UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Sophia Antipolis
B.P. 109
06561 Valbonne Cedex
France
Tél.: 93 65 77 77

Rapports de Recherche

N°1658

Programme 4
Robotique, Image et Vision

ITERATIVE POINT MATCHING FOR REGISTRATION OF FREE-FORM CURVES

Zhengyou ZHANG

Mars 1992

Iterative Point Matching for Registration of Free-Form Curves

---

Une méthode itérative basée sur la mise en correspondance de points pour recaler des courbes de forme générale

---

Zhengyou Zhang

INRIA Sophia-Antipolis, 2004 route des Lucioles
BP 109, F-06561 Valbonne Cedex – FRANCE

E-Mail: zzhang@sophia.inria.fr

Abstract

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, but a more precise registration is required for environment modeling. The algorithm described in this report meets this need. Objects are represented by free-form curves, i.e., arbitrary space curves of the type found in practice. A curve is available in the form of a set of chained points. The proposed algorithm is based on iteratively matching points on one curve to the closest points on the other. A least-squares technique is used to estimate 3-D motion from the point correspondences, which reduces the average distance between curves in the two sets. Both synthetic and real data have been used to test the algorithm, and the results show that it is efficient and robust, and yields an accurate motion estimate. The algorithm can be easily extended to solve similar problems such as 2-D curve matching and 3-D surface matching.

Keywords: Free-Form Curve Matching, 3-D registration, Motion Estimation, Dynamic Scene Analysis, 3-D Vision

Résumé

Le recalage de deux ensembles de primitives géométriques est un problème en général très dur et non résolu. Heureusement, dans beaucoup d'applications pratiques, des connaissances a priori simplifient considerablement le problème. Dans la navigation à base de vision, 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 sont représentés au moyen de courbes. Chaque courbe étant représentée par une liste de points chaînés, aucune contrainte n'est a priori imposée sur la forme de la courbe, donc sur celle de l'objet. L'algorithme proposé est basé sur la mise en correspondance itérative de points d'une courbe avec les points les plus proches d'une autre courbe. 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 courbes dans les deux ensembles. Des données de synthèse et des données réelles one été utilisées pour tester cet algorithme. Les résultats montrent qu'il est efficace et robuste, et donne une estimation précise du mouvement. L'algorithm peut être facilement étendu à des problèmes similaires comme le recalage de courbes 2D ou le recalage de surfaces 3D.

Mots clés: Recalage de courbes, mise en correspondance 3D, estimation du mouvement, analyse de scènes dynamiques, vision 3D

Contents

List of Figures

1 Introduction

Geometric matching remains one of the bottlenecks in computer and robot vision, although progress has been made in recent years for some particular applications. There are two main applications: object recognition and visual navigation. The problem in object recognition is to match observed data to a prestored model representing different objects of interest. The problem in visual navigation is to match data observed in a dynamic scene at different instants in order to recover object motions and to interpret the scene. Best and Jain [1], and Chin and Dyer [2] have made two excellent surveys of pre-1985 work on matching in object recognition. Besl [3] surveys the current methods for geometric matching and geometric representations while emphasizing the latter. Most of the previous work focused on polyhedral objects; geometric primitives such as points, lines and planar patches were usually used. This is of course very limited compared with the real world we live in. Recently, curved objects have attracted the attention of many researchers in computer vision. This paper deals with objects represented by curves, particularly free-form curves, i.e., arbitrary space curves of the type found in practice.

A free-form curve is represented by a set of chained points. Several matching techniques for free-form curves have been proposed in the literature. In the first category of techniques, curvature extrema are detected and then used in matching [4]. However, it is difficult to localize precisely curvature extrema [5, 6], especially when the curves are smooth. Very small variations in the curves can change the number of curvature extrema and their positions on the curves. Thus, matching based on curvature extrema is highly sensitive to noise. In the second category, a curve is transformed into a sequence of local, rotationally and translationally invariant features (e.g., curvature and torsion). The curve matching problem is then reduced to a 1-D string matching problem [7, 8, 9]. As more information is used, the methods in this category tend to be more robust than those in the first category. However, these methods are still subject to noise disturbance because they use arclength sampling of the curves to obtain point sets. The arclength itself is sensitive to noise.

The methods cited above exploit global matching criteria in the sense that they can deal with two sets of free-form curves which differ by a large motion/transformation. This ability to deal with large motions is usually es-

sential for applications to object recognition. In many other applications, for example, visual navigation, the motion between curves in successive frames is in general either small (because the maximum velocity of an object is limited and the sample frequency is high) or known within a reasonable precision (because a mobile vehicle is usually equipped with several instruments such as odometric and inertial systems which can provide such information). In the latter case, we can first apply the given estimate of the motion to the first frame to produce an intermediate frame; then the motion between the intermediate frame and the second frame can be considered to be small. In this paper we propose a new method for the registration of curves undergoing small motion.

The key idea underlying our approach is the following. Given that the motion between two successive frames is small, a curve in the first frame is close to the corresponding curve in the second frame. By matching points on the curves in the first frame to their closest points on the curves in the second, we can find a motion that brings the curves in the two frames closer (i.e., the distance between the two curves becomes smaller). Iteratively applying this procedure, the algorithm yields a better and better motion estimate. Interestingly enough, during the preparation of this paper Besl and McKay published a paper in PAMI (issue February 1992) which exploited the same idea [10]. Our work is an independent and much improved treatment. A more detailed comparison is given in Sect. 6.4.

2 Problem Statement

A 3-D (space) curve segment $\mathcal{C}$ is a vector function $\mathbf{x} : [a, b] \rightarrow \mathbb{R}^3$, where a and b are scalar. In computer vision applications, the data of a space curve are available in the form of a set of chained 3-D points from either a stereo algorithm [11] or a range imaging sensor [12]. If we know the type of the curve, we can obtain its description $\mathbf{x}$ by fitting, say, conics to the point data [13, 14]. In this work, we shall use directly the chained points, i.e., we are interested in free-form space curves without regard to particular curve primitives.

The use of chained points is equivalent to a piecewise linear approximation to a curve. Let $\mathbf{x}_{i,j}$ $(j = 1, \ldots, N_i)$ be the N_i chained points on the curve $\mathcal{C}_i$. The approximation error can be made arbitrarily small by increasing N_i

and decreasing the distances $\|\mathbf{x}_{i,j} - \mathbf{x}_{i,j+1}\|$. At every point $\mathbf{x}_{i,j}$, we compute the tangent direction $\mathbf{u}_{i,j}$ which will be used in the matching procedure. It is not necessary in our algorithm to know precisely the tangent directions. We use the simple estimate

$$\mathbf{u}_{i,j} = (\mathbf{x}_{i,j+1} - \mathbf{x}_{i,j-1}) / \|\mathbf{x}_{i,j+1} - \mathbf{x}_{i,j-1}\| \ ,$$

except at the beginning and end points where

$$\mathbf{u}_{i,1} = (\mathbf{x}_{i,2} - \mathbf{x}_{i,1}) / \|\mathbf{x}_{i,1} - \mathbf{x}_{i,1}\| \ ,$$
$$\mathbf{u}_{i,N_i} = (\mathbf{x}_{i,N_i} - \mathbf{x}_{i,N_i-1}) / \|\mathbf{x}_{i,N_i} - \mathbf{x}_{i,N_i-1}\| \ .$$

Given two 3-D frames of a scene observed at two different positions, each containing a set of curves. Let $\mathcal{C}_i$ $(i = 1, \ldots, m)$ and $\mathcal{C}'_k$ $(k = 1, \ldots, n)$ be the curves observed in the first and second frames, respectively. Let $\mathbf{x}_{i,j}$ $(j = 1, \ldots, N_i)$ and $\mathbf{x}'_{k,l}$ $(l = 1, \ldots, N_k)$ be the points on the curves $\mathcal{C}_i$ and $\mathcal{C}'_j$, respectively. The objective is to find the motion between the two frames, i.e., $\mathbf{R}$ for rotation and $\mathbf{t}$ for translation, such that the following criterion

$$\mathcal{F}(\mathbf{R}, \mathbf{t}) = \sum_{i=1}^{m} \sum_{j=1}^{N_i} p_{i,j} \, d^2(\mathbf{R}\mathbf{x}_{i,j} + \mathbf{t}, \mathcal{C}'_k) + \sum_{k=1}^{n} \sum_{l=1}^{N_k} q_{k,l} \, d^2(\mathbf{R}^T\mathbf{x}'_{k,l} - \mathbf{R}^T\mathbf{t}, \mathcal{C}_i) \tag{1}$$

is minimized, where $d(\mathbf{x}, \mathcal{C})$ denotes the distance of the point $\mathbf{x}$ to the curve $\mathcal{C}$ (to be defined below), $p_{i,j}$ (resp. $q_{k,l}$) takes value 1 if the point $\mathbf{x}_{i,j}$ (resp. $\mathbf{x}'_{k,l}$) can be matched to a point on the curve $\mathcal{C}'_k$ in the second frame (resp. $\mathcal{C}_i$ in the first frame) and takes value 0 otherwise. Of course, the minimization of Eq. (1) must be accompanied by the maximization of

$$\sum_{i=1}^{m} \sum_{j=1}^{N_i} p_{i,j} + \sum_{k=1}^{n} \sum_{l=1}^{N_k} q_{k,l} \ .$$

If not, the trivial solution of Eq. (1) is achieved when $p_{i,j} = q_{k,l} = 0$ for all i, j, k and l.

The above criteria are symmetric in the sense that neither of the two frames prevails over the other. To economize computation, we shall only use the first part of the right hand side of Eq. (1), together with the maximization of $\sum_{i=1}^{m} \sum_{j=1}^{N_i} p_{i,j}$. In other words, the objective function to be minimized is

$$\mathcal{F}(\mathbf{R}, \mathbf{t}) = \frac{1}{\sum_{i=1}^{m} \sum_{j=1}^{N_i} p_{i,j}} \sum_{i=1}^{m} \sum_{j=1}^{N_i} p_{i,j} \, d^2(\mathbf{R}\mathbf{x}_{i,j} + \mathbf{t}, \mathcal{C}'_k) \ . \tag{2}$$

This modification only affects a little bit the accuracy of the final motion estimation. It also slows down a little bit the convergence, in the sense of the number of iterations, of the iterative algorithm described in the next section, but speeds up the whole process.

Furthermore, we assume the motion between the two frames is small or approximately known. In the latter case, we can first apply the approximate estimate of the motion between the two frames to the first one to produce an intermediate frame; then the motion between the intermediate frame and the second frame can be considered to be small. *Small* depends essentially on the scene of interest. If the scene is dominated by a repetitive pattern, the motion should not be bigger than half of the pattern distance. For example, in the situation illustrated in Fig. 1, our algorithm will converge to a local minimum. In this case, other methods based on more global criteria, such as those cited in the introduction section, must be used to recover a rough estimation of the motion. The algorithm described in this paper can then be used to obtain a precise motion estimate.

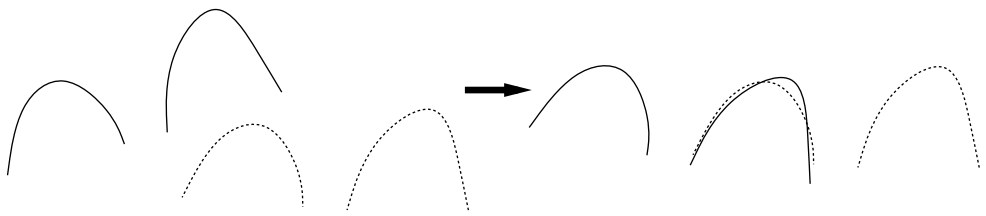


Fig. 1. Our algorithm exploits a local matching technique, and converges to the closest local minimum, which is not necessarily the optimal one

3 Iterative Pseudo Point Matching Algorithm

We describe in this section an iterative algorithm for curve registration by matching points in the first frame, after applying the previously recovered

motion estimate $(\mathbf{R}, \mathbf{t})$, with their closest points in the second. A least-squares estimation reduces the average distance between curves in the two frames. As a point in one frame and its closest point in the other do not necessarily correspond to a single point in space, several iterations are indispensable. Hence the name of the algorithm.

3.1 Finding Closest Points

Let us first define the distance $d(\mathbf{x}, \mathcal{C}'_k)$ between point $\mathbf{x}$ and curve $\mathcal{C}'_k$, which is used in Eq. (2), the criterion defined in the last section. If $\mathcal{C}'_k$ is a parametric curve $(\mathbf{x}'_k : [a, b] \to \mathbb{R}^3)$, then

$$d(\mathbf{x}, \mathcal{C}'_k) = \min_{u \in [a,b]} d(\mathbf{x}, \mathbf{x}'_k(u)) \ , \tag{3}$$

where $d(\mathbf{x}_1, \mathbf{x}_2)$ is the Euclidean distance between the two points $\mathbf{x}_1$ and $\mathbf{x}_2$, i.e., $d(\mathbf{x}_1, \mathbf{x}_2) = \|\mathbf{x}_1 - \mathbf{x}_2\|$. In our case, $\mathcal{C}'_k$ is given as a set of chained points $\mathbf{x}'_{k,l}$ $(l = 1, \ldots, N_k)$. We simply define

$$d(\mathbf{x}, \mathcal{C}'_k) = \min_{l \in \{1,\ldots,N_k\}} d(\mathbf{x}, \mathbf{x}'_{k,l}) \ . \tag{4}$$

See the next section for more discussions on the distance.

The closest point $\mathbf{y}$ in the second frame to a given point $\mathbf{x}$ is the one satisfying

$$d(\mathbf{x}, \mathbf{y}) = \min_{k \in \{1,\ldots,n\}} d(\mathbf{x}, \mathcal{C}'_k) = \min_{k \in \{1,\ldots,n\}} \min_{l \in \{1,\ldots,N_k\}} d(\mathbf{x}, \mathbf{x}'_{k,l}) \ .$$

The worst case cost of finding the closest point is $O(N_k^n)$, where N_k^n is the total number of points in the second frame. The total cost while performing the above computation for each point in the first frame is $O(N_i^m N_k^n)$, where N_i^m is the total number of points in the first frame. The use of k-D trees can considerably speed up this process, see Sect. 4.1.

3.2 Pseudo Point Matching

For each point $\mathbf{x}$ we can always find a closest point $\mathbf{y}$. However, because there are some spurious points in both frames due to sensor capability, or

because some points visible in one frame are not in the other due to sensor/object motion, it probably does not make any sense to pair $\mathbf{x}$ with $\mathbf{y}$. Many constraints can be imposed to remove such spurious pairings. For example, distance continuity along a curve, which is similar to the figural continuity in stereo matching [15, 16], should be very useful to discard the false matches. These constraints are not incorporated in our algorithm in order to maintain the algorithm in its simplest form. Instead, we impose the following two simple constraints, which are all unary.

The first is the maximum tolerance for distance. If the distance between a point $\mathbf{x}_{i,j}$ and its closest one $\mathbf{y}_{i,j}$, denoted by $d(\mathbf{x}_{i,j}, \mathbf{y}_{i,j})$, is bigger than the maximum tolerable distance $D_{\max}$, then we set $p_{i,j} = 0$ in Eq. (2), i.e., we cannot pair a reasonable point in the second frame with the point $\mathbf{x}_{i,j}$. This constraint is easily justified for we know that the motion between the two frames is small and hence the distance between two points reasonably paired cannot be very big. In our algorithm, $D_{\max}$ is set adaptively and in a robust manner during each iteration by analyzing distances statistics. See Sect. 3.3.

The second is the orientation consistency. It can be easily shown that the angle between the tangent of point $\mathbf{x}$ and that of its closest point $\mathbf{y}$ can not go beyond the rotation angle between the two frames [17]. Therefore, we can impose that the angle between the tangents of two paired points should not be bigger than a prefixed value Θ, which is the maximum of the rotation angle expected between the two frames. In our implementation, we set $\Theta = 60°$ to take into account noise effect in the tangent computation. If the tangents can be precisely computed, Θ can be set to a smaller value. This constraint is especially useful when the motion is relatively big.

3.3 Updating the Matching

Instead of using all matches recovered so far, we exploit a robust technique to discard several of them by analyzing the statistics of the distances. To this end, one parameter, denoted by $\mathfrak{D}$, needs to be set by the user, which indicates when he considers the registration between two frames is good. See Sect. 4.3 for the choice of the value $\mathfrak{D}$.

Let $D^{\mathrm{I}}_{\max}$ denote the maximum tolerable distance in iteration I. At this point, each point in the first frame (after applying the previously recovered motion) whose distance to its closest point is less than $D^{\mathrm{I-1}}_{\max}$ is retained, together with its closest point and their distance. Let $\{\mathbf{x}_i\}$, $\{\mathbf{y}_i\}$, and $\{d_i\}$

be the resulting sets of original points, closest points, and their distances after the pseudo point matching, and let N be the cardinal of the sets. Now compute the mean μ and the sample deviation σ of the distances, which are given by

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

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

Depending on the value of μ, we adaptively set the maximum tolerable distance $D^{\mathrm{I}}_{\max}$ as shown below*:

if $\mu < \mathfrak{D}$ `/* the registration is quite good */`
  $D^{\mathrm{I}}_{\max} = \mu + 3\sigma$;
else if $\mu < 3\mathfrak{D}$ `/* the registration is still good */`
  $D^{\mathrm{I}}_{\max} = \mu + 2\sigma$;
else if $\mu < 6\mathfrak{D}$ `/* the registration is not too bad */`
  $D^{\mathrm{I}}_{\max} = \mu + \sigma$;
else `/* the registration is really bad */`
  $D^{\mathrm{I}}_{\max} = \xi$;
endif

Here, ξ is the median of all the distances. That is, the number of d_i's less than ξ is approximately equal to the number of d_i's larger than ξ.

At this point, we use the newly set $D^{\mathrm{I}}_{\max}$ to update the matching previously recovered: a paring between $\mathbf{x}_i$ and $\mathbf{y}_i$ is removed if their distance d_i is bigger than $D^{\mathrm{I}}_{\max}$. The pairings remained are used to compute the motion between the two frames, as to be described below.

Because $D_{\max}$ is adaptively set based on the statistics of the distances, our algorithm is rather robust to relatively big motion and to gross outliers (as to be shown in the experiment section). For example, when the registration is really bad, only half of the originally recovered matches are retained. Even if there remain several false matches in the retained set, the use of least-squares

*Here we assume the distribution of distances is approximately Gaussian when the registration is good. This has been confirmed by experiments. A typical histogram is shown in Fig. 2.

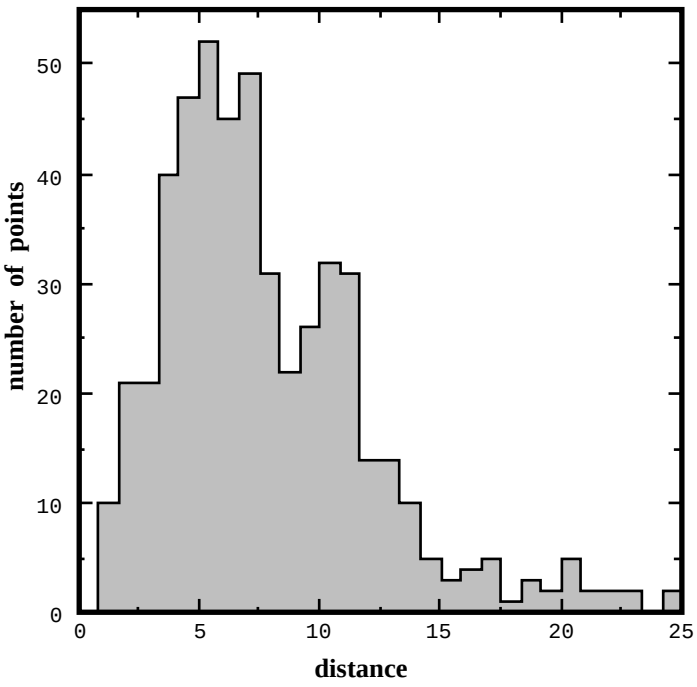


Fig. 2. A histogram of distances

technique yields still a reasonable motion estimate, which is sufficient for the algorithm to converge to the correct solution.

3.4 Computing Motion

At this point, we have a set of 3-D points which have been reasonably paired with a set of closest points, denoted respectively by $\{\mathbf{x}_i\}$ and $\{\mathbf{y}_i\}$. Let N be the number of pairs. Because N is usually much greater than 3 (three points are the minimum for the computed rigid motion to be unique), it is necessary to devise a procedure for computing the motion by minimizing the following mean-squares objective function

$$\mathcal{F}(\mathbf{R}, \mathbf{t}) = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{R}\mathbf{x}_i + \mathbf{t} - \mathbf{y}_i\|^2 \ , \tag{5}$$

which is the direct result of Eq. (2) with the definition of distance given by Eq. (4). Any optimization method, such as steepest descent, conjugate

gradient, or complex, can be used to find the least-squares rotation and translation. Fortunately, several much more efficient algorithms exist for solving this particular problem. In the following, the dual number quaternion method [18] is summarized.

A quaternion $\mathbf{q}$ can be considered as being either a 4-D vector $[q_1, q_2, q_3, q_4]^T$ or a pair $(\breve{\mathbf{q}}, q_4)$ where $\breve{\mathbf{q}} = [q_1, q_2, q_3]^T$. A dual quaternion $\hat{\mathbf{q}}$ consists of two quaternions $\mathbf{q}$ and $\mathbf{s}$, i.e.,

$$\hat{\mathbf{q}} = \mathbf{q} + \varepsilon \mathbf{s} \ , \tag{6}$$

where a special multiplication rule for ε is defined by $\varepsilon^2 = 0$. Two important matrix functions of quaternions are defined as

$$\mathbf{Q}(\mathbf{q}) = \begin{bmatrix} q_4\mathbf{I} + \mathbf{K}(\breve{\mathbf{q}}) & \breve{\mathbf{q}} \\ -\breve{\mathbf{q}}^T & q_4 \end{bmatrix} \ , \tag{7}$$

$$\mathbf{W}(\mathbf{q}) = \begin{bmatrix} q_4\mathbf{I} - \mathbf{K}(\breve{\mathbf{q}}) & \breve{\mathbf{q}} \\ -\breve{\mathbf{q}}^T & q_4 \end{bmatrix} \ , \tag{8}$$

where $\mathbf{I}$ is the identity matrix, and $\mathbf{K}(\breve{\mathbf{q}})$ is the skew-symmetric matrix defined as

$$\mathbf{K}(\breve{\mathbf{q}}) = \begin{bmatrix} 0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_2 & q_1 & 0 \end{bmatrix} \ .$$

A 3-D rigid motion can be represented by a dual quaternion $\hat{\mathbf{q}}$ satisfying the following two constraints:

$$\mathbf{q}^T\mathbf{q} = 1 \qquad \text{and} \qquad \mathbf{q}^T\mathbf{s} = 0 \ . \tag{9}$$

Thus, we have still six independent parameters for representating a 3-D motion. The rotation matrix $\mathbf{R}$ can be expressed as

$$\mathbf{R} = (q_4^2 - \breve{\mathbf{q}}^T\breve{\mathbf{q}})\mathbf{I} + 2\breve{\mathbf{q}}\breve{\mathbf{q}}^T + 2q_4\mathbf{K}(\breve{\mathbf{q}}) \ , \tag{10}$$

and the translation vector $\mathbf{t} = \breve{\mathbf{p}}$, where $\breve{\mathbf{p}}$ is the vector part of the quaternion $\mathbf{p}$ given by

$$\mathbf{p} = \mathbf{W}(\mathbf{q})^T\mathbf{s} \ . \tag{11}$$

The scalar part p_4 of $\mathbf{p}$ is always zero.

A 3-D vector $\mathbf{x}$ is identified with the quaternion $(\mathbf{x}, 0)$, and we shall also use $\mathbf{x}$ to represent its corresponding quaternion if there is no ambiguity in the context. It can then be easily shown that

$$\mathbf{R}\mathbf{x} + \mathbf{t} = \mathbf{W}(\mathbf{q})^T\mathbf{s} + \mathbf{W}(\mathbf{q})^T\mathbf{Q}(\mathbf{q})\mathbf{x} \ .$$

Thus the objective function Eq. (5) can be written as a quadratic function of $\mathbf{q}$ and $\mathbf{s}$

$$\mathcal{F} = \frac{1}{N}[\mathbf{q}^T C_1 \mathbf{q} + N\mathbf{s}^T\mathbf{s} + \mathbf{s}^T C_2 \mathbf{q} + \text{const.}] \ , \tag{12}$$

where

$$C_1 = -2\sum_{i=1}^{N} \mathbf{Q}(\mathbf{y}_i)^T\mathbf{W}(\mathbf{x}_i) = -2\sum_{i=1}^{N} \begin{bmatrix} \mathbf{K}(\mathbf{y})\mathbf{K}(\mathbf{x}) + \mathbf{y}\mathbf{x}^T & -\mathbf{K}(\mathbf{y})\mathbf{x} \\ -\mathbf{y}^T\mathbf{K}(\mathbf{x}) & \mathbf{y}^T\mathbf{x} \end{bmatrix} , \tag{13}$$

$$C_2 = 2\sum_{i=1}^{N}[\mathbf{W}(\mathbf{x}_i) - \mathbf{Q}(\mathbf{y}_i)] = 2\sum_{i=1}^{N} \begin{bmatrix} -\mathbf{K}(\mathbf{x}) - \mathbf{K}(\mathbf{y}) & \mathbf{x} - \mathbf{y} \\ -(\mathbf{x} - \mathbf{y})^T & 0 \end{bmatrix} , \tag{14}$$

$$\text{const.} = \sum_{i=1}^{N}(\mathbf{x}_i^T\mathbf{x}_i + \mathbf{y}_i^T\mathbf{y}_i) \ . \tag{15}$$

By adjoining the constraints (Eq. (9)), the optimal dual quaternion is obtained by minimizing

$$\mathcal{F}' = \frac{1}{N}[\mathbf{q}^T C_1 \mathbf{q} + N\mathbf{s}^T\mathbf{s} + \mathbf{s}^T C_2 \mathbf{q} + \text{const.} + \lambda_1(\mathbf{q}^T\mathbf{q} - 1) + \lambda_2(\mathbf{s}^T\mathbf{q})] \ , \tag{16}$$

where λ_1 and λ_2 are Lagrange multipliers. Taking the partial derivatives gives

$$\frac{\partial \mathcal{F}'}{\partial \mathbf{q}} = \frac{1}{N}\left[(C_1 + C_1^T)\mathbf{q} + C_2^T\mathbf{s} + 2\lambda_1\mathbf{q} + \lambda_2\mathbf{s}\right] = 0 \ , \tag{17}$$

$$\frac{\partial \mathcal{F}'}{\partial \mathbf{s}} = \frac{1}{N}\left[2N\mathbf{s} + C_2\mathbf{q} + \lambda_2\mathbf{q}\right] = 0 \ . \tag{18}$$

Multiplying Eq. (18) by $\mathbf{q}$ gives $\lambda_2 = -\mathbf{q}^T C_2 \mathbf{q} = 0$, because C_2 is skew-symmetric. Thus $\mathbf{s}$ is given by

$$\mathbf{s} = -\frac{1}{2N} C_2 \mathbf{q} \ . \tag{19}$$

Substituting these into Eq. (17) yields

$$A\mathbf{q} = \lambda_1 \mathbf{q} \ , \tag{20}$$

where

$$A = \frac{1}{2}\left[\frac{1}{2N} C_2^T C_2 - C_1 - C_1^T\right] \ . \tag{21}$$

Thus, the quaternion $\mathbf{q}$ is an eigenvector of the matrix A and λ_1 is the corresponding eigenvalue. Substituting the above result back into Eq. (16) gives

$$\mathcal{F}' = \frac{1}{N}(\text{const.} - \lambda_1) \ . \tag{22}$$

The error is thus minimized if we select the eigenvector corresponding to the largest eigenvalue.

Having computed $\mathbf{q}$, the rotation matrix $\mathbf{R}$ is computed from Eq. (10). The dual part $\mathbf{s}$ is computed from Eq. (19) and the translation vector $\mathbf{t}$ can then be solved from Eq. (11).

Other efficient algorithms include quaternion method [19] and singular value decomposition [20]. We have implemented both the quaternion method and the dual number quaternion one. They yield exactly the same motion estimate. One advantage of the dual quaternion method is that the matrices C_1 and C_2 can be incrementally computed. Following [18], they then exhibit better performance for the translation than the singular value decomposition method.

3.5 Summary

We can now summarize the iterative pseudo point matching algorithm as follows:

- **input:** Two 3D frames containing m curves $\mathcal{C}_i$ and n curves $\mathcal{C}'_k$, respectively. Each curve $\mathcal{C}$ is a set of chained 3D points $\mathbf{x}_j$.
- **output:** The optimal motion between the two frames.
- **procedure:**

a) **initialization**
$D^0_{\max}$ is set to $20\mathfrak{D}$, which implies that every point in the first frame whose distance to its closest point in the second frame is bigger than $D^0_{\max}$ is discarded from consideration during the first iteration. The number *20* is not crucial in the algorithm, and can be replaced by a larger one.

b) **preprocessing**
(i) Compute the tangent at each point of the first frame.
(ii) Compute the tangent at each point of the second frame.
(iii) Build the k-D tree representation of the second frame (see Sect. 4.1).

c) **iteration** until convergence of the computed motion
(i) Finding the closest points satisfying the distance and orientation constraints, as described in Sect. 3.2.
(ii) Update the matching through statistic analysis of distances, as described in Sect. 3.3.
(iii) Compute the motion between the two frames from the updated matches, as decribed in Sect. 3.4.
(iv) Apply the motion to all points and their tangents in the first frame.

Several remarks should be made here. The construction and the use of k-D trees for finding closest points will be described in the next section. The motion is computed between the original points in the first frame and the points in the second frame. Therefore, the final motion given by the algorithm represents the transformation between the original first frame and the second frame. The iteration-termination condition is defined as the change in the motion estimate between two successive iterations. The change in translation at iteration I is defined as

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

To measure the change in rotation, we use the rotation axis representation, which is a 3-D vector, denoted by $\mathbf{r}$. Let $\theta = \|\mathbf{r}\|$ and $\mathbf{n} = \mathbf{r}/\|\mathbf{r}\|$, the relation between $\mathbf{r}$ and the quaternion $\mathbf{q}$ is

$$\mathbf{q} = \begin{bmatrix} \sin(\theta/2)\mathbf{n} \\ \cos(\theta/2) \end{bmatrix} .$$

We do not use the quaternions because their difference does not make much sense. We then define the change in rotation at iteration I as

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

We terminate the iteration when both $\delta\mathbf{r}$ and $\delta\mathbf{t}$ are less than 1%.

4 Practical Considerations

In this section, we consider several important aspects in practice, including search for closest points, curve sampling, choice of the parameter $\mathfrak{D}$, and uncertainty.

4.1 Search for Closest Points

As can be observed in the last section, the search for the closest point to a given point is $O(N)$ in time, where $N = N_k^n$ is the total number of points in the second frame. Several methods exist to speed up the search process, including bucketing techniques and k-D trees (abbreviation for *k-dimensional binary search tree*). We have chosen k-D trees, because curves we have in form of chained points are sparse in space. It is not efficient enough to use bucketing techniques because only a few buckets would contain many points, and many others nothing.

The k-D tree is a generalization of bisection in one dimension to k dimensions [21]. In our case, $k = 3$. A 3-D tree is constructed as follows. First choose a plane parallel to yz-plane passing through a data point P to cut the whole space into two (generalized) rectangular parallelepipeds[†] such that there are approximately equal numbers of points on either side of the cut. We obtain then a left son and a right son. Next, each son is further split by a plane parallel to xz-plane such that there are approximately equal numbers of points on either side of the cut, and we obtain a left grandson and a right one. We continue splitting each grandson by choosing a plane parallel to xy-plane, and so on, letting at each step the direction of the cutting plane alternate between yz-, xz- and xy-plane. This splitting process

[†] A generalized rectangular parallelepiped is possibly an infinite volume.

stops when we reach a rectangular parallelepiped not containing any point; the corresponding node is a leaf of the tree. A k-D tree can be constructed in $O(N \log N)$ time with $O(N)$ storage, which are both optimal [21].

We now investigate the use of the 3-D tree in searching for closest points. In fact, given a point $\mathbf{x}$ in the first frame, instead of searching for its closest point in the second frame, we search for all points whose distances to $\mathbf{x}$ is within the maximum tolerable distance $D_{\max}$. Thus, for each point $\mathbf{x}$ we have a list of candidates arranging in increasing distance order, the list being possibly empty. This provides the user with the flexibility to implement more sophisticated method, say relaxation, at the step of pseudo point matching. The search algorithm is a recursive procedure. More formally, a node v of the 3-D tree T is characterized by two items $(P(v), t(v))$. Point $P(v)$ is the point through which the space is cut into two. The parameter $t(v)$, taking value 0, 1, or 2, indicates whether the cutting plane is parallel to yz-, xz-, or xy-plane. The algorithm accumulates the retrieved points in a list U external to the procedure, initialized as empty. The search for the closest points to $\mathbf{x}$ is effected by calling `SEARCH`(`root`$(T), \mathbf{x}, D_{\max}$) of the following procedure:

- **input:** a point $\mathbf{x}$, a 3-D tree T, and the maximum tolerable distance $D_{\max}$.
- **output:** a list U containing all points whose distances to $\mathbf{x}$ is within $D_{\max}$.
- **procedure:** `SEARCH`$(v, \mathbf{x}, D_{\max})$
  - — **if** $(v ==$ leaf$)$ **return** ;
  - — $c_1 = \mathbf{x}[t(v)]$;
  - — $c_2 = P(v)[t(v)]$; `/* `c_2 `has been used to cut the space */`
  - — **if** $(|c_1 - c_2| \leq D_{\max})$ **then if** $(\|\mathbf{x} - P(v)\| \leq D_{\max})$ **then** $U \Leftarrow P(v)$;
  - — **if** $(c_1 - D_{\max} < c_2)$ **then** `SEARCH`$($`leftson`$(v), \mathbf{x}, D_{\max})$;
  - — **if** $(c_2 - D_{\max} < c_1)$ **then** `SEARCH`$($`rightson`$(v), \mathbf{x}, D_{\max})$;

Unfortunately, the worst-case search time is $O(N^{2/3})$ with the 3-D tree method (see [21, pp.77]). Other more efficient algorithms exist, such as a direct access method, but they require much more storage. In practice, we observed good performance with 3-D trees. We found that the search time depends heavily on $D_{\max}$. When $D_{\max}$ is small, the search can be performed very fast (see the experiment section). As we update $D_{\max}$ during each iteration, it becomes quite small after a few iterations.

4.2 Curve Sampling

As described earlier, we use chained points to represent a free-form curve, which is equivalent to a piecewise linear approximation. We have also assumed that the approximation error was small enough. However, the algorithm developed in the last section is based on the use of a simplified, instead of real, definition of the distance between a point and a curve (see Eq. (4)). That is, we use the minimum of all distances from a given point to each *sample* point of the curve. Different sampling of a curve (even the approximation error is negligible) does affect the final estimation of the motion. Take a simple example as shown in Fig. 3. The curve consists of two line segments (Fig. 3a). The sampling in the first frame consists of three points as indicated by the crosses in Fig. 3a. We have two samplings in the second frame. The first sampling consists of three points as indicated by dark dots, and the second sampling consists of five points by adding two additional ones (indicated by empty dots) to the first sampling, as shown in Fig. 3a. The motion result between the two frames with the first sampling is shown in Fig. 3b, and that with the second sampling, in Fig. 3c. Clearly, more samples, better results. To solve the problem resulted from sampling, we should ideally use the real distance definition (Eq. (3)) by considering all points (referred as *curve* points) on the line segments composing the curve, and use the closest *curve* points instead of the closest *sample* points. However, we lose the efficiency achieved with sample points.

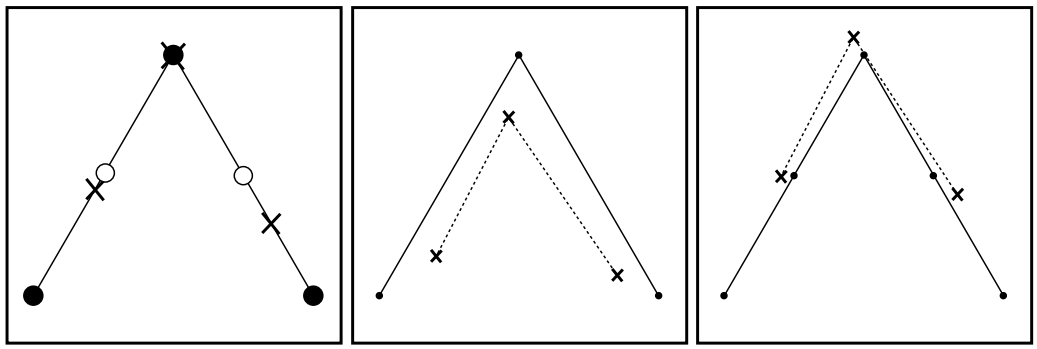


Fig. 3. Influence of curve sampling on motion estimation

Now we describe two methods to overcome the above problem while main-

taining the efficiency of the algorithm. The first consists in simply increasing the number of sample points. The more the number of sample points, the less the sampling will affect the final motion estimation. However, this causes two problems. The first is the increase in the memory required. The second is the increase in the search time because we increase also the size of the k-D tree. Thus a tradeoff must be found. It is clear that the effect of sampling on the final motion estimation is approximately less than a half of the average sample interval, because in the case of a perfect registration (as shown, for example, in Fig. 5) the distance between a point in the first frame and its match in the second frame is not bigger than half of the corresponding interval in the second frame. Therefore, let e (10 mm in our implementation) be the tolerable effect of sampling, then if two neighboring sample points are more than $2e$ away from each other, we add, between them, as many points as necessary such that the distance between every two neighboring points is less than or equal to $2e$. This process is only needed for curves in the second frame, and can be done in the preprocessing stage.

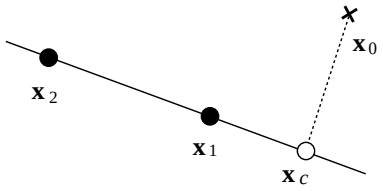


Fig. 4. Computing the closest point

The second method is an approximation to the real definition of the distance between a point and a curve (Eq. (3)). As described in Sect. 4.1, for a given point $\mathbf{x}_0$, we obtain a list of points in the second frame whose distances to $\mathbf{x}_0$ are all less than $D_{\max}$. These points are arranged in increasing distance order. Let $\mathbf{x}_1$ and $\mathbf{x}_2$ be the first and second points in the list. Instead of taking $\mathbf{x}_1$ as the match of $\mathbf{x}_0$, we can assign a virtual point, to be described below, to $\mathbf{x}_0$. The virtual point is $\mathbf{x}_c$, which is the closest point on the line passing through $\mathbf{x}_1$ and $\mathbf{x}_2$ (see Fig. 4). It satisfies the following relations

$$(\mathbf{x}_c - \mathbf{x}_0) \cdot (\mathbf{x}_2 - \mathbf{x}_1) = 0 \ ,$$
$$(\mathbf{x}_c - \mathbf{x}_1) \times (\mathbf{x}_2 - \mathbf{x}_1) = 0 \ ,$$

where $\cdot$ and $\times$ denote the inner product and cross product of two vectors. Explicitly, it is

$$\mathbf{x}_c = \mathbf{x}_1 + \frac{(\mathbf{x}_0 - \mathbf{x}_1) \cdot (\mathbf{x}_2 - \mathbf{x}_1)}{\|\mathbf{x}_2 - \mathbf{x}_1\|^2}(\mathbf{x}_2 - \mathbf{x}_1) \ .$$

Before doing that, we should ensure that $\mathbf{x}_1$ and $\mathbf{x}_2$ are neighbors.

Each of the two methods has its own merit and drawback. We can expect to obtain more precise estimation of motion with the second method. However, we must compute the virtual point for each point in the first frame and during each iteration, while with the first method the additional computation is only performed in the preprocessing stage. We have implemented the first method because we can obtain an estimation with required precision.

4.3 Choice of the Parameter $\mathfrak{D}$

The only parameter needed to be supplied by the user is $\mathfrak{D}$, which indicates when the registration between two frames can be considered to be good. In other words, the value of $\mathfrak{D}$ should correspond to the expected average distance when the registration is good. When the motion is big, $\mathfrak{D}$ should not be very small. Because we set $D^0_{\max} = 20\mathfrak{D}$, if $\mathfrak{D}$ is very small we cannot find any matches in the first iteration and of course we cannot improve the motion estimate. (A solution to this is to set $D^0_{\max}$ bigger, say $30\mathfrak{D}$).

The value of $\mathfrak{D}$ has an impact on the convergence of the algorithm. If $\mathfrak{D}$ is smaller than necessary, then more iterations are required for the algorithm to converge because many good matches will be discarded at the step of matching update. On the other hand, if $\mathfrak{D}$ is much bigger than necessary, it is possible for the algorithm not to converge to the correct solution because possibly many false matches will not be discarded. Thus, to be prudent, it is better to choose a small value for $\mathfrak{D}$.

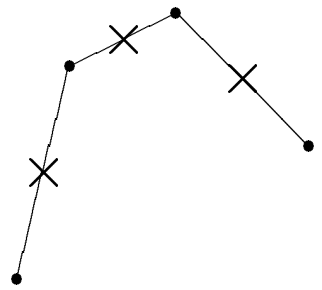


Fig. 5. Illustration of a perfect registration to show how to choose $\mathfrak{D}$

We have worked out a better solution to $\mathfrak{D}$ instead of an ad hoc choice. Let $\bar{D}$ be the average distance between successive points in the second frame, that is

$$\bar{D} = \frac{\sum_{k=1}^{n} \sum_{l=1}^{N_k - 1} \|\mathbf{x}_{k,l} - \mathbf{x}_{k,l+1}\|}{\sum_{k=1}^{n} (N_k - 1)} .$$

Consider a perfect registration shown in Fig. 5. Points from the first frame are marked by a cross and those from the second, by a dot. Assume that a

cross is located in the middle of two dots. Then in this case, the mean μ of the distances between two sets of points is equal to $\bar{D}/2$. Therefore, we can expect $\mu > \bar{D}/2$ when the registration is not perfect. In our implementation, we set $\mathfrak{D} = \bar{D}$ which gives us satisfactory results.

4.4 Uncertainty

The importance of explicitly estimating and manipulating uncertainty is now well recognized by the computer vision and robotics community [22, 23, 24, 25, 26]. This is extremely important when the data available have different uncertainty distribution for example in stereo where uncertainty increases significantly with depth. We have shown in [27] that accounting for uncertainty in motion estimation (via, e.g., a Kalman filter) yields much better results.

For computational tractability and as a reasonable approximation, the uncertainty in a 3-D point reconstructed from stereo is usually modeled as Gaussian; that is, it is characterized by a 3-D position vector and a 3×3 covariance matrix. The algorithm for motion computation described in Sect. 3.4 is very efficient. However, it assumes each point has equal uncertainty. And unfortunately it is difficult to extend it to fully take uncertainty into account. To fully take uncertainty into account, we can use for example Kalman filtering techniques which have been widely and successfully applied to solve quite a number of vision problems [28].

While I was saying difficult to *fully* take uncertainty into account in the algorithm described in Sect. 3.4, I do mean we can extend it to *partially* take uncertainty into account. Indeed, we can associate, to each pairing between the two frames, a weighting factor. Instead of minimizing Eq. (5), we compute $\mathbf{R}$ and $\mathbf{t}$ by minimizing the following function

$$\mathcal{F}(\mathbf{R}, \mathbf{t}) = \frac{1}{N} \sum_{i=1}^{N} w_i \|\mathbf{R}\mathbf{x}_i + \mathbf{t} - \mathbf{y}_i\|^2 \,, \tag{23}$$

where w_i is the positive weighting factor associated with the pairing between $\mathbf{x}_i$ and $\mathbf{y}_i$. Under the dual quaternion representation, the objective function $\mathcal{F}(\mathbf{R}, \mathbf{t})$ can be written as a quadratic function of $\mathbf{q}$ and $\mathbf{s}$

$$\mathcal{F} = \frac{1}{N}[\mathbf{q}^T C_1 \mathbf{q} + W \mathbf{s}^T \mathbf{s} + \mathbf{s}^T C_2 \mathbf{q} + \text{const.}] \,, \tag{24}$$

where

$$C_1 = -2\sum_{i=1}^{N} w_i \mathbf{Q}(\mathbf{y}_i)^T \mathbf{W}(\mathbf{x}_i) \ , \tag{25}$$

$$C_2 = 2\sum_{i=1}^{N} w_i [\mathbf{W}(\mathbf{x}_i) - \mathbf{Q}(\mathbf{y}_i)] \ , \tag{26}$$

$$W = \sum_{i=1}^{N} w_i \ , \tag{27}$$

$$\text{const.} = \sum_{i=1}^{N} w_i (\mathbf{x}_i^T \mathbf{x}_i + \mathbf{y}_i^T \mathbf{y}_i) \ . \tag{28}$$

We can observe the similarity between Eq. (24) and Eq. (12). The quaternion $\mathbf{q}$ is the eigenvector of the matrix

$$A = \frac{1}{2}\left[\frac{1}{2W} C_2^T C_2 - C_1 - C_1^T\right]$$

corresponding to the largest eigenvalue. The quaternion $\mathbf{s}$ is then given by

$$\mathbf{s} = -\frac{1}{2W} C_2 \mathbf{q} \ .$$

The weighting factor w_i should be related to the uncertainty of $\mathbf{R}\mathbf{x}_i + \mathbf{t} - \mathbf{y}_i$. Let $\Lambda_{\mathbf{x}_i}$, $\Lambda_{\mathbf{y}_i}$, and Λ_i be the covariance matrices of $\mathbf{x}_i$, $\mathbf{y}_i$, and $\mathbf{R}\mathbf{x}_i + \mathbf{t} - \mathbf{y}_i$. $\Lambda_{\mathbf{x}_i}$ and $\Lambda_{\mathbf{y}_i}$ are given by the sensing system, e.g., stereo. Λ_i is computed as

$$\Lambda_i = \mathbf{R}\Lambda_{\mathbf{x}_i}\mathbf{R}^T + \Lambda_{\mathbf{y}_i} \ ,$$

where $\mathbf{R}$ takes the rotation matrix computed during a previous iteration as an approximation. The trace of Λ_i roughly indicates the magnitude of the uncertainty of $\mathbf{R}\mathbf{x}_i + \mathbf{t} - \mathbf{y}_i$. Therefore, we choose w_i as

$$w_i = \frac{1}{\mathrm{tr}(\Lambda_i)} = \frac{1}{\mathrm{tr}(\Lambda_{\mathbf{x}_i}) + \mathrm{tr}(\Lambda_{\mathbf{y}_i})} \ .$$

Thus, the weighting factor is independent of the rotation.

4.5 Coarse-to-Fine Strategy

As to be shown in the next section, we find fast convergence of the algorithm during the first few iterations that slows down as it approaches the local minimum. We find also that more search time is required during the first few iterations because the search space is larger at the beginning, as described in Sect. 4.3. Since the total search time is linear in the number of points in the first frame, it is natural to exploit a coarse-to-fine strategy. During the first few iterations, we can use coarser samples (e.g., every five) instead of all sample points on the curve. When the algorithm almost converges, we use all available points in order to obtain a precise estimation.

5 Experimental Results

The proposed algorithm has been implemented in C. In order to maintain the modularity, the code is not optimized. The motion estimation algorithm described in Sect. 4.4 is not used; that is, we do not take into account measurement uncertainty in the experiments described below. In all these experiments, the same parameters are used: $e = 10$ mm (see Sect. 4.2) and $\mathfrak{D}$ is computed as described in Sect. 4.3. It is thus never larger than $2e$. The program is run on a SUN 4/60 workstation, and any quoted times are given for execution on that machine.

This section is divided into three subsections. In the first the algorithm is applied to synthetic data. The results show clearly the typical behaviour of the algorithm to be expected in practice. The second describes the robustness and efficiency of the algorithm using synthetic data with different levels of noise and different samplings. The third describes the experimental results with real data.

5.1 A Case Study

In this experiment, the parametric curve described by $\mathbf{x}(u) = [u^2,\ 5u\sin(u) + 10u\cos(1.5u),\ 0]^T$ is used. The curve is sampled twice in different ways. Each sample set contains 200 points. The second set is then rotated and translated with $\mathbf{r} = [0.02,\ 0.25,\ -0.15]^T$ and $\mathbf{t} = [40.0,\ 120.0,\ -50.0]^T$. We thus get two noise-free frames. (The same noise-free data are used in the experiments described in the next section.)

For each point, zero-mean Gaussian noise with a standard deviation equal to 2 is added to its x, y and z components. We show in Fig. 6 the front and top views of the noisy data. For visual convenience, points are linked. The solid curve is the one in the first frame, and the dashed one, in the second frame. The data are used as is; no smoothing is performed.

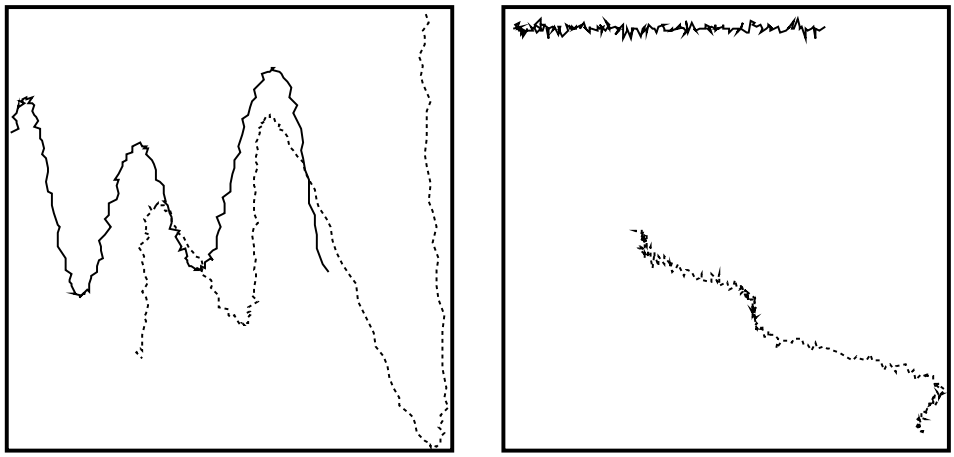


Fig. 6. Front and top views of the data

The first step is then to find matches for the points in the first frame. As $D^0_{\max}$ is big, each point has a match. We find 200 matches in total, which are shown in Fig. 7, where matched points are linked. Many false matches are observed. We then update these matches using the technique described in Sect. 3.3, and 100 matches survive, which are shown in Fig. 8.

Even after the updating, there are still some false matches. Because there are more good matches then false matches, the motion estimation algorithm still yields a reasonable estimate. This can be observed in Fig. 9, where the motion estimated has been applied to the points in the first frame. We can observe the improvement of the registration of the two curves, especially in the top view.

Now we enter the second iteration. We find at this time 176 matches, which are shown in Fig. 10a. (Top view is not shown, because the two curves are very close.) Several false matches are observable. After updating, 146 matches remain, as shown in Fig. 10b. Almost all these matches are correct. Motion is then computed from these matches.

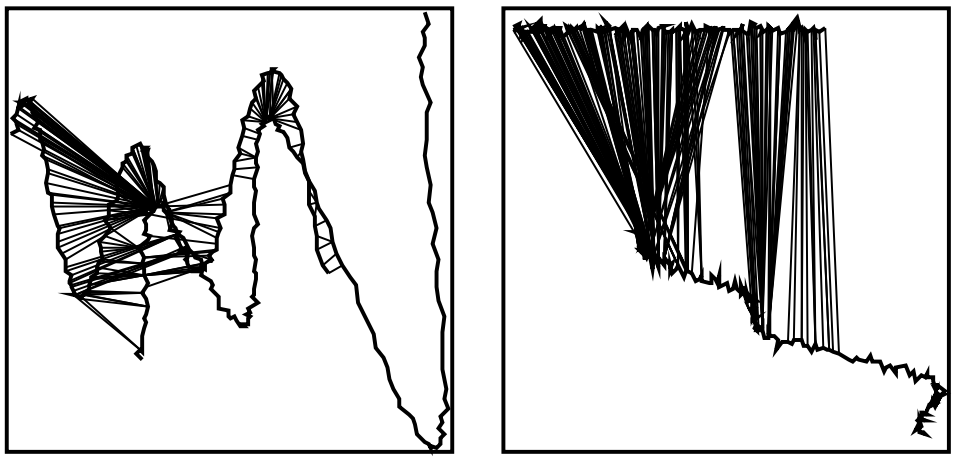


Fig. 7. Matched points in the first iteration before updating (front and top views)

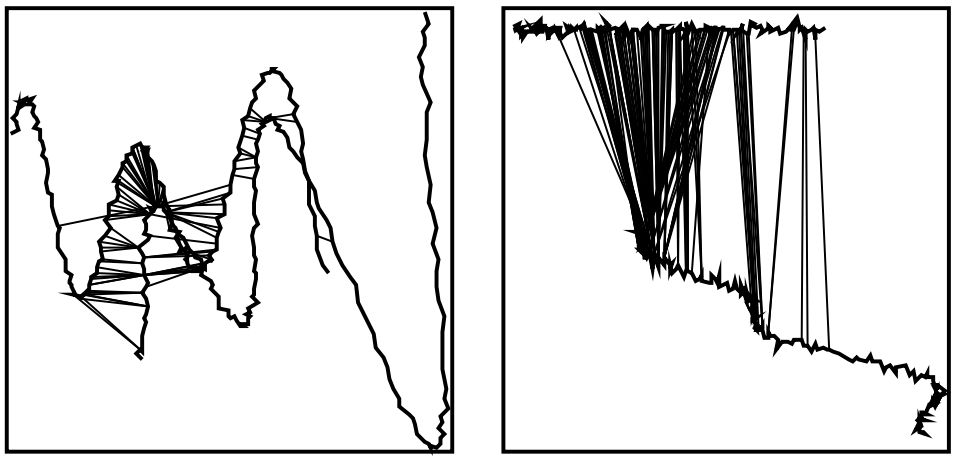


Fig. 8. Matched points in the first iteration after updating (front and top views)

We iterate the process in the same manner. The motion result after 10 iterations is shown in Fig. 11. The registration between the two curves is already quite good.

The algorithm yields after 15 iterations the following motion estimate:

$$\hat{\mathbf{r}} = [2.442 \times 10^{-2},\, 2.503 \times 10^{-1},\, -1.484 \times 10^{-1}]^T \,,$$
$$\hat{\mathbf{t}} = [3.879 \times 10^{1},\, 1.139 \times 10^{2},\, -4.967 \times 10^{1}]^T \,.$$

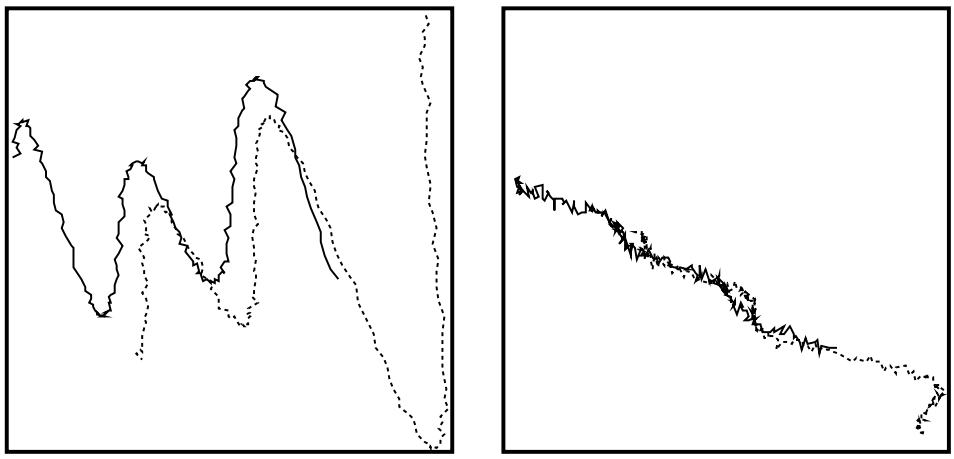
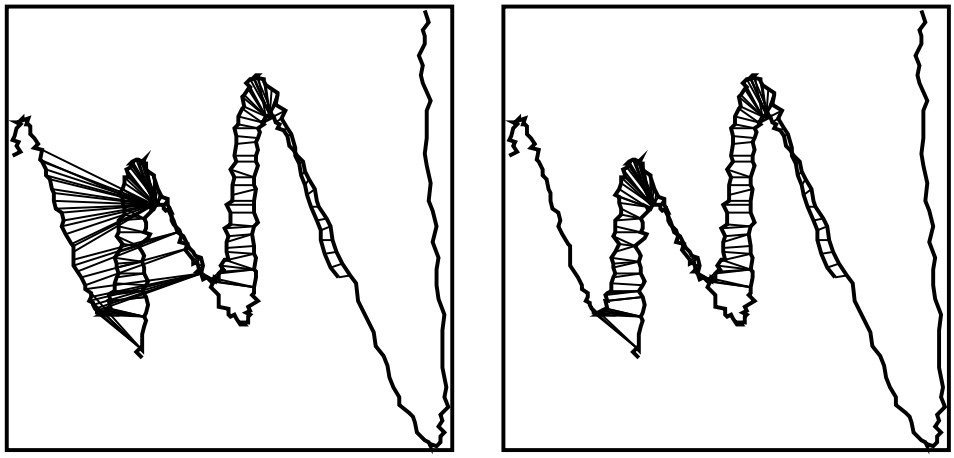


Fig. 9. Front and top views of the motion result after the first iteration



(a) (b)

Fig. 10. Matched points before and after updating in the second iteration (only the front view)

To measure the precision in motion estimation, we define the rotation error as

$$e_r = \|\mathbf{r} - \hat{\mathbf{r}}\|/\|\mathbf{r}\| \times 100\% \ , \tag{29}$$

where $\mathbf{r}$ and $\hat{\mathbf{r}}$ are respectively the real and estimated rotation parameters,

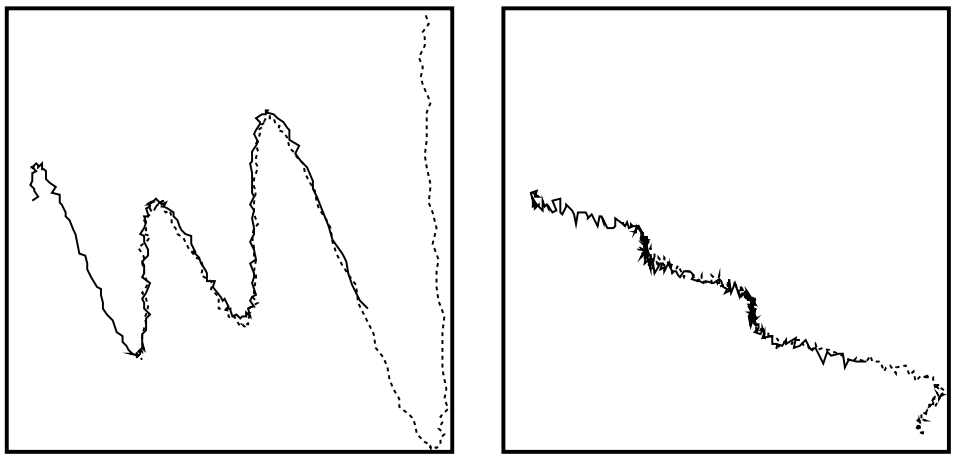


Fig. 11. Front and top views of the motion result after ten iterations

and the translation error as

$$e_t = \|\mathbf{t} - \hat{\mathbf{t}}\|/\|\mathbf{t}\| \times 100\% \ , \tag{30}$$

where $\mathbf{t}$ is the real translation parameter and $\hat{\mathbf{t}}$ is the estimated one. In Fig. 12, we show the evolution of the rotation and translation errors versus the number of iterations. Fast convergence is observed during the first few iterations and relatively slower later. After 15 iterations, the rotation error is 1.6% and the translation error is 4.6%.

We show in Table 1 several intermediate results during different iterations. The results are divided into three parts. The second to fourth rows indicate the execution time (in seconds) required for finding matches, updating the matching, and computing the motion, respectively. The fifth row shows the values of $D_{\max}$ used in different iterations. The last row shows the comparison of the numbers of matches found in different iterations before and after updating. We have the following remarks:

- $D_{\max}$ almost decreases monotonically with the number of iterations. This is because the registration becomes better and better, and $D_{\max}$ is computed dynamically through the statistic analysis of distances.
- The time required for finding matches almost decreases monotonically, too. This is because of the almost monotonic decrease of $D_{\max}$. Less search in k-D tree is required when the search region becomes smaller.

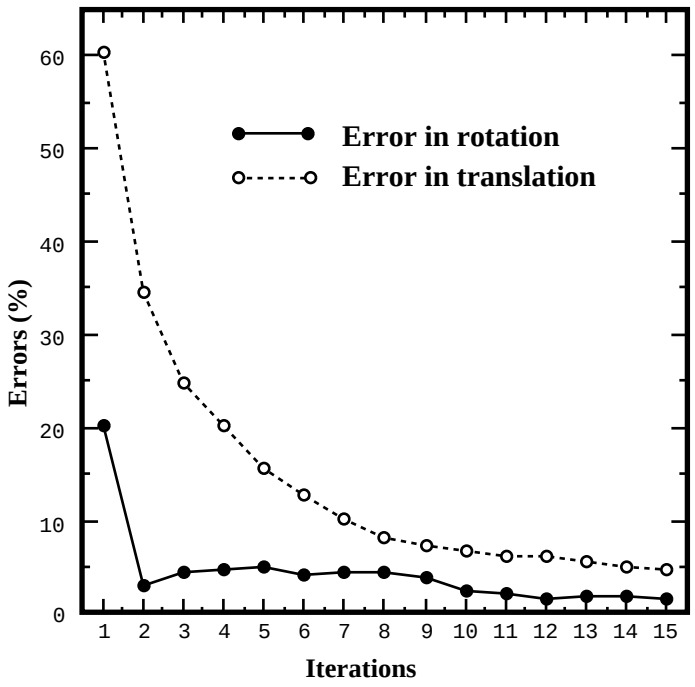


Fig. 12. Evolution of the rotation and translation errors versus the number of iterations

- The time required for updating the matching is negligible.
- The time required for computing the motion is almost constant, as it is related to the number of matches (here almost constant). Furthermore, the motion algorithm is very efficient: about 0.05 seconds for 145 matches.
- The numbers of matches before and after updating do not vary much after the first few iterations. This also implies that the Gaussian assumption of the distance distribution is reasonable.

The total execution time is 6.5 seconds in this experiment.

5.2 Synthetic Data

In this section, we describe the robustness and efficiency of the algorithm using the same synthetic data as in the last section, but with different levels of noise and different samplings. All results given below are the average of ten tries.

Table 1. Several detail results in different iterations

iteration	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
matching time	2.20	1.30	0.62	0.33	0.25	0.28	0.22	0.17	0.15	0.17	0.15	0.12	0.13	0.13	0.12
update time	0.03	0.02	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.02	0.00	0.02
motion time	0.05	0.05	0.05	0.05	0.05	0.03	0.05	0.03	0.02	0.07	0.05	0.03	0.03	0.02	0.02
$D_{\max}$	235	140	78.5	46.1	32.8	34.7	28.0	22.4	18.9	16.7	15.3	13.6	12.3	10.7	9.89
nb. $\frac{\text{before}}{\text{after}}$	$\frac{200}{100}$	$\frac{176}{146}$	$\frac{150}{143}$	$\frac{148}{137}$	$\frac{147}{147}$	$\frac{148}{148}$	$\frac{148}{147}$	$\frac{148}{147}$	$\frac{148}{146}$	$\frac{148}{146}$	$\frac{147}{147}$	$\frac{146}{145}$	$\frac{143}{143}$	$\frac{143}{143}$	$\frac{143}{143}$

The first series of experiments are carried out with respect to different levels of noise. The standard deviation of the noise added to each point varies from 0 to 20. Similar to Fig. 12, we show, as a sample, in Fig. 13 and Fig. 14 the evolutions of the rotation and translation errors versus the number of iterations with a standard deviation equal to 2 and 8. From these results, we observe that

- The translation error decreases almost monotonically, while the shape for the rotation error is more complex.
- Noise has a stronger impact on the rotation parameters than on the translation parameters. When noise is small, there is in general a smaller error in rotation than in translation. When noise is important, the inverse is observed.

We think the above phenomena are due to the fact that the relation between the measurements and the rotation parameters is nonlinear while that between the measurements and the translation parameters is linear.

To visually demonstrate the effect of the noise added and the ability of the algorithm, we show in Fig. 15 and Fig. 16 two sample results. In each figure, the upper row displays the front and top views of the two noisy curves before registration; the lower row displays the front and top views of the two noisy curves after registration. In Fig. 15 and Fig. 16, we have added, to each x, y, and z components of each point of the two curves, zero-mean Gaussian noise with a standard deviation equal to 8 and 16, respectively. Even though the curves are so noisy, the registration between them is surprisingly good.

We now summarize more results in Table 2. The rotation and translation errors are measured in percents, and the execution time, in seconds. Each

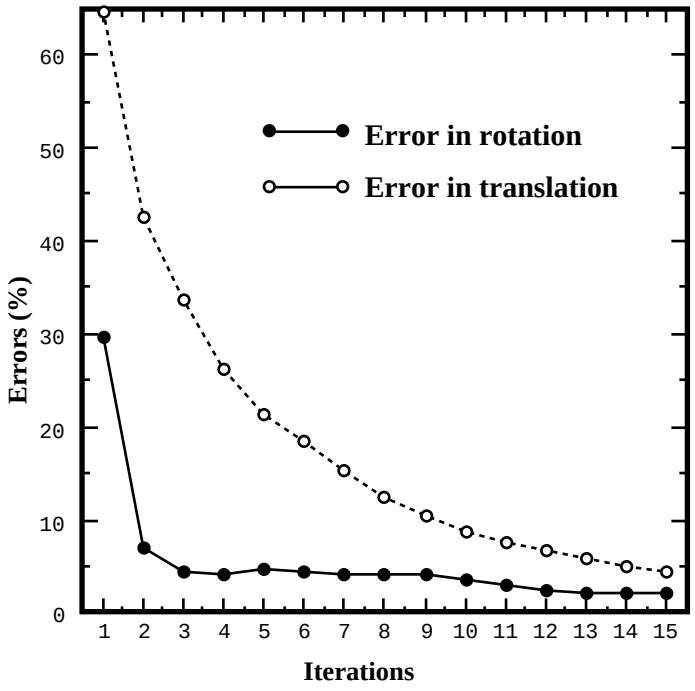


Fig. 13. Evolution of the rotation and translation errors versus the number of iterations with a standard deviation equal to 2

number shown is the average of 10 tries. 15 iterations have been applied. We have the following conclusions:

- The errors in rotation and in translation increase with the increase in the noise added to the data, as expected.
- Noise in the measurements has more effect in the rotation than in translation.
- The algorithm is robust. It yields a reasonable motion estimation even when the data are significantly corrupted.
- The execution time increases also with the increase in the noise added to the data. This is because when the data are very noisy the value of $D_{\max}$ stays big, and the search have to be performed in a large space.

We now investigate the ability of the algorithm with respect to different samplings of curves. The same data are used. Zero-mean Gaussian noise with a standard deviation equal to 2 is added to each x, y, and z components of

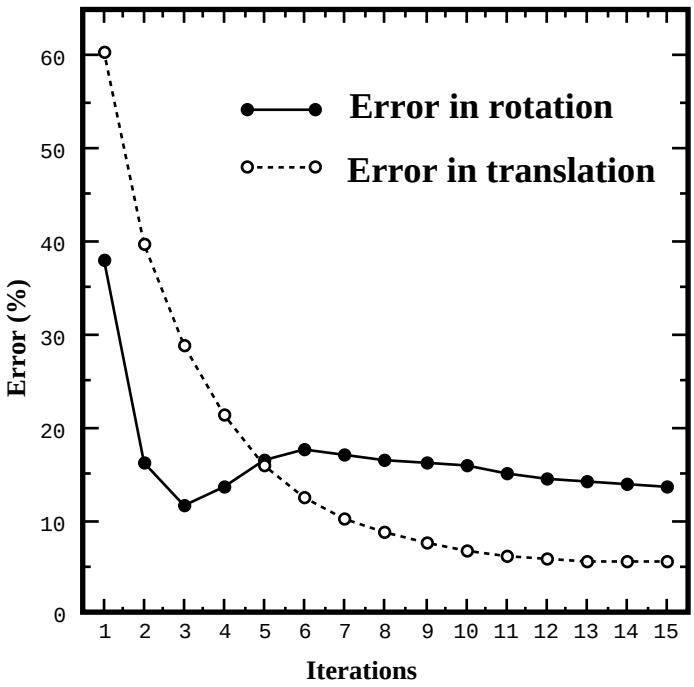


Fig. 14. Evolution of the rotation and translation errors versus the number of iterations with a standard deviation equal to 8

Table 2. A summary of the experimental results with synthetic data

standard deviation	0	2	4	6	8	10	12	14	16	18	20
rotation error	2.25	2.12	4.63	9.62	13.73	14.31	20.47	18.07	23.87	37.04	33.20
translation error	1.77	4.36	4.55	4.84	5.70	7.81	8.93	9.89	17.15	22.00	27.17
execution time	6.27	6.82	8.58	9.26	11.12	11.86	12.59	13.35	16.40	16.56	17.32

each point of the two curves. We have already described in Sect. 4.2 the effect of different samplings of the curves in the second frames. Here we vary the sampling of the curve in the first frame from 1 (i.e., all points) to 10 (i.e., one out of every ten points). Ten tries are carried out for each sampling. The errors in rotation and in translation (in percents), and the execution time (in seconds) versus different samplings are shown in Table 3. Two remarks can be made:

- Generally speaking, the more samples there are in a curve, the less the

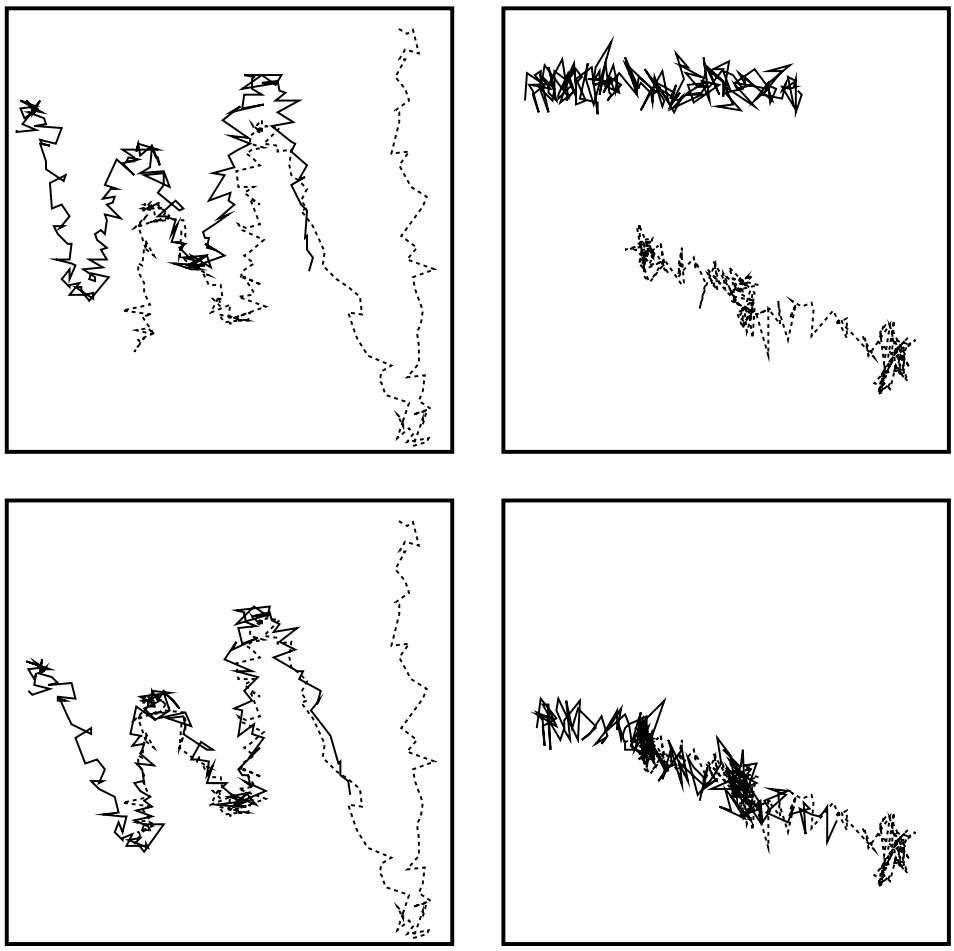


Fig. 15. Front and top views of two noisy curves with a standard deviation equal to 8 before and after registration

error in the estimation of the rotation and translation will be. However, the exact relation is not very clear. Consider sampling = 1 and sampling = 10. The latter has only 20 points while the former has 200 points. The motion error, however, is only twice as large.

- The execution time decreases monotonically as the number of sample points decreases. The relation, however, is not linear. Fast decrease is observed when the number of sample points is high.

In the foregoing discussions we have observed that using coarsely sam-

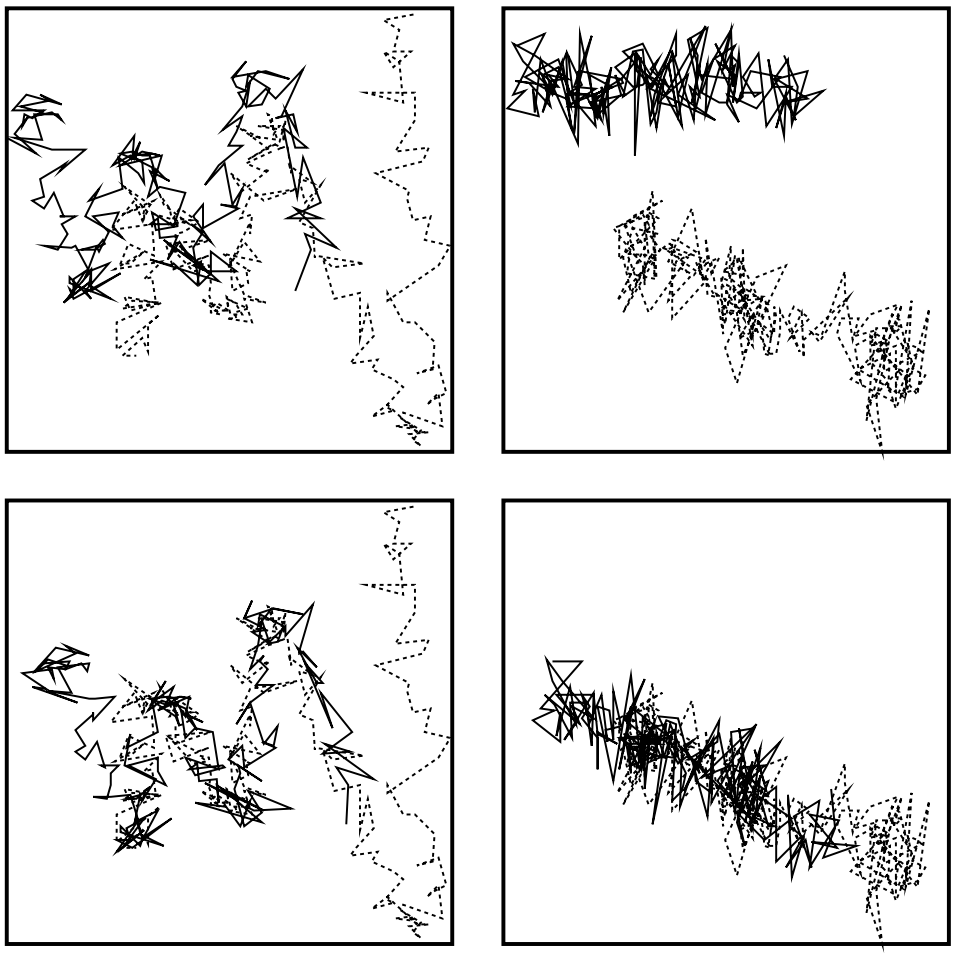


Fig. 16. Front and top views of two noisy curves with a standard deviation equal to 16 before and after registration

pled points in the curves in the first frame does not affect too much the accuracy of the final motion estimation, but it considerably speeds up the whole process. It is natural to think about using a coarse-to-fine strategy such as that described in Sect. 4.5. The finding of fast convergence of the algorithm during the first few iterations (see Fig. 13 and Fig. 14) and the finding of relatively expensive search (see Table 1) justify the following strategy. During the first few iterations, we use coarser, instead of all, sample points, which allows for finding an estimation close to the optimal. We then use all

Table 3. Results with respect to different samplings

fraction of points	1	1/2	1/3	1/4	1/5	1/6	1/7	1/8	1/9	1/10
rotation error	2.12	3.44	4.19	4.88	4.09	7.52	4.75	6.09	5.98	4.90
translation error	4.36	5.14	4.27	4.75	4.11	6.67	8.54	7.45	8.52	7.34
execution time	6.82	3.53	2.41	1.85	1.52	1.28	1.11	1.01	0.89	0.83

sample points to refine this estimate. We have conducted ten experiments using the same data as before by adding zero-mean Gaussian noise with a standard deviation equal to 3. During the first five iterations, only 40 points (one out of every five points) are used. These are followed by ten iterations using all points. The average results of the ten experiments are: rotation error = 4.56%, translation error = 4.29%, and execution time = 3.39 s. For comparison, we performed 15 iterations using all points. The average results of the ten tries are: rotation error = 4.68%, translation error = 4.14%, and execution time = 7.49 s. Only little difference between the final motion estimations is observed, but the algorithm is more than twice faster by exploiting the coarse-to-fine strategy.

5.3 Real Data

In this section, we provide an example with real data. A trinocular stereo system mounted on our mobile vehicle is used to take images of a chair scene (the scene is static but the robot moves). We show in Fig. 17 two images taken by the first camera from two different positions. The displacement between the two positions is about 4 degrees in rotation and 100 millimeters in translation. The chair is about 3 meters from the mobile vehicle.

The curve-based trinocular stereo algorithm developed in our laboratory [11] is used to reconstruct the 3-D frames corresponding to the two positions. There are 36 curves and 588 points in the first frame, and 48 curves and 763 points in the second frame. We show in the upper row of Fig. 18 the front view and the top view of the superposition of the two 3-D frames. The curves in the first frame is displayed in solid lines while those in the second frames, in dashed lines. We apply the algorithm to the two frames. The algorithm converges after 12 iterations. It takes in total 32.5 seconds on a SUN 4/60 workstation and half of the time is spent in the first

Fig. 17. Images of a chair scene taken by the first camera from two different positions

iteration (so we could speed up the process by setting $D^0_{\max}$ to a smaller value). The final motion estimate is

$$\hat{\mathbf{r}} = [-1.527 \times 10^{-3},\ 6.639 \times 10^{-2},\ 2.894 \times 10^{-3}]^T\ ,$$
$$\hat{\mathbf{t}} = [-4.266 \times 10^{0},\ -1.586 \times 10^{0},\ -1.009 \times 10^{2}]^T\ .$$

The motion change is: $\delta\mathbf{r} = 0.78\%$ and $\mathbf{t} = 0.53\%$. The result is shown in the lower row of Fig. 18 where we have applied the estimated motion to the first frame. Excellent registration is observed for the chair. The registration of the border of the wall is a little bit worse because more error is introduced during the 3-D reconstruction for it is far away from the cameras.

Now we exploit the coarse-to-fine strategy. As before, we do coarse matching in the first five iterations by sampling evenly one out of every five points on the curves in the first frame, followed by fine matching using all points. The algorithm converges after 12 iterations and yields exactly the same motion estimation as when only doing fine matching. The execution time, however, decreases from 32.5 seconds to 10.5 seconds, about three times faster. If now we sample evenly one out of every *ten* points on the curves in the first frame, and do coarse matching in the first five iterations and fine matching in the subsequent ones, the algorithm converges after 13 iterations (one

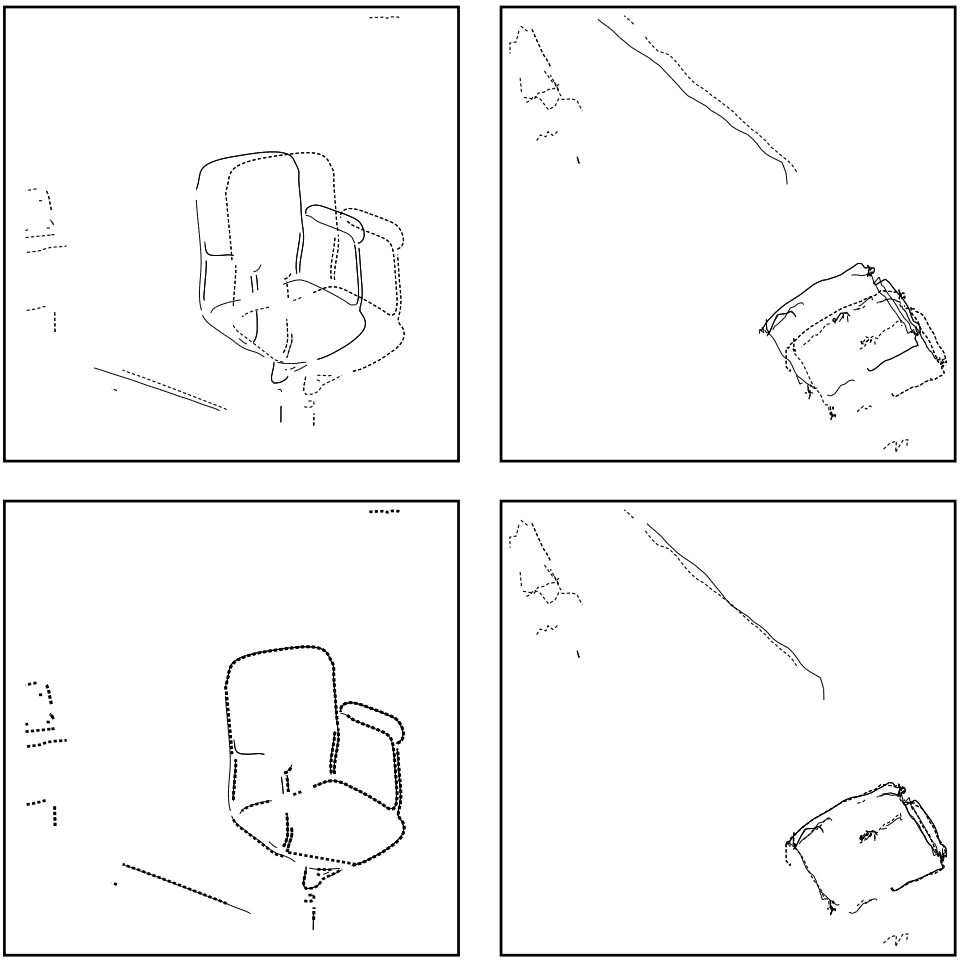


Fig. 18. Superposition of two 3-D frames before and after registration: front and top views

iteration more), and the final motion estimate is

$$\hat{\mathbf{r}} = [-1.438 \times 10^{-3},\ 6.653 \times 10^{-2},\ 2.995 \times 10^{-3}]^T\ ,$$
$$\hat{\mathbf{t}} = [-4.282 \times 10^{0},\ -1.637 \times 10^{0},\ -1.007 \times 10^{2}]^T\ ,$$

which is almost the same as the one estimated using directly all points. The motion change is: $\delta\mathbf{r} = 0.71\%$ and $\mathbf{t} = 0.50\%$. The execution time is now 8.8 seconds.

6 Discussions

6.1 Complexity

As described earlier, each iteration of our algorithm consists of three main steps. The first is to find closest points, at an expected cost of $O(N_i^m \log N_k^n)$, where N_i^m and N_k^n are the number of points in the first and second frames, respectively. The second is to update the matching recovered in the first step, at a cost of $O(N_i^m)$. The last step is to compute the 3-D motion, also at a cost of $O(N_i^m)$. Thus the total complexity of our algorithm is $O(N_i^m \log N_k^n)$. For simplicity, we assume here $N_i^m = N_k^n = N$.

We now compare the complexity of our algorithm with that of the string-based matching methods (e.g., [8]). Typically, a string-based matching method for registration of two curves each containing n points has a cost $O(n \log n)$. Let m be the number of curves in each frame, and assume each curve contains approximately the same number of points (i.e., $n \approx N/m$). Because there are m possible pairings of curves in the two frames, the total cost of a typical string-based matching method is

$$O(m^2 \frac{N}{m} \log \frac{N}{m}) = O(mN \log \frac{N}{m}) \ .$$

A simple computation shows that if $N \geq m^{m/(m-1)}$, then $mN \log \frac{N}{m} \geq N \log N$. In practice, a curve contains at least two points, i.e., $N \geq 2m$. Since $2m \geq m^{m/(m-1)}$ for $m \geq 2$, our algorithm has a lower bound of computational cost. If there is only one curve in each frame (i.e, $m = 1$ and $n = N$), our algorithm has the same complexity as a typical string-based matching method.

6.2 How About Large Motion ?

Because of the local property of the matching criterion used, our algorithm converges to the closest minimum. It is thus best applied in situations where the motion is small or approximately known, and a precise estimation of the motion is required. In the case of large motion, the algorithm can be adapted in two different ways. The first way is to apply first the global methods as cited in the introductory section to obtain an estimation, which is then refined by applying the algorithm described in this paper. The second way is

to obtain a set of initial registrations by sampling the 6-D motion space, and then apply our algorithm to each initial registration. The final estimation corresponding to the global minimum error is retained as the optimal one. This method has been used in [10] to solve the object recognition problem.

6.3 Multiple Object Motions

In a dynamic environment, there is usually more than one moving object. It is important to have reliable algorithm for segmenting the scene into objects using motion information. However, little work has been done so far in this direction.

We have proposed in [29] a framework to deal with multiple object motions. It consists of two levels. The first level deals with the tracking of 3-D tokens from frame to frame and the estimation of their motions. The processing is completely parallel for each token. The second level groups tokens into objects based on the similarity of motion parameters. Tokens coming from a single object should have the same motion parameters. In [29] the tokens used are 3-D line segments, and the experiments have shown that the framework is flexible and powerful. Now if we replace 3-D line segments by 3-D curves and estimate 3-D motion for each curve, the general framework is still applicable.

6.4 Highlights With Respect to Previous Work

As noted in the introduction, independent work on curve matching was conducted by Besl and McKay [10]. They use the same idea: iteratively matching points in one set to the closest points in another set[‡]. The main difference lies in the matching criterion. Refer to Eq. (2). In our algorithm, $p_{i,j}$ can take value either 1 or 0 depending on whether the point in the first set has a reasonable match in the second set or not. This is determined by the maximum tolerable distance $D_{\max}$, which, in turn, is set in a dynamic way by analyzing the statistics of the distances as described in Sect. 3.3. Therefor, our algorithm is capable of dealing with the following situations:

[‡]Besl and McKay show two sets of data differing by a large motion. They then sample the 6-D motion space to obtain a set of initial registrations, as described in Sect. 6.2. However, they do not show the particular initial registration which leads to the final result.

- Gross outliers in the data. The outliers are automatically discarded in the matching and thus have no effect on the final motion estimation.
- Appearance and disappearance in which curves in one set do not appear in the other set. This is usually the case in navigation where objects may enter or leave the field of view.
- Occlusion. An object may occlude other objects, and it may itself be occluded. This is common in both object recognition and navigation.

In the algorithm of Besl and McKay, $p_{i,j}$ takes always value 1. Thus, their algorithm can only deal with the case in which the first set is a *subset* of the second set. It is powerless in the situations described above.

Other differences between the two algorithms include:

- k-D trees are used in our algorithm to speed up the computation for finding the closest points.
- The dual quaternion method is used in our algorithm to compute the 3-D motion, which has a possibility to partially take into account the uncertainty of data points. The singular value decomposition method is used in their algorithm.

7 Conclusions

We have described an algorithm for the registration of free-form curves, i.e., arbitrary space curves of the type found in practice. We have used the assumption that the motion between two frames is small or approximately known, a realistic assumption in many practical applications including visual navigation. A number of experiments have been carried out and good results have been obtained.

Our algorithm has the following features:

- It is simple. The reader can easily reproduce the algorithm.
- It is extensible. More sophisticated strategies such as figural continuity can be easily integrated in the algorithm.
- It is general. First, the representation used is general for representing arbitrary space curves of the type found in practice. Second, the ideas behind the algorithm are applicable to (many) other matching problems. The algorithm can easily be adapted to solve for example 2-D curve matching and 3-D surface matching.

- It is efficient. The most expensive computation is the process of finding closest points, which has a complexity $O(N \log N)$. Exploiting the coarse-to-fine strategy described in Sect. 4.5 considerably speeds up the algorithm with only a small change in the precision in the final estimation.
- It is robust to gross errors and can deal with appearance, disappearance and occlusion of objects, as described in Sect. 6.4. This is achieved by analyzing dynamically the statistics of the distances, as described in Sect. 3.3.
- It yields an accurate estimation because all available information is used in the algorithm.
- It does not require any preprocessing of 3-D point data such as for example smoothing. The data are used as is in our algorithm. That is, there is no approximation error[§].
- It does not require any derivative estimation (which is sensitive to noise), in contrast with many other feature-based or string-based matching methods.

Our algorithm can only partially take the uncertainty of measurements into account. To fully take into account the uncertainty, we should replace the dual quaternion algorithm by other methods such as Kalman filtering techniques. This causes a significant increase in the computational cost of the algorithm.

Our algorithm converges to the closest local minimum, and thus is not appropriate for solving large motion problems. Two possible extensions of the algorithm to deal with large motions have been described in Sect. 6.2: coupling with a global method or sampling the motion space.

In our algorithm, one parameter, the parameter $\mathfrak{D}$, needs to be set by the user. It indicates when the registration can be considered to be good. It has an impact on the convergence rate, as described in Sect. 4.3. In our implementation, $\mathfrak{D}$ is automatically computed using the intervals of chained points. This method works well for all experiments we have carried out. However, a better method probably exists. Intuitively, the parameter $\mathfrak{D}$ is related not only to the intervals of chained points but also to the shape of the curves. $\mathfrak{D}$ should be smaller for rough curves than for smooth ones. We are currently investigating this issue.

We are currently extending the algorithm to solve surface matching problems arising in navigation. When a mobile vehicle navigates in a natural

[§] It is certain that errors have been introduced during the reconstruction of 3-D points, and that they have been propagated in the motion estimation

environment, a correlation-based stereo algorithm or a range finder provides a sequence of dense 3-D maps. Only minor modifications are needed in order to produce an algorithm for registering successive 3-D maps.

Acknowledgment

The author would like to thank Olivier Faugeras for stimulating discussions during the work, and Steve Maybank for carefully reading the draft version.

References

[1] P. Besl and R. Jain, "Three-dimensional object recognition," *ACM Computing Surveys*, vol. 17, pp. 75–145, March 1985.

[2] R. Chin and C. Dyer, "Model-based recognition in robot vision," *ACM Computing Surveys*, vol. 18, pp. 67–108, March 1986.

[3] P. J. Besl, "Geometric modeling and computer vision," *Proc. IEEE*, vol. 76, pp. 936–958, August 1988.

[4] R. Bolles and R. Cain, "Recognizing and locating partially visible objects, the local-feature-focus method," *Int'l J. Robotics Res.*, vol. 1, no. 3, pp. 57–82, 1982.

[5] D. Walters, "Selection of image primitives for general-purpose visual processing," *Comput. Vision, Graphics Image Process.*, vol. 37, no. 3, pp. 261–298, 1987.

[6] E. E. Milios, "Shape matching using curvature processes," *Comput. Vision, Graphics Image Process.*, vol. 47, pp. 203–226, 1989.

[7] T. Pavlidis, "Algorithms for shape analysis of contours and waveforms," *IEEE Trans. PAMI*, vol. 2, no. 4, pp. 301–312, 1980.

[8] J. T. Schwartz and M. Sharir, "Identification of partially obscured objects in two and three dimensions by matching noisy characteristic curves," *Int'l J. Robotics Res.*, vol. 6, no. 2, pp. 29–44, 1987.

[9] H. Wolfson, "On curve matching," *IEEE Trans. PAMI*, vol. 12, no. 5, pp. 483–489, 1990.

[10] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," *IEEE Trans. PAMI*, vol. 14, pp. 239–256, February 1992.

[11] L. Robert and O. Faugeras, "Curve-based stereo: Figural continuity and curvature," in *Proc. IEEE Conf. Comput. Vision Pattern Recog.*, (Maui, Hawaii), pp. 57–62, June 1991.

[12] R. E. Sampson, "3D range sensor-phase shift detection," *Computer*, no. 20, pp. 23–24, 1987.

[13] R. Safaee-Rad, I. Tchoukanov, B. Benhabib, and K. C. Smith, "Accurate parameter estimation of quadratic curves from grey-level images," *CVGIP: Image Understanding*, vol. 54, pp. 259–274, September 1991.

[14] G. Taubin, "Estimation of planar curves, surfaces, and nonplanar space curves defined by implicit equations with applications to edge and range image segmentation," *IEEE Trans. PAMI*, vol. 13, pp. 1115–1138, November 1991.

[15] J. E. W. Mayhew and J. P. Frisby, "Psychophysical and computational studies towards a theory of human stereopsis," *Artif. Intell.*, vol. 17, pp. 349–385, 1981.

[16] S. Pollard, J. Mayhew, and J. Frisby, "PMF: A stereo correspondence algorithm using a disparity gradient limit," *Perception*, vol. 14, pp. 449–470, 1985.

[17] Z. Zhang, O. Faugeras, and N. Ayache, "Analysis of a sequence of stereo scenes containing multiple moving objects using rigidity constraints," in *Proc. Second Int'l Conf. Comput. Vision*, (Tampa, FL), pp. 177–186, IEEE, December 1988.

[18] M. W. Walker, L. Shao, and R. A. Volz, "Estimating 3-D location parameters using dual number quaternions," *CVGIP: Image Understanding*, vol. 54, pp. 358–367, November 1991.

[19] O. Faugeras and M. Hebert, "The representation, recognition, and locating of 3D shapes from range data," *Int'l J. Robotics Res.*, vol. 5, no. 3, pp. 27–52, 1986.

[20] K. Arun, T. Huang, and S. Blostein, "Least-squares fitting of two 3-D point sets," *IEEE Trans. PAMI*, vol. 9, pp. 698–700, September 1987.

[21] F. Preparata and M. Shamos, *Computational Geometry, An Introduction*. New-York: Springer, Berlin, Heidelberg, 1986.

[22] S. Blostein and T. Huang, "Error analysis in stereo determination of a 3-D point position," *IEEE Trans. PAMI*, vol. 9, pp. 752–765, November 1987.

[23] L. Matthies and S. A. Shafer, "Error modeling in stereo navigation," *IEEE J. RA*, vol. 3, pp. 239–248, June 1987.

[24] D. Kriegman, E. Triendl, and T. Binford, "Stereo vision and navigation in buildings for mobile robots," *IEEE Trans. RA*, vol. 5, pp. 792–803, December 1989.

[25] N. Ayache and O. D. Faugeras, "Maintaining Representations of the Environment of a Mobile Robot," *IEEE Trans. RA*, vol. 5, pp. 804–819, December 1989.

[26] R. Szeliski, "Bayesian modeling of uncertainty in low-level vision," *Int'l J. Comput. Vision*, vol. 5, no. 3, pp. 271–301, 1990.

[27] Z. Zhang and O. Faugeras, "Determining motion from 3D line segments: A comparative study," *Image and Vision Computing*, vol. 9, pp. 10–19, February 1991.

[28] Z. Zhang and O. Faugeras, *3D Dynamic Scene Analysis: A Stereo Based Approach*. Springer, Berlin, Heidelberg, 1992.

[29] Z. Zhang and O. Faugeras, "Three-dimensional motion computation and object segmentation in a long sequence of stereo frames," *Int'l J. Comput. Vision*, March 1992.