4.4.

Nous avons également obtenu des résultats très concluants avec le système stéréoscopique à base de miroirs (§ 5.1), et nous présentons ici les résultats sur deux paires d'images (figure 4.5).

La première paire d'images, prise lors du passage de Richard Szeliski à Sophia Antipolis, est très difficile, puisque la zone qui nous intéresse (le visage) n'occupe qu'une région d'environ 200×200 pixels dans chaque image. Le résultat (figure 4.6) est tout de même très satisfaisant : la reconstruction obtenue à partir de la corrélation classique est très bruitée, mais celle obtenue par corrélation fine est plutôt bonne, surtout lorsqu'on y plaque la texture de l'image originale. Les reconstructions sont présentées sous l'angle le plus défavorable, c'est-à-dire de profil alors que les images ont été prises de face. Bien des auteurs évitent d'être jugés trop sévèrement en présentant des reconstruction de 3/4...

Pour la deuxième paire d'images, nous ne disposions pas de données de calibrage des caméras et nous présentons donc ici des reconstructions projectives (dont nous pouvons voir qu'elles sont loin d'être très distordues). Ici le sujet occupe toute l'image, et tous les résultats (figure 4.7) sont donc irréprochables, sauf en quelques endroits très difficiles pour la corrélation (les cils et le blanc des yeux).

[6] http://www.inria.fr/robotvis/demo/diffprop/

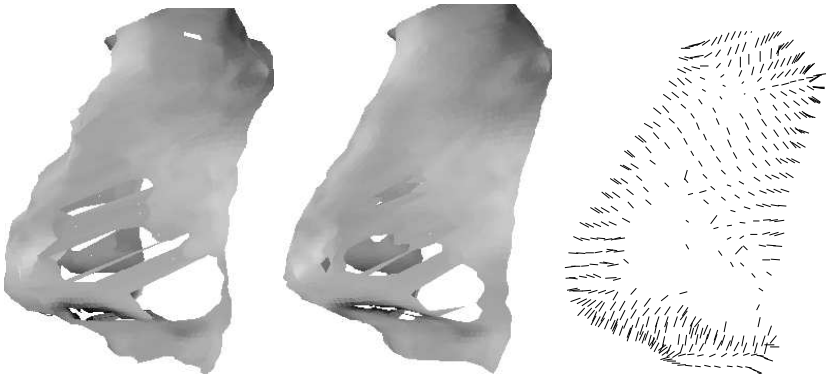


FIG. 4.2 – *La reconstruction du nez de la paire stéréoscopique de la figure 3.12 : en utilisant la méthode de corrélation classique (gauche), avec la corrélation fine (centre), et le champs de normal associé (droite).*

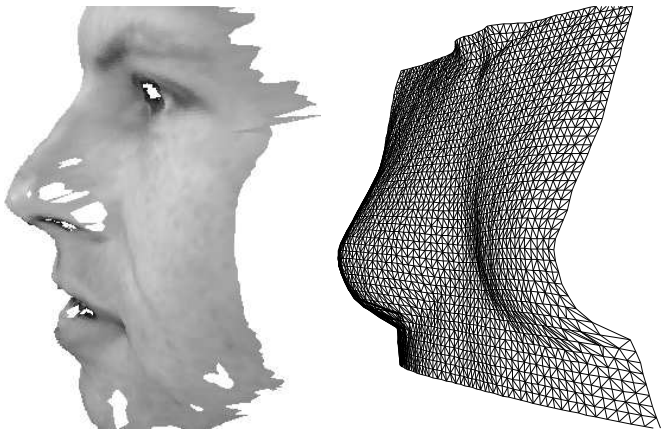


FIG. 4.3 – *La reconstruction du visage, vue de profil, avec plaquage de textures, et la reconstruction du buste, les deux ayant été calculées par corrélation fine.*

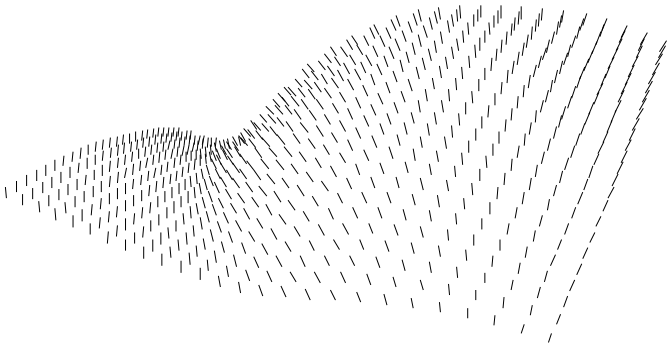


FIG. 4.4 – *Le champ de normales sous-échantillonné d'une région de* 100×100 *pixels du stéréogramme du buste. Il y a 4 pixels entre deux échantillons.*

FIG. 4.5 – *Les deux paires stéréoscopiques* 768×512 *pixels : Richard Szeliski et un quidam.*

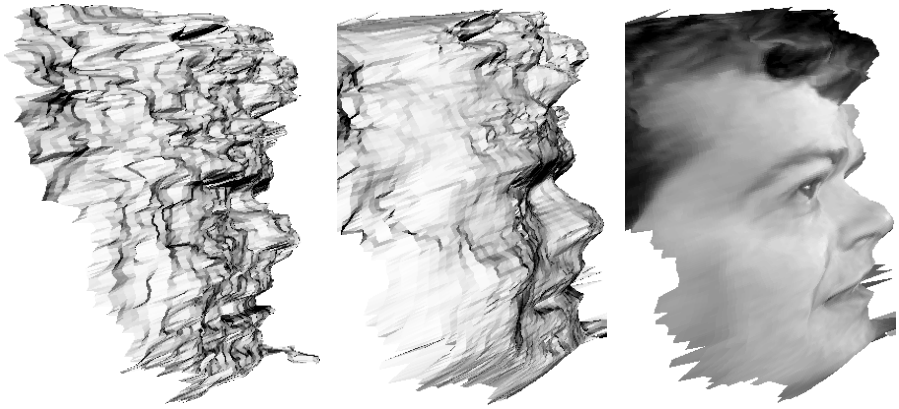


FIG. 4.6 – *La reconstruction du visage de Richard Szeliski : par la méthode classique (gauche), par corrélation fine sans texture (centre) et avec texture (droite).*

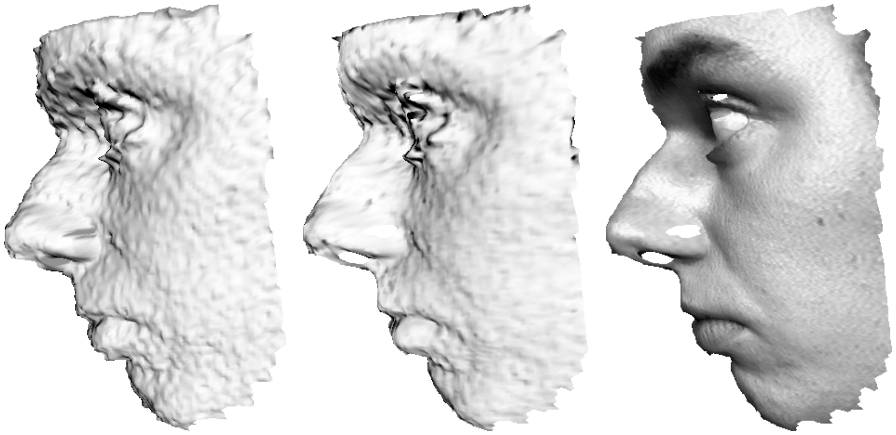


FIG. 4.7 – *La reconstruction du quidam : par la méthode classique (gauche), par corrélation fine sans texture (centre) et avec texture (droite).*

Nous ne présentons pas de résultats sur le calcul des courbures par corrélation fine pour deux raisons. Tout d'abord, il est difficile de présenter ces résultats sous une forme visuellement interprétable. Autant la reconstruction de la position ou de la normale a une signification physique simple et visible, autant la présentation de résultats de courbures est délicate. De plus, les résultats de calcul des dérivées secondes étant bruités, les courbures le seront autant et il peut être difficile de calculer des lieux géométriques dépendant des courbures, comme par exemple des lignes de crête.

Il manque évidemment à ces résultats de reconstruction un exemple où on dispose de la vérité terrain, c'est-à-dire des données exactes sur la géométrie 3-D de la scène. Nous connaissions la géométrie exacte de la cyclide utilisée dans les résultats de l'annexe B, mais nous n'avons pas eu le temps d'évaluer quantitativement ces résultats, essentiellement par manque de temps.

Chapitre 5

Applications

Le sujet des recherches présentées auparavant étant essentiellement applicatif, puisque le but premier était de construire un modèle 3-D très précis d'un objet ou d'une scène. Nous présentons donc dans ce dernier chapitre deux applications. La première est un montage qui a été réalisé au cours de cette thèse, permettant d'obtenir un système stéréoscopique à bas prix. La seconde est un système d'assistance chirurgicale préopératoire utilisant la vision stéréoscopique, idée due en grande partie à mes camarades du projet Épidaure de l'INRIA.

5.1 Vision stéréoscopique à partir d'une seule caméra

La plupart des systèmes d'acquisition vidéo actuels, notamment sur les ordinateurs personnels, ne possèdent pas plusieurs entrées vidéo simultanées. Ceci empêche tout traitement stéréoscopique de paires d'images d'objets en mouvement, et pose plus généralement de multiples problèmes pour faire de la stéréoscopie, dus essentiellement à la non-simultanéité des images. Nous avons donc réalisé [1] un système à base de miroirs pouvant s'adapter sur toute caméra ou appareil photographique et permettant de faire de la stéréovision à partir d'une seule caméra.

Nous nous limiterons essentiellement à une présentation générale du concept et aux équations minimales d'optiques nécessaires à sa compréhension. Pour le plan de fabrication mécanique, on se rapportera à [MD95]. Nous décrirons également comment l'utiliser pour les applications de stéréovision présentées dans les chapitres de cette thèse.

1. Ce montage n'aurait pas pu voir le jour sans quelques discussions riches en idées avec Christian Mercat et l'ingéniosité d'Hervé Mathieu.

5.1.1 Les raisons d'un tel système

Notre laboratoire a une longue expérience dans le domaine de la vision par ordinateur stéréoscopique, et le premier maillon de la chaîne de traitement est l'acquisition vidéo. Pour reconstruire une scène en 3D à partir d'images, il faut plusieurs points de vues, et si la scène possède des éléments dynamiques, ces images doivent être prises simultanément.

Dans le cas d'images monochromes, on peut utiliser des systèmes d'acquisition couleurs ayant une entrée RVB [2] en digitalisant chaque image sur un des canals couleur. Une autre solution est d'utiliser des caméras à lecture d'image différée comme la PULNIX TM9700, mais il faut pouvoir les commander par des sorties logiques. Pour les images couleurs, il est nécessaire d'utiliser plusieurs systèmes d'acquisition synchronisés. Cette méthode peut s'appliquer aussi à des caméras monochromes.

La plupart des systèmes d'acquisition vidéo actuels ne possèdent pas d'entrées simultanées, leurs entrées couleurs sont généralement au standard composite ou S-VHS, et quand il y a plusieurs entrées, celles-ci sont multiplexées.

Un deuxième problème se posait à nous. Dans l'algorithme de stéréovision temps réel [FHM+93], la phase de rectification permet de projeter les images sur un plan particulier. Cette opération peut être évitée si les plans images des cameras (les CCD) sont parallèles à la droite formée par les deux centres optiques des caméras. Une étude succinte nous a montré l'impossibilité de réaliser une telle configuration avec ce système de miroirs pour des raisons géométriques. Pourtant, le fait de se rapprocher de cette configuration permet au moment de la rectification de deformer beaucoup moins les images et donc d'améliorer les résultats de la stéréovision.

La solution proposée ici répond à ces deux problèmes importants de la vision stéréoscopique. Elle est, de plus, très simple à mettre en œuvre puisque que la construction du système ne demande qu'un support rigide et quelques centimètres carrés de miroir standard.

5.1.2 Le système de miroirs

Géométrie du système

Le principe de notre système de miroirs, comme montré en figure 5.1, est de simuler à partir d'une seule caméra la présence de deux caméras virtuelles, permettant ainsi de faire de la vision stéréoscopique à moindres frais.

Une configuration idéale serait d'avoir les caméras en position standard, les plan des rétines des deux caméras et la droite joignant les centres optiques étant tous trois parallèles. Mais hélas, dans notre cas, ceci impliquerait que

2. Rouge Vert Bleu

la zone de recouvrement des deux images soit nulle, ce qui présente bien peu d'intérêt pour faire de la stéréoscopie.

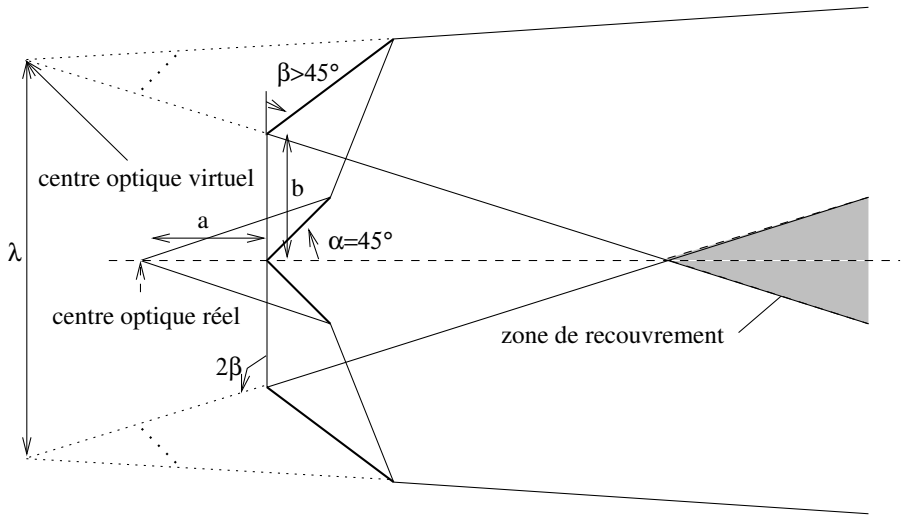


FIG. 5.1 – *Principe géométrique du système de miroirs.*

5.1.3 Calcul des dimensions des miroirs

On « déplie » le système optique en dessinant des chemins optiques rectilignes en vue de côté comme en vue de dessus (figure 5.2), ce qui nous permet de visualiser la taille des miroirs nécessaires pour le système. Dans la figure, ω_1 est le champ de vision horizontal, et ω_2 le champ de vision vertical. En général, avec les caméras CCD communes ou les appareils photo 24×36, on a $\omega_2 = \omega_1 \times \frac{3}{4}$

Un calcul trigonométrique simple nous donne les dimensions $l_1 \times h_1$ des petits miroirs et celles $l_2 \times h_2$ des grands miroirs :

$$l_1 = \frac{a\sqrt{2}}{\operatorname{cotg}\frac{\omega_1}{2} - 1}$$

$$h_1 = 2\sin\frac{\omega_1}{2}\left(a + \frac{\sqrt{2}}{2}l_1\right)$$

$$l_2 = \frac{(a+b)\sqrt{2}}{\operatorname{cotg}\frac{\omega_1}{2} - 1}$$

$$h_2 = 2\sin\frac{\omega_1}{2}\left(a + b + \frac{\sqrt{2}}{2}l_2\right)$$

La ligne de base virtuelle λ, qui est la distance entre les centres optiques virtuels (figure 5.1) est supérieure à $2b$:

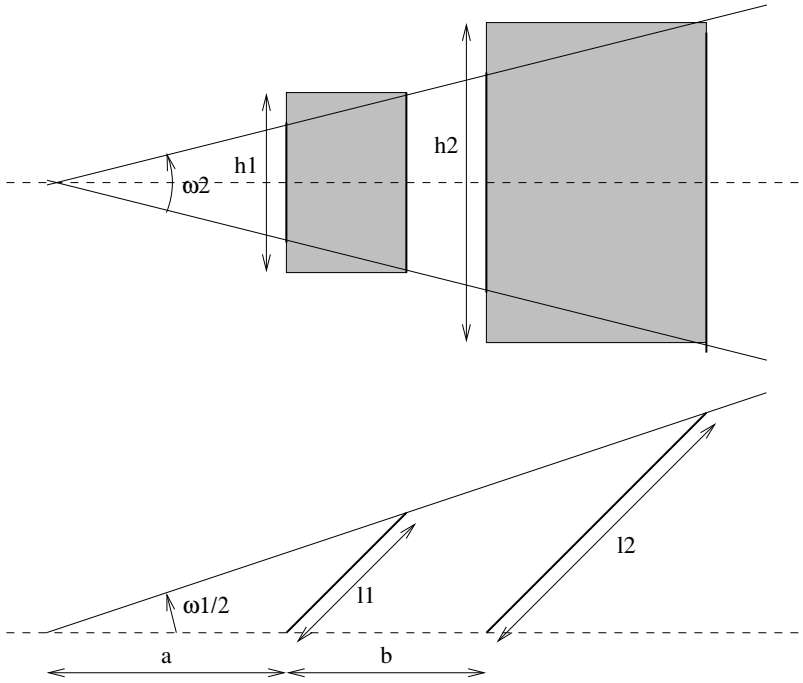


FIG. 5.2 – *Système « déplié » et calcul de la dimension des miroirs.*

$$\lambda = 2b - 2(a+b)\cos(2\beta)$$

Nous avons voulu réaliser un système qui puisse fonctionner avec deux objectifs de distances focales différentes, donc avec deux valeurs différantes de ω_1. La taille des grands miroirs ne pouvant être changée facilement, nous avons donc utilisé deux paires de trous différents pour la fixation de ceux-ci, correspondant à deux valeurs différentes de b et donc de la ligne de base virtuelle (voir le plan de montage [MD95]).

5.1.4 Flou entre les deux images

Le flou présent entre les deux demi-images est dû au flou optique généré par l'arrête du dièdre formé par les deux petits miroirs, qui est à une distance a assez faible du centre optique. Le contenu des demi-images de droite et de gauche lui-même n'est pas flou, mais les images sont plutôt « melangées » sur une bande verticale située au milieu de l'image. La largeur de cette bande est égale à la largeur de la tache de flou optique générée par l'arrête dont le calcul est possible [Pér91] mais n'est pas reporté ici.

Pour diminuer ce flou, plusieurs solutions sont disponibles :

– utiliser une distance a plus grande ;

– augmenter la profondeur de champ en fermant le diaphragme de la caméra ;
– augmenter la profondeur de champ en utilisant un objectif de plus petite focale (il faudra alors utiliser de plus grands miroirs, ou diminuer la distance b).

5.1.5 Plan de montage

Du point de vue optique et mécanique, des systèmes équivalents ont été décrits, par exemple dans [IHI93]. L'idée n'est donc pas nouvelle, notre travail a consisté à réaliser un prototype (figure 5.3) et à valider cette approche pour un algorithme qui nécessite des données de bonne qualité, la stéréoscopie dense par corrélation.

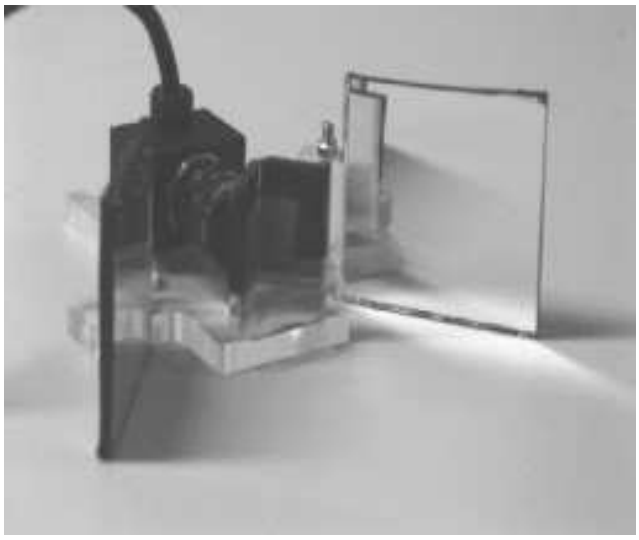


FIG. 5.3 – *Vue du système.*

Le montage du système [MD95] ne présente qu'une seule difficulté qui est le collage des miroirs centraux entre eux puis sur la plaque. Il faut en effet coller ces miroirs le plus verticalement possible de manière à optimiser le recouvrement des champs visuels droit et gauche. Le réglage de l'ouverture des grands miroirs dépend de la focale de la caméra, et de la scène que l'on souhaite observer (figure 5.4).

5.1.6 Utilisation

Ce système peut être utilisé avec toute méthode de stéréoscopie fonctionnant à partir d'images. En effet, il suffit de bien régler la position des miroirs pour que la séparation se situe au centre de l'image, et de séparer l'image en deux juste après l'acquisition. Tout se passe alors comme si on avais deux

FIG. 5.4 – *Image prise avec le système de miroirs.*

images de largeur moitié prises par deux caméras distinctes (on peut également « jeter » une partie de la bande floue située entre les deux images). Une partie des résultats présentés dans cette thèse utilisent effectivement des images prises par ces miroirs (chapitres 3 et 4).

5.2 Projet de système d'assistance chirurgicale

La description de cette seconde application sera plus brève, d'abord parce que le maître d'œuvre en a été Jacques FELDMAR, qui a travaillé sur ce sujet pendant sa thèse dans le projet Épidaure de l'INRIA, mais aussi parce qu'il y a eu une excellente publication sur le sujet [BFAD95].

Pour comprendre en quoi ce système pourrait être utile, il suffit d'expliquer les différentes phases préopératoires[3] d'un patient devant subir, par exemple, une opération d'une tumeur au cerveau. On commence par fixer par des vis au crâne du patient un cadre stéréotaxique fait d'une matière qui sera facilement visible sur les images IRM, qui va servir pendant le reste de la phase préopératoire comme un « repère de référence ». On fait ensuite un examen IRM puis on peut déposer le cadre stéréotaxique (mais il reste toujours les fixations). Le chirurgien peut alors localiser précisément dans le volume de données IRM la zone d'intervention, déterminer quel schéma opératoire utiliser, notamment pour minimiser les lésions au cerveau. Évidemment, toutes ces zones sont repérées par rapport au repère de référence qui apparaît clairement dans le volume IRM. On replace alors le cadre, afin

3. Le lecteur excusera l'inexactitude des termes médicaux employés ici, et le contenu de ce document ne saurait avoir de valeur médicale.

de repérer sur le patient lui-même les zones d'intervention, avant d'effectuer l'opération elle-même, le cadre restant fixé au patient.

L'élément le plus douloureux et le plus traumatisant pour le patient est évidemment la pose du cadre stéréotaxique, et c'est cette étape que nous avons tenté d'éliminer en proposant le système suivant. Dans notre système, le patient commence par passer l'examen IRM sans pose de cadre. Le chirurgien repère les zones d'intervention et détermine le schéma opératoire grâce aux images IRM, et le marquage préopératoire des zones d'intervention se fait toujours sans pose de cadre, grâce à un système de vision stéréoscopique. En effet, pendant cette phase, un système stéréoscopique calibré prend des images du visage du patient et reconstruit la surface du visage en 3-D en temps réel. Ce visage est alors précisément positionné (ou « recalé ») par rapport au volume IRM acquis précédemment, et un système de repérage d'instruments permet de repérer précisément la position 3-D les instruments du chirurgien par rapport au système stéréoscopique.

Pour résumer, nous avons donc la position des instruments par rapport aux caméras, la position du visage par rapport aux caméra par stéréoscopie, la position du visage par rapport au volume IRM par recalage, et la position de la zone à opérer dans le volume IRM. Nous avons donc la position des instruments du chirurgien par rapport au volume IRM et nous pouvons visualiser sur un écran une image de synthèse représentant la position des instruments par rapport au volume IRM. On peut également disposer d'une visualisation de type « réalité augmentée », dans laquelle les structures profondes, extraites de l'IRM, sont superposées aux images vidéo du patient. Quelques exemples démonstratifs sont présentés dans [BFAD95].

5.2.1 Partie vision

La partie vision du système consiste en un système stéréoscopique, qui peut être formé soit d'une paire de caméra synchronisées, soit d'une unique caméra munie d'un système de miroirs semblable à celui décrit § 5.1.

Ce système pourra être équipé soit de caméras noir et blanc, soit de caméras couleurs « Tri-CCD », c'est-à-dire disposant un capteur CCD pour chaque canal. Les caméras couleurs classiques disposent d'un unique capteur CCD, recouvert d'un filtre de telle manière que certains pixels servent au canal rouge, d'autres au bleu, et d'autres au vert. Dans la pratique, on a deux pixels verts pour un pixel rouge et un pixel bleu, et les images rouge, verte et bleue sont donc incomplète (elles comportent des trous d'information aux endroits où sont les pixels des autres couleurs) et décalées l'une par rapport à l'autre. Le résultat est visuellement satisfaisant (grâce à diverses méthodes de filtrage) mais ne permet pas de faire de la stéréoscopie. En clair, si on veut faire de la stéréoscopie couleur, il faut une caméra Tri-CCD.

Le système est fortement calibré, afin d'obtenir une reconstruction euclidienne du visage du patient (chapitre 4). Le calcul des cartes de disparité

par stéréoscopie, couleur ou non, peut être ensuite exécutée en parallèle sur une machine multi-processeurs grâce à la méthode présentée § 3.1.7. La reconstruction euclidienne du visage peut ensuite être recalée, en temps réel, par rapport au volume IRM.

5.2.2 Recalage par rapport à l'IRM

Pour la phase de recalage de la reconstruction obtenue par stéréoscopie par rapport au volume IRM, on se reportera à la thèse de Feldmar [Fel95], et à l'article [BFAD95] traitant de l'application proposée ici.

5.2.3 Repérage de la position des instruments

Pour le repérage de la position des instruments, plusieurs systèmes existent déjà en phase opérationnelle, et on peut citer par exemple le système OptoTrack, fonctionnant grâce à des diodes LED fixées aux instruments du chirurgien. Pour des détails sur les techniques de repérage des instruments, on pourra par exemple se reporter à la thèse de Bainville [Bai96].

5.2.4 Mise en œuvre

La mise en œuvre de l'ensemble du système comme nous l'avons imaginé consiste en un système d'acquisition vidéo monoculaire (les système binoculaires sont très peu répandus et très onéreux) muni du système stéréoscopique à miroirs, relié à une station multi-processeurs ou à un réseau de stations. Cette machine multi-processeurs est chargée d'effectuer la stéréoscopie par corrélation sous PVM. Un processeur collecte les données, effectue le recalage par rapport à l'IRM, et passe les informations de position à une station graphique temps réel. Cette station affiche, éventuellement sur un dispositif stéréoscopique, la position des instruments du chirurgien par rapport aux zones d'intérêt du volume IRM.

Malheureusement nous n'avons pas pu trouver les ressources en temps et en matériel pour tester ce dispositif, amis il mériterait d'être essayé. Il pourrait également être comparés à des dispositifs similaires utilisant des techniques différentes pour la reconstruction du visage, comme de la lumière structurée ou un télémètre laser.

Chapitre 6

Conclusion

Grâce aux méthodes présentées dans ce document, nous arrivons presque au but que nous nous étions fixé au départ, c'est-à-dire de réaliser une sorte d'appareil de numérisation 3-D à base de vision stéréoscopique.

Nous avons commencé par donner diverses solutions au problème du calibrage du système stéréoscopique, qui consiste à trouver la fonction projetant les points de l'espace 3-D dans le plan de la caméra. Suivant le modèle de caméra adopté, avec ou sans distorsion, les techniques de calibrage varient, et nous avons passé en revue les techniques déjà existantes, du calibrage fort à l'aide d'une mire à l'auto-calibrage. Les avancées présentées dans cette thèse concernent le calibrage automatique de la distorsion à partir d'images d'environnements structurés (comme des scènes d'intérieur), une nouvelle méthode de calibrage dite « hybride » utilisant à la fois un calibrage faible automatique et des images d'une mire, et l'auto-calibrage d'un système stéréoscopique où les deux caméras sont rigidement liées.

Les images sont acquises grâce au système stéréoscopique, puis doivent être rectifiées afin de simplifier le processus de stéréoscopie. La présentation originale que nous avons faite de la rectification permet à la fois de mieux la comprendre et de mieux l'utiliser. Par exemple, nous avons proposé une méthode de rectification locale qui permet à la fois de simplifier la mise en correspondance stéréoscopique et de l'accélérer.

Le point fort de cette thèse est la stéréoscopie par corrélation, qui est une technique de mise en correspondance stéréoscopique de surfaces. Nous proposons en effet une nouvelle méthode, appelée corrélation fine, qui donne des résultats de qualité largement supérieure à la corrélation classique, mais qui nécessite un temps de calcul plus important. Le modèle de surface utilisé par la corrélation fine est une surface d'orientation (voire de courbures) quelconque, alors que la méthode classique fait l'hypothèse que la surface observée est parallèle aux plans des caméras.

Le résultat de la stéréoscopie par corrélation est une image des disparités des points observés entre les deux images, à partir de laquelle on obtient

la géométrie 3-D par reconstruction. Notre approche de la reconstruction permet d'obtenir des expressions très simple de la position, de la normale, et des courbures 3-D à partir de la disparité stéréoscopique et de ses dérivées. Cette simplification a été possible grâce à l'introduction du concept de *matrice de reconstruction*.

Le point faible des résultats que nous avons présenté dans cette thèse est évidemment le faible nombre de résultats quantitatifs pour la corrélation comme pour la reconstruction. Il serait en effet nécessaire, pour pleinement valider ces résultats qui sont déjà satisfaisant sur le plan visuel, d'utiliser des images réelles ou synthétiques d'objets de géométrie connue. Ceci permettrait de quantifier l'amélioration apportée par la corrélation fine sur la corrélation classique, ainsi que la précision des normales et courbures calculées par corrélation fine.

Avant d'arriver au système stéréoscopique complet et autonome de nos rêves, il reste encore quelques étapes à franchir, et ce ne sont pas les plus simples. La reconstruction d'un objet que nous obtenons n'est que partielle, et pour avoir automatiquement une reconstruction complète, comme celle présentée en annexe B, il faudrait utiliser une méthode de recalage 3-D pour positionner les morceaux de surface les uns par rapport aux autres. Feldmar [Fel95] propose une méthode efficace pour résoudre ce problème, à laquelle il faut ajouter la prise en compte des données d'intensité de la surface.

Une fois les morceaux de surface recalés les uns par rapport aux autres, il est nécessaire de les fusionner, pour n'obtenir qu'une surface. On obtient alors une description complète de l'objet mais le volume de données est beaucoup trop important, puisqu'on a un point par pixel. Pour réduire ce volume de données, il est nécessaire d'utiliser une technique de décimation [SZL92]. La décimation est un problème qui a déjà été bien cerné dans le cadre de l'infographie, et il existe même une implantation dans la Visualisation Toolkit (vtk)[1] dont le code source est disponible. Si on diminue le nombre de points 3-D par décimation, dont le résultat est une triangulation de la surface, on peut alors utiliser la corrélation fine aux sommet de cette triangulation pour en calculer très précisément la position.

Le système stéréoscopique d'acquisition 3-D ainsi complété pourrait être implanté sur un ordinateur personnel muni d'une caméra. Il permettrait ainsi de réaliser un des vœux les plus chers du chercheur en vision : mettre à la portée de tous un objet simple d'utilisation, fruit de la haute technologie. Quelqu'un veut prendre la suite ?

1. http://www.kitware.com/

Annexe A

Automatic calibration and removal of distortion from scenes of structured environments
Calibrage automatique de la distorsion à partir d'images d'environnements structurés[1]

Most algorithms in 3-D Computer Vision rely on the pinhole camera model because of its simplicity, whereas video optics, especially low-cost wide-angle lens, generate a lot of non-linear distortion which can be critical.

To find the distortion parameters of a camera, we use the following fundamental property: a camera follows the pinhole model if and only if the projection of every line in space onto the camera is a line. Consequently, if we find the transformation on the video image so that every line in space is viewed in the transformed image as a line, then we know how to remove the distortion from the image.

The algorithm consists of first doing edge extraction on a possibly distorted video sequence, then doing polygonal approximation with a large tolerance on these edges to extract possible lines from the sequence, and then finding the parameters of our distortion model that best transform

1. Cette annexe est une version mise à jour de [DF95a].

these edges to segments.

Results are presented on real video images, compared with distortion calibration obtained by a full camera calibration method which uses a calibration grid.

A.1 Introduction

A.1.1 External, internal, and distortion calibration *Paramètres intrinsèques, extrinsèques, et de distorsion*

In the context of 3-D computer vision, camera calibration consists of finding the mapping between the 3-D space and the camera plane. This mapping can be separated in two different transformation: first, the displacement between the origin of 3-D space and the camera coordinate system, which forms the external calibration parameters (3-D rotation and translation), and second the mapping between 3-D points in space and 2-D points on the camera plane in the camera coordinate system, which forms the internal camera calibration parameters.

The internal camera calibration parameters depend of the camera. In the case of an orthographic or affine camera model, optic rays are orthogonal to the camera plane and there are only 3 parameters corresponding to the spatial sampling of the image plane. The perspective (or projective) camera model involves two more camera parameters corresponding to the position of the principal point in the image (which is the intersection of the optical axis with the image plane). For many application which require high accuracy, or in cases where low-cost or wide-angle lens are used, the perspective model is not sufficient and more internal calibration parameters must be added to take into account camera lens distortion.

The distortion parameters are most often coupled with internal camera parameters, but we can also use a camera model in which they are decoupled. Decoupling the distortion parameters from others can be equivalent to adding more degrees of freedom to the camera model.

A.1.2 Brief summary of existing related work *État de l'art*

Here is an overview of the different kinds of calibration methods available. The goal of this section is not to do an extensive review, and the reader can find more information in [Bey92, LT88, Sla80].

The first kind of calibration method is the one that uses a calibration grid with feature points whose world 3-D coordinates are known. These feature points, often called control points, can be corners, dots, or any features that can be easily extracted for computer images. Once the control points are

identified in the image, the calibration method finds the best camera external (rotation and translation) and internal (image aspect ratio, focal length, and possibly others) parameters that correspond to the position of these points in the image. The simplest form of camera internal parameters is the standard pinhole camera [FT87], but in many cases the distorsion due to wide-angle or low-quality lens has to be taken into account [Tsa87, Bey92]. Using a calibration method with a pinhole camera model on lens with non-negligible distortion may result in high calibration errors.

The problem with these methods that compute the external and internal parameters at the same time arises from the fact that there is some kind of coupling between internal and external parameters that result in high errors on the camera internal parameters [WCH92].

Another family of methods is those that use geometric invariants of the image features rather than their world coordinates, like parallel lines [CT90, BMZ92] or the image of a sphere [Pen91].

The last kind of calibration techniques is those that do not need any kind of known calibration points. These are also called auto-calibration methods, and the problem with these methods is that if all the parameters of the camera are unknown, they are still very unstable [FLM92]. Known camera motion helps in getting more stable and accurate results [Ste95, Har94a] but it's not always that easy to get "pure camera rotation".

A few other calibration methods are only interested in distortion calibration, like the plumb line method [Bro71]. Another method presented in [BMB93] uses a calibration grid to find a generic distortion function, represented as a 2-D vector field.

A.1.3 Overview of our method *Résumé de la méthode*

Since many auto-calibration [FLM92] or weak calibration [ZDFL95] techniques rely on a pinhole (i.e. perspective) camera model, our main idea was to calibrate only the image distortion, so that any camera could be considered as a pinhole camera after the application of the inverse of the distortion function to image features. We also don't want to rely on a particular camera motion [Ste95] in order to be able to work on any kind of video recordings or snapshots (e.g. surveillance video recordings) for which there can be only little knowledge on self-motion, or some observed objects may be moving.

The only constraint is that the world seen though the camera must contain 3-D lines and segments. It can be city scenes, interior scenes, or aerial views containing buildings and human-made structures. Edge extraction and polygonal approximation is performed on these images in order to detect possible 3-D edges present in the images, and after this we just look for the distortion parameters that minimize the distortion of the 3-D segments

projected to the image.

After we found a first estimate of the distortion parameters, we perform another polygonal approximation on the undistorted edges, this way 3-D lines that were broken into several segments because of distortion will become one segment, and outliers (3-D curves that were detected as a 3-D segment because of their small curvature) are implicitly eliminated. We continue this iterative process until we fall into a stable minimum of the distortion error after the polygonal approximation step.

A.2 Description of the method *Description de la méthode*

A.2.1 The distortion model *Le modèle de distorsion*

The mapping between 3-D points and 2-D image points can be decomposed into a perspective projection and a function that models the deviations from the ideal pinhole camera. A perspective projection associated with the focal length f maps a 3-D point M whose coordinates in the camera-centered coordinate system are (X, Y, Z) to an "undistorted" image point $m_u = (x_u, y_u)$ on the image plane:

$$\begin{aligned} x_u &= f\frac{X}{Z} \\ y_u &= f\frac{Y}{Z} \end{aligned} \tag{A.1}$$

Then, the image distortion transforms m_u to a distorted image point m_d. The image distortion model [Sla80] is usually given as a mapping from the distorted image coordinates, which are observable in the acquired images, to the undistorted image coordinates, which are needed for further calculations.

Finally, image plane coordinates are converted to frame buffer coordinates, which can be expressed either in pixels or in normalized coordinates (i.e. pixels divided by image dimensions), depending on the unit of f:

$$\begin{aligned} x_i &= s_x x_d + c_x \\ y_i &= y_d + c_y \end{aligned} \tag{A.2}$$

The image distortion function can be decomposed in two terms: radial and tangential distortion. Radial distortion is a deformation of the image along the direction from a point called the center of distortion to the considered image point, and tangential distortion is a deformation perpendicular to this direction. The center of distortion is invariant under both transformations.

It was found that for many machine vision applications, tangential distortion need not to be considered [Tsa87]. The lens distortion can then be written as an infinite series:

$$x_u = x_d(1 + \kappa_1 r_d^2 + \kappa_2 r_d^4 + \cdots) \tag{A.3}$$

where $r_d = \sqrt{x_d^2 + y_d^2}$. Several tests [Bey92, Tsa87] showed that using only the first order radial symmetric distortion parameter κ_1, one could achieve an accuracy of about 0.1 pixels in image space, using lenses exhibiting large distortion together with the other parameters of the perspective camera [FT87].

In our case we want to decouple the effect of distortion from the projection on the image plane, because all we want to calibrate is the distortion. Consequently, in our model, the center of distortion (c_x, c_y) will be different from the principal point. It was shown [Ste93] that this is mainly equivalent to adding decentering distortion terms to the distortion model of equation A.2.1. A higher order effect of this is to apply an (very small) affine transformation to the image, but the affine transform of a pinhole camera is also a pinhole camera.

Moreover, the image aspect ratio that we use in the distortion model may not be the same as the real aspect ratio. The difference between these two aspect ratios will result in another term of tangential distortion. To summarize, κ_1 is the first order distortion, the coordinates of the center of distortion (c_x, c_y) correspond to decentering distortion because the center of distortion may be different from principal point, and the difference between the distorsion aspect ratio s_x and the real aspect ratio correspond to a term of tangential distortion.

In the following, all coordinates are frame buffer coordinates, either expressed in pixels or normalized (by dividing x by the image width and y by the image height) to be unit-less.

The undistorted coordinates are given by the formula:

$$\begin{aligned} x_u &= x_d + (x_d - c_x)\kappa_1 r_d^2 \\ y_u &= y_d + (y_d - c_y)\kappa_1 r_d^2 \end{aligned} \tag{A.4}$$

where $r_d = \sqrt{\left(\frac{x_d - c_x}{s_x}\right)^2 + (y_d - c_y)^2}$ is the distorted radius.

The distorted coordinates in function of the undistorted coordinated are given by the solution of the equation:

$$r_u = r_d\left(1 + \kappa_1 r_d^2\right) \tag{A.5}$$

where $r_u = \sqrt{\left(\frac{x_u - c_x}{s_x}\right)^2 + (y_u - c_y)^2}$ is the undistorted radius and r_d the distorted radius.

This is a polynomial of degree three of the form $r_d^3 + cr_d + d = 0$, with $c = \frac{1}{\kappa_1}$ and $d = -cr_u$, which can be solved using the Cardan method which is a direct method for solving polynomials of degree three. It has either one or three real solutions, depending on the sign of the discriminant:

$$\Delta = Q^3 + R^2$$

where $Q = \frac{c}{3}$ and $R = -\frac{d}{2}$.

If $\Delta > 0$ there is only one real solution:

$$r_d = \sqrt[3]{R + \sqrt{\Delta}} + \frac{Q}{\sqrt[3]{R + \sqrt{\Delta}}} \tag{A.6}$$

and if $\Delta < 0$ there are three real solutions but only one is valid because when r_u is fixed, r_d must be continuous in function of κ_1. The continuity at $\kappa 1 = 0$ gives the solution:

$$r_d = -S \cos T + S\sqrt{3} \sin T \tag{A.7}$$

where $S = \sqrt[3]{\sqrt{R^2 - \Delta}}$ and $T = \frac{1}{3} \arctan \frac{\sqrt{-\Delta}}{R}$

The distorted coordinates are then given by:

$$\begin{aligned} x_d &= c_x + (x_u - c_x)\frac{r_d}{r_u} \\ y_d &= c_y + (y_u - c_y)\frac{r_d}{r_u} \end{aligned} \tag{A.8}$$

It is cheaper in terms of calculations to detect features in the distorted image and to undistort them than to undistort the whole image (which requires solving a third degree polynomial equation at each point and bilinear interpolation to compute the gray level) and to extract the feature from the undistorted image. For some kinds of features which depend on the perspective projection and that must be detected directly in the intensity image, one must nevertheless undistort the whole image. It that case, if calibration time is not crucial but images need to be undistorted quickly, i.e. only the transform function from undistorted to distorted coordinated is to be used more often than its inverse in a program's main loop, then a good solution is to switch the distortion function and its inverse. Equation A.4 would become the distortion function and equation A.8 its inverse. That way the automatic distortion calibration step would be costly because it requires undistorting edge features, but one the camera is calibrated, the un-distortion of the whole intensity images would be faster.

A.2.2 Principle
Principe

The goal of the distortion calibration is to find the transformation (or undistortion) that maps the actual camera image plane onto an image following

the perspective camera model. To find the distortion parameters described in section A.2.1, we use the following fundamental property: a camera follows the perspective camera model if and only if the projection of *every* 3-D line in space onto the camera plane is a line. Consequently, all we need is a way to find projections of 3-D lines in the image (they are not lines anymore in the images, since they are distorted, but curves), and a way to measure how much each 3-D line is distorted in the image. Then we will just have to let vary the distortion parameters and try to minimize the distortion of edges transformed using these parameters.

A.2.3 Edge detection with sub-pixel accuracy *Détection de contours sous-pixeliques*

The first step of the calibration consists of extracting edges from the images. Since image distortion is sometimes less than a pixel at image boundaries, there was definitely a need for an edge detection method with a sub-pixel accuracy. We developed an edge detection method [Dev95], described in annex C, based on the classical Non-Maxima Suppression (NMS) of the gradient norm in the direction of the gradient which gives edge position with a precision varying from 0.05 pixels for a noise-free synthetic image to 0.3 pixels for an image Signal to Noise Ratio (SNR) of 18dB (which is actually a lot of noise, the VHS videotapes SNR is about 50dB).

A.2.4 Finding 3-D segments in a distorted image *Recherche de segments 3-D dans l'image*

In order to calibrate distortion, we must find edges in the image which are most probably images of 3-D segments. The goal is not to get all segments, but to find the most probable ones. For this reason, we do not care if a long segment, because of its distortion, is broken into smaller segments.

Therefore, and because we are using a subpixel edge detection method, we use a very small tolerance for polygonal approximation: the maximum distance between edge points and the segment joining both ends of the edge must typically be less than 0.4 pixels. We also put a threshold on segment length of about 60 pixels for a 640×480 image, because small segments may contain more noise than useful information about distortion.

Moreover, because of the corner rounding effect [DG90, DG93] due to edge detection, we throw out a few edgels (between 3 and 5, depending of the amount of smoothing performed on the image before edge detection) at both ends of each detected segment edge.

A.2.5 Measuring distortion of a 3-D segment in the image
Mesurer la distorsion d'un segment 3-D dans l'image

In order to find the distortion parameters we use a measure of how much each detected segment is distorted. This distortion measure will then be minimized to find the best calibration parameters. One could use for example the mean curvature of the edges, or any distance function on the edge space that would be zero if the edge is a perfect segment and the more the segment would be distorted, the bigger the distance would be.

We chose a simple measure of distortion which consists of doing a least squares approximation of each edge which should be a projection of a 3-D segment by a line [DVF91], and to take for the distortion error the sum of squares of the distances from the point to the line (i.e. the χ^2 of the least square approximation). That way, the error is zero if the edge lies exactly on a line, and the bigger the curvature of the edge, the bigger the distortion error. For each edge, the distortion error is defined to be:

$$\chi^2 = a \sin^2 \omega - 2\,|b|\,|\sin \omega| \cos \omega + c \cos^2 \omega \tag{A.9}$$

where:

$$a = \sum_{i=1}^{n} x_i^2 - \frac{1}{n} \left(\sum_{i=1}^{n} x_i \right)^2 \tag{A.10}$$

$$b = \sum_{i=1}^{n} x_i y_i - \frac{1}{n} \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i \tag{A.11}$$

$$c = \sum_{i=1}^{n} y_i^2 - \frac{1}{n} \left(\sum_{i=1}^{n} y_i \right)^2 \tag{A.12}$$

$$\alpha = a - c \tag{A.13}$$

$$\beta = \frac{\alpha}{2\sqrt{\alpha^2 + 4b^2}} \tag{A.14}$$

$$|\sin \omega| = \sqrt{\frac{1}{2} - b} \tag{A.15}$$

$$\cos \omega = \sqrt{\frac{1}{2} + b} \tag{A.16}$$

ω is the angle of the line in the image, and $\sin \omega$ should have the same sign as b.

A.2.6 The whole calibration process
Le processus de calibrage

The whole distortion calibration process is not done in only one iteration (edge detection, polygonal approximation, and optimization), because there

may be outliers in the segments detected by the polygonal approximation, i.e. edges which aren't really 3-D segments. Moreover, some 3-D segments may be broken into smaller edges because the first polygonal approximation is done on distorted edges. By doing another polygonal approximation after the optimization, on undistorted edges, we can eliminate many outliers easily and sometimes get longer segments which contain more information about distortion. This way we get even more accurate calibration parameters.

A first version of the calibration process is:

1. Load images.
2. Do subpixel edge detection and linking on the images.
3. Do polygonal approximation on distorted edges to extract segment candidates.
4. Compute the distortion error $E_0 = \sum \chi^2$ (sum is done over all the detected segments).
5. Optimize the distortion parameters κ_1, c_x, c_y, s_x to minimize distortion error.
6. Compute the distortion error E_1 for the optimized parameters.
7. If the relative change of error $\frac{E_0 - E_1}{E_1}$ is less than a threshold, stop here.
8. Do polygonal approximation on undistorted edges.
9. Go to step 4.

By minimizing on the four parameters when the data still contains many outliers, there is a risk of getting farther from the optimal parameters. For this reason, steps 3 to 9 are first done with optimization only on κ_1 until the termination condition of step 7 is verified, then c_x and c_y are added, and finally full optimization on the four distortion parameters is performed.

A.3 Experimental setup *Dispositif expérimental*

A.3.1 Hardware *Matériel*

We used various hardware setups to test the accuracy of the distortion calibration, from low-cost video-conference video hardware to high-quality cameras and frame-grabber.

The lowest quality hardware is a very simple video acquisition system included with every Silicon Graphics Indy workstation. This system is not

designed for accuracy nor quality and consists of an IndyCam camera coupled with the standard Vino frame grabber. The acquired image is 640×480 pixels interlaced, and contains a lot of distortion and blur caused by the cheap wide-angle lens. The use of an on-line camera allows very fast image transfer between the frame grabber and the program memory using Direct Memory Access (DMA), so that we are able to do fast distortion calibration. The quality of the whole system seems comparable to this of a VHS videotape.

Other images were acquired using an Imaging Technologies acquisition board together with several different camera setups: a Sony XC75CE camera with 8mm, 12.5mm, and 16mm lens (the smaller the focal length, the more important the distortion), and an old Pulnix TM-46 camera with 8mm lens.

A.3.2 Software
Lociciel

The distortion calibration program is a stand-alone program that can either work on images acquired on-line using a camera and a frame grabber or acquired off-line and saved to disk. Computation of the image gradient and edge detection were done using the Robotvis libraries[2].

The optimization step was performed using the subroutine `lmdif` from MINPACK or the subroutine `dnls1` from SLATEC, both packages being available from Netlib[3].

A.4 Results and comparison with a full calibration method
Résultats et comparaison avec une méthode de calibrage fort

A.4.1 The full calibration method
La méthode de calibrage fort

In order to evaluate the validity of the distortion parameters obtained by our method, we compared them to those obtained by a method for full calibration (both external and internal) that incorporates comparable distortion parameters. The software we used to do full calibration implements the Tsai calibration method [Tsa87] and is freely available (the source code can be found in the Vision List archive[4]). This software implements calibration of external (rotation and translation) and internal camera parameters at the same time. The internal parameter set is composed of the pinhole camera

[2] ftp://krakatoa.inria.fr/pub/robotvis/
[3] http://www.netlib.org/
[4] ftp://teleosresearch.com/VISION-LIST-ARCHIVE/

parameters except for the shear parameter (which is very close to zero on CCD cameras anyway [Bey92]), and of the first radial distorsion parameter. From the result of this calibration mechanism, we can extract the position of the principal point, the image aspect ratio, and the first radial distortion parameter.

As seen in section A.2.1, though, these are not exactly the same parameters as those that we can compute using our method, since we allow more degrees of freedom for the distortion function: two more parameters of decentering distortion and one parameter of tangential distortion. Having different coordinates for the principal point and the center of distortion, and for the image aspect ratio and distortion aspect ratio.

A.4.2 Results
Résultats

The distortion calibration method was applied to sets of about 30 images (see Figure A.1) for each camera/lens combination, and the results for the four parameters of distortion are shown in Table A.1. The initial values for the distortion parameters before the optimization were set to "reasonable" values, i.e. the center of distortion was set to the center of the image, κ_1 was set to zero, and s_x to the image aspect ratio, computed for the camera specifications. For the IndyCam, this gave $c_x = c_y = \frac{1}{2}$, $\kappa_1 = 0$ and $s_x = \frac{3}{4}$.

Figure A.1: Some of the images that were used for distortion calibration.

camera/lens	A	B	C	D	E
κ_1	0.154	0.041	0.016	0.012	-0.041
c_x	0.493	0.635	0.518	0.408	0.496
c_y	0.503	0.405	0.122	0.205	0.490
s_x	0.738	0.619	0.689	0.663	0.590

Table A.1: The distortion parameters obtained on various camera/lens setups using our method, in normalized image coordinates: First radial distortion parameter, position of the center of distortion, and distortion aspect ratio. A is IndyCam, B is Sony XC-75E camera with 8mm lens, C is Sony XC-75E camera with 12.5mm lens, D is Sony XC-75E with 16mm lens, E is Pulnix camera with 8mm lens

camera/lens	A	B	C	D
κ_1	0.135	0.0358	0.00772	0.00375
c_x	0.475	0.514	0.498	0.484
c_y	0.503	0.476	0.501	0.487
s_x	0.732	0.678	0.679	0.678

Table A.2: The distortion parameters obtained using the Tsai calibration method, in normalized image coordinates: First radial distortion parameter, position of the principal point, and image aspect ratio. They do not have exactly the same meaning as in Table A.1, as explained in section A.2.1. See Table A.1 for details on the camera/lens configurations

Figure A.2 shows a sample image, before and after the correction. This image was affected by pin-cushion distortion, corresponding to a positive value of κ_1. Barrel distortion corresponds to negative values of κ_1.

We also calibrated the same cameras using the Tsai method and a calibration grid (Figure A.3) with 128 points, and we computed some parameters corresponding more or less to our distortion parameters from the result of this full calibration method (Table A.2). As explained in section A.2.1, these are not the same as our distortion parameters, because we introduced a few more degrees of freedom in the distortion function, allowing decentering and tangential distortion. This explains why the distortion center found on low-distortion cameras such as the Sony 16mm are so far away from the principal point.

For cameras with high distortion, like the IndyCam and the cameras with 8mm lens, The center of distortion and the distortion aspect ratio are close to the principal point and the image aspect ratio. For better comparison with a grid-based calibration, we should use a camera model that includes the same distortion parameters as ours, i.e. introduce the center of distortion and distortion aspect ratio.

Figure A.2: A distorted image with the detected segments (left) and the same image at the end of the distortion calibration with segments extracted from undistorted edges (right): some outliers were removed and sometimes longer segments are detected.

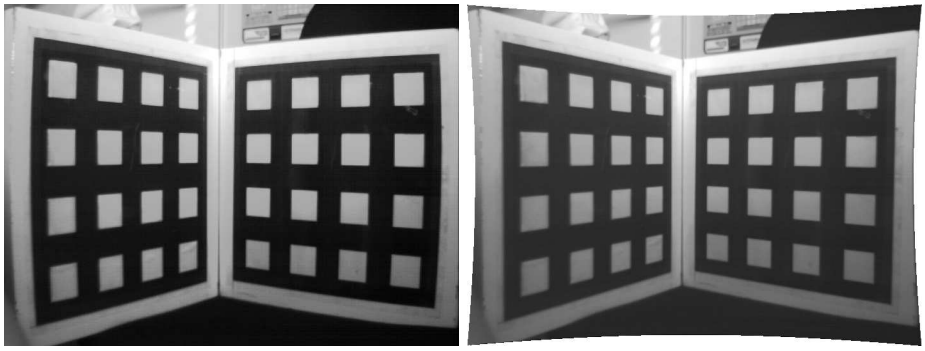


Figure A.3: The calibration grid used for Tsai calibration: original distorted image (left) and image undistorted using the parameters computed by our method (right).

A.5 Discussion

With computer vision applications demanding more and more accuracy in the camera model and the calibration of its parameters, there is definitely a need for calibration methods that don't rely on the simple and linear pinhole camera model. Camera optics still have lots of distortion, and zero-distortion wide-angle lens exist but remain very expensive.

The automatic distortion calibration method presented here has many advantages over other existing calibration methods that use a camera model with distortion [Bey92, BMB93, Ste95, Tsa87]. First, it makes very few assumptions on the observed world: there is no need for a calibration grid [Bey92, BMB93, Tsa87]. All it needs is images of scenes containing 3-D segments, like interior scenes or city scenes. Second, it is completely automatic, and camera motion need not to be known [Ste93, Ste95]. It can be applied to images acquired off-line, as an example they could come from a surveillance videotape or a portative camcorder. The comparison with a full grid-based calibration method presented in this paper is still not complete, since the grid-based calibration method we used didn't have the same distortion parameters as our method. The next step in the validation of this method is to do better comparisons.

Once the distortion is calibrated, any computer vision algorithm that relies on the pinhole camera model can be used, simply by applying the inverse of the distortion either to image features (edges, corners, etc.) or to the whole image. This method could also be used together with auto-calibration or weak calibration methods that would take into account the distortion parameters. The distortion calibration could be done before auto-calibration, so that the latter would use un-distorted features and images, or during auto-calibration, the distortion error being taken into account during the auto-calibration process.

Annexe B

From projective to Euclidean reconstruction
De la reconstruction projective à la reconstruction euclidienne[1]

To make a Euclidean reconstruction of the world seen through a stereo rig, we can either use a calibration grid, and the results will rely on the precision of the grid and the extracted points of interest, or use self-calibration. Past work on self-calibration is focussed on the use of only one camera, and gives sometimes very unstable results.

In this paper, we use a stereo rig which is supposed to be weakly calibrated using a method such as the one described in [DZLF94]. Then, by matching two sets of points of the same scene reconstructed from different points of view, we try to find both the homography that maps the projective reconstruction [Fau92] to the Euclidean space and the displacement from the first set of points to the second set of points.

We present results of the Euclidean reconstruction of a whole object[2] from uncalibrated cameras using the method proposed here.

B.1 Introduction

This article is concerned with the following problem. Given a weakly calibrated stereo rig, i.e. a pair of camera with known epipolar geometry, we know that we can obtain 3-D reconstructions of the environment up to an

1. Cette annexe est une version mise à jour de [DF95b, DF96]
2. http://www.inria.fr/robotvis/demo/proj2eucl/

unknown projective transformation [Fau92, HGC92]. We call such a reconstruction a *projective* reconstruction. In particular, no affine or Euclidean information can a priori be extracted from it unless some further information is available [Fau94]. The problem is then to determine what is the information that is missing and how can it be recovered. We provide a very simple answer to both questions: with one rigid displacement of the stereo rig, the three-dimensional structure of the scene can be in general uniquely recovered up to a similitude transformation using some elementary matrix algebra, assuming that reliable correspondences between the two projective reconstructions obtained from the two viewpoints can be established. We call such a reconstruction a *Euclidean* reconstruction. A similar result was obtained [ZBR95] but the resulting scheme was a closed form solution computed from two views of the scene, whereas this method can be used with many more views, giving more stability on the solution.

This result does not contradict previous results, for example [MF92, LF92] which showed that the *intrinsic* parameters of a camera could be in general recovered from two displacements of the camera because we are using simultaneously two cameras. The method developed here avoids any reference to the intrinsic parameters of the cameras and does not require solving the nonlinear Kruppa equations which are defined in the previous references.

B.2 Goal of the method

Our acquisition system consists of a pair of cameras. This system can be calibrated using a weak calibration method [DZLF94], so that we can make a projective reconstruction [Fau92] of the scene in front of the stereoscopic system, by matching features (points, curves, or surfaces) between the two images.

Projective reconstruction roughly consists of chosing five point matches between the two views and chosing these five points as a projective basis to reconstruct the scene. The five point matches can be either real points (i.e. points that are physically present in the scene) or virtual points. The virtual point matches are calculated by choosing a point in the first camera, and then choosing any point on its epipolar line in the second camera as its correspondant, thus these points satisfy the epipolar constraint but are not the images of a physical point. Let us call $\mathcal{P}$ the resulting projective basis which is thus *attached* to the stereo rig.

Let us now consider a real correspondence (m_1, m_1') between the two images. We can reconstruct the 3-D point M_1 in the projective basis $\mathcal{P}$. Let us now suppose that after moving the rig to another place, the correspondence has become (m_2, m_2'), yielding a 3-D reconstructed point M_2 in the projective basis $\mathcal{P}$. We know from the results of [Fau92, HGC92] that the

two reconstructions are related by a collineation of $\mathcal{P}^3$ which is represented by a full rank 4×4 matrix $\mathbf{H}$ defined up to a scale factor. We denote by the symbol $\cong$ the equality *up to a scale factor*. Thus we have

$$\mathbf{M}_2 \cong \mathbf{H}_{12}\mathbf{M}_1$$

where $\mathbf{M}_1$ and $\mathbf{M}_2$ are homogeneous coordinate vectors of M_1 and M_2 in $\mathcal{P}$.

Let us now imagine for a moment that an orthonormal frame of reference $\mathcal{E}$ is attached to the stereo rig. The change of coordinates from $\mathcal{P}$ to $\mathcal{E}$ is described by a full rank 4×4 matrix $\mathbf{H}_{12}$, also defined up to a scale factor. In the coordinate frame $\mathcal{E}$ the two 3-D reconstructions obtained from the two viewpoints are related by a rigid displacement, not a general collineation. This rigid displacement is represented by the following 4×4 matrix $\mathbf{D}_{12}$:

$$\mathbf{D}_{12} \cong \begin{bmatrix} \mathbf{R}_{12} & \mathbf{t}_{12} \\ \mathbf{0} & 1 \end{bmatrix}$$

where $\mathbf{R}_{12}$ is a rotation matrix. It is well known and fairly obvious that the displacement matrixes form a subgroup of $SL(4)$ which we denote by $E(3)$.

We can now relate the three matrixes $\mathbf{H}_{12}, \mathbf{H}$, and $\mathbf{D}_{12}$ (see figure B.1):

$$\mathbf{H}_{12} \cong \mathbf{H}^{-1}\,\mathbf{D}_{12}\,\mathbf{H} \tag{B.1}$$

Since the choice of $\mathcal{E}$ is clearly arbitrary, the matrix $\mathbf{H}$ is defined up to an arbitrary displacement. More precisely, we make no difference between matrix $\mathbf{H}$ and matrix $\mathbf{DH}$ for an arbitrary element $\mathbf{D}$ of $E(3)$. In mathematical terms, this means that we are interested only in the quotient $SL(4)/E(3)$ of the group $SL(4)$ by its subgroup $E(3)$. Therefore, instead of talking about the matrix $\mathbf{H}$ we talk about its equivalence class $\overline{\mathbf{H}}$. The basic idea of our method is to select in the equivalence class a *canonical element* $\hat{\mathbf{D}}\hat{\mathbf{H}}$, which is the same as selecting a special euclidean frame $\hat{\mathcal{E}}$ among all possible ones and show that equation (B.2) can be solved in general uniquely for $\hat{\mathbf{H}}$ and $\mathbf{D}' \cong \mathbf{D}^{-1}\mathbf{D}_{12}\mathbf{D}$.

B.3 Collineations modulo a displacement

B.3.1 First method

Finding a unique representative of the equivalence classes of the group $SL(4)$ modulo a displacement in $E(3)$ is equivalent to finding a unique decomposition of a collineation (which depends upon 15 parameters) into the product of a displacement (which depends upon 6 parameters) and a member of a subgroup of dimension $15 - 6 = 9$. In fact, we are looking for something similar to the well-known QR or QL decompositions of a matrix into an orthogonal matrix and an upper or lower triangular matrix, where "orthogonal" would be replaced by "displacement".

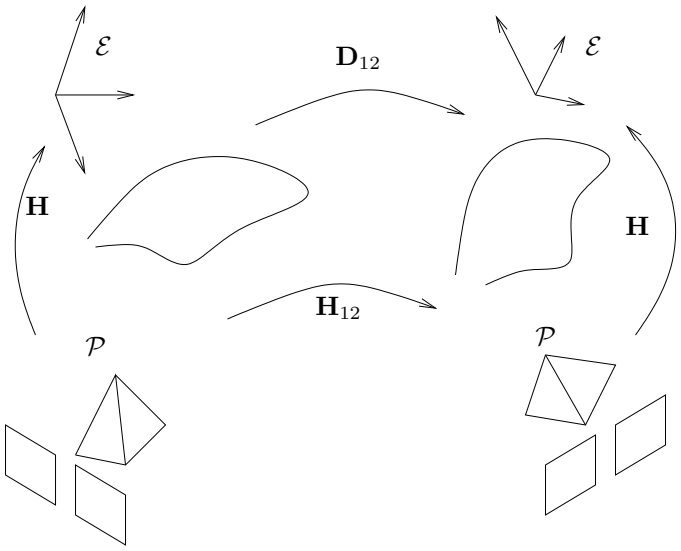


Figure B.1: Given the collineation $\mathbf{H}_{12}$ we want to find the collineation $\mathbf{H}$ that maps the projective reconstruction to the euclidean reconstruction and the displacement $\mathbf{D}_{12}$.

Let us thus consider an element $\mathbf{H}$ of $SL(4)$ and assume that the element h_{44} is non zero. We define the 3×1 vector $\mathbf{t}$ by

$$\mathbf{t} = [h_{14}/h_{44}, h_{24}/h_{44}, h_{34}/h_{44}]^T, \tag{B.2}$$

and write $\mathbf{H}$ as

$$\mathbf{H} = h_{44} \begin{bmatrix} \mathbf{I}_3 & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{l}^T & 1 \end{bmatrix} \tag{B.3}$$

Note that since $\det \mathbf{H} = h_{44} \det \mathbf{A} \neq 0$, this implies that $\det \mathbf{A} \neq 0$. Then there is a unique QL decomposition of $\mathbf{A}$, so that

$$\mathbf{H} = h_{44} \begin{bmatrix} \mathbf{Q} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{L} & \mathbf{0} \\ \mathbf{l}^T & 1 \end{bmatrix} \tag{B.4}$$

where $\mathbf{Q}$ is orthogonal and $\mathbf{L}$ is lower triangular with strictly positive diagonal elements. Thus the group $SL(4)$ modulo the displacements $E(3)$ is isomorphic to the group of the lower triangular matrices with strictly positive diagonal elements. $\mathbf{Q}$ is a rotation if $\det \mathbf{H} > 0$, or a plane symetry if $\det \mathbf{H} < 0$ (remember that the sign of $\det \mathbf{H}$ cannot be changed because $\mathbf{H}$ is of dimension 4.

If we want to decompose $\mathbf{H}$ into a rotation and a translation, we have to remove the constraint on the sign of one the elements of the diagonal of $\mathbf{L}$, e.g. there is no constraint on the sign of the first element of $\mathbf{L}$. In practice, the decomposition will be done using a standard QL decomposition, and

then if **Q** is a plane symmetry rather than a rotation we just have to change the sign of the first element of **L** and of the first column of **Q**, so that the multiplication of both matrices gives the same result and **Q** becomes a rotation.

B.3.2 Second method

Another way to find a unique representative of the equivalence classes of the group of collineations modulo a displacement is to build these representatives by applying constraints on the group of collineations corresponding to the degrees of freedom of a displacement. A simple representative is the one such that the image of the origin is the origin (i.e. the translational term of the collineation is zero), the z axis is globally invariant (i.e. the axis of the rotational term is the z axis), and the image of the y axis is in the yz plane the sign of the y coordinate being invariant (i.e. the angle of the rotation is zero).

These constraints correspond to constraints on the form of matrix **H**. The image of the origin by **H** is the origin itself iff:

$$\mathbf{H}\,[0,0,0,1] = [0,0,0,a] \tag{B.5}$$

The z axis is globally invariant iff:

$$\mathbf{H}\,[0,0,1,0] = [0,0,b,c] \tag{B.6}$$

And the last constraint (the angle of the rotation is zero) corresponds to:

$$\mathbf{H}\,[0,1,0,0] = [0,d,e,f] \tag{B.7}$$

and a, d, and f have the same sign. Consequently, **H** being defined up to a scale factor and non-singular, it can be written as:

$$\mathbf{H} = \begin{bmatrix} g & 0 & 0 & 0 \\ h & d & 0 & 0 \\ j & e & b & 0 \\ k & f & c & 1 \end{bmatrix} \tag{B.8}$$

with $d > 0$ and $f > 0$. Thus equation B.2 becomes:

$$\mathbf{H}_{12} \cong \mathbf{L}^{-1}\,\mathbf{D}_{12}\,\mathbf{L} \tag{B.9}$$

where **L** is a lower triangular matrix with the second and third coordinates of the diagonal positive and the last set to 1.

B.4 Back to the Euclidean world

In this section we show how to recover partly the Euclidean geometry from two projective reconstruction of the same scene. The only thing we have to do is to solve equation B.2 for a lower triangular H. Let us first establish some properties of the colineation between the two reconstructions.

Proposition 2 *Let $\mathcal{A}$ and $\mathcal{B}$ be two projective reconstructions in $\mathcal{P}_3$, the projective space of dimension 3, of the same scene using the same projection matrices from different points of view. Let $\mathbf{H}_{12}$ be the projective transformation (or colineation) from $\mathcal{B}$ to $\mathcal{A}$. Then The eigenvalues of $\mathbf{H}_{12}$ are α (with order of multiplicity 2), $\alpha e^{i\theta}$, and $\alpha e^{-i\theta}$, with $\alpha = \sqrt[4]{\det \mathbf{H}_{12}}$, and the last coordinate of $\mathbf{H}_{12}$, h_{44}, is not zero.*

Equation B.2 yields that $\mathbf{H}_{12}$ and $\mathbf{D}_{12}$ are conjugate (up to a scale factor), then $\mathbf{H}_{12}/\sqrt[4]{\det \mathbf{H}_{12}}$ and $\mathbf{D}_{12}$ have the same eigenvalues, which are: 1 with order of multiplicity two, $e^{i\theta}$, and $e^{-i\theta}$.

Before continuing, we have to prove the following lemma:

Lemme 3 *for each 3×3 real matrix $\mathbf{A}$ whose eigenvalues are $(1, e^{i\theta}, e^{-i\theta})$, there exists a 3×3 lower triangular matrix $\mathbf{L}$ ($l_{ik} = 0$ for $k > i$) with $l_{ii} > 0, i = 1, 2, 3$ defined up to a scale factor, and a rotation $\mathbf{R}$, satisfying $\mathbf{A} = \mathbf{L}^{-1}\,\mathbf{R}\,\mathbf{L}$.*

Since its eigenvalues are either real or conjugate of each other, a real matrix whose eigenvalues are of module one can be decomposed in the form $\mathbf{A} = \mathbf{P}\mathbf{D}_{12}\mathbf{P}^{-1}$, where $\mathbf{D}_{12}$ is a quasi-diagonal matrix of the form:

$$\mathbf{D}_{12} = \begin{bmatrix} B_1 & & 0 \\ & \ddots & \\ 0 & & B_k \end{bmatrix} \tag{B.10}$$

$$\text{with } B_i = [\pm 1] \text{ or } \begin{bmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{bmatrix}$$

We can then compute the QL decomposition of $\mathbf{P}^{-1}$, $\mathbf{P}^{-1} = \mathbf{Q}\mathbf{L}$ which gives:

$$\mathbf{A} = \mathbf{L}^{-1}\mathbf{Q}^T\mathbf{D}_{12}\mathbf{Q}\mathbf{L} = \mathbf{L}^{-1}\mathbf{R}\mathbf{L}$$

where $\mathbf{L}$ is a lower triangular matrix with positive diagonal elements, and $\mathbf{R}$ is an orthogonal matrix. Since $\det \mathbf{A} = \det \mathbf{R} = 1$, then $\mathbf{R}$ is a rotation.□

We now have all the tools needed to prove the following theorem.

Théorème 4 *Let $\mathcal{A}$ and $\mathcal{B}$ be two projective reconstructions of the same scene using the same projection matrices from different points of view. Let $\mathbf{H}_{12}$ be the projective transformation (or colineation) from $\mathcal{B}$ to $\mathcal{A}$. $\mathbf{H}_{12}$ can*

be decomposed in the form $\mathbf{H}_{12} = \lambda \mathbf{L}^{-1}\mathbf{D}_{12}\mathbf{L}$*, where* $\mathbf{L}$ *is lower triangular and* $\mathbf{D}$ *is a displacement. The set of solutions is a two-dimensional manifold, one dimension being the scale factor on the Euclidean space.*

If we take three reconstructions taken from generic points of view, the full Euclidean geometry can be recovered, up to a scale factor.

Let us suppose that $\det \mathbf{H}_{12} = 1$ to eliminate the scale factor on $\mathbf{H}_{12}$. Let $\begin{pmatrix} l \\ 1 \end{pmatrix}$ be an eigenvector of $\mathbf{H}_{12}^T$ corresponding to the eigenvalue 1. This implies:

$$\begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{l}^T & 1 \end{bmatrix} \mathbf{H}_{12} = \begin{bmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{l}^T & 1 \end{bmatrix} \tag{B.11}$$

so that H can be decomposed in the form:

$$\mathbf{H}_{12} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{l}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{l}^T & 1 \end{bmatrix} \tag{B.12}$$

$$\mathbf{H}_{12} = \begin{bmatrix} \mathbf{A} + \mathbf{b}\mathbf{l}^T & \mathbf{b} \\ \mathbf{l}^T \left(\left(1 - \mathbf{l}^T\mathbf{b}\right) \mathbf{I} - \mathbf{A} \right) & 1 - \mathbf{l}^T\mathbf{b} \end{bmatrix} \tag{B.13}$$

Using the lemma 3, $\mathbf{A}$ can be decomposed into:

$$\mathbf{A} = \mathbf{L}^{-1}\mathbf{R}\mathbf{L} \tag{B.14}$$

and we can write b as:

$$\mathbf{b} = \mathbf{L}^{-1}\mathbf{t} \tag{B.15}$$

Thus,

$$\mathbf{H}_{12} = \begin{bmatrix} \mathbf{L}^{-1}\mathbf{R}\mathbf{L} + \mathbf{L}^{-1}\mathbf{t}\mathbf{l}^T & \mathbf{L}^{-1}\mathbf{t} \\ \mathbf{l}^T \left(\left(1 - \mathbf{l}^T\mathbf{L}^{-1}\mathbf{t}\right) \mathbf{I} - \mathbf{L}^{-1}\mathbf{R}\mathbf{L} \right) & 1 - \mathbf{l}^T\mathbf{L}^{-1}\mathbf{t} \end{bmatrix} \tag{B.16}$$

which can be factorized as:

$$\mathbf{H}_{12} = \begin{bmatrix} \mathbf{L}^{-1} & \mathbf{0} \\ -\mathbf{l}^T\mathbf{L}^{-1} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{L} & \mathbf{0} \\ \mathbf{l}^T & 1 \end{bmatrix} \tag{B.17}$$

We showed that this decomposition exists, but it is certainly not unique. If we count the parameters on each side, $\mathbf{H}_{12}$ has 16 parameters minus 3 because 2 eigenvalues must be 1 and the two others have one degree of freedom (the angle of the rotation, θ), which makes 13 parameters on the left side of equation B.3.2, and on the right side we have 6 parameters for the displacement and 9 for the lower triangular matrix which makes 15 parameters. Then the solution to this equation is not unique and the set of soloutions must be a manifold of dimension 2. One of the two remaining parameters is the scale factor on the Euclidean space, which can not be recovered because we have no length reference. We can eliminate it by

setting one of the parameters of the diagonal of **L** to 1 (they can never be zero because **L** is non singular).

It was shown clearly in [ZBR95] that the other parameter represents the incertitude on the choice of the absolute conic from **H**, because one displacement does not define that conic uniquely, so that we cannot recover the complete Euclidean structure from one displacement (i.e. two projective reconstructions). One way to deal with it would be to fix one of the intrinsinc parameters of the cameras as in [ZBR95], e.g. by saying that the x and y axis of the cameras are orthogonal, but since in our scheme the intrinsinc parameters do not appear clearly we could not use this. Another one is to simply use more than one displacement, as we demonstrate it in the next section.

B.5 Euclidean reconstruction of a whole object using stereo by correlation

To test this method, we took several stereoscopic pairs of images of an object using a stereo rig (Figure B.2). In this experiment, we used a mathematical object (called "cyclid") which equation is known, but the fact that we know its geometry was not used in the recovery of its Euclidean geometry. We

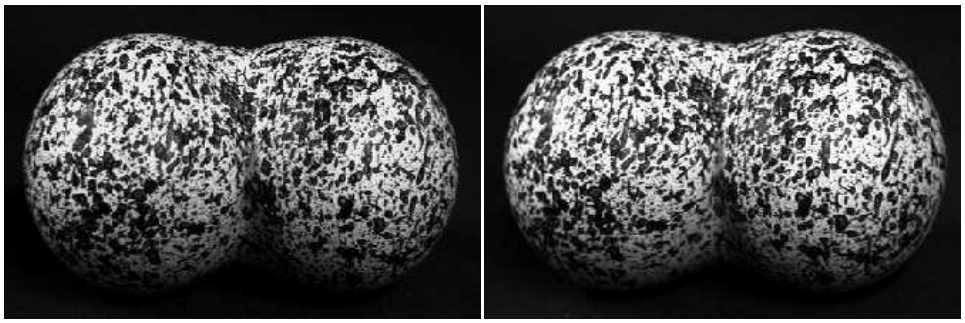


Figure B.2: One of the ten stereoscopic pairs used for the example

performed weak calibration [DZLF94] on these stereo pairs and computed disparity maps using stereo by correlation. We can show that a disparity map computed from a pair of rectified images can be considered as a 3-D projective reconstruction:

Proposition 5 *Let $d(x, y)$ be a disparity map, where x and y are rectified image coordinates. The projective points formed using by the rectified image coordinates as the two first coordinates, the disparity as the third coordinate, and 1 as the last coordinate, form a projective reconstruction, i.e. the 3-D Euclidean coordinates can be recovered by applying a 3-D collineation* **H** *to the points $(x, y, d(x, y), 1)$.*

Let $\mathbf{P}$ and $\mathbf{P}'$ be the projection matrices corresponding respectively to the rectified reference image and the other rectified image. Since the projection of a 3-D projective point M has the same y coordinate in both images, then only the first line of $\mathbf{P}$ and $\mathbf{P}'$ differ:

$$\mathbf{P} = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} \text{ and } \mathbf{P}' = \begin{bmatrix} {p'}_1 \\ p_2 \\ p_3 \end{bmatrix} \tag{B.18}$$

consequently

$$\mathbf{P}M \cong (x, y, 1) \text{ and } \mathbf{P}'M \cong (x + d(x, y), y, 1) \tag{B.19}$$

so that finally

$$M \cong \mathbf{H}(x, y, d(x, y), 1)^T \tag{B.20}$$

with

$$\mathbf{H} \cong \begin{bmatrix} p_1 \\ p_2 \\ {p'}_1 - p_1 \\ p_3 \end{bmatrix}^{-1} \tag{B.21}$$

Thus the disparity map $(x, y, d(x, y), 1)$ is a projective reconstruction.□

As we have seen before, we have 8 unknowns for the matrix $\mathbf{L}$ and 6 unknowns for each displacement, which makes $6 + 8(n - 1)$ unknowns, if n is the number of stereo pairs. We compute these parameters using a least-squares minimization technique: We match points between the rectified reference images of overlaping stereo pairs[3], and the error to be minimized is the squared distance between the points of reconstruction i transformed by the matrix $\mathbf{L}^{-1}\mathbf{D}_{ij}\mathbf{L}$ and the matched points of reconstruction j. This error measurement is done in (image+disparity) space, which is not the real 3-D Euclidean space, but since image space is almost Euclidean and disparity is bounded, it should work fine. This minimization is done in two steps: First, only the matches between rectonstructions i and $i + 1$ are considered, and the minimization is done over $\mathbf{L}$ and the $\mathbf{D}_{i,i+1}$, $1 \leq i < n$, which are represented as a rotation vector and a translation vector. In practice the error function associated with this minimization is well-conditioned, so that we get rather good estimates, whatever the initial point. Second, all the matches between different reconstructions are considered (especially the one between n and 1, which forces the surface to "fold over itself" [WB95]), and the minimization is done once again.

In fact we recovered the complete Euclidean geometry of our object. Figure B.3 shows the reconstruction from the first stereo pair, as seen when transformed by matrix $\mathbf{L}$, and Figures B.4 and B.5 show the complete reconstruction of the object from 10 stereo pairs, with lighting or with texture mapping.

[3]This process was done manually in our experiment but could be automated.

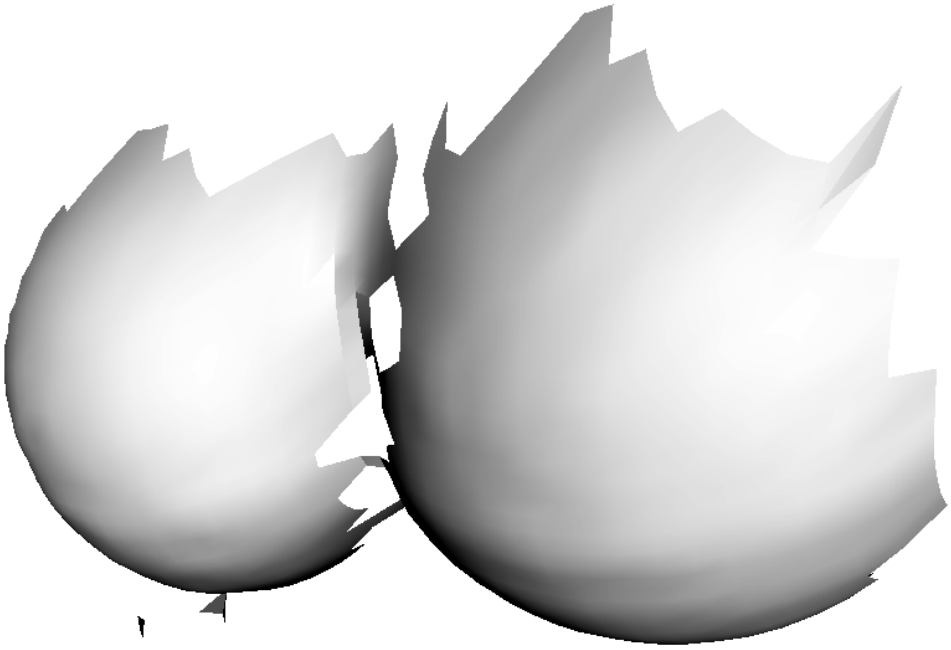


Figure B.3: The Euclidean reconstruction from the first stereo pair

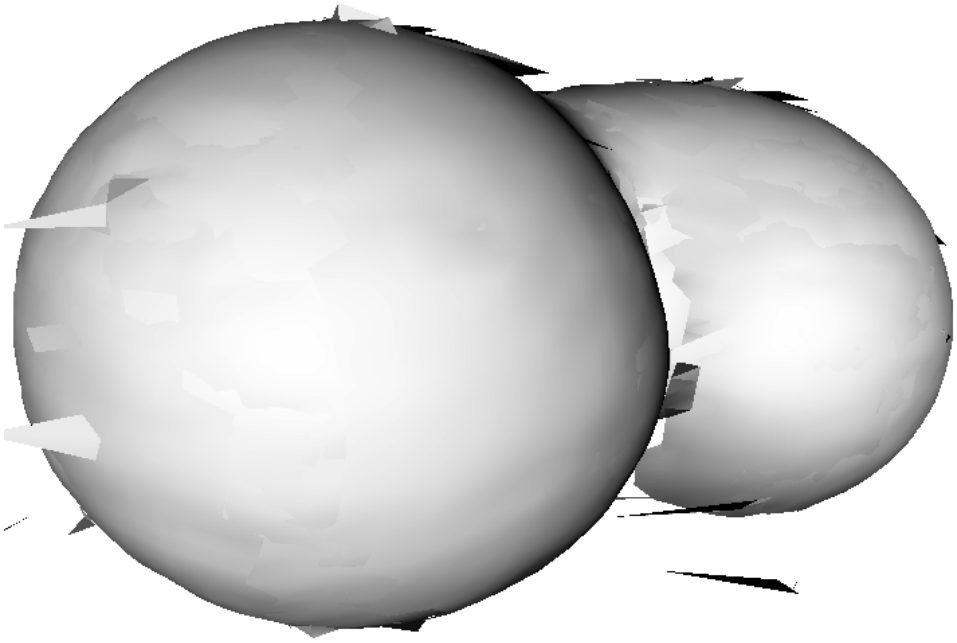


Figure B.4: The complete reconstruction of the object, rendered with lighting

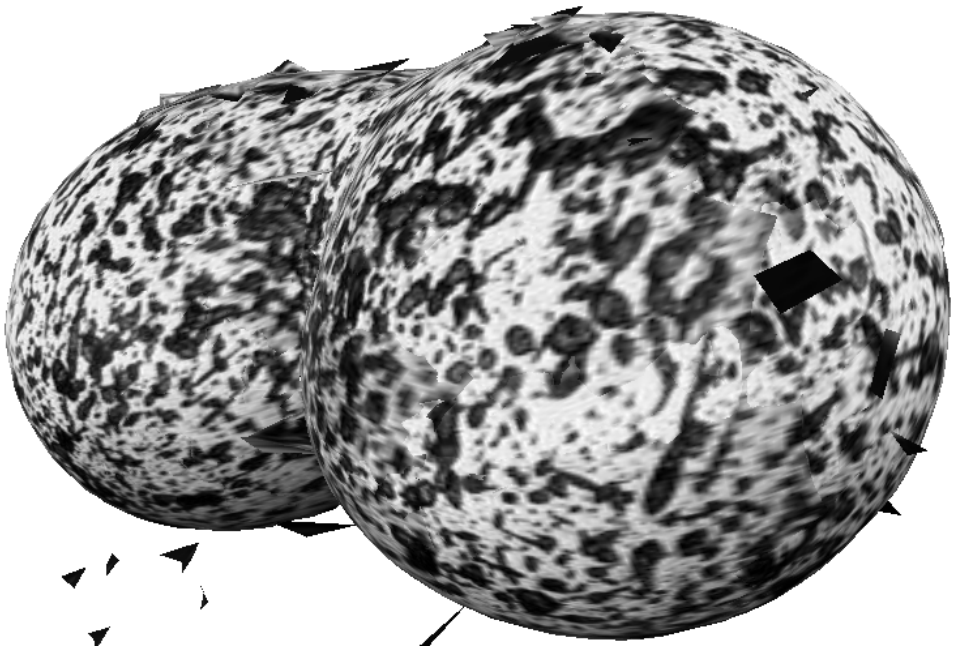


Figure B.5: The complete reconstruction of the object, rendered with lighting and texture mapping from the original images

B.6 Conclusion

In this paper we presented a method to recover partly or completely the Euclidean geometry using an uncalibrated stereo rig. All we need to do this is the fundamental matrix of the stereo rig, which can be calculated by a robust method like [DZLF94], and point matches between the different stereo pairs, which could be computed automatically. Using multiple stereo pairs, we increase the stability of the algorithm by adding more equations than unknowns. We presented results on a real object, which was fully reconstructed in Euclidean space using a few stereo pairs.

The possible applications of this method include the possibility to acquire easily 3-D objects using any set of uncalibrated stereo cameras, for example to modelize an object to be used in virtual reality, or autonomous robot navigation.

In the near future we plan to enhance the system in order to make it completely automatic: we must have a way to match points automatically (feature tracking would be a good starting point) a to perform fusion and simplification of the 3-D reconstruction once the registration is done. Furthermore, in order to test the accuracy of the Euclidean reconstruction, we can either compare the intrinsic and extrinsic parameters of the cameras computing a classical camera calibration method with those recovered using this method, or compare the 3-D reconstruction with a mathematical model

of the object (which is known in the example presented here).

Annexe C

A non-maxima suppression method for edge detection with sub-pixel accuracy *Détection de contours à une précision inférieure au pixel*[1]

We present here a two dimensional edge detector which gives the edge position in an image with a sub-pixel accuracy. The method presented here gives an excellent accuracy (the position bias mean is almost zero and the standard deviation is less than one tenth of a pixel) with a low computational cost, and its implementation is very simple since it is derivated from the well-known Non-Maxima Suppression method [Can83, Der87]. We also justify the method by showing that it gives the exact result in a theoretical one dimensional example. We have tested the accuracy and robustness of the edge extractor on several synthetic and real images and both qualitative and quantitative results are reported here.

Le détecteur de contour bidimensionnels que nous présentons dans cette annexe donne la position du contour dans une image à une précision inférieure au pixel. Cette méthode donne d'excellents résultats – la moyenne de l'erreur en position est pratiquement nulle et son écart type est d'à peine un dixième de pixel en l'absence de bruit – pour un coût en calcul à peine supérieur à une méthode classique, et de plus son implantation est très simple puisqu'elle est dérivée de la méthode classique de suppression des non-maxima locaux (NMS) [Can83, Der87]. Nous justifions également l'utilisation de cette méthode en montrant qu'elle donne le résultat exact

1. La méthode présentée ici a fait l'objet de publications en anglais [Dev93, Dev95].

dans le cas d'un contour monodimensionnel. Nous avons testé la précision et la robustesse de cet extracteur de contours sur plusieurs types d'images, synthétiques et réelles, et rapportons des résultats à la fois qualitatifs et quantitatifs.

C.1 Introduction

Edge detection is now taken for granted by most of the computer vision people, like many other basic tools of computer vision. Since a lot of work has already been done in early vision, people working on higher level features tend to use as input the result of classic low level algorithms and think this is the only information they can get from the image data. In the case of edge detection we used to work on information that was given to within about a pixel, and then had to do some kind of regularization process on it, like polygonal or spline approximation. Both introduce a certain quantity of error, but whereas the approximation error can be chosen, the error on the detected edge position is both fixed and not negligible, especially when one wants to compute differential properties of image curves like orientation, Euclidean curvature or even higher degree properties like affine or projective curvature. Some attempts were made in edge detection at a sub-pixel accuracy a few years ago, for example A. Huertas and G. Medioni [HM86] used a refinement of the zero-crossing of Laplacian but they did not give any results on the accuracy of the edge detection. A.J. Tababai and O.R. Mitchell [TM84] did some interesting work in the one-dimensional case which was extended to two-dimensional images and seemed to work properly. An edge relocation mechanism is also given in [VF92] but the required implementation is rather complex. The best results in terms of accuracy [NB86] were obtained at a high computational cost, because this involved local surface fitting on the intensity data, and the results are not better than our method. In general, the different methods that have been proposed often end up requiring regularization, excessive computer power, or both of them.

For these reasons we made a very simple enhancement of the classical local non-maxima suppression, that gives a much better estimate of the curve position (up to within a tenth of a pixel) or the curve orientation without regularization, and at a very low computational cost. Using this edge detector, we can also calculate higher order differential properties of curves with much less regularization than when using older methods. Besides, this method can be easily integrated in an existing vision system since it is based on a classical and widely-used method.

We present and interpret a wide variety of results to compare this method with existing edge detection methods. We also calculated in the most simple way the local edge orientation to show that the result of our method can

also be used to easily calculate differential properties of the edges.

Les techniques de détection de contours sont maintenant considérées comme acquises par une bonne partie de la communauté de la vision artificielle, de la même manière que les outils de base de vision. Puisqu'une somme considérable de travail a déjà été effectuée en vision de bas niveau ou vision préattentive, les chercheurs travaillant sur des primitives de haut niveau ont tendance à utiliser en entrée des résultats d'algorithmes « classiques » de vision préattentive et à croire que c'est là toute l'information qu'ils peuvent tirer des images. Dans le cas de la détection de contours, on a l'habitude d'utiliser des contours donnés à une précision d'à peu près un pixel pour ensuite les lisser, par exemple par une approximation polygonale ou par des splines. La détection de contours comme l'approximation introduisent chacun une certaine dose d'erreur, mais alors que l'erreur due a l'approximation peut en général être choisie, celle de l'extraction de contours est fixe et non négligeable, surtout lorsqu'on désire calculer des propriétés différentielles de courbes visibles dans l'images, comme l'orientation ou la courbure [Pap95]. De nombreuses méthodes ont déjà été proposéees en matière de détection de contours à une précision inférieure au pixel, par exemple Huertas et Médioni [HM86] ont utilisé une amélioration de la méthode de détection des contours comme les passages par zéro du laplacien, mais n'ont pas donné de résultats quant à la précision de la détection des contours. Tababai et Mitchell [TM84] ont effectué un travail intéressant dans le cas d'images monodimensionnelles, dont l'extension au cas d'images bidimensionnelles semble bien fonctionner. Une méthode d'affinage de position de contours est égallement donnée dans [VF92] mais elle reste plutôt complexe. Les meilleurs résultats en termes de précision [NB86] ont été obtenus en contrepartie d'un coût en calculs important, la méthode mettant en oeuvre une approximation locale de l'intensité par une surface, et les résultats ne sont pas meilleurs que ceux présentés ici. En général les différentes techniques proposées reviennent à utiliser de la régularisation, une puissance calculatoire importante, et parfois même les deux.

C'est pour ces raisons que nous avons développé cette simple amélioration de la méthode de supression des non-maxima locaux, qui donne une bien meilleure estimée de la position du contour (jusqu'à un dixième de pixel de précision) ou de son orientation, sans régularisation et pour un faible coût en calculs. Grace à cette méthode nous pouvons aussi nous permettre de calculer des propriétés différentielles de courbes de degré plus élevé en régularisant moins les données qu'avec les anciennes méthodes. De plus, étant basée sur une méthode classique et largement utilisée, son intégration dans une chaîne de vision est extrèmement simple.

Nous présentons et interprétons des résultats variés afin de pouvoir comparer cette méthode avec d'autres détecteurs de contours. Nous présentons également des résultats de calcul d'orientation de contours de la manière la

plus simple possible, pour montrer qu'on peut également facilement calculer des propriétés différentielles de courbes à partir des résultats de cette extraction de contours.

C.2 On edge extraction *Détection de contours*

La détection de contours proprement dite se décompose en deux étapes. La première, classique, consiste à détecter les pixels par où passer les contours et à les chaîner. La seconde, qui représemte l'essentiel de notre contribution, consiste à affiner la position des points de contours détectés pour obtenir une précision meilleure que le pixel.

C.2.1 The NMS method *Suppression des non-maxima locaux*

This method is based on one of the two methods commonly used for edge detection, the suppression of the local non-maxima of the magnitude of the gradient of image intensity in the direction of this gradient [Fau93] (also called NMS), the other one being to consider edges as the zero-crossings of the Laplacian of image intensity [Har84, HS92]. NMS consists of:

1. Let a point (x, y), where x and y are integers and $I(x, y)$ the intensity of pixel (x, y).

2. Calculate the gradient of image intensity and its magnitude in (x, y).

3. Estimate the magnitude of the gradient along the direction of the gradient in some neighborhood around (x, y).

4. If (x, y) is not a local maximum of the magnitude of the gradient along the direction of the gradient then it is not an edge point.

Usually for step 4 the neighborhood is taken to be 3×3 and the values of the magnitude are linearly interpolated between the closest points in the neighborhood, e.g. in Figure C.1 the value at C is interpolated between the values at A_7 and A_8 and the values at B between those at A_3 and A_4. We have also tried to use *quadratic* interpolation to compute these (the value at A would be interpolated between those at A_7, A_8, and A_1 as in Figure C.2) and compared the results with the linear interpolation. After this edge detection process one usually does hysteresis thresholding [Can83] on the gradient norm and linking to get chains of pixels.

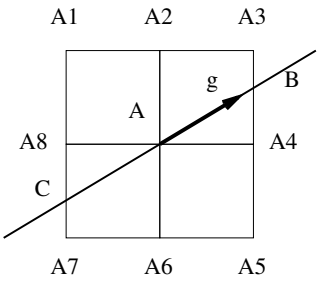


Figure C.1: Checking whether pixel A is a local maximum of the magnitude of the gradient in the direction of the gradient is done by interpolating the gradient magnitude at B and C.

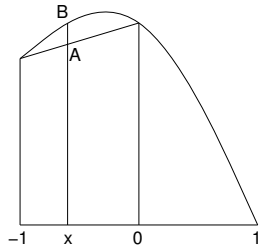


Figure C.2: Examples of linear (A) and quadratic (B) interpolation between three values at -1, 0, and 1.

C.2.2 The sub-pixel approximation *Amélioration de la précision*

Our main improvement of the method is very simple and consists of only adding this single step to the NMS process:

1° If (x, y) is a local maximum then estimate the position of the edge point in the direction of the gradient as the maximum of an interpolation on the values of gradient norm at (x, y) and the neighboring points.

This is the principle, but we still have to find the interpolation between gradient norm values we should apply to find the best position of the edge. We will consider a simple quadratic interpolation of the values of the gradient norm between the 3 values we have in the gradient direction. One could also try to locally fit a simple surface (e.g. bi-quadratic) on the neighborhood of the considered point but we want to keep the computations as simple as possible so that the implementation is fast and easy.

The choice of the quadratic interpolation to find the maximum can be justified because it gives the exact result in the one-dimensional case. Let $L = \{l(i)|i \in \aleph\}$ an infinite line of pixels with a step edge at position $\alpha - \frac{1}{2}$,

coordinate 0 corresponding to the middle of pixel $l(0)$. The continuous intensity function for this step edge, as shown in Figure C.3, is:

$$I(t) = \begin{cases} 0 & \text{if } t > \alpha - \frac{1}{2} \\ 1 & \text{otherwise} \end{cases}$$

so that

$$\begin{aligned} l(i) &= \int_{-\frac{1}{2}}^{\frac{1}{2}} I(t)dt \\ l(-\infty) = \cdots = l(-2) &= l(-1) = 1 \\ l(0) &= \alpha, 0 \leq \alpha \leq 1 \\ l(1) = l(2) &= \cdots = l(+\infty) = 0 \end{aligned}$$

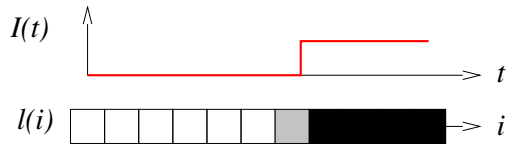


Figure C.3: A one-dimensional step edge and the corresponding gray-level pixel line.

Let ∇ be a general derivation operator. It can be a finite differences operator, or any gradient filtering operator, the only constraint being that ∇ is antisymmetric. It can be written:

$$\begin{aligned} \nabla l(i) &= \sum_{k=1}^{+\infty} g_k(l(i+k) - l(i-k)) \\ &= \cdots - g_2 l(i-2) - g_1 l(i-1) + g_1 l(i+1) + g_2 l(i+2) + \cdots \end{aligned} \tag{C.1}$$

Let us apply this general derivation operator on locations $-1, 0, 1$:

$$\begin{aligned} \nabla l(-1) = &\quad \cdots - g_2 - g_1 + g_1\,\alpha &&= C - (1-\alpha)\,g_1 \\ \nabla l(0) = &\quad \cdots - g_2 - g_1 &&= C - g_1 \\ \nabla l(1) = &\quad \cdots - g_2 - g_1\,\alpha &&= C - \alpha\,g_1 \end{aligned}$$

where

$$C = -\sum_{k=2}^{\infty} g_k$$

Since shifting and rescaling the values of the three points used to find a maximum of the quadratic interpolation do not affect the position of this

maximum, we can simplify things by using $C = 0$ and $g_1 = 1$:

$$\begin{aligned} a = &\quad |\nabla l(-1)| &&= 1 - \alpha \\ b = &\quad |\nabla l(0)| &&= 1 \\ c = &\quad |\nabla l(1)| &&= \alpha \end{aligned}$$

Considering that the *center* of the pixels correspond to integer x coordinates, one can find that the x position of the maximum of the parabola passing through $(-1, a)$, $(0, b)$, and $(1, c)$ is:

$$m = \frac{a - c}{2(a - 2b + c)}. \tag{C.2}$$

It can be easily seen that when $b \geq a$ and $b \geq c$ this value is bounded:

$$-0.5 \leq m \leq 0.5$$

Since integer coordinates correspond to pixel centers and pixel width is 1, this means that the sub-pixel position of an edge point will always be *inside* the pixel.

That gives in this case

$$m = \alpha - \frac{1}{2}$$

which is exactly the theoretical position of the edge! Equation C.2.2 shows that the edge position is invariant to additive and multiplicative changes in the data. Moreover, this result can be extended to any kind of smooth edge, since a smooth edge is the result of the convolution of a step edge with a symmetric blurring operator s. The blurring operator s is symmetric so the action of the derivation on the ramp edge is the same as the action of the convolution of this operator with the blurring operator, which can be considered as another derivating operator, on the corresponding step edge:

$$\nabla(s \circ l)(i) = (\nabla \circ s)l(i) = \nabla' l(i)$$

We should be careful about one thing with the previous computations: we worked with the gradient norm, whereas most people might want to use the squared gradient norm for NMS because it requires less calculations and when the squared gradient norm is a local maximum the gradient norm is one too. But the quadratic interpolation gives a different result, and in the previous case we would find the maximum at:

$$m = \frac{1 - 2\alpha}{4\alpha^2 - 4\alpha - 2}$$

which introduces a bias on the edge position. The maximum of the bias is $\delta m = 0.073$ pixels at $\alpha = 0.19$ and its standard deviation is $\sigma(\delta m) = 0.052$, that is more than $\frac{1}{20}$ pixel (the same magnitude order as the precision we

would like to get). In conclusion, using the squared gradient norm may reduce significantly the precision of the edge extraction.

The comparison between the detected and the theoretical edge position was not done in the two dimensional case because it involves too many parameters (including the position and angle of the edge, the coefficients of a generic two-dimensional derivating filter, and the way the values a and c are calculated). Before seeing some results let us see how this edge detector can be used in a classic image processing chain.

C.3 Using the proposed improvements *Utilisation de la méthode*

The main advantage of this new method over the sub-pixel edge detectors that have already been done is that its cost in terms of calculation is almost nothing (3 additions and 3 multiplications at each detected edge point if we use the quadratic approximation, which make 6 floating-point operations, not including the computation of the gradient) and it can be easily integrated in a simple and well-known algorithm.

The new problem that can appear with a sub-pixel edge detector is that because the point coordinates are not integer we may not be able to apply common techniques such as hysteresis thresholding or linking. Hopefully we have solved this problem the simplest way we could: with this edge detector an edge point has non integer coordinates but can still be attached to the point from which it was calculated, which has integer coordinates. Thus we perform the common operations on the edge points as if they had integer coordinates, like hysteresis thresholding [Can83] using the gradient norm and edge pixels linking, but can use their sub-pixel approximation whenever we want it, e.g. when we need the precise position or want to use some differential properties of the edge.

The solution we used is to save the sub-pixel position of each edge point separately when the edge detection process is finished, as $(\delta x(x, y), \delta y(x, y))$ pairs where x and y are integers and $(\delta x, \delta y) \in [-0.5, 0.5] \times [-0.5, 0.5]$, to do the other edge processes on the other data that has been saved (the edge position up to within a pixel and the gradient norm), and to use the sub-pixel edge position when we need it later in the processing.

In conclusion of this section, we say that one can use this new approach to greatly improve already existing algorithms with only minor modifications in the image processing chain.

C.4 Results
Résultats

C.4.1 The test data
Données de test

It seems to us that the qualities that should have a good edge detector are:

1. A good estimate on the position of the edge, whatever the edge position, orientation and curvature.
2. Good differential properties of the edge data, i.e. the edge orientation should not be biased and have a small variance, and higher order differential properties should be calculated accurately with not too much regularization.
3. All these properties should be robust to noise.

To verify these we used test data consisting of two series of images, each one consisting of a single edge:

- A collection of lines, with light gray on one side and dark gray on the other side, with a wide variety of orientations.
- A collection of filled circles with radiuses going from 3 to 100 pixels.

These are 128×128 images generated with anti-aliasing, the edge contrast is given (we chose 100 for our experiments), and uncorrelated Gaussian noise can be added on the image intensity data. The anti-aliasing can be justified by the fact that common image sensors (e.g. CCD) give the integral of light intensity over each pixel. Two smaller sample images are shown Figure C.4 to demonstrate what the anti-aliased and noisy images look like. We calculated the gradient of image intensity using a Deriche fourth order Gaussian recursive filter [Der93, Der87]. Other gradient filters were also used [Can86, Der90, SC92] and gave comparable results.

C.4.2 The different edge detection methods
Les différentes méthodes testées

We tested many configurations of the edge detector on these images including:

- The classic NMS method on squared gradient norm using linear interpolation with no sub-pixel approximation.
- Our method on either *real* or *squared* gradient norm using either *linear* or *quadratic* interpolation to find the values of the norm in the direction of the gradient.

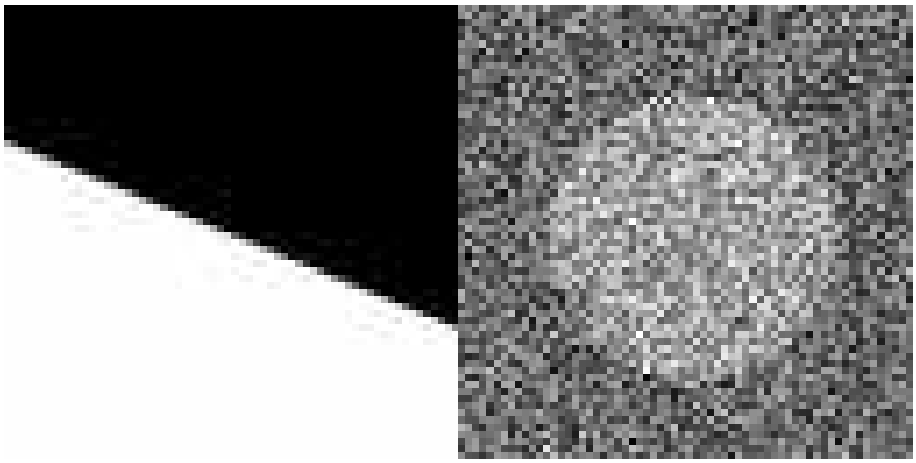


Figure C.4: Two 64×64 images of the same kind as the ones that were used for our experiments: A line at 22.5^0 with no noise and a circle of radius 20 with Gaussian additive noise. The noise standard deviation is 50 which corresponds to a signal to noise ratio (SNR) of 0dB.

Using the result of edge detection, we calculated:

- the position of the calculated pixels with respect to the theoretical edge, and its mean and standard deviation over the edge.
- the difference between the theoretical edge orientation and the orientation of the line joining two *consecutive* edge pixels (this is a very local measure since the distance between two consecutive edge pixels is about one pixel).

For each of these measures and for a given configuration of the edge detector we calculated its mean, standard deviation, and maximum for different edge orientations, calculated over 100 images, and in each image over 100 consecutive edge pixels.

C.4.3 Edge position
Position du contour

The mean and standard deviation of the position bias for the different sub-pixel edge detection methods are presented in Figure C.5. The mean of the position bias is close to zero for straight edges (less than $\frac{1}{200}$ pixel in any case). The Gaussian derivative filter used for preprocessing had a σ of 1.5, this is a typical value to use with real images. For the classic NMS method, the maximum standard deviation of the position bias is 0.40 pixels at 44^0, far over the sub-pixel methods. We can see with this figure that the best method for sub-pixel edge detection is to use the real gradient norm as input and to

estimate the gradient norm in the direction of the gradient using quadratic interpolation between the three neighboring points in the direction of the gradient. For a smaller computational cost, using the squared gradient norm with quadratic interpolation or the real norm with linear interpolation give rather good estimates.

We tested the robustness of the best method (real norm and quadratic approximation) by applying Gaussian noise to the image intensity for 512×512 images. The standard deviation of the edge position bias was calculated for different values of the intensity noise standard deviation and the Gaussian derivative filter standard deviation σ. As shown in Figure C.6, $\sigma = 1.5$ gives the best results when there is not much noise but when the noise standard deviation is more than 40 the edge is almost always cut, so that bigger values of σ should be used. The same thing happens for $\sigma = 2.5$ when noise is more than 65. All these results are comparable in accuracy with those of Nalwa and Binford (compare those results with figures 10 and 11 of [NB86]), though this method is a lot faster and simpler.

C.4.4 Edge orientation
Orientation du contour

To prove that this edge detector gives excellent results we calculated the edge orientation in a very simple way, as the orientation of the line joining two consecutive edge points. This gives good results under zero noise conditions (Figure C.7), but when there is noise in image intensity or when one needs more precision, smoothing the image intensity or using regularization gives better results. Figure C.7 also shows that the edge orientation mean is slightly biased when using this method. This bias is due to the way edge pixels are distributed along the edge: the distance between two edge pixels may vary between 0 and 1 pixel, and this distance is correlated with the orientation of the line joining these two pixels. These figures can be compared with figures 8.12 and 8.13 in [HS92].

To calculate a better value of the edge orientation we could use some kind of regularization on the data, like for example calculate a least squares approximation of n consecutive points by a line or a higher order curve. This would reduce significantly both the standard deviation of the measures and the angle bias we noticed before, but the local character of the measure would be lost if the data is too much regularized.

Since the computation of edge orientation was just used as a simple example to illustrate the accuracy of this edge detection process, we will not develop further the discussion on how to compute edge orientation in a good way.

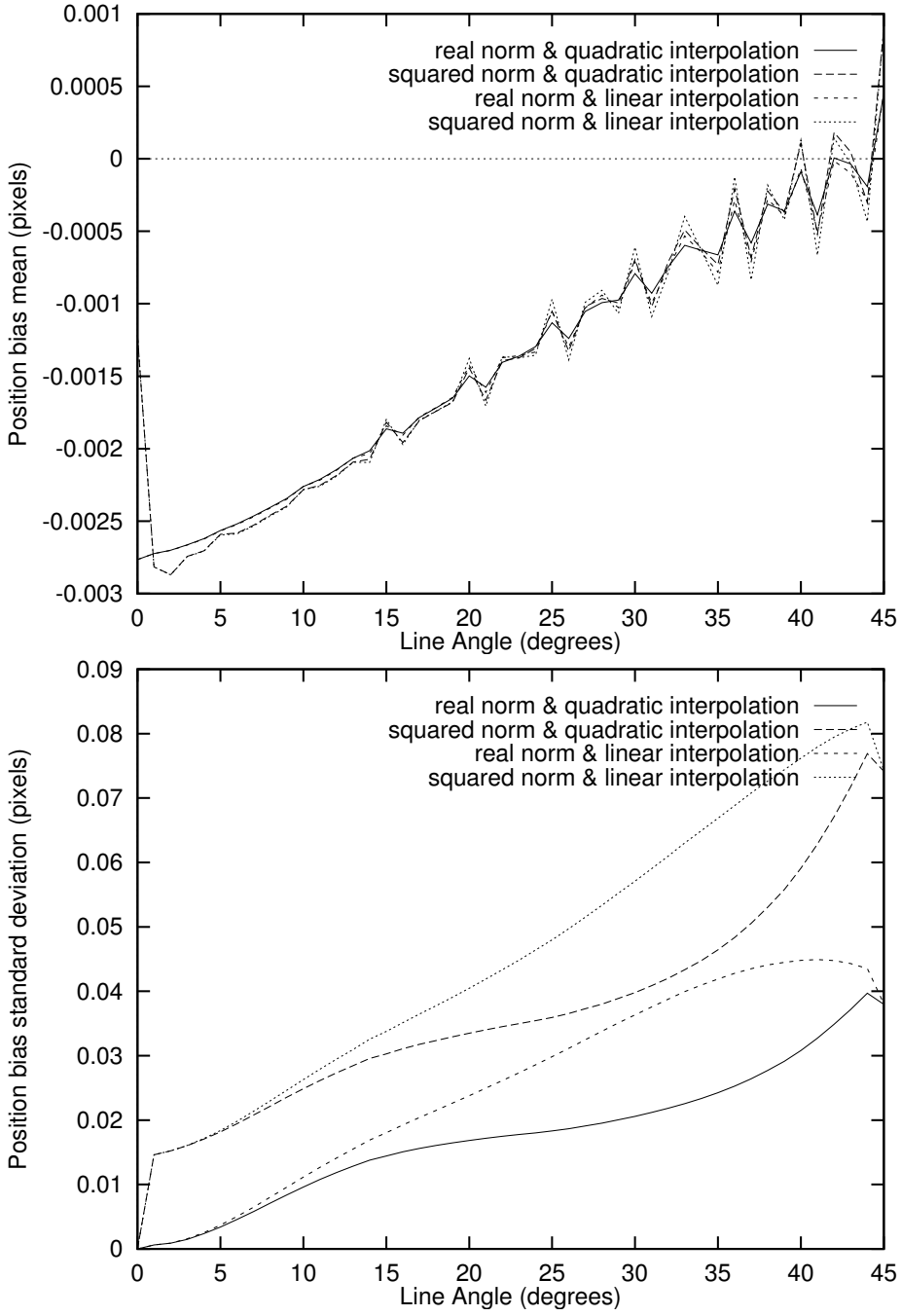


Figure C.5: Position bias mean and standard deviation as a function of edge orientation with $\sigma = 1.5$, under zero-noise conditions. Only the standard deviation is significant, since the mean is almost zero.

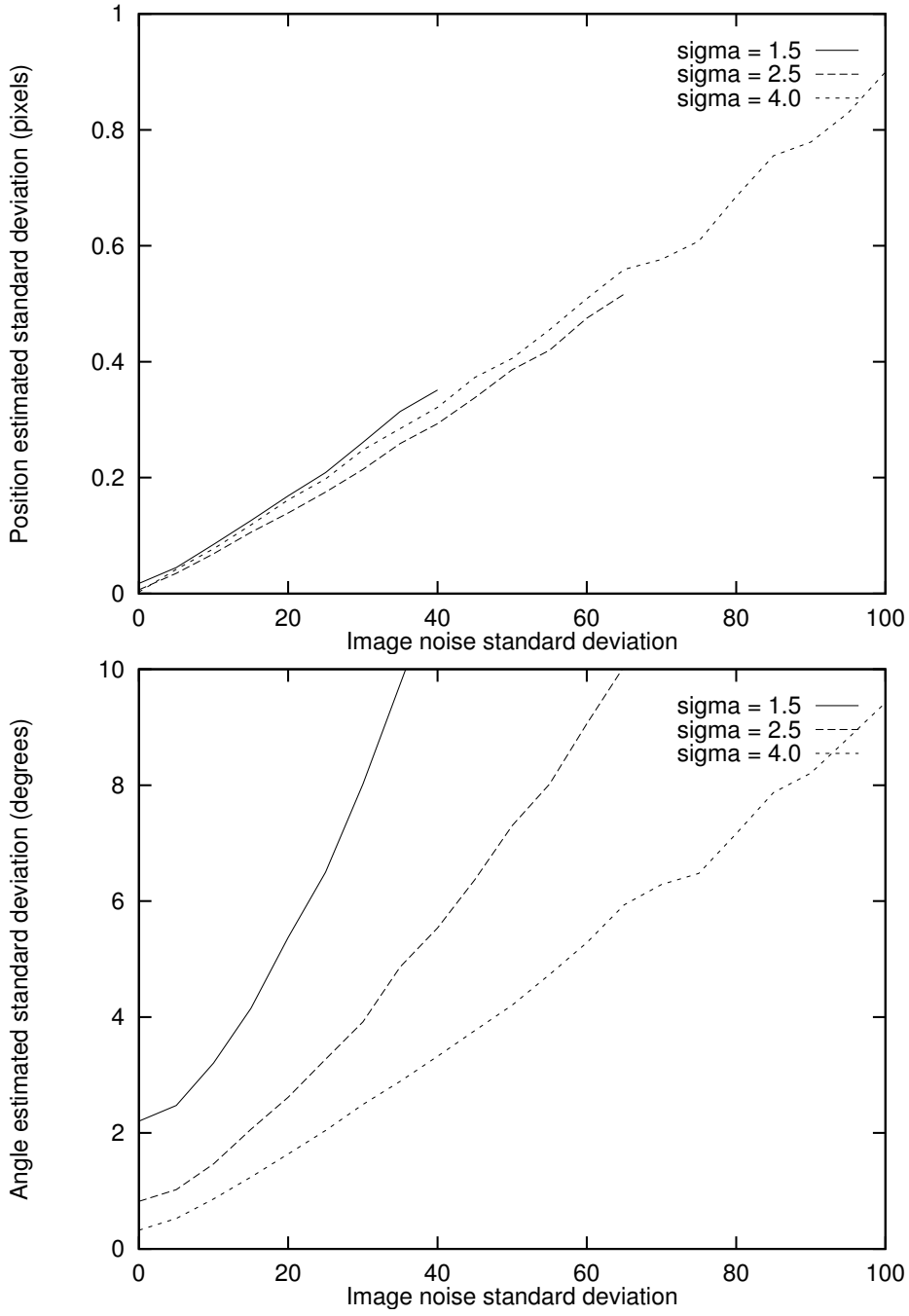


Figure C.6: Position and angle bias standard deviation as a function of image noise standard deviation for different values of the smoothing parameter σ. Edge Orientation is 22.5^0 and edge contrast is 100.

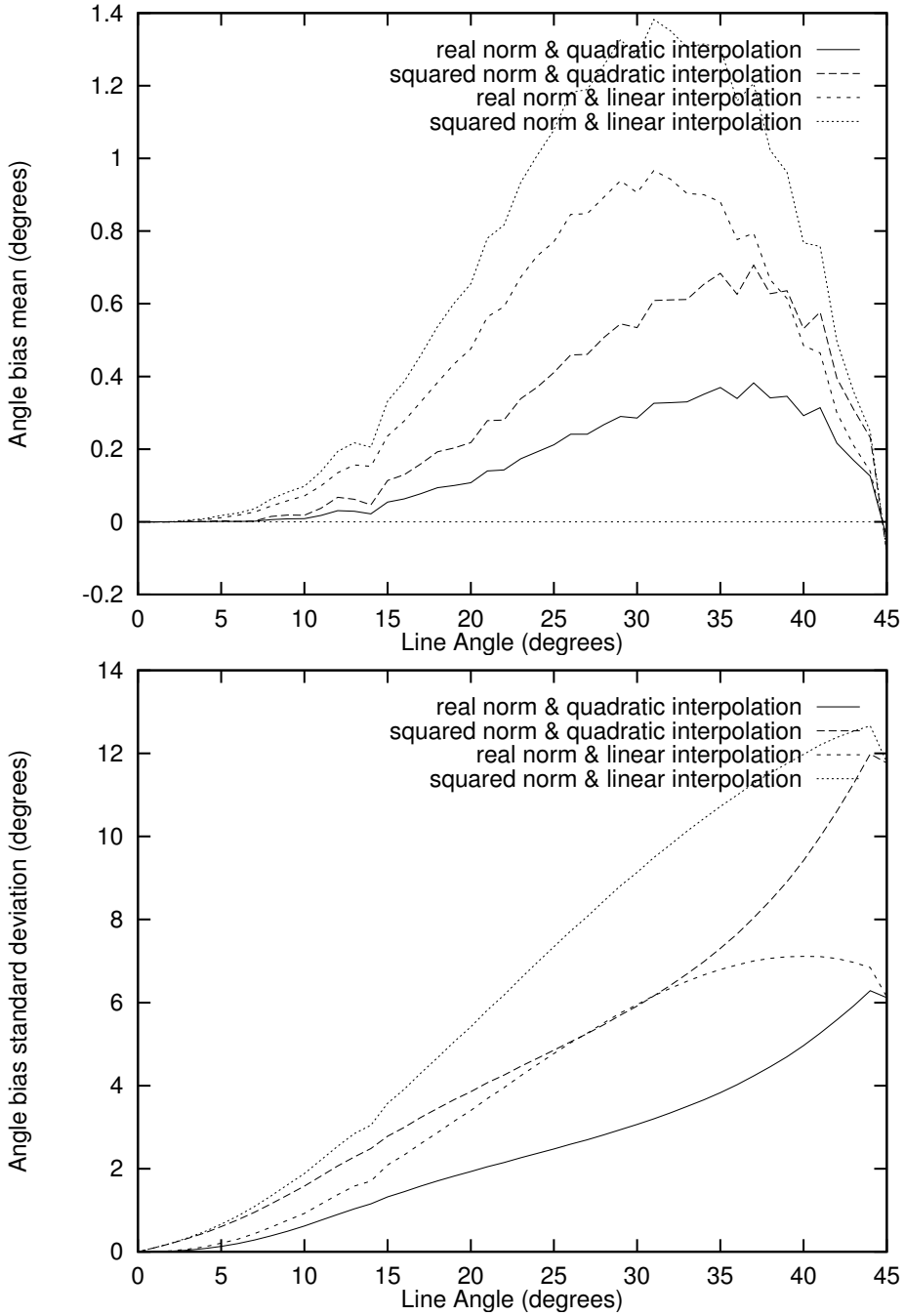


Figure C.7: Angle bias mean and standard deviation as a function of edge orientation with $\sigma = 1.5$. The angle is calculated using the line joining two consecutive edge pixels (the distance between two consecutive edge pixels is less than 1 pixel).

C.4.5 Other results
Autres résultats

For the best configuration of the edge detection process (i.e. use the real norm and quadratic interpolation between neighbors for the NMS), we calculated the edge position bias and orientation bias mean and standard deviation (Figure C.8) on the circle images. We can see that the edge position is always found inside the circle. This behavior is normal and not due to the edge detection method, as it comes from the Gaussian filtering, as shown in [Ber84], and any smoothing operator will give the same kind of results. We compared the measured displacement with the theoretical displacement [Ber84] and the two curves fit perfectly. The only way to avoid this error is to have a model that not only takes into account the image intensity smoothing [NB86], but also the local curvature or a more general shape (like a corner shape in [DB93a]).

We can also notice that the angle bias mean and standard deviation are lower than those that were found for straight edges. This is maybe due to the fact that straight edges generate more "bad" situations than curves.

We present results on part of a real aerial image (Figure C.9) of both the classic method and our method. We can easily see that the shape of curved features that was almost lost with the classic method is still present with our method. The brain image (Figure C.10) is another example of the precision of our edge detection method. These images were taken from the database created at the SPIE Conference "Application of Artificial Intelligence X: Machine Vision and Robotics".

C.5 Conclusion

In this section we presented an enhancement of the Non-Maxima Suppression edge detection method which gives us the edge position at a sub-pixel accuracy. Since this method is very simple and costs only a few additions and multiplications per detected edge pixel, it can be easily implemented and incorporated in a real-time vision system, and it should be used to increase the precision and reliability of vision algorithms that use edges as input.

The result we got on a wide variety of synthetic and real images are very promising since they show that the precision of this edge detector is less than $\frac{1}{10}$ of a pixel. We computed the accuracy of this edge detection method in an objective manner, so that the results can be easily compared with other algorithms.

One advantage over existing edge refinement methods [HM86, TM84, VF92, NB86] is its very low computational cost, since we only do a one-dimensional interpolation where most other methods work on two-dimensional data. Besides, other methods usually try to find a better esti-

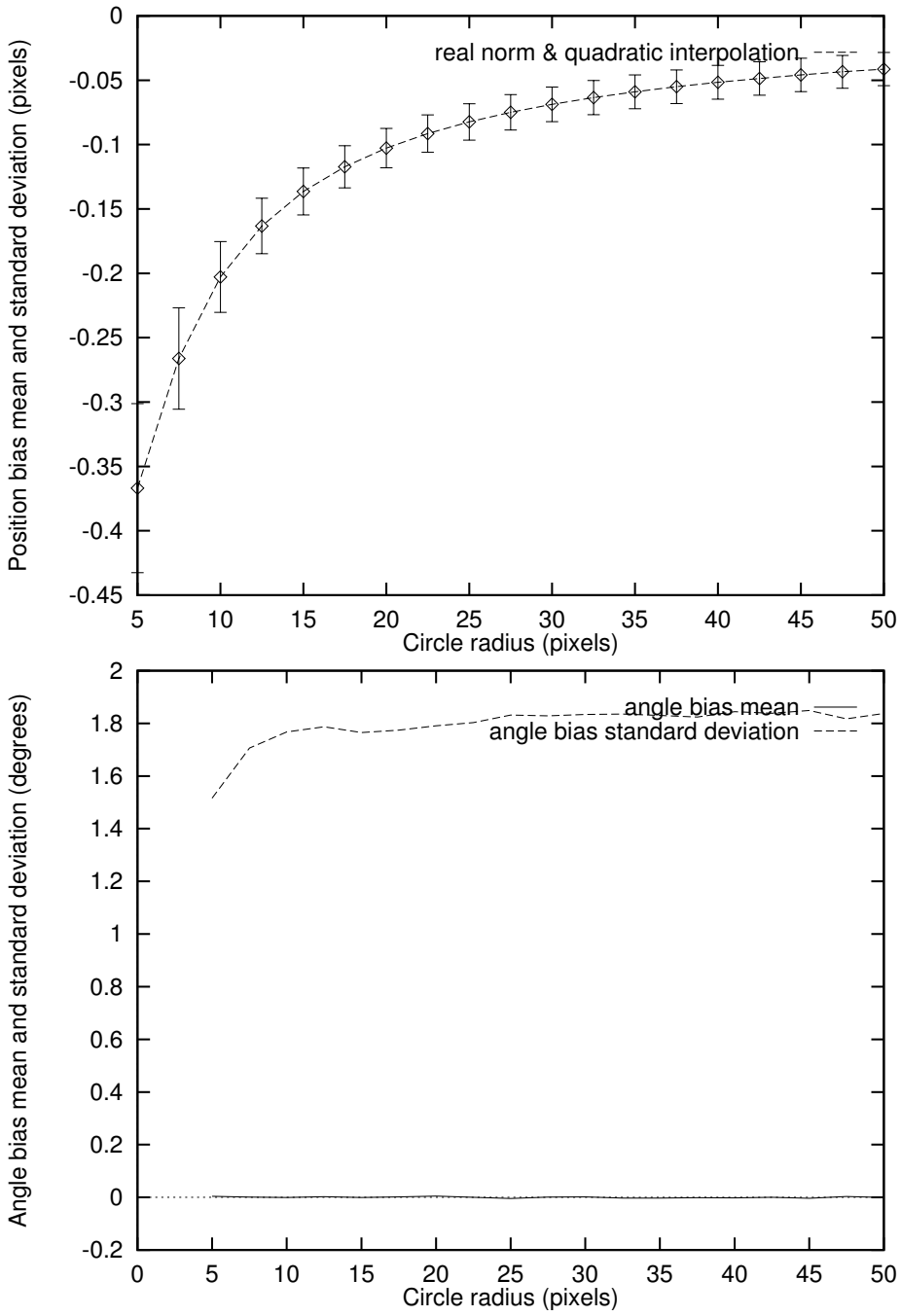


Figure C.8: Measured edge displacement as a function of circle radius, and angle bias mean and standard deviation. We used a Gaussian filter with $\sigma = 2.0$ to calculate derivatives. The displacement in the case of curved edges is not due to the edge detection method (it is due to image smoothing), and corresponds to what is predicted by the theory [Ber84].

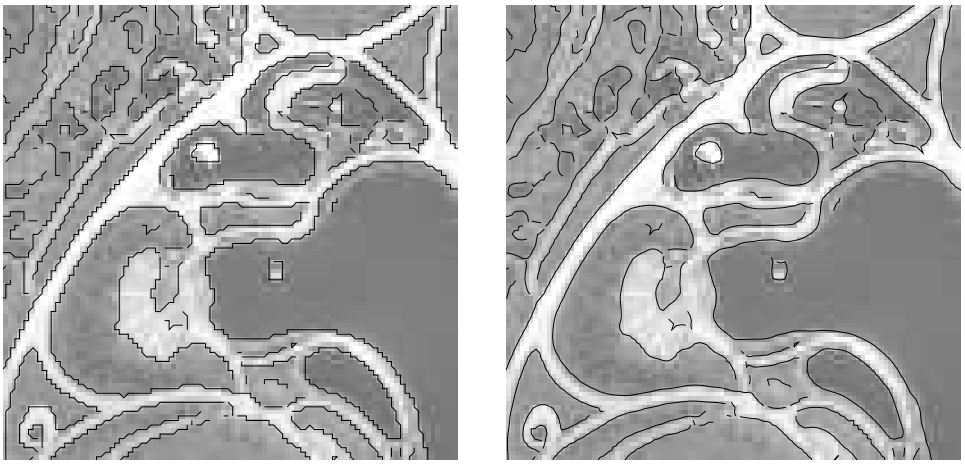


Figure C.9: Results of the classical NMS algorithm (left) and of the sub-pixel approximation (right) on a 100x100 region of an aerial image. The gradient was calculated using a second order Deriche recursive filter with $\alpha = 1.2$.

mate of the edge position using edge pixels that are given up to within a pixel, thus using regularization, whereas this method gets a better estimate of the edge position directly from the image intensity data.

In the future, we plan to use this edge detector to calculate some higher degree differential properties of the curves such as Euclidean, affine, or even projective curvature, and we will also use it to enhance the precision of existing and future algorithms.

Dans cette annexe nous avons présenté une amélioration de la méthode de détection de contours par suppression des non-maxima locaux (NMS) qui nous donne la position des contours à une précision inférieure au pixel. Cette amélioration étant très simple et ne coûtant que quelques additions et multiplication supplémentaires par point de contour détecté, elle peut être facilement implantée et incorporée dans un système de vision temps-réel, et elle augmente alors la précision et la fiabilité des algorithmes de vision qui utilisent les contours en entrée.

Un avantage sur les méthodes existantes de rafinement des contours [HM86, TM84, VF92, NB86] est son très faible coût en calcul, puisqu'on fait seulement une interpolation monodimensionelle là où la plupart des méthodes travaillent sur des données à deux dimensions. De plus, les autres méthodes essayent en général de trouver une meilleure estimée de la position du contour en utilisant des points détectés à un pixel près et doivent donc lisser les données, alors que cette méthode trouve la position du contour directement à partir des données d'intensité. Cette méthode a également été utilisée pour calculer des propriétés différentielles de degré élevé de courbes [Pap95] (courbures euclidienne, affine, ou projective).

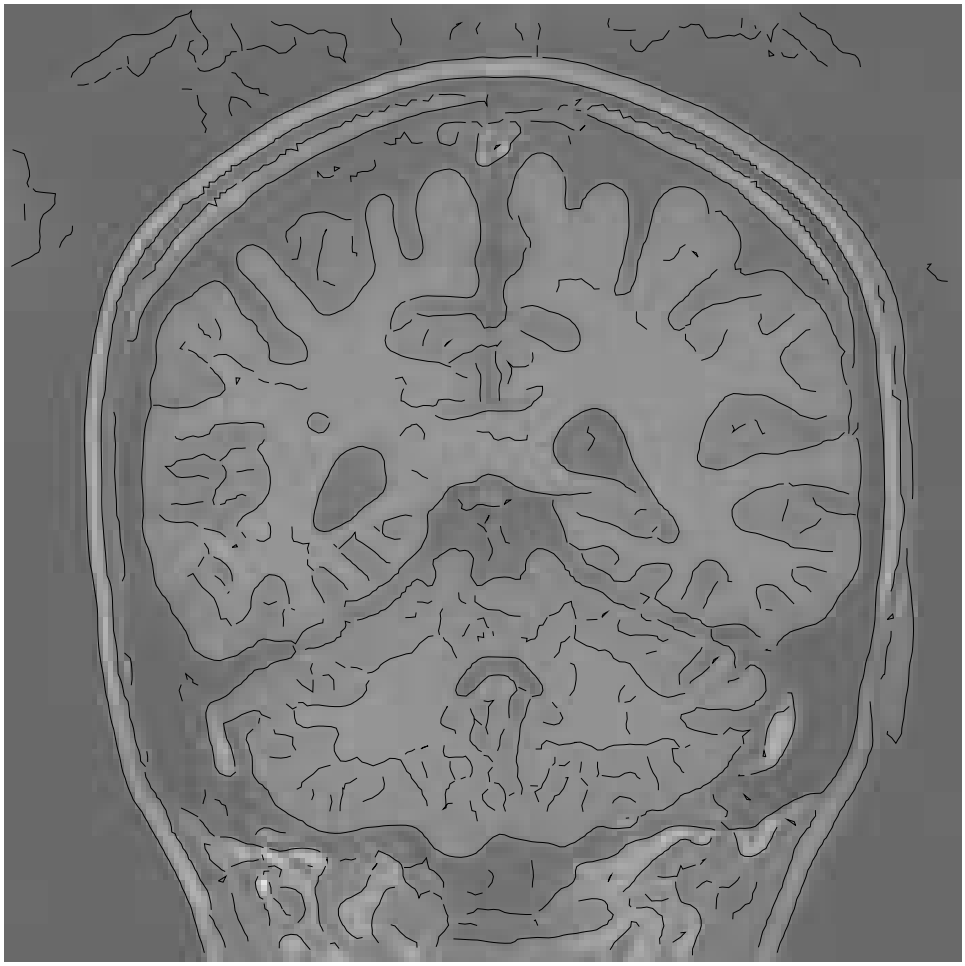


Figure C.10: Result of the edge detection at sub-pixel accuracy on a 175×175 pixel magnetic resonance image of a human brain.

Annexe D

La régression bilinéaire

DANS CETTE PARTIE nous allons décrire la méthode permettant de calculer une régression bilinéaire, c'est-à-dire comment calculer l'approximation d'en ensemble de N points (x_i,y_i,z_i) par un plan d'équation

$$z(x,y) = ax + by + c \tag{D.1}$$

qui minimise la fonction de coût suivant, dite du « chi-2 » :

$$\chi^2(a,b,c) = \sum_{i=1}^{N} \left(\frac{z_i - ax_i - by_i - c}{\sigma_i} \right)^2 \tag{D.2}$$

où σ_i est l'incertitude associée à la coordonnée z_i du point i.

D.1 Minimisation du chi-2

Pour déterminer a, b, et c, on minimise l'équation D. Lorsque le minimum est atteint, les dérivées partielles de $\chi^2(a,b,c)$ s'annulent :

$$0 = \frac{\partial \chi^2}{\partial a} = -2 \sum_{i=1}^{N} \frac{x_i \left(z_i - ax_i - by_i - c \right)}{\sigma_i^2} \tag{D.3}$$

$$0 = \frac{\partial \chi^2}{\partial b} = -2 \sum_{i=1}^{N} \frac{y_i \left(z_i - ax_i - by_i - c \right)}{\sigma_i^2} \tag{D.4}$$

$$0 = \frac{\partial \chi^2}{\partial c} = -2 \sum_{i=1}^{N} \frac{z_i - ax_i - by_i - c}{\sigma_i^2} \tag{D.5}$$

Posons :

$$S = \sum_{i=1}^{N} \frac{1}{\sigma_i^2} \tag{D.6}$$

$$S_x = \sum_{i=1}^{N} \frac{x_i}{\sigma_i^2} \qquad S_y = \sum_{i=1}^{N} \frac{y_i}{\sigma_i^2} \qquad S_z = \sum_{i=1}^{N} \frac{z_i}{\sigma_i^2} \tag{D.7}$$

$$S_{xx} = \sum_{i=1}^{N} \frac{x_i^2}{\sigma_i^2} \qquad S_{xy} = \sum_{i=1}^{N} \frac{x_i y_i}{\sigma_i^2} \qquad S_{xz} = \sum_{i=1}^{N} \frac{x_i z_i}{\sigma_i^2} \tag{D.8}$$

$$S_{yy} = \sum_{i=1}^{N} \frac{y_i^2}{\sigma_i^2} \qquad S_{yz} = \sum_{i=1}^{N} \frac{y_i z_i}{\sigma_i^2} \qquad S_{zz} = \sum_{i=1}^{N} \frac{z_i^2}{\sigma_i^2} \tag{D.9}$$

Les équations D.5 deviennent alors :

$$aS_{xx} + bS_{xy} + cS_x = S_{xz} \tag{D.10}$$
$$aS_{xy} + bS_{yy} + cS_y = S_{yz} \tag{D.11}$$
$$aS_x + bS_y + cS = Sz \tag{D.12}$$

D.2 Solution de la régression

La solution de ces équations est :

$$a = \frac{S_{xz}\, a_1 + S_{yz}\, a_2 + S_z\, a_3}{\Delta} \tag{D.13}$$
$$b = \frac{S_{xz}\, a_2 + S_{yz}\, b_2 + S_z\, b_3}{\Delta} \tag{D.14}$$
$$c = \frac{S_{xz}\, a_3 + S_{yz}\, b_3 + S_z\, c_3}{\Delta} \tag{D.15}$$

avec :

$$a_1 = S\, S_{yy} - S_y^2 \tag{D.16}$$
$$a_2 = S_x\, S_y - S\, S_{xy} \tag{D.17}$$
$$a_3 = S_y\, S_{xy} - S_x\, S_{yy} \tag{D.18}$$
$$b_2 = S\, S_{xx} - S_x^2 \tag{D.19}$$
$$b_3 = S_x\, S_{xy} - S_y\, S_{xx} \tag{D.20}$$
$$c_3 = S_{xx}\, S_{yy} - S_{xy}^2 \tag{D.21}$$
$$\Delta = S\, c_3 + S_y\, b_3 + S_x\, a_3 \tag{D.22}$$

D.3 Nouvelle écriture du chi-2

Le chi-2 peut alors être réécrit sous la forme simplifiée :

$$\chi^2 = c^2\,S + 2\,a\,c\,S_x + 2\,b\,c\,S_y - 2\,c\,S_z + a\,a\,S_{xx} + 2\,a\,b\,S_{xy} - 2\,a\,S_{xz} + b^2\,S_{yy} - 2\,b\,S_{yz} + S_{zz} \quad \text{(D.23)}$$

D.4 Incertitude sur les coefficients de régression

Le calcul n'est cependant pas terminé, et il reste à évaluer l'incertitude sur les estimations de a, b, et c. Pour celà, nous utilisons la loi de propagation des erreurs, qui dit que la variance σ_f^2 de la valeur d'une fonction quelconque f est :

$$\sigma_f^2 = \sum_{i=1}^{N} \sigma_i^2 \left(\frac{\partial f}{\partial y_i}\right)^2 \quad \text{(D.24)}$$

ce qui donne dans notre cas :

$$\sigma_a^2 = \frac{S\,S_{yy} - S_y^2}{S\,S_{xx}\,S_{yy} - S\,S_{xy}^2 + 2\,S_{xy}\,S_x\,S_y - S_y^2\,S_{xx} - S_{yy}S_x^2} \quad \text{(D.25)}$$

$$\sigma_b^2 = \frac{S\,S_{xx} - S_x^2}{S\,S_{xx}\,S_{yy} - S\,S_{xy}^2 + 2\,S_{xy}\,S_x\,S_y - S_y^2\,S_{xx} - S_{yy}S_x^2} \quad \text{(D.26)}$$

$$\sigma_c^2 = \frac{S_{xx}\,S_{yy} - S_{xy}^2}{S\,S_{xx}\,S_{yy} - S\,S_{xy}^2 + 2\,S_{xy}\,S_x\,S_y - S_y^2\,S_{xx} - S_{yy}S_x^2} \quad \text{(D.27)}$$

D.5 Cas où les incertitudes sur les mesures sont inconnues

Il faudrait en plus pouvoir déterminer la *qualité* de l'approximation, or dans notre cas nous ne connaissons pas les incertitudes σ_i associées à chaque mesure de disparité z_i [1]. Nous ne pouvons donc alors que faire l'hypothèse que l'approximation est bonne, et il suffit alors de poser $\sigma_i = 1$ dans toutes les équations et de diviser σ_a, σ_b et σ_c par [2] $\sqrt{\chi^2/(N-3)}$.

1. On aurait pu également chercher à déterminer expérimentalement ces incertitudes.
2. Le 3 qui apparaît est le nombre de degrés de liberté du modèle.

Annexe E

Code source

LE CODE SOURCE présenté ici[1] correspond à différentes sections du chapitre 3 : La méthode classique, avec en plus le traitement de la couleur (§ 3.1), la méthode avec prise en compte de l'inclinaison et de la courbure (§ 3.2.2), et la parallélisation de la stéréoscopie par corrélation (§ 3.1.7). Il utilise très peu d'appels à des bibliothèques extérieures (les routines de rectification et de filtrage sont simples à réécrire) et devrait donc être facilement compréhensible, d'autant plus qu'il est largement commenté.

Les définitions suivantes sont nécessaires pour la compréhension de certaines parties du code :

```
/* you should really define STEREO_RECTFLOAT : because of the
   arithmetic coprocessor, working on floats is much faster than on
   chars and ints.  Even in PVM mode, the more important resulting
   network traffic doesn't seem to be a bottleneck and floats are
   still faster.  Besides, many features won't work if you don't define
   this (such as zero-mean and normalized criterions) */
#define STEREO_RECTFLOAT

#ifdef STEREO_RECTFLOAT
typedef float rectimg_t;
#ifdef MD96
#define STEREO_FLOATBUF
#endif
#ifdef STEREO_FLOATBUF
typedef float tmpbuf_t;
#else
typedef double tmpbuf_t;
#endif
#ifdef STEREO_DOUBLEIMG
typedef double tmpimg_t;
#else
typedef float tmpimg_t;
#endif
#else  /* !STEREO_RECTFLOAT */
typedef unsigned char rectimg_t;
#if (SIZEOF_INT >= 4)
typedef int tmpimg_t;
typedef int tmpbuf_t;
#else
typedef long tmpimg_t;
typedef long tmpbuf_t;
#endif /* SIZEOF_INT < 4 */
#endif /* !STEREO_RECTFLOAT */

/* correlation flags */
#define STEREO_ZEROMEAN 0x1
#define STEREO_NORMALIZE 0x2
#define STEREO_FUNC 0x4 /* mask to get correlation function */
#define STEREO_SSD 0x0
#define STEREO_CC 0x4
#define STEREO_BACK 0x8

/* for steCorPVM.c */
#define STEREO_PVM_PARAMS 1     /* PVM message for image parameters */
#define STEREO_PVM_IMAGES 2     /* PVM message for images */
#define STEREO_PVM_DISPARITY 3  /* PVM message for disparity map */
#define STEREO_PVM_MASTER_EXIT 4 /* PVM message if master exits */
#define STEREO_PVM_SLAVE_EXIT 5 /* PVM message if slave exits */
```

1. Ce code est ©1993-1996 Frédéric DEVERNAY & INRIA.

E.1 Corrélation couleur optimisée

Le code de stéréoscopie couleur par corrélation comporte seul un point d'entrée, qui est la fonction `steCorrelation()`. Cette fonction prend en entrée :

- deux images comportant un nombre quelconque de plans (1 pour des images N&B, 3 pour des images couleurs) ;
- l'intervalle de disparité ;
- la taille de la fenêtre de corrélation ;
- le critère de corrélation à utiliser ;
- s'il faut ou non effectuer une validation (§ 3.1.5).

et renvoie :

- les images rectifiées filtrées par la moyenne dans le cas d'un critère moyenné (type Z) ;
- la carte de disparité ;
- optionellement, la carte des scores de corrélation.

```
/*
 * correlation functions in color
 *
 * Author: Frederic Devernay
 * (C) INRIA 1993-96
 */

#ifdef HAVE_CONFIG_H
#include "config.h"
#endif
#include "RstereoP.h"

/* prototypes */
int steCorrelation(rectimg_t **images1, int width1, int height1, int depth,
                   rectimg_t **images2, int width2, int height2, int flags,
                   int dispNear, int dispFar, int corWidth, int corHeight,
                   float *disparityImg, float *scoremin);
static int steCorColor(rectimg_t **images1, int width1, int height1,
                       int depth, rectimg_t **images2, int width2,
                       int height2, int flags, int dispMin, int dispMax,
                       float dispError, int corWidth, int corHeight,
                       float *disparityImg, float *sdev1, float *sdev2,
                       float *scoremin);

/*
 * compute critrowcol for one line of the images
 */
inline static void
steComputeCritrowcol(tmpbuf_t *crit,
```

```
                    tmpbuf_t *critrow,
                    int ycritrow,
                    tmpbuf_t *critrowcol,
                    rectimg_t **images1,
                    int y1,
                    rectimg_t **images2,
                    int y2,
                    int min_x,
                    int max_x,
                    int depth,
                    int disparity,
                    int corWidth,
                    int corFunc)
{
    int x, d;

    /* first, compute crit */
    if (depth) {
        if(corFunc) {
            for(x=min_x-(corWidth-1); x<max_x; x++) { /* CC */
                crit[x] = 0;
                for( d=0; d<depth; d++ ) {
#ifdef STEREO_RECTFLOAT
                    crit[x] -= images1[d][y1+x]*images2[d][y2+disparity+x];
#else
                    crit[x] -= (images1[d][y1+x]-128)
                               *(images2[d][y2+disparity+x]-128);
#endif
                }
            }
        } else {
            for(x=min_x-(corWidth-1); x<max_x; x++) { /* SSD */
                crit[x] = 0;
                for( d=0; d<depth; d++ )
                    crit[x] += (images1[d][y1+x]-images2[d][y2+disparity+x]) *
                        (images1[d][y1+x]-images2[d][y2+disparity+x]);
            }
        }
    } else { /* depth == 0 */
        if(corFunc) {
            for(x=min_x-(corWidth-1); x<max_x; x++) { /* CC */
#ifdef STEREO_RECTFLOAT
                crit[x] = - (*images1)[y1+x]*(*images2)[y2+disparity+x];
#else
                crit[x] = - ((*images1)[y1+x]-128)
                           *((*images2)[y2+disparity+x]-128);
#endif
            }
        } else {
            for(x=min_x-(corWidth-1); x<max_x; x++) { /* SSD */
                crit[x] = (*images1)[y1+x]-(*images2)[y2+disparity+x];
                crit[x] *= crit[x];
            }
        }
    }

    critrow[ycritrow+min_x] = 0.0;
    /* first <corWidth> column */
    for(x=min_x-(corWidth-1); x<=min_x; x++) {
        critrow[ycritrow+min_x] += crit[x];
    }
    /* remember, we already substracted critrow[ycritrow+x] of the
       preceding line at the end of the last main loop */
    critrowcol[min_x] += critrow[ycritrow+min_x];
    /* propagation on other columns */
    for(x=min_x+1; x<max_x; x++) {
        critrow[ycritrow+x] =( critrow[ycritrow+x-1] - crit[x-corWidth]
                               + crit[x] );
        critrowcol[x] += critrow[ycritrow+x];
    }
}

/*
 * subpixel approximation, given three scores
 */
inline static double
steSubpixel(double before,
            double now,
            double after)
{
#ifndef STEREO_ROOF
    /* sub-pixel approximation, using the extremum of a parabola. */
    /* parenthesis are important in the denominator to avoid round-off */
    /* errors */
    return (after-before)/(2.0*((now-after)+(now-before)));
#else /* STEREO_ROOF */
    /* Sub-pixel approximation, using the extremum of a roof. It should
       be better but autocorrelation maps showed (using imageStats, in
       the test directory) that the parabola approximation was better */
    if(after<before) {          /* after BETTER than before */
        return (after-before)/(2.0*(now-before));
    } else {                    /* before BETTER than after */
        return (after-before)/(2.0*(now-after));
    }
#endif
}

/*
 * do we have new minimas for this score line?
 */
inline static void
steMinima(tmpbuf_t *scorenew,
          tmpimg_t *score,
          tmpimg_t *scoreprec,
          float *scoremin,
          float *disparityImg,
          int min_x,
          int max_x_prec,
          int y,                /* really y*width */
          double disparityprec, /* where to fill the disparity map */
          float dispError)
{
    int x;

    for(x=min_x; x < max_x_prec; x++) {
        /* do we have a minimum for the preceding value of disparity & is it */
        /* the best one ever ? */
        if((score[y+x] < scoremin[y+x]) &&
           (score[y+x] <= scorenew[x]) &&
           (score[y+x] < scoreprec[y+x])) {
            scoremin[y+x] = score[y+x];
            if((scorenew[x]    >= HUGE_VAL) || /* must really be a
                                                  local minimum */
               (scoreprec[y+x] >= HUGE_VAL)) { /* extremes are not tolerated */
                disparityImg[y+x] = dispError;
            } else {
                disparityImg[y+x] =
                    disparityprec + steSubpixel(scoreprec[y+x], score[y+x],
                                                scorenew[x]);
            }
        }
        scoreprec[y+x] = scorenew[x];
    }
}

/*
 * compute the disparity map between two rectified images
 *
 * The error value is dispFar==disparityImg[0].
 * The score value is lower at best matches. If !flags&STEREO_NORMALIZE,
 * the score will be divided by the correlation window area (in order to
 * have the same score magnitudes, independently of window size).
 * Warning: if (flags&STEREO_ZEROMEAN), the input images will be
 * mean-filtered.
 *
 * returns: OK or ERROR
 *
 * Author: Frederic Devernay
 * (C) INRIA 1993-96
 */
int
steCorrelation(rectimg_t **images1, /* reference images (must be
                                       scaled to reflect weights given to
                                       each channel). Pass (&image1) if
                                       depth==1 */
               int width1,      /* width of images1 in pixels (>=corWidth) */
               int height1,     /* height of images1 in pixels (>=corHeight) */
               int depth,       /* depth (= number of images) */
               rectimg_t **images2, /* other images (may be mean filtered) */
               int width2,      /* width of image2 in pixels (>=corWidth) */
               int height2,     /* height of image2 in pixels */
               int flags,       /* constructed by OR-ing Flags from
                                   the following list: STEREO_ZEROMEAN
                                   (mean-filter rectified images).
                                   STEREO_NORMALIZE (use a normalized
                                   correlation function, i.e. divide
                                   it by the local image norm).
                                   STEREO_SSD or STEREO_CC
                                   (correlation function to be used,
                                   either Sum of Squared Differences
                                   or Cross-Correlation).  STEREO_BACK
                                   (perform back-correlation and
                                   validation) */
               int dispNear,    /* disparity corresponding to near points. */
               int dispFar,     /* disparity for far points. */
               int corWidth,    /* width of the correlation window (>=1) */
               int corHeight,   /* height of the correlation window (>=1) */
               float *disparityImg,/* output disparity image
                                   (width1xheight1) point (x,y) is the
                                   disparity associated with
                                   image1(x,y). dispMin+0.5 <
                                   disparity(x,y) < dimMax-0.5
                                   (extremes of the interval cannot be
                                   extrema of the correlation score) */
               float *scoremin) /* image formed by the minimum scores,
                                   which correspond to the best
                                   correlation (width1xheight1). There
                                   is a pointwise correspondence with
                                   disparityImg and image1.  It can be
                                   NULL. */
{
    float *sdev1 = NULL, *sdev2 = NULL;
    float *disparity21 = NULL;
    int freeDisp2 = 0;
    int dispMin, dispMax;
    float dispError;
    int i;

    /* check input parameters */
    if((images1 == NULL) ||
       (width1 < corWidth) ||
       (height1 < corHeight) ||
       (depth < 1) ||
       (images2 == NULL) ||
       (width2 < corWidth) ||
       (corWidth < 1) ||
       (corHeight < 1) ||
       (disparityImg == NULL)) {
        fprintf(stderr,"steCorColor: invalid input parameters:\n");
        fprintf(stderr," images1=%p\n width1=%d\n height1=%d\n depth=%d\n",
                images1, width1, width2, depth);
        fprintf(stderr," images2=%p\n width2=%d\n height2=%d\n flags=%d\n",
                images2, width2, height2, flags);
        fprintf(stderr," dispNear=%d\n dispFar=%d\n",
                dispNear, dispFar);
        fprintf(stderr," corWidth=%d\n corHeight=%d\n disparityImg=%p\n",
                corWidth, corHeight, disparityImg);
        fprintf(stderr," sdev1=%p\n sdev2=%p\n scoremin=%p\n",
                sdev1, sdev2, scoremin);
        return ERROR;
    }

    if ((flags&STEREO_NORMALIZE) && ((flags&STEREO_BACK)||(scoremin!=NULL))) {
        sdev1 = (float*)malloc(width1*height1*sizeof(float));
    }
    if(flags&STEREO_NORMALIZE) {
        sdev2 = (float*)malloc(width2*height2*sizeof(float));
    }
    if (flags&STEREO_BACK) {
        /* sdev2 is also used as the back correlation disparity map */
        if (sdev2!=NULL)
            disparity21 = sdev2;
        else {
            disparity21 = (float*)malloc(width2*height2*sizeof(float));
```

```
            freeDisp2++;
        }
    }

    /* compute dispMin, dispMax, and dispError */
    if(dispNear<dispFar) {
        dispMin = dispNear;
        dispMax = dispFar;
    } else {
        dispMin = dispError = ceil(dispFar-0.5);
        dispMax = floor(dispNear+0.5);
    }
    dispError = dispFar;

    /* preprocess rectified images for the first pass */
    if(flags&STEREO_ZEROMEAN) {         /* mean filter if requested */
#ifdef STEREO_RECTFLOAT
        for( i=0; i<depth; i++ ) {
            steFilter(images1[i], width1, height1, corWidth, corHeight,
                      STEREO_ZEROMEAN, images1[i]);
            steFilter(images2[i], width2, height2, corWidth, corHeight,
                      STEREO_ZEROMEAN, images2[i]);
        }
#else
        fprintf(stderr,"zero mean not yet implemented\n");
#endif
    }
    if(sdev1 != NULL) {         /* local norm if requested */
#ifdef STEREO_RECTFLOAT
        if(depth == 1) {
            steFilter(*images1, width1, height1, corWidth, corHeight,
                      STEREO_NORMALIZE, sdev1);
        } else {
            float *tmpimage;
            int j;

            tmpimage = (float*)malloc( width1*height1*sizeof(float) );
            assert( tmpimage != NULL );
            memset( sdev1, 0, width1*height1*sizeof(float) );
            for( i=0; i<depth; i++ ) {
                steFilter(images1[i], width1, height1, corWidth, corHeight,
                          STEREO_NORMALIZE, tmpimage);
                for( j=0; j<width1*height1; j++ )
                    sdev1[j] += tmpimage[j]*tmpimage[j];
            }
            free( tmpimage );
            for( j=0; j<width1*height1; j++ )
                sdev1[j] = sqrt(sdev1[j]);
        }
#else
        fprintf(stderr,"normalization not yet implemented\n");
#endif
    }
    if(sdev2 != NULL) {         /* local norm if requested */
#ifdef STEREO_RECTFLOAT
        if(!depth) {
            steFilter(*images2, width2, height2, corWidth, corHeight,
                      STEREO_NORMALIZE, sdev2);
        } else {
            float *tmpimage;
            int j;

            tmpimage = (float*)malloc( width2*height2*sizeof(float) );
            assert( tmpimage != NULL );
            memset( sdev2, 0, width2*height2*sizeof(float) );
            for( i=0; i<depth; i++ ) {
                steFilter(images2[i], width2, height2, corWidth, corHeight,
                          STEREO_NORMALIZE, tmpimage);
                for( j=0; j<width2*height2; j++ )
                    sdev2[j] += tmpimage[j]*tmpimage[j];
            }
            free( tmpimage );
            for( j=0; j<width2*height2; j++ )
                sdev2[j] = sqrt(sdev2[j]);
        }
#else
        fprintf(stderr,"normalization not yet implemented\n");
#endif
    }
    if ( steCorColor(images1, width1, height1, depth,
                     images2, width2, height2, flags,
                     dispMin, dispMax, dispError,
                     corWidth, corHeight,
                     disparityImg, sdev1, sdev2, scoremin) == ERROR )
        return ERROR;
    if(flags&STEREO_BACK) {
        if ( steCorColor(images2, width2, height2, depth,
                         images1, width1, height1, flags,
                         -dispMax, -dispMin, -dispError,
                         corWidth, corHeight,
                         disparity21, NULL, sdev1, NULL) == ERROR )
            return ERROR;
        steValidate(disparityImg, width1, height1, dispError,
                    disparity21, width2, height2, -dispError,
                    1.0);       /* validation of disparity12 */
    }
    if (sdev1 != NULL)
        free (sdev1);
    if (sdev2 != NULL)
        free (sdev2);
    if (freeDisp2)
        free (disparity21);
    return OK;
}

/*
 * subroutine of steCorrelation that does the basic correlation stuff
 *
 * see above for an explanation of the parameters.
 */
static int
steCorColor(rectimg_t **images1,
            int width1,
            int height1,
            int depth,
            rectimg_t **images2,
            int width2,
            int height2,
            int flags,
            int dispMin,        /* minimum disparity in pixels */
            int dispMax,        /* maximum disparity in pixels */
            float dispError,    /* disparity error value */
            int corWidth,
            int corHeight,
            float *disparityImg,
            float *sdev1,
            float *sdev2,
            float *scoremin)
{
    /* the size of critrow */
    const int max_ycritrow = corHeight*width1;
    /* offset from the lower-right corner of the correlation window to its */
    /* center */
    const int image1Offset = -((corHeight-1)/2)*width1 - (corWidth-1)/2;
    const int image2Offset = -((corHeight-1)/2)*width2 - (corWidth-1)/2;
    /* bound on usable disparities */
    int dispSup;
    int dispInf;
    const int corFunc = (flags&STEREO_CC) != 0;

    int xy, max_xy, x, min_x=0, max_x, max_x_prec, y1, max_y1;
    int minheight;
    int ycritrow, y2;
    int disparity;
    int freeScoremin=0, freeScorenew = 0;
    tmpimg_t *scoretmp;
    tmpbuf_t *crit;
    tmpbuf_t *critrow;
    tmpbuf_t *critrowcol;
    tmpbuf_t *scorenew;
    tmpimg_t *score;
    tmpimg_t *scoreprec;

    if( depth == 1 )
        depth = 0;              /* for faster color/b&w tests */
    /* memory allocation */
    crit = (tmpbuf_t*)malloc(width1*sizeof(tmpbuf_t));
    critrow = (tmpbuf_t*)malloc(max_ycritrow*sizeof(tmpbuf_t));
    critrowcol = (tmpbuf_t*)malloc(width1*sizeof(tmpbuf_t));
    if(!(flags&STEREO_NORMALIZE)) {
        /* no normalization */
        scorenew = critrowcol;
    } else {
        /* normalization */
        freeScorenew++;
        scorenew = (tmpbuf_t*)malloc(width1*(sizeof(tmpbuf_t)));
        /* if the user asks for the score, we must have sdev1 too */
        if ((scoremin != NULL) && (sdev1 == NULL)) {
            fprintf(stderr,"steCorColor: to get the score (scoremin) with a \
normalized criterion, you must provide sdev1.\n");
            return ERROR;
        }
    }
    score = (tmpimg_t*)malloc(width1*height1*sizeof(tmpimg_t));
    scoreprec = (tmpimg_t*)malloc(width1*height1*sizeof(tmpimg_t));
    if(scoremin == NULL) {
        freeScoremin++;
        scoremin = (float*)malloc(width1*height1*sizeof(float));
    }
    if((crit == NULL) ||
       (critrow == NULL) ||
       (critrowcol == NULL) ||
       (scorenew == NULL) ||
       (score == NULL) ||
       (scoreprec == NULL) ||
       (scoremin == NULL)) {
        perror("steCorColor");
        fprintf(stderr,"steCorColor: memory allocation error.\n");
        return ERROR;
    }

    /* adjusting disparity bounds to remove unusable values */
    dispSup = width2 - (int)corWidth;
    dispInf = -(width1 - (int)corWidth);
    if(dispMin > dispSup)
        dispMin = dispSup;
    if(dispMin < dispInf)
        dispMin = dispInf;
    if(dispMax > dispSup)
        dispMax = dispSup;
    if(dispMax < dispInf)
        dispMax = dispInf;
    /* beginning of the correlation itself. */
    minheight = height1 < height2 ? height1 : height2;
    /* fill the disparity map with dispError and the scores with HUGE_VAL*/
    max_xy = height1*width1;
    for(xy = 0; xy<max_xy; xy++) {
        disparityImg[xy] = dispError;
        score[xy] = scoreprec[xy] = scoremin[xy] = HUGE_VAL;
    }
    /* shift the origin for the "centered" images */
    scoremin += image1Offset;
    disparityImg += image1Offset;
    if(flags&STEREO_NORMALIZE)
        sdev2 += image2Offset;
    /* initialize the remaining arrays */
    for(x=corWidth-1; x<width1; x++)
        scorenew[x] = critrowcol[x] = HUGE_VAL;
    max_x_prec = width1;                /* no line was computed before. */
    /* used in steMinima */
    /* main loop, on the disparitites */
    for(disparity = dispMin; disparity <= dispMax; disparity++) {
        /* bounds for x where score can be computed */
        /* x is the lower righ corner of a correlation window */
        if (disparity > 0) {
            min_x = corWidth - 1;
            max_x = width2 - disparity;
            if(max_x > width1) {
                max_x = width1;
            }
        } else {
            min_x = corWidth - disparity - 1;
            max_x = width1;
            if(max_x > (width2 - disparity))
```

```
            max_x = width2 - disparity;
        }
        if(max_x_prec > max_x)
            scorenew[max_x_prec-1] = HUGE_VAL; /* so this won't be a minimum */
        /* Here we can have (min_x > minwidth) || (max_x < 0) but we don't */
        /* care much, because in this case, for() loops will never be run */
        /* anyway. */
        /* Prepare the first line of the disparity map. */
        /* Empty the scores buffer. */
        for(x=min_x; x<max_x; x++) {
            critrowcol[x] = 0.0;
        }
        max_y1 = (corHeight-1)*width1;
        for(y1=0, y2=0, ycritrow = 0;
            y1<max_y1;
            y1+=width1, y2+=width2, ycritrow+=width1) {
            steComputeCritrowcol(crit, critrow, ycritrow, critrowcol,
                                 images1, y1,
                                 images2, y2, min_x, max_x, depth, disparity,
                                 corWidth, corFunc);
        }
        /* Middle lines of the disparity map. */
        max_y1 = minheight*width1;
        for (y1=(corHeight-1)*width1,
                 y2=(corHeight-1)*width2,
                 ycritrow = (corHeight-1)*width1;
             y1<max_y1;
             y1+=width1, y2+=width2) {
            steComputeCritrowcol(crit, critrow, ycritrow, critrowcol,
                                 images1, y1,
                                 images2, y2, min_x, max_x, depth, disparity,
                                 corWidth, corFunc);
            if(flags&STEREO_NORMALIZE) { /* else, scorenew==critrowcol */
                for(x=min_x; x<max_x; x++) /* we want normalization */
                    scorenew[x] = critrowcol[x]/sdev2[disparity+y2+x];
                /* No need to divide by sdev1 also, because sdev1
                   will be the same for all values of the disparity, at one
                   given point. */
            }
            /* test for possible best matches */
            steMinima(scorenew, score, scoreprec, scoremin, disparityImg,
                      min_x, max_x_prec, y1, disparity-1, dispError);
            /* prepare the next iteration */
            ycritrow += width1;
            if(ycritrow >= max_ycritrow)
                ycritrow = 0;
            /* prepare the new score with respect to the score of the */
            /* preceding line, which was moved to score[] during steMinima() */
            for(x=min_x; x<max_x; x++)
                critrowcol[x] -= critrow[ycritrow+x];
        } /* for y1 */
        max_x_prec = max_x;     /* prepare for the next call to steMinima */
        scoretmp = score;       /* scoreprec contains the new score, in fact */
        /* so swap it with score */
        score = scoreprec;
        scoreprec = scoretmp;
    }
    /* We need to check if the scores for the last disparity value is less */
    /* than the best score. In this case, set the disparity to dispError. */
    max_y1 = minheight*width1;
    for(y1=(corHeight-1)*width1;  y1<max_y1; y1+=width1)
        for(x=min_x; x<max_x_prec; x++)
            if(score[y1+x] < scoremin[y1+x]) {
                scoremin[y1+x] = score[y1+x]; /* in case we want to use */
                /* scoremin */
                disparityImg[y1+x] = dispError;
            }

    /* Free working arrays if necessary. */
    scoremin -= image1Offset;
    disparityImg -= image1Offset;
    sdev2 -= image2Offset;
    free (crit);
    free (critrow);
    free (critrowcol);
    free (score);
    free (scoreprec);
    if(freeScoremin) {
        free(scoremin);
    } else {
        /* if we use a normalized criterion and we must provide the score,
           divide the score by sdev1 */
        max_xy = width1*height1;
        if (flags&STEREO_NORMALIZE) {
            for (xy=0; xy<max_xy; xy++)
                if (disparityImg[xy] != dispError)
                    scoremin[xy] /= sdev1[xy];
        } else {
            for (xy=0; xy<max_xy; xy++)
                if (disparityImg[xy] != dispError)
                    scoremin[xy] /= corWidth*corHeight;
        }
        /* clean the score array, put zeroes where appropriate */
        for (xy=0; xy<max_xy; xy++)
            if (disparityImg[xy] == dispError)
                scoremin[xy] = 0;
    }
    if(freeScorenew)
        free(scorenew);
    return OK;
}
```

E.2 Corrélation avec calcul simultané des dérivées

Ce code fait appel à des fonctions de minimisation aux moindres carrés non-linéaires (miniNLS) et de minimisation d'une fonction objective (miniTN), qui peuvent être écrites à partir des outils de Netlib[2]. Les deux points d'entrée sont le calcul de la disparité et de ses dérivées à partir d'estimées initiales en un point et sur toute l'image (`steCorStretchPoint()` et `steCorStretch()`).

```
/*
 * stereo by stretch and shear correlation
 *
 * AUTHOR: Frederic Devernay
 * (C) INRIA 1993-96
 */

#ifdef HAVE_CONFIG_H
#include "config.h"
#endif
#include "RstereoP.h"
#include <Rmini.h>

static const rectimg_t *_ptI1, *_ptI2;  /* position of the current point
                                           in images */
static int _imageWidth;         /* width of I1 and I2 */
static int _halfni, _halfnj;    /* (ni-1)/2 and (nj-1)/2 */
static int _ninj;               /* ni*nj */
static float _dx, _dy;          /* first derivatives of disparity */
static float _dx2, _dxy, _dy2;  /* second derivatives of disparity */
static double *_deltaxI, *_deltaxJ; /* delta in x for an increment of i or j */
                                /* in the correlation window. size ni and nj */

/*
 * compute the correlation error for SSD
 */
static void
steCorStretchSSDFunction(int m, /* number of functions == ni x nj */
                         int n, /* number of variables == 3 */
                         const double *x, /* vector of parameters (length n) */
                         double *fvec) /* function values at x (length m) */
{
    int i, j, ivec;
    int jnx;

    double disparity, dx, dy, dx2, dxy, dy2;
    double pixelWidthI2;

    disparity = x[0];
    switch (n) {
    case 1:
        dx = _dx;
        dy = _dy;
        dx2 = _dx2;
        dxy = _dxy;
        dy2 = _dy2;
        break;
    case 3:
        dx = x[1];
        dy = x[2];
        dx2 = _dx2;
        dxy = _dxy;
        dy2 = _dy2;
        break;
    case 6:
        dx = x[1];
        dy = x[2];
        dx2 = x[3];
        dxy = x[4];
        dy2 = x[5];
        break;
    default:
        fprintf (stderr, "steCorStretchSSDFunction: illegal number of \
variables\n");
        return;
        break;
    }

    pixelWidthI2 = dx+1.0;      /* with of a I1 pixel in I2 */

    /* fill the two offset arrays */
    for(i=-_halfni; i<=_halfni; i++)
        _deltaxI[i] = disparity + pixelWidthI2*i + dx2*(i*i)/2.0;
    for(j=-_halfnj; j<=_halfnj; j++)
        _deltaxJ[j] = dy*j + dy2*(j*j)/2.0;

    for(j=-_halfnj, jnx = - _halfnj*_imageWidth, ivec = 0;
        j<=_halfnj;
        j++, jnx += _imageWidth) {
        for(i=-_halfni; i<=_halfni; i++, ivec++) {
            double x2, fracx2, i1, i2;
            int floorjnx2;

            /* x2 = disparity + (dx+1.0)*i + dx2*(i*i)/2.0
                  + dy*j + dy2*(j*j)/2.0
                  + dxy*i*j; */
            x2 = _deltaxI[i]+_deltaxJ[j] + dxy*i*j;
            fracx2 = x2 - floor(x2);
            floorjnx2 = jnx + (int)x2; /* x2 is positive, so
                                          (int) == floor() */
            i2 = _ptI2[floorjnx2]*(1.0-fracx2) + _ptI2[floorjnx2+1]*fracx2;
            i1 = _ptI1[jnx+i];

            fvec[ivec] = i2-i1; /* SSD */
        }
    }
}

/*
 * compute the jacobian of the correlation error for SSD
 *
 * only
```

[2] http://www.netlib.org/

```
 */
static void
steCorStretchSSDJacobian(int m, /* number of functions */
                         int n, /* number of variables */
                         const double *x, /* vector of parameters (length n) */
                         double *fjac) /* jacobian at x (mXn, fortran array) */
{
    int i, j, ivec;
    int jnx;

    double disparity, dx, dy;
    double dx2, dxy, dy2;
    double pixelWidthI2;
    double dd;

    disparity = x[0];
    switch (n) {
    case 1:
        dx = _dx;
        dy = _dy;
        dx2 = _dx2;
        dxy = _dxy;
        dy2 = _dy2;
        break;
    case 3:
        dx = x[1];
        dy = x[2];
        dx2 = _dx2;
        dxy = _dxy;
        dy2 = _dy2;
        break;
    case 6:
        dx = x[1];
        dy = x[2];
        dx2 = x[3];
        dxy = x[4];
        dy2 = x[5];
        break;
    default:
        fprintf (stderr, "steCorStretchSSDJacaboian: illegal number of \
variables\n");
        return;
        break;
    }

    pixelWidthI2 = dx+1.0;      /* with of a I1 pixel in I2 */

    /* fill the two offset arrays */
    for(i=-_halfni; i<=_halfni; i++)
        _deltaxI[i] = disparity + pixelWidthI2*i + dx2*(i*i)/2.0;
    for(j=-_halfnj; j<=_halfnj; j++)
        _deltaxJ[j] = dy*j + dy2*(j*j)/2.0;

    for(j=-_halfnj, jnx = - _halfnj*_imageWidth, ivec = 0;
        j<=_halfnj;
        j++, jnx += _imageWidth)
        for(i=-_halfni; i<=_halfni; i++, ivec ++) {
            double x2;
            int floorjnx2;

            x2 = _deltaxI[i]+_deltaxJ[j] + dxy*i*j;
            floorjnx2 = jnx + (int)x2; /* x2 is positive, so (int)==floor() */
            dd = _ptI2[floorjnx2+1] - _ptI2[floorjnx2];
            switch (n) {
            case 1:
                fjac[ivec] = dd;
                break;
            case 3:
                fjac[ivec] = dd;
                fjac[ivec + m] = i*dd;
                fjac[ivec + 2*m] = j*dd;
                break;
            case 6:
                fjac[ivec] = dd;
                fjac[ivec + m] = i*dd;
                fjac[ivec + 2*m] = j*dd;
                fjac[ivec + 3*m] = ((i*i)/2.0)*dd;
                fjac[ivec + 4*m] = (i*j)*dd;
                fjac[ivec + 5*m] = ((j*j)/2.0)*dd;
                break;
            default:
                assert((n==1) || (n==3) || (n==6));
                break;
            }
        }
}


/*
 * compute the correlation error for CC and its gradient
 */
static void
steCorStretchCCFuncGrad(int n,  /* number of variables == 3 */
                        const double *x, /* vector of parameters (length n) */
                        double *f, /* function value at x */
                        double *g) /* function gradient at x */
{
  int i, j;
  int jnx;
  double dd;

  double disparity, dx, dy;
  double pixelWidthI2;

  disparity = x[0];
  if(n > 1) {                   /* first derivatives are variables */
      dx = x[1];
      dy = x[2];
  } else {                      /* first derivatives are constants */
      dx = _dx;
      dy = _dy;
  }
  pixelWidthI2 = dx+1.0;        /* with of a I1 pixel in I2 */

  *f = *g = 0.0;

  /* fill the two offset arrays */
  for(i=-_halfni; i<=_halfni; i++)
    _deltaxI[i] = disparity + (double)i * pixelWidthI2;
  for(j=-_halfnj; j<=_halfnj; j++)
    _deltaxJ[j] = (double)j * dy;

  for(j=-_halfnj, jnx = - _halfnj*_imageWidth;
      j<=_halfnj;
      j++, jnx += _imageWidth)
    for(i=-_halfni; i<=_halfni; i++) {
      double x2        = _deltaxI[i]+_deltaxJ[j];
      double fracx2    = x2 - floor(x2);
      int    floorjnx2 = jnx + (int)x2; /* x2 is positive, so
                                           (int) == floor() */
      double i2        = _ptI2[floorjnx2]*(1.0-fracx2)
                       + _ptI2[floorjnx2+1]*fracx2;
      double i1        = _ptI1[jnx+i];

      *f -= i1*i2;              /* CC */
      dd = -(_ptI2[floorjnx2+1] - _ptI2[floorjnx2]) * i1;
      g[0] += dd;               /* dCC/dd */
      if(n > 1) {
          g[1] += i * dd;       /* dCC/d(dx) */
          g[2] += j * dd;       /* dCC/d(dy) */
      }
    }
}

/*
 * refine the disparity and its derivatives at one point
 *
 * MT-LEVEL: Unsafe
 *
 * Returns: OK or ERROR
 *
 * AUTHOR: Frederic Devernay
 * (C) INRIA 1993-96
 */
int
steCorStretchPoint(const rectimg_t *ptI1, /* pointer to the centerpoint of
                                             the correlation window in I1 */
                   int width1,  /* width of image I1 */
                   const rectimg_t *ptI2, /* pointer to the point with the
                                             SAME coordinates in I2 */
                   int width2,  /* width of image I2 */
                   int flags,   /* STEREO_SSD or STEREO_CC */
                   int order,   /* order of derivatives up to which to
                                   minimize: 0, 1 or 2 */
                   int corWidth, /* width of the correlation window (odd) */
                   int corHeight, /* height of the correlation window (odd) */
                   int positionI1, /* x coordinate of ptI1 in I1 */
                   float *disparity, /* input: initial value of the
                                        disparity. output: final value of
                                        the disparity */
                   float *disparityX, /* input: initial first
                                         derivative over x of the
                                         disparity. output: optimized
                                         derivative (order>=1). */
                   float *disparityY, /* input: initial first
                                         derivative over y of the
                                         disparity. output: optimized
                                         derivative (order>=1). */
                   float *disparityX2, /* input: initial second
                                          derivative over x of the
                                          disparity. output: optimized
                                          derivative (order>=2). */
                   float *disparityXY, /* input: initial second cross
                                          derivative of the
                                          disparity. output: optimized
                                          derivative. (order>=2) */
                   float *disparityY2) /* input: initial second
                                          derivative over y of the
                                          disparity. output: optimized
                                          derivative. */
{
    double x[6];
    double *deltaxI, *deltaxJ;
    int nfunc=0;

    /* verify the parameters: ni and nj must be odd */
    if( !((corWidth & 1) && (corHeight & 1)) ) {
        fprintf(stderr,"steCorStretchPoint: width and height of the \
correlation window must be odd\n");
        return ERROR;
    }
    if( (order < 0) || (order > 2) ) {
        fprintf( stderr, "steCorStretchPoint: order (= %d) must be \
between 0 and 2.\n", order );
        return ERROR;
    }
    if ((order == 2) && ((flags & STEREO_FUNC) == STEREO_CC)) {
        fprintf( stderr, "steCorStretchPoint: second order model not yet \
implemented for CC criterion.\n");
        return ERROR;
    }
    if( width1 != width2 ) {
        fprintf( stderr, "steCorStretchPoint: width of both images must be \
identical (%d != %d). case of different image widths not yet implemented.\n",
                 width1, width2);
        return ERROR;
    }
    /* memory allocation */
    if( ((deltaxI = (double*)malloc(corWidth*sizeof(double))) == NULL) ||
        ((deltaxJ = (double*)malloc(corHeight*sizeof(double))) == NULL) ) {
        fprintf(stderr,"steCorStretchPoint: memory allocation error\n");
        return ERROR;
    }

    _ptI1 = ptI1;
    _ptI2 = ptI2;
    _imageWidth = width1;
    _halfni = (corWidth - 1)/2;
    _halfnj = (corHeight - 1)/2;
    _ninj = corWidth*corHeight;

    /* these arrays are from -corWidth/2..corWidth/2 and -nj/2..corHeight/2 */
    _deltaxI = deltaxI + _halfni;
    _deltaxJ = deltaxJ + _halfnj;
```

```
    x[0] = *disparity;
    if(order > 0) {
        x[1] = *disparityX;
        x[2] = *disparityY;
    } else {
        _dx = *disparityX;
        _dy = *disparityY;
    }

    switch(order) {
    case 0:
      nfunc = 1;                /* only the disparity */
      x[0] = *disparity;
      _dx = disparityX ? *disparityX : 0.0;
      _dy = disparityY ? *disparityY : 0.0;
      _dx2 = disparityX2 ? *disparityX2 : 0.0;
      _dxy = disparityXY ? *disparityXY : 0.0;
      _dy2 = disparityY2 ? *disparityY2 : 0.0;
      break;
    case 1:
      nfunc = 3;                /* disparity and its 1st derivatives */
      x[0] = *disparity;
      x[1] = *disparityX;
      x[2] = *disparityY;
      _dx2 = disparityX2 ? *disparityX2 : 0.0;
      _dxy = disparityXY ? *disparityXY : 0.0;
      _dy2 = disparityY2 ? *disparityY2 : 0.0;
      break;
    case 2:
      nfunc = 6;                /* disparity and its 1st & 2nd derivatives */
      x[0] = *disparity;
      x[1] = *disparityX;
      x[2] = *disparityY;
      x[3] = *disparityX2;
      x[4] = *disparityXY;
      x[5] = *disparityY2;
      break;
    }
    if( (flags & STEREO_FUNC) == STEREO_SSD ) {
#ifdef DEBUG
        miniNLS(_ninj, nfunc, x, NULL, NULL, NULL, steCorStretchSSDFunction,
                steCorStretchSSDJacobian, NULL, 0.0, 0);
#else
        miniNLS(_ninj, nfunc, x, NULL, NULL, NULL, steCorStretchSSDFunction,
                steCorStretchSSDJacobian, NULL, 0.0, -1);
#endif
    } else { /* STEREO_CC */
#ifdef STEREO_STRETCH_CC
        miniTN(nfunc,x,NULL,NULL,steCorStretchCCFuncGrad,1,1);
#else
        fprintf (stderr, "Sorry, CC not available\n");
        exit (1);
#endif
    }

    *disparity = x[0];
    if (order>=1) {
        *disparityX = x[1];
        *disparityY = x[2];
    }
    if (order >=2) {
        *disparityX2 = x[3];
        *disparityXY = x[4];
        *disparityY2 = x[5];
    }

    free((char*) deltaxI);
    free((char*) deltaxJ);

    return OK;
}

/*
 * refine the disparity and its derivatives on a full image.
 *
 * the first point of each line (d[i*imageWidth]) must be set to the
 * error value of the disparity (meaning there is no disparity at this
 * point). in fact the first point of a line never has a disparity
 * value. dx[] and dy[] can be initialized at zero if no value is
 * available.
 *
 * returns: OK or ERROR
 *
 * AUTHOR: Frederic Devernay
 * (C) INRIA 1993-96
 */
int
steCorStretch(const rectimg_t *image1, /* reference image (may be mean
                                          filtered or not), will be normalized
                                          if flags&STEREO_NORMALIZE */
              int width1,       /* width of image1 i-n pixels (>=corWidth) */
              int height1,      /* height of image1 in pixels (>=corHeight) */
              const rectimg_t *image2, /* other image (may be mean
                                          filtered), will be normalized if
                                          flags&STEREO_NORMALIZE */
              int width2,       /* width of image2 in pixels (>=corWidth) */
              int height2,      /* height of image2 in pixels
                                   (>=height1) */
              int flags,        /* constructed by OR-ing Flags from
                                   the following list:
                                   STEREO_SSD or STEREO_CC
                                   (correlation function to be used,
                                   either Sum of Squared Differences
                                   or Cross-Correlation). */
              int order,        /* order of derivatives up to which to
                                   minimize: 0, 1 or 2 */
              float dispError,  /* error value (dispError<=dispMin+0.5
                                   or dispError>dispMax-0.5) */
              int corWidth,     /* width of the correlation window (>=1) */
              int corHeight,    /* height of the correlation
                                   window (>=1) */
              float *disparityImg, /* output disparity image
                                   (width1xheight1) point (x,y) is the
                                   disparity associated with
                                   image1(x,y)
                                   dispMin+0.5<disparity(x,y)<dimMax-0.5
                                   (extremes of the interval cannot be
                                   extrema of the correlation score) */
              float *disparityXImg, /* input: initial first derivative
                                   over x of the disparity. output:
                                   optimized derivative (order>=1). */
              float *disparityYImg, /* input: initial first derivative
                                   over y of the disparity. output:
                                   optimized derivative (order>=1). */
              float *disparityX2Img, /* input: initial second
                                   derivative over x of the
                                   disparity. output: optimized
                                   derivative (order>=2). */
              float *disparityXYImg, /* input: initial second cross
                                   derivative of the
                                   disparity. output: optimized
                                   derivative (order>=2). */
              float *disparityY2Img) /* input: initial second
                                   derivative over y of the
                                   disparity. output: optimized
                                   derivative (order>=2). */
{
    if( (image1 == NULL) ||
        (width1 < corWidth) ||
        (height1 < corHeight) ||
        (image2 == NULL) ||
        (width2 < corWidth) ||
        (height2 < height1) ||
        (order < 0) || (order > 2) ||
        (corWidth < 1) ||
        (corHeight < 1) ||
        (disparityImg == NULL) ) {
        fprintf(stderr,"steCorStretch: invalid input parameters.\n");
        return ERROR;
    }
    if( (width1 != width2) || (height1 != height2) ) {
        fprintf(stderr,"steCorStretch: both images must have the same size, \
not yet implemented for images of different sizes\n");
        return ERROR;
    }
    if( dispError != disparityImg[0] ) {
        fprintf(stderr,"steCorStretch: sorry, you must still give \
dispError == disparityImg[0]\n");
        return ERROR;
    }
    if ((order>=1) && ((disparityXImg == NULL) || (disparityYImg == NULL))) {
        fprintf(stderr,"steCorStretch: you must provide disparityXImg and \
disparityYImg for order 1 model\n");
        return ERROR;
    }
    if ((order>=2) && ((disparityX2Img == NULL) || (disparityY2Img == NULL)
        || (disparityXYImg == NULL))) {
        fprintf(stderr,"steCorStretch: you must provide disparityX2Img, \
disparityXYImg and disparityY2Img for order 2 model\n");
        return ERROR;
    }
    {                           /* main processing */
        int xy, fromXY, toXY;
        int x, fromX, toX;

        fromXY = width1 * (corHeight+1)/2;
        toXY = width1 * (height1-(corHeight+1)/2);
        fromX = (corWidth+1)/2;
        toX = width1-(corWidth+1)/2;

        /* clear the first (corHeight+1)/2 lines */
        if (order>=1) {
            memset( (void*)disparityXImg, 0, fromXY*sizeof(float) );
            memset( (void*)disparityYImg, 0, fromXY*sizeof(float) );
        }
        if (order>=2) {
            memset( (void*)disparityX2Img, 0, fromXY*sizeof(float) );
            memset( (void*)disparityXYImg, 0, fromXY*sizeof(float) );
            memset( (void*)disparityY2Img, 0, fromXY*sizeof(float) );
        }
        for(xy=fromXY; xy <= toXY; xy+=width1) {
            /* for each line */
            fprintf(stderr,"line %d\n",xy/width1);
            /* refine the disparit and its derivatives */
            for( x=fromX; x<=toX; x++ ) {
                if( disparityImg[xy+x] != dispError ) {
                    steCorStretchPoint(image1 + xy + x,
                                       width1,
                                       image2 + xy + x,
                                       width2,
                                       flags,
                                       order,
                                       corWidth,
                                       corHeight,
                                       x,
                                       disparityImg + xy + x,
                                       disparityXImg ? disparityXImg + xy + x
                                       : NULL,
                                       disparityYImg ? disparityYImg + xy + x
                                       :NULL,
                                       disparityX2Img ? disparityX2Img + xy + x
                                       :NULL,
                                       disparityXYImg ? disparityXYImg + xy + x
                                       :NULL,
                                       disparityY2Img ? disparityY2Img + xy + x
                                       :NULL);
                } else {
                    if (order>=1) {
                        disparityXImg[xy + x] = disparityYImg[xy + x] = 0.0;
                    }
                    if (order >= 2) {
                        disparityX2Img[xy + x] = 0.0;
                        disparityXYImg[xy + x] = 0.0;
                        disparityY2Img[xy + x] = 0.0;
                    }
                } /* if(disparity!=disperror) */
            } /* for x */
        } /* for xy */
    } /* main processing */

    return OK;
}
```

E.3 Corrélation sous PVM

Toutes les informations sur le package PVM sont disponibles sur la PVM Home Page[3] et sur Netlib[4].

La notice d'utilisation se trouve au début du source de `steCorPVMInit()` (on trouvera également des explications § 3.1.7), et les différents points d'entrée pour l'utilisateur sont :

- `steCorPVMInit()` qui initialise PVM et crée les tâches esclaves ;
- `steCorPVMMaster()` que la tâche maître doit appeler pour rectifier et calculer la carte de disparité d'une paire d'images ;
- `steCorPVMTerminate()` que la tâche maître appelle pour ordonner aux esclaves de s'arrêter ;
- `steCorPVMSlave()` que la tâche esclave appelle une seule fois pour se mettre en attente de données et les traîter ;
- `steCorPVMExit()` pour sortir « proprement » de la machine PVM en cas d'erreur fatale.

Il est à noter que ce code a été testé de manière intensive, y compris dans toutes les situations de panne possibles. En particulier, dès que la tâche maître ou une des tâches esclaves meurt, pour une raison quelconque, toutes les tâches s'arrêtent. Il arrive souvent en PVM que ce cas-là et celui d'autres pannes ne soit pas bien traîté, et que les tâches se mettent à se multiplier indéfiniment, saturant immédiatement l'ensemble des processeurs disponibles, ou que la tâche maître ne se termine jamais, le programme restant bloqué.

E.3.1 Initialisation

```
/*
 * stereo by correlation using PVM
 *
 * The master task must execute in this order:
 * steCorPVMInit() => returns inum == 0
 * steCorPVMMaster() any number of times with same image params
 * steCorPVMTerminate()
 * steCorPVMExit()
 *
 * The slave task (which can be the same executable) must execute:
 * steCorPVMInit() => returns inum != 0
 * steCorPVMSlave() only once
 * steCorPVMExit()
 *
 * FEATURES:
 * - exit if any of the tasks dies
 * - memory saving (rectification is done by the master task on the part of
 *   the image that will be sent in, prepareBand())
 *
 * TODO:
 * - don't automatically spawn on all hosts of the PVM: add host n only if
 *   speed(n) > sum(i=1..n-1,sp(i))/nbbands
 * - dynamic reconfiguration of slave image params (easy)
 *
 * BUGS:
 * [empty bug list]
 *
 * AUTHOR: Frederic Devernay
 * (C) INRIA 1993-96
 */

#ifdef HAVE_CONFIG_H
#include "config.h"
#endif
#include "RstereoP.h"

#ifdef HAVE_PVM                 /* all this only works with PVM */
#include <pvm3.h>
#endif

#define PVMGROUP "stecor"       /* the PVM group basename */
#define DEBUG
/* #define DEBUG_SLAVES */

#ifdef HAVE_PVM

/*
 * find the group name and figure out wether I'm master or slave
 *
 * returns 1 if master, 0 if slave.
 */
static int
steCorPVMGroup(char groupname[20])
{
    int mytid;
    int parent;

    mytid = pvm_mytid();
    if (mytid < 0) {
        pvm_perror( "steCorPVMGroup" );
        return ERROR;
    }
    parent = pvm_parent();
    if( parent == PvmNoParent ) {
        /* if I have no parent, I am the master task */
        sprintf(groupname,PVMGROUP"%x",mytid);
        return 1;
    } else {
        sprintf(groupname,PVMGROUP"%x",parent);
        /* check if there is a stecorXXXX PVM group, where XXXX is the
           parent's tid. If yes, I'm a slave task */
        if( pvm_gsize(groupname) <= 0 ) {
            sprintf(groupname,PVMGROUP"%x",mytid);
            return 1;
        } else
            return 0;
    }
}

/*
 * spawn slave tasks
 */
static int
spawnSlaves( int argc,
             char **argv,
             int nbbands,
             int nhost,
             struct pvmhostinfo *hostp,
             int **tid
             )
{
    char *task;
    char **args;
    int nslave;
#ifdef STEREO_PVM_ADAPTATIVE
    int i;
    int sumspeed;
    int continueBubble, tmp;
    int *hosts;
#endif

    /* prepare arguments */
    task = argv[0];
    args = (char**)malloc(argc*sizeof(char*));
    if( argc > 1 )
        memcpy( args, argv+1, (argc-1)*sizeof(char*) );
    args[argc-1] = NULL;
#ifdef STEREO_PVM_ADAPTATIVE
    /* adaptative method. the following heuristic is used:
     * - sort hosts in descending speed order
     * - add host n only iff
     * speed(n) > sum(i=1..n-1,speed(i))/nbjobs
     * i.e. if host n could process at least 1 job during the total runtime.
     * the problem is that spawing tasks one by one generates too much
     * overhead, maybe PVM 3.4 will cure this */
    hosts = (int*)malloc(nhost * sizeof(int));
    for( i=0; i<nhost; i++) {
        hosts[i] = i;
    }
    do {                        /* bubble sort */
        continueBubble = 0;
        for(i=0; i<(nhost-1); i++) {
            if( hostp[hosts[i]].hi_speed <  hostp[hosts[i+1]].hi_speed ) {
                tmp = hosts[i+1];
                hosts[i+1] = hosts[i];
                hosts[i] = tmp;
                continueBubble = 1;
            }
        }
    } while( continueBubble );
    nslave = 0;
    sumspeed = 0;
    while( (nslave < nhost)
           && (hostp[hosts[nslave]].hi_speed > (sumspeed/nbbands)) ) {
        sumspeed += hostp[hosts[nslave]].hi_speed;
        nslave++;
    }
    assert( (nslave > 0) && (nslave <= nhost) );
    *tid = (int*)malloc(nslave * sizeof(int));
    for( i=0; i<nslave; i++ )
#ifdef DEBUG_SLAVES
        if( pvm_spawn( task, args, PvmTaskHost+PvmTaskDebug,
                       hostp[hosts[i]].hi_name, 1, *tid+i ) < 1 )
```

[3] http://www.epm.ornl.gov/pvm/

[4] http://www.netlib.org/pvm3/

```
#else
        if( pvm_spawn( task, args, PvmTaskHost, hostp[hosts[i]].hi_name,
                  1, *tid+i ) < 1 )
#endif
            {
                pvm_perror( "spawnSlaves" );
                fprintf (stderr,"spawnSlaves: maybe the executable isn't in \
PVM_PATH, try giving its complete path, beginning with \"/\".")
                return -1;
            }
    free( hosts );
#else /* !STEREO_PVM_ADAPTATIVE */
    nslave = nbbands;
    if( (nslave <= 0) || (nslave > nhost) )
        nslave = nhost;

    *tid = (int*)malloc(nslave * sizeof(int));
#ifdef DEBUG_SLAVES
    if( pvm_spawn( task, args, PvmTaskDefault+PvmTaskDebug, NULL,
                  nslave, *tid ) < nslave )
#else
    if( pvm_spawn( task, args, PvmTaskDefault, NULL,
                  nslave, *tid ) < nslave )
#endif
        {
            pvm_perror( "spawnSlaves" );
            return -1;
        }
#endif /* !STEREO_PVM_ADAPTATIVE */
    free( args );
    return nslave;
}

#endif HAVE_PVM

/*
 * Initialise the PVM tasks (master and slave) for stereo by correlation.
 *
 * after a call to steCorPVMInit, one should call steCorPVMMaster() or
 * steCorPVMSlave() to start correlation, depending on the value of
 * inum. Other PVM processing can be done before calling these
 * functions, or (for the master task) after having called
 * steCorPVMTerminate.()
 *
 * AUthor: Frederic Devernay
 */
int
steCorPVMInit( int argc, char **argv, /* command to spawn (the path
                                         should be absolute in general) */
               int *mytid,      /* this process' task ID */
               int *nslave,     /* Input: number of bands or 0 to use
                                   all hosts. Output: number of slave
                                   processes. If *nslave<=0 on input,
                                   then nslave is set to nhost, where
                                   nhost is the number of hosts in the
                                   virtual machine. If *nslave>nhost,
                                   *nslave is set to nhost. If this
                                   task is a slave, then *nslave == 0
                                   on output */
               int **tid,       /* Output: pointer to array of length
                                   nslave of task IDs. */
               int *encoding
               )
{
    struct pvmhostinfo *hostp;
    int nhost, narch;
    int master;                 /* 1 if I'm the master task */
    char groupname[20];         /* PVM group name */
    int inum;
    int parenttid;

#ifndef HAVE_PVM
    fprintf(stderr,"steCorPVMInit: PVM not enabled.\n");
    return ERROR;
#else
    parenttid = pvm_parent();
    *mytid = pvm_mytid();
    if (*mytid < 0) {
        pvm_perror( "steCorPVMInit" );
        return ERROR;
    }

    if (pvm_config( &nhost, &narch, &hostp ) < 0) {
        pvm_perror( "steCorPVMInit" );
        return ERROR;
    }
#ifdef DEBUG
    {
        int i;
        fprintf(stderr,"%d nodes on the PVM:\nhost\tarch\tspeed\n", nhost);
        for ( i=0; i< nhost; i++ )
            fprintf(stderr,"%s\t%s\t%d\n", hostp[i].hi_name,
                    hostp[i].hi_arch, hostp[i].hi_speed);
    }
#endif
    /* if in a homogeneous virtual machine, use raw data transfers */
    /* narch is the number of data formats, not the number of architectures */
    *encoding = (narch == 1) ? PvmDataRaw : PvmDataDefault;

    /* determine the group name */
    master = steCorPVMGroup( groupname );
    /* note that the group stuff is only used to determine who is the
       master and who are the slaves. I also thought of testing it
       using pvm_parent(), but even the master task can have a parent,
       e.g. if it's spawned from xpvm. */
    inum = pvm_joingroup( groupname ); /* join the group */
#ifdef DEBUG
    fprintf(stderr,"I'm a %s task, with instance number %d in group %s\n",
            master?"master":"slave", inum,  groupname);
#endif /* DEBUG */
    if( (master&&(inum!=0)) || (!master&&(inum==0)) ) {
        fprintf(stderr, "steCorPVMInit: I should be a %s task but I have \
inum %d (master should be 0) in group %s\n", master?"master":"slave",
                inum, groupname);
        return ERROR;
    }
    if (master) {               /* I'm the master task of this group */
        if( (parenttid != PvmNoParent) &&
            (pvm_pstat( parenttid ) != PvmOk) ) {
            fprintf(stderr, "steCorPVMInit: I should be the master task but \
my PVM parent died, maybe it was the master task but died to soon, so I \
prefer not spawning slaves.\n");
            return ERROR;
        }
        *nslave = spawnSlaves( argc, argv, *nslave, nhost, hostp, tid);
        if(*nslave < 0)
            return ERROR;
        /* notify the master if one of the slaves dies */
        pvm_notify( PvmTaskExit, STEREO_PVM_SLAVE_EXIT, *nslave, *tid );
    } else {
        *nslave = 0;
        *tid = NULL;
        /* notify the slave if the master dies */
        pvm_notify( PvmTaskExit, STEREO_PVM_MASTER_EXIT, 1, &parenttid );
    }

    return OK;
#endif /* HAVE_PVM */
}

/*
 * Exit the PVM task cleanly
 */
int
steCorPVMExit( void )
{
#ifndef HAVE_PVM
    fprintf(stderr,"steCorPVMExit: PVM not enabled.\n");
    return ERROR;
#else
    char groupname[20];

    steCorPVMGroup(groupname);
#ifdef DEBUG
    fprintf(stderr,"leaving group %s\n",groupname);
#endif
    if( pvm_lvgroup( groupname ) < 0 ) {
        pvm_perror( "steCorPVMExit" );
        return ERROR;
    }
    pvm_exit();

    return OK;
#endif /* HAVE_PVM */
}
```

E.3.2 Maître

```
/*
 * stereo by correlation using PVM, master part
 *
 * see steCorPVM.c for comments
 *
 * AUTHOR: Frederic Devernay
 * (C) INRIA 1993-96
 */

#ifdef HAVE_CONFIG_H
#include "config.h"
#endif
#include "RstereoP.h"

#ifdef HAVE_PVM                 /* all this only works with PVM */
#include <pvm3.h>
#endif

#define DEBUG

#ifdef HAVE_PVM

/*
 * rectify part of an image
 */
static void
rectifyOneImage( int bandid,
                 int lastbandid,
                 int maxbandheight,
                 int ystart,
                 int corHeight,
                 unsigned char **images,
                 int width,
                 int height,
                 int depth,
                 const double *rectification,
                 rectimg_t **rectifiedImages,
                 int rwidth,
                 int bandheight,
                 rectimg_t ***rimg )
{
    double rect[9] = {1.0, 0.0, 0.0,
                      0.0, 1.0, 0.0,
                      0.0, 0.0, 1.0};
    int d;

    if(rectifiedImages == NULL) { /* rectify only a band */
        if(*rimg == NULL) {
            *rimg = (rectimg_t**)malloc(depth*sizeof(rectimg_t*));
            for(d=0; d<depth; d++)
                (*rimg)[d] = (rectimg_t*)malloc(rwidth*maxbandheight
                                                *sizeof(rectimg_t));
        }
        if( bandid != lastbandid ) {
            /* we have to rectify a band of the image */
            /* part of it could be simply moved if bandid == lastbandid+-1 */
            if (rectification)
                memcpy(rect, rectification, sizeof(rect));
            if( bandid == (lastbandid + 1) ) {
                /* copy the beginning from the end of the preceding band */
                for(d=0; d<depth; d++)
                    memcpy( (*rimg)[d], (*rimg)[d] +
                            rwidth*(maxbandheight-(corHeight-1)),
```

```
                        rwidth*(corHeight-1)*sizeof(rectimg_t) );
                /* rectify the remaining */
                steShiftRect(rect, 0, ystart+(corHeight-1), 0, 0);
                for(d=0; d<depth; d++)
                    steRectify(images[d], width, height, rect,
                            (*rimg)[d]+rwidth*(corHeight-1), rwidth,
                            bandheight-(corHeight-1), 127.0, STEREO_LINEAR);
            } else {
                /* note that there could be a case for
                   bandid == (*lastbandid - 1), but it would not be used in
                   the present implementation since bandids are always
                   increasing */
                steShiftRect(rect, 0, ystart, 0, 0);
                for(d=0; d<depth; d++)
                    steRectify(images[d], width, height, rect,
                            (*rimg)[d], rwidth,
                            bandheight, 127.0, STEREO_LINEAR);
            }
        }
    } else {
        if(*rimg == NULL)
            *rimg = (rectimg_t**)malloc(depth*sizeof(rectimg_t*));
        for(d=0; d<depth; d++)
            (*rimg)[d] = rectifiedImages[d] + rwidth*ystart;
    }
}

/*
 * prepare the next band
 *
 * prepare the next band that should be sent (bandheight and pointers
 * to rectified image bands). If rectifiedImage[12] is NULL, then only
 * a band of the rectified images is computed and put in rimg1 and
 * rigm2 (if these are pointers to NULL, these areas are
 * allocated).
 */
static void
prepareBand( int bandid,
             int *lastbandid,   /* the previous band that was sent or -1 */
             int maxbandheight, /* maximum height of an image band */
             int corHeight,     /* height of the correlation window */
             unsigned char **images1,
             int width1,        /* dimensions of first image */
             int height1,
             int depth,
             unsigned char **images2,
             int width2,        /* dimension of second image */
             int height2,
             const double *rectification1,
             rectimg_t **rectifiedImages1, /* can be NULL */
             int rwidth1, /* dimensions of 1st rectified image */
             int rheight1,
             const double *rectification2,
             rectimg_t **rectifiedImages2, /* can be NULL */
             int rwidth2,       /* dimensions of 2nd rectified image */
             int rheight2,
             int *bandheight,   /* output: bandheight */
             rectimg_t ***rimg1,        /* output: rectified image bands */
             rectimg_t ***rimg2)
{
    int ystart;

    ystart = bandid*(maxbandheight - (corHeight-1));
    assert((ystart < rheight1) && (ystart < rheight2));
    *bandheight = maxbandheight;

    /* adjust banheight */
    if((ystart + *bandheight) >= rheight1)
        *bandheight = rheight1 - ystart;
    if((ystart + *bandheight) >= rheight2)
        *bandheight = rheight2 - ystart;
    assert(*bandheight>=corHeight);

#ifdef DEBUG
    fprintf(stderr,"master preparing band ID %d, beginning at %d with \
height %d\n",bandid,ystart,*bandheight);
#endif

    rectifyOneImage( bandid, *lastbandid, maxbandheight, ystart, corHeight,
                     images1, width1, height1, depth, rectification1,
                     rectifiedImages1, rwidth1, *bandheight, rimg1 );
    rectifyOneImage( bandid, *lastbandid, maxbandheight, ystart, corHeight,
                     images2, width2, height2, depth, rectification2,
                     rectifiedImages2, rwidth2, *bandheight, rimg2 );
    *lastbandid = bandid;       /* the band which is ready to be sent */
}

static void
sendBand( int tid,              /* task ID of the slave task */
          int encoding,
          int bandid,
          int bandheight,
          rectimg_t **rimg1,
          int rwidth1,
          int depth,
          rectimg_t **rimg2,
          int rwidth2 )
{
    int d;

    /* send the image bands */
#ifdef DEBUG
    fprintf(stderr,"master sending band ID %d\n",bandid);
#endif
    pvm_initsend( encoding );
    pvm_pkint(&bandid, 1, 1); /* the disparity band ID */
    pvm_pkint(&bandheight, 1, 1);
#ifdef STEREO_RECTFLOAT
    for( d=0; d<depth; d++)
        pvm_pkfloat(rimg1[d], rwidth1*bandheight, 1);
#else
    for( d=0; d<depth; d++)
        pvm_pkbyte(rimg1[d], rwidth1*bandheight, 1);
#endif
    for( d=0; d<depth; d++)
        pvm_pkfloat(rimg2[d], rwidth2*bandheight, 1);
    pvm_send( tid, STEREO_PVM_IMAGES );
}

/* wait for the next chunk of disparity map */
static int
receiveDisp( int *tid,          /* task ID where the band comes from */
             int maxbandheight,
             int corHeight,
             float *disparity12,
             int rwidth1,
             int rheight1 )
{
    int bufid, bytes, msgtag, bandid, disparitysize, ystart;

    bufid = pvm_recv( -1, -1 );
    if( pvm_bufinfo(bufid, &bytes, &msgtag, tid) < 0 ) {
        pvm_perror("receiveDisp");
        return ERROR;
    }
    switch( msgtag ) {
    case STEREO_PVM_DISPARITY:
        pvm_upkint(&bandid, 1, 1); /* the disparity band ID */
        pvm_upkint(&disparitysize, 1, 1); /* the size of the buffer */
        ystart = corHeight/2 + bandid*(maxbandheight - (corHeight-1));
        assert((ystart>=0) && (ystart<rheight1));
        pvm_upkfloat(disparity12 + ystart*rwidth1, disparitysize, 1);
#ifdef DEBUG
        fprintf(stderr,"master received disparity band %d\n",bandid);
#endif
        return OK;
    case STEREO_PVM_SLAVE_EXIT:
        pvm_upkint(tid, 1, 1);
        fprintf(stderr, "receiveDisp: slave task %x exited, can't continue\n",
                *tid);
        return ERROR;
    default:
        fprintf(stderr,"receiveDisp: master received unknown message tag \
(%d) from task %x\n", msgtag, *tid);
        return ERROR;
    }
}

#endif /* HAVE_PVM */

/*
 * rectify images and compute the disparity map
 *
 * This function can be called several times for images of the same size
 * between calls to steCorPVMInit and steCorPVMTerminate.
 *
 * AUTHOR: Frederic Devernay
 * (C) INRIA 1993-96
 */

int
steCorPVMMaster(
                /* PVM-specific parameters */
                int nslave,
                int *tid,       /* task IDs of slaves */
                int encoding,
                int nbbands,    /* Number of bands. */
                /* remaining parameters are those of steCorDriver() */
                unsigned char **images1, /* 1st image or NULL */
                int width1,     /* dimensions of first image */
                int height1,
                int depth,      /* depth of image1 */
                unsigned char **images2, /* 2nd image or NULL */
                int width2,     /* dimension of second image */
                int height2,
                const double *rectification1, /* 1st rectification matrix
                                   (normalized w/ respect to image
                                   coordinates, i.e. it transforms a
                                   width1Xheight1 image, not a
                                   rwidth1Xrheight1 one, into a
                                   width1Xheight1 image). This
                                   transformation is from rectified
                                   coords to original coords! NULL
                                   means identity (just resize image1) */
                rectimg_t **rectifiedImages1, /* input (if image1==NULL):
                                   1st rectified rwidth1Xrheight1
                                   image.  (if image1!=NULL:
                                   rwidth1Xrheight1 workspace or NULL
                                   to allocate it output: 1st
                                   rectified image (mean-filtered if
                                   meanfilter!=0).  The points outside
                                   of the original image will be set
                                   to 127.0 */
                int rwidth1,    /* dimensions of 1st rectified image */
                int rheight1,
                const double *rectification2, /* 2nd rectification matrix */
                rectimg_t **rectifiedImages2, /* input (if image2==NULL):
                                   2nd rectified rwidth2Xrheight2
                                   image.  output: 2nd rectified image
                                   (mean-filtered if meanfilter!=0) */
                int rwidth2,    /* dimensions of 2nd rectified image */
                int rheight2,
                int flags,      /* constructed by OR-ing Flags from
                                   the following list: STEREO_ZEROMEAN
                                   (mean-filter rectified images).
                                   STEREO_NORMALIZE (use a normalized
                                   correlation function, i.e. divide
                                   it by the local image norm).
                                   STEREO_SSD or STEREO_CC
                                   (correlation function to be used,
                                   either Sum of Squared Differences
                                   or Cross-Correlation).  STEREO_BACK
                                   (perform back-correlation and
                                   validation) */
                double dispNear, /* disparity corresponding to near
                                   points for a width1Xheight1
                                   image. */
                double dispFar, /* disparity for far points, for a
                                   width1Xheight1 image. The error
                                   value (which is disparity[0]) will
                                   be an integer beyond
                                   dispFar*rwidth1/width1.  The
                                   disparity interval effectively used
                                   will be the biggest possible
```

```
                                included in [dispNear,dispFar]
                                (scaled to rectified image
                                size). */
                int corWidth,   /* width of the correlation window (>=1) */
                int corHeight,  /* height of the correlation window (>=1) */
                float *disparity12,/* input: rwidth1Xrheight1 workspace,
                                or NULL if only rectification is
                                wanted (in that case,
                                rectifiedImage1 or rectifiedImage2
                                must be non-NULL.  output:
                                disparity from 1st to 2nd image */
                float *scoremin) /* input: max(rwidth1Xrheight1,
                                rwidth2Xrheight2) workspace, will
                                be allocated if NULL */
{
#ifndef HAVE_PVM
    fprintf(stderr,"steCorPVMMaster: Warning: PVM not enabled, using \
steCorDriver instead.\n");
    steCorDriver(images1, width1, height1, depth, images2, width2, height2,
                 rectification1, rectifiedImages1, rwidth1, rheight1,
                 rectification2, rectifiedImages2, rwidth2, rheight2,
                 flags, dispNear, dispFar, corWidth, corHeight,
                 disparity12, scoremin);
    return ERROR;
#else
    double rect1[9], rect2[9];
    int dispNearI, dispFarI;
    int d;

    /* check the input parameters */
    if(((images1!=NULL) && ((width1<=0) || (height1<=0))) ||
       ((images2!=NULL) && ((width2<=0) || (height2<=0))) ||
       (depth <= 0) ||
       (rwidth1<=0) || (rheight1<=0) ||
       (rwidth2<=0) || (rheight2<=0) ||
       ((disparity12 != NULL) && ((corWidth == 0) || (corHeight == 0))) ||
       ((disparity12==NULL)&&(rectifiedImages1==NULL)
        &&(rectifiedImages2==NULL))) {
        fprintf(stderr, "steCorPVMMaster: invalid input parameters\n");
        return ERROR;
    }
    if (scoremin!=NULL) {
        fprintf(stderr, "steCorPVMMaster: score image doesn't work yet\n");
        return ERROR;
    }

    /* rectification */
    if((images1 != NULL) && (*images1 != NULL)) {/* we have to rectify it */
        if( ((rwidth1 != width1) || (rheight1 != height1))) {
            memcpy(rect1, rectification1, 9*sizeof(double));
            steResizeRect( rect1, width1, height1, width1, height1,
                           rwidth1, rheight1, width1, height1 );
            rectification1 = (double*)rect1;
        }
        if(rectifiedImages1 != NULL) {
            for( d=0; d<depth; d++) {
                if(rectifiedImages1[d] == NULL)
                    rectifiedImages1[d]=(rectimg_t*)malloc(rwidth1*rheight1
                                                           *sizeof(rectimg_t));
                steRectify(images1[d], width1, height1, rectification1,
                           rectifiedImages1[d], rwidth1, rheight1,
                           127.0, STEREO_LINEAR);
            }
        }
    } else if(rectifiedImages1 == NULL) {
        fprintf(stderr, "steCorPVMMaster: invalid input parameters\n");
        return ERROR;
    }
    if((images2 != NULL) && (*images2 != NULL)) { /* we have to rectify it */
        if( ((rwidth2 != width2) || (rheight2 != height2))) {
            memcpy(rect2, rectification2, 9*sizeof(double));
            steResizeRect( rect2, width2, height2, width2, height2,
                           rwidth2, rheight2, width2, height2 );
            rectification2 = (double*)rect2;
        }
        if(rectifiedImages2 != NULL) {
            for( d=0; d<depth; d++) {
                if(rectifiedImages2[d] == NULL)
                    rectifiedImages2[d]=(rectimg_t*)malloc(rwidth2*rheight2
                                                           *sizeof(rectimg_t));
                steRectify(images2[d], width2, height2, rectification2,
                           rectifiedImages2[d], rwidth2, rheight2,
                           127.0, STEREO_LINEAR);
            }
        }
    } else if(rectifiedImages2 == NULL) {
        fprintf(stderr, "steCorPVMMaster: invalid input parameters\n");
        return ERROR;
    }
    /* correlation */
    if(disparity12 != NULL) {
        int maxbandheight;      /* height of a band
                                   (remember: the bands overlap) */
        int bandssent;          /* total number of bands sent */
        int bandsreceived;      /* number of bands received */
        int bandsbeingprocessed; /* number of bands being processed */
        int lastbandid;         /* the last band sent
                                   (used to optimize rectification) */
        rectimg_t **rimg1 = NULL, **rimg2 = NULL;

        /* we must have (rheight1-(corHeight-1)) <=
           (bandheight-(corHeight-1))*nbbands: */
        /* i.e. all the bands joined together must cover the image */
        maxbandheight = (rheight1 - corHeight)/nbbands + corHeight;
        assert(rheight1
               <= (maxbandheight-(corHeight-1))*nbbands+(corHeight-1));
        if((maxbandheight-(corHeight-1))*(nbbands-1)> rheight1-corHeight) {
            /* adjust the number of bands so that all the bands are
               inside the image and are at least corHeight high */
            nbbands = (rheight1-corHeight)/(maxbandheight-(corHeight-1))+1;
            assert((maxbandheight-(corHeight-1))*(nbbands-1)
                   <= rheight1-corHeight);
            assert(rheight1
                   <= (maxbandheight-(corHeight-1))*nbbands+(corHeight-1));
        }
        /* compute dispMin, dispMax, and dispError */
        dispNear = rwidth1*dispNear/(double)width1;
        dispFar = rwidth1*dispFar/(double)width1;
        if(dispNear<dispFar) {
            dispNearI = ceil(dispNear-0.5);
            dispFarI = floor(dispFar+0.5);
        } else {
            dispFarI = ceil(dispFar-0.5);
            dispNearI = floor(dispNear+0.5);
        }
        /* send parameters to slave tasks */
        pvm_initsend( encoding );
        /* first rectified image */
        pvm_pkint(&rwidth1, 1, 1);
        pvm_pkint(&maxbandheight, 1, 1);
        /* depth for both images */
        pvm_pkint(&depth, 1, 1);
        /* second rectified image */
        pvm_pkint(&rwidth2, 1, 1);
        pvm_pkint(&maxbandheight, 1, 1);
        /* other parameters */
        pvm_pkint(&flags, 1, 1);
        pvm_pkint(&dispNearI, 1, 1);
        pvm_pkint(&dispFarI, 1, 1);
        pvm_pkint(&corWidth, 1, 1);
        pvm_pkint(&corHeight, 1, 1);
        /* should we use pvm_bcast instead? it would be more costly */
        if( pvm_mcast(tid, nslave, STEREO_PVM_PARAMS) < 0 ) {
            pvm_perror( "steCorPVMMaster" );
            return ERROR;
        }

        /* fill in the first and last few lines of the disparity map */
        {
            int i;

            for( i=0; i<(corHeight/2)*rwidth1; i++) {
                disparity12[i] = dispFarI;
            }
            for( i=(rheight1-(corHeight-1)/2)*rwidth1; i<rheight1*rwidth1;
                 i++) {
                disparity12[i] = dispFarI;
            }
        }

        lastbandid = -2;        /* no bands were sent */
        bandssent = 0;
        bandsbeingprocessed = 0;
        bandsreceived = 0;

        /* now send the bands and wait for them! */
        do {
            int slavetid;
            int bandheight;

#ifdef DEBUG
            fprintf(stderr,"bands sent = %d, bands being processed = \
%d, bands received = %d\n",bandssent, bandsbeingprocessed, bandsreceived);
#endif
            /* prepare the next band to send */
            if(bandssent < nbbands)
                prepareBand(bandssent, &lastbandid, maxbandheight, corHeight,
                            images1, width1, height1, depth,
                            images2, width2, height2,
                            rectification1, rectifiedImages1,
                            rwidth1, rheight1,
                            rectification2, rectifiedImages2,
                            rwidth2, rheight2,
                            &bandheight, &rimg1, &rimg2);

            /* wait for a slave to send back its results */
            if( (bandsbeingprocessed >= nslave) ||
                (bandssent >= nbbands) ) {
                if( receiveDisp( &slavetid, maxbandheight, corHeight,
                                 disparity12, rwidth1, rheight1 ) != OK )
                    return ERROR;
                bandsreceived++;
                bandsbeingprocessed--;
            } else {
                slavetid = tid[bandssent]; /* this slave will receive
                                              the next band */
            }
            /* if we still have bands to process... */
            if( bandssent < nbbands ) {
                /* send a new band to that free task */
                sendBand( slavetid, encoding, bandssent, bandheight,
                          rimg1, rwidth1, depth, rimg2, rwidth2);
                bandssent++;
                bandsbeingprocessed++;
            }
        } while(bandsreceived < nbbands);

        /* free allocated memory */
        if(rectifiedImages1 == NULL) {
            for(d=0; d<depth; d++)
                free(rimg1[d]);
            free(rimg1);
        }
        if(rectifiedImages2 == NULL) {
            for(d=0; d<depth; d++)
                free(rimg2[d]);
            free(rimg2);
        }
    } /* end of correlation */

    return OK;
#endif /* HAVE_PVM */
}

/*
 * Send a termination signal to all slave PVM tasks
 *
 * AUTHOR: Frederic Devernay
 * (C) INRIA 1993-96
 */
int
steCorPVMTerminate( int nslave,
                    int *tid,
                    int encoding
```

```
                    )
{
#ifndef HAVE_PVM
    fprintf(stderr,"steCorPVMExit: PVM not enabled.\n");
    return ERROR;
#else
    int terminate_bandid = -1;  /* a negative bandid means terminate */

    pvm_initsend(encoding);
    pvm_pkint(&terminate_bandid, 1, 1);
    if ( pvm_mcast(tid, nslave, STEREO_PVM_IMAGES) < 0 ) {
        pvm_perror( "steCorPVMTerminate" );
        return ERROR;
    }
    return OK;
#endif /* HAVE_PVM */
}
```

E.3.3 Esclave

```
/*
 * stereo by correlation using PVM, slave part
 *
 * see steCorPVM.c for comments
 *
 * AUTHOR: Frederic Devernay
 * (C) INRIA 1993-96
 */

#ifdef HAVE_CONFIG_H
#include "config.h"
#endif
#include "RstereoP.h"

#ifdef HAVE_PVM                 /* all this only works with PVM */
#include <pvm3.h>
#endif

#define DEBUG

/* extern functions defined in this file: */
int steCorPVMSlave( int, int );

#ifdef HAVE_PVM

static int
receiveParams( int parent,
               int *rwidth1, int *rheight1,
               int *depth,
               int *rwidth2, int *rheight2,
               int *flags,
               int *dispNear, int *dispFar,
               int *corWidth,
               int *corHeight )
{
    int bufid, bytes, msgtag, tid;

    bufid = pvm_recv( -1, -1 );
    if( pvm_bufinfo(bufid, &bytes, &msgtag, &tid) < 0 ) {
        pvm_perror("receiveParams");
        return ERROR;
    }
    switch( msgtag ) {
    case STEREO_PVM_PARAMS:
        /* these parameters are the same for all bands, they are multicasted */
        /* first rectified image */
        pvm_upkint(rwidth1, 1, 1);
        pvm_upkint(rheight1, 1, 1);
        /* depth for both images */
        pvm_upkint(depth, 1, 1);
        /* second rectified image */
        pvm_upkint(rwidth2, 1, 1);
        pvm_upkint(rheight2, 1, 1);
        /* other parameters */
        pvm_upkint(flags, 1, 1);
        pvm_upkint(dispNear, 1, 1);
        pvm_upkint(dispFar, 1, 1);
        pvm_upkint(corWidth, 1, 1);
        pvm_upkint(corHeight, 1, 1);
        if( tid != parent ) {
            fprintf(stderr,"receiveParams: received STEREO_PVM_PARAMS from \
task %x instead of master task %x\n", tid, parent);
            return ERROR;
        }
        return OK;
    case STEREO_PVM_MASTER_EXIT:
        pvm_upkint(&parent, 1, 1);
        fprintf(stderr, "receiveParams: master task %x exited, can't \
continue\n", parent);
        return ERROR;
    default:
        fprintf(stderr,"receiveParams: slave received unknown message tag \
(%d) from master task %x\n", msgtag, parent);
        return ERROR;
    }
}

static int
receiveImages( int parent,
               int *bandid,
               int *bandheight,
               rectimg_t **rectifiedImages1,
               int rwidth1,
               int rheight1,
               int depth,
               rectimg_t **rectifiedImages2,
               int rwidth2,
               int rheight2 )
{
    int bufid, bytes, msgtag, tid;
    int d;

    bufid = pvm_recv( -1, -1 );
    if( pvm_bufinfo(bufid, &bytes, &msgtag, &tid) < 0 ) {
        pvm_perror("receiveImages");
        return ERROR;
    }
    switch( msgtag ) {
    case STEREO_PVM_IMAGES:
        pvm_upkint(bandid, 1, 1); /* the band ID */
        if( *bandid < 0 )       /* terminating condition */
            return OK;
        pvm_upkint(bandheight, 1, 1); /* the band height in pixels */
        assert((*bandheight<=rheight1) && (*bandheight<=rheight2));
#ifdef STEREO_RECTFLOAT
        for( d=0; d<depth; d++)
            pvm_upkfloat(rectifiedImages1[d], rwidth1*(*bandheight), 1);
        for( d=0; d<depth; d++)
            pvm_upkfloat(rectifiedImages2[d], rwidth2*(*bandheight), 1);
#else
        for( d=0; d<depth; d++)
            pvm_upkbyte(rectifiedImages1[d], rwidth1*(*bandheight), 1);
        for( d=0; d<depth; d++)
            pvm_upkbyte(rectifiedImages2[d], rwidth2*(*bandheight), 1);
#endif
#ifdef DEBUG
        fprintf(stderr,"slave received images for band %d\n",*bandid);
#endif
        if( tid != parent ) {
            fprintf(stderr,"receiveImages: received STEREO_PVM_IMAGES from \
task %x instead of master task %x\n", tid, parent);
            return ERROR;
        }
        return OK;
    case STEREO_PVM_MASTER_EXIT:
        pvm_upkint(&parent, 1, 1);
        fprintf(stderr, "receiveImages: master task %x exited, can't \
continue\n", parent);
        return ERROR;
    default:
        fprintf(stderr,"receiveImages: master received unknown message \
tag (%d) from task %x\n", msgtag, parent);
        return ERROR;
    }
}

#endif /* HAVE_PVM */

/*
 * Execute a slave task for stereo by correlation
 *
 * This function runs correlation when the master tasks calls
 * steCorPVMMaster() and terminates when the master task calls
 * steCorPVMTerminate().
 *
 * AUTHOR: Frederic Devernay
 * (C) INRIA 1993-96
 */
int
steCorPVMSlave( int mytid,      /* my task ID */
                int encoding )  /* PVM encoding type */
{
    int parent;                 /* PVM parent */
    int rwidth1, rheight1;
    int depth, d;
    rectimg_t **rectifiedImages1;
    int rwidth2, rheight2;
    rectimg_t **rectifiedImages2;
    int flags;
    int dispNear, dispFar;
    int corWidth, corHeight;
    float *scoremin;
    float *disparity12;
    int disparityoffset, disparitysize;
    int bandid;

#ifndef HAVE_PVM
    fprintf(stderr,"steCorPVMExit: PVM not enabled.\n");
    return ERROR;
#else
    parent = pvm_parent();
    if(parent < 0) {
        pvm_perror( "steCorPVMSlave" );
        return ERROR;
    }
    if( receiveParams( parent, &rwidth1, &rheight1, &depth,
                       &rwidth2, &rheight2,
                       &flags, &dispNear, &dispFar,
                       &corWidth, &corHeight ) != OK )
        return ERROR;
    /* allocate memory for the rectified images */
    rectifiedImages1 = (rectimg_t**)malloc( depth*sizeof(rectimg_t*) );
    rectifiedImages2 = (rectimg_t**)malloc( depth*sizeof(rectimg_t*) );
    for( d=0; d<depth; d++ ) {
        rectifiedImages1[d] = (rectimg_t*)malloc( rwidth1*rheight1
                                                  *sizeof(rectimg_t) );
        rectifiedImages2[d] = (rectimg_t*)malloc( rwidth2*rheight2
                                                  *sizeof(rectimg_t) );
    }
    /* allocate memory for the result */
    disparity12 = (float*)malloc(rwidth1*rheight1*sizeof(float));
    assert(disparity12 != NULL);
    scoremin = (float*)malloc(rwidth1*rheight1*sizeof(float));
    assert(scoremin != NULL);

    disparityoffset = rwidth1*(corHeight/2);

    while(1) {                  /* main slave loop */
        int bandheight;

        if( receiveImages( parent, &bandid, &bandheight,
                           rectifiedImages1, rwidth1, rheight1, depth,
                           rectifiedImages2, rwidth2, rheight2 ) != OK )
            return ERROR;
        if( bandid < 0 )        /* terminating condition */
            break;              /* exit the main loop */
        steCorrelation(rectifiedImages1, rwidth1, bandheight, depth,
                       rectifiedImages2, rwidth2, bandheight, flags,
                       dispNear, dispFar,
                       corWidth, corHeight,
                       disparity12, NULL);
```

```
        /* send back the disparity map */
        pvm_initsend( encoding );
        pvm_pkint(&bandid, 1, 1); /* the disparity band ID */
        disparitysize = rwidth1*(bandheight-corHeight+1);
        pvm_pkint(&disparitysize, 1, 1); /* the size of the disparity buffer */
        pvm_pkfloat(disparity12 + disparityoffset, disparitysize, 1);
        pvm_send(parent, STEREO_PVM_DISPARITY);
#ifdef DEBUG
        fprintf(stderr,"slave sent disparity for band %d\n",bandid);
#endif
    }
    /* free buffers */
    for( d=0; d<depth; d++ ) {
        free(rectifiedImages1[d]);
        free(rectifiedImages2[d]);
    }
    free(rectifiedImages1);
    free(rectifiedImages2);
    free(disparity12);
    free(scoremin);

    return OK;
#endif /* HAVE_PVM */
}
```

Glossaire

birapport : soient quatre points a,b,c,d de $\mathcal{P}^1$, de paramètres projectifs α_a,α_b,α_c,α_c. Le birapport (a,b,c,d) est défini par et il est indépendant du choix de la paramétrisation de $\mathcal{P}^1$ [Fau93]. Le birapport de quatre droites concourrantes de $\mathcal{P}^2$ est égal au birapport de leurs points d'intersection avec une droite quelconque de $\mathcal{P}^2$.

calibrage : problème qui consiste, étant donné un ensemble de points de contrôle de coordonnées 3-D connues (X_i,Y_i,Z_i), à déterminer les paramètres de la fonction de projection associée au capteur pour que leur projection correspondent au mieux aux mêmes points extraits des images (x_i,y_i).

centre optique : (noté C) centre de projection dans un modèle de caméra perspectif (voir figure 1.1, page 5).

distance focale : (notée f) distance entre le centre optique C et le point principal c dans un modèle perspectif.

droite épipolaire : projection du rayon optique joignant le centre optique d'une caméra à un point de l'espace dans une autre caméra

plan focal : (noté Π^f) plan parallèle au plan rétinien Π^r passant par le centre optique C (voir figure 1.1, page 5).

plan rétinien : (noté Π^r) plan de l'espace sur lequel se forme l'image par projection des points de l'espace par rapport au centre optique d'une caméra projective (voir figure 1.1, page 5).

point principal : (noté c) projection orthogonale du centre optique C sur le plan rétinien Π^r (voir figure 1.1, page 5).

rayon optique : droite passant par le centre optique C, un point de l'espace M, et sa projection m sur le plan rétinien Π^r (voir figure 1.1, page 5).

épipôle : projection dans une image du centre optique d'une autre caméra. C'est aussi le point par lequel passent toutes les droites épipolaires associées à cette autre caméra.

Index

Bibliographie

[AAK71] Y.I. Abdel-Aziz et H.M. Karara, Direct linear transformation into object space coordinates in close-range photogrammetry, in *Proceedings of the Symposium on Close-Range Photogrammetry, University of Illinois at Urbana-Champaign, Urbana, Illinois*, pages 1–18, janvier 1971.

[ABB+94] E. Anderson, Z. Bai, C. Bishof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov et D. Sorensen, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, 3600 University City Science Center, Philadelphia, PA 19104-2688, édition second, 1994.

[AF87] N. Ayache et B. Faverjon, Efficient registration of stereo images by matching graph descriptions of edge segments, *The International Journal of Computer Vision* **1**,2 (avril 1987).

[AG92] P. Aschwanden et W. Guggenbühl, Experimental results from a comparative study on correlation-type registration algorithms, in *ISPRS Workshop*, Bonn, Germany, 1992.

[AH88] N. Ayache et C. Hansen, Rectification of images for binocular and trinocular stereovision, in *International Conference on Pattern Recognition*, octobre 1988. 9th, Beijing, China.

[Bai96] Éric Bainville, *Modélisation géométrique et dynamique d'un geste chirurgical*, PhD thesis, Université Joseph Fourier, mars 1996.

[BB81] H.H. Baker et T.O. Binford, Depth from edge- and intensity-based stereo, in *Proceedings 7th Joint Conference on Artificial Intelligence, Vancouver, Canada*, pages 631–636, août 1981.

[BBH93] R.C. Bolles, H.H. Baker et M.J. Hannah, The JISCT stereo evaluation, in *Proceedings of the ARPA Image Understanding Workshop*, Defense Advanced Research Projects Agency, Washington, DC, avril 1993, pages 263–274, Morgan Kaufmann Publishers, Inc.

[Bel93] Peter N. Belhumeur, A binocular stereo algorithm for reconstructing sloping, creased, and broken surfaces in the presence of half-occlusion, in *Proceedings of the 4th International Conference on Computer Vision*, Berlin, Germany, mai 1993, IEEE Computer Society Press.

[Bel96] P. N. Belhumeur, A bayesian approach to binocular stereopsis, *The International Journal of Computer Vision*, 1996.

[Ber84] V. Berzins, Accuracy of laplacian edge detectors, *Computer Vision, Graphics, and Image Processing* **27** (1984), 195–210.

[Bey92] Horst A. Beyer, Accurate calibration of CCD-cameras, in *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society, Urbana Champaign, IL, juin 1992, IEEE.

[BF82] S. T. Barnard et M. A. Fishler, Computational stereo, *ACM Computing Surveys* **14**,4 (décembre 1982), 553–572.

[BFAD95] Fabienne Betting, Jacques Feldmar, Nicholas Ayache et Frédéric Devernay, A new framework for fusing stereo images with volumetric medical images, in *First International Conference on Computer Vision, Virtual Reality and Robotics in Medicine*, Springer-Verlag (édité par), Nice, France, avril 1995.

[BG88] Marcel Berger et Bernard Gostiaux, *Differential Geometry: Manifolds, Curves, and Surfaces*, *Graduate Texts in Mathematics* numéro 115, Springer–Verlag, 1988.

[BL93] Anup Basu et Sergio Licardie, Modeling fish-eye lenses, in *Proceedings of the International Robotics and Systems Conference*, M. Kikode, T. Sato et K. Tatsuno (édité par), IEEE, Yokohama, Japan, juillet 1993, volume 3, pages 1822–1828.

[BM92] Peter N. Belhumeur et David Mumford, A bayesian treatment of the stereo correspondence problem using half-occluded regions, in *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society, Urbana Champaign, IL, juin 1992, IEEE.

[BMB93] P. Brand, R. Mohr et P. Bobet, Distorsions optiques : correction dans un modèle projectif, rapport technique numéro 1933, LIFIA–INRIA Rhône-Alpes, 1993.

[BMZ92] P. Beardsley, D. Murray et A. Zissermann, Camera calibration using multiple images, in *Proceedings of the 2nd European Conference on Computer Vision*, G. Sandini (édité par), Santa Margherita, Italy, mai 1992, pages 312–320, Springer–Verlag.

[Bou90] Jean-Pierre Bourguignon, *Calcul Variationnel*, École Polytechnique, 1990.

[BPYH85] Michael Brady, Jean Ponce, Alan Yuille et Asada H., Describing surfaces, *Computer Vision, Graphics, and Image Processing* **32** (1985), 1–28.

[Bro71] Duane C. Brown, Close-range camera calibration, *Photogrammetric Engineering* **37**,8 (1971), 855–866.

[BZ87] A. Blake et A. Zisserman, *Visual Reconstruction*, MIT Press, 1987.

[BZ95] Paul A. Beardsley et Andrew Zisserman, Affine calibration of mobile vehicles, in *Proceedings of Europe-China Workshop on Geometrical Modeling and Invariants for Computer Vision*, R. Mohr et C. Wu (édité par), Xi'an, China, avril 1995, pages 214–221, Xidian University Press.

[BZM94] Paul Beardsley, Andrew Zisserman et David Murray, Navigation using affine structure from motion, in *Proceedings of the 3rd European Conference on Computer Vision*, J-O. Eklundh (édité par), Stockholm, Sweden, mai 1994, pages 85–96, *Lecture Notes in Computer Science* volume 2, Springer–Verlag.

[Can83] J.F. Canny, Finding edges and lines in images, rapport technique numéro AI-TR-720, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, juin 1983.

[Can86] J. F. Canny, A computational approach to edge detection, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **8**,6 (novembre 1986), 769–798.

[Car92] Elie Cartan, *La Théorie des Groupes Finis et Continus et la Géométrie Différentielle traitée par la Méthode du Repère Mobile*, Jacques Gabay, 1992. Original edition, Gauthiers-Villars, 1937.

[CC90] Chienchung Chang et Shankar Chatterjee, Multiresolution stereo – a bayesian approach, in *Proceedings of the 10th International Conference on Pattern Recognition*, Atlantic City, N.J., octobre 1990, pages 908–912, Computer Society Press.

[CCK91] Chienchung Chang, Shankar Chatterjee et Paul R. Kube, On an analysis of static occlusion in stereo vision, in *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, Lahaina, Hawai, juin 1991, pages 722–723, IEEE.

[Csu96] Gabriella Csurka, *Modelisation projective des objets tridimensionnels en vision par ordinateur*, PhD thesis, Université de Nice Sophia Antipolis, avril 1996.

[CT90] Bruno Caprile et Vincent Torre, Using Vanishing Points for Camera Calibration, *The International Journal of Computer Vision* **4**,2 (mars 1990), 127–140.

[CVSG89] L. Cohen, L. Vinet, P.T. Sander et A. Gagalowicz, Hierarchical region based stereo matching, in *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, San Diego, CA, juin 1989, pages 416–421, Computer Society Press.

[CZZF95] Gabriella Csurka, Cyril Zeller, Zhengyou Zhang et Olivier Faugeras, Characterizing the uncertainty of the fundamental matrix, rapport technique numéro 2560, INRIA, 1995.

[DA89] Umesh R. Dhond et J.K. Aggarwal, Structure from stereo - a review, *IEEE Transactions on Systems, Man, and Cybernetics* **19**,6 (1989), 1489–1510.

[DB93a] R. Deriche et T. Blaszka, Recovering and characterizing image features using an efficient model based approach, in *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society, New-York, juin 1993, pages 530–535, IEEE.

[DB93b] F. Du et M. Brady, Self-calibration of the intrinsic parameters of cameras for active vision systems, in *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society, New-York, NY, juin 1993, pages 477–482, IEEE.

[DDL94] N. Daucher, M. Dhome et J. T. Lapresté, Camera calibration from sphere images, in *Proceedings of the 3rd European Conference on Computer Vision*, J-O. Eklundh (édité par), Stockholm, Sweden, mai 1994, *Lecture Notes in Computer Science* volume 800-801, Springer–Verlag.

[Der87] R. Deriche, Using Canny's criteria to derive a recursively implemented optimal edge detector, *The International Journal of Computer Vision* **1**,2 (mai 1987), 167–187.

[Der90] R. Deriche, Fast algorithms for low-level vision, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **1**,12 (janvier 1990), 78–88.

[Der93] R. Deriche, Recursively implementing the gaussian and its derivatives, rapport technique numéro 1893, INRIA, Unité de Recherche Sophia-Antipolis, 1993.

[Dev93] Frédéric Devernay, A fast and efficient subpixelic edge detector, in *Quatrièmes Journées Orasis*, Mulhouse, France, octobre 1993.

[Dev95] Frédéric Devernay, A non-maxima suppression method for edge detection with sub-pixel accuracy, rr numéro 2724, INRIA, novembre 1995.

[DF94a] Frédéric Devernay et Olivier Faugeras, Computing differential properties of 3-D shapes from stereoscopic images wi-

thout 3-D models, in *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society, Seattle, WA, juin 1994, pages 208–213, IEEE.

[DF94b] Frédéric Devernay et Olivier D. Faugeras, Computing differential properties of 3-D shapes from stereoscopic images without 3-D models, rapport technique numéro 2304, INRIA, juillet 1994.

[DF95a] Frédéric Devernay et Olivier Faugeras, Automatic calibration and removal of distortion from scenes of structured environments, in *Investigative and Trial Image Processing*, Leonid I. Rudin et Simon K. Bramble (édité par), SPIE, San Diego, CA, juillet 1995, *Proceedings of the Proceedings of the International Society for Optical Engineering* volume 2567.

[DF95b] Frédéric Devernay et Olivier Faugeras, From projective to euclidean reconstruction, rr numéro 2725, INRIA, novembre 1995.

[DF96] Frédéric Devernay et Olivier Faugeras, From projective to euclidean reconstruction, in *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society, San Francisco, CA, juin 1996, pages 264–269, IEEE.

[DG90] R. Deriche et G. Giraudon, Accurate corner detection: An analytical study, in *Proceedings of the 3rd International Conference on Computer Vision*, Osaka, Japan, décembre 1990, pages 66–70, IEEE Computer Society Press.

[DG93] R. Deriche et G. Giraudon, A computational approach for corner and vertex detection, *The International Journal of Computer Vision* **10**,2 (1993), 101–124.

[DLDL96] C. Delherm, J.M. Lavest, M. Dhome et J.T. Lapresté, Dense reconstruction by zooming, in *Proceedings of the 4th European Conference on Computer Vision*, Bernard Buxton (édité par), Cambridge, UK, avril 1996, volume 2, pages 427–438.

[DVF91] Rachid Deriche, Régis Vaillant et Olivier Faugeras, From noisy edge points to 3D reconstruction of a scene: A robust approach and its uncertainty analysis, in *Proceedings of the 7th Scandinavian Conference on Image Analysis*, Alborg, Denmark, août 1991, pages 225–232.

[DVF92] R. Deriche, R. Vaillant et O. Faugeras, *From Noisy Edges Points to 3D Reconstruction of a Scene: A Robust Approach and Its Uncertainty Analysis*, volume 2, pages 71–79, World Scientific, 1992. Series in Machine Perception and Artificial Intelligence.

[DZLF94] R. Deriche, Z. Zhang, Q.T. Luong et O. Faugeras, Robust recovery of the epipolar geometry for an uncalibrated stereo rig, in *Proceedings of the 3rd European Conference on Computer Vision*, J-O. Eklundh (édité par), Stockholm, Sweden, mai 1994, pages 567–576, Vol. 1, *Lecture Notes in Computer Science* volume 800-801, Springer–Verlag.

[EW87] R.D. Eastman et A.M. Waxman, Using disparity functionals for stereo correspondence and surface reconstruction, *Computer Vision, Graphics, and Image Processing* **39** (1987), 73–101.

[Fau79] Olivier Faugeras, Digital color image processing within the framework of a human visual model, *IEEE Transactions on Acoustic, Speech and Signal Processing* **ASSP-27**,4 (août 1979), 380–393.

[Fau92] Olivier Faugeras, What can be seen in three dimensions with an uncalibrated stereo rig?, in *Proceedings of the 2nd European Conference on Computer Vision*, G. Sandini (édité par), Santa Margherita, Italy, mai 1992, pages 563–578, Springer–Verlag.

[Fau93] O. Faugeras, *Three-Dimensional Computer Vision: a Geometric Viewpoint*, MIT Press, 1993.

[Fau94] Olivier Faugeras, Cartan's moving frame method and its application to the geometry and evolution of curves in the euclidean, affine and projective planes, in *Applications of Invariance in Computer Vision*, Joseph L. Mundy, Andrew Zisserman et David Forsyth (édité par), pages 11–46, *Lecture Notes in Computer Vision* volume 825, Springer–Verlag, 1994. also INRIA Tech. Rep. 2053.

[Fel95] J. Feldmar, *Recalage rigide, non rigide et projectif d'images medicales tridimensionnelles*, thèse de doctorat, École Polytechnique, Palaiseau, France, décembre 1995.

[FFH+92] Olivier Faugeras, Pascal Fua, Bernard Hotz, Ruihua Ma, Luc Robert, Monique Thonnat et Zhengyou Zhang, Quantitative and qualitative comparison of some area and feature-based stereo algorithms, in *Robust Computer Vision: Quality of Vision Algorithms*, Wolfgang Förstner et Stephan Ruwiedel (édité par), pages 1–26, Wichmann, Karlsruhe, Germany, 1992.

[FHM+93] Olivier Faugeras, Bernard Hotz, Hervé Mathieu, Thierry Viéville, Zhengyou Zhang, Pascal Fua, Eric Théron, Laurent Moll, Gérard Berry, Jean Vuillemin, Patrice Bertin et Catherine Proy, Real time correlation based stereo: algorithm implementations and applications, rapport technique numéro 2013, INRIA Sophia-Antipolis, France, 1993.

[Fle91] M.M. Fleck, A topological stereo matcher, *The International Journal of Computer Vision* **6**,3 (1991), 197–226.

[FLM92] Olivier Faugeras, Tuan Luong et Steven Maybank, Camera self-calibration: theory and experiments, in *Proceedings of the 2nd European Conference on Computer Vision*, G. Sandini (édité par), Santa Margherita, Italy, mai 1992, pages 321–334, Springer–Verlag.

[FLR+95] Olivier Faugeras, Stéphane Laveau, Luc Robert, Gabriella Csurka et Cyril Zeller, 3-D reconstruction of urban scenes from sequences of images, rapport technique numéro 2572, INRIA, juin 1995.

[FR96] Olivier Faugeras et Luc Robert, What can two images tell us about a third one?, *The International Journal of Computer Vision* **18**,1 (avril 1996), 5–20.

[FT86] Olivier Faugeras et Giorgio Toscani, The calibration problem for stereo, in *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, Miami Beach, FL, juin 1986, pages 15–20, IEEE.

[FT87] Olivier Faugeras et Giorgio Toscani, Structure from Motion using the Reconstruction and Reprojection Technique, in *IEEE Workshop on Computer Vision*, IEEE Computer Society, Miami Beach, novembre-décembre 1987, pages 345–348.

[Fua91] P. Fua., Combining stereo and monocular information to compute dense depth maps that preserve depth discontinuities, in *International Joint Conference on Artificial Intelligence*, Sydney, Australia, août 1991.

[GLY92] Davi Geiger, Bruce Ladendorf et Alan Yuile, Occlusions and binocular stereo, in *Proceedings of the 2nd European Conference on Computer Vision*, G. Sandini (édité par), Santa Margherita, Italy, mai 1992, pages 425–433, Springer–Verlag.

[Gri81] W.E.L. Grimson, *From Images to Surfaces*, MIT Press : Cambridge, 1981.

[Gru85] A. W. Gruen, Data processing methods for amateur photographs, *Photogrammetric Record* **11**,65 (avril 1985), 567–579.

[HA87] W. Hoff et N. Ahuja, Extracting surfaces from stereo images, in *Proceedings of the 1st International Conference on Computer Vision*, London, England, juin 1987, IEEE Computer Society Press.

[Har84] R. Haralick, Digital step edges from zero crossing of second directional derivatives, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **6**,1 (janvier 1984), 58–68.

[Har94a] Richard Hartley, Projective reconstruction and invariants from multiple images, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **16**,10 (1994), 1036–1040.

[Har94b] Richard Hartley, Projective reconstruction from line correspondences, in *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society, Seattle, WA, juin 1994, pages 903–907, IEEE.

[Har97] Richard Hartley, Theory and practice of projective rectification, *The International Journal of Computer Vision*, 1997. à paraître.

[HG94] R. Hartley et R. Gupta, Linear pushbroom cameras, in *Proceedings of the 3rd European Conference on Computer Vision*, J-O. Eklundh (édité par), Stockholm, Sweden, mai 1994, *Lecture Notes in Computer Science* volume 800-801, Springer–Verlag.

[HGC92] Richard Hartley, Rajiv Gupta et Tom Chang, Stereo from uncalibrated cameras, in *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society, Urbana Champaign, IL, juin 1992, pages 761–764, IEEE.

[HM86] A. Huertas et G. Medioni, Detection of intensity changes with subpixel accuracy using laplacian-gaussian masks, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **8**,5 (septembre 1986), 651–664.

[HN94] T.S. Huang et A.N. Netravali, Motion and structure from feature correspondences: A review, *Proc. IEEE* **82**,2 (février 1994), 252–268.

[HS92] R.M. Haralick et L.G. Shapiro, *Computer and Robot Vision*, volume 1, Addison-Wesley, 1992.

[HZF93] B. Hotz, Z. Zhang et P. Fua, Incremental construction of local DEM for an autonomous planetary rover, in *Proc. Workshop on Computer Vision for Space Applications*, Antibes, France, septembre 1993, pages 33–43.

[IHI93] M. Inaba, T. Hara et H. Inoue, A stereo viewer based on a single camera with view-control mechanisms, in *Proceedings of the International Robotics and Systems Conference*, Yokohama, Japon, juillet 1993.

[JB92] J.R. Jordan III et A.C. Bovik, Using chromatic information in dense stereo correspondence, *Pattern Recog.* **25**,4 (1992), 367–383.

[JM92] D.G. Jones et J. Malik, A computational framework for determining stereo correspondence from a set of linear spatial filters, in *Proceedings of the 2nd European Conference on Compu-*

ter Vision, Santa Margherita Ligure, Italy, mai 1992, Springer–Verlag.

[KKSR95] R. Klette, A. Koschan, K. Schlüns et V. Rodehorst, Evaluation of surface reconstruction methods, in *Proc. of the New Zealand Image and Vision Computing '95 Workshop*, Lincoln, Canterbury, août 1995, pages 3–12.

[KO94] T. Kanade et M. Okutomi, A stereo matching algorithm with an adaptive window: Theory and experiment, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **16**,9 (septembre 1994), 920–932.

[Kos93] Andreas Koschan, What is new in computational stereo since 1989: A survey on current stereo papers, rapport technique numéro 93-22, Technical University of Berlin, Department of Computer Science, août 1993.

[Kos94] Andreas Koschan, How to utilize color information in dense stereo matching and edge-based stereo matching, in *Proc. 3rd Int. Conf. on Automation, Robotics and Computer Vision ICARCV '94*, Singapore, novembre 1994, volume 1, pages 419–423.

[KR95] A. Koschan et V. Rodehorst, Towards real-time stereo employing parallel algorithms for edge-based and dense stereo matching, in *Proc. of the IEEE Workshop on Computer Architectures for Machine Perception CAMP'95*, Como, Italy, septembre 1995, pages 234–241.

[KRS96] Andreas Koschan, Volker Rodehorst et Kathrin Spiller, Color stereo vision using hierarchical block matching and active color illumination, in *Proceedings of the International Conference on Pattern Recognition*, Vienna, Austria, août 1996, Computer Society Press.

[KvD76] J.J. Koenderink et A.J. van Doorn, Geometry of binocular vision and a model for stereopsis, *Biological Cybernetics* **21** (1976), 29–35.

[KvDS96] J. J. Koenderink, A. J. van Doorn et M. Stavridi, Bidirectional reflection distribution function expressed in terms of surface scattering modes, in *Proceedings of the 4th European Conference on Computer Vision*, Bernard Buxton (édité par), Cambridge, UK, avril 1996, volume 2, pages 28–39.

[Lav96] Stéphane Laveau, *Géométrie d'un système de N caméras. Théorie, estimation et applications*, PhD thesis, École Polytechnique, mai 1996.

[LCK93] C.Y. Lee, D.B. Cooper et D. Keren, Computing correspondence based on region and invariants without feature extraction and segmentation, in *Proceedings of the International Conference*

on Computer Vision and Pattern Recognition, IEEE Computer Society, New-York, NY, juin 1993, pages 655–656, IEEE.

[LF92] Tuan Luong et Olivier Faugeras, Active stereo with head movements, in *2nd Singapore International Conference on Image Processing*, Singapore, septembre 1992, pages 507–510.

[LF96a] Stéphane Laveau et Olivier Faugeras, Oriented projective geometry for computer vision, in *Proceedings of the 4th European Conference on Computer Vision*, Bernard Buxton (édité par), Cambridge, UK, avril 1996, pages 147–156.

[LF96b] Quang-Tuan Luong et Olivier Faugeras, Camera calibration, scene motion and structure recovery from point correspondences and fundamental matrices, *The International Journal of Computer Vision*, 1996. Use luong-faugeras:97.

[LH81] H.C. Longuet-Higgins, A computer algorithm for reconstructing a scene from two projections, *Nature* **293** (1981), 133–135.

[Lot96] Jean-Luc Lotti, *Mise en correspondance stéréo par fenêtres adaptatives en imagerie haute résolution*, PhD thesis, Université de Nice - Sophia Antipolis, France, février 1996.

[LT88] R. K. Lenz et R. Y. Tsai, Techniques for calibration of the scale factor and image center for high accuracy 3-D machine vision metrology, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **10** (1988), 713–720.

[LTS94] R. A. Lane, N. A. Thacker et N. L. Seed, Stretch-correlation as a real-time alternative to feature-based stereo matching algorithms, *Image and Vision Computing* **12**,4 (mai 1994), 203–212.

[Luo91] Q.T. Luong, La couleur en vision par ordinateur: une revue, *Traitement du Signal* **8**,1 (1991), 3–34.

[Luo92] Quang-Tuan Luong, *Matrice Fondamentale et Calibration Visuelle sur l'Environnement-Vers une plus grande autonomie des systèmes robotiques*, PhD thesis, Université de Paris-Sud, Centre d'Orsay, décembre 1992.

[Mat93] Hervé Mathieu, A multi-DSP 96002 BOARD, rapport technique numéro 153, INRIA, mai 1993.

[MD95] Hervé Mathieu et Frédéric Devernay, Système de miroirs pour la stéréoscopie, rt numéro 172, INRIA, juin 1995.

[MF92] S. J. Maybank et O. D. Faugeras, A theory of self-calibration of a moving camera, *The International Journal of Computer Vision* **8**,2 (août 1992), 123–152.

[MN84] G. Medioni et R. Nevatia, Description of three-dimensional surfaces using curvature properties, in *Proceedings of the ARPA Image Understanding Workshop*, L.S. Baumann (édité par), Defense Advanced Research Projects Agency, New Orleans, LA,

octobre 1984, pages 219–229, Science Applications International Corporation.

[MN85] Gérard Medioni et Ram Nevatia, Segment-based stereo matching, *Computer Vision, Graphics, and Image Processing* **31** (1985), 2–18.

[MZ92] Joseph L. Mundy et Andrew Zisserman (édité par), *Geometric Invariance in Computer Vision*, MIT Press, 1992.

[NB86] V. S. Nalwa et T. O. Binford, On detecting edges, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **8**,6 (1986), 699–714.

[Nis84] Keith Nishihara, Practical real-time imaging stereo matcher, *Optical Engineering* **23**,5 (1984).

[NMSO96] Yuichi Nakamura, Tomohiko Matsuura, Kiyohide Satoh et Yuichi Ohta, Occlusion detectable stereo – occlusion patterns in camera matrix, in *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society, San Francisco, CA, juin 1996, pages 371–378, IEEE.

[OK85] Y. Ohta et T. Kanade, Stereo by intra- and inter-scanline search, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **7** (1985), 139–154.

[ON94] Michael Oren et Shree K. Nayar, Generalization of lambert's reflectance model, in *Proceedings of the SIGGRAPH*, Andrew Glassner (édité par), ACM SIGGRAPH, Orlando, Florida, juillet 1994, pages 239–246, *Computer Graphics Proceedings, Annual Conference Series*, ACM Press.

[OYT92] M. Okutomi, O. Yoshizaki et G. Tomita, Color stereo matching and its application to 3-d measurement of optic nerve head, in *Proceedings of the International Conference on Pattern Recognition*, Den Hag, The Netherlands, 1992, volume 1, pages 509–513.

[Pap95] Théodore Papadopoulo, *Analyse du mouvement de courbes rigides 3D à partir de séquences d'images monoculaires*, PhD thesis, Université de Paris-Sud Centre d'Orsay, mai 1995. Existe aussi en version anglaise (INRIA RR-2779).

[Pen91] M. A. Penna, Camera calibration: A quick and easy way to determine the scale factor, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **13** (1991), 1240–1245.

[Pér91] J. Ph. Pérez, *Optique géométrique, ondulatoire et polarisation*, Masson, Paris Milan Barcelone Bonn, édition 3e, 1991.

[PMF85] S.B. Pollard, J.E.W. Mayhew et J.P. Frisby, PMF : a stereo correspondence algorithm using a disparity gradient constraint, *Perception* **14** (1985), 449–470.

[Poy] Charles A. Poynton, Poynton's color technology page. `http://www.inforamp.net/~poynton/Poynton-color.html`

[RD96] L. Robert et R. Deriche, Dense depth map reconstruction: A minimization and regularization approach which preserves discontinuities, in *Proceedings of the 4th European Conference on Computer Vision*, Bernard Buxton (édité par), Cambridge, UK, avril 1996.

[RH94] L. Robert et M. Hebert, Deriving orientation cues from stereo images, in *Proceedings of the 3rd European Conference on Computer Vision*, J-O. Eklundh (édité par), Stockholm, Sweden, mai 1994, pages 377–388, *Lecture Notes in Computer Science* volume 800-801, Springer–Verlag.

[Rob93] Luc Robert, *Perception stéréoscopique de courbes et de surfaces tridimensionnelles. Applications à la robotique mobile*, PhD thesis, École Polytechnique, Paris, France, mars 1993.

[Rob95] L Robert, Camera calibration without feature extraction, *Computer Vision, Graphics, and Image Processing* **63**,2 (mars 1995), 314–325. also INRIA Technical Report 2204.

[SB94] B. Serra et M. Berthod, Subpixel contour matching using continuous dynamic programming, in *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society, Seattle, WA, juin 1994, pages 202–207, IEEE.

[SC92] J. Shen et S. Castan, An optimal linear operator for step edge detection, *CVGIP: Graphics Models and Image Processing* **54**,2 (mars 1992), 112–133.

[SD88] Charles V. Stewart et Charles R. Dyer, The trinocular general support algorithm: A three-camera stereo algorithm for overcoming binocular matching errors, in *Proceedings of the 2nd International Conference on Computer Vision*, Tampa, FL, décembre 1988, pages 134–138, IEEE Computer Society Press.

[SFB96] Charles Stewart, Robin Flatland et Kishore Bubna, Geometric constraints and stereo disparity computation, *The International Journal of Computer Vision* **20**,3 (1996), 143–168.

[Sla80] C. C. Slama (édité par), *Manual of Photogrammetry*, American Society of Photogrammetry, édition fourth, 1980.

[Spi79] Michael Spivak, *A Comprehensive Introduction to Differential Geometry*, volume I–V, Publish or Perish, Berkeley, CA, 1979. Second edition.

[Ste93] Gideon P. Stein, Internal camera calibration using rotation and geometric shapes, Master's thesis, Massachusetts Institute of Technology, juin 1993. AITR-1426.

[Ste95] Gideon P. Stein, Accurate internal camera calibration using rotation with analysis of sources of error, in *Proceedings of the 5th International Conference on Computer Vision*, Boston, MA, juin 1995, IEEE Computer Society Press.

[Sto91] Jorge Stolfi, *Oriented Projective Geometry, A Framework for Geometric Computations*, Academic Press, Inc., 1250 Sixth Avenue, San Diego, CA, 1991.

[SZL92] William J. Schroeder, Jonathan A. Zarge et William E. Lorensen, Decimation of triangle meshes, in *Proceedings of the SIGGRAPH*, Edwin E. Catmull (édité par), ACM SIGGRAPH, Chicago, Illinois, juillet 1992, pages 65–70, *Computer GraphicsProceedings, Annual Conference Series* volume 25, ACM Press.

[Ter86] Demetri Terzopoulos, Regularization of inverse visual problems involving discontinuities, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **8** (1986), 413–424.

[TM84] Ali J. Tababai et O. Robert Mitchell, Edge location to subpixel values in digital imagery, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **6**,2 (mars 1984), 188–201.

[Tos87] G. Toscani, *Système de Calibration optique et perception du mouvement en vision artificielle*, PhD thesis, Paris-Orsay, 1987.

[Tsa87] Roger Y. Tsai, A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses, *IEEE Journal of Robotics and Automation* **3**,4 (août 1987), 323–344.

[VF92] Thierry Viéville et Olivier D. Faugeras, Robust and fast computation of unbiased intensity derivatives in images, in *Proceedings of the 2nd ECCV*, Giulio Sandini (édité par), Santa-Margherita, Italy, 1992, pages 203–211, Springer–Verlag.

[WB95] C. S. Wiles et M. Brady, Closing the loop on multiple motion, in *5th International Conference on Computer Vision*, pages 308–313, 1995.

[WCH92] J. Weng, P. Cohen et M. Herniou, Camera calibration with distortion models and accuracy evaluation, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **14**,10 (octobre 1992), 965–980.

[Wil91] Richard P. Wildes, Direct recovery of three-dimensional scene geometry from binocular stereo disparity, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **13**,8 (août 1991), 761–774.

[Wil94] Reg G. Willson, *Modeling and Calibration of Automated Zoom Lenses*, PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1994. CMU-RI-TR-94-03.

[WS67] Günther Wyszecki et W. S. Stiles, *Color Science: Concepts and Methods, Quantitative Data and Formulas*, John Wiley and Sons, Inc., New York, London, Sidney, 1967.

[ZBR95] Andrew Zisserman, Paul A. Beardsley et Ian D. Reid, Metric calibration of a stereo rig, in *Proc. Workshop on Visual Scene Representation*, Boston, MA, juin 1995.

[ZDFL95] Z. Zhang, R. Deriche, O. Faugeras et Q.T. Luong, A robust technique for matching two uncalibrated images through the recovery of the unknown epipolar geometry, *Artificial Intelligence Journal* **78** (octobre 1995), 87–119.

[Zel96] Cyril Zeller, *Calibration Projective Affine et Euclidienne en Vision par Ordinateur*, PhD thesis, École Polytechnique, février 1996.

[ZF94a] Cyril Zeller et Olivier Faugeras, Applications of non-metric vision to some visual guided tasks, in *Proceedings of the International Conference on Pattern Recognition*, Jerusalem, Israel, octobre 1994, pages 132–136, Computer Society Press. A longer version in INRIA Tech Report RR2308.

[ZF94b] Cyril Zeller et Olivier Faugeras, Applications of non-metric vision to some visual guided tasks, research report numéro 2308, INRIA, juillet 1994.

[ZF95] Cyril Zeller et Olivier Faugeras, Projective, affine and metric measurements from video sequences, in *Proceedings of the International Symposium on Optical Science, Engineering and Instrumentation*, SPIE, San Diego, juillet 1995.

[ZFD95] Z. Zhang, O. Faugeras et R. Deriche, Calibrating a binocular stereo through projective reconstruction using both a calibration object and the environment, in *Proc. Europe-China Workshop on Geometrical modelling and Invariants for Computer Vision*, R. Mohr et C. Wu (édité par), Xi'an, China, avril 1995, pages 253–260.

[Zha96a] Z. Zhang, Determining the epipolar geometry and its uncertainty: A review, rapport technique numéro 2927, INRIA Sophia-Antipolis, France, juillet 1996.

[Zha96b] Z. Zhang, On the epipolar geometry between two images with lens distortion, in *International Conference on Pattern Recognition*, Vienna, Austria, août 1996, volume I, pages 407–411.

[ZLF96] Zhengyou Zhang, Quang-Tuan Luong et Olivier Faugeras, Motion of an uncalibrated stereo rig: self-calibration and metric reconstruction, *IEEE Transactions on Robotics and Automation* **12**,1 (février 1996), 103–113.

Résumé

Ce document traite de plusieurs aspects de la vision stéréoscopique par ordinateur. Cette méthode consiste à partir d'une ou de plusieurs paires d'images à « reconstruire » une scène observée en trois dimensions, c'est-à-dire à produire une description des objets et surfaces observés ainsi que leur position dans l'espace.

Le premier problème abordé est celui du calibrage, dont l'objet est de calculer les paramètres des caméras (focale, centre optique, etc.) ainsi que leur position, soit à partir d'images d'objets de géométrie et de position connue, soit de manière automatique (on parle alors d'auto-calibrage). Des résultats nouveaux sont présentés sur l'auto-calibrage de la distorsion optique et sur l'auto-calibrage d'une paire de caméras rigidement liées à partir de plusieurs paires d'images. Ensuite sont présentées différentes méthodes permettant de rectifier les images de manière à simplifier la mise en correspondance, puis d'effectuer cette mise en correspondance par une technique de corrélation. Outre des améliorations des résultats classiques, de nouvelles méthodes permettant d'obtenir une plus grande précision sont discutées. La dernière phase, dite de reconstruction, permet d'obtenir une description des surfaces observée allant jusqu'aux propriétés différentielles d'ordre un et deux (plan tangent et courbures à la surface), à partir des résultats de stéréoscopie par corrélation. Ce document se termine par quelques applications réalisées au cours de ces recherches telles qu'un système d'aide chirurgicale pré-opératoire ou une caméra stéréo bon marché.
Mots clef : vision par ordinateur, stéréoscopie, calibrage, distorsion optique, géométrie projective, géométrie différentielle.

Abstract

This dissertation deals with several aspects of stereoscopic computer vision, which consists of "reconstructing" a scene observed from several cameras in 3-D, i.e. to build a description of the observed objetcs ans surfaces, and their position in 3-D space.

The first problem we introduce is the camera calibration problem, where the goal is to compute the camera parameters (focal length, optical center, etc.) and their position, either from images of a calibration object or automatically (the last case is the so-called self-calibration problem). We present new results on self-calibration of optical distortion and on self-calibration of a rigid stereo rig from several pairs of images. Then we discuss on image rectification, which simplifies a lot the stereo matching problem, and on stereoscopy by correlation, which performs the pointwise image matching. We got very good results using some classical methods, and we also developped new methods which give a better accuracy. The last step of stereoscopic vision, called the reconstruction, gives a 3-D description of a surface and its differential properties up to order 2 from the results of the correlation methods. Finally, we show a few applications, like a system for fusing stereo images with volumetric medical images, and a DIY stereo camera.
Keywords: Computer Vision, Stereoscopy, Calibration, Optical Distortion, Projective Geometry, Differential Geometry.