Visual Comput (2008) 24: 679–688
DOI 10.1007/s00371-008-0248-6

ORIGINAL ARTICLE

A. Cengiz Oztireli
Cagatay Basdogan

A new feature-based method for robust and efficient rigid-body registration of overlapping point clouds

Published online: 29 May 2008

C. Basdogan (✉)
College of Engineering, Koc University,
34450 Istanbul, Turkey
cbasdogan@ku.edu.tr

A.C. Oztireli
Department of Computer Science, ETH,
8092 Zurich, Switzerland
cengizo@student.ethz.ch

Abstract We propose a new feature-based registration method for rigid-body alignment of overlapping point clouds (PCs) efficiently under the influence of noise and outliers. The proposed registration method is independent of the initial position and orientation of PCs, and no assumption is necessary about their underlying geometry. In the process, we define a simple and efficient geometric descriptor, a novel k-NN search algorithm that outperforms most of the existing nearest neighbor search algorithms used for the same task, and a new algorithm to find corresponding points between PCs based on the invariance of Euclidian distance under rigid-body transformation.

Keywords 3D registration · Feature extraction · Distance invariants · Geometric descriptors · Nearest neighbor search

1 Introduction

Point-based representation of 3D objects is becoming a new standard in computer graphics. An open problem in this area is the rigid body registration or alignment of two unstructured point clouds (PCs) containing noise and outliers.

There are two different approaches to solve this problem: one is based on the minimization of distance between M and P as in the case of the widely used iterative closest point (ICP) algorithm [3], and the other is based on the selection of distinct features common to both M and P [5].

For the latter kind of algorithms, a common approach is to assign a geometric shape descriptor to each point in both PCs and then match the compatible points using the descriptor values. Hence, the feature-based approaches are based on the idea that only a small number of correspondences is sufficient to compute the optimal transformation.

The major advantages of using the feature-based methods over the ICP-based methods are:

(1) It is not necessary to search for all points in PCs to find the corresponding pairs. Thus, redundant or irrelevant points such as outliers, or points that do not have correspondences have no direct effect on the registration.
(2) The registration process is independent of the initial alignment of PCs.

The major disadvantages are:

(1) PCs must have distinct features.
(2) Even so, finding distinct features common to both PCs can be quite challenging if they contain noise and outliers.
(3) In general, the feature-based methods are slower than the ICP-based methods.

We propose a new feature-based registration method that provides efficient and robust solutions to the problems discussed above. Our method is not restricted to 3D and can be used effectively for higher dimensional problems as well. No assumptions are made about the underlying geometric structure of PCs (i.e. PCs do not necessarily represent surfaces).

The major contributions of this paper can be grouped into three:

1. We propose a new four-stage feature selection method. Our method is able to extract compatible features of PCs being registered even if a geometric descriptor having a low discriminatory power is used. We show that our simple geometric descriptor works well with the proposed method.
2. We propose a new algorithm to determine the local neighbors of all points in a PC efficiently. In our study, we use this algorithm to find the neighbors of a point in order to calculate its geometric descriptor, but it is a general purpose algorithm that can easily be used in other applications as well.
3. We propose a novel algorithm for finding the corresponding points in PCs being registered. It is efficient and robust to outliers. In fact, we show that this algorithm can be used independently of the proposed four-stage feature selection method if the noise level in PCs is not significant.

2 Geometric descriptor

A good geometric descriptor must be:

(1) invariant to rigid body transformations,
(2) insensitive to noise,
(3) discriminative, and
(4) efficient to compute.

Descriptors based on local differential properties of a surface such as curvature are sensitive to noise [7, 10, 17] and thus are not widely used for the registration problem. Descriptors that partition the region around a point into bins such as 3D harmonic shape contexts [7], or Spin images [11] are commonly used in object recognition. Although these multi-valued descriptors provide a richer description of the local region around a point, selecting feature points of a PC by computing and comparing the multi-valued descriptors is a computationally intensive task. For these reasons, low dimensional descriptors such as "surface variation" [16], integral volume descriptor [10] are often preferred over the high dimensional ones for the solution of a registration problem.

We propose a new low-dimensional descriptor that is much easier to compute than the existing ones, but it has a poor discriminatory power. Our goal is to show that our four-stage feature selection method can compensate for its inferior discriminatory power.

2.1 Our descriptor

The proposed descriptor d is the distance between point $\boldsymbol{q}$ and the center of mass of the neighboring points around $\boldsymbol{q}$.

$$d = \|\boldsymbol{q} - \boldsymbol{c}\|, \quad \boldsymbol{c} = \frac{\sum \boldsymbol{p}_i}{N}, \quad \boldsymbol{p}_i \in \mathfrak{N}(\boldsymbol{q}) \tag{1}$$

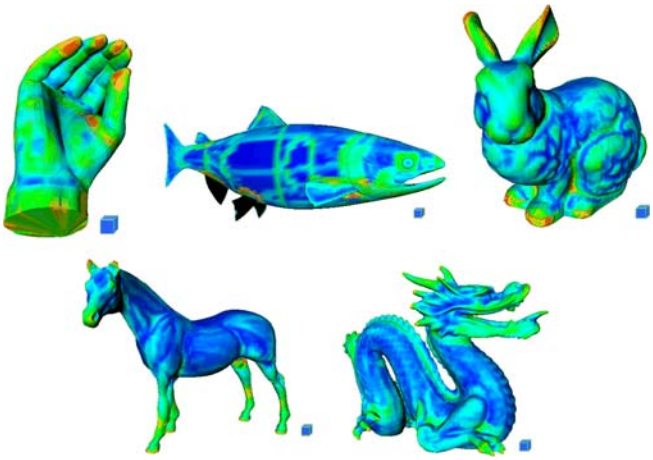

Fig. 1. The distribution of descriptor values for each model (the *green color* indicates low descriptor values while the *red color* indicates high values). The *small box* shown next to each model represents the neighborhood size used in the calculation of the descriptor value of each point. Note that the sizes of the models are not to scale

where, N is the number of neighboring points in the neighborhood $\mathfrak{N}(\boldsymbol{q})$ of $\boldsymbol{q}$. This neighborhood can be defined using a hyper square, hyper sphere, etc. (Note that this descriptor will not be invariant under rigid body transformations if an axis-aligned hyper square is used.) The proposed descriptor is very efficient to compute, but sensitive to noise. The distribution of the descriptor values for various 3D models is shown in Fig. 1.

2.2 Computing the descriptor

In order to calculate the descriptor value of a point, we have to determine its local neighbors. A common approach for this task is to use a k-nearest neighbor (k-NN) search algorithm. Algorithms that partition the search space into sub-spaces for efficient search such as k–d trees are commonly used for k-NN queries. However, exact k-NN queries become impractically slow when the neighborhood sizes are large. In addition, the time complexity of the algorithm grows exponentially with the dimension [14]. These algorithms and data structures are typically designed to find the k-NN of an arbitrary point in any dimensional space. But in many research problems of computer graphics involving a PC, such as simplification of surfaces [15], extraction of feature points [16], computation of shape descriptors [7, 10, 11], and estimation of surface normals [13], one needs to determine the local neighbors of each point in the PC rather than finding those of an arbitrary point in 3D space. Hence, this problem can be considered as a restricted version of the k-NN search where the query points are the elements of a PC. The algorithm that we propose in the next section exploits this fact to find the local neighbors of each point in a PC

efficiently. It is an extension of the algorithm originally proposed by Nene et al. [14].

3 Calculating the local neighbors of a point

In the following section, we first introduce the algorithm proposed by Nene et al. [14] (for details we refer the reader to the paper). In Sect. 3.2, we show how this algorithm can be extended to find the local neighbors of each point in a PC rather than its nearest neighbor only. We call this new algorithm *fast incremental search (FINS)*.

3.1 Nearest neighbor search using Nene's algorithm

The basic idea behind Nene's algorithm is to restrict the search space for finding the closest point to a query point $\boldsymbol{q}$ to a small hyper square, centered at $\boldsymbol{q}$, with a size of 2ϵ (Fig. 2a). This is achieved by first sorting all points in the data according to their coordinates in each dimension and then performing binary searches on the sorted lists to define the boundaries of the square. Then, this small hyper square is searched for the point closest to $\boldsymbol{q}$.

A list of indices P_n is constructed for each dimension n such that it contains the points $\boldsymbol{p}_i$ that have the n^{th} components within the limits of the hyper square. Then lists P_n are intersected to find the points within the square. The complexity of all the intersections is at most $O(|P_{\min}|N_{\text{D}})$, where $P_{\min}$ is the list with the smallest number of elements and N_{D} is the number of dimensions.

3.2 The proposed algorithm

In Nene's algorithm, the choice of ϵ is critical since the goal is to find the nearest point to a query point in an efficient manner. If it is chosen too small, then there will be no points in the hyper square other than the query point itself. If it is chosen too large, the search will be inefficient [14]. In our problem, we are interested in finding the local neighbors of a point efficiently rather than the near-

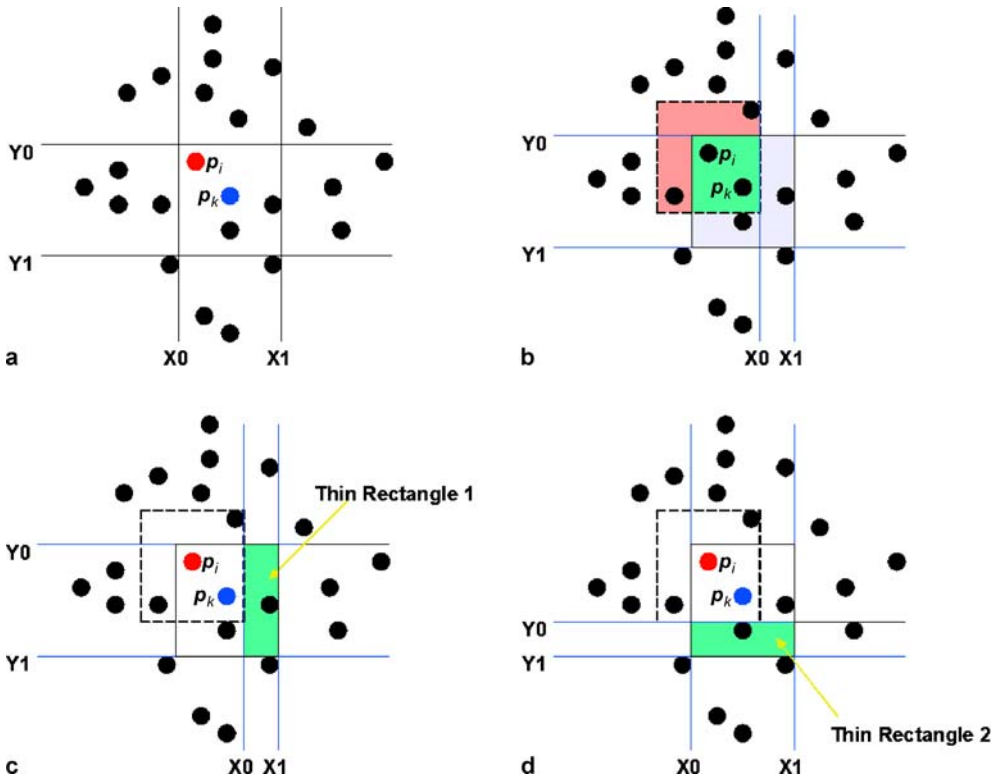

Fig. 2. a An inefficient way to find the neighboring points of $\boldsymbol{p}_k$: First, the points that are between the lines X0 and X1 (11 points without counting $\boldsymbol{p}_k$) and the ones between the lines Y0 and Y1 (10 points without counting $\boldsymbol{p}_k$) are found and stored in two different sets. Then, the intersection of these two sets is calculated, which requires a total of ten operations. **b**, **c**, **d** In contrast, the search can be restricted to the thin rectangles only by using the points contained in the square around $\boldsymbol{p}_i$ (the square with *dashed lines*). The *green colored region* in **b** shows the intersection of the squares of $\boldsymbol{p}_i$ and $\boldsymbol{p}_k$. In **c**, there are only three points between X0 and X1, hence only three operations are necessary to find the points within the thin rectangle 1. Similarly, in **d**, there are only two points between Y0 and Y1 and thus two operations are necessary. In total, five operations are necessary to find the points within the thin rectangles and two more operations are necessary to find the points in the intersection region in **b** (since there are two points in the square around $\boldsymbol{p}_i$, not counting $\boldsymbol{p}_i$ and $\boldsymbol{p}_k$) instead of ten operations as in **a**

est neighbor only. Hence, ϵ can be a large value depending on the number of local neighbors to be found. Increasing ϵ will also increase the number of points in lists P_n and hence the number of operations necessary for calculating the intersections. Since the number of operations to calculate all intersections is $O(|P_{\min}|N_D)$ in the worst case scenario, if $|P_{\min}|$ is a small value, then all the intersection operations can be executed efficiently. For example, if $|P_{\min}| = 5$, only five integer comparisons are necessary for each intersection operation in the worst case.

In order to extend Nene's algorithm to search for local neighbors of a point, we observe that if the distance between two points $\boldsymbol{p}_i$ and $\boldsymbol{p}_k$ is sufficiently small, then most of the neighboring points of $\boldsymbol{p}_k$ will also be the neighboring points of $\boldsymbol{p}_i$ (Fig. 2b). As shown in Fig. 2c and d, the points that are not in the overlapping region of hyper squares of $\boldsymbol{p}_i$ and $\boldsymbol{p}_k$ lie in the thin hyper rectangles. Nene's algorithm can be used to determine the points in these thin rectangles efficiently by exploiting the fact that if the number of elements in the list $P_{\min}$ is small (see X0–X1 region in Fig. 2c and Y0–Y1 region in Fig. 2d), then the intersection operations discussed in Sect. 3.1 can be executed efficiently. Hence, one can take advantage of the incremental movement of the hyper square from $\boldsymbol{p}_i$ to $\boldsymbol{p}_k$ to find the neighbors of $\boldsymbol{p}_k$ efficiently once the neighbors of $\boldsymbol{p}_i$ are determined. Note that one can easily get the points within the limits of the thin rectangles for n^{th} dimension by accessing the sorted lists of the n^{th} components starting from the neighbors of pi, without any binary search. The complete algorithm is given in Algorithm 1.

Algorithm 1. FINS

$V = \emptyset$
i = index of an arbitrary point in P
while there is an unvisited point in P **do**
  if V is empty or all points in SP_i are visited **then**
    i = index of an arbitrary unvisited point in P
    V = set of points in the square centered at $\boldsymbol{p}_i$
    SP_i = list of points of V sorted in ascending order based on their distances to $\boldsymbol{p}_i$
    label $\boldsymbol{p}_i$ as visited
  else
    k = index of the first unvisited point in SP_i
    move the square such that it is centered at $\boldsymbol{p}_k$
    T = set of points within thin rectangles
    V = set of points in $V \cup T$ that are enclosed by the square centered at $\boldsymbol{p}_k$
    SP_k = list of points of V sorted in ascending order based on their distances to $\boldsymbol{p}_k$
    label $\boldsymbol{p}_k$ as visited
    $i = k$
  end if
end while

Note that Nene's algorithm is a general purpose algorithm for finding the nearest neighbor of an arbitrary point located anywhere in N_D dimensional space whereas the query point in our algorithm must be a member of a PC in order to exploit the incremental movements of the hyper square.

3.3 Time complexity of FINS

The execution time of the FINS algorithm depends on the number of queries in which the thin hyper rectangles are utilized to determine the neighboring points of a query point. We call these queries "efficient" queries. We have experimented with PC representation of various 3D models as well as with PCs with no geometric representation in higher dimensional spaces (i.e. randomly distributed points) and observed that the percentage of efficient queries to the total number of queries was always higher than 90% for neighborhood sizes larger than 10σ (see Fig. 3a), where σ is defined as the average of distances from each point in a PC to its closest neighbor.

In practice, there is no need to maintain a complete list of all neighbors of each point. Only a few neighbors in very close proximity of each point are sufficient to find the close neighbors of all points efficiently (Fig. 3b).

We compared the performance of the FINS algorithm implemented in Java with the state-of-the-art ANN library [2] implemented in C++. The results are shown in Fig. 3c and d. The nearest neighbor queries were executed for each point and the total execution times were reported in the figure. In our experiments, the number of nearest neighbors returned by the ANN library was adjusted such that they were contained within a sphere of radius ϵ. Recall that FINS returns the neighboring points within a cube of size 2ϵ. We made a simple modification in FINS (labeled as FINS+K in the plots to emphasize the difference) so that it returns a sorted list of neighbors in a sphere of radius ϵ. The results for both randomly distributed points and the dragon model show that the performance of FINS is superior to the ANN library. In particular, the performance difference gets larger as the sampling density is increased. Since the PCs that are sampled from real world objects are typically dense containing millions of points, FINS can be very effective in processing these data sets.

3.4 Setting the size of the hyper square in FINS

For computational efficiency and robustness, the size of the hyper square must be restricted to 2ϵ. In many problems, determining the suitable neighborhood size is often difficult and heuristic approaches are typically used. We have developed an algorithm that enables the user to set the number of neighbors to N_{TARGET} (with a deviation of N_{DEV}) rather than choosing ϵ directly. We propose the following simple algorithm for this purpose:

1. Select a number of sample points, N_{SAMPLE}, randomly from PC and set ϵ_0 to an initial value. $k \leftarrow 0$.
2. Calculate the set of closest neighbors SP_i for $\boldsymbol{p}_i$, $1 \leq i \leq N_{\text{SAMPLE}}$ using ϵ_k.
3. Set $n_{\min}$ = the number of SP_i satisfying $|SP_i| < N_{\text{TARGET}} - N_{\text{DEV}}$
4. Set $n_{\max}$ = the number of SP_i satisfying $|SP_i| > N_{\text{TARGET}} + N_{\text{DEV}}$

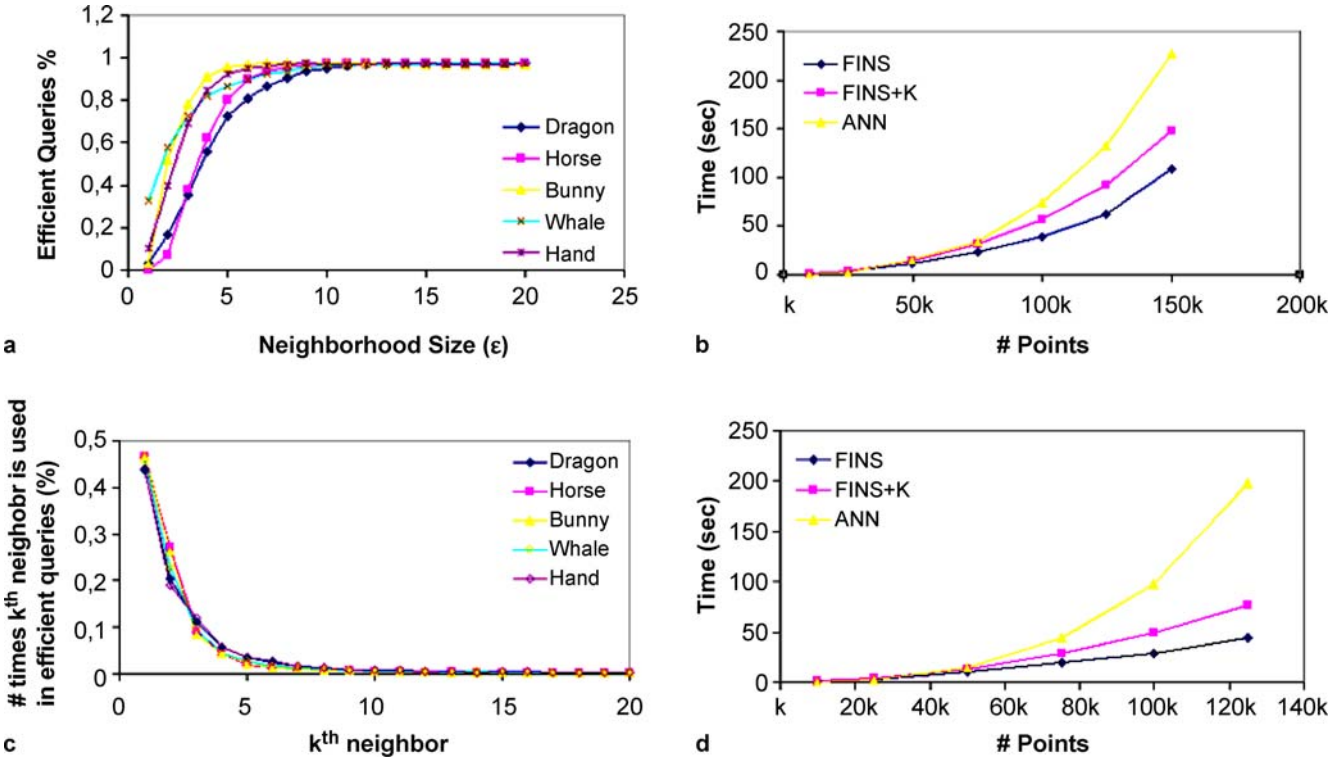

Fig. 3. a The ratio of "efficient" queries to the total number of all queries is plotted as a function of neighborhood size ϵ, which is shown in units of σ in the plot. **b** Total running times of the ANN library and FINS are plotted as a function of the number of points. The points used in the analysis are randomly distributed within a 3D cube and are used as both data and query sets. Thus, the number of queries is equal to the number of data points. The neighborhood size is set to 10% of the size of the cube. **c** The relation between the k^{th} neighbor and the number of times that neighbor is used in efficient queries. Only few close neighbors (5) of each point are essentially used in the efficient queries. **d** Multi-resolution representations of the dragon model are obtained using QSlim [9] and the total running times of the ANN library and FINS are plotted as a function of the number of points in the simplified models. The neighborhood size is set to 10% of the smallest dimension of the oriented bounding box of the model

5. Set $\epsilon_{k+1} = \epsilon_k + \sigma \frac{n_{\min} - n_{\max}}{N_{\text{SAMPLE}}}$, where σ here is estimated using the sample points. $k \leftarrow k+1$.
6. Iterate steps 2, 3, 4, 5 until $\frac{|\epsilon_{k+1} - \epsilon_k|}{\epsilon_k} < 0.05$ or the maximum number of iterations $N_{\text{MAX\_ITER}}$ is reached.

This algorithm attempts to minimize the number of neighborhoods that contain number of points fewer than $N_{\text{TARGET}} - N_{\text{DEV}}$ or more than $N_{\text{TARGET}} + N_{\text{DEV}}$ by adjusting ϵ. The factor σ provides stability such that the increments in ϵ are comparable to the distances between the points.

4 Feature selection

After assigning a descriptor value to each point using FINS, the next step is to identify the feature points based on the descriptor values. The main idea at this step is to select the points with distinct descriptor values as the feature points. This idea works well if the descriptor has a high discriminatory power. If not, as in the case of our descriptor, a one-step selection process may result in many feature points with no distinct characteristics. In addition, noise and outliers in data may adversely affect the selection process. For these reasons, we propose the following four-stage feature detection method which works well even with descriptors having low discriminatory powers.

4.1 A four-stage process for feature detection

1. Calculate a descriptor value for each point of a PC using the proposed geometric descriptor. Note that the local neighbors are determined by the FINS algorithm (Sects. 2 and 3).
2. Construct a histogram of descriptor values as suggested in [10] and then select the points in the bins that contain number of points less than a threshold value. The number of bins is set according to Scott's rule and the threshold is set to one percent of the number of points of the PC as suggested in [10].
3. Group the points selected in stage 2 based on their proximity to each other. Eliminate the groups containing a number of points less than the mean minus the standard deviation of the total number of points in all

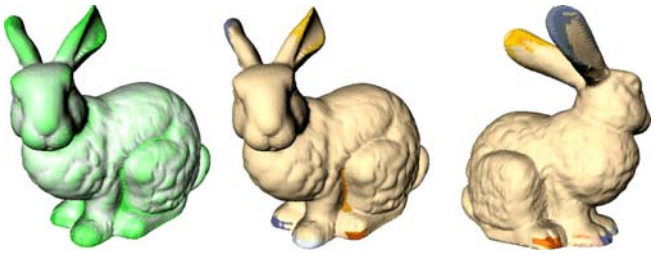

Fig. 4. The distribution of the descriptor values for the Stanford Bunny is shown in the first figure on the *left* (*darker green* indicates higher descriptor values) while the other two figures show the feature regions displayed in different colors

groups (see the feature regions for the Stanford Bunny in Fig. 4).
4. Select the center of mass of each group as the feature point of the PC.

The first two stages of this process are typical steps of a feature selection process while the last two are newly introduced to select feature points of a PC using a descriptor with a low discriminatory power.

4.2 Grouping (clustering) of distinct points

To cluster the points selected in stage 2 into feature regions, we have developed a simple recursive algorithm. The algorithm starts from an arbitrary point and inserts it into an initially empty list $Group_i$. Then, all the points that are less than a threshold distance th_C away from this arbitrary point are visited recursively and added to the list $Group_i$.

5 Finding correspondences

Once the feature points are identified on both data sets P and M, we have to establish correspondences between them. Euclidean distance between two points is invariant under orthogonal transformation (i.e. rigidity constraint) and can be used to find the corresponding feature points.

We propose a new algorithm based on the rigidity constraint, named *matching difference vectors (MDV)*, to find the corresponding points in PCs being registered.

5.1 Earlier work

The current methods for the correspondence finding problem aim to reduce the number of points used for finding the correspondences either by taking samples from the data [5], or using feature extraction methods [10, 17], or restricting the search space using special targets attached to the physical objects [1, 4]. The last strategy has a limited application area while the accuracy of the former methods heavily depends on the sampling strategy or the feature extraction technique as well as the shape characteristics of the objects to be registered. Moreover, the sampling and feature-based techniques suggested in [6, 10, 17] try to exploit the fact that the PCs being registered have an underlying surface representation, which may further constrain their usage in some applications.

MDV makes no assumptions about the underlying structure of PCs and the sampling strategy. It is efficient and robust to outliers, but sensitive to noise in the data. MDV can be used directly without any preprocessing if there is not much noise in PCs. If the noise level is significant, we suggest using the proposed four-stage feature selection method as a precursor to MDV. In Sect. 5.2, we present several observations leading to MDV. These observations enable MDV to find the corresponding pairs in PCs efficiently.

5.2 Observations

1. If three correspondences can be found correctly (i.e. one pair of corresponding triangles), a unique transformation between P and M can be calculated [5, 17]. More than three correspondences may be used for robustness [10].
2. One corresponding pair is actually enough to calculate the optimum translation. One can then use the fact that $\|\boldsymbol{Rx}\| = \|\boldsymbol{x}\|$ to restrict the search space. Hence, for a given point $\boldsymbol{p}_i \in P$, it is sufficient to search for its corresponding point $\boldsymbol{m}_j \in M$ such that they have the same magnitudes. If the points in M are sorted according to their distances from the origin, then $\boldsymbol{m}_j$ can be searched in a spherical shell, centered at the origin and having a radius of $\|\boldsymbol{p}_i\|$, using a binary search.
3. Given a corresponding pair $(\boldsymbol{p}, \boldsymbol{m})$, the close neighbors of $\boldsymbol{p}$ and $\boldsymbol{m}$ are very likely to make corresponding pairs as well. Note that this observation may not hold if the sampling densities of M and P are very different. This third observation is similar to the "proximity constraint" proposed by Liu et al. to improve the performance of the original ICP algorithm [12].

5.3 Our approach: Matching difference vectors (MDV)

Given two points $\boldsymbol{p}_i$ and $\boldsymbol{m}_j$, we seek to determine if they are corresponding points. Let SP_i and SM_j be the lists of points that are in the close neighborhood of $\boldsymbol{p}_i$ and $\boldsymbol{m}_j$, respectively. The points in both lists are sorted according to their distances to $\boldsymbol{p}_i$ and $\boldsymbol{m}_j$, respectively. For each point in SP_i, we search for its distance from $\boldsymbol{p}_i$ among the distances between $\boldsymbol{m}_j$ and each point in SM_j. If $(\boldsymbol{p}_k, \boldsymbol{m}_k)$ represent the k^{th} pair that satisfies the distance criterion $(|\|\boldsymbol{p}_i - \boldsymbol{p}_k\| - \|\boldsymbol{m}_j - \boldsymbol{m}_k\|| < th_{\text{DI}})$, then we check if the following distance criterion is also satisfied

$$|\|\boldsymbol{p}_l - \boldsymbol{p}_k\| - \|\boldsymbol{m}_l - \boldsymbol{m}_k\|| < th_{\text{DI}} \tag{2}$$

for each l, where $(\boldsymbol{p}_l, \boldsymbol{m}_l)$ is a matching pair that satisfies both threshold criteria and is already stored in a list

$$Z = \{(\boldsymbol{p}_l, \boldsymbol{m}_l)\} \quad 1 < l < k.$$

If the number of pairs in Z, $|Z|$, is greater than a certain threshold, $\boldsymbol{m}_j$ is accepted as one of the possible corresponding points of $\boldsymbol{p}_i$. If there is only one such point, then $(\boldsymbol{p}_i, \boldsymbol{m}_j)$ is accepted as the true corresponding pair and added to the final list of corresponding pairs CP. If $|CP| \geq 5$, MDV is terminated (refer to Observation 1). Moreover, the search space in M is restricted to a spherical shell (refer to Observation 2) after finding the first true corresponding pair.

In general, a small value for the lower limit of $|Z|$ (i.e. a small number of matching pairs in the neighborhoods of $\boldsymbol{p}_i$ and $\boldsymbol{m}_j$) is sufficient to decide if $(\boldsymbol{p}_i, \boldsymbol{m}_j)$ is a corresponding pair (refer to Observation 3). In our implementation, this limit is $|Z| \geq |SM_j|/5$. A higher limit improves the robustness but reduces the efficiency of the algorithm since there will be only a few pairs passing the limit test.

The pseudocode of the algorithm is given in Algorithm 2.

Algorithm 2. MDV

Input: P, M, SP_i , SM_j
Output: Set of corresponding points CP
$Z = \emptyset$, $CP = \emptyset$, found = false
for all $\boldsymbol{p}_i \in P$ **do**
  if $|CP| > 0$ **then**
    M_{SEARCH} = do binary range search on S for points $\boldsymbol{m}_j \in M$ that satisfy $|\|\boldsymbol{p}_i\| - \|\boldsymbol{m}_j\|| < th_{\text{DI}}$
  else
    $M_{\text{SEARCH}} = M$
  end if
  for all $\boldsymbol{m}_j \in M_{\text{SEARCH}}$ **do**
    for all $\boldsymbol{p}_k \in SP_i$ **do**
      do binary range search on SM_j for points $\boldsymbol{m}_k$ SM_j that satisfy $|\|\boldsymbol{p}_i - \boldsymbol{p}_k\| - \|\boldsymbol{m}_j - \boldsymbol{m}_k\|| < th_{\text{DI}}$
      if $\boldsymbol{m}_k$ is unique and $|\|\boldsymbol{p}_l - \boldsymbol{p}_k\| - \|\boldsymbol{m}_l - \boldsymbol{m}_k\|| < th_{\text{DI}}$ for each $(\boldsymbol{p}_l, \boldsymbol{m}_l) \in Z$ **then**
        add $(\boldsymbol{p}_k, \boldsymbol{m}_k)$ to Z
      end if
    end for
    if $|Z| > |SM_j|/5$ **then**
      if (found) **then**
        found = false
        break
      end if
      found = true
      store $\boldsymbol{m}_j$ as $\boldsymbol{m}_c$
    end if
  end for
  if(found) **then**
    add $(\boldsymbol{p}_i, \boldsymbol{m}_c)$ to CP
    if this is the first corresponding pair $(\boldsymbol{p}_i, \boldsymbol{m}_c)$ found **then**
      set $\boldsymbol{p} = \boldsymbol{p} - \boldsymbol{p}_i$ and $\boldsymbol{m} = \boldsymbol{m} - \boldsymbol{m}_c$ for all $\boldsymbol{p} \in P$ and $\boldsymbol{m} \in M$
      sort all the points in M according to their distances to the origin and call this sorted list as S
      (see Observation 2)
    end if
  end if
  $Z = \emptyset$, found = false
  if $|CP| \geq 5$ (see Observation 1) **then**
    break
  end if
end for

5.4 The effect of noise on MDV

One can directly use MDV on P and M to find the correspondences even without detecting their feature points if the noise magnitude in the data is low. However, once the noise magnitude is comparable to the average of distances from each point in a PC to its closest neighbor (σ), MDV may fail to find the true correspondences. To account for the high magnitudes of noise, the threshold value for distance (th_{DI}) must to be increased, which in turn increases the number of points that satisfy the threshold criteria. As a result, for a given query point in P, finding a unique corresponding point in M becomes more difficult or even impossible. If the noise magnitude is high, the proposed four-stage feature detection process must be used prior to MDV.

6 Results

6.1 Data, noise and outliers, parameters

We used the 3D models from the Princeton Shape Benchmark project [8] and the Large Geometric Models Archive of Georgia Institute of Technology [18] in our registration experiments. We added independent, uniform, and random noise with zero mean. The magnitude of the noise is varied as integer multiples of σ (see Fig. 5a). Outliers are also randomly distributed within the bounding box of the models. Moreover, after adding noise and outliers, the order of the points in each model is changed randomly for each trial of the registration experiments.

In all experiments with the proposed feature-based method, the following values for the parameters are used (see the conclusion section for a detailed discussion on how to set these parameters): th_{DI} is set to the magnitude of noise. For Algorithm 1 (Sect. 4), th_{C} is set to 4σ. The parameter ϵ is calculated as explained in Sect. 3.4 while N_{SAMPLE}, N_{TARGET}, N_{DEV}, and $N_{\text{MAX\_ITER}}$ are set to (the number of points in the model/500), 200, 10, and 100, respectively.

6.2 Results

We are able to register all models that contain noise up to 5σ successfully (Fig. 5b). To evaluate the quality of our registration method, we apply arbitrary rotation matrix $\boldsymbol{R}$ and translation vector $\boldsymbol{t}$ to the points in P and then compare this transformation with the corresponding one obtained through the registration process $(\boldsymbol{R}_p, \boldsymbol{t}_p)$. We define the following error metrics to measure the quality of the registration: $e_{\text{ROT}} = \|\boldsymbol{R}_p\boldsymbol{R} - \boldsymbol{I}\|$ and $e_{\text{TRANS}} = \|\boldsymbol{t}_p + \boldsymbol{t}\|$, where $\boldsymbol{I}$ is the 3×3 identity matrix and $\|.\|$ represents the Euclidean norm.

We conducted our experiments with a Pentium (R) 1.86 GHz CPU system having 1GB RAM and running under Windows OS. The software code was written in Java. The computation times for all the models are tabulated in Table 1.

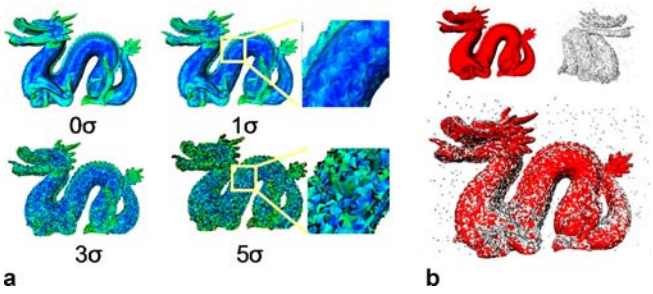

Fig. 5. a The dragon model with different magnitudes of noise (0σ, 1σ, 3σ, 5σ). As the magnitude of noise is increased, the distinct features of the model start to disappear. As a result, desired and also some undesired points may be selected as distinct points in stage 2 (refer to Sect. 4.1). Since most of these undesired points are randomly scattered around the different regions of the model, they make small groups and are eliminated in stage 3. **b** Registration of the original dragon model to its transformed noisy copy using the proposed feature-based registration method. The noise level is 3σ and the number of outliers is 1% of the total number of points in the model (109k)

6.3 Performance of MDV as a standalone algorithm

As mentioned earlier, MDV can be used as a standalone algorithm to register PCs even without detecting their feature points if the magnitude of noise in the data is lower than 0.1σ.

To investigate the performance of MDV as a standalone algorithm without extracting features, we conducted registration experiments with PCs having low noise and a high number of outliers. Since the magnitude of noise was low, the parameter N_{TARGET} in FINS was set to a lower value of 50 to improve the efficiency. We tested the performance of MDV under three different conditions:

(1) M and P are PCs of the whole model.
(2) M is the PC of a whole model and P is a randomly selected subset of M containing $\frac{1}{4}$ of points in M.
(3) M and P have overlapping regions of varying ratio, but P is not a subset of M.

Each experiment was repeated five times and the order of points in the PCs was randomized in each trial. The average of the running times of those five trials is reported in all tables and graphs. The number of outliers added artificially to each PC was always 10% of its number of points and they were distributed randomly within the bounding box of the PC. This excessive number of outliers was used to illustrate the robustness of MDV (see Fig. 6a). The

Table 1. The performance of the proposed feature-based registration method. All reported times (FINS, Feature, MDV) are in seconds and include the computations performed on both PCs (P and M). P is a transformed copy of the M. Noise (3σ) and outliers (1%) are added to both PCs independently

Shape	#Points	#Outliers	FINS	Feature	MDV	e_{ROT}	$e_{\text{TRANS}}(\sigma)$
Dragon	109411	1094	39.248	7.939	0.250	0.0395	2.9000
Horse	48485	484	11.096	3.781	0.047	0.0011	0.1220
Bunny	35947	359	6.790	2.960	0.047	0.0321	0.5143
Fish	18143	181	2.716	1.848	0.125	0.0130	0.5251
Hand	7929	79	0.998	0.630	0.047	0.1023	1.8160

Table 2. The performance of MDV as a standalone algorithm under the first and second conditions discussed in Sect. 6.3. All reported times are in seconds and include the computations performed on both PCs (P and M). P is a transformed copy of the M. Noise (0.1σ) and outliers (10%) are added to both PCs independently

Shape	#Points	#Outliers	Pre time	Exec time	e_{ROT}	$e_{\text{TRANS}}(\sigma)$
			(a)			
Dragon	109411	10941	15.578	3.172	0.0002	0.0009
Horse	48485	4848	5.828	1.922	0.0001	0.0084
Bunny	35947	3594	3.281	1.157	0.0005	0.0170
Fish	18143	1814	1.797	0.969	0.0003	0.0012
Hand	7929	792	0.860	0.468	0.0005	0.0082
			(b)			
Dragon	109411	10941	16.359	3.123	0.0002	0.0004
Horse	48485	4848	5.367	2.015	0.0001	0.0005
Bunny	35947	3594	3.164	1.188	0.0001	0.0003
Fish	18143	1814	1.847	1.057	0.0001	0.0011
Hand	7929	792	0.930	0.792	0.0005	0.0082

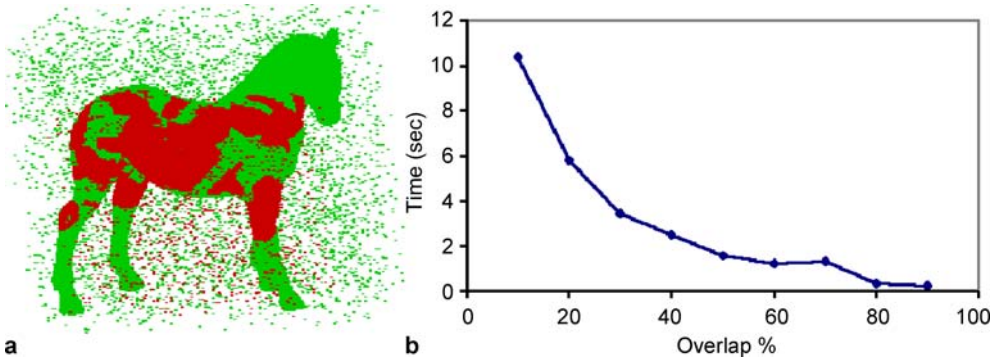

Fig. 6. a The points randomly selected from the Horse model (shown in *red*) are registered to the whole model shown in *green color* (Condition 2 in Sect. 6.3) using MDV standalone (i.e. without finding feature points). Random noise of 0.1σ is added to both PCs. Note the significant number of outliers in both PCs (10%). **b** The relation between the ratio of overlapping regions (Condition 3 in Sect. 6.3) and the total execution time for the dragon model when MDV is used as a standalone algorithm. The number of points in the PCs is held constant around 10k for each overlapping ratio

results of the experiments conducted under the first and second conditions are presented in Tables 2a and b, respectively. In both tables, we call the amount of time it takes to find the neighbors of each point in P and M (using the FINS algorithm) the "Pre Time" and the execution time of MDV the "Exec Time". As the ratio of overlapping regions is decreased (the third condition), the probability that a given query point in P will have a correspondence in M also decreases, which results in an increase in the execution times (Fig. 6b).

7 Discussion and conclusion

We developed a new feature-based registration method for aligning PCs containing noise and outliers. As a part of this development, we introduced a simple geometric descriptor and a new k-NN search algorithm (i.e. FINS) to find the neighbors of each point in a PC efficiently. Although the proposed descriptor does not have a high discriminatory power, distinct features of a PC can be identified successfully when it is used with the proposed four-stage feature selection method. For this purpose, first the close neighbors of each point are determined using the FINS algorithm and its descriptor value is calculated using simply the coordinates of its neighbors. Second, the points with distinct descriptor values are selected using a histogram. Third, the selected points are clustered into groups and the ones containing an insignificant number of points are eliminated using simple statistics. Fourth, the center of mass of points in each remaining group is defined as the feature point of the PC. Following the detection of all feature points in both PCs using the proposed four-stage method, a novel algorithm, called MDV, is executed to find the correspondences between the feature points.

We should emphasize that the proposed geometric descriptor is sensitive to noise, but this does not prevent the selection of compatible feature points in our approach. As the magnitude of noise is increased, more points are selected from the different regions of PCs that do not contain distinct features (see Fig. 5a). Since these points are randomly scattered, they form small groups only. These groups are mostly eliminated at the third stage of the proposed method using simple statistics.

Although we show the application of our method to the registration of PCs in 3D space only, FINS, MDV and the whole process can be used in higher dimensional problems without making any assumptions about the underlying geometry of the PCs.

There are three important parameters set by the user in our registration method. The parameter N_{TARGET} used in the FINS algorithm defines the desired number of neighbors of a point. We set N_{TARGET} to 200 in our experiments. Although a more elegant method could be developed based on the number and the distribution of points, our effortless selection worked very well for all the models used in this study (note that the models contain a number of points that range from 7k to 100k, noise up to 5σ, and a significant number of outliers). Another parameter th_{C} is a threshold for clustering the points with distinct descriptor values into groups based on their proximity to each other. The value of th_{C} can be set approximately to a few multiples of σ. As th_{C} is increased, one goes from several small feature regions scattered around the model to a few large regions only. Small and scattered feature regions are eliminated at the third stage of the proposed feature selection method. Finally, th_{DI} is another threshold parameter used in MDV to decide if the magnitudes of two vectors influenced by the noise are equal. In this regard, the value of th_{DI} can be set approximately equal to the magnitude of noise.

References

1. Akca, D.: Full automatic registration of laser scanner point clouds. In: Optical 3-D Measurement Techniques VI, pp. 330–337. Zurich (2003)
2. Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., Wu, A.Y.: An optimal algorithm for approximate nearest neighbor searching fixed dimensions. J. ACM **45**(6), 891–923 (1998). doi: http://doi.acm.org/10.1145/293347.293348, http://www.cs.umd.edu/∼mount/ANN
3. Besl, P.J., McKay, H.D.: A method for registration of 3-d shapes. IEEE Trans. Pattern Anal. Mach. Intell. **14**(2), 239–256 (1992)
4. Bornaz, L., Lingua, A., Rinaudo, F.: A new software for the automatic registration of 3d digital models acquired using laser scanner devices. In: CIPA WG6 International Workshop on Scanning for Cultural Heritage Recording, pp. 52–57. Corfù (2002)
5. Chen, C.-S., Hung, Y.-P., Cheng, J.-B.: RANSAC-based DARCES: A new approach to fast automatic registration of partially overlapping range images. IEEE Trans. Pattern Anal. Mach. Intell. **21**(11), 1229–1234 (1999)
6. Dorai, C., Wang, G., Jain, A.K., Mercer, C.: Registration and integration of multiple object views for 3d model construction. IEEE Trans. Pattern Anal. Mach. Intell. **20**(1), 83–89 (1998)
7. Frome, A., Huber, D., Kolluri, R., Bulow, T., Malik, J.: Recognizing objects in range data using regional point descriptors. In: European Conference on Computer Vision (ECCV), pp. 224–237. Springer, Prague (2004)
8. Funkhouser, T., Kazhdan, M., Min, P., Shilane, P.: Shape-based retrieval and analysis of 3D models. Commun. ACM **48**(6), 58–64 (2005)
9. Garland, M.: QSlim: Simplification Software. http://graphics.cs.uiuc.edu/∼garland/software/qslim.html
10. Gelfand, N., Mitra, N.J., Guibas, L.J., Pottmann, H.: Robust global registration. In: Desbrun, M., Pottmann, H. (eds.) Proceedings of the Third Eurographics Symposium on Geometry Processing (Vienna, Austria, 2005), ACM Int. Conf. Proc. Ser., vol. 255, pp. 197–206. Eurographics Association, Aire-la-Ville (2005)
11. Johnson, A.E., Hebert, M.: Using spin images for efficient object recognition in cluttered 3d scenes. IEEE Trans. Pattern Anal. Mach. Intell. **21**(5), 433–449 (1999)
12. Liu, Y., Rodrigues, M.A., Qian, W.: Using neighboring relationships to eliminate false matches for accurate registration of free-form surfaces. J. Digital Imaging **15**(1), 267–269 (2002)
13. Mitra, N.J., Nguyen, A.: Estimating surface normals in noisy point cloud data. In: SCG '03: Proceedings of the Nineteenth Annual Symposium on Computational Geometry, pp. 322–328. ACM, New York, NY (2003)
14. Nene, S.A., Nayar, S.K.: A simple algorithm for nearest neighbor search in high dimensions. IEEE Trans. Pattern Anal. Mach. Intell. **19**(9), 989–1003 (1997). doi: http://dx.doi.org/10.1109/34.615448
15. Pauly, M., Gross, M., Kobbelt, L.P.: Efficient simplification of point-sampled surfaces. In: Proceedings of the IEEE Conference on Visualization, pp. 163–170. IEEE Computer Society, Boston, MA (2002)
16. Pauly, M., Keiser, R., Gross, M.H.: Multi-scale feature extraction on point-sampled surfaces. Comput. Graph. Forum **22**(3), 281–290 (2003)
17. Roth, G.: Registering two overlapping range images. In: Proceedings of the Second International Conference on 3-D Digital Imaging and Modeling, pp. 191–200. IEEE, Ottawa (1999)
18. Turk, G., Mullins, B.: Large geometric models archive. http://www.cc.gatech.edu/projects/large_models/

A.C. OZTIRELI is currently a graduate student in the Computer Graphics Laboratory at ETH-Zurich. He graduated with a summa cum laude distinction from the Computer Engineering Department of Koc University in 2006.

C. BASDOGAN is a member of faculty in the College of Engineering at Koc University. Before joining Koc, he worked at NASA-JPL/Caltech, MIT, and Northwestern University Research Park. He received his Ph.D. degree from Southern Methodist University in 1994. Dr. Basdogan conducts research and development in the areas of haptics, computer graphics, virtual reality technology, biomechanics, robotics, control systems, and mechatronics.