Titre: Title:	Appariement de points caractéristiques trouvés à même les régions d'avant-plan de vidéos à spectres visible et infrarouge
Auteur: Author:	Pier-Luc St-Onge
Date:	2007
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	St-Onge, P.-L. (2007). Appariement de points caractéristiques trouvés à même les régions d'avant-plan de vidéos à spectres visible et infrarouge [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/8113/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/8113/
Directeurs de recherche: Advisors:	Guillaume-Alexandre Bilodeau
Programme: Program:	Non spécifié

UNIVERSITÉ DE MONTRÉAL

APPARIEMENT DE POINTS CARACTÉRISTIQUES TROUVÉS À MÊME LES RÉGIONS D'AVANT-PLAN DE VIDÉOS À SPECTRES VISIBLE ET INFRAROUGE

PIER-LUC ST-ONGE
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
DÉCEMBRE 2007

Your file Votre référence
ISBN: 978-0-494-36941-8
Our file Notre référence
ISBN: 978-0-494-36941-8

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

APPARIEMENT DE POINTS CARACTÉRISTIQUES TROUVÉS À MÊME LES RÉGIONS D'AVANT-PLAN DE VIDÉOS À SPECTRES VISIBLE ET INFRAROUGE

présenté par : ST-ONGE Pier-Luc
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de :

M. GAGNON Michel, Ph.D., président
M. BILODEAU Guillaume-Alexandre, Ph.D., membre et directeur de recherche
Mme. CHERIET Farida, Ph.D., membre

À la fin de la guerre...

...entre les clans du Blu-ray Disc et du HD DVD.

REMERCIEMENTS

J'aimerais remercier les membres du jury pour avoir accepté d'évaluer mon mémoire.

J'aimerais remercier spécifiquement M. Michel Gagnon qui a été mon professeur-parrain et mon superviseur pour mon premier stage de recherche. Il a été l'un des premiers à m'avoir encouragé à continuer mes études en maîtrise. Il m'a aussi écrit une lettre de référence pour ma demande de bourse auprès du FQRNT.

Je remercie M. Sylvain Foisy pour m'avoir offert mon second stage en recherche et pour m'avoir écrit une lettre de référence pour ma demande de bourse du FQRNT.

Je remercie Mme Farida Cheriet qui a aussi été mon professeur-parrain, mais pour un stage en entreprise dans le domaine du traitement vidéo. Elle m'a aidé à trouver mon directeur de recherche, M. Guillaume-Alexandre Bilodeau.

Je remercie mon directeur de recherche, M. Guillaume-Alexandre Bilodeau, pour m'avoir accueilli dans son laboratoire et pour son aide lors de la rédaction des demandes de bourse du FQRNT et du CRSNG. Il m'a aidé tout au long de ma recherche, que ce soit pour l'établissement du sujet du projet, pour fixer des priorités et des objectifs ou pour trouver l'élément-clé qui m'a permis d'avancer dans le développement des méthodes présentées dans le présent mémoire.

Je remercie le Fonds québécois de la recherche sur la nature et les technologies pour leur support financier.

Je remercie mes collègues Soufiane Ammouri, Guillaume Desjardins, François Morin, Atousa Torabi et Parisa Darvish Zadeh Varcheie pour leur participation dans la réalisation des séquences vidéo et pour leurs conseils.

Finalement, je remercie mes parents et ma sœur qui m'ont soutenu et encouragé tout au long de ma recherche. J'ai effectivement eu des moments plus difficiles que d'autres et ils ont su me motiver et m'aider pour que je puisse terminer mon travail.

RÉSUMÉ

Notre projet porte sur la vision stéréo à l'aide de paires d'images constituées d'une image du spectre visible et d'une image du spectre infrarouge. Cependant, la reconstruction de la scène se limite uniquement à donner un indice sur la profondeur relative des objets en mouvement.

La littérature scientifique contient déjà plusieurs algorithmes de calibration de caméras, de recherche de points caractéristiques, d'appariement de points, de reconstruction, etc. Dans notre cas, les deux caméras ne sont pas calibrées. De plus, la plupart des techniques classiques d'appariement de points ne permettent pas de reconnaître un point de l'image visible dans l'image infrarouge.

Notre méthode se base tout d'abord sur la soustraction de l'arrière-plan de chaque image individuellement. Les blobs d'avant-plan deviennent donc une source d'information invariante en fonction du type d'image utilisé. De là, nous avons deux méthodes : la méthode du squelette et la méthode du processus DCE. La méthode du squelette consiste à trouver les axes de symétrie locale d'une forme. Nous analysons ensuite les intersections des axes à une, trois ou plus arêtes. L'orientation des arêtes de ces points d'intersections et la distance minimale entre ces intersections et le contour du blob sont une partie des informations utilisées pour apparier les points d'une image à l'autre. La méthode du processus DCE consiste à simplifier le contour de chaque blob pour ne garder qu'un nombre préfixé de sommets. L'orientation des arêtes des sommets retenus et le degré de pertinence des sommets retenus sont utilisés pour apparier les points. L'appariement s'effectue à l'aide d'une matrice de correspondance de type choix mutuel.

Dans notre système, nous appliquons deux filtres successifs pour éliminer les paires de points aberrantes : l'usage des paires de blobs et un algorithme RANSAC pour trouver la matrice fondamentale modélisant l'ensemble des paires de points. Cette dernière méthode est largement utilisée et documentée dans la littérature.

Le résultat final de notre programme est la disparité horizontale médiane de chaque paire de blobs trouvée, c'est-à-dire un indice de profondeur relative pour chaque élément de la

scène ne faisant pas partie de l'arrière-plan. Nous comparons cette disparité horizontale avec les disparités déterminées dans des valeurs témoins. Pour ce faire, nous utilisons un score calculant la différence absolue entre chaque disparité calculée et sa disparité correspondante connue. Cette différence est par la suite normalisée selon la longueur de chaque disparité. Pour chaque paire d'image, nous retenons 1.0 moins la moyenne du score de chaque paire de blobs. Les scores finaux de chaque paire d'images vont de 0.0 à 1.0, où 1.0 est une note parfaite.

Les résultats obtenus montrent que nos méthodes de recherches de points caractéristiques dans des images de différents types donnent souvent des résultats de l'ordre de 0.8 à 0.9 sur 1.0. Nous avons comparé nos méthodes avec une adaptation de la méthode utilisant la congruence de phase : nos résultats numériques sont légèrement meilleurs que ceux de cette méthode. Nous avons aussi utilisé des méthodes basées sur les algorithmes des valeurs propres minimales et sur l'opérateur de Harris et Stephens : ces méthodes ne font pas le poids lorsqu'il s'agit d'apparier des points de différents types d'images, car elles n'arrivent généralement pas à trouver toutes les bonnes paires de blobs dans une paire d'images.

Finalement, notre score final est trop sensible pour évaluer de petites disparités. Par conséquent, le résultat visuel peut être meilleur que le résultat numérique. Donc, d'autres tests pourraient démontrer numériquement que nos méthodes permettent d'identifier la distance relative entre chaque personne et l'ordre de celles-ci : de la plus près à la plus éloignée.

ABSTRACT

Our project's goal is to do some stereo vision with pairs of images where one of the images is from the visible spectrum and the other is from the infrared spectrum. The reconstruction is limited to only a relative depth value for each moving object in the scene: a foreground blobs' horizontal disparity.

In the literature, we can find many camera calibration algorithms, feature points finders, pairing algorithms and reconstruction algorithms. In our case, both cameras are not calibrated. Furthermore, most classical techniques to create reliable pairs of points are not able to recognize a point of the visible image in the infrared image.

Our method is based on the background subtraction of each image individually. All foreground blobs become an invariant source of information according to the type of image used. From there, we have two methods: the skeleton method and the DCE method. The skeleton method consists in finding in the skeleton the local axes of symmetry of the shape. Then, we analyze the intersections of the axes having one, three or more edges. The orientation of the intersection points' edges and the minimum distance between the intersection points and the blobs' contour are part of the information used to create the pairs of points. The method based on the DCE process consists in simplifying the blob's contour in order to keep only a fixed number of significant vertices. Again, the orientation of the two edges of the chosen vertices and the relevance measure of these vertices in the approximated contour are used to create the pairs of points. The pairing process is done with a correspondence matrix where each point in a pair has chosen the other mutually.

In our system, we apply two successive filters to eliminate the outliers in the pairs of points: the use of blob pairs and the use of a RANSAC algorithm that models inliers with a fundamental matrix. This last method is largely used and documented in the literature.

The final result of our project is the median horizontal disparity of each found pair of blobs, i.e. a relative depth value for all elements of the scene that are not in the background. We compare this horizontal disparity with the disparities that are in ground

truths. In the comparison, we use a score calculating the absolute difference between each computed disparity and its known corresponding disparity in the ground truth. This difference is then normalized according to the length of each disparity (the calculated one and the ground truth one). For each pair of images, we retain 1.0 minus the mean of the scores of all valid pairs of blobs. The final scores are from 0.0 to 1.0, where 1.0 is a perfect score.

The test results show that our methods that find feature points in different types of image usually give final scores between 0.8 and 0.9 on 1.0. We have compared our methods with an adaptation of the method using the phase congruency: the numerical results (the final score) of our methods are slightly better than the results with the phase congruency. We have also used methods based on the minimal eigenvalue algorithm and on the Harris and Stephens operator: these methods are just not able to find good pairs of points in different types of image, mainly because they are generally not able to find the good pairs of blobs.

Finally, our final score criterion is too sensitive to evaluate properly small disparities. Consequently, the visual result may be better than the numerical result. So, more tests may prove numerically that our methods can identify the relative distance between each person in the scene and the order of these: from the closest to the farthest.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

BLOB	Binary Large OBject, groupe de pixels interconnectés, région d'avant-plan
DCE	Discrete Curve Evolution
DdG	Différence de Gaussiennes
OpenCV	Open Computer Vision
RANSAC	RANdom SAmple Consensus

LISTE DES ANNEXES

INTRODUCTION

Depuis leur tendre enfance, les êtres humains, tout comme plusieurs autres races d'animaux, sont capables de voir et d'interpréter une scène en trois dimensions grâce à leurs deux yeux. Chaque œil voit la même scène, mais de deux positions distinctes. Il en résulte donc deux images légèrement différentes qui sont envoyées au cerveau pour la reconstruction finale de la scène. Non seulement l'interprétation de la scène se fait en un temps record, c'est-à-dire plusieurs dizaines de fois par seconde, mais il s'agit là d'un processus qui semble peu fatiguant pour les millions de neurones qui constituent les cerveaux. Pourtant, ce qui semble si aisé pour le commun des mortels est en fait une tâche très difficile pour un ordinateur.

Malgré tout, la littérature scientifique contient déjà des centaines d'articles et plusieurs livres à propos de systèmes informatiques capable de reconstruire une scène tridimensionnelle à partir d'images de la même scène. Le système peut être calibré ou non calibré (section 1.1) et à deux ou à trois caméras. Dans la plupart des cas, les chercheurs utilisent deux images provenant de deux caméras identiques. De plus, afin de faciliter la mise en correspondance, les images sont rectifiées (Trucco & Verri, 1998) pour que les lignes de pixels d'une image soient vis-à-vis les lignes de pixels correspondantes de l'autre image.

Dans le domaine de la vidéosurveillance, les caméras infrarouges commencent de plus en plus à être à la mode, et ce, grâce à leur prix devenant de plus en plus abordable (Hajebi & Zelek, 2006). Les caméras infrarouges permettent parfois de voir ce qu'une caméra du spectre visible ne peut pas voir, par exemple, une scène où l'éclairage est faible ou absent. Enfin, des chercheurs ont déjà réussi à créer des prototypes de vision stéréo pour des images infrarouges (Hajebi & Zelek, 2006) : il s'agit en fait de cartes de disparités[1] donnant grossièrement une idée de la profondeur de quelques points dans une image.

---

[1] Dans le cadre du présent projet, une disparité est la mesure du déplacement entre un point d'une image et le point correspondant dans l'autre image alors que les images sont alignées et superposées. La longueur du déplacement permet d'évaluer la profondeur du point dans la scène. Voir les disparités à la figure 1.8.

Le défi du présent projet consiste à créer un système de vision stéréo non calibré utilisant une caméra du spectre visible et une caméra du spectre infrarouge. Alors que l'idée est intéressante, il faut noter que le cerveau de l'être humain ne serait même pas capable de reconstruire une scène à partir des paires d'images utilisées dans le présent projet. En effet, nous avons testé notre système de vision humain avec deux images rectifiées dont l'une avait les couleurs inversées. Ainsi, au lieu de voir la scène en trois dimensions, nous voyons plutôt deux images superposées sous la forme d'une moyenne pondérée qui varie en fonction de l'importance que notre cerveau accorde à chaque image à chaque instant. Ce dernier petit test est valide, car regarder une image du spectre visible et une autre du spectre infrarouge donne l'impression de regarder une photo et son négatif. Bref, l'ordinateur peut parfois surpasser l'homme pour certaines tâches.

Contexte et problématique

Pour des fins de vidéosurveillance, l'usage de caméras infrarouges est de plus en plus courant, de même que l'usage de plusieurs caméras par scène pour avoir différents points de vue. Comme nous l'avons déjà mentionné, ce qui est visible dans une partie du spectre électromagnétique peut ne pas l'être dans une autre partie du spectre. Ainsi, l'usage de divers types de capteurs (visible et infrarouge) permet d'augmenter nos chances de détecter un événement quelconque dans une scène dont la luminosité et la température peut changer.

À partir de là, nous voudrions aller plus loin en tentant d'effectuer de la vision stéréo à partir des éléments de la scène qui sont effectivement visibles dans chaque partie du spectre électromagnétique étudiée. Typiquement, la vision stéréo s'effectue avec des images de même type. Ainsi, lorsque vient le temps d'apparier des points correspondants d'une image à l'autre, ces points sont semblables et ont un voisinage semblable. Par contre, les points correspondants dans une paire d'images visible et infrarouge ne sont pas nécessairement semblables en termes de couleurs et ils n'ont pas nécessairement le même voisinage de pixels. Par exemple, un chandail à carreaux porté par une personne aura une texture particulièrement détaillée et colorée dans l'image du spectre visible,

mais il n'y aura probablement aucune texture particulière dans l'image infrarouge puisque le chandail devrait normalement avoir une température constante sur la surface du corps. Ces différences font en sorte qu'il est difficile d'apparier des points correspondants dans des images de types différents.

Objectifs

Étant donné la difficulté du problème, nos objectifs ne peuvent comporter la reconstruction complète et détaillée d'une scène à partir d'images de nature différente. Cependant, dans un contexte de vidéosurveillance, nous voudrions pouvoir évaluer grossièrement la position des protagonistes de la scène, non seulement de gauche à droite et de haut en bas, mais aussi de proche à éloigné. Il s'agit donc de donner une certaine mesure de profondeur aux BLOBs (ou blobs[1]) d'avant-plan dans chaque image de la scène dont l'arrière-plan est fixe ou temporellement constant.

En ayant cette information, il sera alors possible d'évaluer la position tridimensionnelle relative des personnes dans une scène, et ce, afin d'avoir davantage d'informations sur ce qui se passe : rien d'anormal ou un acte de violence ou de délinquance. Cela permettrait donc d'ajouter une dimension supplémentaire à toutes les autres méthodes développées dans le domaine de la vidéosurveillance.

Finalement, en fonction des méthodes développées, nous voulons évaluer leur performance et trouver les paramètres qui pourraient les optimiser.

Voici donc la liste détaillée de nos objectifs :

- Développer des détecteurs de points caractéristiques compatibles avec des images visibles et des images infrarouges;
- Développer des algorithmes d'appariement de points caractéristiques provenant d'images de différents types;
- Développer au moins un filtre contre les paires de points aberrantes;
- Établir une valeur de profondeur relative pour les paires de blobs correspondants;

[1] Pour notre projet, les blobs sont les régions d'avant-plan trouvées par une soustraction d'arrière-plan. Ces régions sont des groupes de pixels interconnectés. Voir la section 2.2 pour plus de détails.

- Tester l'efficacité des méthodes développées, et ce, en fonction de leurs différents paramètres.

Aperçu de la méthode proposée

Nous commençons par filmer une scène quelconque à l'aide d'une caméra du spectre visible et une seconde du spectre infrarouge. Évidemment, ces deux caméras doivent être synchronisées pour nous assurer de retrouver les mêmes événements dans chaque paire d'images étudiée.

Puisque les algorithmes de soustraction d'arrière-plan s'adaptent normalement à tout type d'image, nous commençons par trouver les blobs d'avant-plan dans chaque image individuellement. À partir de ces blobs, nous trouvons des points potentiellement correspondants que nous appelons points caractéristiques : ils caractérisent un point de la scène, et ce, en étant autant que possible invariants aux changements d'apparence d'une image à l'autre. Un maximum de points caractéristiques d'une image sont ensuite appariés à certains points caractéristiques de l'autre image, et ce, en fonction de certains critères de ressemblance qui ne dépendent pas de la couleur et des textures du voisinage de chaque point caractéristique.

Cette méthode génère quelques bonnes paires de points correspondants, mais aussi des paires de points qui ne vont pas ensemble : ce sont les paires aberrantes. Pour éliminer les mauvaises paires de points, nous utilisons deux filtres successifs. Le premier filtre utilise le fait qu'il y a forcément des paires de blobs (ou régions) d'avant-plan qui correspondent aux personnes ou aux objets en mouvement dans la scène. Donc, en créant des paires de blobs, il est alors possible de réduire l'espace d'appariement pour les paires de points. Par la suite, le second filtre tente d'éliminer les dernières paires de points aberrantes en utilisant la géométrie *épipolaire* que nous pouvons vulgariser en géométrie de la vision stéréo artificielle.

Finalement, les paires de points permettent d'évaluer la disparité des paires de blobs. Ces disparités sont en fait le déplacement absolu d'un blob d'une image à l'autre lorsque ces

images sont disposées l'une sur l'autre. Ce sont ces disparités que nous évaluons pour établir la précision et l'exactitude de nos algorithmes et de ceux de la littérature.

Contributions

Notre contribution est essentiellement la réalisation d'un système de recherche de points caractéristiques correspondants dans des images de différents types. Spécifiquement, notre première contribution est la proposition de deux algorithmes différents pour trouver des points caractéristiques. Ces deux algorithmes sont basés sur trois techniques connues de la littérature : la soustraction d'arrière-plan (section 2.2.1), le squelette d'un blob (section 2.3) et le processus DCE (section 2.4). Notre deuxième contribution est la proposition de critères d'appariement pour les points caractéristiques générés par chacune de nos deux méthodes. Enfin, comme troisième contribution, nous proposons l'usage des paires de blobs pour améliorer nos résultats.

Par ailleurs, ces méthodes sont évaluées à l'aide d'une méthode de test que nous avons développée (section 3.1). Cette méthode permet donc d'évaluer la précision des disparités des paires de blobs correspondants.

Plan du mémoire

Le premier chapitre décrit les différents systèmes de caméras, les algorithmes classiques de recherche de points caractéristique, les méthodes d'appariement et de filtrage de paires de points. Le second chapitre décrit en détails notre méthodologie et notre contribution, c'est-à-dire les algorithmes qui sont pratiquement implémentés dans le cadre du présent projet. Le troisième chapitre contient le détail de notre expérimentation ainsi que les résultats et une brève discussion pour chaque résultat. Finalement, le mémoire se termine par nos conclusions et quelques suggestions de travaux futurs.

CHAPITRE 1 REVUE DE LA LITTÉRATURE

La première tâche à accomplir dans un système de vision stéréo est l'acquisition de séquences vidéo synchronisées ou d'images. Le système de caméras peut être calibré ou non (section 1.1). Par la suite, on effectue la recherche de points caractéristiques[1] dans les paires d'images. Ces points caractéristiques doivent être tels qu'il puisse être possible de les apparier d'une image à l'autre. Enfin, puisqu'il y a parfois quelques erreurs d'appariement, il faut filtrer les paires de points aberrantes. Ces trois dernières étapes sont approfondies dans les sections 1.2, 1.3 et 1.4.

1.1 Systèmes de caméras calibrés

Lors de la capture vidéo, la projection de chaque point de la scène dans une image dépend de la position de la caméra (paramètres extrinsèques) et des spécifications de l'objectif utilisé (paramètres intrinsèques).

À l'aide des paramètres extrinsèques, on peut transformer les coordonnées $[X_s, Y_s, Z_s]^T$ d'un point de la scène en des coordonnées $[X_c, Y_c, Z_c]^T$ selon le référentiel de chaque caméra (Trucco & Verri, 1998, pp. 124-127). Cette transformation est décrite à l'équation 1.1. Dans cette équation, R est une matrice de rotation, alors que T est un vecteur de translation.

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = R \begin{bmatrix} X_s \\ Y_s \\ Z_s \end{bmatrix} + \mathrm{T} \tag{1.1}$$

Par la suite, les paramètres intrinsèques permettent de compléter la projection d'un point de la scène (maintenant positionné selon le référentiel de la caméra) à l'image numérique générée par la caméra (Trucco & Verri, 1998, pp. 36-37). Tout d'abord, la distance focale de l'objectif définit le champ de vision de la caméra. Ensuite, l'objectif cause une distorsion radiale plus ou moins sévère. Enfin, les coordonnées de l'image sur le capteur photographique sont transformées en coordonnées finales en pixels.

---

[1] Voir la section 1.2 pour notre définition des points caractéristiques.

À l'aide de certains motifs carrés sur un volume à plusieurs plans[1], il est possible de calculer les paramètres intrinsèques et extrinsèques de chaque caméra, et ce, en se servant aussi des points caractéristiques de chaque image qui correspondent aux points saillants de la scène. Par la suite, ces paramètres permettent de faciliter la reconstruction de la scène, et ce, par triangulation (Trucco & Verri, 1998, pp. 162-163).

Dans la littérature, on voit souvent des paires d'images rectifiées de telle sorte qu'il y a une correspondance ligne à ligne entre les deux images. Cette rectification est aussi facilitée grâce aux paramètres intrinsèques et extrinsèques (Trucco & Verri, 1998, pp. 157-161). Après la rectification, le calcul des disparités ou de la profondeur des pixels dans chaque image est effectué indépendamment pour chaque ligne, ce qui rend le problème plus facile à résoudre : Zitnick et Kanade (2000) présentent une solution qui semble bien fonctionner.

Dans le cadre du présent projet, nous n'avons pas calibré le système de caméras, et ce, même si les deux caméras utilisent un modèle de projection de type *sténopé* dans lequel on retrouve des paramètres intrinsèques et extrinsèques propres à chacune des caméras du spectre visible et du spectre infrarouge. Cela ajoute donc un degré de difficulté supplémentaire pour la reconstruction, mais, de toute façon, nous n'allons pas jusqu'à reconstruire la scène complètement. Tout ce qui nous intéresse est d'avoir une idée de la position des objets en mouvement. Pour ce faire, nous positionnons les caméras à au moins un demi-mètre de distance. Les trépieds sont au niveau. Les deux caméras doivent regarder vers le même point le plus éloigné de la scène : ce point doit être au centre du champ de vision de chaque caméra. Le champ de vision doit être à peu près le même : en regardant les images de la paire stéréo, on doit avoir l'impression que le champ de vision est équivalent, peu importe la spécification des lentilles utilisées. Bref, le système n'est pas calibré, ni en terme de positionnement, ni en terme de paramètres de projection, mais c'est suffisant pour arriver à obtenir de bonnes disparités relatives.

---

[1] Ce volume contient plusieurs points saillants facilement identifiables dans les images. Les coordonnées des points saillants dans la scène sont connues puisqu'elles sont définies par rapport à un référentiel sur le volume en question.

1.2 Extraction des points caractéristiques

Pour être utilisables, les points caractéristiques, ou les points saillants de la scène projetés dans chaque image, doivent être invariants aux diverses transformations dues aux changements de point de vue des caméras. En effet, selon le point de vue, les rayons lumineux peuvent être réfléchis différemment selon la forme géométrique, la texture et le type de surface des objets dans la scène. Ainsi, un point de la scène peut paraître un peu différent dans les deux images de la paire stéréo, mais un bon algorithme d'extraction de points caractéristiques parviendra à les trouver et à les distinguer des autres points de la scène.

Les prochaines sections contiennent la description de plusieurs types de points caractéristiques. Ils ont été utilisés principalement pour rechercher des paires de points dans des images de même type, c'est-à-dire deux images du spectre visible ou deux images du spectre infrarouge.

1.2.1 Détecteur de coins de Harris et Stephens

Le détecteur de coins de Harris et Stephens (1988) est un algorithme utilisé dans plusieurs projets où l'on tente de trouver des points caractéristiques. S'il est tant utilisé, c'est qu'il donne des résultats appréciables tout en étant relativement très rapide à l'exécution.

L'algorithme de Harris est dérivé de celui de Moravec (1980). Ce dernier avait proposé de dupliquer une image et de déplacer légèrement la copie en question, et ce, dans différentes directions. La figure 1.1 présente les quatre différents déplacements (x, y) retenus. La différence entre les deux images I superposées permet d'établir l'équation 1.2 où w est une matrice de poids dont la somme est 1. Afin de calculer la valeur de $E_{x,y}$ pour chaque pixel $I_{i,j}$, les poids non nuls de w sont centrés en fonction de $I_{i,j}$. Ainsi, w agit comme une fenêtre sur le voisinage de $I_{i,j}$.

$$E_{x,y} = \sum_{u,v} w_{u,v} \left(I_{u+x,v+y} - I_{u,v} \right)^2 \tag{1.2}$$

L'idée est que si E est faible pour un déplacement (x, y), alors ou bien la région définie par w est relativement constante localement, ou bien il y a une arête orientée selon le déplacement étudié. De même, E sera plus élevé lorsque le déplacement sera perpendiculaire à une arête. Dans le cas d'un coin, ce sont toutes les orientations du déplacement qui donnent un E élevé. Ainsi, en retenant le E minimal pour toutes les orientations testées dans une région donnée, seules les régions ayant un coin obtiendront une valeur finale supérieur à un certain seuil.

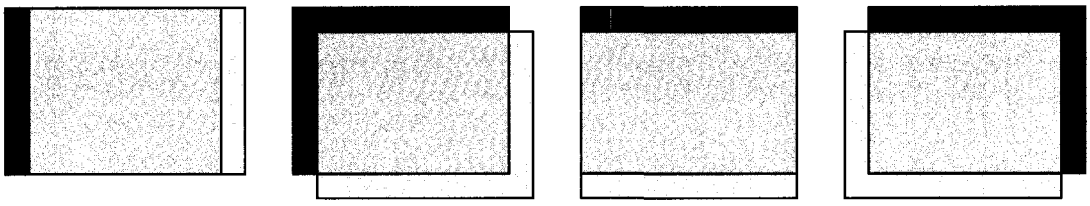

Figure 1.1 Les quatre déplacements suggérés par l'algorithme de Moravec : (1, 0), (1, 1), (0, 1) et (-1, 1). L'image en teinte foncée (rouge) est l'image d'origine et l'image semi-transparente en teinte pâle (jaune) est la copie déplacée.

Harris et Stephens ont donc proposé une méthode de recherche de points caractéristiques en corrigeant les lacunes de l'algorithme de Moravec. Les trois lacunes mentionnées sont : le fait que seuls quatre déplacements sont étudiés, une trop grande sensibilité au bruit et une trop grande réponse aux arêtes. Pour ce faire, l'équation 1.2 est reformulée par l'équation 1.4. Sachant que $(I_{u+x,v} - I_{u,v}) = \Delta I$, que $\Delta I/\Delta u \approx \partial I/\partial u$, que $\Delta u = x$ et que $\partial I/\partial u = \partial I/\partial x$, alors on peut appliquer le même traitement pour v et y afin d'effectuer la généralisation en deux dimensions :

$$(I_{u+x,v+y} - I_{u,v}) = \left(x\frac{\partial I}{\partial x}(u,v) + y\frac{\partial I}{\partial y}(u,v) + \mathrm{O}(x^2, y^2) \right), \tag{1.3}$$

où le terme d'erreur de l'approximation est exprimé par $\mathrm{O}(x^2, y^2)$.

$$E_{x,y} = \sum_{u,v} w_{u,v} \left(x\frac{\partial I}{\partial x}(u,v) + y\frac{\partial I}{\partial y}(u,v) + \mathrm{O}(x^2, y^2) \right)^2 \tag{1.4}$$

Ensuite, l'équation 1.4 se simplifie davantage en omettant le terme d'erreur et en utilisant une nouvelle notation.

$$E_{x,y} \approx \sum_{u,v} w_{u,v}\left(xX_{u,v} + yY_{u,v}\right)^2 \tag{1.5}$$

où

$$\begin{aligned} X &= I \otimes (-1,0,1) \approx \partial I/\partial x \\ Y &= I \otimes (-1,0,1)^{\mathrm{T}} \approx \partial I/\partial y \end{aligned} \tag{1.6}$$

où le symbole « ⊗ » représente le produit de convolution,

ce qui permet d'écrire

$$E(x,y) = Ax^2 + 2Cxy + By^2 \tag{1.7}$$

où

$$\begin{aligned} A &= X^2 \otimes w \\ B &= Y^2 \otimes w \\ C &= (XY) \otimes w \end{aligned} \tag{1.8}$$

Dans les équations de 1.8, w est une surface gaussienne qui permet de réduire le bruit. Les produits $X^2 = XX$, $Y^2 = YY$ et XY sont des produits terme à terme et non des produits matriciels. L'équation 1.7 peut s'écrire sous forme matricielle (équation 1.9).

$$E(x,y) = \begin{pmatrix} x & y \end{pmatrix} M \begin{pmatrix} x \\ y \end{pmatrix} \text{ où } M = \begin{pmatrix} A & C \\ C & B \end{pmatrix} \tag{1.9}$$

En pratique, x et y sont inconnus et peuvent être dans n'importe quelle direction. De plus, puisque A, B et C ont la taille de l'image d'origine, alors le reste de l'analyse s'effectue sur la matrice M seulement. Harris et Stephens concluent que le calcul final est $R = |M| - k * (Trace(M))^2$, ce qui donne l'équation 1.10. Dans cette dernière, le coefficient k est de l'ordre de 0.04 (Kovesi, 2003). Il suffit alors d'appliquer un seuillage sur R afin de trouver les coins.

$$R = \left(AB - C^2\right) - k * (A + B)^2 \tag{1.10}$$

Cet algorithme est accessible via la fonction *cvGoodFeaturesToTrack()* de la bibliothèque OpenCV (Intel, 2006).

1.2.1.1 Valeurs propres minimales

Le présent algorithme de recherche de points caractéristiques ressemble un peu à celui de Harris et Stephens (1988). Selon les fonctions *icvCalcHarris()* et *icvCalcMinEigenVal()* de la bibliothèque OpenCV, il suffit de modifier l'équation 1.10. Le résultat est l'équation 1.11. Encore une fois, cet algorithme est accessible via *cvGoodFeaturesToTrack()*.

$$R = \frac{(A+B) - \sqrt{(A-B)^2 + 4C^2}}{2} \tag{1.11}$$

1.2.2 SIFT

SIFT, ou Scale-Invariant Features Transform, est un type de point caractéristique développé par David G. Lowe (1999). Selon le chercheur, il y a essentiellement quatre étapes dans cet algorithme : la détection du point caractéristique, sa localisation précise, le calcul de son orientation et l'élaboration de son descripteur (David G. Lowe, 2004).

La détection des points caractéristiques se fait à même quatre niveaux de différences de Gaussiennes (DdG) appliquées sur une image en teintes de gris. La détection s'effectue aussi à diverses résolutions de l'image initiale.

Lorsqu'on veut éliminer certaines hautes fréquences dans une image, on utilise un filtre gaussien. Ce filtre est en fait une Gaussienne 2-D (équation 1.12 tirée de (David G. Lowe, 2004)) dont la variance est σ et dont la moyenne (0,0) est vis-à-vis le centre du filtre. En appliquant ce filtre sur l'image initiale en effectuant un produit de convolution, on obtient une image floue. La différence entre l'image initiale et cette image floue est justement le résultat de la DdG appliquée sur l'image initiale. En termes du domaine du traitement de signal, il s'agit d'un filtre passe-bas, mais on peut généraliser en disant qu'il s'agit d'un filtre passe-bande.

$$G(x,y,\sigma)=\frac{1}{2\pi\sigma^2}e^{-(x^2+y^2)/2\sigma^2} \tag{1.12}$$

Dans le cas de SIFT, les quatre DdG se suivent et le ratio entre deux σ est k : $\sigma_0 = \sigma_0$, $\sigma_1 = k\sigma_0$, $\sigma_2 = k^2\sigma_0$, $\sigma_n = k^n\sigma_0$. Selon Lowe, la valeur idéale de k est $\sqrt{2}$ et les quatre DdG retenues permettent d'étudier l'image au travers d'une *octave* complète.

En ayant ces différences d'images de plus en plus floues, on obtient, à chaque niveau ou à chaque bande de fréquences, des pixels minimums locaux et maximums locaux. Ces pixels sont une information spécifique à une fréquence spécifique, donc des points caractéristiques. Pour s'assurer de leur stabilité, il faut que les huit pixels autour aient tous une valeur inférieure (maximum local) ou supérieure (minimum local). De même, en empilant les DdG pour former un bloc de voxels[1], il faut que les neuf voxels du dessus et les neufs du dessous aient tous une valeur inférieure ou supérieure au maximum ou minimum local respectivement. Le point caractéristique ainsi trouvé a pour première description le niveau de DdG.

Dans le cas de SIFT, un point caractéristique peut représenter un disque plus grand qu'un pixel. Ainsi, le seul moyen pour identifier ce *gros* point caractéristique est de réduire la taille de l'image initiale et de recommencer l'exercice des quatre DdG : il s'agit de la seconde octave dans laquelle l'image initiale est réduite par un facteur de deux. Il peut donc y avoir plusieurs octaves de suite en divisant toujours l'image par un facteur de deux. Les points caractéristiques peuvent être décrits en fonction de l'octave dans laquelle on les a trouvés.

La seconde partie de l'algorithme consiste à localiser précisément les points caractéristiques. À l'aide de la valeur du point caractéristique et de celle de ses voxels voisins, Brown et Lowe (2002) ont modélisé une équation quadratique tridimensionnelle afin d'y rechercher la position exacte de l'extremum. Une fois que cette position précise est trouvée, il est même possible d'évaluer le niveau de contraste autour du point caractéristique : si le contraste est trop faible, le point sera rejeté étant donné qu'il s'agit

[1] Alors qu'un pixel est typiquement un petit carré de couleur, un voxel est un petit cube de couleur.

probablement d'un extremum dû au bruit. Enfin, puisqu'il est normalement incorrect et instable de fixer un point caractéristique à une position arbitraire sur une arête dans l'image, Lowe élimine les points sur les arêtes, c'est-à-dire là où le rayon de courbure est relativement très grand. À l'inverse, un point sur un coin est dans une région où la courbure est très prononcée.

Jusqu'à présent, l'algorithme SIFT ne dépend pas trop de la teinte des couleurs étant donné qu'on retient uniquement les extrémums à contraste élevé. Donc, même si un coin de la scène paraît différemment dans une image visible et une image infrarouge, SIFT arrive quand même à trouver le point dans chaque image. Maintenant, si le point de la scène n'est pas visible dans l'une ou l'autre des images (même couleur avec l'arrière-plan ou même température), il y a un risque de ne pas pouvoir former une paire de points. De plus, étant donné que les deux caméras sont différentes, la résolution de chacune peut être différente, ce qui fait en sorte qu'un même point de la scène puisse être trouvé dans des octaves différents et à des niveaux de DdG différents selon le type d'image. Cela risque d'avoir un impact sur l'appariement des points.

La troisième partie de l'algorithme SIFT est le calcul de l'orientation des points caractéristiques. Pour ce faire Lowe trouve l'image floue (image filtrée par une Gaussienne) la plus près du point caractéristique dans l'octave ayant permis de trouver ce point. Dans cette image floue, pour chaque pixel, on calcule la norme et l'orientation du gradient, et ce, en prenant les pixels voisins au-dessus, en dessous, à gauche et à droite. Pour décrire les points caractéristiques, on utilise un histogramme à 36 partitions angulaires couvrant 360 degrés. Dans cet histogramme, on accumule les orientations des gradients du voisinage du point caractéristique : chaque orientation de gradient est pondérée par la norme du gradient et par une Gaussienne 2-D dont le centre est le point caractéristique et dont la variance est 1.5 fois la variance σ utilisée pour rendre l'image floue. À partir de là, on sélectionne toutes les partitions de l'histogramme dont la valeur est à au moins 80% la valeur maximale. Dans chaque cas, on génère un point caractéristique : un même point caractéristique peut donc être dupliqué afin de supporter les multiples orientations à cette position dans l'image. Chaque orientation (partition)

retenue est finalement précisée en utilisant un modèle quadratique utilisant les données des deux partitions voisines à la partition retenue dans l'histogramme.

Cette orientation pour chaque point caractéristique permet donc d'apparier des points correspondants même s'ils n'ont pas la même orientation d'une image à l'autre. En effet, il reste à déterminer le descripteur de chaque point. Ce descripteur doit être invariant en fonction de l'orientation de chaque point dans une image, donc la construction du descripteur utilise l'orientation comme référentiel angulaire. Dans les faits, il faut encore une fois utiliser l'image floue associée au point caractéristique. On utilise un voisinage de 16 pixels par 16 pixels orienté selon l'orientation du point caractéristique. Encore une fois, on accumule des gradients, transformés selon le référentiel angulaire, dans 16 (4*4) boîtes de quatre par quatre pixels. Dans ces boîtes, on a un histogramme utilisant huit partitions angulaires. On obtient donc un descripteur à 128 données (16 boîtes de huit partitions).

En conclusion, puisqu'on utilise des gradients pour décrire les points caractéristiques SIFT et que les gradients sont dépendants des teintes de gris de chaque image initiale, il est généralement impossible d'apparier des points caractéristiques dans une image visible avec des points caractéristiques d'une image infrarouge. Dans la figure 1.2, la majorité des paires de points sont aberrantes. En fait, la seule paire valide et facilement identifiable est celle qui rejoint le coin gauche de la lumière de gauche au plafond.

1.2.2.1 PCA-SIFT

L'algorithme PCA-SIFT (principal components analysis SIFT), présenté par Ke et Sukthankar (2004), tente d'améliorer le descripteur de l'algorithme SIFT présenté dans la section précédente. Cette fois, le voisinage, centré sur chaque point caractéristique et orienté en fonction de celui-ci, a une taille de 41 par 41 pixels au lieu de 16 par 16 pixels dans le cas de SIFT. À l'aide des 41 par 41 pixels, on calcule les gradients horizontaux (selon l'orientation du point caractéristique) et verticaux. Donc, on a deux valeurs pour chaque pixel. Or, le calcul requiert deux pixels voisins pour chaque gradient :

f'(x) = (f(x+1)-f(x-1))/2. Ainsi, on obtient 39*39 gradients horizontaux et verticaux, donc un total de 3042 valeurs pour le descripteur de PCA-SIFT.

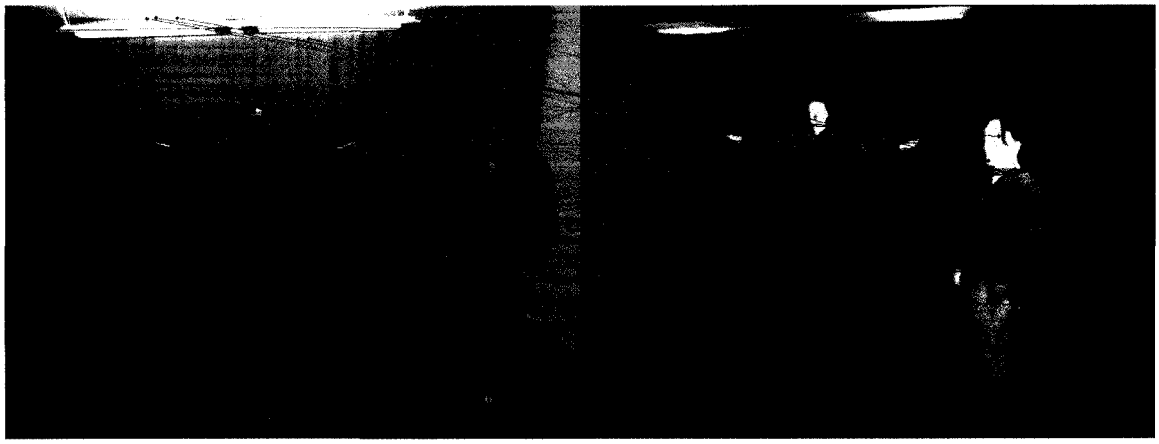

Figure 1.2 Exemple d'appariement de points avec l'algorithme complet de SIFT. Calculs effectués par l'implémentation proposée dans (Vedaldi, 2006)

Cette quantité d'information est trop élevée pour un simple descripteur, alors Ke et Sukthanka (2004) ont effectué une analyse par composantes principales sur une base de données de 21 000 régions de 41 par 41 pixels. L'analyse des composantes principales consiste à faire ressortir mathématiquement les points communs entre les diverses régions. Ces points communs sont ce qu'on appelle les composantes principales et elles sont classées par ordre décroissant de pertinence. Ainsi, en obtenant les principales composantes des descripteurs de 3042 valeurs, il est possible d'approximer ces immenses descripteurs avec les 36 composantes les plus significatives.

Sans avoir testé cet algorithme, nous anticipons un résultat semblable à celui affiché à la figure 1.2 étant donné que c'est un algorithme basé sur SIFT. Néanmoins, selon Ke et Sukthanka (2004), leur algorithme donne de meilleurs résultats que l'algorithme SIFT.

1.2.2.2 GLOH

Dans l'algorithme SIFT à la section 1.2.2, on a vu que le descripteur final était construit à partir de 16 régions carrées (4 par 4) contenant un histogramme de huit partitions angulaires. Les 16 régions carrées représentent donc les gradients sur un voisinage de 16 par 16 pixels centré sur chaque point caractéristique.

L'algorithme *Gradient location-orientation histogram* (GLOH), mentionné et sommairement décrit par Mikolajczyk et Schmid (2005), utilise plutôt des régions angulaires comme une cible au jeu de fléchettes : trois divisions radiales de rayon 6, 11 et 15 pixels et huit divisions angulaires du rayon 6 au rayon 15 (voir la figure 1.3). Ainsi, on a le disque central ayant un rayon de 6 pixels et deux autres niveaux de huit partitions angulaires. On obtient donc un total de 17 régions angulaires et radiales avec GLOH au lieu de 16 régions carrées avec SIFT. De plus, à l'intérieur de chaque région, GLOH utilise 16 partitions d'orientation de gradients au lieu de huit pour SIFT. Bref, le descripteur brut contient 272 valeurs (17 * 16).

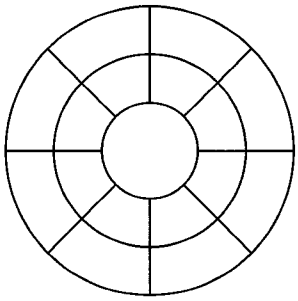

Figure 1.3 Représentation des 17 partitions angulaires et radiales dans l'algorithme GLOH

La taille des descripteurs est cependant trop élevée et GLOH réduit ce nombre de valeurs à 128 grâce à une analyse en composantes principales (PCA) en prétraitement. En effet, en analysant 47 000 images, Mikolajczyk et Schmid ont trouvé les 128 composantes principales des descripteurs à 272 valeurs. Donc, pour chaque nouveau descripteur à 272 valeurs, il suffit de trouver le coefficient idéal pour chacune des 128 composantes principales, et ce, pour approximer le descripteur à 272 valeurs.

Sans avoir testé cet algorithme, nous anticipons un résultat semblable à celui affiché à la figure 1.2 étant donné que c'est un algorithme basé sur SIFT. De plus, selon les résultats de Mikolajczyk et Schmid (2005), SIFT et GLOH sont à peu près équivalents en terme du nombre de bons appariements.

1.2.2.3 SURF

SURF, *Speeded Up Robust Features*, présenté par Bay, Tuytelaars et Van Gool (2006), est un algorithme comparable à SIFT puisqu'on tient compte de l'orientation du point caractéristique pour calculer son descripteur final. Cependant, la différence est que l'algorithme SURF utilise l'ondelette de Haar dans les DdG pour déterminer l'orientation dominante du point caractéristique et pour calculer les descripteurs.

Pour calculer le descripteur, le voisinage carré du point caractéristique, orienté selon l'orientation de ce dernier, est divisé en 16 régions carrées (4 par 4). Jusque-là, c'est comme pour l'algorithme SIFT. Ensuite, en subdivisant chaque région carrée en 25 cases (5 par 5), on obtient l'information nécessaire pour appliquer l'ondelette de Haar horizontalement et verticalement. On obtient respectivement des valeurs d_x et d_y. Dans chaque région, Bay, Tuytelaars et Van Gool définissent le descripteur comme étant la somme des valeurs d_x et d_y, mais aussi la somme de leur valeur absolue : Σd_x, Σd_y, $\Sigma|d_x|$, $\Sigma|d_y|$. Donc, on a 16 régions de 4 valeurs, ce qui donne des descripteurs de 64 valeurs pour chaque point caractéristiques.

SURF a aussi une version à 128 valeurs par descripteurs. Étant donné que les coefficients des ondelettes horizontales et verticales de Haar correspondent, les d_x sont classés selon les d_y et vice-versa : $\Sigma(d_x : d_y<0)$, $\Sigma(d_x : d_y\geq 0)$, $\Sigma(d_y : d_x<0)$, $\Sigma(d_y : d_x\geq 0)$, $\Sigma|d_x : d_y<0|$, $\Sigma|d_x : d_y\geq 0|$, $\Sigma|d_y : d_x<0|$, $\Sigma|d_y : d_x\geq 0|$.

1.2.3 Divers types de points caractéristiques

Puisque la littérature est riche en méthodes pour trouver des points caractéristiques, la présente section contient une brève description de quelques-unes des méthodes les plus connues.

1.2.3.1 SUSAN

L'algorithme SUSAN, *Smallest Univalue Segment Assimilating Nucleus*, est développé par Smith et Brady (1997). Il s'agit d'un algorithme un peu plus orienté vers la recherche des arêtes dans l'image. Évidemment, les coins font aussi partie des résultats.

Cependant, l'algorithme ne suggère pas de descripteur permettant l'appariement des points caractéristiques.

L'algorithme SUSAN utilise un filtre présenté à l'équation 1.13 pour déterminer si un point fait partie d'un bord d'objet ou non. Cette équation est le score de chaque pixel $I(u, v)$ de l'image d'origine qui est transformée en teintes de gris. Pour chaque pixel $I(u', v')$ dans un voisinage circulaire $V_{u,v}$ centré sur $I(u, v)$, on calcule la différence entre $I(u', v')$ et le pixel central $I(u, v)$. Cette différence est normalisée par rapport à un seuil t. On applique l'exposant six à cette différence normalisée. Le résultat est forcément positif à cause de l'exposant pair. En appliquant l'exponentielle de base naturelle au négatif du résultat précédent, on se retrouve avec des valeurs près de un pour des pixels $I(u', v')$ et $I(u, v)$ semblables. Autrement, $I(u', v')$ et $I(u, v)$ sont différents et l'exponentielle tendra vers zéro. En effectuant la somme des exponentielles pour chaque $I(u', v')$ dans $V_{u,v}$, on obtient un score total dont la valeur maximale possible est la surface de $V_{u,v}$. Ce score est défini par

$$S(u,v) = \sum_{(u',v') \in V_{u,v}} e^{-\left(\frac{I(u',v') - I(u,v)}{t}\right)^6} . \qquad (1.13)$$

Selon Smith et Brady, lorsque $S(u, v)$ vaut moins de 75% de la surface de $V_{u,v}$, alors on a un point près d'un bord d'objet. En effet, le seuil de 75% permet de trouver les pixels près des arêtes et des coins. Étant donné que nous recherchons surtout les coins pour le présent projet, il est préférable de garder les points tels que $S(u, v)$ est inférieur à 33% de la surface de $V_{u,v}$. En d'autres mots, ce dernier critère cherche des coins d'au plus 120 degrés dont la teinte de gris est semblable à celle du pixel central de $V_{u,v}$. Le résultat est présenté à la figure 1.4.

1.2.3.2 La méthode de Kadir et Brady

Instinctivement, l'être humain arrive à identifier des points spéciaux dans une image contenant rien de nécessairement reconnaissable. Il s'agit d'irrégularités, de raretés, de régions complexes, etc. Ces régions sont identifiées par la méthode de Kadir et Brady

(2001) qui cherche les endroits dans l'image où l'entropie de l'intensité des pixels est élevée dans un voisinage de taille fixe. De plus, cette recherche de raretés s'effectue dans des clones de plus en plus petits de l'image d'origine, et ce, afin de trouver des éléments particuliers de différentes grosseurs.

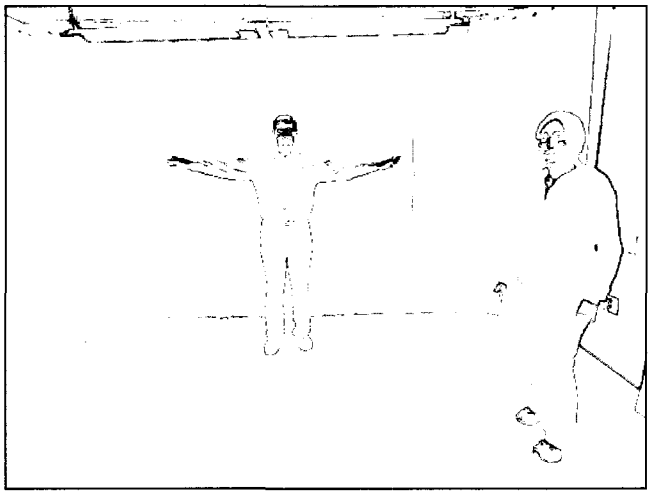

Figure 1.4 Exemple d'application de l'algorithme SUSAN

Dans une paire d'images, si un point spécial se trouve dans une des images, il est normal d'espérer retrouver le point correspondant semblable dans l'autre image. Malgré le potentiel non négligeable de la méthode, celle-ci n'arrive pas à trouver suffisamment de points correspondants dans une paire d'images infrarouges selon les résultats de Hajebi et Zelek (2006). Par contre, les résultats positifs de Kadir et Brady montrent des coefficients de corrélation élevés pour des paires de régions correspondantes : ces régions viennent du même type d'images, c'est-à-dire des images du spectre visible. Bref, bien que la méthode semble avoir un bon potentiel pour des images du spectre visible, nous risquons de ne pas avoir autant de succès étant donné que, dans notre projet, la nature des images est différente d'une image à l'autre et que les résultats de Hajebi et Zelek montrent une faiblesse de la méthode pour des images infrarouges.

1.2.3.3 PC2 - Congruence de phase

La congruence de phase (PC_2) (Kovesi, 2000b, 2003) est un bon algorithme de recherche de points caractéristiques pour des paires d'images infrarouges (Hajebi & Zelek, 2006,

2007). En effet, selon Hajebi et Zelek (2006), leurs expériences montrent que PC_2 donne de meilleurs points caractéristiques pour la vision stéréo que l'algorithme SIFT et que les détecteurs de points de Kadir et de Harris. Selon Kovesi, son algorithme est meilleur que l'opérateur de Harris qui se base sur des gradients et un seuil pour trouver les points caractéristiques : certains points caractéristiques valables peuvent être ignorés à cause des faibles gradients dans leur voisinage. Toujours selon lui, PC_2 est aussi plus général que l'algorithme SUSAN (section 1.2.3.1) qui considère que « les arêtes et les coins sont formés par des régions dont la luminosité est relativement constante » (Kovesi, 2003). Alors, qu'en est-il exactement?

Le principe de base est de trouver les endroits dans le signal (image) où les différentes fréquences sont en phase (figure 1.5). Ainsi, la congruence de phase peut donc trouver des points caractéristiques là où les gradients sont faibles, puisque tout est une question de fréquences en phase : même si l'amplitude des fréquences est faible, ce n'est pas cela qui compte.

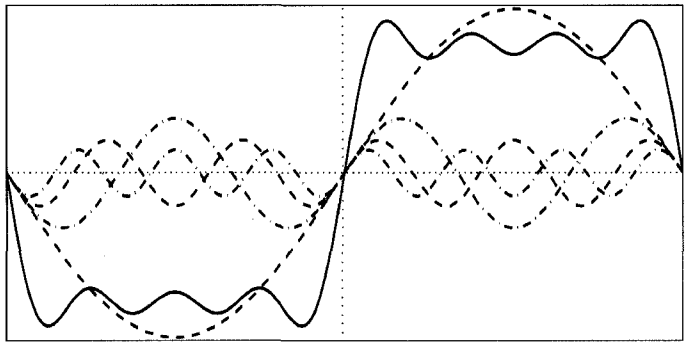

Figure 1.5 Signal carré décomposé en différentes fréquences qui sont en phase là où le gradient est le plus fort dans le signal carré. Image tirée de (Kovesi, 2003)

La congruence de phase de base se calcule avec l'équation 1.14. Chacun des termes est représenté à la figure 1.6. Dans cette figure, toutes les fréquences sont représentées vectoriellement en fonction de leur phase ($\phi_n(x)$) et de leur amplitude ($A_n(x)$). En sommant tous ces vecteurs vectoriellement, on obtient la phase moyenne du signal complet ($\bar{\phi}(x)$), ainsi que son amplitude globale ($|E(x)|$). Selon l'équation 1.14, si l'amplitude de $E(x)$ est près de la somme de l'amplitude des fréquences, alors la congruence est près de 1, donc élevée. Par contre, si les phases sont différentes et que la

sommation des vecteurs aboutit dans le cercle en pointillés dans la figure 1.6, alors le point étudié est dans une zone trop bruitée dans le signal pour être considéré comme un point caractéristique valable. Bref, puisque l'amplitude globale est divisée par la somme des amplitudes, l'intensité des gradients n'a donc aucun impact sur cet algorithme.

$$PC_1(x) = \frac{|E(x)|}{\sum_n A_n} = \frac{\sum_n A_n \cos(\phi_n(x) - \bar{\phi}(x))}{\sum_n A_n} \tag{1.14}$$

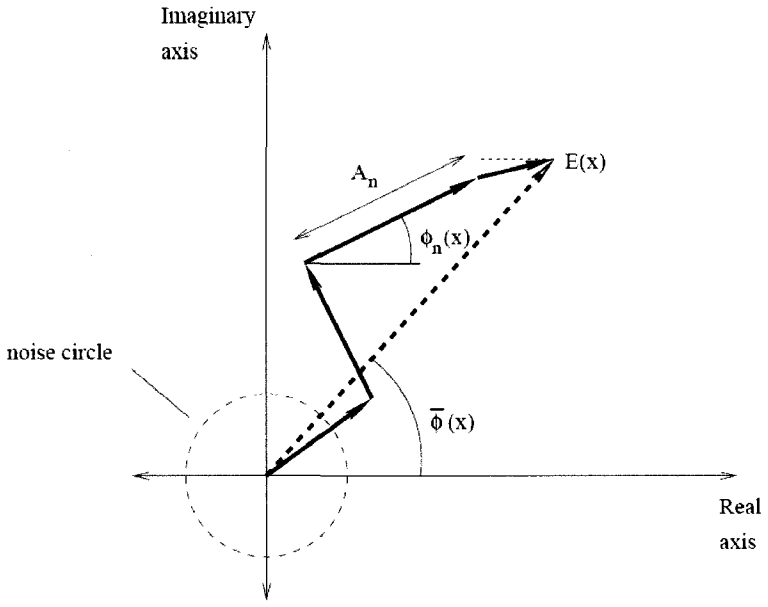

Figure 1.6 Sommation de phases et leur amplitude pour trouver la phase moyenne et l'amplitude totale. Image tirée de (Kovesi, 2003)

Dans les faits, Kovesi utilise une équation un peu plus sophistiquée que l'équation 1.14. Dans l'équation 1.15, $W(x)$ est un poids pour chaque fréquence, l'opérateur $\lfloor \ \rfloor$ tronque son contenu à zéro si le contenu est négatif, le terme en sinus absolu permet d'avoir une réponse plus fine de l'opérateur PC_2, T est le seuil du bruit (le cercle en pointillés dans la figure 1.6) et ε est une très faible valeur pour éviter les divisions par zéro.

$$PC_2(x) = \frac{\sum_n W(x) \lfloor A_n (\cos(\phi_n(x) - \bar{\phi}(x)) - |\sin(\phi_n(x) - \bar{\phi}(x))|) - T \rfloor}{\sum_n A_n + \varepsilon} \tag{1.15}$$

La généralisation de cet algorithme en deux dimensions pour les images est décrite dans (Kovesi, 2000b). Entre autres, on utilise des filtres Log-Gabor (Kovesi, 2007) pour

identifier la phase et l'amplitude du signal dans diverses orientations et à diverses fréquences. Par défaut, Kovesi utilise six orientations différentes et quatre plages de fréquences différentes isolées grâce à l'ondelette Log-Gabor (Kovesi, 2003). Au total, chaque pixel de l'image source peut être décrit par 24 (6*4) coefficients Log-Gabor (Hajebi & Zelek, 2006).

Kovesi se sert des coefficients Log-Gabor pour calculer des valeurs propres maximales (équation 1.17) et minimales (équation 1.18) similaires à celle de l'équation 1.11. Pour chaque orientation, Kovesi calcule le PC_2 à l'aide des quatre plages de fréquences. En évaluant les équations 1.16, on se retrouve avec les valeurs d'une matrice de covariances (comme celle de l'équation 1.9) des congruences de phases pour toutes les orientations θ_i. Dans (Hajebi & Zelek, 2006), ils ont utilisé l'équation 1.17, ce qui donne une réponse forte sur les arêtes dans les images. Par contre, l'équation 1.18 donne, tout comme l'équation 1.11, une réponse forte pour les coins. Toutes ces équations sont tirées du travail de Kovesi (2003).

$$\begin{aligned} a &= \sum_i (PC_2(\theta_i)\cos(\theta_i))^2 \\ b &= 2\sum_i (PC_2(\theta_i)\cos(\theta_i)) \cdot (PC_2(\theta_i)\sin(\theta_i)) \\ c &= \sum_i (PC_2(\theta_i)\sin(\theta_i))^2 \end{aligned} \tag{1.16}$$

$$M = \frac{1}{2}\left(c + a + \sqrt{b^2 + (a-c)^2}\right) \tag{1.17}$$

$$m = \frac{1}{2}\left(c + a - \sqrt{b^2 + (a-c)^2}\right) \tag{1.18}$$

Avec un seuillage, il est possible d'identifier les points caractéristiques à même les images M et m qui répondent fortement aux arêtes et aux coins respectivement. La figure 1.7 présente le résultat des points trouvés dans m pour chaque image. On remarque qu'il y a des points dans l'image de gauche qui sont présents dans l'image de droite. Si on prend, par exemple, les points vis-à-vis les aisselles de l'acteur de gauche, la chemise de l'acteur a une couleur plus foncée que l'arrière-plan dans l'image de gauche et une

température plus élevée que l'arrière-plan dans l'image de droite. Les points ne sont pas correctement appariés dans la figure 1.8 à cause de l'algorithme d'appariement par corrélation de Kovesi (2000a). C'est un signe que le descripteur utilisé n'était pas adéquat, même si la congruence de phase peut trouver des points caractéristiques pertinents. Le descripteur utilisé par Hajebi et Zelek, présenté à la section 1.3.2.3, est peut-être la solution à ce problème d'appariement, mais il se base aussi sur l'apparence de chaque image : le descripteur en question est l'ensemble de coefficients Log-Gabor pour chaque pixel.

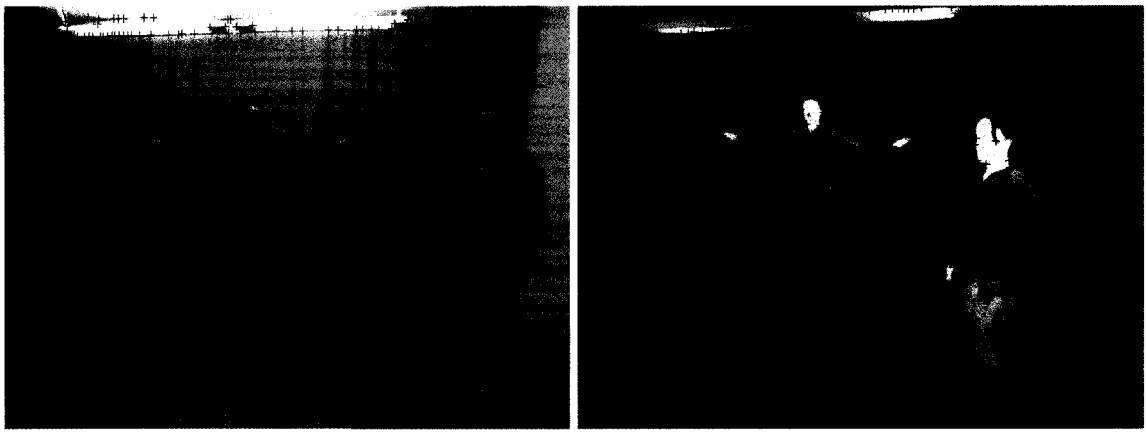

Figure 1.7 Exemple de points caractéristiques trouvés dans chaque image par l'algorithme de congruence de phase

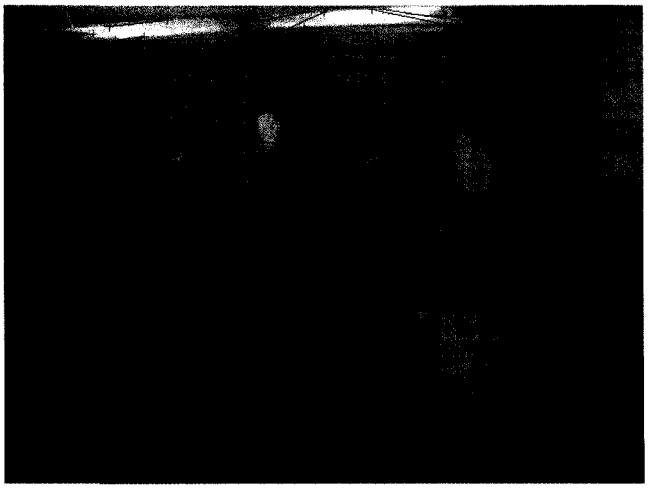

Figure 1.8 Exemple d'appariement des points de la figure 1.7 avec l'implémentation de Kovesi (2000a) pour une image visible et une image infrarouge

Finalement, la congruence de phase est un algorithme très lourd puisqu'il requiert typiquement 49 transformées de Fourier. À moins de diviser la tâche sur une grappe de calculs, le code Matlab[1] prend présentement une dizaine de secondes par image sur une machine AMD Opteron à 2.4 GHz à double noyau.

1.2.3.4 FAST

L'algorithme FAST, *Features from Accelerated Segment Test*, est un algorithme proposé par Rosten et Drummond (2005) et une optimisation est proposée par Edward Rosten et Drummond (2006). L'algorithme final est conçu pour des applications spécifiques et en temps réel.

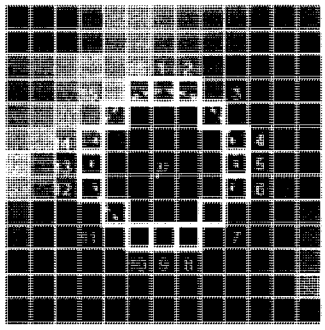

Figure 1.9 Un pixel p et les 16 pixels qui l'entoure. En pointillés, on a un arc de 12 pixels continus de plus faible intensité que p. Image tirée de (Edward Rosten & Drummond, 2006)

Pour un pixel p quelconque, on le compare avec les 16 pixels sur un cercle qui l'entoure (figure 1.9). Si on parvient à trouver, sur ce cercle, un arc de 12 pixels continus qui sont tous plus pâles que p ou tous plus foncés que p, alors p est un coin. Telle est la version classique et simpliste de l'algorithme FAST.

Afin d'améliorer la vitesse de calcul de cet algorithme, Rosten et Drummond (2006) ont décidé de créer un arbre de décision à partir d'images d'entraînement. Ces images d'entraînement sont choisies en fonction de chaque application. Dans notre cas, on pourrait avoir deux arbres de décision différents : un pour les images visibles et un autre pour les images infrarouges.

---

[1] Une implémentation en C++ du même algorithme n'est pas plus rapide étant donné que les transformées de Fourier sont déjà optimisées dans Matlab.

L'arbre en question est construit en déterminant à chaque fois laquelle des 16 positions x sur le cercle est généralement la plus adéquate pour prendre une décision à propos de tous les pixels p de toutes les images dans l'ensemble d'entraînement. Le critère pour déterminer la position x à choisir est une fonction d'entropie tenant compte du nombre de pixels p qui sont des coins et du nombre de ceux qui ne le sont pas, ainsi que les pixels p qui sont plus foncé, plus pâle ou équivalent au pixel vis-à-vis la position x (voir le travail de Edward Rosten et Drummond (2006) pour plus de détails sur la fonction d'entropie). Donc, en testant chaque position x sur le cercle, on évalue leur entropie totale et on note la position x donnant la plus grande entropie. Ceci permet donc de séparer l'ensemble des p en fonction de leur valeur par rapport au pixel vis-à-vis x : les plus foncés, les plus pâles et les similaires. Dans chacun de ces trois groupes, on trouve individuellement la prochaine position x parmi les 15 positions restantes sur le cercle. L'algorithme continue jusqu'à ce que l'entropie tombe à zéro pour toutes les positions x restantes d'un sous-ensemble de tous les pixels p de toutes les images d'entraînement : ce sous-ensemble contient alors que des coins ou que des pixels qui ne sont pas des coins.

L'arbre de décision est ensuite converti en code source donnant une immense structure de décisions imbriquées. En effet, l'arbre est codé et compilé dans l'exécutable, d'où la grande rapidité de l'algorithme.

1.2.3.5 KLT

L'algorithme KLT est développé par Kanade, Lucas et Tomasi (Lucas & Kanade, 1981; Shi & Tomasi, 1994; Tomasi & Kanade, 1991). Il s'agit essentiellement d'une technique développée pour trouver et suivre des points caractéristiques d'une image à l'autre dans une même vidéo : on ne peut donc pas l'appliquer à des paires d'images de différents types comme dans notre cas.

La technique commence par effectuer une étude du déplacement des éléments de la scène d'une image à l'autre : calcul du flux optique. Cette étude se base sur le fait que la plupart des pixels d'une image se retrouvent dans l'autre image avec un certain déplacement. Ensuite, on calcule les points caractéristiques à l'aide de la valeur propre

minimale d'une matrice « de suivi » (Tomasi & Kanade, 1991) de taille deux par deux construite pour chaque voisinage de 15 par 15 pixels possibles dans la première image. Ces points caractéristiques sont ensuite recherchés dans la seconde image.

1.2.3.6 Moments invariants

Selon Mikolajczyk et Schmid (2005), les divers moments sont généralisés par l'équation ci-dessous (1.19). Dans cette équation, $p+q$ est l'ordre qui est soit 1 ou 2 (01, 10, 02, 11 et 20 pour pq respectivement). De même, a est le degré qui est soit 1 ou 2. Enfin, d est le sens des dérivées sur l'image I, donc d est selon x ou selon y. On a donc un total de 20 (5*2*2) valeurs par descripteur pour un voisinage de Δx par Δx.

$$M_{pq}^{a} = \frac{1}{\Delta x \Delta y} \sum_{x,y} x^{p} y^{q} \left(I_{d}(x,y)\right)^{a} \tag{1.19}$$

Dans le travail de Shah, Aggarwal, Eledath et Ghosh (1997), on décrit un autre ensemble de moments invariants. On définit d'abord les moments standards :

$$M_{pq} = \sum_{x,y} x^{p} y^{q} I(x,y)\,. \tag{1.20}$$

Les centroïdes sont donc définis par $\bar{x} = M_{10}/M_{00}$ et $\bar{y} = M_{01}/M_{00}$. Ils permettent d'obtenir des moments invariants par rapport à la translation :

$$\mu_{pq} = \sum_{x,y} (x-\bar{x})^{p} (y-\bar{y})^{q} I(x,y)\,. \tag{1.21}$$

Enfin, on peut définir des moments invariants selon les redimensionnements :

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\lambda}}, \tag{1.22}$$

où $\lambda = 1 + (p + q) / 2$. Shah, Aggarwal, Eledath et Ghosh suggèrent d'utiliser des moments d'ordre deux et d'ordre trois pour ne pas avoir une trop grande sensibilité au bruit dans l'image.

1.3 Appariement des points caractéristiques

Une fois que les points caractéristiques sont trouvés, il faut les apparier. Il s'agit donc d'une tâche très importante dans la vision stéréo étant donné que c'est à la base de toute reconstruction de la scène : il faut des paires de points fiables.

Globalement, la technique consiste à prendre chaque point de l'image de gauche et de chercher le point correspondant dans l'image de droite. Et vice-versa. Pour ce faire, il y a deux types de techniques : l'appariement basé sur la corrélation (section 1.3.1) et l'appariement basé sur les points caractéristiques (section 1.3.2). Dans cette dernière technique, on utilise une matrice de correspondance (section 1.3.2.1) pour déterminer les paires de points. Enfin, ce type de matrice contient des scores (sections 1.3.2.2 et 1.3.2.3).

1.3.1 Appariement basé sur la corrélation

Typiquement, en vision stéréo, on utilise deux caméras semblables positionnées de telle sorte que les différents éléments de la scène paraissent plus ou moins au même endroit dans chacune des images de la paire d'images : plus un élément *bouge* d'une image à l'autre, plus la disparité est grande. Ainsi, en prenant un point caractéristique $[i, j]^T$ dans l'image de gauche, on peut rechercher le point correspondant vis-à-vis la même coordonnée dans l'autre image et dans un certain voisinage V (figure 1.10). Il s'agit d'une première méthode présentée par Trucco et Verri (1998, pp. 146-147).

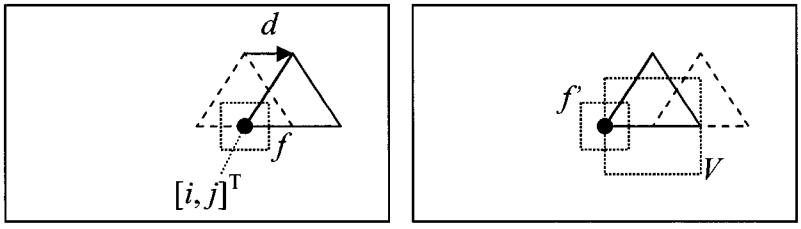

Figure 1.10 Paire stéréo contenant un triangle en trait continu. Le triangle de l'autre image est en pointillés longs. La disparité est d. Le voisinage de recherche est V en pointillés courts. La fenêtre pour le produit de corrélation est f et f' en pointillés courts.

Dans le voisinage V, lorsqu'on teste deux points de correspondance potentiels, il faut calculer la corrélation des régions f et f' : alors que f est centré sur $[i, j]^T$, la corrélation sera maximisée uniquement à l'endroit où le centre de f' sera le point caractéristique

cherché dans l'image de droite. Il faut donc tester plusieurs disparités $d = [d1, d2]^T$ afin de maximiser la corrélation totale $c(d)$ présentée à l'équation 1.23 tirée de l'ouvrage de Trucco et Verri (1998). Dans cette équation, on remarque que les deux régions f et f' ont une taille de $2w+1$ et la fonction de corrélation est représentée par une fonction $\psi(u, v)$. Selon Trucco et Verri, cette fonction peut être de la forme $\psi(u, v) = uv$ ou $\psi(u, v) = -(u-v)^2$.

$$c(d) = \sum_{k=-w}^{w} \sum_{l=-w}^{w} \psi\left(I_g(i+k, j+l), I_d(i+k-d_1, j+l-d_2)\right) \tag{1.23}$$

Évidemment, cette méthode fonctionne lorsque les images sont du même type de capteur et lorsque les disparités ne sont pas trop grandes. De plus, cet algorithme est lourd, puisqu'il faut pratiquement chercher toutes les possibilités sur V et calculer la corrélation totale à chaque fois.

1.3.2 Appariement basé sur les points caractéristiques

Il s'agit de la seconde méthode présentée par Trucco et Verri (1998). L'idée est d'apparier les points caractéristiques de chaque image, et ce, en fonction des descripteurs (éléments de description) de chaque point. Les descripteurs sont généralement propres à chaque type de point caractéristique. L'avantage de cette méthode est que l'espace de recherche est grandement réduit par rapport au voisinage V de la section précédente. Le désavantage est qu'il est possible d'y avoir des paires sévèrement aberrantes si les descripteurs ne sont pas suffisamment discriminatoires pour éviter les erreurs d'appariement.

1.3.2.1 Matrice de correspondance

Ici, chaque point caractéristique de l'image de gauche est comparé aux points de l'image de droite. Le résultat de la comparaison est un score numérique. Le score de chaque paire est donc recensé dans une matrice appelée « matrice de correspondance ». Chaque ligne de la matrice correspond à un point de l'image de gauche et chaque colonne correspond à un point de l'image de droite. Une fois que la matrice est remplie, il faut

déterminer les paires de points en fonction des scores dans la matrice. Il faut alors favoriser les scores maximaux, c'est-à-dire les paires de points ayant eu le score le plus élevé possible.

En fait, il y a plusieurs méthodes pour déterminer les paires de points avec la matrice de correspondance, mais nous en avons vu qu'une seule dans la littérature : il s'agit du choix mutuel. Le choix mutuel est la méthode utilisée par Hajebi et Zelek (2006) et dans le travail de Kovesi (2000a).

Le choix mutuel consiste à chercher pour chaque point d'une image son point favori dans l'autre image, et ce, pour chaque image. Pour chaque ligne dans la matrice, c'est-à-dire que pour chaque point de l'image de gauche, il faut trouver le score maximal dans cette ligne, c'est-à-dire le point favori de l'image de droite. Pour chaque colonne dans la matrice, c'est-à-dire pour chaque point de l'image de droite, il faut trouver le score maximal dans cette colonne, c'est-à-dire le point favori de l'image de gauche. Enfin, pour chaque point de l'image de gauche, s'il est le favori de son favori (les points de l'image de gauche et de l'image de droite se choisissent mutuellement), il y a donc un choix mutuel.

Le choix mutuel permet d'utiliser chaque point caractéristique au plus une seule fois. Cela respecte une contrainte voulant qu'un point dans une image soit associé à au plus un point de l'autre image (Hajebi & Zelek, 2006, p. 4). Par ailleurs, le choix mutuel est relativement très fiable dans la formation des paires de points, puisqu'il y a une double vérification de la paire étant donné que les deux points se sont choisis mutuellement. Cependant, un point (A) ayant choisi un autre point (B) faisant partie d'une paire (B, C) sera alors rejeté même si (A) était un point caractéristique pertinent pour représenter la scène étudiée.

1.3.2.2 Produit de corrélation sur les points de l'image

À la section 1.3.1, les paires de points étaient trouvés par essais et erreurs à l'aide du calcul de la corrélation de la région f autour de chaque point dans une paire. Ici, on répète sensiblement le même calcul de corrélation. Cependant, il n'y a pas d'essai ni

erreur, mais plutôt un calcul de corrélation pour chaque paire possible de points et les résultats sont ensuite compilés dans une matrice de correspondance.

Selon le travail de Kovesi (2000a), ce dernier commence par calculer la moyenne du voisinage (de taille $w * w$) de chaque pixel cible (en niveau de gris) et soustrait cette moyenne au pixel cible en question. Il ne reste donc que les hautes fréquences dans chaque image (im_g et im_d). Ensuite, le calcul du score de la matrice de correspondance pour chaque paire de points caractéristiques $[x,y]^T$ est décrit à l'équation 1.24. Dans cette équation, on utilise un seuil d_{max} qui correspond à la distance maximale permise entre les points d'une paire potentielle lorsqu'on superpose les deux images.

$$score\left(\begin{bmatrix} x_g \\ y_g \end{bmatrix}, \begin{bmatrix} x_d \\ x_d \end{bmatrix}\right) = \begin{cases} -\infty & \text{si la distance entre les points} \geq d_{\max} \\ c\left(\begin{bmatrix} x_g \\ y_g \end{bmatrix}, \begin{bmatrix} x_d \\ x_d \end{bmatrix}\right) & \text{sinon} \end{cases}$$

$$\text{où } c\left(\begin{bmatrix} x_g \\ y_g \end{bmatrix}, \begin{bmatrix} x_d \\ x_d \end{bmatrix}\right) = \frac{\sum_{i=-r}^{r}\sum_{j=-r}^{r} im_g(x_g + j, y_g + i) im_d(x_d + j, y_d + i)}{\sqrt{\sum_{i=-r}^{r}\sum_{j=-r}^{r} im_g(x_g + j, y_g + i)^2 \sum_{i=-r}^{r}\sum_{j=-r}^{r} im_d(x_d + j, y_d + i)^2}} \quad (1.24)$$

$$\text{où } r = (w-1)/2$$

Enfin, les paires de points sont déterminées par choix mutuel.

1.3.2.3 Produit de corrélation sur les coefficients Log-Gabor

Dans l'article de Hajebi et Zelek (2006), ils ont plutôt opté pour un produit de corrélation sur les coefficient Log-Gabor de chaque pixel au lieu du voisinage de chaque pixel. En effet, ils ont trouvé leurs points caractéristiques grâce à l'algorithme de congruence de phase (section 1.2.3.3), algorithme qui calculait par défaut 24 coefficients Log-Gabor pour chaque pixel. Ainsi, deux points correspondants devraient avoir sensiblement les mêmes coefficients Log-Gabor, d'où la définition du score à l'équation 1.25 tirée de (Hajebi & Zelek, 2006). Dans cette équation, F contient les coefficients Log-Gabor f_j.

$$score(F_g, F_d) = \frac{\sum_j f_{g,j} f_{d,j}}{\sqrt{\sum_j f_{g,j}^2 f_{d,j}^2}} \tag{1.25}$$

Le problème dans l'équation 1.25 est que le dénominateur semble manquer un symbole de sommation. Pourtant, le dénominateur de l'équation 1.24 sépare bien la sommation des pixels de im_g de la sommation des pixels de im_d. D'ailleurs, Hajebi et Zelek se sont référés à Wiskott, Fellous, Kruger et von der Malsburg (1997) qui avaient utilisé deux sommations au dénominateur au lieu d'une seule. Bref, l'équation 1.25 devrait plutôt être comme à l'équation 1.26. Néanmoins, puisque l'erreur semble se répéter dans (Hajebi & Zelek, 2007), l'équation 1.25 est peut-être valide après tout.

$$score(F_g, F_d) = \frac{\sum_j f_{g,j} f_{d,j}}{\sqrt{\sum_j f_{g,j}^2 \sum_j f_{d,j}^2}} \tag{1.26}$$

1.3.2.4 Appariement par distance euclidienne

Dans le cas de l'algorithme SIFT (section 1.2.2), on a typiquement une image de référence et une image d'une scène dans laquelle on doit rechercher l'objet présent dans l'image de référence. Les points caractéristiques trouvés dans l'image de référence sont classés dans un index. Ensuite, les points caractéristiques de l'image de la scène sont appariés à certains points de l'index, et ce, à l'aide d'un critère de distance euclidienne (David G. Lowe, 2004). La méthode s'optimise par l'usage de groupements dans l'index et par des recherches par voisinage ou via des arbres de recherche probabilistes.

1.4 Filtrage des paires de points caractéristiques

Malgré toutes les précautions prises pour trouver de bonnes paires de points caractéristiques, il reste toujours des paires de points qui sont invalides. Il faut donc filtrer les paires de points à l'aide d'un algorithme RANSAC (section 1.4.1) utilisant la géométrie épipolaire (section 1.4.2) pour modéliser la transformation linéaire liant chaque point dans une paire de points (sections 1.4.2.1 et 1.4.2.2).

1.4.1 Algorithme RANSAC

L'algorithme RANSAC veut dire « RANdom SAmple Consensus » (Fischler & Bolles, 1981). Il s'agit d'un algorithme général permettant de modéliser un ensemble de données P tout en ne tenant pas compte des données aberrantes.

L'idée de base de l'algorithme RANSAC est de commencer avec S_i, un sous-ensemble de P choisi aléatoirement à l'itération i, en espérant que les données choisies soient fiables. La taille de S_i est n, c'est-à-dire une constante ou un minimum à respecter en fonction du modèle à créer. Ensuite, on crée un modèle M_i (une droite, un plan, une surface, etc.) avec les données de S_i. Parmi les données restantes ($P \backslash S_i$), on garde celles qui respectent le modèle M_i avec une certaine erreur T. L'union de ces dernières données avec celles de S_i donne l'ensemble consensus S_i*. Cet ensemble contient idéalement toutes les données fiables ou non aberrantes. À partir de là, on peut décider si S_i* est suffisamment grand pour être gardé. Si c'est le cas, on peut créer un modèle M_i* à partir des données de S_i*. Le modèle final est donc M_i*. Si S_i* est trop petit, alors on recommence avec un nouveau S_i (Fisher, 2002). Et on recommence ainsi pendant un certain nombre d'itérations k. Après avoir effectué les k itérations, s'il n'y a aucun modèle retenu, on peut retourner une erreur (Kovesi, 2000a) ou on peut utiliser le plus grand S_i* trouvé (Hartley & Zisserman, 2003, p. 118).

Il y a plusieurs variantes à cet algorithme de base. Selon Fischler et Bolles (1981), une fois que M_i* est trouvé, on peut compléter S_i* avec les données refusées, mais qui seraient maintenant compatibles avec M_i*. Ainsi, avec le S_i* augmenté, on peut calculer le M_i* final. On peut aussi chercher à maximiser la taille de S_i* d'une itération à l'autre (Kovesi, 2000a). Il y a aussi beaucoup d'autres variantes : c'est selon le goût de l'auteur de l'algorithme et selon le type de problème à résoudre.

Selon Hartley & Zisserman (2003, pp. 119-121), le nombre d'itérations k peut être déterminé pendant l'exécution des itérations. En effet, lors de la construction de S_i, la probabilité de choisir une bonne donnée à chaque pige est w. Afin d'avoir un S_i parfait, il faut que les n valeurs choisies soient bonnes. La probabilité d'obtenir un tel S_i est w^n.

Ainsi, la probabilité de ne pas avoir ce S_i idéal est $(1\text{-}w^n)$. La probabilité de n'avoir aucun S_i idéal au cours des k itérations est $(1\text{-}w^n)^k$. Enfin, la probabilité z d'avoir au moins un S_i idéal au cours des k itérations est décrit à l'équation 1.27 où z est généralement une probabilité fixée à 0.99. Ainsi, lors d'une itération, sachant la taille de P et de S_i*, on peut évaluer w ($|S_i^*|/|P|$) pour ensuite calculer k.

$$z = 1 - \left(1 - w^n\right)^k \tag{1.27}$$

Il reste maintenant à établir le modèle pour la vision stéréo.

1.4.2 Géométrie épipolaire

Si la distorsion radiale de chaque image est corrigée, alors la projection de chaque point $[X_s, Y_s, Z_s]^T$ de la scène sur chaque image devient une opération linéaire. Ainsi, pour une paire d'images stéréo, il existe une relation matricielle permettant de lier chaque coordonnée en pixels $[x, y, 1]^T$ dans une paire de points correspondants. Cette relation se traduit en deux théories : l'homographie et la matrice fondamentale.

La géométrie épipolaire est à la base de toute rectification et reconstruction d'une scène à partir d'une paire de caméras non calibrées (Trucco & Verri, 1998, chapitre 7). Dans cette géométrie, le centre optique de la caméra de droite projeté dans l'image de gauche est l'épipole dans l'image de gauche. En passant une droite de l'épipole via chaque point caractéristique, on obtient les droites épipolaires. Ces droites illustrent le fait que chaque point caractéristique de l'image de droite est bel et bien une projection causant une ambiguïté quant à la distance entre le point dans la scène et le centre optique de la caméra de droite. Ainsi, en ayant les droites épipolaires dans l'image de gauche et un autre ensemble de droites épipolaires dans l'image de droite, il devient donc possible d'évaluer grossièrement la position des points dans la scène, et ce, à l'aide de techniques de triangulation.

Cependant, le but de la présente section n'est pas de proposer des techniques de reconstruction, mais plutôt de proposer un modèle où les points caractéristiques sont liés

via des droites épipolaires vers un épipole. Si les points caractéristiques divergent de leur droite épipolaire, alors les paires de points concernés sont des données aberrantes.

1.4.2.1 Homographie

L'homographie consiste à transformer une coordonnée dans une image en la coordonnée correspondante dans l'autre image. Dans l'équation 1.28, $\vec{x}_d$ peut être de la forme $[kx_d, ky_d, k]^T$, mais puisqu'il s'agit d'une coordonnée homogène, c'est équivalent à $[x_d, y_d, 1]^T$. Bref, la matrice H est ni plus ni moins qu'une matrice de transformation homogène de taille trois par trois.

$$\vec{x}_d = H\vec{x}_g \tag{1.28}$$

Selon Hartley et Zisserman (2003, pp. 123-125) et selon le travail de Kovesi (2000a), il faut au moins quatre paires de points pour trouver la matrice H. Dans le cas où il y en a davantage, on essaie de trouver l'homographie générale de tout cet ensemble de paires de points. Enfin, s'il y a des paires aberrantes, Kovesi, Hartley et Zisserman suggèrent des algorithmes RANSAC pour résoudre ce problème. Le moyen pour déterminer si une paire respecte le modèle H_i (M_i dans la section 1.4.1) est d'appliquer l'équation 1.28 et d'évaluer l'erreur de l'inégalité s'il y a lieu.

1.4.2.2 Matrice fondamentale

La matrice fondamentale est la matrice F dans l'équation 1.29. Encore une fois, il s'agit d'une matrice trois par trois, mais, contrairement à la matrice d'homographie qui effectue la transformation d'une coordonnée de projection en une autre, la matrice fondamentale convertit $\vec{x}_g$ en les composantes de la droite épipolaire de $\vec{x}_d$. Enfin, le produit scalaire entre $\vec{x}_d$ et sa propre droite épipolaire est zéro étant donné que $\vec{x}_d$ est traversée par sa droite épipolaire, d'où le produit égal à zéro dans l'équation 1.29.

$$\vec{x}_d^T F\vec{x}_g = 0 \tag{1.29}$$

Il y a plusieurs façons de calculer la matrice fondamentale (Hartley & Zisserman, 2003; Trucco & Verri, 1998), mais il faut, en général, huit paires de points. Encore une fois,

s'il y a davantage de paires de points, il faut trouver la matrice fondamentale qui modélise la majorité des paires de points non aberrantes. Il y a plusieurs algorithmes RANSAC pour effectuer cette tâche (Hartley & Zisserman, 2003; Kovesi, 2000a). Pour notre part, nous utilisons l'implémentation de la bibliothèque OpenCV (Intel, 2006).

Ces algorithmes peuvent utiliser l'équation 1.29 pour vérifier si le produit est près de zéro. Il est aussi possible de calculer les droites épipolaires dans une image et de vérifier la distance entre chaque point caractéristique et sa droite épipolaire correspondante. Cette dernière méthode est utilisée par Kovesi (2000a).

1.5 Le projet de référence

Le présent projet de maîtrise a officiellement débuté par l'étude des travaux de Hajebi et Zelek (2006). Alors que ces derniers ont travaillé sur le problème de la vision stéréo pour des paires d'images constituées uniquement d'images infrarouges, nous avons opté pour essayer de résoudre le même problème, mais pour des paires d'images constituées d'une image du spectre visible et d'une image du spectre infrarouge.

Hajebi et Zelek ont déterminé que le meilleur algorithme pour trouver des points caractéristiques correspondants dans des images infrarouges est la congruence de phase (section 1.2.3.3). Avec cet algorithme, ils trouvent les points caractéristiques sur les arêtes et les coins dans chaque image. Le descripteur de chaque point trouvé est constitué des 24 coefficients Log-Gabor calculés pour chaque pixel (section 1.3.2.3). L'appariement se fait à l'aide d'une matrice de correspondance de type choix mutuel (section 1.3.2.1). La valeur de comparaison pour chaque paire de points possible est un produit de corrélation présenté à l'équation 1.25. Enfin, ils ont utilisé un filtre RANSAC (section 1.4.1) basé sur la matrice fondamentale (section 1.4.2.2) pour filtrer les paires de points aberrantes. Le résultat final de leur méthode est une carte de disparités indiquant une valeur de distance relative entre chaque point de la scène et le système de caméras.

CHAPITRE 2 MÉTHODOLOGIE

Dans ce chapitre, nous décrivons en détails les algorithmes utilisés dans ce projet de recherche. Nous proposons un aperçu de l'approche à la section 2.1. Nous continuons avec la section 2.2 présentant le prétraitement des séquences vidéo. Nous décrivons notre principale contribution aux sections 2.3 et 2.4 : il y a un nouveau type de points caractéristiques par section avec leur moyen respectif d'appariement des points. Nous présentons des filtres de paires de points aux sections 2.5 et 2.6. Nous terminons ce chapitre par le calcul des disparités à la section 2.7.

2.1 Aperçu de la méthode

La figure 2.1 présente un aperçu de la principale portion du système développé : la portion restante concerne l'architecture de test du projet.

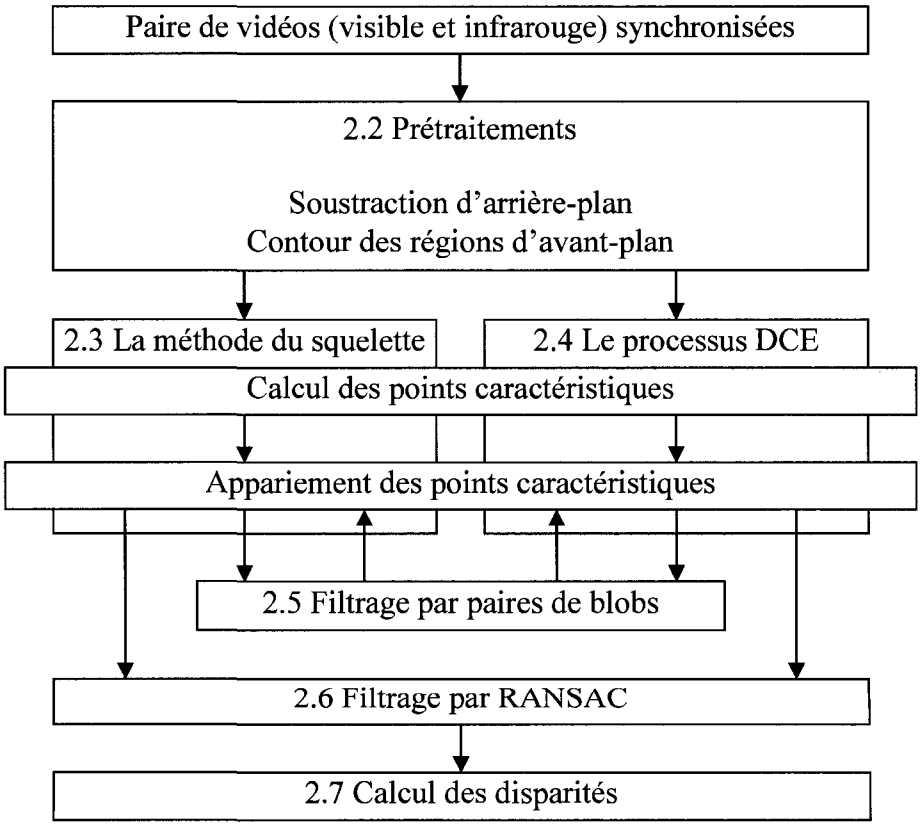

Figure 2.1 Plan du chapitre et aperçu global de l'algorithme

En entrée, le système reçoit une paire de vidéos synchronisées constituée d'une vidéo du spectre visible et d'une autre du spectre infrarouge. Typiquement, la caméra visible est positionnée à gauche de la caméra infrarouge, ce qui fait que les images de gauche dans les paires stéréo sont les images du spectre visible.

Dans la section 1.2, on a vu une série de types de points caractéristiques. Un problème avec les algorithmes classiques de recherche de points caractéristiques est qu'une image du spectre visible peut avoir des coins qui soient absents des images du spectre infrarouge et vice-versa. Un deuxième problème est que la plupart des descripteurs de points caractéristiques peuvent être dépendants du système de couleurs utilisé. Ainsi, certains points correspondants peuvent être décrits différemment en infrarouge et en visible par un même type de descripteur, ce qui rend impossible la création d'un algorithme d'appariement basé sur les descripteurs de points. Dans notre cas, le système doit donc prétraiter les deux vidéos afin d'y extraire l'information nécessaire assurant la recherche des points caractéristiques dans des images de types différents. Le prétraitement en question est la soustraction d'arrière-plan suivie du calcul du contour des régions (blobs).

Les contours permettent d'obtenir des points caractéristiques dont les descripteurs sont invariants par rapport aux couleurs de chaque vidéo : on a des points saillants sur les squelettes des blobs et sur le contour des blobs approximé par le processus DCE (Latecki & Lakamper, 1999). Les descripteurs obtenus permettent alors d'apparier les points d'une image à l'autre en utilisant une matrice de correspondance contenant le score d'appariement de chaque paire de points possible.

Les paires de points sont par la suite filtrées par deux filtres successifs : l'usage des paires de blobs et l'algorithme RANSAC (Fischler & Bolles, 1981). Ce dernier ajoute une contrainte supplémentaire permettant de respecter le modèle de la géométrie épipolaire utilisée dans la reconstruction de scènes en trois dimensions (Hajebi & Zelek, 2006; Hartley & Zisserman, 2003; Trucco & Verri, 1998). Dans notre cas, nous n'allons pas jusque là, mais nous approximons la distance relative des objets en mouvement par rapport au système de caméras.

2.2 Prétraitements

Puisque les principaux buts de la recherche sont le calcul des points caractéristiques et l'appariement de ces points, et ce, à partir d'images de différentes natures, il faut trouver une source d'informations invariantes selon le type de capteur utilisé. Cette source d'informations est en fait l'ensemble des blobs constituant l'avant-plan de chaque séquence vidéo. En effet, nous utilisons la forme et la position de ces blobs plutôt que leurs couleurs ou leurs textures. Par conséquent, nous considérons que la forme de deux blobs correspondants devrait être semblable d'une image à l'autre. Donc, non seulement les caméras doivent être suffisamment distancées entre elles pour qu'on obtienne des disparités permettant de distinguer les éléments proches des caméras de ceux qui sont éloignés, mais il faut aussi que les caméras ne soient pas trop distancées pour que la forme géométrique des éléments de la scène ne causent pas trop de changements dans le contour des blobs.

Les blobs sont identifiés à l'aide de diverses techniques de soustraction d'arrière-plan (section 2.2.1). Par la suite, ils sont décrits à l'aide de leur contour (section 2.2.2). Dès que ces informations sont disponibles pour chaque paire d'images, les différents algorithmes des sections 2.3 et 2.4 peuvent ensuite commencer leur recherche de points caractéristiques.

2.2.1 Soustraction d'arrière-plan

Typiquement, une soustraction d'arrière-plan consiste à identifier les pixels affichant les éléments en mouvement dans la scène ou les pixels des éléments ne faisant pas partie de l'arrière-plan, c'est-à-dire le décor. Par exemple, la figure 2.2 illustre une situation où nous connaissons l'arrière-plan et où nous devons enlever l'arrière-plan de l'image courante. Le résultat de la soustraction d'arrière-plan est illustré à la figure 2.3.

Dans la littérature, il y a plusieurs sortes d'algorithmes de soustraction d'arrière-plan : moyenne temporelle (Shoushtarian & Bez, 2005), simple Gaussienne (McKenna, Jabri, Duric, Rosenfeld, & Wechsler, 2000), amalgame de Gaussiennes (Stauffer & Grimson, 1999), rectangle (Bourezak & Bilodeau, 2006), etc. Ceux-ci ne sont pas détaillés dans le

présent mémoire, puisque le résultat n'est qu'une entrée au programme principal : on suppose que la soustraction d'arrière-plan utilisée est suffisamment fiable pour obtenir des blobs peu troués et peu bruités sur leur contour. Cependant, les algorithmes de soustraction d'arrière-plan sont loin d'être parfaits. Ainsi, pour chaque source vidéo, nous avons dû choisir le meilleur algorithme et la meilleure configuration pour chaque algorithme utilisé. Cette configuration quasi-optimale sera donnée dans la section des résultats (section 3.1.2). Par conséquent, l'impact de la soustraction d'arrière-plan sur les résultats finaux devrait normalement être à un niveau acceptable, c'est-à-dire minime. Et même si l'impact est non négligeable, les résultats montrent une certaine robustesse de la part du programme principal.

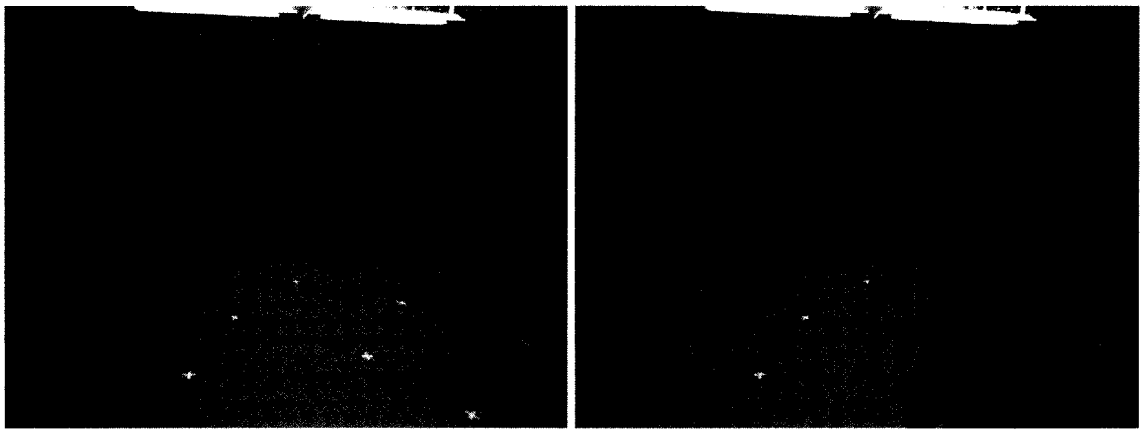

Figure 2.2 À gauche, on a l'arrière-plan d'une scène. À droite, on a l'image courante

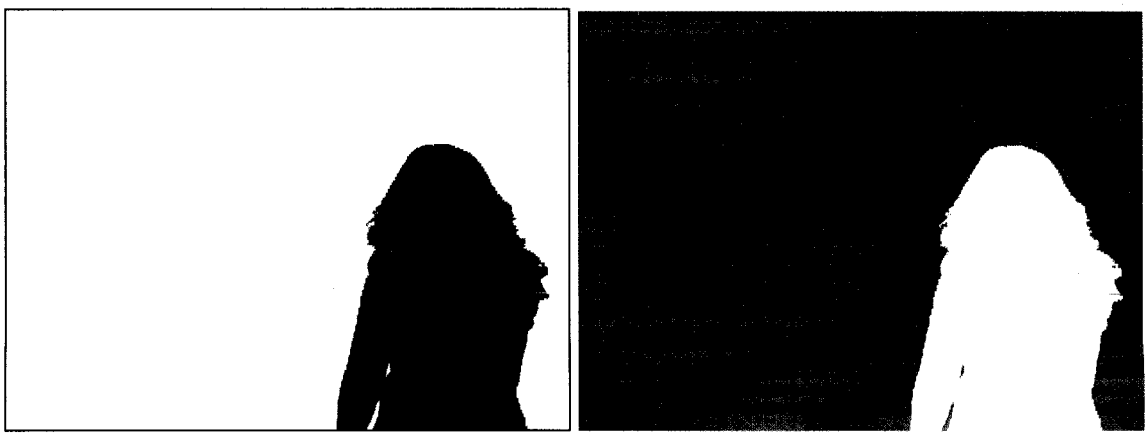

Figure 2.3 À gauche, on a le résultat de la soustraction d'arrière-plan avec l'algorithme de la moyenne temporelle. À droite, on a une représentation visuelle de ce que nos méthodes des sections 2.3 et 2.4 utilisent pour trouver des points caractéristiques

2.2.2 Contour des régions d'avant-plan

Alors que la méthode du squelette (section 2.3) a besoin des pixels de chaque blob, la méthode du processus DCE (section 2.4) a plutôt besoin du contour détaillé au pixel près de chaque blob. Les contours sont calculés à l'aide de la fonction *cvFindContours()*[1] de la bibliothèque OpenCV (Intel, 2006). Nous obtenons alors une suite de coordonnées cartésiennes faisant le tour de chaque blob et de chaque trou de blob.

2.3 La méthode du squelette

Un squelette est simplement une version ultra-mince et représentative d'un blob. Dans la figure 2.4, on a un exemple de squelette blanc dans un blob noir. Si le squelette était un réseau routier, alors tous ses culs-de-sac et ses intersections seraient des points intéressants à considérer pour représenter notre blob. En effet, ces intersections et ces culs-de-sac sont en fait des points d'articulation et des extrémités des objets en avant-plan dans la scène. Donc, par hypothèse, nous supposons que ces points d'articulation et ces extrémités sont présents dans les deux images d'une paire stéréo, et ce, peu importe la nature de chaque image. En somme, ces points ne sont aucunement dépendants des couleurs, des textures, des conditions d'éclairage, du type de capteur, etc. Cependant, l'hypothèse est vraie uniquement lorsque la forme des blobs est semblable d'une image à l'autre, c'est-à-dire lorsque les caméras ont un point de vue suffisamment semblable.

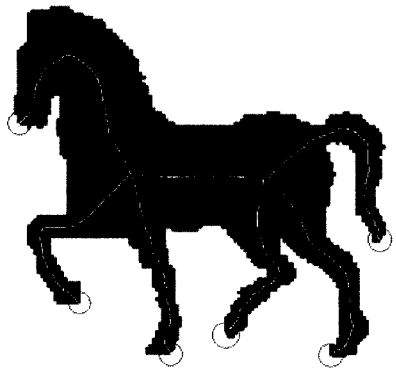

Figure 2.4 Exemple de squelette (adapté à partir de la figure 1c de Bai *et al.* (2007)). Les ellipses en rouge indiquent les points intéressants (caractéristiques) sur le squelette

[1] Voir une brève description de cette fonction à l'ANNEXE I.

Contrairement à la méthode utilisée par Bai *et al.* (2007) construisant un squelette quasi-parfait, notre méthode de calcul, présentée à la section 2.3.1, se concentre uniquement à la recherche des points caractéristiques, c'est-à-dire les intersections et les culs-de-sac dans un graphe proche du vrai squelette. Par la suite, les intersections en question peuvent être décrites en fonction du nombre d'arêtes et en fonction de l'orientation de ces arêtes. Ces informations deviendront des critères indispensables pour l'appariement des points (section 2.3.2).

2.3.1 Calcul des points caractéristiques

2.3.1.1 Préparation des blobs

Pour récapituler, nous commençons avec deux séquences vidéo synchronisées et qui sont analysées simultanément en prenant à la fois une image de chaque vidéo. À la suite d'une soustraction d'arrière-plan (section 2.2.1), il ne reste plus que les régions d'avant-plan (figure 2.5).

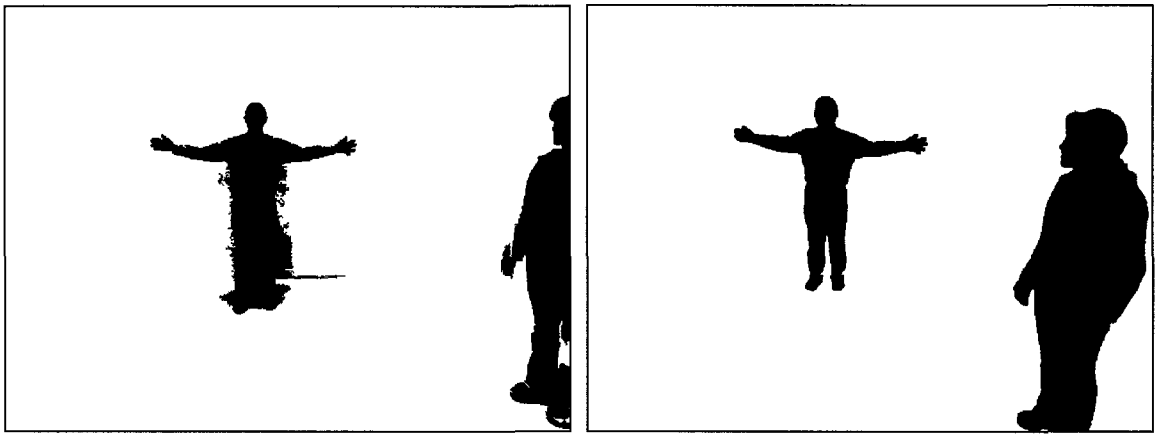

Figure 2.5 Exemple de régions d'avant-plan d'une paire stéréo. À gauche, l'avant-plan de l'image du spectre visible. À droite, l'avant-plan de l'image du spectre infrarouge

Étant donné que les squelettes sont sensibles aux trous (figure 2.6) et au bruit dans le contour des blobs, il nous faut supprimer les petits trous et adoucir ces contours qui sont déjà précis au pixel près. Pour ce faire, nous utilisons deux méthodes différentes.

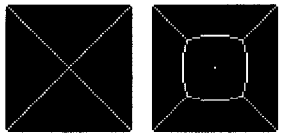

Figure 2.6 Exemple de squelette blanc sur deux blobs carrés noirs dont l'un a un trou blanc d'un seul pixel

La première méthode comprend l'approximation des contours de blob avec l'algorithme de Douglas-Peucker (Douglas & Peucker, 1973) qui est déjà implémenté dans la bibliothèque OpenCV[1]. Cet algorithme n'a qu'un seul paramètre : l'erreur d'approximation sous forme de distance maximale en pixels (2.5 pixels par défaut). Notre programme principal permet d'attribuer deux valeurs différentes de ce paramètre : une valeur pour les contours externes et une autre pour les contours internes. En effet, les blobs ont des contours internes pour les trous. Finalement, si la taille d'un trou est inférieure à un certain nombre de pixels (256 par défaut), le trou est supprimé du blob avant même d'effectuer l'approximation. Avec les contours approximatifs, nous redessinons et remplissons à neuf les blobs sans les trous trop petits.

La deuxième méthode de filtrage du contour et des trous consiste à effectuer une fermeture de chaque blob individuellement. La fermeture consiste à effectuer une dilatation suivie d'une érosion : les petits trous se referment et le bruit dans le contour est lissé. Les paramètres sont les dimensions d'une ellipse utilisée par OpenCV pour calculer la fermeture.

Dans notre implémentation finale, il est possible d'utiliser l'une ou l'autre des deux précédentes méthodes, et ce, en la choisissant sur la ligne de commande de l'exécutable.

2.3.1.2 Transformée de distances

Une fois que le blob a des contours lisses et que ses trous sont significatifs[2], il faut appliquer une transformée de distances sur le blob en question. La transformée de

[1] Voir une brève description de l'algorithme à l'ANNEXE II.

[2] Un trou significatif a au moins 256 pixels carrés par défaut. Lors de la soustraction d'arrière-plan, il peut arriver que certains pixels d'une région d'avant-plan soient classés par erreur dans les pixels de l'arrière-plan. Ceci se produit lorsque l'avant-plan a une teinte semblable à celle de l'arrière-plan. Donc, avec des trous d'au moins 256 pixels, nous corrigeons la majorité des erreurs de soustraction d'arrière-plan.

distances utilisée calcule des distances euclidiennes approximatives avec un filtre 3x3 (Borgefors, 1986). Après la transformée de distances, chaque pixel du blob possède une valeur correspondant à la distance entre ce pixel et le plus proche point du contour. Un exemple de cette transformée est présenté à la figure 2.7. Grâce à notre système de vision humain, nous arrivons facilement à voir le squelette des blobs. Ainsi, à partir des transformées de distances, il est non seulement possible d'y extraire le squelette, mais surtout, les points caractéristiques.

2.3.1.3 Trouver quelques points du squelette

Les transformées de distances sont des fonctions $D(i, j)$ où i et j indiquent la position d'un pixel et D donne la distance en pixels entre le pixel étudié et le plus proche pixel de l'arrière-plan. Un exemple de cette fonction est dessiné grossièrement à la figure 2.8. Ainsi, du point de vue du pixel vis-à-vis le sommet S, la plupart des pixels adjacents ont une valeur égale, légèrement supérieure (vers T) ou fortement inférieure (vers A et B).

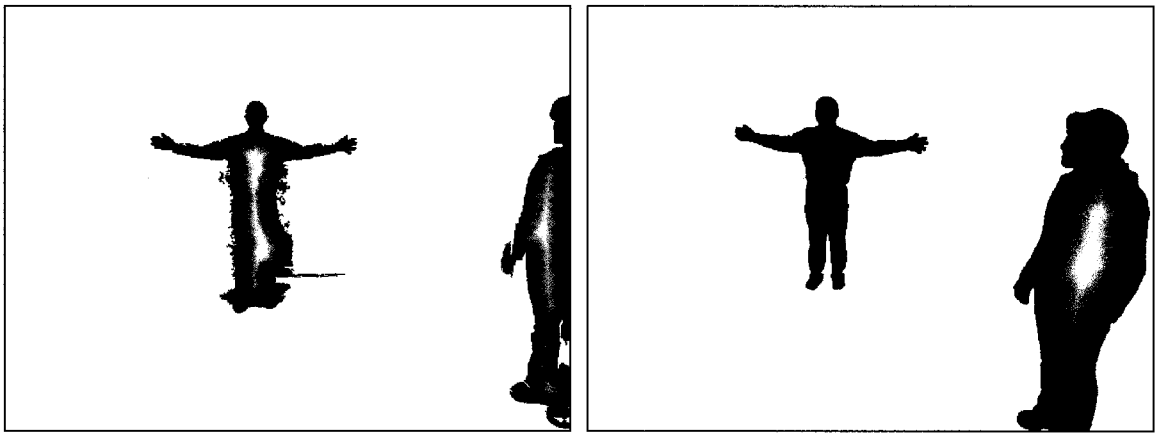

Figure 2.7 Exemple visuel de la transformée de distances

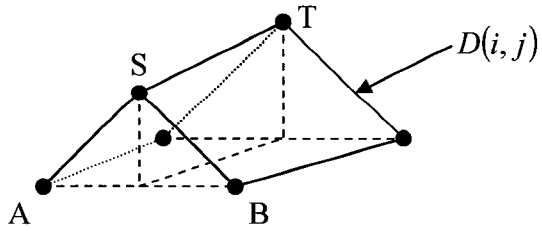

Figure 2.8 Coupe d'une vue tridimensionnelle d'une transformée de distances

Pour trouver les sommets en question, il faut rechercher les valeurs $D(i,j)$ qui se trouvent dans une région de courbure fortement négative (orientée vers le bas). J'ai donc utilisé un opérateur de Laplace (équation 2.1) sur les transformées de distances de chaque blob individuellement et un seuil de 4 pour identifier les sommets. On obtient alors un ensemble de points pour chaque blob (figure 2.9).

$$L_8 = \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix} \tag{2.1}$$

2.3.1.4 Identification des points caractéristiques

Pour chaque blob, on doit construire un arbre avec les points trouvés. L'idée est de partir avec un graphe complet où les sommets sont les points trouvés. Ensuite, on cherche l'arbre couvrant minimum à l'aide de l'algorithme de Prim (Cheriton & Tarjan, 1976). Par hypothèse, l'arbre couvrant minimum est le squelette que nous cherchons. Connaissant la structure de l'arbre, il est possible de ne retenir que les sommets qui ne sont pas attachés à exactement deux arêtes. Les sommets retenus sont les points caractéristiques. Ces points, affichés à la figure 2.10, confirment l'hypothèse de l'arbre couvrant minimum.

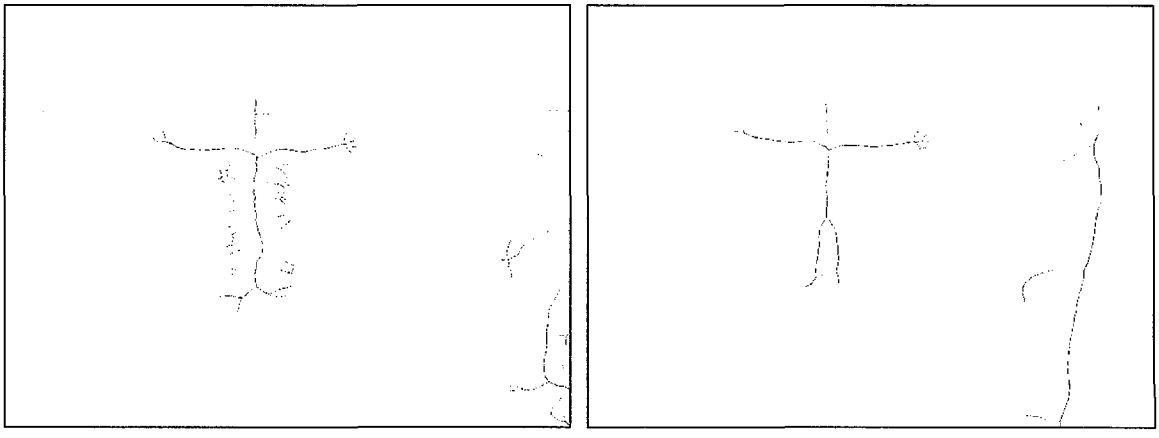

Figure 2.9 Maxima locaux trouvés dans les transformées de distances

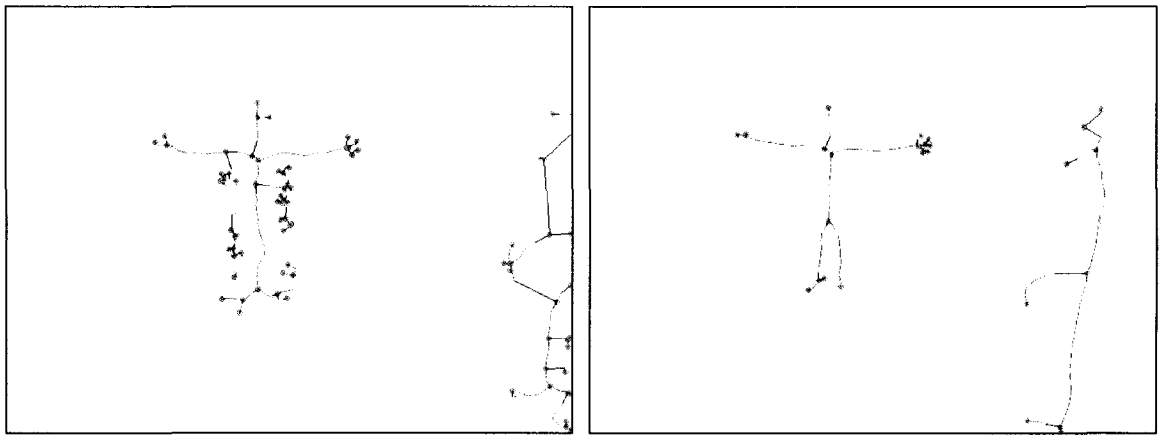

Figure 2.10 Les gros points gris pâles sont les points caractéristiques. Les lignes noires lient chaque point caractéristique à ses sommets voisins dans l'arbre couvrant minimum. Les autres points en gris foncés sont les mêmes qu'à la figure 2.9

2.3.2 Appariement des points

Une fois que les points caractéristiques sont trouvés dans chacune des deux images de la paire stéréo, il faut trouver des paires de points qui correspondent à des points de la scène. Pour ce faire, il faut arriver à décrire les points caractéristiques de telle sorte qu'il puisse être possible d'apparier des points semblables. Par exemple, dans la figure 2.10, les deux points caractéristiques au centre de la poitrine de l'acteur de gauche sont semblables d'une image à l'autre. Nous allons donc décrire quatre critères qui permettront de rendre ces deux paires de points facilement identifiables. Pour introduire ces quatre critères, nous allons commencer par décrire l'algorithme qui effectue l'appariement des points de l'image de gauche avec ceux de l'image de droite.

L'identification des paires de points correspondants se concrétise à l'aide d'une matrice de correspondance (section 1.3.2.1). Chaque point de l'image de gauche (g) est comparé à chaque point de l'image de droite (d). Le résultat de chaque comparaison, ce que nous appelons le *scoreTotal*, est enregistré dans la matrice de correspondance. Ce score total est décrit à l'équation 2.2 où l'on retrouve quatre autres scores correspondant aux quatre critères des points trouvés sur le squelette.

$$scoreTotal = scoreDist + scoreEucl + scoreAretes + scoreOrient \qquad (2.2)$$

Dès que la matrice de correspondance est remplie des scores de toutes les paires de points possibles, il faut retenir les paires de points qui se choisissent mutuellement. Par choisir, nous voulons dire qu'un point a forcément un score total maximal avec un point de l'autre image. Ainsi, pour chaque point, nous trouvons le point de l'autre image qui donne le meilleur score total. Et si deux points ont un score maximal uniquement lorsqu'ils sont jumelés, alors nous les gardons pour former une paire de points. Il s'agit donc de la méthode du choix mutuel présentée à la section 1.3.2.1 et utilisée par Hajebi et Zelek (2006). Malheureusement, malgré la robustesse de cet algorithme, certains points pertinents peuvent ne pas être appariés : lorsqu'un point (A) préfère un point (B) pour maximiser son *scoreTotal*, si le point (B) préfère plutôt un point (C), alors le point (A) est tout simplement rejeté même si son deuxième meilleur choix, un point (D), aurait pu donner un *scoreTotal* presque aussi élevé. Maintenant que le choix des paires de points est établi, il faut détailler les quatre scores constituant le score total.

À la section 2.3.1.2, nous avons mentionné que chaque blob subissait individuellement une transformée de distances. Les valeurs obtenues allaient de zéro à la plus grande distance dans le blob. Ces valeurs doivent maintenant être linéairement normalisées en des valeurs de zéro à un pour donner des distances relatives (*dr*).[1] Ainsi, un point qui se situe au bout des doigts d'un acteur obtient une distance relative près de zéro. À l'inverse, un point au centre de la poitrine de l'acteur obtient une distance relative près de un. Afin de ne pas apparier un point sur un doigt et un point sur la poitrine, il suffit de favoriser les paires de points tels que leur distance relative est équivalente. Dans les faits, l'équation 2.3 pénalise ($scoreDist \in [-1,0]$) les paires de points dont les points n'ont pas une distance relative équivalente. Le résultat de cette équation est le premier terme de l'équation 2.2.

$$scoreDist = -\left|dr_g - dr_d\right| \tag{2.3}$$

Même si les deux caméras ne sont pas calibrées, il faut quand même qu'elles soient positionnées en regardant le même point de fuite (caméras convergentes) et en ne

[1] Une fonction de OpenCV permet une telle normalisation.

tanguant ni à gauche ni à droite. De plus, les images utilisées ont la même taille. Par conséquent, les acteurs paraissent généralement à peu près à la même position dans les deux images. Afin d'éliminer certaines paires de points aberrantes, les paires de points sont formées de points relativement proches l'un de l'autre lorsqu'on superpose les deux images l'une sur l'autre. Le second critère concerne donc la position des points dans leur image respective. Le score de chaque paire de points est calculé selon la sigmoïde de l'équation 2.4. Dans cette dernière, δ est la distance euclidienne en pixels entre les deux points lorsque les images sont superposées et D est la longueur de la diagonale des deux images. Ainsi, plus δ est grand, plus le score total sera pénalisé par la sigmoïde.

$$scoreEucl = \frac{1}{1+e^{-3+6\delta/D}} \tag{2.4}$$

Dans le squelette, il y a deux types de points caractéristiques : les culs-de-sac qui ont une seule arête voisine et les intersections qui ont au moins trois arêtes voisines. Ainsi, le nombre d'arêtes d'un point est le troisième critère. Dans l'équation 2.5, on a nb_g pour le nombre d'arêtes du point de l'image gauche et nb_d pour le nombre d'arêtes du point de l'image droite. Chaque *nb* appartient à l'une des deux classes suivantes : un ou plus de trois. Ainsi, lorsque deux *nb* font partie de la même classe, les points correspondants ont ensemble un *scoreTotal* (équation 2.2) plus élevé. On peut remarquer que *scoreAretes* a un poids deux fois plus élevé que les autres scores, car nous voulons autant que possible éviter d'associer un point cul-de-sac avec un point d'intersection. Par exemple, nous ne voulons pas apparier un bout de doigt avec un point au centre de la poitrine d'un des acteurs Selon nos tout premiers résultats, ce facteur de deux a semblé suffisant.

$$scoreAretes = \begin{cases} 2 & \text{si } (nb_g \geq 3) = (nb_d \geq 3) \\ 0 & \text{sinon} \end{cases} \tag{2.5}$$

Enfin, dans l'exemple des deux points caractéristiques au centre de la poitrine de l'acteur de gauche dans la figure 2.10, ces deux points peuvent être différenciés par l'orientation de leurs arêtes. De même, l'extrémité d'un doigt de la main gauche ne sera pas appariée à l'extrémité d'un doigt de la main droite, car les doigts n'ont pas du tout la même

orientation. Bref, le quatrième critère est l'orientation des arêtes voisines aux points caractéristiques. Globalement, le quatrième score s'énonce par l'équation 2.6. Cette équation est complètement indépendante de la précédente (2.5), mais en tient compte indirectement grâce à la fonction max() qui retient la plus grande valeur entre nb_g et nb_d dans le but de pénaliser les paires de points où le nombre d'arêtes est différent.

$$scoreOrient = \frac{\sum_{p \in P_a} \cos\theta_p}{\max(nb_g, nb_d)} \tag{2.6}$$

Dans l'équation 2.6, on a l'ensemble P_a. Cet ensemble contient des paires d'arêtes correspondantes p. En effet, avant d'appliquer l'équation 2.6, il faut apparier les arêtes des points de la paire de points (figure 2.11). La raison est que les arêtes ne sont pas triées et, même en étant triées, il ne serait pas plus facile de les apparier. Ainsi, le moyen utilisé pour apparier les arêtes est une seconde matrice de correspondance. Puisque cette matrice compare toutes les arêtes du point de l'image gauche avec toutes les arêtes du point de l'image droite, le critère de comparaison est le cosinus de l'angle en chaque paire d'arêtes : plus l'angle est faible, plus le cosinus sera élevé, donc plus le score sera élevé. Dans cette matrice, nous cherchons les paires d'arêtes qui se choisissent mutuellement : pour chaque arête du point de gauche, nous trouvons son arête favorite du point de droite et vice-versa. Ainsi, si l'arête du point de gauche est la favorite de sa favorite, alors on a une paire où chaque arête est utilisée qu'une seule fois. Encore une fois, cette méthode n'optimise pas le nombre de paires retenues, mais il vaut mieux éviter les paires d'arêtes douteuses que de les garder. Bref, la somme des cosinus correspondant aux paires d'arêtes retenues est le numérateur de l'équation 2.6.

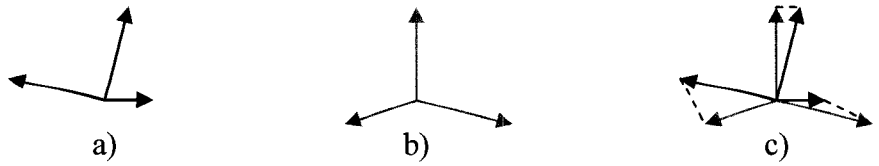

Figure 2.11 En a), on a les arêtes d'un point de l'image de gauche. En b), on a les arêtes d'un point de l'image de droite. En c), on a apparié les arêtes (lignes en pointillés) en fonction de l'angle entre les deux arêtes dans une paire d'arêtes

Finalement, ces quatre critères d'appariement sont les seuls critères que nous avons pu trouver afin de décrire les points caractéristiques indépendamment des couleurs et des textures dans le voisinage de chaque point caractéristique dans chaque image.

2.4 Le processus DCE

Longin Jan Latecki et Rolf Lakämper (1999) ont développé un algorithme d'approximation du contour d'une région. Cet algorithme se nomme *Discrete Curve Evolution* ou DCE. L'idée de base de cet algorithme est d'approximer un contour en supprimant toujours le ou les sommets les moins importants (selon l'équation 2.7) pour décrire la forme de la région. L'algorithme itère donc jusqu'à ce qu'il reste trois sommets ou moins dans le contour, d'où le surnom *processus DCE.*

Puisque le processus DCE garde toujours un maximum d'informations à propos de la forme générale de la région à toutes les itérations, un contour approximé contient que des sommets pertinents pour décrire la forme de la région. Ce sont ces sommets qui sont considérés comme le deuxième type de points caractéristiques de notre projet.

2.4.1 Calcul des points

Selon Latecki et Lakämper (1999), la théorie du processus DCE commence par définir un polygone P^i pour chaque itération i. À l'itération zéro, on a le contour original de la région. Ce contour initial, reçu en entrée dans notre algorithme, est généralement bruité (figure 2.12) et il y a un bon nombre de sommets restants qui peuvent être supprimés grâce au processus DCE. Théoriquement, le dernier P^i a au plus trois sommets. En effet, à chaque itération, il ne faut pas seulement enlever le sommet ayant le moins d'importance, mais enlever tous les sommets qui ont le même plus faible degré d'importance. Par exemple, dans un carré, les quatre sommets ont tous la même importance, donc ils sont tous supprimés à la dernière itération du processus DCE.

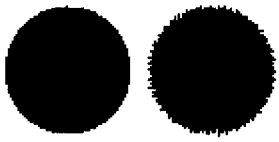

Figure 2.12 À gauche, un exemple d'une région dont le contour est peu bruité. À droite, un exemple d'une région dont le contour est très bruité (à droite)

Le fameux degré d'importance mentionné au paragraphe précédent est décrit à l'équation 2.7 (tirée de (Latecki & Lakamper, 1999)).

$$K(\beta, l_1, l_2) = \frac{\beta l_1 l_2}{l_1 + l_2} \tag{2.7}$$

Dans un contour P^i, chaque sommet a deux sommets voisins. La figure 2.13 illustre le cas d'un sommet quelconque appelé B. Dans l'équation 2.7, β est en fait l'angle extérieur du sommet B dans la figure 2.13. Il est effectivement logique que, si β est très petit, le sommet B n'apporte pas beaucoup d'information sur la forme de la région.

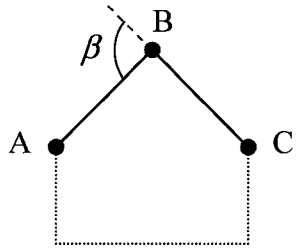

Figure 2.13 Zoom sur trois sommets consécutifs d'un contour P^i

Toujours dans l'équation 2.7, les valeurs l_1 et l_2 valent respectivement $\|\overline{AB}\|$ et $\|\overline{BC}\|$ normalisés en fonction du périmètre total de P^i. Bien que cette normalisation soit mentionnée dans l'article de référence, elle est coûteuse et inutile. En effet, l'équation 2.7 peut s'écrire autrement (équation 2.8) en échangeant respectivement l_1 et l_2 pour λ_1/p et λ_2/p où p est le périmètre de P^i et où λ_1 et λ_2 valent respectivement $\|\overline{AB}\|$ et $\|\overline{BC}\|$. Puisque ce périmètre p est une constante à l'itération i où l'on cherche la valeur minimale de K, il n'est donc pas nécessaire de réévaluer le périmètre du contour à chaque itération. Donc, nous pouvons laisser tomber p (équation 2.9). Enfin, β peut être en degrés ou en radians, ça ne dérange pas en autant que l'on garde la même unité à chaque fois.

$$K(\beta,\lambda_1,\lambda_2,p)=\frac{\beta\frac{\lambda_1}{p}\frac{\lambda_2}{p}}{\frac{\lambda_1}{p}+\frac{\lambda_2}{p}}=\frac{\beta\lambda_1\lambda_2}{p(\lambda_1+\lambda_2)} \tag{2.8}$$

$$K(\beta,\lambda_1,\lambda_2,p)\propto\frac{\beta\lambda_1\lambda_2}{\lambda_1+\lambda_2} \tag{2.9}$$

Dans notre implémentation de cet algorithme, nous ne supprimons pas tous les sommets dont le degré d'importance est le plus faible trouvé à une certaine itération. La raison est que nous ne voulons pas nous retrouver avec une situation comme l'exemple du carré où tous les sommets restants sont supprimés d'un seul coup. Ainsi, nous supprimons un seul sommet à chaque itération, ce qui permet de contrôler le nombre de sommets restants. D'ailleurs, ce nombre de sommets est un paramètre de notre implémentation. De plus, les contours extérieurs et intérieurs peuvent avoir un nombre de sommets différents. Typiquement, les contours extérieurs ont 32 sommets et les sommets intérieurs, 12. Évidemment, plus il y a de sommets, plus il peut y avoir du bruit dans le contour approximatif, mais moins il manque de détails.

Enfin, Latecki et Lakämper proposent une méthode pour empêcher que la suppression d'un sommet crée des croisements dans le contour approximatif. L'idée consiste à ne pas supprimer un sommet dont ses deux voisins et lui-même forment un triangle englobant un autre sommet de P^i. Ce phénomène se produit principalement avec des points près de régions concaves et courbées. Dans notre cas, les points à risque font généralement partie des 32 sommets les plus significatifs, donc ils sont rarement supprimés. Ceci est dû au fait que nous cherchons principalement à supprimer le bruit des contours. Donc, les chances d'obtenir un croisement sont faibles. Telle est donc la justification pour ne pas alourdir davantage notre implémentation du processus DCE qui se contente de supprimer un à un les sommets les moins importants.

2.4.2 Appariement des points

Tout comme dans la section 2.3.2, tous les points caractéristiques trouvés dans une image sont comparés à tous ceux de l'autre image. L'appariement des points utilise encore une matrice de correspondance pour trouver les paires de points. Les valeurs de la matrice sont des scores comme celui de l'équation 2.2. Dans le cas de l'algorithme DCE, il n'y a que trois scores pour déterminer le score total de chaque paire possible de points (équation 2.10) : les deux points doivent avoir une importance similaire dans leur contour respectif, les deux points sont proches l'un de l'autre lorsque les deux images sont superposées et l'orientation des deux arêtes voisines de chaque point est semblable.

$$scoreTotal = scoreKrel + scoreEucl + scoreOrient \tag{2.10}$$

Le premier score, *scoreKrel*, est un score comparant les valeurs d'importance K de chaque point (équation 2.9). Dans le cas de l'algorithme du squelette, la transformée de distances de chaque blob est normalisée de zéro à un, ce qui donne toujours des scores dans l'intervalle [-1,0] dans le cas de l'équation 2.3. Dans le cas présent, les valeurs de K ne sont ni normalisées de zéro à un, ni en fonction du périmètre de leur contour (p dans l'équation 2.8). Étant donné qu'un point de la scène peut se retrouver sur un contour externe dans une image et sur un contour interne dans l'autre image et étant donné que les contours internes sont généralement plus petits, nous ne devons pas normaliser les valeurs de K de chaque point en fonction du périmètre du contour. Autrement, deux points réellement correspondants ne seraient pas comparables. Donc, il faut tout de même pouvoir calculer la différence entre deux K tout en normalisant le résultat final dans l'intervalle [-1,0]. Ainsi, dans l'équation 2.11, K_g et K_d sont les valeurs d'importance des deux sommets dans la paire étudiée et le dénominateur sert à normaliser *scoreKrel*. Comme à l'équation 2.3, il s'agit d'un score pénalisant pour les sommets qui n'ont pas une valeur semblable.

$$scoreKrel = \frac{-\left|K_g - K_d\right|}{\begin{cases} \sqrt{K_g^2 + K_d^2} & \text{si } \sqrt{K_g^2 + K_d^2} > 0 \\ 1 & \text{sinon} \end{cases}} \tag{2.11}$$

La métrique *scoreEucl* est la même qu'à l'équation 2.4. De même, la métrique *scoreOrient* est pratiquement la même qu'à l'équation 2.6, excepté que nous avons toujours deux arêtes par sommet dans le cas présent.

2.5 Filtrage par paires de blobs

Maintenant que nous avons plusieurs paires de points trouvées grâce aux précédents algorithmes et sachant que chaque point connaît le blob auquel il appartient, il est possible d'établir des paires de blobs ou de régions d'avant-plan. Pour ce faire, il suffit de comparer tous les blobs de l'image de gauche avec tous les blobs de l'image de droite, et ce, en notant le résultat de la comparaison dans une matrice de correspondance. Cette fois-ci, le score de chaque case de la matrice est le nombre de paires de points liant les deux blobs concernés. De plus, nous utilisons une méthode que nous avons appelée le maximum restant.

Le maximum restant consiste à choisir la paire de blobs ayant le plus grand score dans toute la matrice. Du point de vue du blob de l'image de gauche, cela équivaut à choisir son blob de droite favori, et vice-versa. Une fois que la paire est choisie, on enlève la ligne du blob de gauche et la colonne du blob de droite dans la matrice. On cherche alors la prochaine paire de blobs en cherchant la valeur maximale restante dans toute la matrice. On itère ainsi jusqu'à ce que la matrice soit vidée.

Le maximum restant peut donner des résultats semblables à ceux du choix mutuel (section 1.3.2.1), mais le maximum restant est une méthode beaucoup moins sévère. En effet, dans l'exemple des points (A), (B) et (C), si (A) choisissait normalement (B) alors que ce dernier choisit (C), le maximum restant pourrait alors trouver un compromis en appariant (A) avec un autre point (D) et en gardant la paire (B, C). Ce raisonnement s'applique aussi pour les paires de blobs. En effet, lors d'un conflit de choix mutuel, il peut arriver que le deuxième meilleur choix d'un blob soit finalement le bon choix. De plus, étant donné le faible nombre de blobs dans une image, les chances de se tromper sur le second meilleur choix sont relativement faibles. Évidemment, il peut y avoir la

création de mauvaises paires de blobs, mais il s'agit là d'un risque que nous étions prêts à accepter.

Maintenant que les paires de blobs sont connues, il faut recalculer les paires de points en ajoutant une contrainte supplémentaire : les deux points étudiés dans une paire doivent correspondre à une paire de blobs valide. Plus directement, pour chaque paire de blobs, nous recalculons les score totaux des sections 2.3.2 et 2.4.2, et ce, pour tous les points du blob de l'image de gauche et tous les points du blob de l'image de droite. L'espace de recherche est donc grandement réduit et, à cause du choix mutuel des paires de points, nous avons plus de chances de trouver davantage de paires de points, et ce, en filtrant plusieurs des paires aberrantes. Bref, à cause de l'usage de plus petites matrices de correspondance pour chaque paire de blobs, il est plus simple de recommencer les calculs que de récupérer les anciens scores totaux.

Pour chaque paire de blobs, il est maintenant possible de superposer les deux blobs et de les aligner par rapport à leur centroïde. Lorsque vient le temps de recalculer le *scoreTotal* pour l'appariement des points selon chaque algorithme des sections 2.3 et 2.4, il est possible d'y ajouter un autre score : *scoreDistBlobBlob*. Ce score évalue le déplacement horizontal et vertical des points dans la paire étudiée lorsque les blobs sont alignés sur leur centroïde : il y a un bonus de 1.0 si le déplacement horizontal est égal ou inférieur au quart de la largeur maximale des boîtes englobantes des deux blobs et il y a un autre bonus de 1.0 si le déplacement vertical est égal ou inférieur au quart de la hauteur maximale des deux mêmes boîtes englobantes. Étant donné que les blobs ne sont jamais vraiment de taille égale et que le centroïde est rarement exactement au même endroit dans les deux blobs correspondants, nous nous devons de donner une certaine marge de manœuvre pour le seuil de la distance entre deux points correspondants lorsque les blobs sont alignés. Nous avons trouvé empiriquement que le quart des dimensions maximales des deux boîtes englobantes était suffisamment restrictif pour éliminer les paires très aberrantes et suffisamment permissif pour permettre certaines imprécisions. Enfin, puisque ce filtre se fie sur la paire de blobs, toute information en découlant mérite

d'avoir un poids relativement élevé dans le calcul des *scoreTotal*, d'où les poids de 1.0 pour les mesures horizontales et verticales.

Bref, non seulement *scoreEucl* permet d'éviter les paires de points éloignés, mais *scoreDistBlobBlob* va plus loin en interdisant les paires aberrantes à l'intérieur d'un même objet, et ce, en tenant compte de la position du centroïde de chaque blob dans leur image respective. Ainsi, par exemple, les paires de points liant la tête dans une image au pied dans l'autre image sont éliminées.

À toutes fins pratiques, nous aurions pu évaluer directement la disparité des paires de blobs en fonction de leur position dans leur image respective, mais nous avons décidé de nous concentrer sur l'amélioration de la qualité des paires de points en vue d'une reconstruction plus élaborée que le simple calcul des disparités présenté à la section 2.7.

2.6 Filtrage par RANSAC

Le filtrage par RANSAC (section 1.4.1) sert à respecter les contraintes de la géométrie épipolaire en établissant la matrice fondamentale (section 1.4.2.2) liant la majorité des points de l'image de gauche aux points de l'image de droite. Par majorité, nous voulons dire que l'algorithme RANSAC cherche une matrice fondamentale qui satisfait un nombre maximum de paire de points, ce qui élimine certaines paires de points aberrantes.

En pratique, nous utilisons la fonction *cvFindFundamentalMat()* de la bibliothèque OpenCV (Intel, 2006). Cette fonction est spécialisée dans le calcul de la matrice fondamentale. Un de ses paramètres est le type d'algorithme utilisé : dans notre cas, nous utilisons l'algorithme RANSAC. Après l'exécution de la fonction, celle-ci nous retourne non seulement la matrice fondamentale, mais aussi la liste de toutes les paires de points filtrées.

2.7 Calcul des disparités

Dans la scène, les objets en mouvement sont vus sous forme de blobs dans chaque image. Afin de connaître grossièrement la distance relative entre ces objets et la paire de caméras, il faut calculer la disparité de chaque blob d'une image à l'autre. Nous

définissons la disparité par le déplacement du blob d'une image à l'autre et par rapport à un référentiel. Le référentiel en question est le point de fuite dans chacune des images, c'est-à-dire un point éloigné de la scène projeté dans chaque image. Or, ce point de fuite a volontairement été placé au centre dans le champ de vision de chaque caméra (section 1.1). Donc, lorsqu'un blob se déplace peu d'une image à l'autre, c'est que l'objet ou l'acteur correspondant est éloigné des caméras. À l'inverse, lorsqu'un blob se déplace beaucoup d'une image à l'autre, l'objet ou l'acteur correspondant est près de la paire de caméras. Enfin, puisque le type de scène étudiée concerne des humains semblant bouger horizontalement dans les images, seules les disparités horizontales comptent.

Pour chaque paire de blobs, nous recensons toutes les paires de points. À l'aide de ces paires de points, nous trouvons le déplacement horizontal médian, car il s'agit d'une donnée statistique stable face aux quelques paires de points aberrantes qui peuvent passer au travers des deux filtres décrits aux sections 2.5 et 2.6. Ce déplacement horizontal médian est ensuite converti en déplacement horizontal absolu (non signé et non orienté). C'est ce déplacement absolu qui détermine la proximité de l'objet par rapport aux caméras.

2.8 Complexité des méthodes

Pour l'étude de la complexité, nous considérons uniquement le coût du traitement d'une paire d'images de taille m par n. Nous considérons que le nombre de points sur un contour et le nombre de points caractéristiques par blob sont de l'ordre de m.

- Soustraction d'arrière-plan : $\Omega(mn)$ puisque la complexité dépend de l'algorithme utilisé;
- Approximation du contour (squelette) : $O(m^2)$;
- Fermeture (squelette) : $O(mnp)$ où p est le coût de la dilatation et de l'érosion;
- Transformée de distances (squelette) : $O(mn)$;
- Opérateur de Laplace et seuillage (squelette) : $O(bmn)$ où b ($b << m$) est le nombre de blobs;

- Appariement des points (squelette et DCE) : $O(b^2m^2k^2)$ où k est la dimension de la matrice de correspondance pour les paires d'arêtes ($k << m$). Dans le cas du processus DCE, k vaut toujours deux et m vaut 32 au maximum par défaut;
- Processus DCE : $O(bm^2)$ où m est le nombre initial de sommets sur le contour de chaque blob et de chaque trou de blob;
- Calcul des paires de blobs (squelette et DCE) : $O(m + b^2)$;
- Application du filtre des paires de blobs (squelette et DCE) : $O(bm^2k^2)$;
- Application du filtre RANSAC : $\Omega(m^2)$ puisque la complexité dépend de l'implémentation dans OpenCV;
- Calcul des disparités médianes : $O(bm*\log(m))$.

Le calcul a une complexité variable d'une étape à l'autre et d'un algorithme à l'autre. En fixant la résolution des images (mn), la complexité globale est environ quadratique.

CHAPITRE 3 RÉSULTATS ET DISCUSSION

Ce chapitre est divisé en deux grandes sections : la méthodologie de l'expérimentation (section 3.1) et les résultats comprenant une discussion (section 3.2).

3.1 Expérimentation

L'expérimentation commence par la réalisation et l'enregistrement des séquences vidéo (section 3.1.1). De ces séquences vidéo, il nous faut créer des valeurs témoins (section 3.1.2) pour les comparer aux résultats de notre méthode décrite au chapitre précédent. Pour des fins de comparaison entre nos algorithmes et ceux de la littérature, deux autres algorithmes de points caractéristiques s'ajoutent aux tests (section 3.1.3). Le processus du test, qui consiste à évaluer les méthodes présentées au chapitre précédent, est décrit à la section 3.1.4. Nous présentons les différents paramètres du programme dans la section 3.1.5. Finalement, de ces paramètres, nous déterminons une liste de cas de test (section 3.1.6).

3.1.1 Réalisation des séquences vidéos

Les vidéos sont réalisées dans un environnement dont l'éclairage et l'arrière-plan sont contrôlés. En effet, l'éclairage est constant. L'arrière-plan est constitué de rideaux bleus, d'un plancher gris et d'un plafond blanc. La scène mesure environ cinq mètres par quatre mètres (voir le plan à la figure 3.1).

Les deux caméras, une Sony DFW-SX910 et une Flir Thermovision A40, sont installées sur des trépieds presque au niveau et à environ la même hauteur, soit 1.5 mètres. La caméra Sony est la caméra de gauche et la Flir, la caméra de droite (figure 3.1). La distance qui les sépare est environ 1.5 mètres (Grande) ou 75 centimètres (Petite). Nous considérons ces deux cas afin d'évaluer l'impact de ce changement sur les disparités des blobs. Enfin, la caméra Sony est équipée d'une lentille Pentax de longueur focale de 4.2 millimètres. La caméra Flir est équipée d'une lentille de longueur focale de 9.0 millimètres placée devant la lentille de base de 36.0 millimètres. Ainsi, les deux caméras

ont environ le même champ de vision qui est suffisamment large pour voir à la fois le plafond et le plancher, et ce, dès le milieu de la scène (à 2m des caméras).

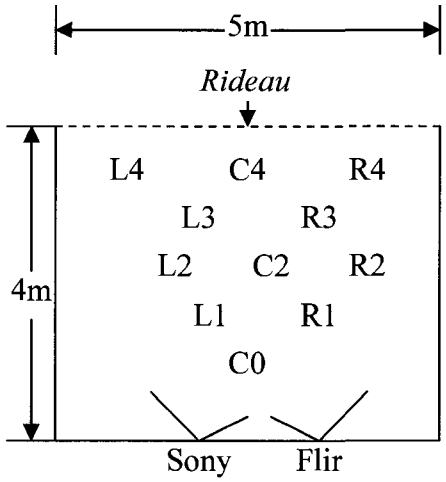

Figure 3.1 Plan de la scène avec toutes les positions possibles des acteurs

Les deux caméras sont synchronisées à 7.5 images par seconde. Nous corrigeons la distorsion radiale de chaque vidéo individuellement à l'aide d'un filtre maison. Essentiellement, ce filtre projette chaque image sur une sphère virtuelle et projette ensuite l'image de la sphère sur le plan d'origine, mais en étirant certaines portions de l'image d'origine (figure 3.2). Ce filtre n'a qu'un seul paramètre : un coefficient permettant de faire varier le rayon de la sphère virtuelle.

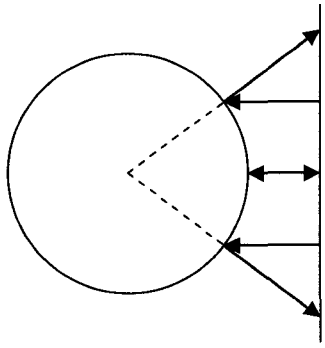

Figure 3.2 Principe de correction de la distorsion radiale : l'image d'origine est projetée sur une sphère virtuelle et l'image sur la sphère est projetée à nouveau sur le plan initial

Finalement, nous transformons chaque image pour avoir une résolution de 640 par 480 pixels. Selon la section 1.1, les pixels au centre de chaque image devraient pointer vers les mêmes points de la scène. Donc, en ayant la même résolution et le même champ de

vision apparent (les blobs ont une taille similaire) pour les deux images, il est alors possible d'utiliser le même référentiel pour les deux images.

Il y a deux scénarios : à deux acteurs et à trois acteurs. Nous avons conçu les scénarios pour que les acteurs soient à différentes distances l'un par rapport à l'autre et aussi par rapport aux caméras. Ces scénarios sont présentés dans les tableaux 3.1 et 3.2. Pour chaque acte, on y retrouve la nouvelle position de l'acteur qui doit se déplacer. Les différentes positions sont illustrées dans la figure 3.1.

Tableau 3.1 Scénario à deux acteurs

Acteur	Actes														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	C0		R1		R2		R4		R3		C2		R1		R3
B		C4		L4		L3		L1		L2		L1		C0	

Tableau 3.2 Scénario à trois acteurs

Acteur	Actes														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	C0			L1			L2			L3			L4		
B		C4			C4			C2			R3			C0	
C			R3			R2			R2			R1			R2

Bref, en tout, nous avons deux scénarios et deux distances différentes entre les caméras. Il y a donc quatre vidéos à enregistrer. Malheureusement, à cause de certaines limitations techniques, il n'est pas possible de réaliser chaque scénario en utilisant trois caméras simultanément : visible, visible et infrarouge, toutes côte à côte avec un espace de 75 centimètres entre les deux premières et les deux dernières. Ainsi, il aurait été possible d'avoir exactement la même scène pour évaluer l'impact du changement de distance entre les deux caméras.

3.1.2 Création des valeurs témoins

Puisque les valeurs témoins ne viennent pas de bases de données publiques et que nous avons nos propres séquences vidéos, il nous faut donc créer nos propres valeurs témoins. Notre programme principal reçoit en entrée des régions d'avant-plan et doit trouver la

disparité de ces blobs. Donc, nos valeurs témoins doivent nous permettre d'évaluer la précision de la disparité des paires de blobs.

Le programme permettant de créer interactivement les valeurs témoins reçoit en argument les paramètres de soustraction d'arrière-plan pour chaque vidéo dans la paire de vidéos. L'algorithme choisi pour la soustraction d'arrière-plan de chaque type de vidéo est la *moyenne temporelle* (Shoushtarian & Bez, 2005) : on somme chaque pixel de l'arrière-plan individuellement, on calcule la couleur moyenne de chaque pixel et on tolère une différence fixe lors du test d'arrière-plan. Cet algorithme est rapide et donne des résultats appréciables lorsqu'on le combine à un léger filtre de bruit gaussien, un détecteur d'ombres et un filtre supprimant les blobs de moins de 256 pixels carrés.

Pour chaque paire d'images, l'interface présente visuellement chaque blob dans chaque image. L'usager n'a alors qu'à cliquer sur un point en particulier dans un des blobs d'une image et sur le point correspondant dans le blob correspondant dans l'autre image pour créer une paire de points. Cette paire de points est ensuite enregistrée.

Pour chaque paire de blobs, il est possible d'enregistrer plusieurs paires de points : la disparité moyenne sera alors utilisée pour les tests. Par ailleurs, un même blob peut faire partie de deux paires différentes de blobs étant donné qu'il y a, tôt ou tard, des fusions et des divisions inattendues de blobs.

Le programme interactif permet d'éditer les paires de points à toutes les n paires d'images. Dans le cadre du présent projet, nous avons utilisé une image sur huit pour créer les valeurs témoins. Pour les autres images, le programme effectue un suivi automatique entre chaque paire d'images étudiée, et ce, pour propager la disparité moyenne de chaque paire de blobs. Puisque le suivi n'est pas très fiable, nos résultats tiennent compte uniquement des paires de points entrées par l'usager.

Enfin, le fichier binaire final contient les paramètres des deux soustractions d'arrière-plan, le nombre d'images au total, le saut n entre chaque paire d'images, les paires de blobs pour toutes les paires d'images et les paires de points pour chaque paire de blobs.

Dans ce fichier, les blobs sont enregistrés en donnant à chacun la position et les dimensions de la boîte rectangulaire qui les englobe individuellement.

À la fin, il suffit de fournir ce fichier de valeurs témoins à notre programme principal. Ce dernier peut alors récupérer les paramètres de chaque soustraction d'arrière-plan. Puisque la soustraction d'arrière-plan est identique dans chaque programme, le programme principal arrivera à retrouver les paires de blobs en fonction des coordonnées des boîtes qui les englobent. Enfin, pour chaque paire de blobs, le programme principal calcule leur disparité moyenne.

3.1.3 Points caractéristiques de comparaison

Dans les sections 2.3 et 2.4, nous avons présenté nos deux algorithmes de recherche de points caractéristiques. Afin de comparer nos méthodes avec celles de la littérature, nous avons retenu trois algorithmes disponibles dans deux différentes implémentations : la congruence de phase (section 3.1.3.1), les valeurs propres minimales (section 3.1.3.2) et l'opérateur de Harris et Stephens (section 3.1.3.2).

3.1.3.1 Le filtre de congruence de phase

Le filtre de congruence de phase est l'algorithme présenté à la section 1.2.3.3 (page 19). Cet algorithme a été utilisé dans (Hajebi & Zelek, 2006) pour trouver les points caractéristiques dans les deux images infrarouges de la paire stéréo. Il ne reste plus qu'à étudier comment cet algorithme se débrouille avec des images de type différent.

Calcul des points

Pour calculer les points caractéristiques, nous réutilisons le code Matlab de Kovesi (2000a) : *phasecong2.m* et *nonmaxsuppts.m*. Dans le cas de *phasecong2.m*, nous utilisons les paramètres par défaut étant donné que ce sont les mêmes qui sont utilisés par Hajebi et Zelek (2006). Dans le cas de *nonmaxsuppts.m*, notre programme principal permet d'utiliser ses deux paramètres, c'est-à-dire le rayon d'une dilatation et le seuil.

Contrairement à Hajebi et Zelek qui ont fait leur projet dans Matlab, nous avons implémenté notre code en C++ afin d'utiliser la libraire OpenCV et d'autres

bibliothèques C++ maison. Malgré tout, nous avons pu utiliser *phasecong2.m* tel quel, mais cela nous a empêché d'avoir accès facilement aux coefficients Log-Gabor (voir la section 1.3.2.3) pour décrire chaque point caractéristique. De plus, contrairement à Hajebi et Zelek qui ont utilisé les valeurs de M (équation 1.17) ayant une forte réponse sur les arêtes, nous avons préféré utiliser les valeurs de m (équation 1.18) puisque nous cherchons surtout les coins dans chaque image. Bref, nous sommes uniquement capables d'identifier quelques points caractéristiques dans chaque image et nous avons dû adapter la méthode d'origine pour trouver d'autres informations afin de comparer les points caractéristiques.

Puisqu'il ne faut pas se fier aux couleurs pour apparier des points provenant d'images de différents types, il faut trouver un type d'information pour au moins permettre la comparaison des points dans une paire potentielle. Il reste tout de même l'orientation des gradients. En effet, même si les couleurs et les teintes changent d'une image à l'autre, la forme d'un coin ou du voisinage d'un point caractéristique devrait être peu modifiée. L'idée est d'avoir une approximation de l'orientation des gradients autour de chaque point caractéristique. Cette approximation est un vecteur décrit à l'équation 3.1 et illustré à la figure 3.3. Dans cette équation, on voit que, pour tout point caractéristique (x, y), nous calculons l'orientation sur une fenêtre de cinq par cinq pixels. L'exception à cette règle est lorsque le point à caractériser est à moins de deux pixels du bord de l'image. Dans ce cas, u et v vont de -2 à 0 ou de 0 à 2 selon le cas. Finalement, cette double sommation doit être normalisée pour devenir un vecteur unitaire $[a, b]^T$.

$$Orient(x,y) = \sum_{u=-2}^{2}\sum_{v=-2}^{2}\left(I(x+u,y+v) - I(x,y)\right)\begin{bmatrix} u \\ v \end{bmatrix} \tag{3.1}$$

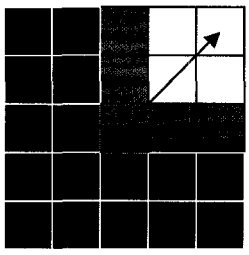

Figure 3.3 Application de l'équation 3.1 calculant l'orientation d'un coin. Le pixel central entouré de vert est le point caractéristique. La flèche en rouge indique l'orientation du coin

Les points caractéristiques ont finalement une brève description : le blob auquel ils appartiennent, leur position et leur orientation approximative. Puisque la congruence de phase peut trouver des points caractéristiques dans l'arrière-plan, ces points sont discriminés des points de l'avant-plan : ils sont identifiés différemment des points de l'avant-plan, ce qui permettra d'utiliser uniquement ces derniers pour le calcul des paires de blobs et des disparités.

Appariement des points

Tout comme dans les sections 2.3 et 2.4, nous utilisons une matrice de correspondance contenant les scores de chaque paire de points possibles entre les points d'une image et ceux de l'autre image. Le score total pour les paires de points trouvés par congruence de phase est défini par l'équation 3.2. Évidemment, cette équation n'est pas du tout comparable à l'équation 1.25 (page 31), mais cela fait partie de notre adaptation de la méthode de Hajebi et Zelek (2006).

$$scoreTotal = \tfrac{1}{2}scoreAngle + scoreBlob + scoreEucl \tag{3.2}$$

Dans le *scoreTotal* de l'équation 3.2, le *scoreAngle* est calculé en fonction de l'orientation de chaque point caractéristique (voir l'équation 3.1). Chaque orientation est un vecteur unitaire $[a, b]^T$. Deux points caractéristiques semblables ont environ la même orientation, à un signe près. En effet, étant donné que les images ne proviennent pas du même type de capteur, le niveau de gris autour d'un point caractéristique peut varier

différemment dans chaque image. Ainsi, pour caractériser la différence d'orientation des points caractéristiques, nous avons utilisé le sinus de l'angle entre les deux vecteurs unitaires gauche (*g*) et droite (*d*) (équation 3.3). Pourquoi le sinus et non le cosinus? Parce que la valeur du sinus évolue plus rapidement pour des faibles différences d'angle : $|\partial \sin(x)/\partial x|_{x=0} > |\partial \cos(x)/\partial x|_{x=0}$. Donc, la fonction sinus est beaucoup plus sévère envers les orientations légèrement ou moyennement différentes. Ensuite, pour tenir compte du signe de chaque vecteur, nous gardons la valeur absolue du sinus. Enfin, la métrique à l'équation 3.3 est en fait une pénalité au *scoreTotal* étant donné que les valeurs élevées du sinus indiquent un angle tendant vers $\pi/2$.

$$scoreAngle = -\left|a_g b_d - a_d b_g\right| \tag{3.3}$$

Finalement, *scoreAngle* est pertinent pour discriminer de très mauvaises paires de points, mais, empiriquement, nous avons trouvé que les deux autres scores étaient plus importants pour éviter d'avoir trop de paires aberrantes. Ainsi, *scoreAngle* a une pondération de un demi dans le *scoreTotal.*

Parmi les deux autres scores, on retrouve *scoreBlob* qui est ni plus ni moins qu'un bonus pour les points faisant partie du même type de région. En effet, selon la fin de la sous-section précédente, les points caractéristiques appartenant à l'arrière-plan sont identifiés différemment des points des blobs d'avant-plan. Ainsi, *scoreBlob* est défini à l'équation 3.4 où pt_g est le point de gauche, pt_d est le point de droite et *AP* est l'ensemble des pixels de l'avant-plan.

$$scoreBlob = \begin{cases} 1 & \text{si } (pt_g \in AP) = (pt_d \in AP) \\ 0 & \text{sinon} \end{cases} \tag{3.4}$$

Enfin, *scoreEucl* est le même qu'à l'équation 2.4.

3.1.3.2 La méthode des valeurs propres minimales et l'opérateur de Harris

La bibliothèque OpenCV (Intel, 2006) contient déjà quelques fonctions pour rechercher des points caractéristiques dans une image en teintes de gris. Étant donné la notoriété de

cette bibliothèque, nous avons décidé de la confronter à nos deux principaux algorithmes présentés aux sections 2.3 et 2.4.

Calcul des points

La méthode des valeurs propres minimales (section 1.2.1.1) et l'opérateurs de Harris et Stephens (section 1.2.1) sont déjà implémentés dans OpenCV et sont accessibles via la fonction *cvGoodFeaturesToTrack()*. Notre programme principal utilise cette fonction et permet d'ajuster trois de ses paramètres. Les deux premiers paramètres sont le niveau de qualité et la distance minimale en pixels entre chaque point caractéristique retenu. Le troisième paramètre consiste à sélectionner l'algorithme des valeurs propres minimales ou l'opérateur de Harris et Stephens.

Appariement des points

La définition des points caractéristiques est identique à celle présentée à la section précédente, c'est-à-dire qu'on sait si le point fait partie de l'avant-plan ou de l'arrière-plan, on connaît la position du point dans son image et on a aussi l'orientation des gradients autour du point caractéristique (équation 3.1). De même, le calcul du *scoreTotal* est identique à celui de l'équation 3.2.

3.1.4 Processus de test

Notre programme principal est en fait un outil de mesure pour tester nos algorithmes. Donc, à plusieurs endroits dans le calcul des disparités, notre programme mesure l'état du calcul. Le programme mesure, entre autres, le nombre de paires de points avant et après les filtres, le nombre de bonnes paires de blobs, la précision des disparités, etc. Toutes les mesures sont présentées dans la figure 3.4. Dans cette figure, on peut noter l'usage de certains paramètres. Ceux-ci seront décrits en détail à la section suivante.

Selon la figure 3.4, il est possible d'appliquer le filtre RANSAC une ou plusieurs fois. Il faut savoir que ce filtre est déjà implémenté dans la bibliothèque OpenCV (voir la section 1.4.2.2) et possède deux paramètres : le niveau de confiance (c) et la distance maximale (d). L'idée est que ces paramètres sont très permissifs par défaut afin de tolérer des

paires de points imparfaites mais pertinentes : il peut arriver que les points dans une paire valide ne soient pas parfaitement positionnés au pixel près. Donc, d'un filtrage RANSAC à l'autre, nous augmentons successivement les critères de sélection en effectuant la racine carrée du niveau de confiance, qui tendra vers 1.0 ou 100%, et de la distance maximale en pixels, qui tendra aussi vers 1.0.

Dans le calcul de la précision des disparités, nous évaluons uniquement la disparité des paires de blobs trouvées : on ne pénalise pas pour les paires de blobs erronées ou manquantes. Le calcul de la précision moyenne est à l'équation 3.5 dans laquelle nous utilisons l'ensemble B des paires de blobs valides b. Donc, pour chaque paire de blobs b, on a une disparité horizontale absolue δ_b et une disparité de référence δ_{rb}, c'est-à-dire la valeur témoin. Le score final est donc près de 1.0 lorsque les disparités trouvées et celles de référence se ressemblent.

$$scoreFinal = 1 - \frac{\sum_{b \in B} \left(\frac{|\delta_b - \delta_{rb}|}{\sqrt{\delta_b^2 + \delta_{rb}^2}} \right)}{|B|} \tag{3.5}$$

Enfin, tous ces résultats et mesures peuvent être enregistrés dans un fichier texte dans un format facilement récupérable pour la compilation des données dans Excel.

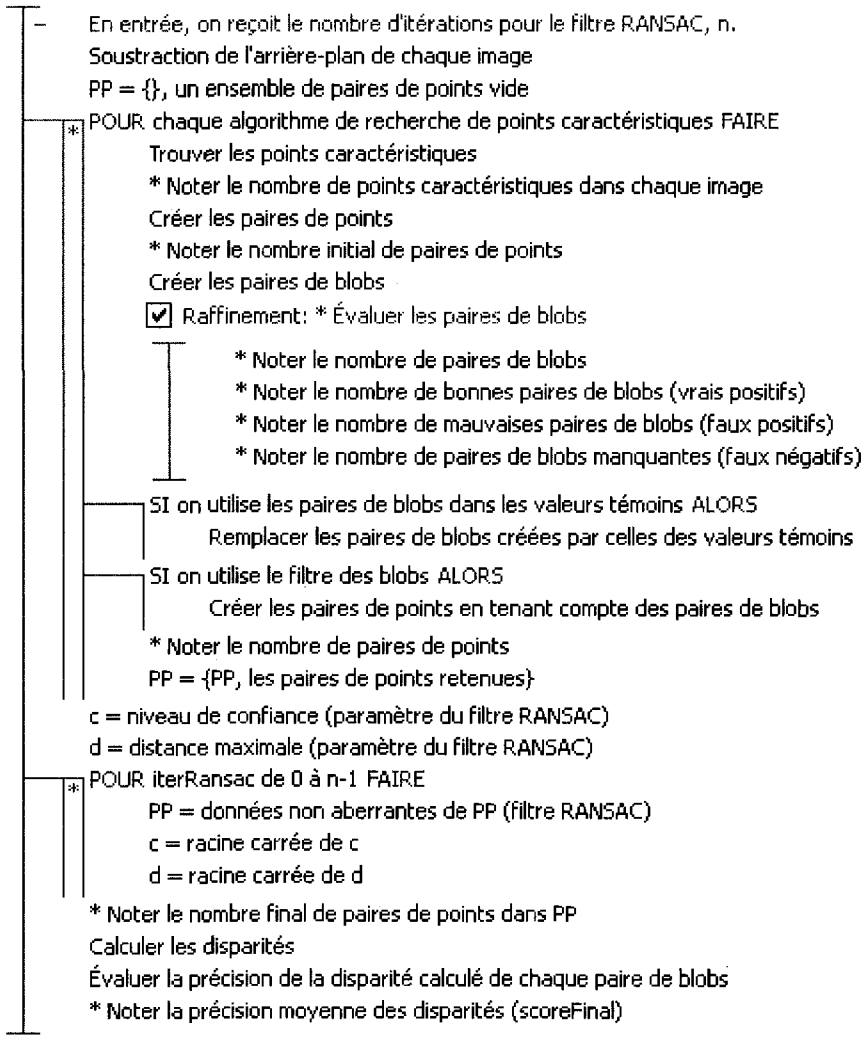

Figure 3.4 Pseudo-code schématique du calcul et des tests effectués. Les mesures sont identifiées par une étoile (*) au début de la ligne de description

Il faut noter que *scoreFinal* est une mesure de précision des disparités qui est plus stricte que la mesure de l'ordre des blobs du plus près au plus éloigné des caméras. Cependant, notre hypothèse est que des disparités de la bonne longueur permettraient ultimement de déduire une distance relative fiable entre un blob et les caméras.

3.1.5 Paramètres de test

L'exécutable final est sous le nom de *analyseurStereo.exe*. Cet exécutable reçoit une liste d'arguments *Args_Test* et une liste de chercheurs de points caractéristiques *sff* ayant chacun leur propre liste d'arguments. En fait, la syntaxe de l'appel à l'exécutable est :

« analyseurStereo <Args_Test> [sff –a algorithme ...]* ». Dans cette syntaxe, les paramètres de test sont dans *<Args_Test>* (tableau 3.3) et on peut utiliser de zéro à quatre chercheurs de points caractéristiques : la méthode du squelette (tableau 3.4), le processus DCE (tableau 3.5), la congruence de phase (tableau 3.6) et les valeurs propres minimales ou l'opérateur de Harris et Stephens (tableau 3.7).

Tableau 3.3 Paramètres de test

Syntaxe	Description
-gt nomFichier	Argument obligatoire. Nom du fichier contenant les valeurs témoins. Dans ce fichier, on a aussi les paramètres pour la soustraction d'arrière-plan de chaque séquence vidéo.
-o nomFichier	Nom du fichier texte pour enregistrer les résultats. Tous les paramètres de test feront partie du contenu du fichier.
-ugt	Si utilisé, on utilise les valeurs témoins pour le filtrage des blobs.
-blob	Si utilisé, on utilise le filtre des blobs (section 2.5).
-ransac 0..2	Nombre de filtres RANSAC (section 2.6) à appliquer successivement (0 par défaut).
-c 0.33…0.99	Niveau de confiance (0.9 par défaut) : paramètre de la fonction OpenCV *cvFindFundamentalMat(...,double param2, ...)* (z à l'équation 1.27)
-d 0.5..5	Distance maximale en pixels (2.0 par défaut) : paramètre de la fonction OpenCV *cvFindFundamentalMat(...,double param1, ...)* (T à la page 32)

Tableau 3.4 Paramètres de la méthode du squelette (-a skel)

Syntaxe	Description
-f wx wy	Appliquer une fermeture (dilatation et érosion) sur le blob. Les deux valeurs wx et wy sont les dimensions de l'ellipse utilisée pour la fermeture. Si ce paramètre est utilisé, il désactive les trois autres paramètres ci-dessous.
-t nbPixels	Taille minimale des trous en pixels carrés (256 par défaut) pour garder le trou pour le calcul du squelette.
-de distance	Pour l'approximation des contours externes, il faut fournir la précision voulue sous forme de distance maximale en pixels (2.5 par défaut).
-di distance	Pour l'approximation des contours internes (les trous), il faut fournir la précision voulue sous forme de distance maximale en pixels (2.5 par défaut).

Tableau 3.5 Paramètres du processus DCE (-a dce)

Syntaxe	Description
-t nbPixels	Taille minimale des trous en pixels carrés (256 par défaut) pour garder le trou pour le calcul du squelette.
-e 1..256	Nombre maximum de points caractéristiques sur le contour externe (32 points par défaut).
-i 1..256	Nombre maximum de points caractéristiques sur le contour interne pour les trous (12 points par défaut).

Tableau 3.6 Paramètres de la congruence de phase (-a pc2)

Syntaxe	Description
-wd path	Paramètre obligatoire. Puisque l'algorithme est codé dans Matlab, le moteur de Matlab doit connaître le répertoire contenant le code source.
-r 1..3 1..3	Pour chaque image dans la paire, on définit le rayon (2 par défaut) utilisé dans l'algorithme de seuillage de type suppression des non-maxima (*nonmaxsuppts.m*).
-t 0..1 0..1	Seuil pour chaque image dans la paire (0.25 par défaut).

Tableau 3.7 Paramètres de la méthode des valeurs propres (-a eigen)

Syntaxe	Description
-ql 0..1	Niveau de qualité (0.3 ou 30% par défaut). C'est un seuil pour comparer tous les points au meilleur point caractéristique.
-md 1..256	Distance minimale (2 pixels par défaut) entre deux points caractéristiques trouvés.
-h	Si utilisé, c'est le critère de Harris (section 1.2.1) qui est utilisé au lieu du calcul des valeurs propre minimales.

3.1.6 Cas de test

Il y a quatre paires de vidéos à tester. Pour chaque paire de vidéos, nous utilisons un algorithme de recherche de points caractéristiques. Cela donne au total 20 cas différents (4 paires de vidéos * 5 algorithmes en incluant l'opérateur de Harris et Stephens). Pour chaque cas, nous utilisons le filtre des blobs éliminant les paires de points correspondant à des paires de blobs invalides. Par défaut, nous n'utilisons pas les paires de blobs des valeurs témoins (section 2.5). Enfin, nous utilisons une seule itération du filtre RANSAC

(section 2.6) avec ses paramètres par défaut, et ce, afin de filtrer les dernières paires de points aberrantes en modélisant une matrice fondamentale (section 1.4.2.2). Bref, nous avons là les cas de référence qui permettent déjà d'évaluer l'impact de la position des caméras.

Ensuite, pour chaque paire de vidéos, nous utilisons simultanément les algorithmes du squelette et du processus DCE avec leurs paramètres par défaut. Nous évaluons l'impact de l'utilisation de deux algorithmes au lieu d'un seul. Même si notre implémentation permet d'utiliser n'importe quelle combinaison d'algorithmes (figure 3.4), nous nous en tenons qu'à une seule paire afin d'évaluer le potentiel d'une méthode hybride.

Pour chaque paire de vidéos, en utilisant la méthode du squelette et le processus DCE individuellement avec leurs paramètres par défaut, nous varions les cas de test (tableau 3.8). On a, tout d'abord, le cas sans filtre. On a ensuite le cas avec le filtre des blobs sans ou avec les paires de blobs des valeurs témoins, mais sans filtre RANSAC. On a ensuite le cas avec un RANSAC et les valeurs témoins pour le filtre des blobs. On a ensuite le filtre RANSAC à deux itérations et avec le filtre des blobs, mais sans les valeurs témoins. Nous terminons avec le filtre des blobs avec une et deux itérations de RANSAC, mais avec des paramètres variés pour l'algorithme RANSAC. Bref, nous évaluons l'impact de l'usage des filtres et des paramètres du filtre RANSAC.

Tableau 3.8 Différents cas de test pour les filtres

Cas #	-blob	-ugt	-ransac	-c	-d
1					
2	x				
3	x	x			
4	x	x	1		
5	x		2		
6	x		1	0.8	
7	x		1	0.95	
8	x		2	0.8	
9	x		2	0.95	
10	x		1		1
11	x		1		3
12	x		2		1
13	x		2		3

Finalement, pour chaque paire de vidéos et en utilisant les filtres par défaut, nous testons chaque algorithme, sauf celui de la congruence de phase, en faisant varier un paramètre ou deux à la fois de l'algorithme testé : les paramètres de même nature sont modifiés proportionnellement. Ceci a pour but de trouver la meilleure configuration possible pour chaque algorithme. Dans le cas de la congruence de phase, nous utilisons déjà les paramètres suggérés par Kovesi (2000a) et l'exécution de cet algorithme est beaucoup trop longue pour être étudiée en profondeur dans le cadre de la présente recherche.

3.1.7 Métriques de tests

Chacune des quatre paires de vidéos a une durée d'environ 800 images. Puisque nous étudions une image sur huit dans chaque paire de vidéos, alors cela donne environ 100 lignes de résultats bruts par cas de test. Il y a au total plus de cinq douzaines de cas de test. Dans chaque ligne de résultats, on retrouve les données mesurées lors du processus de test (figure 3.4). Bref, l'ensemble des résultats bruts doivent être compilés pour pouvoir les présenter dans le présent mémoire.

Nous avons retenus plusieurs métriques pour comparer les différents cas : le nombre moyen de bonnes paires de blobs, le nombre moyen de paires de blobs manquantes, le score final moyen et l'écart-type des scores finaux (équation 3.5). Le calcul des différentes moyennes permet de compiler la centaine de valeurs brutes par cas de test. L'écart-type des scores finaux permet de déterminer si l'algorithme a souvent des scores élevés ou si la distribution de ses scores est étendue. Dans ce dernier cas, cela voudrait dire que l'algorithme est instable.

Les différents groupes de cas de test présentés à la section précédente sont classés dans des sous-sections.

3.2 Les résultats

Pour l'instant, notre implémentation permet de traiter environ une à deux paires d'images à la seconde, sauf dans le cas de la congruence de phase qui prend environ 20 secondes par paire d'images. Ainsi, ce dernier algorithme a déjà un lourd désavantage sur toutes les autres méthodes, car son usage rendrait la tâche d'optimisation plus difficile pour développer des implantations en temps réel.

3.2.1 Cas de test de référence

Dans la figure 3.5, les algorithmes du squelette, du processus DCE et de la congruence de phase génèrent clairement plus de bonnes paires de blobs que les deux autres algorithmes. On constate aussi que les algorithmes DCE, du squelette et de la congruence de phase négligent environ le même nombre de paires de blobs. Donc, les deux autres algorithmes (valeurs propres minimales et opérateur de Harris et Stephens) sont réellement moins performants pour détecter les paires de blobs.

Par ailleurs, en observant le comportement de chaque algorithme pour chaque vidéo, on remarque une hausse de bonnes paires de blobs et de paires de blobs manquantes lorsqu'il y a trois acteurs. Évidemment, plus le nombre d'acteurs est élevé, plus le nombre de bonnes paires de blobs doit être élevé. De même, à cause des multiples occlusions dans les vidéos, il est normal que notre implémentation ne soit pas capable de détecter autant de paires de blobs que ce que nous avons enregistré dans les valeurs témoin. Dans le cas de la distance entre chaque caméra (grande ou petite), les résultats ne sont pas clairs, mais le nombre d'acteurs semble avoir moins d'impact lorsque les deux caméras sont plus rapprochées (petite). Ainsi, la position des caméras peut causer plus ou moins d'occlusions, ce qui semble influencer les résultats finaux.

Finalement, l'algorithme DCE a une légère avance sur l'algorithme du squelette et celui de la congruence de phase. Il surclasse aussi les deux autres algorithmes.

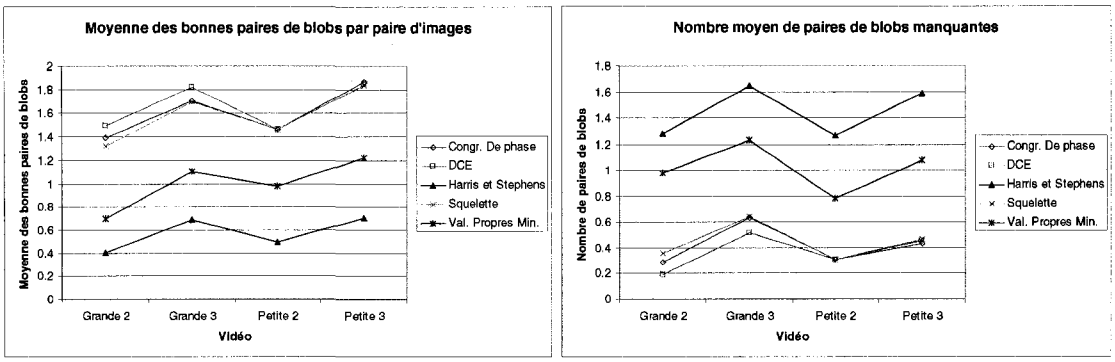

Figure 3.5 Nombre moyen de bonnes paires de blobs et de paires de blobs manquantes

Dans la figure 3.6, nous avons le résultat sur le nombre moyen de paires de points retenus à la suite du filtre RANSAC. Encore une fois, l'algorithme de Harris et Stephens et celui des valeurs propres minimales sont les moins performants pour trouver des paires de points, d'où leur incapacité à trouver un nombre suffisant de paires de blobs. Néanmoins, la bibliothèque OpenCV semble favoriser l'algorithme des valeurs propres minimales et la figure 3.6 confirme que cet algorithme donne de meilleurs résultats que ceux de l'opérateur de Harris et Stephens.

Toujours dans la même figure, on remarque que l'usage de la congruence permet de trouver davantage de paires de points que nos deux méthodes. Ceci est dû au fait que la congruence de phase n'a pas un nombre de points limité comme c'est le cas avec la méthode du processus DCE (maximum 32 points par blob pour le contour externe et 12 points par trou). De plus, la méthode du squelette trouve généralement moins de paires de points, car les points caractéristiques peuvent se faire plus rares que les points de contour comme dans le cas du processus DCE. Néanmoins, nos deux méthodes trouvent davantage de paires de points lorsqu'il y a un acteur de plus dans le scénario. Finalement, le nombre de paires de points n'est pas important si ces paires de points sont aberrantes, d'où le besoin d'évaluer leur précision.

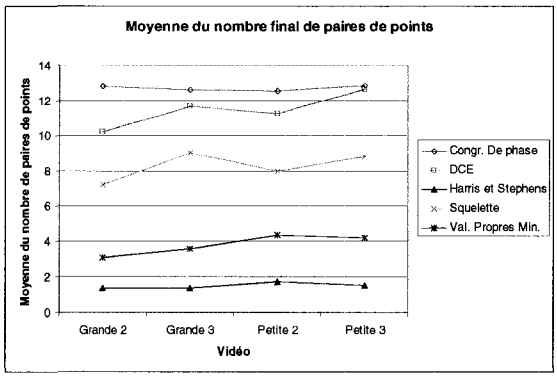

Figure 3.6 Moyenne du nombre final de paires de points pour chaque algorithme

Dans la figure 3.7, on peut comparer la précision des disparités des paires de blobs valides. Il est à noter qu'un algorithme peut avoir un score final parfait s'il n'a aucune paire de blobs valide. Ainsi, les résultats des algorithmes de Harris et Stephens et des valeurs propres minimales sont biaisés étant donné qu'il leur manque souvent des paires de blobs (figure 3.5). D'ailleurs, ils ont un écart-type relativement élevé parce qu'ils ont un mélange de scores relativement faibles et de scores parfaits.

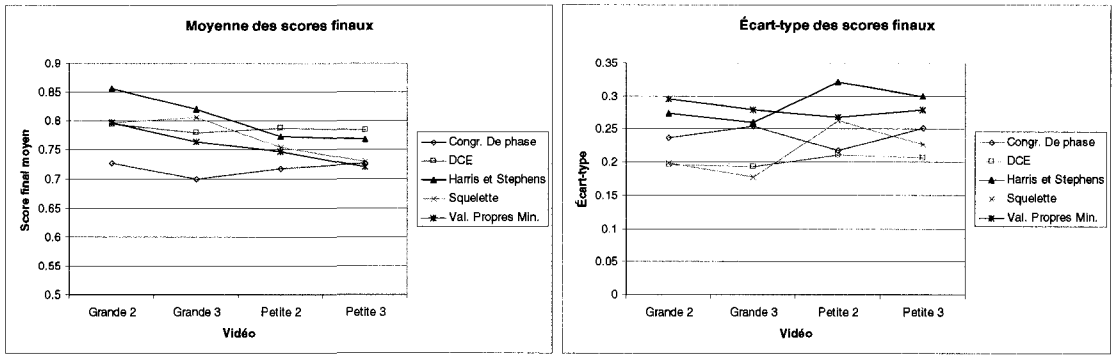

Figure 3.7 Moyenne et écart-type des scores finaux

Dans le cas de la congruence de phase, les scores finaux sont faibles en moyenne. Donc, les disparités calculées ne sont pas très fiables. Et c'est sans compter la lenteur de cet algorithme par rapport aux autres.

Lorsque les caméras sont grandement distancées, l'algorithme du squelette est à son meilleur : scores moyens élevés et faible écart-type. Ceci serait dû au filtre RANSAC qui aurait de la difficulté à calculer une matrice fondamentale stable avec des paires de points dont la disparité est plus petite pour les cas Petite 2 et 3. Par contre, dans l'ensemble des

cas, l'algorithme DCE obtient des moyennes élevées et stables de scores finaux d'une vidéo à l'autre, ainsi que des écart-types relativement faibles et stables d'une vidéo à l'autre. Bref, encore une fois, l'algorithme DCE est légèrement supérieur à l'algorithme du squelette. Bien sûr, il surclasse les trois autres algorithmes.

Dans les sections qui suivent, tous les prochains résultats seront comparés aux cas de références présentés dans la présente section. Les cas de référence utilisent les paramètres par défaut présentés à la section 3.1.5.

3.2.2 Utilisation de deux algorithmes à la fois

Dans la figure 3.8, on a les résultats des scores finaux, c'est-à-dire leur moyenne et leur écart-type, pour l'usage d'un ou de deux algorithmes, et ce, en utilisant que les paramètres par défaut. L'étude des paires de blobs révèlerait les mêmes résultats qu'à la figure 3.5, car le test des paires de blobs est appliqué pour chaque algorithme individuellement (figure 3.4).

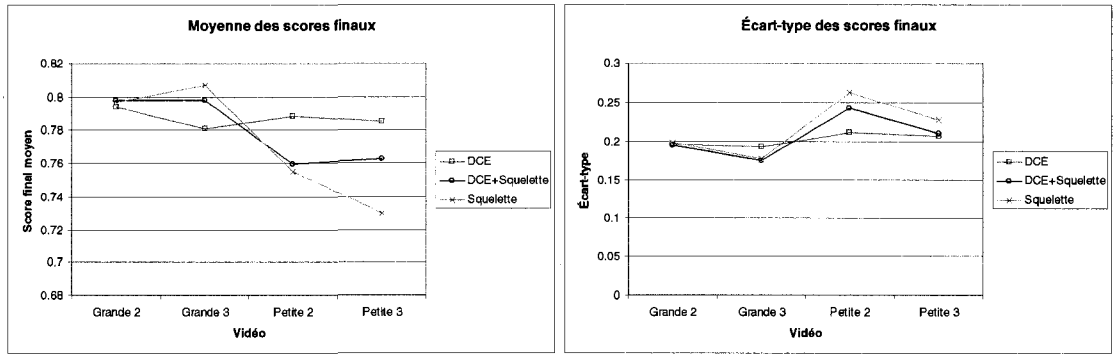

Figure 3.8 Comparaison des algorithmes DCE, hybride (DCE+squelette) et du squelette

Globalement, nous concluons que le mélange de deux algorithmes n'améliore pas nécessairement les résultats de chaque algorithme individuel. En effet, les résultats de la méthode hybride semblent se situer entre les résultats de chaque algorithme individuel, comme si on avait fait la moyenne des deux courbes de référence. Selon le pseudo-code à la figure 3.4, seul le filtre RANSAC peut avoir un impact sur les résultats de la méthode hybride. Donc, ce dernier filtre n'arriverait pas à retenir les meilleures paires de points trouvées par les deux méthodes individuellement. Pire encore, il se pourrait que

l'algorithme RANSAC supprime de bonnes paires de points. Voilà pourquoi la méthode hybride semble donner des résultats moyens. La seule exception serait l'écart-type de la méthode hybride pour les séquences vidéo où les caméras sont les plus distancées : l'écart-type est légèrement plus faible, donc la méthode hybride aurait des scores plus stables que les deux autres algorithmes.

À la figure 3.9, on a le nombre moyen de paires de points à la suite du filtre de RANSAC, et ce, pour chaque vidéo et pour chaque algorithme. Normalement, les paires de points filtrées pour chaque algorithme individuel devraient s'additionner dans la méthode hybride, mais il faut croire qu'il restait encore des paires aberrantes qui se sont fait éliminer par le filtre RANSAC après la méthode hybride. Donc, il y a quand même lieu de croire que cette méthode hybride a du potentiel, mais les résultats de la figure 3.8 ne sont malheureusement pas concluants.

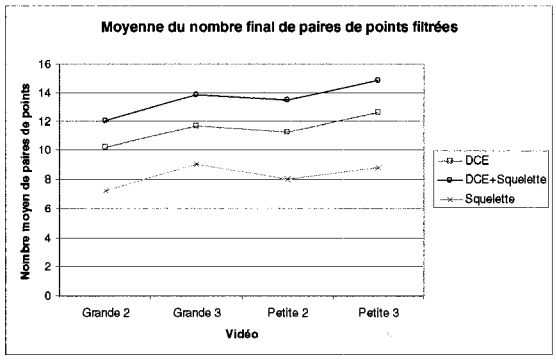

Figure 3.9 Nombre de paires de points filtrées selon l'algorithme

Toujours à la figure 3.9, on comprend un peu plus pourquoi l'algorithme DCE est légèrement meilleur que l'algorithme du squelette : pour chaque vidéo, l'algorithme DCE permet d'obtenir davantage de paires de points filtrées et la disparité médiane retenue est probablement plus fiable. Cependant, le nombre de paires de points n'est pas garant d'un meilleur score final, car le cas hybride n'a pas nécessairement de meilleurs scores finaux.

3.2.3 Impact des filtres pour l'algorithme du squelette

La figure 3.10 présente les résultats de l'application des filtres pour la méthode du squelette. Dans ces résultats, on voit que l'usage des paires de blobs des valeurs témoins (vt) pour le filtre des paires de blobs (bl) améliore tous les scores finaux et réduit les écart-types. Donc, il est pertinent d'utiliser ce filtre. Malheureusement, notre filtre de blobs n'est pas aussi performant lorsqu'on utilise les blobs trouvés automatiquement par notre programme. En d'autres mots, il y a des erreurs dans l'appariement des blobs.

Étrangement, l'usage du filtre RANSAC, à une ou deux itérations, n'a pas énormément d'impact positif sur la précision des disparités des paires de blobs. En effet, dans le cas où les caméras sont rapprochées (Petite 2 et 3), l'usage du filtre RANSAC réduit la précision des disparités. À vrai dire, il s'agit d'un résultat confirmant la nécessité de distancer les caméras pour avoir une matrice fondamentale numériquement stable. Autrement, cette matrice n'est pas fiable et nuit au filtre RANSAC.

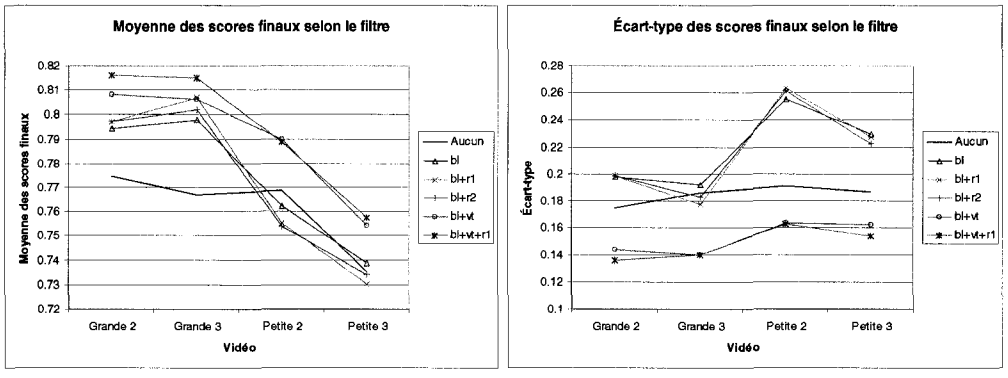

Figure 3.10 Impact de l'application de divers filtres à la suite de la méthode du squelette : blob (bl), valeurs témoins (vt), RANSAC (r1), deux RANSAC (r2). La référence est bl+r1

Finalement, puisque la seconde itération du filtre RANSAC était sensé être plus sévère, on assiste effectivement à une légère amélioration des résultats, donc cela laisse croire qu'il serait possible d'améliorer les résultats en modifiant les paramètres du niveau de confiance et de la distance maximale (tableau 3.3).

À la figure 3.11, on a les résultats des paramètres de l'algorithme RANSAC selon le nombre d'itérations (1 pour r1 et 2 pour r2). Le paramètre de niveau de confiance (-c

dans le tableau 3.3) n'a eu aucun impact, car les différents résultats étaient parfaitement identiques aux résultats du cas de référence (bl+r1) ou au cas à deux RANSAC (bl+r2). Voilà pourquoi les résultats du paramètre –c ne sont pas affichés dans les graphiques de la figure 3.11.

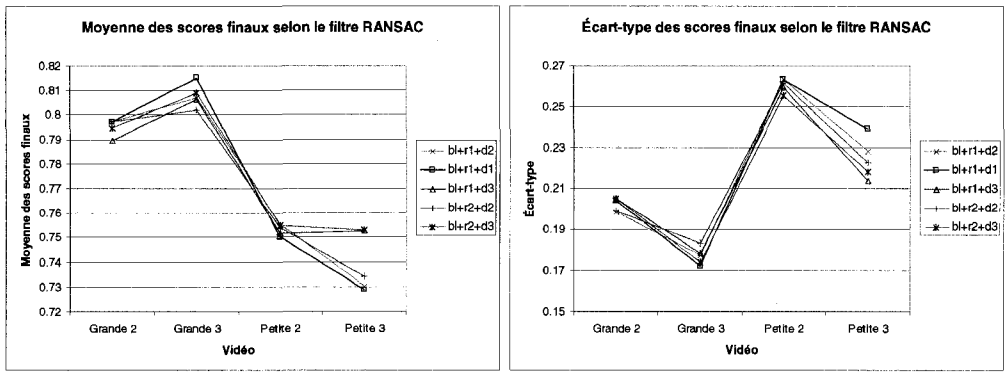

Figure 3.11 Impact du paramètre de la distance maximale (d1 ou d3) pour l'algorithme RANSAC et impact du nombre d'itérations de RANSAC (r1 ou r2). Dans tous les cas, le filtre des blobs est activé (bl). La référence est bl+r1+d2

Dans le cas du paramètre de la distance maximale (-d), celui-ci semble avoir un léger impact sur les résultats, mais cela dépend de la distance entre les caméras. En effet, lorsque la distance entre les caméras est grande, l'algorithme RANSAC semble plus stable pour déterminer la matrice fondamentale, donc on peut minimiser la distance maximale permise (utiliser –d 1) afin d'éliminer davantage de paires de points aberrantes. Par contre, lorsque la distance entre les caméras est plus petite, l'algorithme RANSAC est moins précis et il faut utiliser une distance –d plus élevée, c'est-à-dire 3.0. Par contre, si on utilise une valeur de 1.0 pour le paramètre –d, ce dernier restera à 1.0 peut importe le nombre d'itérations du filtre RANSAC étant donné que ce paramètre –d est mis à jour à l'aide d'une racine carrée. Par conséquent, seul le paramètre par défaut de –c aurait pu avoir un impact d'une itération à l'autre, mais cela n'a pas eu lieu.

Bref, le filtre des paires de blobs est indispensable et l'algorithme RANSAC fonctionne mieux lorsque les caméras sont suffisamment distancées. Par contre, si les caméras sont un peu rapprochées, on peut utiliser une distance maximale (paramètre –d) plus élevée afin de tolérer un maximum de bonnes paires de points en fonction de la matrice fondamentale calculée dans l'algorithme RANSAC.

3.2.4 Impact des filtres pour l'algorithme DCE

À la figure 3.12, on a la moyenne et l'écart-type des scores finaux pour l'algorithme DCE, et ce, en fonction des divers filtres utilisés. Encore une fois, on voit que l'usage du filtre des paires de blobs (bl) aide à améliorer la moyenne des scores finaux. Cependant, ce résultat est plus ou moins fiable étant donné que l'écart-type des scores finaux augmente avec l'usage des blobs trouvés automatiquement par notre programme (bl sans vt). Par contre, l'usage des valeurs témoins (bl+vt) améliore significativement les écart-types, ce qui signifie qu'il y a un potentiel non négligeable à utiliser le filtre des paires de blobs et qu'il faudra améliorer notre algorithme d'appariement des blobs.

Alors que l'usage des valeurs témoins réduit significativement l'écart-type des scores finaux, la moyenne de ces scores n'est pas nécessairement supérieure. En effet, dans le cas de la vidéo à deux acteurs où la distance entre les deux caméras est petite et où nous avons utilisé une itération de l'algorithme RANSAC (cas Petite 2 avec bl+vt+r1), l'usage des valeurs témoins est plus nuisible que si nous avions pris les blobs trouvés automatiquement par notre programme (cas bl+r1). À ce sujet, nous pouvons blâmer l'algorithme RANSAC, car le cas sans ce dernier filtre, (bl+vt), offre une meilleure moyenne. D'ailleurs, l'usage du filtre RANSAC ne semble pas toujours améliorer les résultats. Même la double itération ne semble pas aider généralement. Cela veut donc dire que les paramètres du filtre RANSAC par défaut ne sont pas au point.

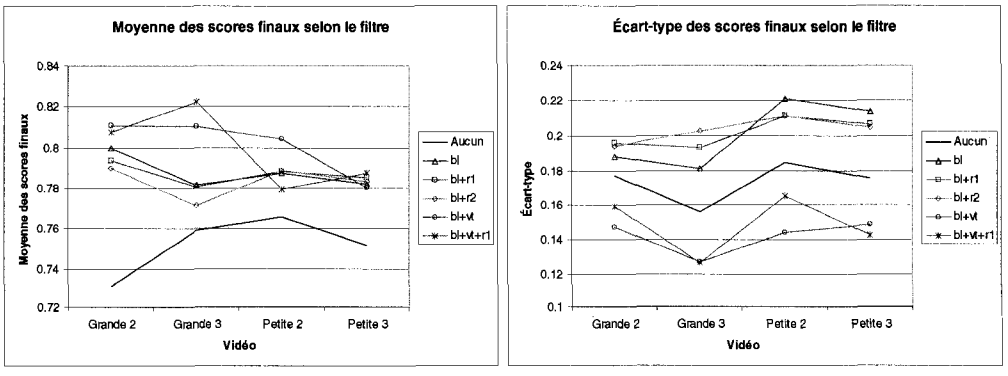

Figure 3.12 Impact de l'application de divers filtres à la suite du processus DCE : blob (bl), valeurs témoins (vt), RANSAC (r1), deux RANSAC (r2). La référence est bl+r1

À la figure 3.13, on a les résultats de la variation du paramètre –d (tableau 3.3) qui est la distance maximale pour l'algorithme RANSAC. Encore une fois, le paramètre –c n'a pas modifié les résultats des cas de référence (bl+r1) et (bl+r2), donc seuls ces deux derniers cas sont illustrés dans la figure 3.13.

L'usage du double RANSAC n'améliore toujours pas les résultats en général.

Tout comme dans la section précédente, le paramètre –d à la valeur de 1 permet d'avoir de meilleurs résultats lorsque la distance entre les deux caméras est grande. Inversement, lorsque ce paramètre a une valeur de 3, c'est dans les cas où les caméras sont rapprochées que les moyennes sont meilleures. Bref, ce paramètre a non seulement un impact sur les résultats, mais nous savons maintenant qu'il faudra le calibrer en fonction de la distance entre les caméras.

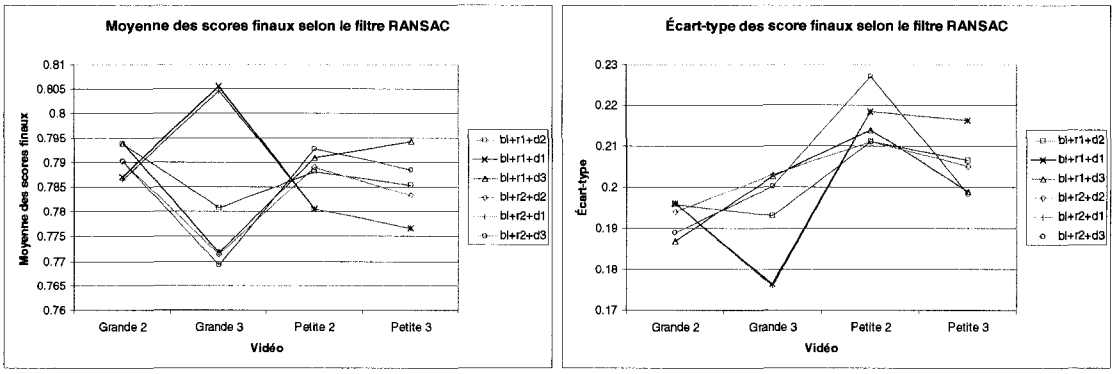

Figure 3.13 Impact du paramètre de la distance maximale (d1 ou d3) pour l'algorithme RANSAC et impact du nombre d'itérations de RANSAC (r1 ou r2). Dans tous les cas, le filtre des blobs est activé (bl). La référence est bl+r1+d2

Il reste un petit détail à noter pour le cas (bl+r2+d1). En effet les résultats sont légèrement différents du cas (bl+r1+d1) malgré le fait que la racine carrée de –d reste 1.0 et que le paramètre –c n'avait pourtant jamais eu d'impact. Or, cette fois-ci, on remarque que la seconde itération du filtre RANSAC n'améliore toujours pas les résultats, mais que ceci pourrait être dû au niveau de confiance qui serait devenu trop strict et qui jouerait en défaveur envers le score final moyen.

3.2.5 Étude des paramètres de la méthode du squelette

Dans cette étude, nous évaluons l'impact de l'utilisation de la fermeture à divers rayons (–f dans le tableau 3.4) au lieu de l'approximation du contour. La fermeture a pour but d'éliminer les petits trous et d'éliminer une partie du bruit dans les contours, et ce, en dilatant et en érodant les blobs. Alors que l'approximation du contour approxime explicitement le contour en fonction d'une distance maximale d'approximation, la fermeture cause indirectement une approximation du contour externe, ce qui permet d'éviter d'avoir trop de points caractéristiques causés par le bruit dans le contour (section 2.3.1.1). Par ailleurs, dans la présente étude, nous évaluons l'impact du seuillage sur la taille minimale des trous (paramètre –t). Enfin, lors de l'approximation des contours externes et internes, nous évaluons l'impact du paramètre de distance maximale d'approximation (paramètres –de et –di). Donc, ces trois derniers paramètres (en incluant –t) sont désactivés lorsque la fermeture est utilisée (voir le tableau 3.4).

Les résultats de l'application de la fermeture sont présentés à la figure 3.14. Selon ces résultats, la fermeture n'apporte pas vraiment de meilleurs résultats à tout coup, mais semble uniformiser les résultats d'une vidéo à l'autre. Donc, afin de contrer les petits défauts de la soustraction d'arrière-plan, la fermeture peut être utile pour lisser le contour des blobs, et ce, pour l'ensemble des vidéos utilisées. De plus, la fermeture permettrait ultimement d'éviter d'avoir à optimiser les différents paramètres en fonction de chaque vidéo; cela reste à être démontré par davantage de tests.

Il n'y a pas d'explication possible pour justifier la diminution de performance du cas de référence pour les vidéos Petite 2 et Petite 3. Normalement, lorsque les caméras sont plus distancées, la forme des blobs est plus différente et les squelettes devraient être très différents eux aussi. Alors, à part de mettre la faute sur les occlusions, la diminution de performance lorsque les caméras sont rapprochées n'a pas d'explication logique.

Les résultats des différents filtres des trous sont présentés à la figure 3.15. Selon ces courbes, il semblerait que nous puissions obtenir de meilleurs résultats en éliminant les trous de 256 à 1024 pixels carrés en plus de ceux qui ont moins de 256 pixels carrés.

Donc, non seulement ces résultats prouvent que la soustraction d'arrière-plan n'était pas complètement parfaite, mais aussi que ce filtre des trous est une bonne façon de tolérer les erreurs de notre soustracteur d'arrière-plan.

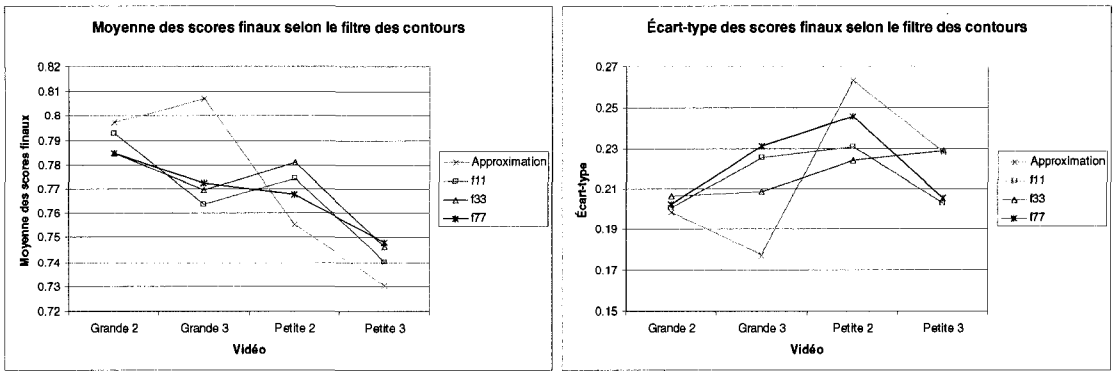

Figure 3.14 Résultats des différents degrés de fermeture pour la correction des contours de blobs. Le cas de référence est « Approximation ». Les cas fxy indiquent la taille x*y de l'ellipse utilisée pour effectuer la fermeture

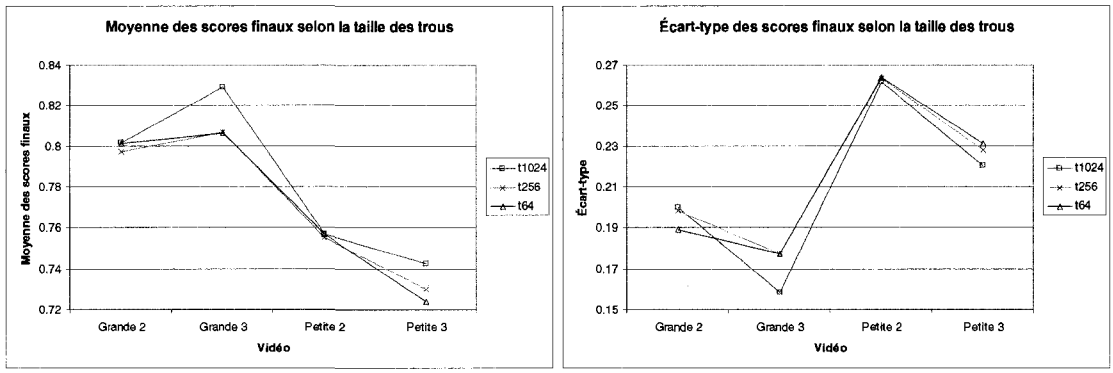

Figure 3.15 Résultats des différents seuils utilisés pour filtrer les trous dans les blobs. Les cas t*** indiquent le nombre minimal de pixels carrés pour qu'un trou soit gardé dans le calcul du squelette. Le cas de référence est t256

À la figure 3.16, on a les résultats des diverses approximations des contours des blobs. Lorsque nous approximons un contour, les arêtes du contour final sont à une distance maximale du contour d'origine. Tous comme dans le cas de la fermeture, il n'est pas évident d'évaluer si un cas d'approximation des contours est meilleur qu'un autre. En effet, les résultats dépendent des vidéos : il y a de meilleurs résultats pour les vidéos dont les caméras sont grandement séparées (Grande 2 ou 3). Néanmoins, pour toutes les vidéos, le cas de15+di15, c'est-à-dire le cas à une approximation de 1.5 pixels pour les contours externes et internes, donne les pires résultats. Autrement, lorsqu'on choisit une

distance d'approximation maximale de 3.5 pixels pour le contour externe, nous n'obtenons pas toujours de meilleurs résultats que le cas de référence, mais au moins les résultats sont un peu plus stables d'une vidéo à l'autre.

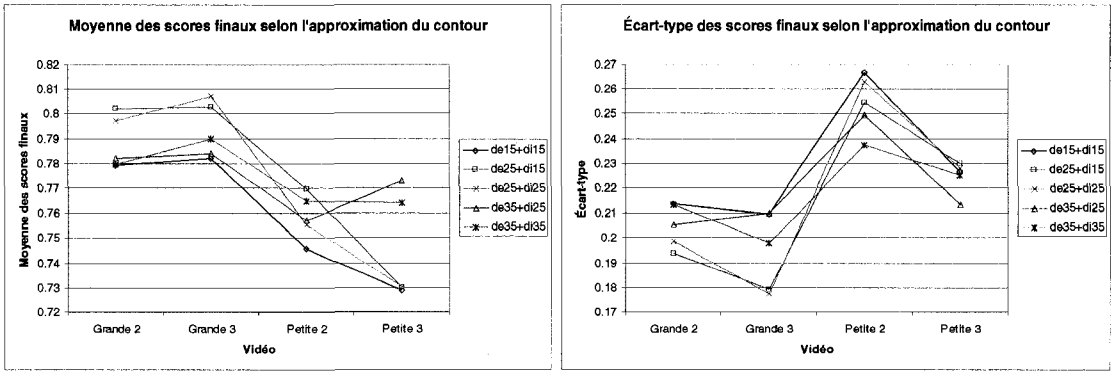

Figure 3.16 Résultats des différentes distances maximales d'approximation pour les contours externes (de) et internes (di). Les valeurs 15, 25 et 35 valent en réalité 1.5, 2.5 et 3.5. Le cas de référence est de25+di25 (2.5 pixels pour chaque type de contour)

3.2.6 Étude des paramètres de la méthode par processus DCE

La présente méthode utilise trois différents paramètres : la taille minimale des trous dans les blobs (–t), le nombre de points retenus sur le contour externe (–e) et le nombre de points retenus sur le contour interne (–i) (tableau 3.5).

La figure 3.17 présente les résultats du paramètre limitant la taille minimale des trous dans les blobs. Tout comme dans le cas de la méthode du squelette, il semblerait que la suppression des trous de moins de 1024 pixels carrés aiderait à améliorer les résultats. Cependant, cette fois-ci, il y a une séquence vidéo pour laquelle ce n'est pas vrai : la séquence vidéo à deux acteurs où la distance entre les caméras est petite. Par ailleurs, si on supprime que les trous plus petits que 64 pixels carrés, on obtient à tout coup des résultats inférieurs au cas de référence. Bref, à l'avenir, nous pouvons sans trop de souci doubler ou quadrupler la taille minimale permise des trous.

Les résultats du paramètre –e sont présentés à la figure 3.18. Selon les moyennes, il ne semble pas y avoir de tendance unanime, mais la réduction du nombre de points sur le contour final permet d'augmenter légèrement les moyennes. Par contre, le cas de référence semble plus stable envers chaque vidéo, autant pour la moyenne que pour

l'écart-type. Enfin, en réduisant le nombre de points, on remarque que les moyennes sont plus faibles pour les vidéos ayant trois acteurs par rapport aux cas correspondants à deux acteurs. Il s'agit probablement d'un signe que l'algorithme d'appariement a de la difficulté lorsqu'il y a plus de deux acteurs.

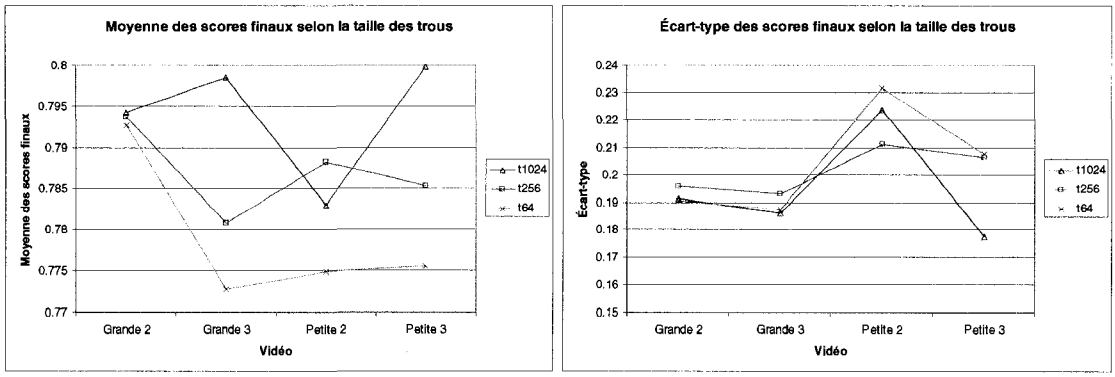

Figure 3.17 Résultats des différents seuils utilisés pour filtrer les trous dans les blobs. Les cas t*** indiquent le nombre minimal de pixels carrés pour qu'un trou soit gardé dans le calcul du squelette. Le cas de référence est t256

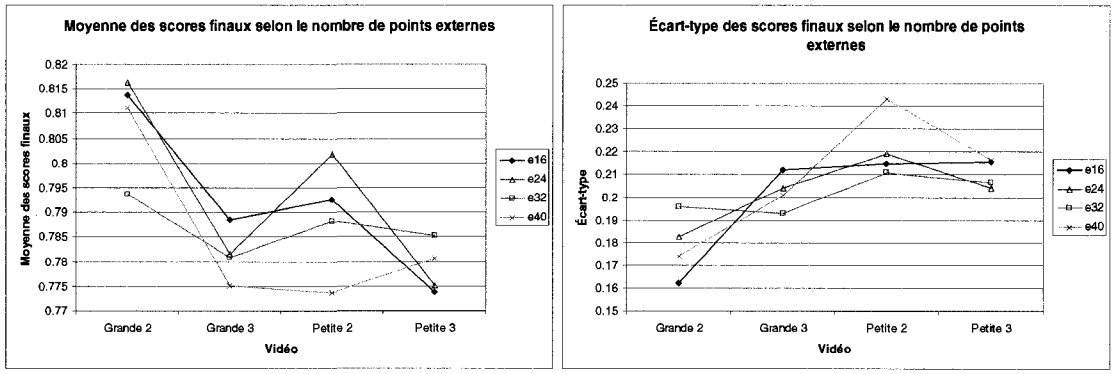

Figure 3.18 Impact du nombre de points retenus sur le contour externe à la fin du processus DCE. Les cas eX représentent le nombre X de points retenus. Le cas de référence est e32

Les résultats du paramètres –i sont présentés à la figure 3.19. Ces résultats sont plutôt étranges étant donné qu'il y a deux directions à prendre pour optimiser la majorité des moyennes et des écart-types : réduire le nombre de points à six ou augmenter ce nombre à seize. Parfois, dans les images du spectre visible, les trous sont causés par des régions concaves entourées par des pixels d'ombre (exemple : les jambes en marchant). Si on utilise un nombre élevé de points sur le contour interne, on obtient alors une approximation plus fidèle du contour original et cela permet d'apparier quelques points

supplémentaires ou d'éviter certaines mauvaises paires de points. Par contre, dans le cas des images infrarouges, il n'y a pas d'ombre. Ainsi, nous permettons que certains points d'un trou dans l'image visible puissent être appariés avec des points du contour externe dans l'image infrarouge : la longueur des arêtes doit alors être comparable pour avoir une valeur d'importance K (équation 2.7) équivalente. Voilà pourquoi les résultats s'améliorent lorsqu'on réduit le nombre de points retenus pour le contour des trous.

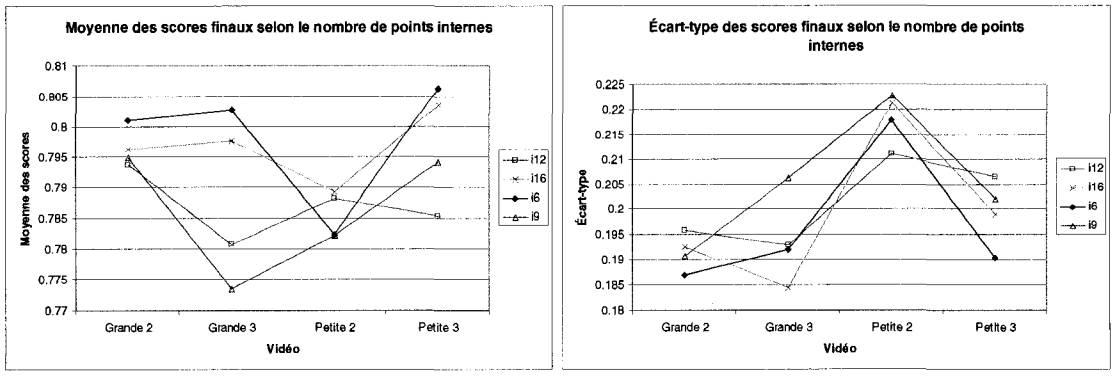

Figure 3.19 Impact du nombre de points retenus sur les contours internes à la fin du processus DCE. Les cas iX représentent le nombre X de points retenus. Le cas de référence est i12

3.2.7 Étude des paramètres de la méthode des valeurs propres minimales

Pour cet algorithme, nous avons testé l'impact du niveau de qualité (–ql dans le tableau 3.7) et de la distance minimale (–md) entre chaque point caractéristique retenu.

Selon les moyennes illustrées dans la figure 3.20, l'usage d'un niveau de qualité élevé semble avoir un meilleur potentiel qu'un niveau de qualité plus bas. Cependant, le score final peut avoir une valeur parfaite si la méthode ne permet de trouver aucune paire valide de blobs (1 - 0 / 0 = 1 dans l'équation 3.5). En effet, en observant les écart-types, on constate que les niveaux de qualité supérieurs (0.40 et 0.50) causent une plus grande variation des scores dans les vidéos où les caméras sont peu distancées (Petite 2 et Petite 3). Donc, puisque les résultats de la figure 3.20 sont biaisés, il faut plutôt étudier les bonnes paires de blobs et celles qui sont manquantes à la figure 3.21. Dans cette figure, on remarque qu'il y a effectivement moins de bonnes paires de blobs lorsque le niveau de qualité est élevé. De plus, la moyenne des paires de blobs manquantes augmente en

fonction du niveau de qualité. Ainsi, cela prouve que, plus le niveau de qualité est élevé, moins la méthode est capable de trouver un nombre suffisant de points caractéristiques permettant l'appariement des blobs et, donc, l'obtention d'un meilleur score final non biaisé.

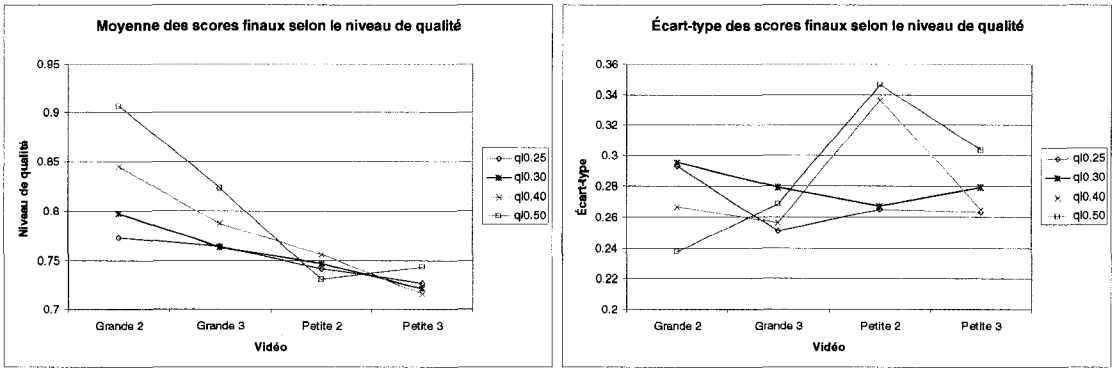

Figure 3.20 Moyenne et écart-type des scores finaux pour chaque niveau de qualité des points caractéristiques trouvés par la méthode des valeurs propres minimales. Le cas de référence est ql0.30

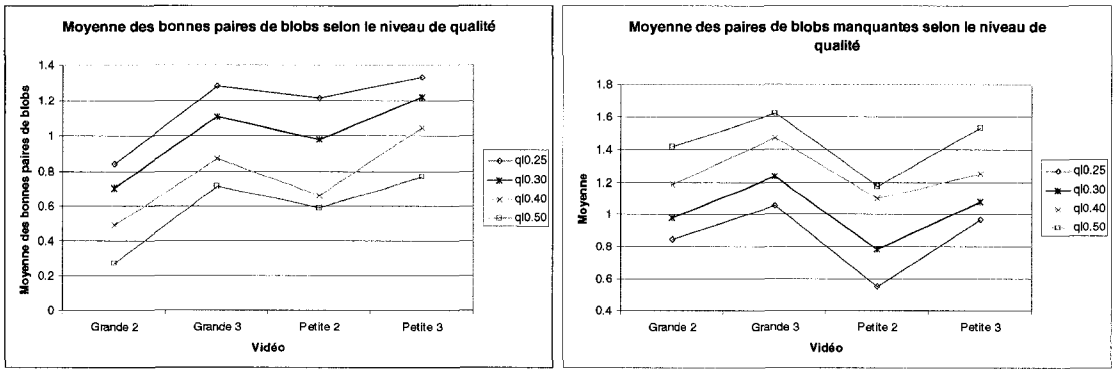

Figure 3.21 Moyenne des bonnes paires de blobs et des paires de blobs manquantes selon le niveau de la qualité des points caractéristiques trouvés par la méthode des valeurs propres minimales. Le cas de référence est ql0.30

Selon les figures 3.22 et 3.23, nous pouvons en déduire que l'usage d'une plus grande distance entre les points caractéristiques retenus permet parfois d'obtenir de meilleurs scores finaux. Cela est dû au fait qu'il y a parfois un plus grand nombre de mauvaises paires de blobs et parfois un plus grand nombre de paires de blobs manquantes. Donc, les scores finaux sont biaisés s'ils restent élevés tout en ayant des paires de blobs manquantes. Il en résulte que le cas de référence (md2) et le cas de la distance de trois pixels (md3) donnent les résultats les plus fiables. Nous avons aussi essayé le cas à une

distance de un pixel (md1), mais ce cas donne exactement les mêmes résultats que le cas de référence (md2).

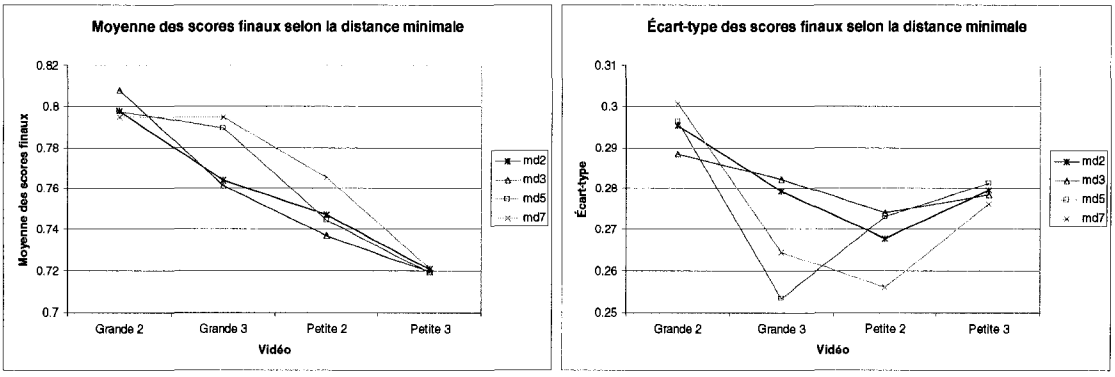

Figure 3.22 Moyenne et écart-type des scores finaux pour chaque distance minimale entre les points caractéristiques trouvés par la méthode des valeurs propres minimales. Le cas de référence est md2

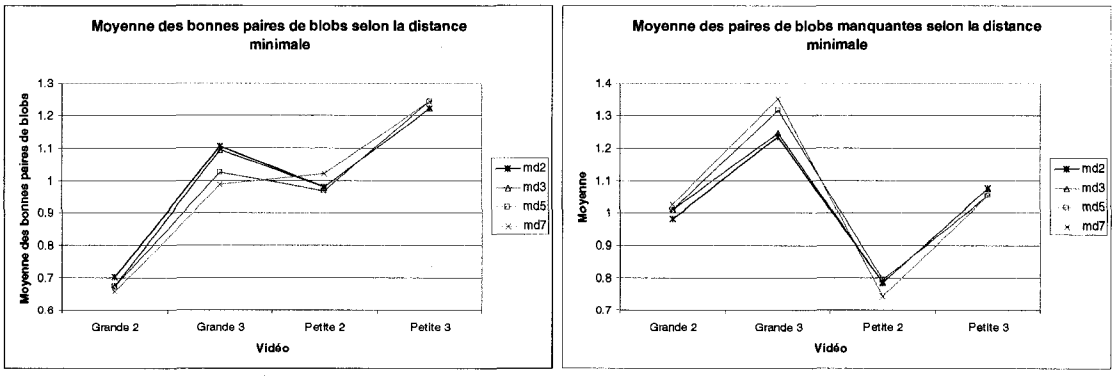

Figure 3.23 Moyenne des bonnes paires de blobs et des paires de blobs manquantes selon la distance minimale entre les points caractéristiques trouvés par la méthode des valeurs propres minimales. Le cas de référence est md2

Bref, à la lumière de cette étude, la méthode des valeurs propres minimales telle qu'elle est implémentée ne semble pas appropriée pour des paires d'images provenant de différents capteurs. En effet, le filtre des paires de blobs ne fonctionne pas à son meilleur, car les paires de blobs elles-mêmes, déterminées à partir des premières paires de points, ne sont pas fiables. Néanmoins, les résultats sont meilleurs lorsqu'on réduit le niveau de qualité (–ql).

3.2.8 Étude des paramètres de la méthode de Harris et Stephens

Comme à la section précédente, nous évaluons l'impact des paramètres de niveau de qualité des points caractéristiques et de la distance minimale entre ces points, et ce, pour la méthode de Harris et Stephens.

Selon les figures 3.24 et 3.25, nous obtenons sensiblement les mêmes résultats qu'à la section précédente, c'est-à-dire qu'une hausse du niveau de qualité des points caractéristiques entraîne une baisse du nombre de ces points, ce qui donne davantage de paires de blobs manquantes. Donc, il faut utiliser des niveaux de qualité de 0.25 à 0.30.

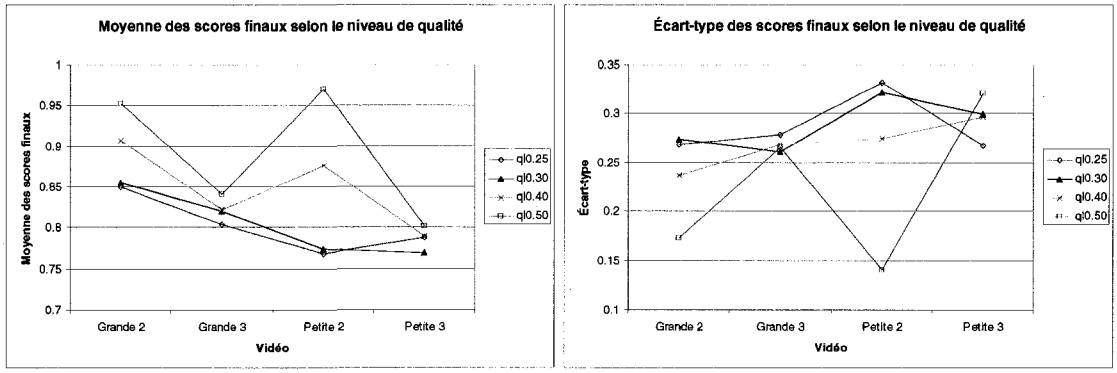

Figure 3.24 Moyenne et écart-type des scores finaux pour chaque niveau de qualité des points caractéristiques trouvés par la méthode de Harris et Stephens. Le cas de référence est ql0.30

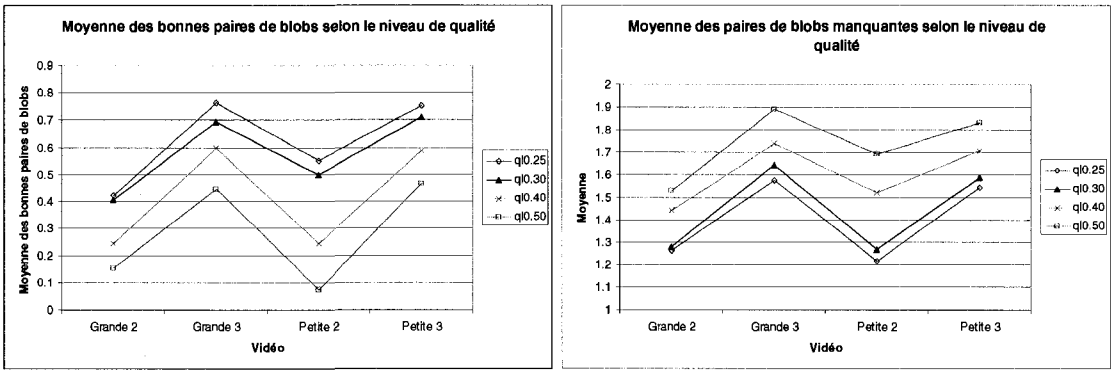

Figure 3.25 Moyenne des bonnes paires de blobs et des paires de blobs manquantes selon le niveau de la qualité des points caractéristiques trouvés par la méthode de Harris et Stephens. Le cas de référence est ql0.30

Les résultats de l'étude de la distance minimale entre les points caractéristiques trouvés par la méthode de Harris et Stephens sont présentés dans les figures 3.26 et 3.27. Encore une fois, lorsque la distance minimale entre les points est élevée, cela cause la perte de

certains points caractéristiques et, donc, la perte de bonnes paires de blobs. Dans le cas de Petite 2, les cas md5 et md7 ont un score moyen élevé, car ils ont beaucoup de scores finaux de 1, ce qui indique un manque de paires de blobs valides. Bref, il faut garder un maximum de points caractéristiques à l'aide d'une distance de deux (le cas de référence) ou trois pixels maximum entre ces points.

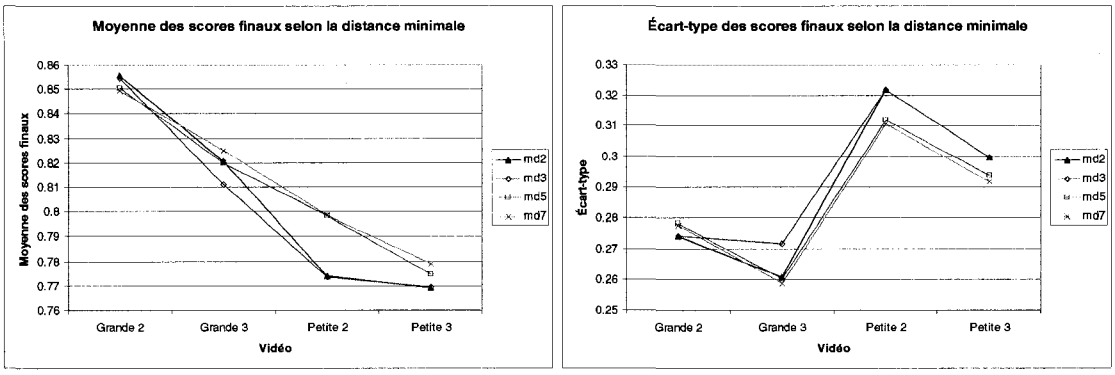

Figure 3.26 Moyenne et écart-type des scores finaux pour chaque distance minimale entre les points caractéristiques trouvés par la méthode de Harris et Stephens. Le cas de référence est md2

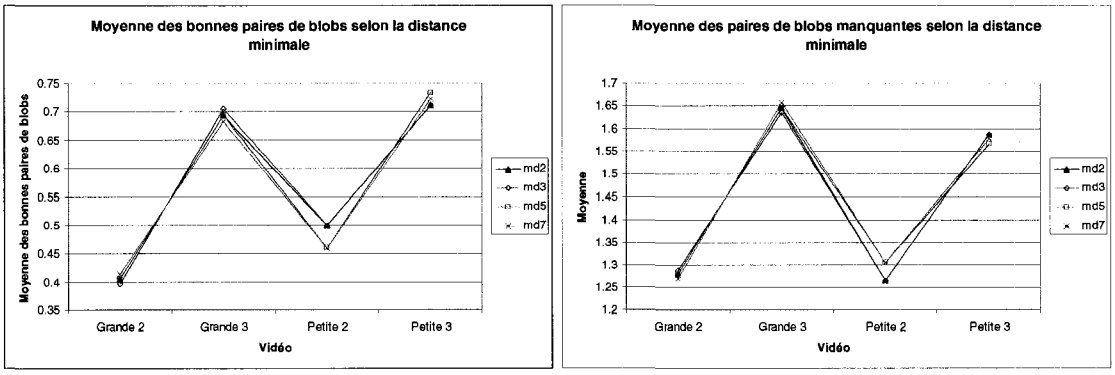

Figure 3.27 Moyenne des bonnes paires de blobs et des paires de blobs manquantes selon la distance minimale entre les points caractéristiques trouvés par la méthode de Harris et Stephens. Le cas de référence est md2

Dans la figure 3.27, on remarque une fois de plus que les séquences vidéo à trois acteurs ont plus de paires de blobs manquantes et vice-versa. Évidemment, s'il y a deux acteurs, on peut manquer jusqu'à deux paires de blobs. Par contre, dans le scénario à trois acteurs, si la méthode d'appariement est déficiente, alors il est normal d'avoir davantage de paires de blobs manquantes.

3.2.9 Résultats selon les actes dans les scénarios

À la section 3.1.1, nous avons présenté la configuration de l'environnement (figure 3.1) dans lequel nous avons enregistré deux scénarios différents, à deux et trois acteurs (tableaux 3.1 et 3.2), et avec deux configurations de caméras, c'est-à-dire éloignées (Grande) et rapprochées (Petite). Les scénarios avaient pour but d'évaluer nos algorithmes en fonction de diverses distances entre les acteurs et de diverses distances entre les acteurs et les caméras. La présente section présente les résultats concernant le potentiel de chaque méthode utilisée dans chacun des actes de chaque scénario. Pour obtenir le potentiel, nous avons ciblé une paire d'images (visible et infrarouge) pour chaque acte telle que les résultats de la majorité des méthodes sont les meilleurs possibles.

Selon la figure 3.28, on constate que les moyennes des scores finaux tournent autour de 0.9 pour plusieurs des actes du scénario pour les algorithmes du squelette, du processus DCE et de la congruence de phase. Globalement, la méthode du squelette semble être la plus stable d'un acte à l'autre, mais il ne s'agit que de son potentiel (les paires d'images ont été choisies). Par ailleurs, la congruence de phase se débrouille assez bien. Il faut croire que notre adaptation de l'appariement des points est assez bonne finalement.

L'acte le plus manqué est R1-L4. Pourtant, selon les paires de points trouvés et illustrés à la figure 3.29, le résultat semble très acceptable pour identifier quel acteur est le plus près de la caméra. Malheureusement, nous suspectons une trop grande sensibilité de notre score final pour les faibles disparités. En effet, lorsque deux petites disparités sont comparées, une différence de un seul pixel a énormément d'impact sur le score final qui est normalisé. Par exemple, selon les valeurs témoins de la figure 3.29 (à gauche), l'acteur de gauche a une disparité horizontale moyenne relativement très faible. Par contre, selon les résultats de notre implémentation, la disparité horizontale médiane est plus grande. Il en résulte donc un faible score final lorsqu'on compare ces deux disparités pourtant équivalentes ou semblables. Bref, il faudra réviser notre formule du score final qui est beaucoup trop sensible pour les petites disparités. Après vérification, nous constatons que ce phénomène se produit aussi pour les actes R4-L3 et R4-L1.

Dans le cas de l'acte C2-L1 pour l'algorithme des valeurs propres minimales, le mauvais score est dû à l'usage d'une mauvaise paire de blobs : les paires de points sont aberrantes et donnent un résultat totalement erroné.

Finalement, dans le cas de l'acte R3-C0, nos deux méthodes (Squelette et DCE) trouvent exactement la même paire de blobs, mais la disparité médiane trouvée avec la méthode du squelette se rapproche davantage de la bonne réponse, d'où le meilleur résultat que celui de la méthode du processus DCE.

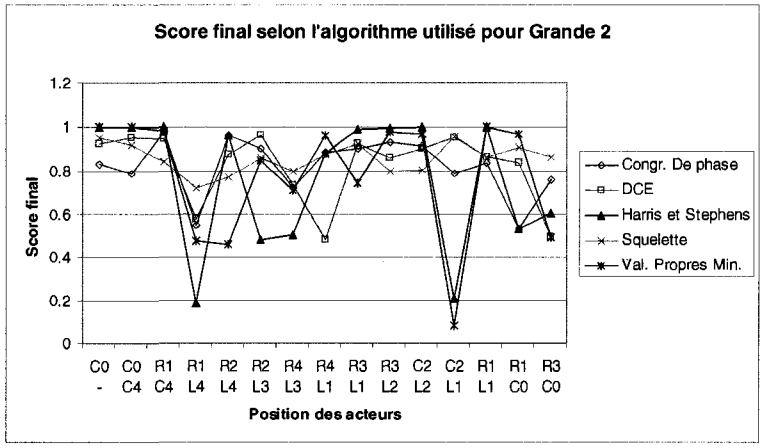

Figure 3.28 Score final selon chaque acte et selon chaque algorithme pour la séquence Grande 2

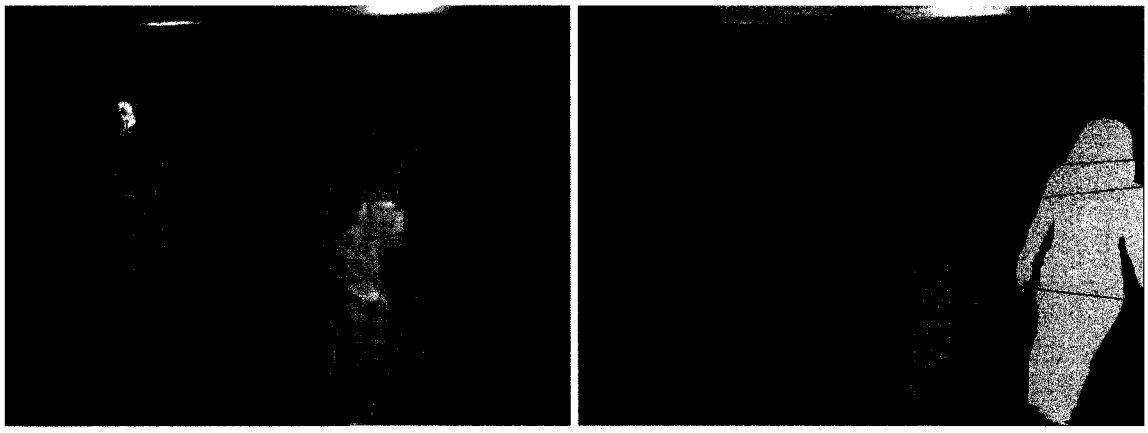

Figure 3.29 Images choisies pour représenter l'acte R1-L4 de la séquence Grande 2. À gauche, on a les disparités témoins et leur moyenne en orange (ligne large et pâle). À droite, on retrouve la moyenne des images visible et infrarouge, les paires de points retenues avec le processus DCE et une distance relative des blobs : plus le blob est pâle, plus l'acteur correspondant est près des caméras. En fait, seuls les blobs du spectre visible ont une teinte représentant la profondeur relative

La figure 3.30 présente les scores finaux pour chaque acte du même scénario, mais en ayant les caméras plus rapprochées. Encore une fois, nous remarquons de plus faibles résultats à chaque fois qu'il y a usage des positions C4, L4 et R4 : il s'agit non seulement du phénomène du score final trop sensible envers les faibles disparités, mais aussi de quelques défauts de soustraction d'arrière-plan dus aux ombres et aux couleurs semblables entre les acteurs et l'arrière-plan. Par exemple, dans le cas de l'algorithme DCE et de l'acte R4-L1 (figure 3.31), l'actrice en R4 est mal définie par la soustraction d'arrière-plan dans l'image du spectre visible, ce qui a causé quelques paires de points aberrantes définitivement plus longues que la disparité témoin de cette paire de blob.

Par ailleurs, il est plutôt difficile de comparer les méthodes. Certes, les méthodes du squelette, du processus DCE et de la congruence de phases sont les plus stables, mais il est plutôt difficile d'évaluer laquelle des méthodes est supérieure aux autres, car il y a plusieurs croisements. Néanmoins, la courbe de Squelette semble assez souvent au-dessus de la courbe de DCE.

Pour ce qui est des méthodes de Harris et Stephens et des valeurs propres minimales, ou bien elles sont instables, ou bien leurs scores finaux plafonnent à 1.0 alors que ce score en apparence parfait indique un manque de bonnes paires de blobs.

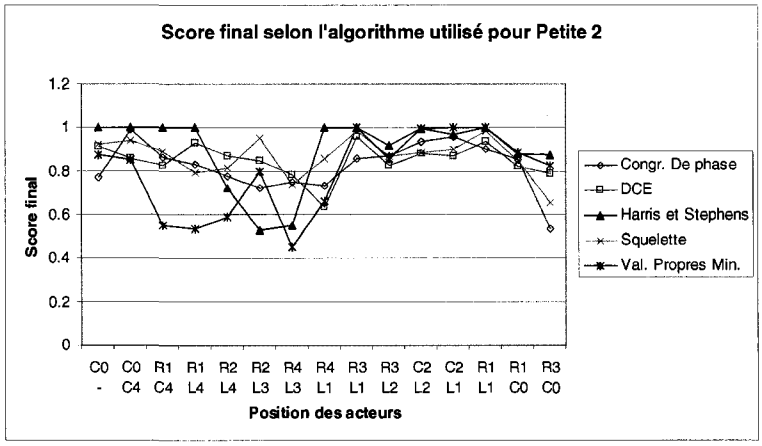

Figure 3.30 Score final selon chaque acte et selon chaque algorithme pour la séquence Petite 2

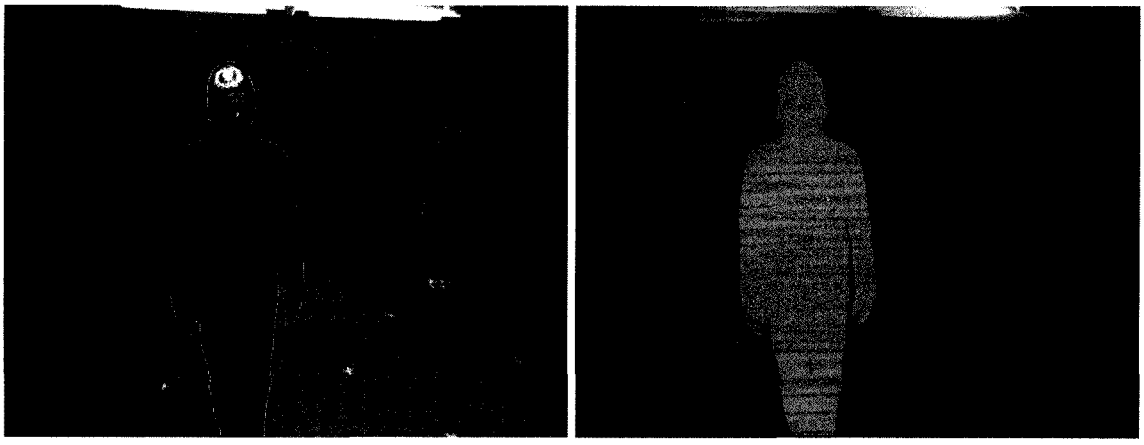

Figure 3.31 Images choisies pour représenter l'acte R4-L1 de la séquence Petite 2. À gauche, on a le contour des blobs, les disparités témoins et leur moyenne en orange (ligne large et pâle). À droite, on retrouve la moyenne des images visible et infrarouge, les paires de points retenues avec le processus DCE et une distance relative des blobs : plus le blob est pâle, plus l'acteur correspondant est près des caméras. En fait, seuls les blobs du spectre visible ont une teinte représentant la profondeur relative

Finalement, l'acte R3-C0 de la séquence Petite 2 (figure 3.32) obtient de mauvais résultats principalement parce que l'acteur en C0 cache presque entièrement l'actrice en R3, et ce, du point de vue de la caméra de gauche. Dans ce cas, il faudrait utiliser des techniques de segmentation afin de séparer les deux blobs fusionnés. Par ailleurs, étant donné que les divers algorithmes favorisent les courtes disparités pour éviter des paires aberrantes (équation 2.4), notre programme a préféré apparier le blob fusionné dans l'image visible avec le blob correspondant à l'actrice de droite dans l'image infrarouge : en pratique, cette paire de blobs est valide, mais il manque la paire de blobs qui correspond à l'acteur en C0.

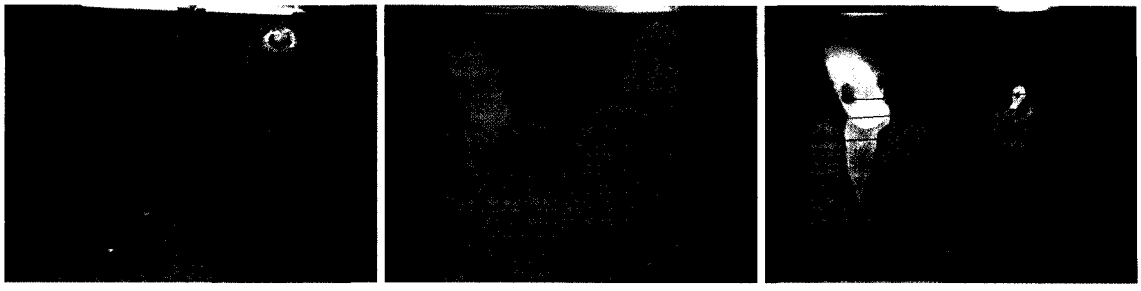

Figure 3.32 Exemple d'occlusion majeure causant de mauvaises paires de points. Dans l'image centrale, seuls les blobs du spectre visible ont une teinte représentant la profondeur relative. Plus le blob est pâle, plus l'acteur correspondant est près des caméras

Les résultats pour le scénario à trois acteurs filmé par les deux caméras grandement séparées sont présentés à la figure 3.33. À l'acte C0-C4-R3 et pour l'algorithme DCE, nous avons encore un problème avec un acteur en R3 partiellement caché par l'acteur en C0 (figure 3.34 à droite). Malheureusement, notre implémentation néglige la paire de blobs correspondant à l'acteur en C0, car celle de l'acteur en R3 permet d'avoir des disparités de plus faible longueur pour éviter les paires de points aberrantes (équation 2.4). Malgré tout, les paires de points retenues ne sont malheureusement pas toutes d'une très grande qualité, d'où le mauvais résultat en C0-C4-R3.

Par contre, le cas C0-C4 est un sans faute (figure 3.34 à gauche), et ce, malgré sa note relativement moyenne (0.87 sur 1.0). En effet, la paire de blobs de l'acteur près des caméras va à l'encontre de l'effet voulu de l'équation 2.4 qui force les petites disparités. Ensuite, les paires de points sont toutes pertinentes. Donc, le résultat qualitatif est définitivement meilleur que le résultat quantitatif.

Vers la fin de ce scénario à trois acteurs, il y a encore énormément d'occlusions faisant en sorte que la soustraction d'arrière-plan ne puisse pas bien segmenter chaque acteur dans chaque image. Par exemple, dans les deux derniers actes, l'acteur en R1 ou en R2 est caché par l'actrice en C0 dans les deux cas. Bref, un des maillons faibles de notre méthode est l'usage de la soustraction de l'arrière-plan pour segmenter les acteurs dans chaque image.

Finalement, la figure 3.33 montre que l'algorithme du squelette est très légèrement plus stable que le processus DCE. Ces deux algorithmes sont eux-mêmes légèrement meilleurs que la congruence de phase, sauf dans quelques cas problèmes où un des acteurs est dans le fond de la scène (le cas L1-C4-R2 par exemple). Enfin, les deux derniers algorithmes sont trop instables et donnent encore des scores finaux parfaits montrant le manque de bonnes paires de blobs. Néanmoins, l'algorithme des valeurs propres minimales semble plus stable que l'algorithme de Harris et Stephens.

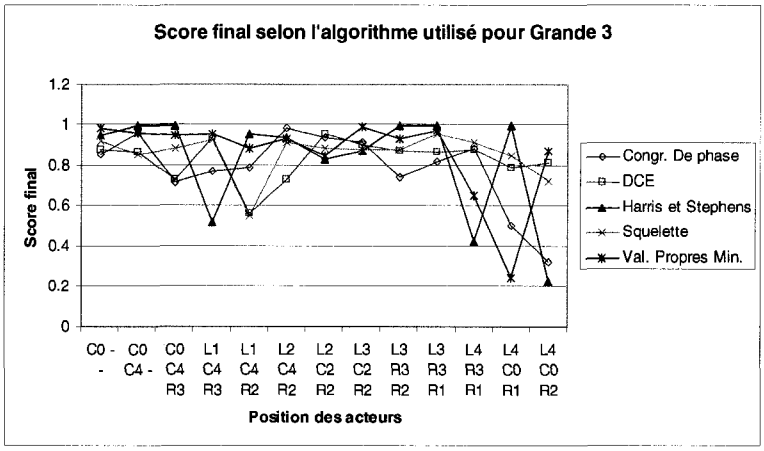

Figure 3.33 Score final selon chaque acte et selon chaque algorithme pour la séquence Grande 3

Figure 3.34 Résultats de l'algorithme DCE pour les actes C0-C4 et C0-C4-R3 respectivement. Seuls les blobs du spectre visible ont une teinte représentant la profondeur relative. Plus le blob est pâle, plus l'acteur correspondant est près des caméras

Dans la figure 3.35, les résultats sont différents de la figure 3.33, car les occlusions ne sont pas les mêmes étant donné que les caméras sont positionnées différemment. Nous notons, entre autres, que les deux derniers actes ont été pénibles pour nos méthodes. En effet, l'actrice en C0 se retrouve dans le même blob que l'acteur en L4 (figure 3.36) : la paire de blobs correspondante existe et est valide, mais la disparité calculée par notre programme est beaucoup trop grande pour la disparité enregistrée dans les valeurs témoins. Ceci explique donc le faible score final de l'algorithme DCE pour les deux derniers actes (figure 3.36). Bref, il s'agit encore une fois d'un problème de segmentation.

Globalement, dans le cas de Petite 3 (figure 3.36), l'algorithme DCE semble donner des résultats légèrement supérieurs à ceux de l'algorithme du squelette. Ces deux algorithmes sont plus stables que les trois autres. Définitivement, les résultats sont plus stables au milieu du scénario lorsqu'il y a moins d'occlusions. Enfin, lorsque les acteurs s'éloignent des caméras (de L2-C2-R2 à L3-R3-R2 par exemples), les scores finaux des méthodes du squelette et du processus DCE ont tendances à diminuer justement parce que les disparités sont plus petites et l'erreur relative est plus grande.

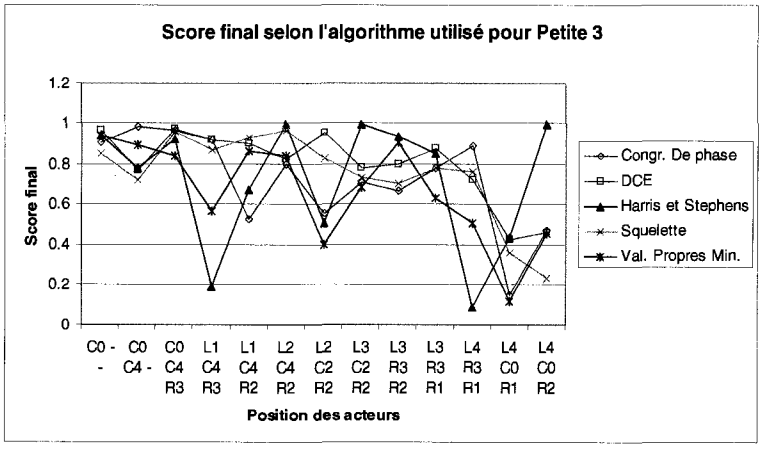

Figure 3.35 Score final selon chaque acte et selon chaque algorithme pour la séquence Petite 3

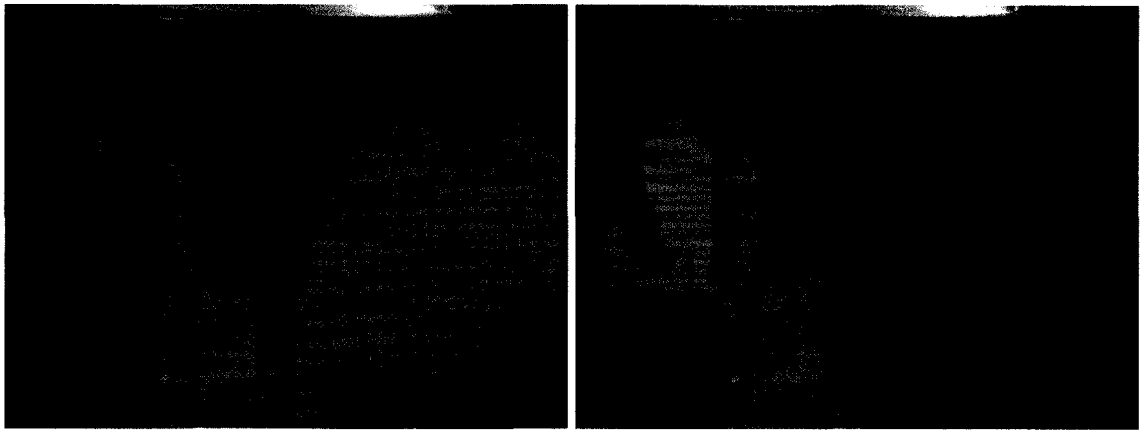

Figure 3.36 Les deux derniers actes pour l'algorithme DCE. Dans les deux cas, l'acteur de gauche est apparié au même blob que l'actrice au centre en avant. Par conséquent, alors que la disparité devait être faible (moins de dix pixels), la disparité médiane est nettement trop grande. Dans ces images, seuls les blobs du spectre visible ont une teinte représentant la profondeur relative. Plus le blob est pâle, plus l'acteur correspondant est près des caméras, sauf s'il y a des erreurs dans les disparités, ce qui est le cas ici

3.3 Sommaire des résultats

Selon nos résultats, nos deux méthodes surpassent légèrement notre adaptation de l'utilisation de la congruence de phase qui surpasse elle-même largement les algorithmes des valeurs propres minimales et de l'opérateur de Harris et Stephens. Entre nos deux méthodes, il n'est pas encore très clair laquelle est la meilleure étant donné que les différentes courbes présentées dans les sections précédentes se croisent souvent : tantôt l'une est meilleure, tantôt c'est l'autre qui est meilleure. Cependant, l'algorithme DCE semble trouver davantage de paires de points que l'algorithme du squelette, ce qui pourrait jouer en sa faveur lors du choix du meilleur algorithme. D'un autre côté, l'algorithme du squelette nous a parfois semblé plus stable, ce qui joue aussi en sa faveur. De même, nous avons trouvé que l'algorithme des valeurs propres minimales est plus stable que l'algorithme de Harris et Stephens.

Nous avons aussi testé les deux filtres de paires de points. Le premier filtre, celui utilisant les paires de blobs, est indispensable, mais il requiert des paires de blobs fiables. En utilisant ce filtre, nos méthodes du squelette et du processus DCE performent relativement bien par rapport aux méthodes adaptées de la littérature : la congruence de phase, les valeurs propres minimales et l'opérateur de Harris et Stephens. Dans le cas du second filtre, l'algorithme RANSAC modélisant des matrices fondamentales, le fameux niveau de confiance n'a montré aucun impact significatif ou positif lors de sa modification. Par contre, le paramètre de distance maximale améliore les résultats selon la configuration des deux caméras : une faible valeur pour le cas des caméras éloignées et une valeur plus élevée pour les caméras rapprochées. Ceci confirme que l'algorithme RANSAC utilisé pour calculer la matrice fondamentale est plus stable lorsque les caméras sont plus éloignées. De plus, les paramètres par défaut de cet algorithme n'étaient pas les bons.

Nous avons aussi testé les différents paramètres de quelques algorithmes dont les deux nôtres. Pour ces derniers, nous avons trouvé qu'il fallait, de préférence, éliminer les trous de moins de 1024 pixels carrés dans nos blobs. Nous avons constaté que l'algorithme du squelette est un peu plus stable lorsque les contours sont approximés par une légère

fermeture (3 par 3) ou par une approximation d'au plus 3.5 pixels de l'original. Dans le cas de l'algorithme DCE, si on retient 32 points pour le contour externe des blobs, on obtient des résultats plus stables que si on en retient plus ou moins. Trente-deux (32) points est donc le nombre de points requis pour bien représenter un humain avec un peu de bruit dans le contour. Par contre, dans les contours internes, nous pouvons augmenter ou réduire le nombre de points, ce qui a pour effet de garder davantage de détails ou d'avoir des points internes comparables à ceux des points externes sur le blob correspondant dans l'autre image.

Selon les scénarios utilisés, nous obtenons de meilleurs résultats lorsqu'il n'y a pas d'occlusions. En effet, avec une meilleure soustraction d'arrière-plan combinée à une segmentation des différents acteurs coincés dans des blobs communs, nous devrions obtenir des disparités semblables à celles des valeurs témoins. D'ailleurs, lorsque les acteurs sont éloignés et que leur disparité est petite d'une image à l'autre, nous avons constaté que notre score final est trop sensible, car le résultat visuel (qualitatif) est parfois meilleur que le résultat numérique (quantitatif).

CONCLUSION ET TRAVAUX FUTURS

Notre projet consistait à développer et à tester un système de vision stéréo en traitant simultanément des paires d'images constituées d'une image du spectre visible et d'une image du spectre infrarouge. Pour l'instant, notre système ne permet toujours pas de reconstruire la scène en entier et en détails, mais il se concentre sur les éléments en mouvement ou dans l'avant-plan pour leur donner un coefficient de profondeur.

Selon la littérature sur le sujet de la vision stéréo, il y a effectivement plusieurs méthodes permettant de trouver des points correspondants d'une image à l'autre. Malheureusement, ces méthodes ne sont pas adéquates pour des images aussi différentes qu'une image du spectre visible et qu'une image infrarouge. De même, il existe bien quelques techniques d'appariement par matrice de correspondance, mais cette matrice nécessite un critère de comparaison qui n'est pas nécessairement au point pour des images de type différent. Bref, à notre connaissance, les méthodes classiques de recherche et d'appariement de points caractéristiques ne peuvent pas nous aider. D'ailleurs, lors de l'expérimentation, même l'algorithme de la congruence de phase avait un critère d'appariement de notre conception et qui considérait les différences entre les types d'images. Néanmoins, après avoir trouvé nos propres critères d'appariement, nous les avons bien évidemment utilisés avec une matrice de correspondance.

Nos principales contributions consistent en deux méthodes de recherche de points caractéristiques basées sur des techniques classiques ou vues dans la littérature mais utilisées dans d'autres contextes : la méthode du squelette et la méthode du processus DCE. Bien sûr, pour chacune de ces méthodes, nous avons défini leur critère d'appariement. Par ailleurs, nous avons introduit l'usage des paires de blobs pour améliorer le rendement de l'appariement des points. Enfin, puisque la littérature présentait déjà un filtre RANSAC adapté pour modéliser des paires de points dans un système de vision stéréo, nous avons décidé de l'utiliser dans le but d'essayer d'améliorer nos résultats. La sortie de notre programme est la disparité horizontale médiane de chaque paire de blobs correspondants.

Selon nos résultats, nos deux méthodes surpassent légèrement notre adaptation de l'utilisation de la congruence de phase qui surpasse elle-même largement les algorithmes des valeurs propres minimales et de l'opérateur de Harris et Stephens. Nous avons aussi testé les deux filtres de paires de points : le filtre des paires de blobs et l'algorithme RANSAC. D'une part, le filtre des paires de blobs fonctionne mieux lorsque les paires de blobs sont fiables. D'autre part, l'algorithme RANSAC gagnerait à avoir une meilleure configuration que celle que nous avons choisie par défaut. Enfin, le calcul de la matrice fondamentale est plus stable lorsque les caméras sont plus éloignées.

Nous avons aussi testé les différents paramètres de quelques algorithmes dont les deux nôtres. Entre autres, nous obtenons de meilleurs résultats lorsque nous éliminons les trous de moins de 1024 pixels carrés dans nos blobs. De plus, l'algorithme du squelette est un peu plus stable lorsque les contours sont approximés par une légère fermeture (3 par 3) ou par une approximation d'au plus 3.5 pixels de l'original. Enfin, dans le cas de l'algorithme DCE, si on retient 32 points pour le contour externe des blobs, on obtient des résultats plus stables que si on en retient plus ou moins.

Selon les scénarios utilisés, nous obtenons de meilleurs résultats lorsqu'il n'y a pas d'occlusions. De plus, lorsque les acteurs sont éloignés, le résultat visuel est parfois meilleur que le résultat numérique du score final.

Travaux futurs

Il y a plusieurs approches possibles pour améliorer notre travail. Nous pensons à l'usage de meilleurs algorithmes de soustraction d'arrière-plan et possiblement l'usage d'algorithmes de segmentation pour diviser certains blobs. Par la suite, il serait bien d'ajouter un algorithme de suivi afin d'améliorer le calcul des paires de blobs. Le filtre RANSAC gagnerait à être mieux configuré ou à le rendre moins sensible à la distance entre les caméras : nous pensons, entre autres, à augmenter davantage le niveau de confiance (le paramètre –c) de l'algorithme RANSAC tel que suggéré par la bibliothèque OpenCV. Il serait aussi intéressant de rechercher des points caractéristiques dans l'arrière-plan, question de situer les acteurs par rapport à leur environnement. Il serait

aussi possible d'effectuer un rendu tridimensionnel des blobs pour effectuer un suivi tridimensionnel. Pour le domaine de la vidéosurveillance, il serait alors facile d'identifier la vitesse des acteurs, et ce, afin de prévoir les comportements anormaux.

La méthode d'évaluation du présent projet peut aussi être améliorée. Nous pensons immédiatement à notre score final qui est trop sensible pour les petites disparités.

Finalement, nous avons évalué la précision des disparités, mais il serait aussi possible d'évaluer explicitement si nos algorithmes sont capables de déterminer quel blob est devant l'autre. Pour l'instant, notre hypothèse était que des disparités parfaites permettraient de déterminer l'ordre des blobs en fonction de leur distance par rapport au système de caméras. Or, étant donné que notre score final est trop sensible pour de petites disparités, ce score trop sensible ne permet pas vraiment d'évaluer l'ordre des blobs. Pour évaluer l'ordre des blobs, il suffirait de générer des valeurs témoins contenant des informations sur la distance relative entre les acteurs et les caméras. Enfin, les grandes disparités signifieraient que les acteurs correspondants seraient près des caméras.

RÉFÉRENCES

Bai, X., Latecki, L. J., & Liu, W.-Y. (2007). Skeleton Pruning by Contour Partitioning with Discrete Curve Evolution. *Pattern Analysis and Machine Intelligence, IEEE Transactions on, 29*(3), 449-462.

Bay, H., Tuytelaars, T., & Van Gool, L. (2006). SURF: Speeded Up Robust Features. In *Computer Vision – ECCV 2006* (pp. 404-417).

Borgefors, G. (1986). Distance transformations in digital images. *Computer Vision, Graphics, and Image Processing, 34*(3), 344-371.

Bourezak, R., & Bilodeau, G. (2006). Object detection and tracking using iterative division and correlograms. (pp. 38-38).

Brown, M., & Lowe, D. G. (2002). Invariant Features from Interest Point Groups. *British Machine Vision Conference, BMVC 2002* (pp. 656-665).

Cheriton, D., & Tarjan, R. E. (1976). Finding Minimum Spanning Trees. *SIAM Journal on Computing, 5*(4), 724-742.

Douglas, D. H., & Peucker, T. K. (1973). Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization, 10*(2), 112-122.

Fischler, M. A., & Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM, 24*(6), 381-395.

Fisher, R. B. (2002). The RANSAC (Random Sample Consensus) Algorithm. Consulté le 10 octobre 2007, tiré de Google, 10 octobre 2007, http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/FISHER/RANSAC/.

Hajebi, K., & Zelek, J. S. (2006). Sparse Disparity Map from Uncalibrated Infrared Stereo Images. *Proceedings of the 3rd Canadian Conference on Computer and Robot Vision (CRV'06)* (pp. 17-17).

Hajebi, K., & Zelek, J. S. (2007). Dense Surface from Infrared Stereo. *Workshop on Applications of Computer Vision (WACV07)* (pp. 21-21).

Harris, C., & Stephens, M. (1988). A Combined Corner and Edge Detector. *Fourth Alvey Vision Conference, Manchester, UK* (pp. 147-151).

Hartley, R., & Zisserman, A. (2003). *Multiple view geometry in computer vision* (2e éd.). Cambridge, UK: Cambridge University Press.

Intel. (2006). Open Source Computer Vision Library. Consulté le 17 septembre 2007, tiré de Google, 2 août 2007, http://www.intel.com/technology/computing/opencv/index.htm.

Kadir, T., & Brady, M. (2001). Saliency, Scale and Image Description. *International Journal of Computer Vision, 45*(2), 83-105.

Ke, Y., & Sukthankar, R. (2004). PCA-SIFT: a more distinctive representation for local image descriptors. *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on* (Vol. 2, pp. II-506-II-513 Vol.502).

Kovesi, P. (2000a). MATLAB and Octave Functions for Computer Vision and Image Processing. Consulté le 24 septembre 2007, tiré de http://www.csse.uwa.edu.au/~pk/Research/MatlabFns/index.html.

Kovesi, P. (2000b). Phase congruency: A low-level image invariant. *Psychological Research, 64*(2), 136-148.

Kovesi, P. (2003). Phase Congruency Detects Corners and Edges. *DICTA 2003, Sydney* (pp. 309-318)University of Queensland.

Kovesi, P. (2007). What Are Log-Gabor Filters and Why Are They Good? Consulté le Google, http://www.csse.uwa.edu.au/~pk/Research/MatlabFns/PhaseCongruency/Docs/convexpl.html.

Latecki, L. J., & Lakamper, R. (1999). *Polygon Evolution by Vertex Deletion.*

Lowe, D. G. (1999). Object recognition from local scale-invariant features. (Vol. 2, pp. 1150-1157 vol.1152).

Lowe, D. G. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision, V60*(2), 91-110.

Lucas, B. D., & Kanade, T. (1981). An Iterative Image Registration Technique with an Application to Stereo Vision. *International Joint Conference on Artificial Intelligence, Vancouver, BC, Can* (Vol. 2, pp. 674-679).

McKenna, S. J., Jabri, S., Duric, Z., Rosenfeld, A., & Wechsler, H. (2000). Tracking Groups of People. *Computer Vision and Image Understanding, 80*(1), 42-56.

Mikolajczyk, K., & Schmid, C. (2005). A performance evaluation of local descriptors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on, 27*(10), 1615-1630.

Moravec, H. P. (1980). *Obstacle avoidance and navigation in the real world by a seeing robot rover.* Technical Report inédit, Robotics Institute, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

Rosten, E., & Drummond, T. (2005). Fusing points and lines for high performance tracking. *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on* (Vol. 2, pp. 1508-1515 Vol. 1502).

Rosten, E., & Drummond, T. (2006). Machine Learning for High-Speed Corner Detection. In *Computer Vision – ECCV 2006* (pp. 430-443).

Shah, S., Aggarwal, J. K., Eledath, J., & Ghosh, J. (1997). Multisensor integration for scene classification: an experiment in human form detection. (Vol. 2, pp. 199-202 vol.192).

Shi, J., & Tomasi, C. (1994). Good features to track. *CVPR 1994* (pp. 593-600).

Shoushtarian, B., & Bez, H. E. (2005). A practical adaptive approach for dynamic background subtraction using an invariant colour model and object tracking. *Pattern Recognition Letters, 26*(1), 5-26.

Smith, S. M., & Brady, J. M. (1997). SUSAN - A New Approach to Low Level Image Processing. *International Journal of Computer Vision, 23*(1), 45-78.

Stauffer, C., & Grimson, W. E. L. (1999). Adaptive background mixture models for real-time tracking. *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.* (Vol. 2, pp. 252 Vol. 252).

Tomasi, C., & Kanade, T. (1991). *Detection and Tracking of Point Features* (CMU-CS-91-132): Carnegie Mellon University.

Trucco, E., & Verri, A. (1998). *Introductory techniques for 3-D computer vision* (Prentice Hall[e] éd.). Upper Saddle River, NJ.

Vedaldi, A. (2006). An open implementation of SIFT. *SIFT.* Consulté le 11 octobre 2007, tiré de Google, http://vision.ucla.edu/~vedaldi/code/sift/sift.html.

Wiskott, L., Fellous, J. M., Kruger, N., & von der Malsburg, C. (1997). Face recognition by elastic bunch graph matching. *Image Processing, 1997. Proceedings., International Conference on* (Vol. 1, pp. 129-132 vol.121).

Zitnick, C. L., & Kanade, T. (2000). A cooperative algorithm for stereo matching and occlusion detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 22*(7), 675-684.

ANNEXE I FONCTION cvFindContours()

Après avoir effectué la soustraction de l'arrière-plan pour chaque image à un instant donné (section 2.2.1), nous avons déjà la liste de tous les pixels faisant partie de l'avant-plan. Ceux-ci peuvent être dessinés en blanc sur une image temporaire initialement remplie de noir. Dans le cas de l'algorithme du squelette (section 2.3), il n'en faut pas plus pour continuer. Par contre, dans le cas du processus DCE (section 2.4), il nous faut le détail du contour de chaque blob dans chaque image.

La bibliothèque OpenCV peut alors nous venir en aide grâce à sa fonction *cvFindContours()* qui permet de trouver tous les contours dans une image contenant des blobs blancs sur un fond noir. Les paramètres que nous avons utilisés pour cette fonction permettent de trouver la position de tous les sommets du contour de chaque blob et de chaque trou des blobs. Un contour est constitué de quatre types d'arêtes : horizontal, vertical et les deux diagonales (figure I.1).

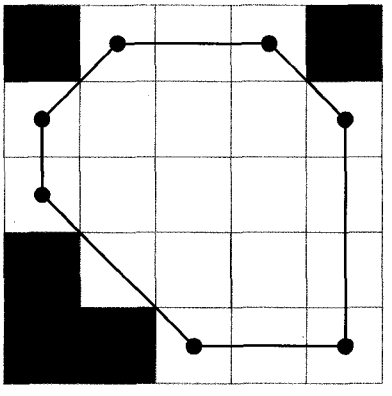

Figure I.1 Un exemple de contour retourné par cvFindContours() pour un blob blanc sur un fond noir. Le contour est constitué de sommets (les gros points noirs) et d'arêtes (selon quatre orientations différentes)

Par la suite, il est possible d'extraire tous les sommets sur le contour de chaque blob ou de chaque trou dans un blob. C'est avec ces sommets que nous pouvons exécuter le processus DCE.

ANNEXE II ALGORITHME DOUGLAS-PEUCKER

Afin d'approximer un contour constitué d'une suite de sommets liés par des arêtes, on peut utiliser l'algorithme Douglas-Peucker avec le paramètre d'erreur d'approximation ε. Pour comprendre l'algorithme, nous présentons l'exemple de la figure II.1. Dans cette figure, le demi-contour ABCDEF est initialement approximé par le segment AF. Pour que ce segment soit retenu, il faut que tous les sommets de A à F soit à une distance inférieure ou égale à ε du segment AF. Ce n'est pas le cas, alors on sépare le segment AF avec le sommet le plus éloigné du segment AF : D. Ainsi, on obtient les segments finaux AD et DF, car les sommets B et C sont suffisamment près du segment AD et le sommet E est suffisamment près du segment DF. Autrement, il faudrait itérer davantage pour séparer AD ou DF.

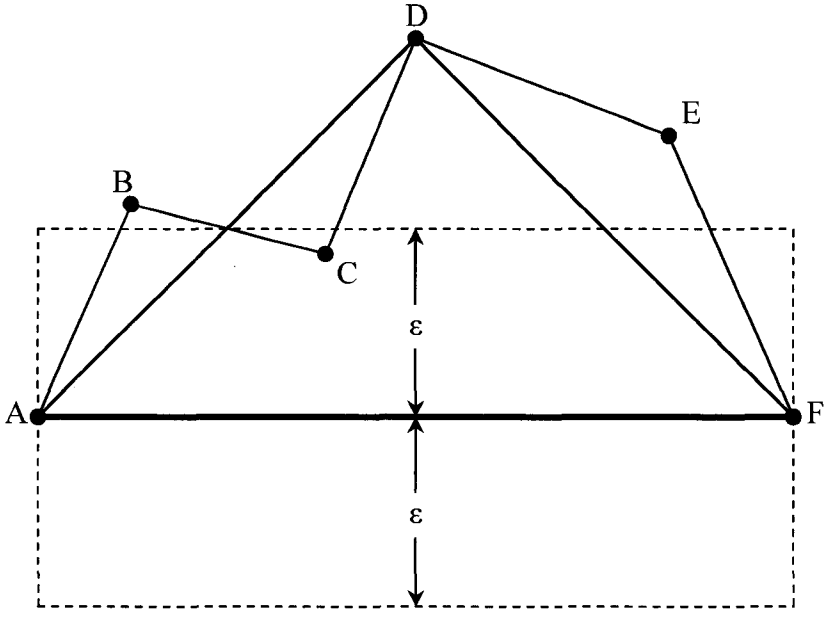

Figure II.1 Portion de contour ABCDEF approximé par le contour ADF en utilisant l'algorithme Douglas-Peucker avec le paramètre ε

Dans le cas d'un contour complet, on peut le diviser en deux pour avoir deux suites de segments à approximer.