Recalage de deux nuages de points 3D

Registration of Free-Form Surfaces Represented by Point Sets

par Zhengyou ZHANG
INRIA Sophia-Antipolis, 2004 route des Lucioles
B.P. 93, 06902 Sophia-Antipolis Cedex

Résumé

Une méthode a été développée pour recaler deux nuages de points 3D obtenus en utilisant la stéréo par corrélation. Le recalage de deux ensembles de primitives géométriques est un problème en général très difficile et non résolu. Heureusement, dans beaucoup d'applications, des connaissances a priori simplifient considérablement le problème. Par exemple, le mouvement entre deux positions successives est généralement soit petit soit approximativement connu. A partir de cette estimée grossière, notre algorithme permet de calculer le mouvement avec une très bonne précision, nécessaire à l'obtention d'un modèle satisfaisant de l'environnement. Les objets observés sont représentés au moyen de nuages de points 3D. Ces points sont considérés comme des échantillons d'une surface. Aucune contrainte n'est a priori imposée sur la forme des objets. L'algorithme proposé est basé sur une mise en correspondance itérative des points d'une vue avec leurs plus proches voisins dans l'autre vue. Une méthode statistique basée sur la distribution de distances est utilisée pour éliminer les appariements aberrants. Une technique de moindres carrés est utilisée pour estimer le mouvement 3D à partir des correspondances de points. L'application de ce mouvement réduit la distance moyenne entre les surfaces dans les deux ensembles. Des données réelles ont été utilisées pour tester cet algorithme. Les résultats montrent qu'il est efficace et robuste, et qu'il donne une estimation précise du mouvement.

Mots clés : Mise en correspondance de surfaces, Recalage 3D, Estimation du mouvement, Analyse de scène dynamique, Vision pour la robotique.

Abstract

A method has been developed for registering two dense 3-D maps obtained by using a correlation-based stereo system. Geometric matching in general is a difficult unsolved problem in computer vision. Fortunately, in many practical applications, some a priori knowledge exists which considerably simplifies the problem. In visual navigation, for example, the motion between successive positions is usually either small or approximately known. From this initial estimate, our algorithm can compute the motion with very good precision, which is required for environment modeling. Objects are represented by a set of 3-D points, which are considered as the samples of a surface. No constraint is imposed on the form of the objects. The proposed algorithm is based on iteratively matching points of one view to the closest points of the another view. A statistical method based on the distance distribution is used to discard the outliers. A least-squares technique is used to estimate 3-D motion from the point correspondences, which reduces the average distance between points in the two sets. Real data have been used to test the algorithm. The results show that it is efficient and robust, and yields an accurate motion estimate.

Key words : *Free-Form Surface Matching, 3-D Registration, Motion Estimation, Dynamic Scene Analysis, Robot Vision*

1. Introduction

Le travail présenté dans cet article s'inscrit dans le contexte de la navigation d'un véhicule autonome planétaire (VAP) utilisant la vision. Au cours d'une mission typique, le VAP sera appelé à se déplacer de manière autonome entre deux points spécifiés par les contrôleurs de la mission. En chemin il lui faudra déterminer l'itinéraire le plus sûr, éviter les obstacles et vérifier sa position à l'aide des amers qu'il rencontrera. S'il doit prélever des échantillons, il aura aussi à modéliser la configuration des roches de manière à pouvoir planifier ses actions. Pour ce faire, il lui faudra percevoir et modéliser son environnement à l'aide de ses divers capteurs, caméras et télémètre laser entre autres, de la carte de terrain à faible résolution transmise par l'orbiteur et des informations fournies par son propre système de navigation. Au cours de sa traversée du terrain, le véhicule sera à même d'accumuler les mesures, donc de construire son modèle de manière incrémentale en combinant les diverses sources d'informations. C'est là l'une des données fondamentales du problème car il est illusoire d'attendre d'un capteur unique ou d'un seul algorithme une interprétation sans ambiguïté d'une scène aussi complexe qu'un paysage d'extérieur avec ses objets aux formes complexes et imprévisibles. De même une seule vue d'une scène s'avère souvent insuffisante, il est préférable d'en combiner plusieurs pour produire des interprétations crédibles.

L'objectif de cette étude est de calculer d'une manière très précise le déplacement entre des vues successives afin de recaler différentes cartes de profondeurs denses. Une carte de profondeur dense est soit acquise par un capteur actif (e.g., ERIM [22], ALIS (CEA/LETI, Grenoble)), soit reconstruite par un système stéréoscopique basé sur la corrélation [6], soit obtenue en fusionnant les deux [27]. Cette étape est indispensable pour les raisons suivantes : mieux localiser le VAP, éliminer des erreurs dans la reconstruction, construire un modèle numérique de terrain (MNT) plus global. C'est un problème en général très difficile. Nous avons étudié l'état de l'art, et il n'existe pas encore d'algorithme robuste. Pour le VAP, le problème se simplifie en récupérant des informations du déplacement provenant d'autres instruments à bord. Nous allons décrire un algorithme basé sur la minimisation de la distance entre deux surfaces.

Cet article est organisé de la manière suivante :

- La section 2 présente un bref descriptif des méthodes proposées dans la littérature, et justifie le choix de l'approche iconique.
- La section 3 décrit l'algorithme que nous avons développé pour résoudre le problème du recalage pour le VAP, et compare cet algorithme avec un algorithme similaire dans la littérature.
- La section 4 met en évidence quelques aspects importants pour implémenter cet algorithme.
- La section 5 présente les résultats de notre algorithme sur des données réelles.
- La section 6 soulève des discussions sur les extensions possibles de l'algorithme.
- La section 7 conclut cet article.

2. L'étude du problème

Nous avons établi l'état de l'art des méthodes proposées dans la littérature pour recaler deux cartes de profondeur. Nous pouvons diviser les méthodes en deux catégories.

- **Approche basée sur des primitives**. Les primitives utilisées sont des points de forte courbure ou des courbes invariantes par transformation. Une carte de profondeur peut être décrite par un graphe avec des primitives définissant des noeuds du graphe et des relations géométriques définissant des arêtes. Le recalage de deux cartes de profondeur revient à mettre en correspondance deux graphes : *isomorphisme de sous-graphes*. Des heuristiques sont souvent introduites pour réduire la complexité.
- **Approche basée sur l'ensemble des données**. Une carte de profondeur est considérée comme une surface, ayant la forme (*Monge patch*)

$$\mathbf{x}(x,y) = [x,\ y,\ z(x,y)]^T \quad \text{avec}(\mathrm{x},\mathrm{y}) \in \mathbb{R}^2 \ .$$

  L'idée est de trouver la transformation par la minimisation d'un critère lié à la distance entre deux surfaces.

Dans l'approche basée sur des primitives, on utilise souvent des propriétés différentielles invariantes par une transformation rigide comme, par exemple, la courbure gaussienne. Les primitives couramment utilisées sont des :

1. points particuliers [8, 9, 15], ceux qui ont une courbure localement maximale dont la valeur absolue est supérieure à un seuil.
2. contours. Un contour peut être l'endroit où l'élévation change brusquement, dit *contour de falaise* [21]. Il peut aussi être un *profil de distance* [20], c'est-à-dire une courbe dont les points sont situés à égale distance d'un point donné. Dans certains cas spécifiques, un contour peut aussi être une courbe de profondeur constante [12], ou simplement une courbe de niveau.
3. morceaux de surface [13, 16]. Chaque morceau de surface est classé en différentes catégories selon le signe des courbures gaussienne et moyenne. Ce type de primitives est plutôt utilisé dans une scène limitée, par exemple, une scène comportant quelques objets à reconnaître. Dans une scène naturelle, il y aura beaucoup de morceaux de surface de sorte que la mise en correspondance deviendra impraticable.

Parmi les méthodes basées sur l'ensemble des données, nous trouvons

1. une technique similaire à la corrélation [7], applicable quand le nombre de degrés de liberté de la transformation entre deux vues est réduit (2, par exemple).
2. une technique différentielle [11], applicable quand le mouvement entre deux vues est très petit ou bien quand on a déjà une très bonne estimée du mouvement.
3. une technique basée sur la cohérence ou la compatibilité entre deux vues [9, 15] (quantifiée par la distance entre deux surfaces si on considère que les points sont des échantillons d'une surface). Szeliski [23] propose une technique comprenant une autre contrainte, la contrainte de lissage.

La différence essentielle entre l'approche basée sur l'ensemble des données et celle basée sur des primitives réside sur les informations à traiter durant le recalage. Les informations utilisées dans l'approche basée sur des primitives sont beaucoup plus concises que dans l'approche basée sur l'ensemble des données. L'approche basée sur des primitives est donc en général préférable. Mais dans un environnement naturel tel que celui du VAP, nous rencontrons souvent des surfaces lisses. De plus, la carte visuelle dont nous disposons est très bruitée. Nos expériences sur des méthodes différentielles dans le passé montrent qu'il est très difficile de localiser précisément des points ayant de fortes courbures (ou des extrêma de courbure). Une faible variation dans une courbe ou une surface peut produire un nombre différent d'extrêma de courbure et des positions différentes. La mise en correspondance basée sur des extrêma de courbure est très sensible au bruit. Nous pouvons trouver dans la littérature [26, 18] des arguments contre leur utilisation dans un problème de mise en correspondance. D'autre part, bien que les points de forte courbure soient, d'après la théorie de la géométrie différentielle, invariants par transformation rigide, les méthodes pour détecter ces points dépendent en général du point de vue, et donc les points détectés ne sont plus invariants. De plus, une carte visuelle est très locale (e.g., normalement seulement une partie d'un rocher est observable). Les points particuliers détectés dans deux cartes correspondant au même rocher n'ont probablement rien en commun. En un mot, avec l'état de l'art des méthodes développées et considérant l'environnement rencontré par le VAP, nous ne pouvons pas détecter d'une manière

très robuste et localiser d'une manière très précise des points particuliers. Mais sans aucun doute, en utilisant des points particuliers, nous pouvons calculer une estimation du déplacement proche de la réalité, qui peut servir, comme une estimation initiale, à d'autres approches plus précises.

L'utilisation de contours de falaise ou de profils de distance conduit au même problème que celle de points particuliers, puisque leurs représentations dépendent fortement de la localisation des points particuliers choisis. Des courbes de profondeur constante ou des courbes de niveau sont utiles seulement dans certains cas spécifiques. Par exemple, pour recaler deux cartes visuelles avec des courbes de niveau, nous devons représenter les courbes par rapport à un horizon commun aux deux cartes. Sinon, les courbes de niveau ne sont plus significatives. Donc cette approche est applicable si le VAP se déplace dans un terrain complètement plat.

L'approche basée sur l'ensemble des données utilise toutes les informations disponibles. Elle consiste à trouver le déplacement en minimisant la différence entre deux cartes (la distance entre deux surfaces si l'on considère une carte visuelle comme une surface). S'il y a n points en recouvrement, nous avons $3n$ équations à 6 inconnues (3 pour la rotation et 3 pour la translation). Nous avons donc un système sur-déterminé à résoudre. La redondance importante nous permet de trouver une estimation précise du déplacement, à condition bien sûr que nous disposions d'une estimation initiale avec une précision raisonnable. Or le VAP peut nous fournir effectivement une telle estimation avec ses moyens embarqués. C'est pourquoi nous avons choisi cette approche.

La méthode décrite par la suite est similaire à la troisième technique de l'approche basée sur l'ensemble des données. Un travail similaire a été effectué indépendamment par Besl et McKay [3]. Une comparaison sera donnée dans la section 3.6.

3. La description de l'algorithme

Dans cette section, nous décrivons les étapes principales de l'algorithme que nous avons développé. Cet algorithme est une procédure itérative. Il prend en entrée deux ensembles de points 3D (MNTs ou cartes visuelles construites par la stéréovision par corrélation). En sortie, il fournit une estimation optimale de la transformation entre les deux ensembles.

Un ensemble de points 3D peut être considéré comme un ensemble d'échantillons d'une surface, représentant les objets observés par le VAP. Nous utilisons directement ces points 3D, et aucune contrainte n'est a priori imposée sur la forme des objets. Ceci est bien approprié pour le VAP qui doit accomplir des missions dans des environnements non-structurés. Par la suite, les points 3D acquis dans la première position sont notés $\mathbf{x}_i$ $(i = 1, \ldots, m)$, et les points 3D acquis dans la deuxième position sont notés $\mathbf{x}'_j$ $(j = 1, \ldots, n)$.

Formellement, si S' est la surface de la deuxième vue dont les échantillons disponibles sont $\mathbf{x}'_j$ $(j = 1, \ldots, n)$, nous cherchons une transformation $(\mathbf{R}, \mathbf{t})$ qui minimise la fonction suivante :

$$\mathcal{F}(\mathbf{R}, \mathbf{t}) = \frac{1}{\sum_{i=1}^{m} p_i} \sum_{i=1}^{m} p_i d^2(\mathbf{R}\mathbf{x}_i + \mathbf{t}, S') \ , \qquad (1)$$

où $d(\mathbf{x}, S')$ est la distance euclidienne du point $\mathbf{x}$ à la surface S', et p_i prend la valeur 1 si $\mathbf{x}_i$ peut être apparié avec un point sur S' et 0 sinon.

Cependant, la minimisation de $\mathcal{F}(\mathbf{R}, \mathbf{t})$ est extrêmement difficile, non seulement parce que $d(\mathbf{R}\mathbf{x}_i + \mathbf{t}, S')$ est très non-linéaire (le point sur S' correspondant à $\mathbf{x}_i$ n'est pas connu à l'avance) mais encore parce que p_i peut prendre la valeur soit 1 soit 0 (un problème de programmation en nombre entier). Puisque nous disposons d'une estimée raisonnable du deplacement entre deux vues, le problème se simplifie. Si l'on applique cette estimée à la première vue, la vue transformée diffère seulement un peu de la deuxième vue. Alors il est évident que le correspondant d'un point de la vue transformée doit être proche de celui-ci. L'algorithme décrit ci-dessous essaie d'apparier les points de la première vue $(\mathbf{x}_i)$, après avoir appliqué l'estimée du déplacement $(\mathbf{R}, \mathbf{t})$ calculée précédemment, avec leurs plus proches points de la deuxième vue. Une technique de moindres carrés est utilisée pour calculer le déplacement à partir de ces appariements. Ceci revient à minimiser la distance moyenne entre les surfaces dans les deux vues. Comme un point de la première vue et son voisin de la deuxième vue ne correspondent pas nécessairement à un point physique dans l'espace, quelques itérations sont indispensables.

3.1. TROUVER DES POINTS LES PLUS PROCHES

Etant donné un point $\mathbf{x}$ de la première vue, nous cherchons son voisin le plus proche dans la deuxième vue, noté $\mathbf{y}$. Par définition, nous avons

$$d(\mathbf{x}, \mathbf{y}) = \min_{j \in \{1, \ldots, n\}} d(\mathbf{x}, \mathbf{x}'_j) \ ,$$

où $d(\mathbf{x}_1, \mathbf{x}_2)$ est la distance euclidienne entre les deux points $\mathbf{x}_1$ et $\mathbf{x}_2$, i.e., $d(\mathbf{x}_1, \mathbf{x}_2) = \|\mathbf{x}_1 - \mathbf{x}_2\|$. Nous avons donc approximé la distance $d(\mathbf{x}, S')$ dans l'équation 1 par la distance ci-dessus.

Le coût dans le pire cas pour trouver le point le plus proche est $O(n)$, où n est le nombre total des points dans la deuxième vue. Si on fait la même chose pour chaque point dans la première vue, le coût total est $O(mn)$, où m est le nombre total des points dans la première vue. L'utilisation de la technique dite *k-D trees* permet d'accélérer considérablement le processus (voir Sect. 4.1).

3.2. METTRE EN CORRESPONDANCE DES POINTS

Pour chaque point $\mathbf{x}$ on peut toujours trouver le point le plus proche $\mathbf{y}$. Pourtant, il y a des points erronés dans les deux vues à cause des erreurs des capteurs, ou bien des points visibles dans une vue et non visibles dans l'autre vue à cause du déplacement des capteurs ou des objets. Il peut n'avoir aucun sens d'apparier simplement $\mathbf{x}$ avec $\mathbf{y}$. Plusieurs contraintes peuvent être imposées pour éliminer des appariements erronés. Par exemple, la continuité

des distances dans un voisinage est très utile pour écarter des faux appariements. Ces contraintes ne sont pas utilisées dans notre algorithme pour maintenir celui-ci dans la forme la plus simple possible. Par contre, nous pouvons imposer les deux contraintes suivantes, qui sont très simples.

La première est la tolérance maximale pour la distance. Si la distance entre un point $\mathbf{x}_i$ et son plus proche point $\mathbf{y}_i$, notée $d(\mathbf{x}_i, \mathbf{y}_i)$, est supérieure à la distance maximale tolérable $D_{\max}$, cet appariement sera éliminé, c'est-à-dire, il n'existe pas un point dans la deuxième vue qui peut être apparié d'une manière raisonnable avec le point $\mathbf{x}_i$. Cette contrainte peut être facilement justifiée parce qu'on sait que le mouvement entre les deux vues est petit, et donc la distance entre deux points qui peuvent raisonnablement être appariés ne doit pas être très grande. Dans notre algorithme, $D_{\max}$ est calculée d'une manière adaptative et robuste pendant chaque itération en analysant la statistique des distances (voir Sect. 3.3).

La deuxième est la cohérence d'orientation. Nous pouvons estimer la normale à la surface en chaque point. Il est facile de prouver que l'angle entre la normale en un point $\mathbf{x}$ et celle en son plus proche point $\mathbf{y}$ ne peut pas être supérieure à l'angle de rotation entre les deux vues [32]. Nous pouvons donc imposer que l'angle entre les normales des points appariés ne doit pas être supérieur à une valeur préfixée Θ, qui est le maximum espéré de l'angle de rotation entre deux vues. Dans notre implémentation, nous n'avons pas intégré cette contrainte, parce que le calcul de la normale à partir d'un nuage de points 3D est un peu coûteux.

3.3. METTRE À JOUR LES APPARIEMENTS

Au lieu d'utiliser tous les appariements trouvés, nous exploitons une technique robuste pour en écarter quelques-uns en analysant la statistique des distances. Pour ce faire, un paramètre, noté $\mathfrak{D}$, devant être fourni par l'utilisateur, indique que le recalage entre les deux vues est bon. Voir la section 4.3 pour le choix de la valeur $\mathfrak{D}$.

Soit $D^{\mathrm{I}}_{\max}$ la distance maximale tolérable dans l'itération I. A cet instant, chaque point de la première vue (après avoir appliqué le mouvement estimé précédemment) dont la distance à son plus proche point est inférieure à $D^{\mathrm{I}-1}_{\max}$ est retenu, en même temps que son plus proche point et leur distance. Soient $\{\mathbf{x}_i\}$, $\{\mathbf{y}_i\}$, et $\{d_i\}$ respectivement les ensembles de points originaux, points les plus proches, et leurs distances. Soit N le nombre d'éléments dans ces ensembles. Nous pouvons maintenant calculer la moyenne μ et l'écart type σ de ces distances, qui sont donnés par

$$\mu = \frac{1}{N}\sum_{i=1}^{N} d_i \ ,$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(d_i - \mu)^2} \ .$$

Selon la valeur de μ, nous modifions $D^{\mathrm{I}}_{\max}$ d'une manière adaptative comme montré ci-dessous [1] :

```
if μ < 𝔇                    /* le recalage est très bon */
   D^I_max = μ + 3σ;
else if μ < 3𝔇               /* le recalage est encore bon */
   D^I_max = μ + 2σ;
else if μ < 6𝔇               /* le recalage n'est pas /*
   D^I_max = μ + σ;             /* trop mauvais */
else                         /* le recalage est vraiment /*
   D^I_max = ξ;                 /* très mauvais */
endif
```

Nous retardons l'explication du symbole ξ (voir Sect. 4.4).

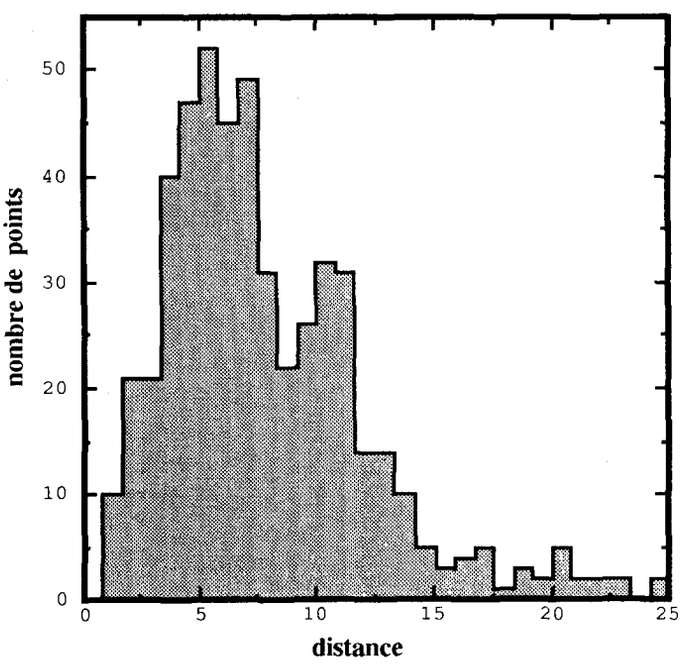

Figure 1. – Un histogramme de distances.

Maintenant, nous pouvons mettre à jour les appariements trouvés précédemment en utilisant la nouvelle valeur $D^{\mathrm{I}}_{\max}$: un appariement entre $\mathbf{x}_i$ et $\mathbf{y}_i$ est enlevé si leur distance d_i est supérieure à $D^{\mathrm{I}}_{\max}$. Les appariements qui restent sont utilisés pour calculer le mouvement entre les deux vues (Sect. 3.4).

Parce que $D_{\max}$ est modifiée adaptivement selon la statistique des distances, notre algorithme est robuste aux grosses erreurs dans les mesures. Même si quelques faux appariements restent après la mise à jour, l'utilisation d'une technique de moindres carrés permet d'obtenir une estimée raisonnable du mouvement, qui est

[1] Ici la distribution des distances est supposée être approximativement gaussienne quand le recalage entre les deux vues est bon. Ceci a été confirmé par des expériences. Un histogramme représentatif est montré dans la figure 1. Plus strictement, la distribution χ est une meilleure approximation. Mais il est bien connu en statistique que la distribution peut être bien approximée par une gaussienne si le nombre d'échantillons est grand, e.g., supérieur à 30 (théorème de la limite centrale). Or, nous avons un nombre de correspondances largement supérieur.

suffisante pour que notre algorithme converge vers la solution correcte.

3.4. CALCULER LE MOUVEMENT

Nous avons maintenant un ensemble de points 3D dans la première vue qui ont été raisonnablement appariés avec un ensemble de points les plus proches, notés respectivement $\{\mathbf{x}_i\}$ et $\{\mathbf{y}_i\}$. Soit N le nombre de paires. Parce que N est souvent beaucoup plus grand que 3 (nombre minimal d'appariements pour calculer un mouvement rigide et unique), il est nécessaire de concevoir une procédure pour calculer le mouvement en minimisant la fonction suivante (d'après l'équation 1) :

$$\mathcal{F}(\mathbf{R},\mathbf{t}) = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{R}\mathbf{x}_i + \mathbf{t} - \mathbf{y}_i\|^2 , \qquad (2)$$

qui est en fait la moyenne des distances au carré. Toute méthode d'optimisation comme *gradient à descente maximum, gradient conjugué*, ou *simplex*, peut être utilisée pour trouver la rotation et la translation au sens des moindres carrés. Heureusement, il existe des techniques beaucoup plus efficaces pour résoudre ce problème particulier, y compris la méthode du quaternion [5, 10], la méthode du quaternion dual [25], et celle de la décomposition en valeurs singulières [1]. Nous avons implémenté les méthodes du quaternion et du quaternion dual. Elles donnent exactement la même estimée du mouvement.

3.5. RÉSUMÉ DE L'ALGORITHME

Nous pouvons maintenant décrire notre algorithme plus formellement :

- **entrée :** Deux vues 3D contenant m et n points 3D, respectivement.
- **sortie :** Le mouvement optimal entre les deux vues.
- **procédure :**

  a) **initialisation**

  $D^0_{\max}$ est initialisée à $20\mathfrak{D}$, ce qui sous-entend que tout point de la première vue dont la distance à son plus proche point dans la deuxième vue est supérieure à $D^0_{\max}$ est écarté lors de la première itération. Le chiffre *20* n'est pas crucial dans l'algorithme, et peut être remplacé pas une valeur plus grande.

  b) **prétraitement**

  Construire la représentation de l'arbre k-D de la deuxième vue (voir la section 4.1).

  c) **itération** jusqu'à la convergence du mouvement calculé

  (i) Trouver les plus proches points satisfaisant la contrainte de distance, comme décrit dans la section 3.2.

  (ii) Mettre à jour les appariements à l'aide de l'analyse statistique des distances, comme décrit dans la section 3.3.

  (iii) Calculer le mouvement entre les deux vues à partir des appariements retenus, comme décrit dans la section 3.4.

  (iv) Appliquer le mouvement à tous les points de la première vue.

On doit faire quelques remarques ici. La construction et l'utilisation de l'arbre k-D pour trouver les plus proches points seront décrites dans la section 4.1. Le mouvement est calculé entre les points *originaux* de la première vue et les points de la deuxième vue. Donc, l'estimée finale donnée par l'algorithme représente la transformation entre la première vue originale et la deuxième vue.

Nous pouvons montrer que l'algorithme présenté converge vers le minimum local tout en étant borné par une courbe supérieure. En pratique, nous observons une convergence monotone si les données sont assez précises. Sinon, l'algorithme peut osciller autour du minimum local.

La condition de terminaison d'itération est définie comme la variation de l'estimée du mouvement entre deux itérations successives. La variation en translation après l'itération $\imath$ est définie comme

$$\delta\mathbf{t} = \frac{\|\mathbf{t}_\imath - \mathbf{t}_{\imath-1}\|}{\|\mathbf{t}_\imath\|} .$$

Pour mesurer la variation en rotation, nous utilisons la représentation de l'axe de rotation, qui est un vecteur 3D, noté $\mathbf{r}$. La norme de $\mathbf{r}$ donne l'angle de rotation, et $\mathbf{r}$ est parallèle à l'axe de rotation. Nous définissons donc la variation en rotation après l'itération $\imath$ comme

$$\delta\mathbf{r} = \frac{\|\mathbf{r}_\imath - \mathbf{r}_{\imath-1}\|}{\|\mathbf{r}_\imath\|} .$$

Nous terminons l'itération quand $\delta\mathbf{r}$ et $\delta\mathbf{t}$ sont toutes les deux inférieures à 1%, ou bien quand le nombre d'itérations atteint un seuil préfixé (50, par exemple).

3.6. COMPARAISON AVEC D'AUTRES TRAVAUX

Au cours du développement de cet algorithme, un travail indépendant par Besl et McKay [3] a été publié. Ils utilisent la même idée : apparier itérativement des points d'une vue avec leurs plus proches points de l'autre vue. La différence principale est sur le critère de minimisation. Se référer à l'équation 1. Dans l'algorithme de Besl et McKay, p_i prend toujours la valeur 1. Par contre, dans notre algorithme, p_i peut prendre la valeur soit 1 soit 0 selon que le point de la première vue a un appariement raisonnable dans la deuxième vue ou non. L'appariement est déterminé par la distance maximale tolérable $D_{\max}$, qui, à son tour, est déterminée d'une manière dynamique en analysant la statistique des distances comme décrit dans la section 3.3. Donc, notre algorithme est capable de traiter les situations suivantes :

- Erreurs importantes dans les données. Les points correspondants sont automatiquement écartés pendant la mise en correspondance, et donc n'ont pas d'effet sur l'estimation finale.
- Apparition ou disparition des points. Une partie des points d'un point de vue n'est pas visible dans l'autre. C'est souvent le cas

pour la navigation où des objets peuvent entrer dans le champ de vue ou le quitter.

- Occlusion. Un objet peut cacher d'autres objets, et lui-même peut être caché. C'est un problème fréquent dans la reconnaissance d'objets et dans la navigation.

Comme p_i prend toujours la valeur 1 dans l'algorithme de Besl et McKay, leur algorithme peut seulement traiter le cas où la première vue est un sous-ensemble de la deuxième. Il est incapable de traiter les situations ci-dessus. On peut souligner d'ailleurs que les résultats de Besl et McKay sont obtenus sur des exemples relativement artificiels[2], alors que cet article contient de nombreux résultats sur des données stéréoscopiques réelles.

4. Quelques considérations pratiques

Dans cette section, nous étudions quelques aspects pratiques importants, comprenant la recherche des points les plus proches, la stratégie du plus grossier au plus fin, le choix du paramètre $\mathfrak{D}$, et le cas où l'estimée initiale est très mauvaise.

4.1. LA RECHERCHE DES POINTS LES PLUS PROCHES

Comme on a pu constater dans la section précédente, la recherche du point le plus proche d'un point donné est de complexité $O(n)$ en temps, où n est le nombre total des points de la deuxième vue. Plusieurs méthodes existent pour accélérer la recherche, y compris la technique de la baquétisation et la technique de l'arbre binaire (*k-D trees* en anglais, abréviation pour *k-dimensional binary search tree*). Nous avons choisi la technique de l'arbre binaire, parce que les points sont clairsemés dans l'espace 3D. Il n'est pas très efficace d'utiliser la technique de la baquétisation en 3D parce que seulement quelques baquets contiennent beaucoup de points[3].

L'arbre binaire à k dimensions est une généralisation de bissection en une dimension à k dimensions [19]. Dans notre cas, $k = 3$. Un arbre 3D est construit comme suit. D'abord, choisir un plan parallèle au plan yz passant par un point P pour diviser l'espace total en deux parallélépipèdes rectangulaires (généralisés)[4] tel qu'il y ait le même nombre de points de chaque côté. Nous obtenons donc un fils gauche et un fils droit. Ensuite, chaque fils est encore divisé par un plan parallèle au plan xz tel qu'il y ait le même nombre de points de chaque côté de la coupe, et nous obtenons un petit-fils gauche et un petit-fils droit. Nous avons au total quatre petits-fils. Nous continuons à diviser chaque petit-fils en choisissant un plan parallèle au plan xy, et ainsi de suite, en faisant alterner à chaque étape la direction du plan de coupe entre yz, xz et xy. Le processus de division se termine quand on arrive à un parallélépipède qui ne contient aucun point; le nœud correspondant est la feuille de cet arbre. Un arbre binaire peut être construit en temps $O(n \log n)$ et en stockage $O(n)$, qui sont tous les deux optimaux [19].

L'utilisation standard de l'arbre binaire est de trouver tous les points dont les distances à un point donné sont toutes inférieures à une valeur donnée. Dans notre cas, nous voulons trouver le point qui est le plus proche. Une possibilité est d'utiliser la technique standard pour trouver tous les points ayant une distance inférieure à une valeur donnée, disons $D_{\max}$, et puis trouver le point ayant la plus petite distance. Nous avons développé un algorithme récursif qui autorise une variation de la distance donnée, donc le programme est plus efficace. Plus formellement, un nœud v de l'arbre 3D T est caractérisé par deux termes $(P(v), t(v))$. Le point $P(v)$ est le point par lequel l'espace est divisé en deux. Le paramètre $t(v)$, qui prend la valeur 0, 1 ou 2, indique si le plan de coupe est parallèle au plan yz, xz, ou xy. Deux variables globales P et D sont utilisées pour stocker le point trouvé et la distance correspondante. P est initialisée à -1, et D est initialisée à $D_{\max}$. A la sortie, si P est encore -1, cela veut dire qu'on n'arrive pas à trouver un point avec une distance inférieure à $D_{\max}$. La recherche pour le point le plus proche du point $\mathbf{x}$ est effectuée en appelant `SEARCH(root(`T`),` $\mathbf{x}$`)` de la procédure suivante :

- **entrée :** un point $\mathbf{x}$, l'arbre 3D T; deux variables globales P et D, initialisées à -1 et $D_{\max}$, respectivement.
- **sortie :** le point le plus proche P et la distance correspondante D, si D est inférieure à $D_{\max}$.
- **procedure :** `SEARCH(`$v, \mathbf{x}$`)`
  - **if** $(v ==$ leaf$)$ **return**;
  - $c_1 = \mathbf{x}[t(v)]$;
  - $c_2 = P(v)[t(v)]$; `/* `c_2` a été utilisé pour diviser l'espace */`
  - **if** $(|c_1 - c_2| \leqslant D)$ **and** $(\|\mathbf{x} - P(v)\| \leqslant D)$ **then** $P \Leftarrow P(v), D \Leftarrow \|\mathbf{x} - P(v)\|$;
  - **if** $(c_1 - D < c_2)$ **then** `SEARCH(leftson(`v`),` $\mathbf{x}$`)`;
  - **if** $(c_2 - D < c_1)$ **then** `SEARCH(rightson(`v`),` $\mathbf{x}$`)`;

où $\mathbf{x}$ et $P(v)$ sont des vecteurs 3D, et $\mathbf{x}[t(v)]$ et $P(v)[t(v)]$ sont leurs coordonnées x, y, ou z selon que la valeur de $t(v)$ est 0, 1, ou 2.

Malheureusement, le temps de recherche dans le pire cas est $O(n^{2/3})$ avec un arbre 3D (voir [19, pp. 77]). D'autres algorithmes plus efficaces existent, par exemple, la méthode de l'accès direct, mais ils nécessitent beaucoup plus de mémoire. En pratique, nous avons observé une bonne performance des arbres 3D.

4.2. LA STRATÉGIE DU PLUS GROSSIER AU PLUS FIN

Dans les expériences que nous avons menées, nous avons observé une convergence rapide de l'algorithme pendant les premières itérations qui ralentit quand il approche du minimum local. Comme le temps total de recherche des points les plus proches est

[2] Un certain nombre de points sont d'abord échantillonnés sur une surface modèle. On leur ajoute un bruit et applique un déplacement. On essaye ensuite de retrouver le déplacement entre ces points et le modèle.

[3] Une autre possibilité est la baquétisation en 2D, par exemple, en projetant tous les points sur le sol, ou sur le plan image des capteurs. Nous n'avons pas comparé cette technique avec l'arbre binaire.

[4] Un parallélépipède rectangulaire généralisé peut être un volume infini.

proportionnel au nombre de points de la première vue, il est naturel d'exploiter une stratégie du plus grossier au plus fin. Pendant les premières itérations, nous pouvons utiliser des échantillons plus clairsemés (par exemple, un point retenu tous les 25) au lieu de tous. Quand l'algorithme converge presque, nous utilisons tous les points disponibles pour obtenir une estimation précise. Nous verrons cela plus en détail dans la section sur les résultats expérimentaux.

4.3. CHOIX DU PARAMÈTRE $\mathfrak{D}$

Le seul paramètre qui doit être fourni par l'utilisateur est $\mathfrak{D}$, pour indiquer quand le recalage entre deux vues peut être considéré comme bon. Autrement dit, la valeur de $\mathfrak{D}$ doit correspondre à la distance moyenne espérée quand le recalage est bon. Quand le mouvement est grand, $\mathfrak{D}$ ne doit pas être trop petit. Comme nous prenons $D^0_{\max} = 20\mathfrak{D}$, nous n'arriverions pas à trouver des appariements pendant la première itération si $\mathfrak{D}$ est très petit, et bien sûr nous ne pourrions plus améliorer l'estimation du mouvement. (Une solution est de donner à $D^0_{\max}$ une valeur plus grande, par exemple $30\mathfrak{D}$.)

La valeur de $\mathfrak{D}$ a un impact sur la convergence de l'algorithme. Si $\mathfrak{D}$ est plus petit que nécessaire, on a besoin d'effectuer plus d'itérations pour que l'algorithme converge parce que beaucoup de bons appariements sont écartés pendant la mise à jour des appariements. Par contre, si $\mathfrak{D}$ est beaucoup plus grand que nécessaire, il est possible que l'algorithme ne converge pas vers la bonne solution parce que probablement beaucoup de faux appariements ne sont pas éliminés. Donc, pour être plus prudent, il vaut mieux choisir une petite valeur pour $\mathfrak{D}$.

L'échantillonnage de la surface de la deuxième vue a une influence sur la précision de l'estimation finale à cause de l'utilisation de la distance entre les points, au lieu de la distance entre un point et une surface. Soit $\bar{D}$ la distance moyenne entre les points 3D adjacents de la deuxième vue. Même si les surfaces des deux vues sont bien recalées, la moyenne de la distance μ est égale à $\bar{D}/2$ si les points de la première vue sont localisés entre les points de la deuxième vue. En général, on a $\mu > \bar{D}/2$. Donc, si on calcule $\bar{D}$, on peut prendre pour $\mathfrak{D}$ la valeur $\bar{D}$. Dans notre implémentation, $\mathfrak{D}$ est instanciée à 10 centimètres, qui correspond à peu près à deux fois la résolution (i.e., $\bar{D}$) des données obtenues avec un système de vision stéréoscopique équipé de caméras de résolution 512×512 avec un champ de profondeur de 10 mètres.

4.4. COMMENT FAIRE DANS LE CAS D'UNE TRÈS MAUVAISE ESTIMÉE INITIALE?

Dans cette section, nous discutons du choix de la valeur de ξ que nous n'avons pas encore expliqué. Dans la section 3.3, nous avons expliqué comment mettre à jour les appariements en utilisant une analyse statistique des distances, et nous avons supposé que l'histogramme des distances a une distribution approximativement gaussienne quand le recalage entre deux vues est bon. Mais quand on a une très mauvaise estimée initiale du mouvement entre deux vues, on observe en général une forme très compliquée de la distribution des distances. Nous montrons dans la figure 2 un histogramme représentatif de la distribution des distances quand l'estimée initiale est très mauvaise.

Comme nous pouvons le remarquer, la forme de l'histogramme dans la figure 2 est très irrégulière. Il existe plusieurs pics. De plus, nous trouvons beaucoup de points près de zéro. Ceci montre la difficulté de notre approche. Quand l'estimée initiale est très mauvaise, on trouve probablement des appariements pour de petites distances à cause du mauvais alignement des points, c'est-à-dire que ces appariements ne sont pas raisonnables. Une solution possible est d'émettre une hypothèse correspondant à chaque pic, puis évaluer chaque hypothèse en parallèle. Le critère pour mesurer la qualité d'une hypothèse peut être une fonction du nombre d'appariements et de la moyenne des distances finales. A la fin, l'hypothèse qui donne le meilleur score est retenue comme la transformation entre les deux vues.

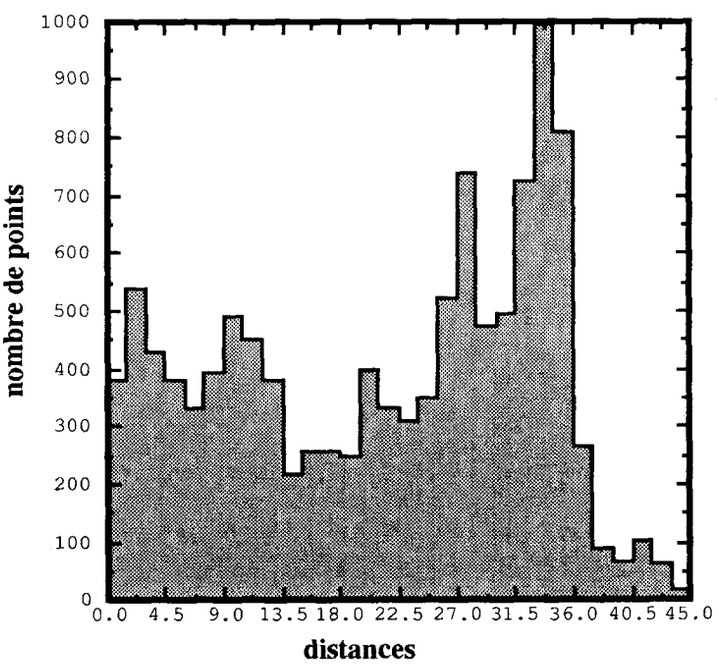

Figure 2. – Histogramme des distances quand l'estimée initiale du mouvement est très mauvaise.

Nous avons adopté une méthode plus simple. Le pic maximal est un bon candidat en général (du moins on l'espère) pour une correspondance raisonnable entre les deux vues. Nous avons donc choisi dans notre implémentation la vallée après le pic maximal pour fixer la valeur de ξ (voir la figure 3). Tous les appariements pour des distances supérieures à ξ sont écartés. Pour éviter la perturbation dû au bruit, on impose que le nombre de points sur la vallée ne doit pas être supérieur à 60% du nombre de points sur le pic. Dans les expériences que nous avons menées, cette méthode nous a donné des résultats satisfaisants, comme nous allons le voir par la suite.

Figure 4. – *Le premier triplet d'images prises au LETI.*

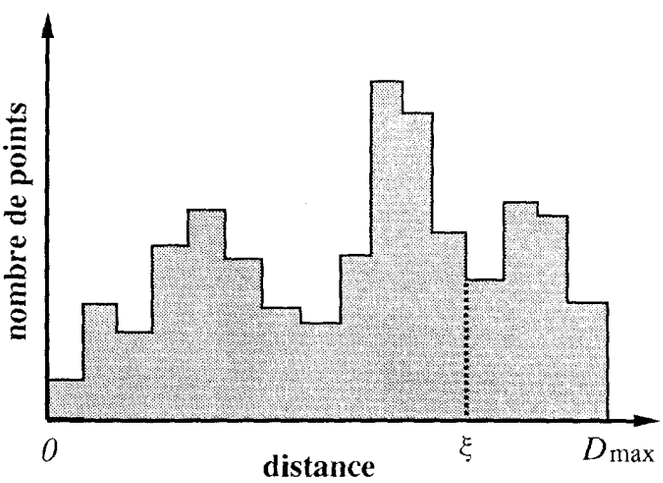

Figure 3. – Comment choisir la valeur ξ.

5. Résultats expérimentaux

Nous décrivons dans cette section des expériences que nous avons menées avec l'algorithme décrit dans les deux dernières sections [5].

5.1. DONNÉES DU LETI [6]

Nous montrons dans les figures 4 et 5 deux triplets d'images d'une scène de cailloux prises au LETI depuis deux positions différentes. Le déplacement entre ces deux positions est presque une pure translation d'un mètre.

L'algorithme de stéréovision par corrélation reconstruit 48589 points pour le premier point de vue et 48054 points pour le deuxième. La superposition de ces deux cartes 3D est montrée dans la figure 6, où la première carte est dessinée en quadrangles, et la deuxième en surface grisée. Nous pouvons remarquer la différence importante entre ces deux vues (les surfaces de la première vue sont devant celles de la deuxième).

La construction de la représentation de l'arbre 3D pour le second point de vue a pris 27 secondes CPU sur une station de travail SUN 4/60. Partant de cette différence (i.e., avec l'estimée initiale du mouvement égale à zéro), et utilisant un point parmi neuf dans la première position, le recalage a pris 68 secondes CPU après 20 itérations pour produire le résultat montré en Fig. 7. Il est clair que les quadrangles et les surfaces grisées sont mélangées (on voit beaucoup mieux avec une animation). Cela veut dire que les deux vues sont bien recalées [7]. Si on utilise tous les points de la première vue, le programme prend 610 secondes CPU pour obtenir pratiquement le même résultat. Le temps d'exécution est linéaire en fonction du nombre de points de la première position.

L'histogramme que nous avons montré en Fig. 2 est en fait tiré de cette expérience. En Fig. 8, nous montrons l'histogramme obtenu pour la deuxième itération et celui obtenu pour la dernière itération. Nous pouvons observer comment la forme de l'histogramme change avec le recalage. L'échelle des distances a diminué de 45 pour la première itération à 4,5 pour la dernière.

5.2. DONNÉES DU LAAS [8]

Nous montrons en Fig. 9 et Fig. 10 deux triplets d'images d'une scène de cailloux prises au LAAS depuis deux positions

[5] Nous avons implémenté un algorithme similaire pour le recalage entre deux ensembles de courbes sans contrainte [29, 28]. Le lecteur peut trouver dans [28] des expériences que nous avons menées avec des données de synthèse, qui montrent bien les caractéristiques de l'algorithme.

[6] LETI/Centre d'études nucléaires de Grenoble, av. des Martyrs, F-38041 Grenoble Cedex.

[7] Malheureusement, le déplacement entre les deux vues n'est pas calibré. On ne peut apprécier la qualité du recalage que d'un point de vue visuel.

[8] LAAS-CNRS, 7 av. du Colonel Roche, F-31077 Toulouse Cedex.

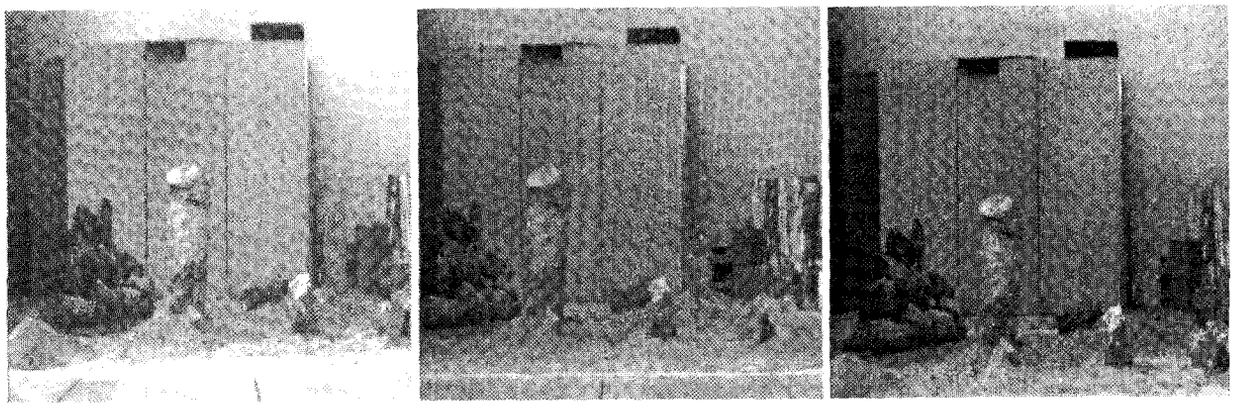

Figure 5. – Le deuxième triplet d'images prises au LETI.

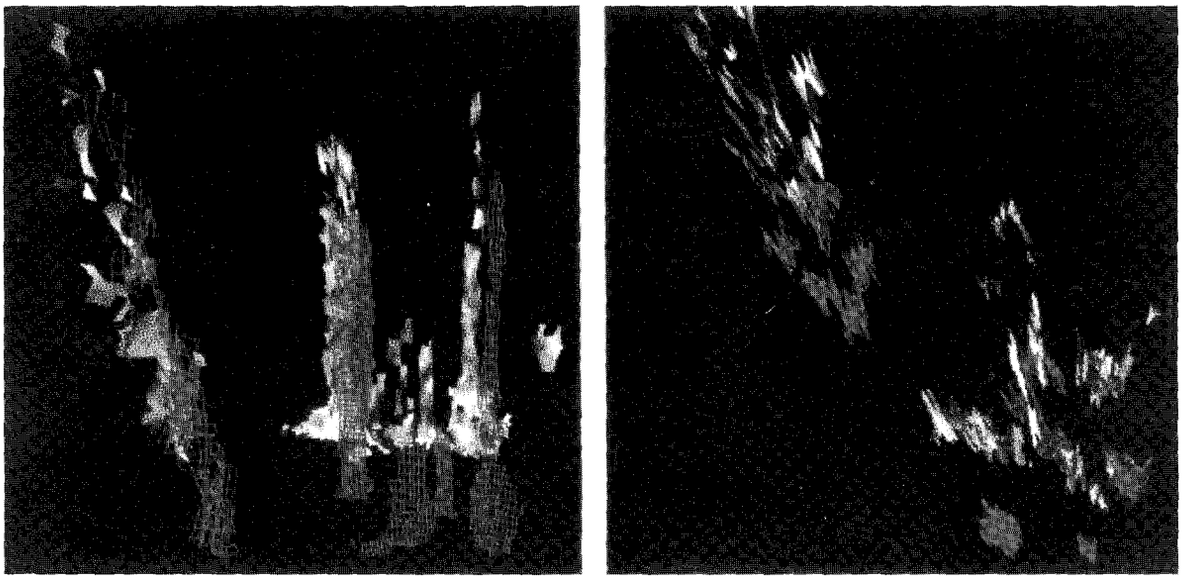

Figure 6. – Superposition des deux cartes 3D avant recalage : vue de face et vue de dessus.

différentes. Les caméras se situent à peu près à 6 mètres de lascène. Il y a 30 degrés en rotation et 3,75 mètres en translation entre ces deux positions.

Le système de stéréovision par corrélation reconstruit 71505 points pour la première position et 51503 points pour la deuxième. Comme nous disposons sur 2 autres triplets similaires de marques sur les cailloux, nous pouvons recaler les deux vues manuellement. Le résultat du recalage manuel est montré en Fig. 11, où la première carte est dessinée en quadrangles, et la deuxième en surfaces grisées. Nous pouvons remarquer que le recalage entre les deux vues est correct. Par la suite, nous allons comparer les résultats de notre algorithme avec ce résultat en utilisant différentes estimées initiales. Les deux vues sont maintenant exprimées dans le même repère en appliquant l'estimée manuelle à la première vue. Donc, l'estimée finale espérée de la transformation entre les deux vues sera désormais zéro. A partir de la figure 11, on peut constater qu'il y a beaucoup de points qui ne sont visibles que d'une vue (ceux qui n'ont pas de correspondances) à cause des occlusions, disparitions et apparitions.

Si on utilise une estimée initiale égale à $[0.0, 0.0, 0.17, 0.0, -1.5, 0.0]^T$ [9] (c'est-à-dire, une rotation de 10 degrés et une translation de 1.5 mètres), la différence au départ est assez grande, comme

[9] Nous utilisons la représentation de l'axe de rotation pour représenter la transformation entre deux vues. Elle est donc composée de 6 paramètres. Les trois premiers sont pour la rotation : la norme est l'angle de rotation (en radians) et le vecteur est parallèle à l'axe de rotation. Les trois derniers constituent le vecteur de translation (en mètres).

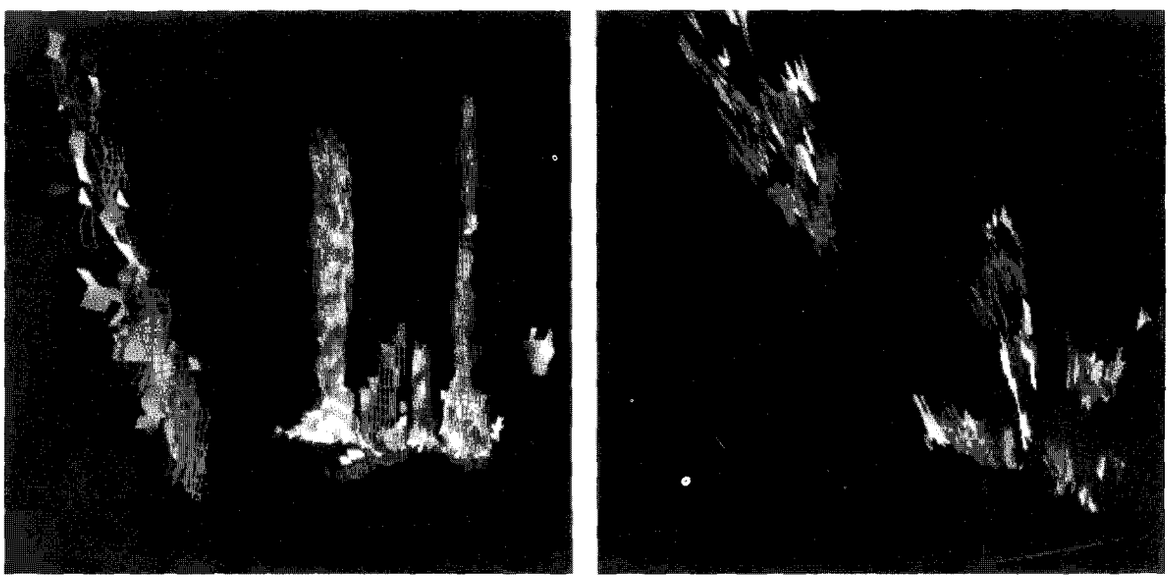

Figure 7. – Superposition des deux cartes 3D après recalage : vue de face et vue de dessus.

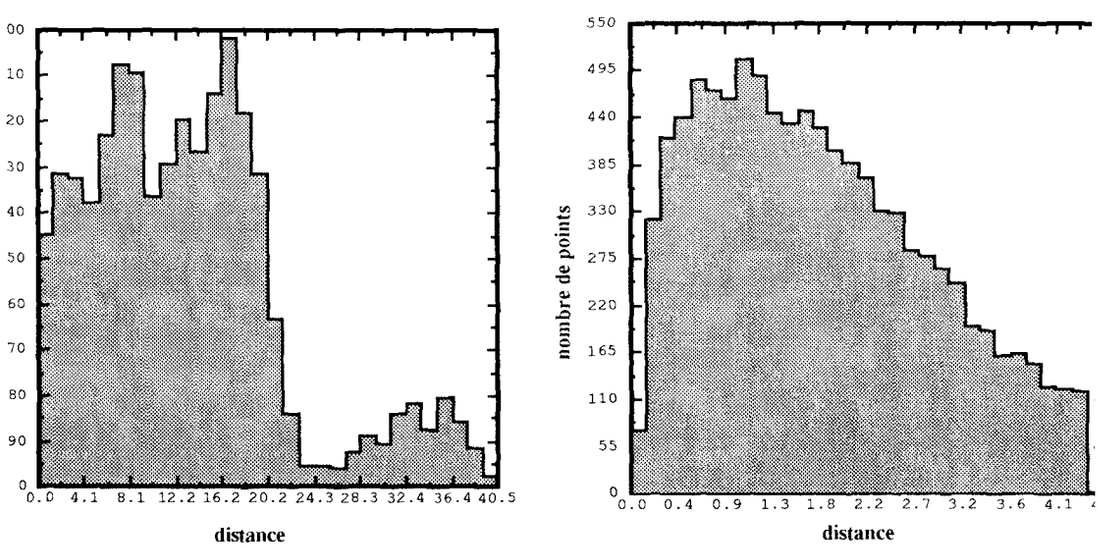

Figure 8. – Histogrammes des distances de la deuxième et de la dernière itération.

montrée en Fig. 12. Après 40 itérations, l'estimée du mouvement est $[9.760916 \times 10^{-3}, -5.288758 \times 10^{-3}, 5.247401 \times 10^{-3}, 1.618380 \times 10^{-2}, -2.557654 \times 10^{-2}, -1.913078 \times 10^{-2}]^T$, ce qui indique aussi la différence entre l'estimation automatique et l'estimation manuelle. Cette différence est en fait très petite : 0.7 degré en rotation et 3.58 centimètres en translation. Le résultat du recalage est montré en Fig. 13.

Maintenant, si on augmente l'écart entre l'estimation initiale et l'estimation finale, que va-t-on obtenir? L'estimée initiale dans cet exemple est $[0.0, 0.0, 0.35, 0.5, -2.0, 0.2]^T$ (c'est-à-dire, une rotation de 20 degrés et une translation de 2.07 mètres). La différence entre les deux vues correspondant à cette estimée est montrée en Fig. 14. Après 40 itérations, l'estimée du mouvement est $[7.885261 \times 10^{-3}, -1.208467 \times 10^{-2}, 4.158183 \times 10^{-3}, 2.661822 \times 10^{-2}, -1.230099 \times 10^{-2}, -4.845936 \times 10^{-2}]^T$. Il y a donc une différence de 0.86 degrés en rotation et de 5.66 centimètres en translation. Le résultat du recalage est montré en Fig. 15. On voit que même quand l'estimée initiale est très différente de l'estimée réelle, on obtient encore des résultats

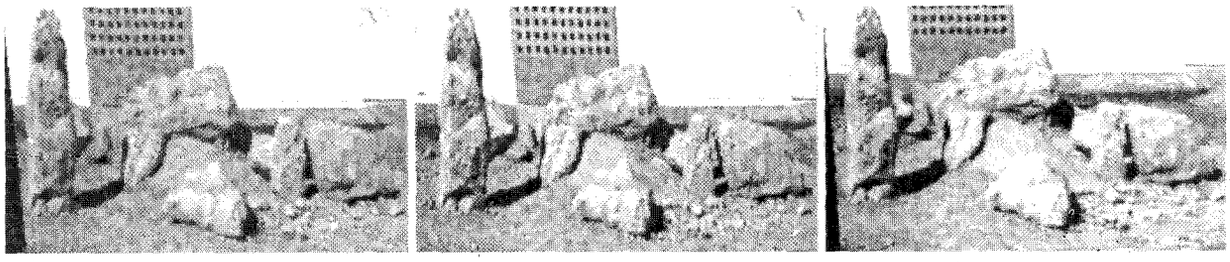

Figure 9. – Le premier triplet d'images prises au LASS.

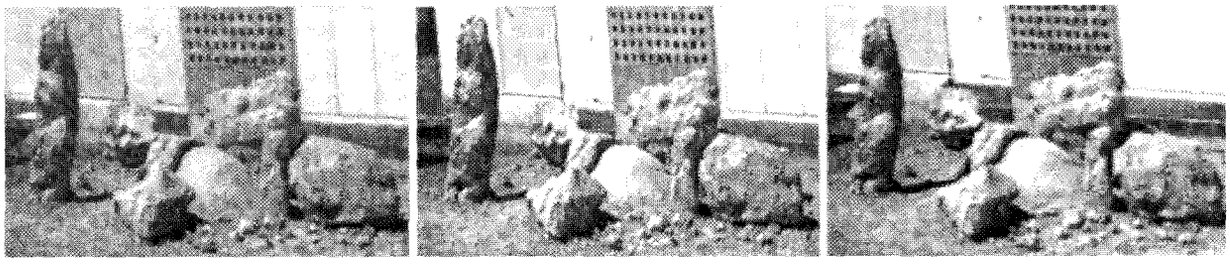

Figure 10. – Le deuxième triplet d'images prises au LAAS.

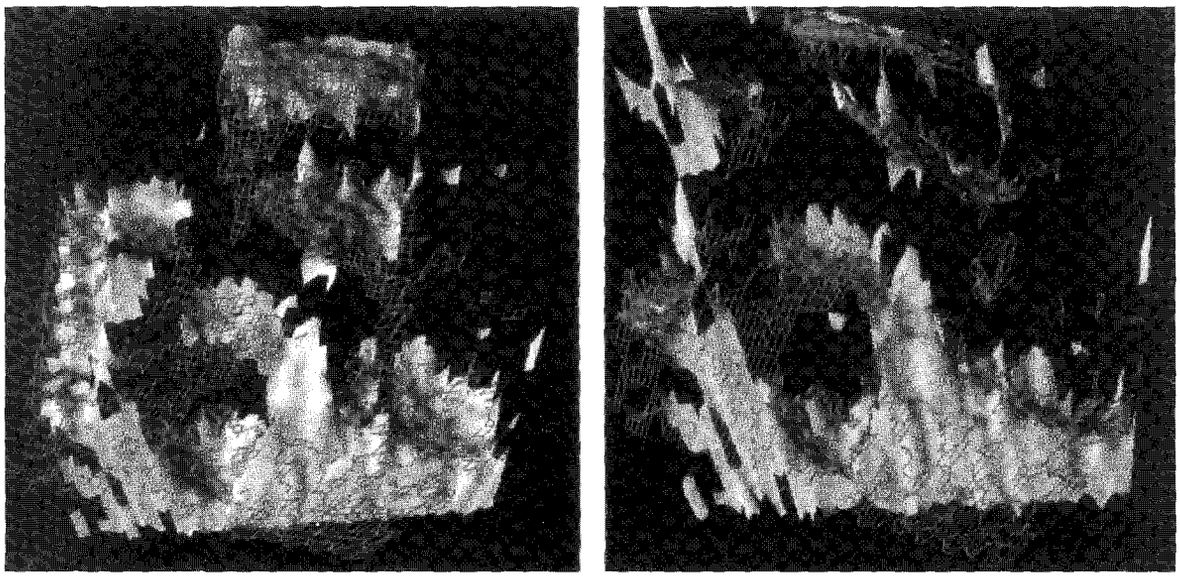

Figure 11. – Superposition des deux cartes 3D après le recalage manuel : vue de face et vue de dessus.

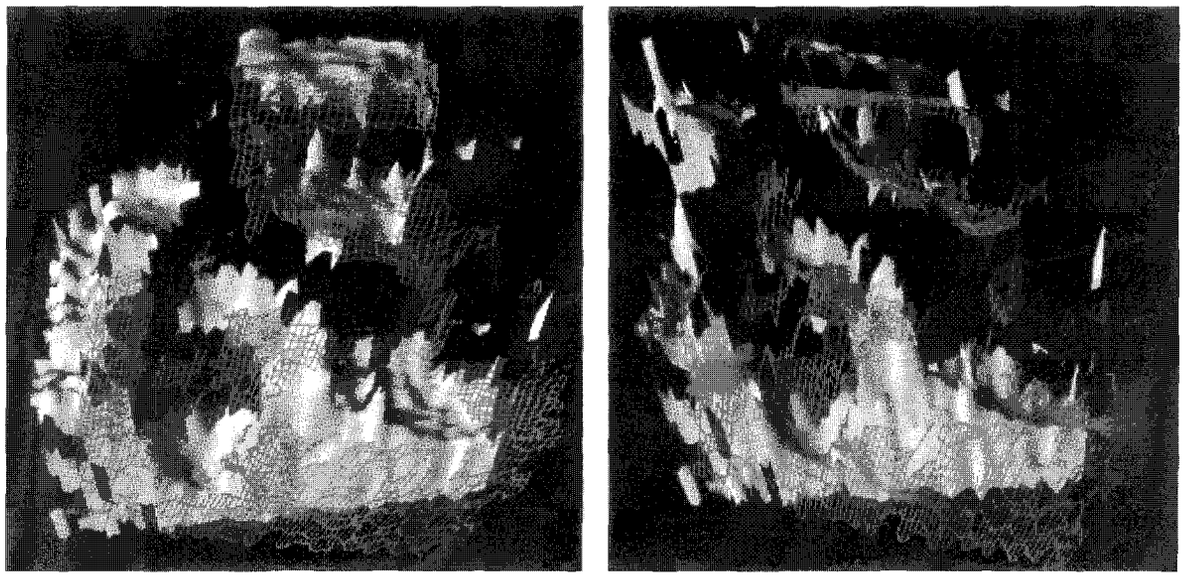

Figure 12. – Exemple 1 : Superposition des deux cartes 3D de départ : vue de face et vue de dessus.

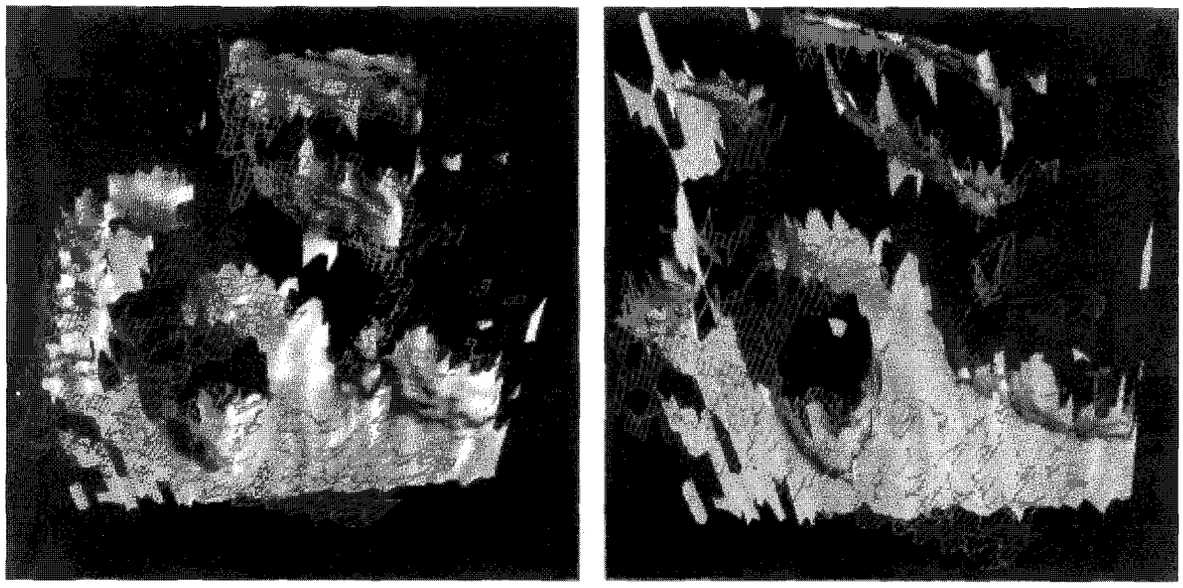

Figure 13. – Exemple 1 : Superposition des deux cartes 3D après le recalage : vue de face et vue de dessus.

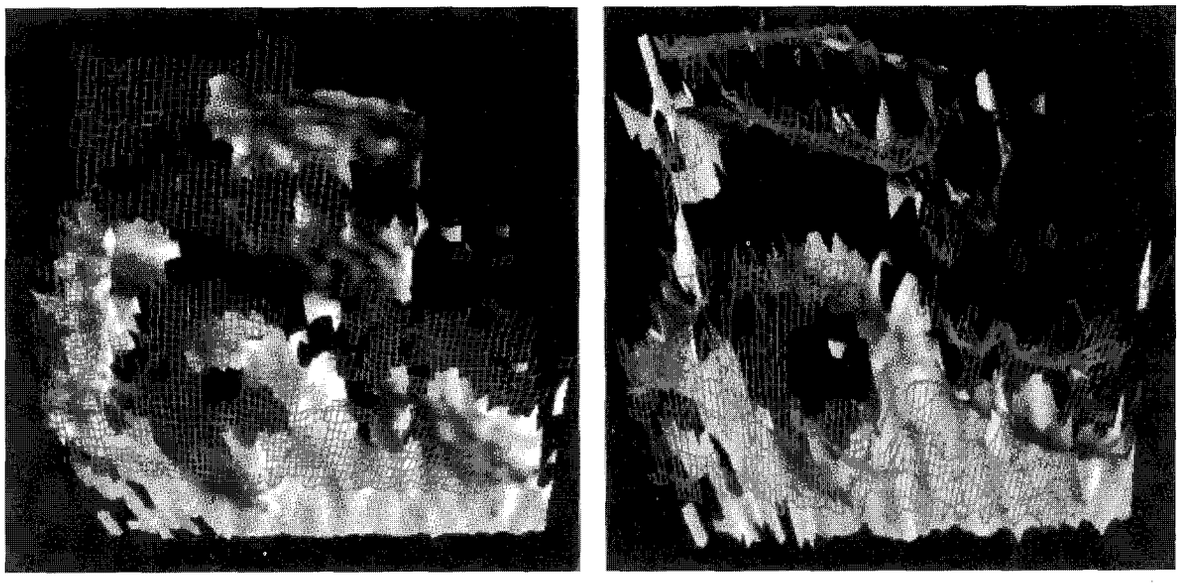

Figure 14. – Exemple 2 : Superposition des deux cartes 3D de départ : vue de face et vue de dessus.

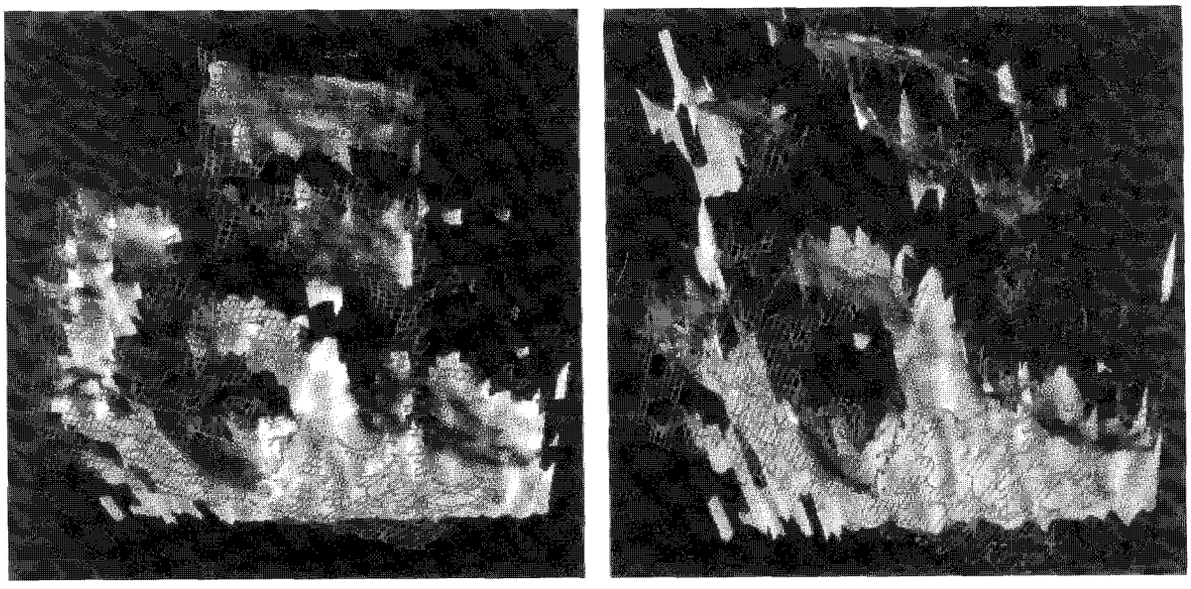

Figure 15. – Exemple 2 : Superposition des deux cartes 3D après le recalage : vue de face et vue de dessus.

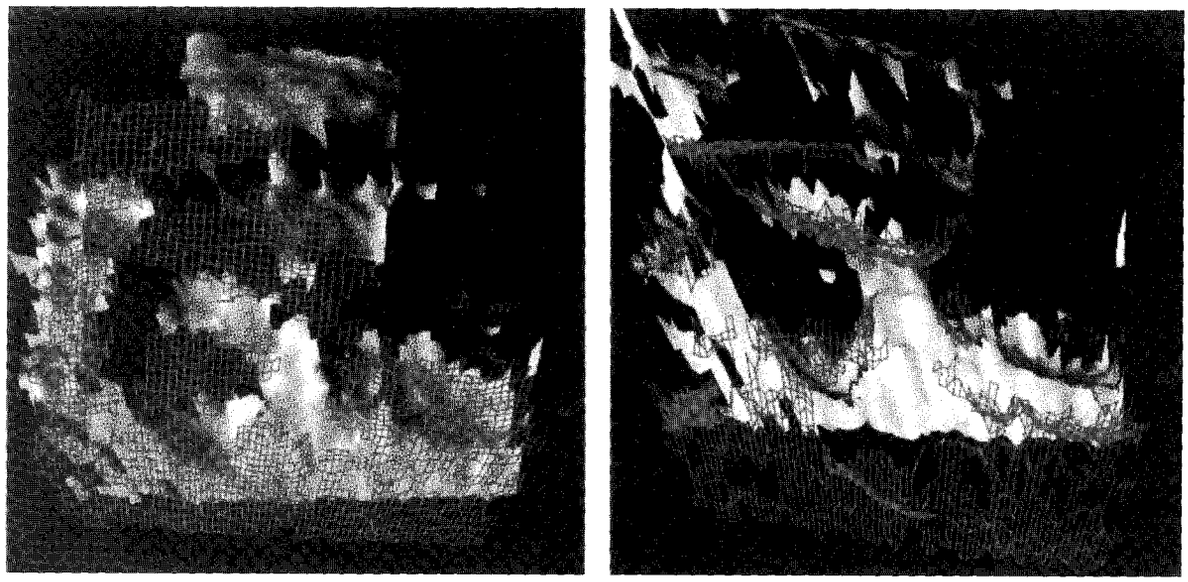

Figure 16. – Exemple 3 : Superposition des deux cartes 3D de départ : vue de face et vue de dessus.

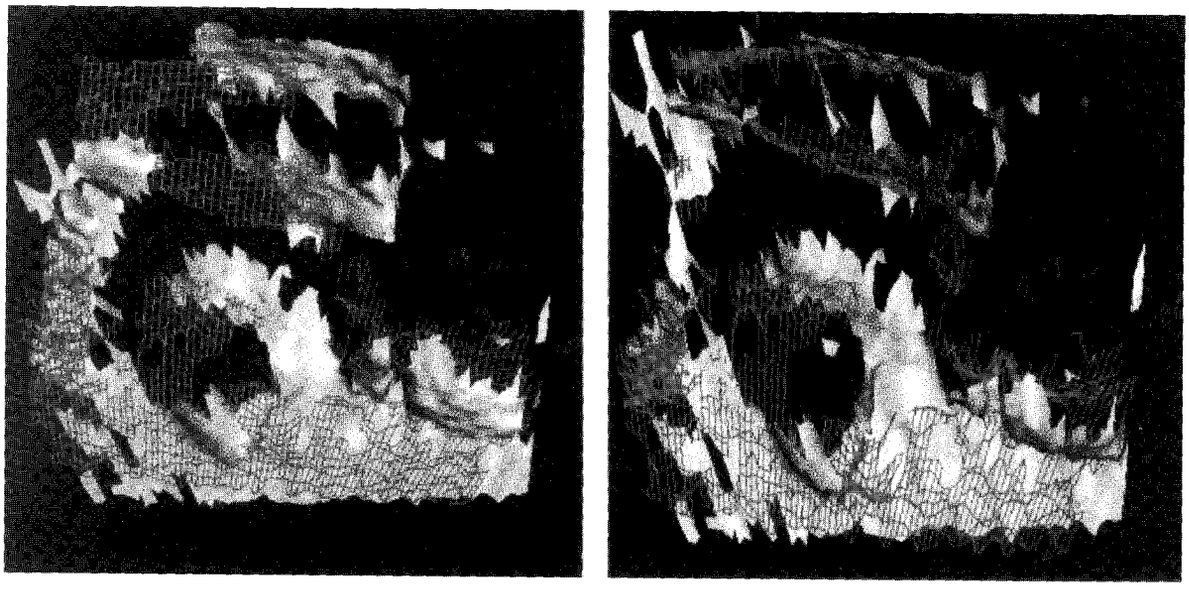

Figure 17. – Exemple 3 : Superposition des deux cartes 3D après 40 itérations : vue de face et vue de dessus.

satisfaisants. Avec plus d'itérations, on obtient un meilleur résultat encore.

Que se passe-t-il si on augmente encore l'écart entre l'estimation initiale et l'estimation finale ? L'estimée initiale dans cet exemple est $[0.0, 0.0, 0.35, -0.5, -2.5, 0.2]^T$ (c'est-à-dire, une rotation de 20 degrés et une translation de 2.56 mètres). La différence entre les deux vues correspondant à cette estimée est montrée en Fig. 16. Après 40 itérations, l'estimée du mouvement est $[-3.825570 \times 10^{-2}, 6.001669 \times 10^{-2}, 3.071064 \times 10^{-1}, 3.449387 \times 10^{-1}, -1.795149 \times 10^{0}, 3.954597 \times 10^{-1}]^T$. Le résultat est médiocre, comme montré en Fig. 17, mais c'est mieux qu'au départ.

Si on continue, le résultat s'améliore. Après 80 itérations, l'estimée du mouvement est $[1.012586 \times 10^{-2}, -9.563972 \times 10^{-3}, 8.020001 \times 10^{-3}, 3.110009 \times 10^{-2}, -3.634908 \times 10^{-2}, -3.905361 \times 10^{-2}]^T$. Donc, la différence avec l'estimée manuelle est de 0.92 degrés en rotation et 6.18 centimètres en translation, ce qui est raisonnablement faible, comme montré en Fig. 18.

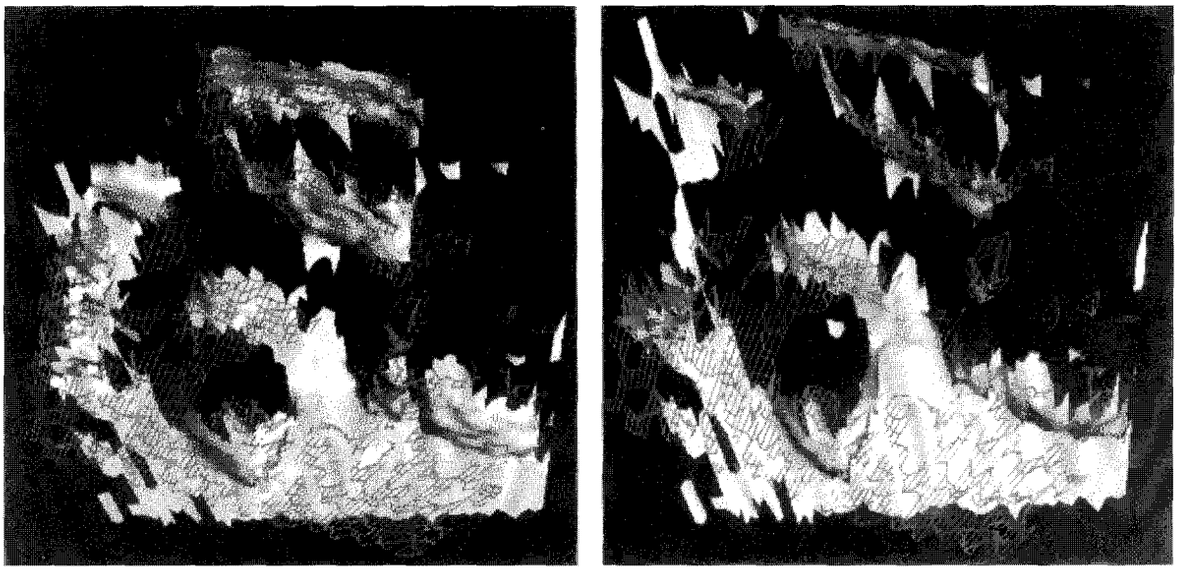

Figure 18. – Exemple 3 : Superposition des deux cartes 3D après le recalage : vue de face et vue de dessus.

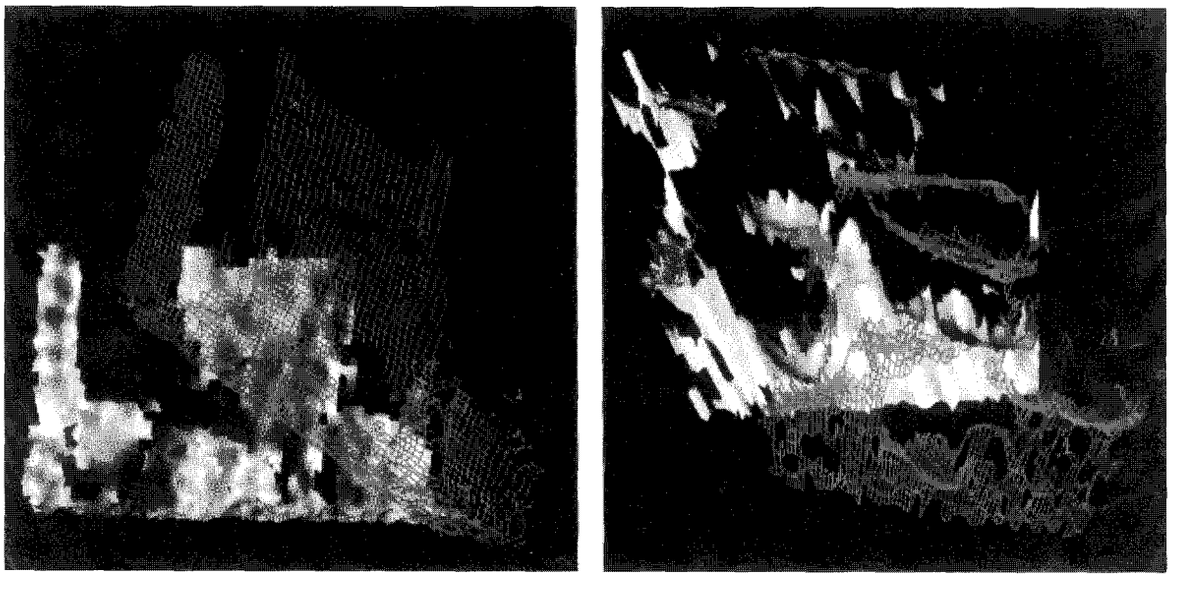

Figure 19. – Exemple 4 : Superposition des deux cartes 3D de départ : vue de face et vue de dessus.

Jusqu'à présent, nos expériences ont toutes été menées avec une rotation autour d'un axe perpendiculaire au plan du sol. Que se passe-t-il si le VAP se trouve sur deux pentes différentes (par exemple, le VAP grimpe sur un tas de cailloux) ? Voici un exemple correspondant à cette situation. L'estimée initiale est $[0.35, 0.0, 0.17, -0.5, -2.5, 0.2]^T$. Il y a donc une rotation de 20 degrés par rapport au plan du sol, une rotation de 10 degrés autour de l'axe perpendiculaire au plan du sol, et une translation de 2.56 mètres. La différence entre les deux vues correspondant à cette estimée est montrée en Fig. 19. Après 40 itérations, l'estimée du mouvement est $[1.125988 \times 10^{-2}, 4.676589 \times 10^{-4}, 1.893187 \times 10^{-3}, 4.918958 \times 10^{-3}, -2.754730 \times 10^{-2}, 6.532630 \times 10^{-3}]^T$. La différence avec le recalage manuel est donc de 0.65 degrés en rotation et de 2.87 centimètres en translation, ce qui est tout à fait correct. Le résultat du recalage est montré en Fig. 20. Cet exemple montre que l'algorithme est plus robuste à la rotation qui n'est pas sur le sol qu'à celle sur le sol. Ceci peut être expliqué par le fait

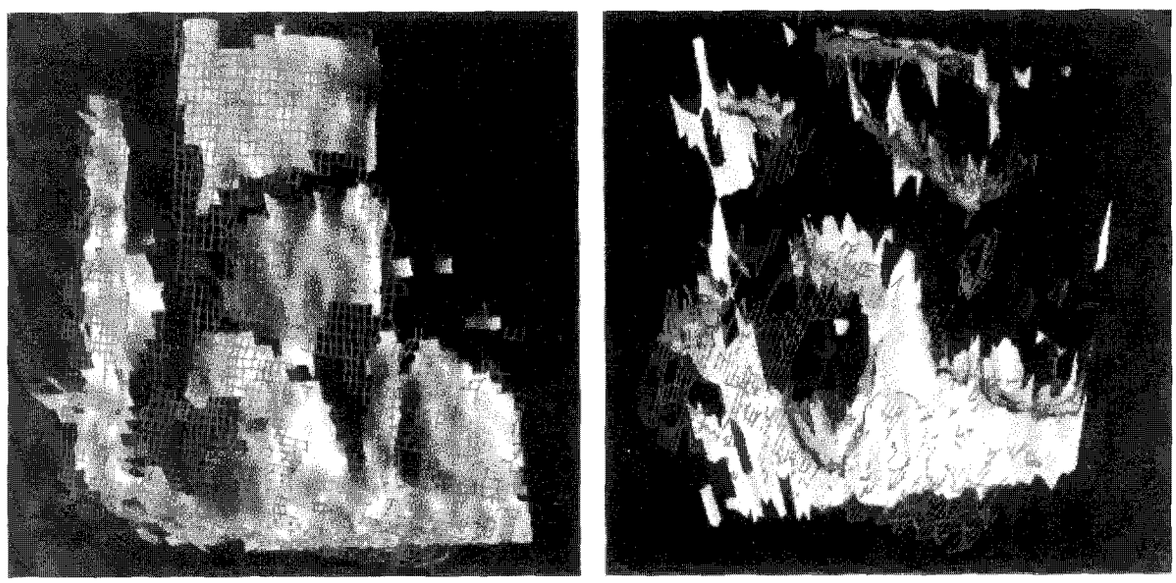

Figure 20. – Exemple 4 : Superposition des deux cartes 3D après le recalage : vue de face et vue de dessus.

que le point le plus proche trouvé dans cette situation est plus proche du vrai point correspondant que dans l'autre situation.

Nous récapitulons les résultats présentés dans la table 1. Ceux-ci ont été obtenus en utilisant tous les points de la première vue. Pour justifier la stratégie *du plus grossier au plus fin*, nous considérons le deuxième exemple. Si on utilise un point retenu tous les 16 de la première vue, l'estimée du déplacement apres 40 itérations est $[1.063305 \times 10^{-2}, -1.789202 \times 10^{-2}, 6.939914 \times 10^{-3}, 3.879424 \times 10^{-2}, -2.809201 \times 10^{-2}, -8.193954 \times 10^{-2}]^T$. Il y a donc une différence de 1.25 degrés en rotation et de 9.49 centimètres en translation avec l'estimée manuelle. Le temps de calcul est alors de 49 secondes. La différence entre cette estimée et l'estimée obtenue en utilisant tous les points est de 0.4 degrés en rotation et de 3.9 centimètres en translation. Cette estimée est un peu moins bonne, mais le temps de calcul a été considérablement réduit (de 771 à 49 secondes, soit 16 fois moins!). Maintenant, si l'on effectue 20 itérations avec un point de la première vue retenu sur 16, suivi par 20 itérations avec tous les points, on obtient l'estimée du déplacement suivante : $[7.106948 \times 10^{-3}, -1.364340 \times 10^{-2}, 7.548452 \times 10^{-3}, 3.235070 \times 10^{-2}, -3.200847 \times 10^{-2}, -5.565781 \times 10^{-3}]^T$. Le temps de calcul est alors de 384 secondes. La différence entre cette estimée et l'estimée obtenue en utilisant tous les points est de 0.2 degrés en rotation et de 2.2 centimètres en translation. En tenant compte de la résolution des données disponibles (à peu près 5 centimètres), la différence est négligeable, mais le temps de calcul est réduit de moitié. D'après cet exemple et de nombreuses expériences que nous avons menées, l'approche *du plus grossier au plus fin* est nécessaire pour obtenir un temps de calcul raisonnable avec seulement une perte négligeable en précision de l'estimation du déplacement. Le nombre d'itérations avec des points clairsemés dépend de la précision de l'estimation demandée et de la contrainte en temps de calcul. A la limite, on peut n'utiliser que des points clairsemés.

Tableau 1. – Résumé des résultats avec les données du LAAS

	Différence initiale			Différence finale		Nombre	Temps de
Exemple	$\mathbf{r}_{sol}$	$\mathbf{r}_{hors}$	$\mathbf{t}$	$\mathbf{r}$	$\mathbf{t}$	d'iterations	calcul
1	10	0	150	0.70	3.58	40	770
2	20	0	207	0.86	5.66	40	771
3	20	0	256	18.06	187.	40	750
3 bis	20	0	256	0.92	6.18	80	1512
4	10	20	256	0.65	2.87	40	757

* La rotation $\mathbf{r}$ est exprimée en degrés. $\mathbf{r}_{sol}$ est pour le composant de la rotation sur le sol. $\mathbf{r}_{hors}$ est pour le composant de la rotation qui n'est pas sur le sol.

† La translation $\mathbf{t}$ est exprimée en centimètres.

‡ Le temps de calcul est exprimé en secondes CPU sur une station de travail SUN 4/75.

6. Discussions

6.1. L'ÉCHANTILLONNAGE DE LA SURFACE POUR LE SECOND POINT DE VUE

Comme décrit plus haut, les points sont considérés comme des échantillons d'une surface de forme quelconque, l'objectif étant de trouver une transformation entre deux surfaces en minimisant leur distance. Mais l'algorithme que nous avons développé dans la section 3 est basé sur la distance entre deux ensembles de points. C'est-à-dire que l'on utilise le minimum de toutes les distances

d'un point donné (de la première vue) à *chaque échantillon* de la surface (de la deuxième vue), au lieu de la distance d'un point donné à *la surface*. Sans aucun doute, ceci influence la précision de l'estimée finale. Plus le nombre d'échantillons sera grand, moins l'influence de l'échantillonnage sur l'estimation finale sera importante.

Pour résoudre le problème dû à l'échantillonnage, il faudrait utiliser idéalement la distance entre un point et une surface. Par exemple, on peut approximer la surface par une triangulation de Delaunay, et puis examiner la distance entre le point donné et tous les triangles. Mais à ce moment-là, nous perdons l'efficacité obtenue avec l'utilisation des échantillons. Une autre possibilité est la suivante. Trouver d'abord trois points les plus proches du point donné. Ensuite calculer la distance du point donné au plan passant par ces trois points. Cette distance peut être utilisée comme une approximation raisonnable de la vraie distance.

Dans notre travail, nous n'avons pas exploité ces techniques, parce que nous avons des points suffisamment denses dans la deuxième vue. L'influence de l'échantillonnage sur l'estimation du mouvement est donc limitée. Dans le futur, nous allons mettre en oeuvre ces techniques pour voir leur apport quand les points disponibles ne sont pas suffisamment denses.

6.2. L'INCERTITUDE SUR LES POSITIONS 3D

L'importance d'estimer et de manipuler l'incertitude sur les données d'une manière explicite est maintenant bien reconnue par la communauté de la vision par ordinateur et de la robotique [4, 17, 14, 2, 24]. C'est extrêmement important quand les données disponibles ont une incertitude différente, par exemple en stéréovision où celle-ci augmente rapidement avec la profondeur. Nous avons montré dans [31] que l'on obtient des résultats meilleurs si on tient compte de l'incertitude sur les données dans l'estimation du mouvement (en utilisant, par exemple, un filtre de Kalman).

La vraie distribution de l'incertitude est très compliquée, et il est donc très difficile, voire impossible, de la modéliser. Pour une approximation raisonnable, l'incertitude sur la position d'un point 3D reconstruit par la stéréovision est souvent considérée comme gaussienne; c'est-à-dire, qu'elle est caractérisée par un vecteur de position et une matrice de covariance. L'algorithme décrit en Sect. 3.4 est très efficace pour estimer le mouvement, mais on a supposé que chaque position des points a la même incertitude. Et malheureusement, il est très difficile de l'étendre pour tenir compte complètement de l'incertitude. Pour ce faire, on peut utiliser par exemple les techniques de filtrage de Kalman qui ont été abondamment utilisées pour résoudre avec succès beaucoup de problèmes de la vision [30].

En fait, on peut facilement étendre l'algorithme pour tenir compte *partiellement* de l'incertitude. On peut associer, à chaque appariement entre deux vues ($\mathbf{x_i}$, $\mathbf{y}_i$), un facteur de pondération w_i. La méthode du quaternion et la méthode du quaternion dual permettent de calculer efficacement le mouvement en résolvant un problème de moindres-carrés pondérés par un facteur scalaire.

Le facteur de pondération w_i doit être lié à l'incertitude de $\mathbf{R}\mathbf{x}_i + \mathbf{t} - \mathbf{y}_i$. Soient $\Lambda\mathbf{x_i}$, $\Lambda\mathbf{y_i}$, et Λi les matrices de covariance de $\mathbf{x_i}$, $\mathbf{y_i}$, et $\mathbf{R}\mathbf{x}_i + \mathbf{t} - \mathbf{y}_i$. $\Lambda\mathbf{x_i}$ and $\Lambda\mathbf{y_i}$ sont données par le système de perception, e.g., la stéréovision. Λi peut être calculée comme suit :

$$\Lambda i = \mathbf{R}\Lambda\mathbf{x_i}\mathbf{R}^T + \Lambda\mathbf{y_i} \ ,$$

où $\mathbf{R}$ est une approximation, calculée lors de la dernière itération, de la matrice de rotation. La trace de Λi indique à peu près la grandeur de l'incertitude de $\mathbf{R}\mathbf{x}_i + \mathbf{t} - \mathbf{y}_i$. Donc, nous choisissons w_i en fait de la façon suivante :

$$w_i = \frac{1}{\mathrm{tr}(\Lambda \mathrm{i})} = \frac{1}{\mathrm{tr}(\Lambda\mathbf{x_i}) + \mathrm{tr}(\Lambda\mathbf{y_i})} \ .$$

Le facteur de pondération est donc indépendant du déplacement.

Cette technique n'est pas intégrée dans la présente implémentation, mais sera testée sur la maquette du VAP à laquelle nous participons.

6.3. APPROCHE SYMÉTRIQUE

Comme souligné par un des rapporteurs de cet article, le critère que nous avons adopté, l'equation (1), n'est pas symétrique (apparier seulement les points de la première vue aux points de la deuxième vue) alors que le problème est parfaitement symétrique. Nous avons effectué cette simplification parce que nous avons constaté pendant les expériences menées sur les courbes [28] que l'apport de l'utilisation d'un critère symétrique est moins significatif que l'accroissement du temps de calcul. Par exemple, pour les données de synthèse présentées dans [28] en ajoutant un bruit gaussien avec l'écart type de 6, l'erreur de l'estimée après 10 itérations donnée par l'algorithme avec le critère non-symétrique est 7.56% en rotation et 7.61% en translation; celle donnée par l'algorithme avec le critère symétrique est 6.40% en rotation et 6.42% en translation. On constate une légère amélioration avec le critère symétrique, mais le temps de calcul est passé de 7.64 secondes CPU à 15.88 secondes CPU, soit deux fois plus. Donc dans les applications fortement contraintes par la rapidité du calcul, le critère non-symétrique est préférable.

6.4. QUE FAIRE SANS ESTIMATION INITIALE?

Pour le VAP, parce qu'on a des instruments comme l'odométrie, la centrale gyro-accélérométrique, etc, ce problème normalement ne se pose pas (sauf si ces instruments sont en panne).

Dans le cas où le mouvement entre deux vues est grand et où l'on n'a pas d'estimation initiale, notre algorithme peut être adapté pour traiter ce genre de problème de deux manières. La première est d'appliquer des méthodes globales que nous avons citées dans la section 2 pour obtenir une estimation, qui est ensuite raffinée en appliquant notre algorithme. La deuxième est d'échantillonner l'espace du mouvement (qui est à 6 dimensions), puis d'appliquer notre algorithme en considérant chaque échantillon comme une estimation initiale. L'estimation finale correspondant à l'erreur minimale globale est retenue comme l'estimation optimale du mouvement entre les deux vues.

6.5. MOUVEMENTS MULTIPLES

Dans un environnement tel que celui du VAP (qui est censé fonctionner sur Mars), il est rare que l'on rencontre plusieurs objets en mouvement. Notre algorithme n'est pas prévu pour traiter ce genre de problème. Par contre, dans un environnement dynamique, il y a souvent plus d'un objet en mouvement. S'il y a des petits objets en mouvement mis à part le VAP ("petit" étant mesuré par le nombre de point de ces objets par rapport à celui de l'environnement statique perçu), notre algorithme va les considérer comme des données aberrantes à cause du mécanisme intégré décrit en Sect. 3.3.

Une approche pour traiter ce problème est la suivante : segmenter d'abord la première vue en plusieurs morceaux, appliquer ensuite l'algorithme décrit ci-dessus à chaque morceau. On a donc une estimée du mouvement pour chaque morceau. On préfère sur-segmenter la première vue que la sous-segmenter, parce que sinon un morceau peut contenir plusieurs objets avec des mouvements différents. A la fin, les morceaux peuvent être regroupés en objets selon la similarité du mouvement estimé.

7. Conclusion

Nous avons décrit un algorithme pour le recalage de deux ensembles de points 3D, reconstruits par un système de stéréovision par corrélation dans un environnement non-structuré. Nous avons utilisé l'hypothèse que le mouvement entre les deux vues est petit ou approximativement connu, une hypothèse réaliste dans beaucoup d'applications y compris la navigation visuelle. Un nombre important d'expériences ont été menées, et les résultats sont tout à fait convaincants.

Notre algorithme a les caractéristiques suivantes :

- Il est simple. Il est facile à reproduire.
- Il est extensible. Des stratégies plus complexes comme la continuité des distances dans un voisinage sont faciles à intégrer.
- Il est général. Tout d'abord, la représentation utilisée est générale pour représenter la scène observée en pratique. Ensuite, les idées soutenant cet algorithme sont applicables à d'autres problèmes d'appariement, par exemple, la mise en correspondance de courbes [10].
- Il est efficace. La partie plus coûteuse en temps de calcul est le processus pour trouver les points les plus proches, qui a une complexité $O(m \log n)$. On peut accélérer considérablement l'algorithme en exploitant une stratégie du plus grossier au plus fin décrite dans la section 4.2.
- Il est robuste à des erreurs grossières et peut traiter des problèmes comme l'apparition, la disparition, et l'occlusion d'objets, comme décrit dans la section 3.6, grâce à l'analyse statistique dynamique des distances, comme décrit dans la section 3.3.
- Il produit une estimation précise parce que toute information disponible est utilisée dans l'algorithme.
- Il ne nécessite aucun prétraitement des points 3D, comme le lissage. Les données sont utilisées telles quelles. C'est-à-dire qu'il n'y a pas d'erreur d'approximation.
- Il n'a besoin d'aucune opération de dérivation (qui est sensible au bruit), par contraste avec les méthodes basées sur des primitives particulières.

Notre algorithme peut seulement tenir compte partiellement de l'incertitude des mesures. Pour en tenir compte au maximum, on doit remplacer la méthode du quaternion ou celle du quaternion dual par d'autres méthodes comme le filtrage de Kalman, et ceci entraînera un accroissement important du coût en calcul.

Notre algorithme converge vers le minimum local le plus proche. Il ne convient donc pas pour résoudre des problèmes avec des mouvements importants. Deux solutions possibles ont été proposées dans la section 6.4 : coupler avec une méthode globale ou échantillonner l'espace du mouvement.

Dans notre algorithme, le paramètre $\mathfrak{D}$ a besoin d'être instancié par l'utilisateur. Il indique quand le recalage peut être considéré comme bon. Ce paramètre est important parce qu'il contrôle la mise à jour des appariements. Il a donc un impact sur la convergence de l'algorithme, comme décrit dans la section 4.3. C'est une limite de notre algorithme. Dans notre implémentation, $\mathfrak{D}$ est en relation avec la résolution des données disponibles. Heureusement, le résultat de notre algorithme n'est pas sensible à la valeur de $\mathfrak{D}$. Au lieu d'affecter la valeur de 10 centimètres à $\mathfrak{D}$, il peut aussi bien prendre la valeur de 8 ou 12 centimètres. Une question se pose maintenant [11] : est-il possible de supprimer ce paramètre $\mathfrak{D}$? Le paramètre $\mathfrak{D}$ a été introduit avec le souci que l'estimée initiale peut être médiocre. Si on se fie au résultat fourni par les instruments embarqués du VAP, par exemple la centrale inertielle, autrement dit, si l'estimée initiale est déjà bonne, on peut directement utiliser 3σ (le premier cas dans Sect. 3.3) pour mettre à jour les appariements. Le paramètre $\mathfrak{D}$ n'est donc pas nécessaire.

Nous avons discuté certaines améliorations possibles de notre algorithme : utiliser la vraie distance (Sect. 6.1), tenir compte de l'incertitude (Sect. 6.2), et exploiter un critère symétrique (Sect. 6.3). En retour, il y aura un accroissement considérable du temps de calcul.

L'algorithme que nous avons présenté dans cet article n'est pas recommandé si des primitives (coins, segments, etc.) peuvent être détectées d'une manière précise et robuste, parce que dans ce cas l'approche basée sur des primitives est plus efficace.

REMERCIEMENTS

Ce travail a été partiellement financé par le programme VAP du CNES. Je tiens à exprimer mes remerciements à Bernard Hotz et Catherine Proy pour le soin avec lequel ils ont relu cet article. J'adresse aussi mes sincères remerciements aux experts

[10] Nous avons déjà implémenté un algorithme similaire pour le recalage entre deux ensembles de courbes sans contrainte [28].

[11] Je remercie un des rapporteurs pour avoir soulevé cette discussion.

anonymes pour leur excellent travail de rapporteur de cet article. Leurs commentaires détaillés m'ont beaucoup aidé à améliorer cet article.

BIBLIOGRAPHIE

[1] K.S. ARUN, T.S. HUANG, et S.D. BLOSTEIN. Least-squares fitting of two 3-D point sets. *IEEE Trans. PAMI*, 9(5) :698–700, septembre 1987.

[2] N. AYACHE et O. D. FAUGERAS. Maintaining Representations of the Environment of a Mobile Robot. *IEEE Trans. RA*, 5(6) :804–819, décembre 1989.

[3] P. J. BESL et N. D. McKAY. A method for registration of 3-D shapes. *IEEE Trans. PAMI*, 14(2) :239–256, février 1992.

[4] S.D. BLOSTEIN et T.S. HUANG. Error analysis in stereo determination of a 3-D point position. *IEEE Trans. PAMI*, 9(6) :752–765, novembre 1987.

[5] O.D. FAUGERAS et M. HEBERT. The representation, recognition, et locating of 3D shapes from range data. *Int'l J. Robotics Res.*, 5(3) :27–52, 1986.

[6] P. FUA. A parallel stereo algorithm that produces dense depth maps and preserves image features. *Machine Vision et Applications*, 1992. A paraître.

[7] D. B. GENNERY. Visual terrain matching for a Mars rover. In *Proc. IEEE Conf. Comput. Vision Pattern Recog.*, pages 483–491, San Diego, CA, juin 1989.

[8] D. B. GOLDGOF, T. S. HUANG, et H. LEE. Feature extraction et terrain matching. In *Proc. IEEE Conf. Comput. Vision Pattern Recog.*, pages 899–904, Ann Arbor, Michigan, juin 1988.

[9] M. HEBERT, C. CAILLAS, E. KROTKOV, I. S. KWEON, et T. KANADE. Terrain mapping for a roving planetary explorer. In *Proc. Int'l Conf. Robotics Automation*, pages 997–1002, 1989.

[10] B.K.P. HORN. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 7 :629–642, avril 1987.

[11] B.K.P. HORN et J.G. HARRIS. Rigid body motion from range image sequences. *CVGIP : Image Understanding*, 53(1) :1–13, janvier 1991.

[12] B. KAMGAR-PARSI, J. L. JONES, et A. ROSENFELD. Registration of multiple overlapping range images : Scenes without distinctive features. *IEEE Trans. PAMI*, 13(9) :857–871, septembre 1991.

[13] N. KEHTARNAVAZ et S. MOHAN. A framework for estimation of motion parameters from range images. *Comput. Vision, Graphics Image Process.*, 45 :88–105, 1989.

[14] D.J. KRIEGMAN, E. TRIENDL, et T.O. BINFORD. Stereo vision et navigation in buildings for mobile robots. *IEEE Trans. RA*, 5(6) :792–803, décembre 1989.

[15] I. S. KWEON, M. HEBERT, et T. KANADE. Perception for rugged terrain. 1988 Year End Report "Autonomous Planetary Rover at Carnegie Mellon" CMU-RI-TR-89-3, Carnegie Mellon University, The Robotics Institute, Pittsburgh, PA 15213, janvier 1989.

[16] P. LIANG et J. S. TODHUNTER. Representation et recognition of surface shapes in range images : A differential geometry approach. *Comput. Vision, Graphics Image Process.*, 52 :78–109, 1990.

[17] L. MATTHIES et S. A. SHAFER. Error modeling in stereo navigation. *IEEE J. RA*, 3(3) :239–248, juin 1987.

[18] E. E. MILIOS. Shape matching using curvature processes. *Comput. Vision, Graphics Image Process.*, 47 :203–226, 1989.

[19] F. PREPARATA et M. SHAMOS. *Computational Geometry, An Introduction*. Springer, Berlin, Heidelberg, New-York, 1986.

[20] G. M. RADACK et N. I. BADLER. Local matching of surfaces using a boundary-centered radial decomposition. *Comput. Vision, Graphics Image Process.*, 45 :380–396, 1989.

[21] J. J. RODRíGUEZ et J. K. AGGARWAL. Navigation using image sequence analysis et 3-D terrain matching. In *Proc. Workshop on Interpretation of 3D Scenes*, pages 200–207, Austin, TX, novembre 1989.

[22] R. E. SAMPSON. 3D range sensor-phase shift detection. *Computer*, 20 :23–24, 1987.

[23] R. SZELISKI. Estimating motion from sparse range data without correspondence. In *Proc. Second Int'l Conf. Comput. Vision*, pages 207–216, Tampa, FL, décembre 1988. IEEE.

[24] R. SZELISKI. Bayesian modeling of uncertainty in low-level vision. *Int'l J. Comput. Vision*, 5(3) :271–301, 1990.

[25] M. W. WALKER, L. SHAO, et R. A. VOLZ. Estimating 3-D location parameters using dual number quaternions. *CVGIP : Image Understanding*, 54(3) :358–367, novembre 1991.

[26] D. WALTERS. Selection of image primitives for general-purpose visual processing. *Comput. Vision, Graphics Image Process.*, 37(3) :261–298, 1987.

[27] G. ZHANG et A. WALLACE. Edge classification et depth reconstruction by fusion of range and intensity edge data. In *Proc. Second European Conf. Comput. Vision*, pages 744–748, Santa Margherita Ligure, Italy, mai 1992.

[28] Z. ZHANG. Iterative point matching for registration of free-form curves. Research Report 1658, INRIA Sophia-Antipolis, mars 1992.

[29] Z. ZHANG. On local matching of free-form curves. In *Proc. British Machine Vision Conf.*, pages 347–356, University of Leeds, UK, septembre 1992.

[30] Z. ZHANG et O. FAUGERAS. *3D Dynamic Scene Analysis : A Stereo Based Approach*. Springer, Berlin, Heidelberg, 1992.

[31] Z. ZHANG et O.D. FAUGERAS. Determining motion from 3D line segments : A comparative study. *Image et Vision Computing*, 9(1) :10–19, février 1991.

[32] Z. ZHANG, O.D. FAUGERAS, et N. AYACHE. Analysis of a sequence of stereo scenes containing multiple moving objects using rigidity constraints. In *Proc. Second Int'l Conf. Comput. Vision*, pages 177–186, Tampa, FL, décembre 1988. Aussi comme un chapitre dans R. Kasturi et R.C. Jain (eds), *Computer Vision : Principles*, IEEE computer society press, 1991.

CURRICULUM VITAE

Zhengyou ZHANG est né en Chine le premier avril 1965. Il a obtenu le diplôme d'études supérieures en électronique à l'université de Zhejiang, Chine, en 1985, le diplôme D.E.A. en informatique à l'université de Nancy, France, en 1987, et le doctorat en science (spécialisé informatique) à l'université de Paris-Sud, France, en 1990.

De 1987 à 1990 il était chercheur associé dans le groupe de Vision par Ordinateur et Robotique de l'Institut National de Recherche en Informatique et en Automatique (INRIA), France. Il est maintenant chargé de recherche à l'INRIA. Ses recherches actuelles portent sur la vision par ordinateur, la robotique mobile, l'analyse de scène dynamique et la fusion multisensorielle. Il est l'auteur du livre (avec O. Faugeras) *3D Dynamic Scene Analysis : A Stereo Based Approach* (Springer, Berlin, Heidelberg, 1992).

Manuscrit reçu le 1er octobre 1992.