Efficient Algorithms for Robust Feature Matching*

David M. Mount† Nathan S. Netanyahu‡
Jacqueline Le Moigne§

June 29, 1998

Abstract

One of the basic building blocks in any point-based registration scheme involves matching feature points that are extracted from a sensed image to their counterparts in a reference image. This leads to the fundamental problem of point matching: Given two sets of points, find the (affine) transformation that transforms one point set so that its distance from the other point set is minimized. Because of measurement errors and the presence of outlying data points, it is important that the distance measure between the two point sets be robust to these effects. We measure distances using the partial Hausdorff distance.

Point matching can be a computationally intensive task, and a number of of theoretical and applied approaches have been proposed for solving this problem. In this paper we present two algorithmic approaches to the point matching problem, in an attempt to reduce its computational complexity, while still providing a guarantee of the quality of the final match. Our first method

*To appear in *Pattern Recognition*. A preliminary version of this paper appeared in *Proceedings of the CESDIS Image Registration Workshop*, NASA Goddard Space Flight Center (GSFC), Greenbelt, MD, 1997, and NASA Publication CP-1998-206853, pp. 247–256.

†Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland. Email: `mount@cs.umd.edu`. The support of the National Science Foundation under grant CCR–9712379 is gratefully acknowledged.

‡Center for Automation Research, University of Maryland, College Park, and Space Data and Computing Division, NASA/GSFC, Greenbelt, Maryland. Email: `nathan@cfar.umd.edu`. This research was carried out, in part, while the author was also affiliated with the Center of Excellence in Space Data and Information Sciences (CESDIS), Code 930.5, NASA/GSFC. The support of the Applied Information Sciences Branch (AISB), Code 935, NASA/GSFC, under contract NAS 5555-37 and grant NAG5-6699 is gratefully acknowledged.

§Universities Space Research Association/CESDIS, Code 930.5, Space Data and Computing Division, NASA/GSFC, Greenbelt, Maryland. Email: `lemoigne@cesdis.gsfc.nasa.gov`. The support of the AISB, Code 935, NASA/GSFC is gratefully acknowledged.

is an approximation algorithm, which is loosely based on a branch-and-bound approach due to Huttenlocher and Rucklidge.[1,2] We show that by varying the approximation error bounds, it is possible to achieve a tradeoff between the quality of the match and the running time of the algorithm. Our second method involves a Monte Carlo method for accelerating the search process used in the first algorithm. This algorithm operates within the framework of a branch-and-bound procedure, but employs point-to-point alignments to accelerate the search. We show that this combination retains many of the strengths of branch-and-bound search, but provides significantly faster search times by exploiting alignments. With high probability, this method succeeds in finding an approximately optimal match. We demonstrate the algorithms' performances on both synthetically generated data points and actual satellite images.

Keywords: Image registration, point pattern matching, Hausdorff distance, approximation algorithms.

1 Introduction

In pursuing an image registration task, we are given two images of roughly the same scene, and are asked to determine the transformation that most nearly maps one image into the other. This problem is of particular interest in remote sensing and medical imaging. Our interest in the problem stems from remote sensing applications, where it is known that two satellite or aerial images correspond to roughly the same geographic region, but the exact alignment between the images is not known.

There are two main approaches to image registration. One approach makes direct use of the original data (or edge gradient data) and the other is based on feature matching. Features may be control points, corners, line segments, etc. They are assumed to be available as a result of applying standard feature extraction algorithms.

The first approach, which is based on a correlation measure between the two images, is computationally expensive, and is also sensitive to noise unless substantial preprocessing is employed. See Brown's survey paper[3] for more detail. Feature-based methods, on the other hand, tend to yield more accurate results, as features are usually more reliable than intensity or radiometric values. Note, though, that feature matching may run into difficulties if the feature extraction process yields significant numbers of missing or spurious features. Feature-based methods must be robust to these effects if they are to be useful. Feature-based methods can also be computationally expensive, especially when large point sets are involved and when the transformation space has many degrees of freedom. (See the papers by Ton and Jain[4] and Alt and Guibas[5] for further information.)

In an attempt to arrive at a sound registration scheme that is both accurate and fast, we consider the problem of point matching as part of a robust feature-based matching module. We believe that such a generic module will prove to be a useful

component of a versatile image registration toolbox (currently under development by a team of researchers at NASA/GSFC [6,7]), and will enhance its overall functional capability. The goal of this paper is to explore algorithmic methods for solving the robust point matching problem. The two algorithms presented in this paper can be broadly classified with respect to the following four factors, proposed by Brown[3] for classifying any image registration method.

Feature space: The domain in which information for matching is extracted. Specifically, we consider control points that were extracted in the image domain. These could come from any source, but in most of our tests we used feature points obtained by wavelet decomposition of the images.

Search space: The class of transformations that establish the correspondence between the sensed image and the reference data. Specifically, we consider two-dimensional affine transformations, allowing for translation, rotation, scaling along each axis, and shearing. We also consider subspaces of ths class of transformations, allowing translation and rotation only.

Search strategy: We present two search strategies for finding the optimal transformation. The first is based on a branch-and-bound search of transformation space, and the second combines this with a method based on alignment judiciously chosen candidate feature points.

Similarity metric: The figure of merit assigned to a match that is determined by a specific transformation is based on the (directional) partial Hausdorff distance (described below in detail). This is a robust measure, in the sense that a fixed fraction of the data may be spurious or missing without biasing the results.

Unlike the general algorithmic problem of point pattern matching, an important element of image registration applications is that the search typically begins with *a priori* information about bounds of the transformation. This information is derived from bounds on the uncertainty in the position or orientation of the imaging system. Thus, unlike model-based pattern matching applications, which involves searching large spaces for a small number of likely matches, the goal of image registration is to produce a single match with high reliability and accuracy.

1.1 Problem Formulation and the Nature of Errors

The point pattern matching problem can be defined abstractly as follows: We are given two point sets A and B lying (conceptually, at least) in two different spaces. Also, we are given a space $\mathcal{T}$ of transformations mapping one space into the other. Finally, we are provided with some measure of the distance between any two point sets. The problem is to find the transformation $\tau \in \mathcal{T}$ that minimizes the distance between $\tau(A)$ and B.

Two important assumptions about the feature extraction process play critical roles in the rest of this paper. The first is that the extracted feature points are subject to *perturbation errors*, which result from a combination of the image digitization process, expansion or shrinkage of objects due to variations in lighting conditions, and the vagaries of the feature extraction algorithm. Depending on the feature extraction algorithms used, the magnitude of this error is typically predictable (e.g., proportional to the size of a pixel in the image).

The second source of error is the presence of *outliers*, that is, feature points from either image that are simply not present in the other image. Outliers can result from many sources: the fact that the two images cover different regions, the presence of clutter or occluding objects in either image, and the sensitivity of the feature extraction algorithm to variations in lighting, point of view, or other aspects of the imaging process. Unlike perturbation errors, the errors resulting from outliers can be unpredictably large.

A common approach to handling outliers caused by occlusion first detects the presence of occlusion, then prunes the occluded pixels out of the images (by replacing them with some special value), and finally applies a nonrobust distance measure between the pruned images.[8] In remote sensing, the most common source of obscuration is clouds. However, the danger with such an approach is that errors in the pruning heuristics may result in the elimination of desirable features. The approach used here is that of using all of the available data, but employing a distance measure that is robust to the presence of outliers.

Borrowing a term from the field of robust statistics, we can define the robustness of a similarity measure in terms of the concept of a breakdown point. We can think of the pattern matching problem as the task of estimating the parameters of the matching transformation. Define the *breakdown point* of an estimator to be the fraction of outlying data points (up to 50%) that may cause the estimator to take on an arbitrarily large aberrant value. See Donoho and Huber[9] and Rousseeuw and Leroy[10] for exact definitions. (We do not consider breakdown points in excess of 50%, since if more than half of the points are outliers, it becomes impossible to distinguish inliers from outliers in any reasonable way. In image processing applications, however, it is often possible to handle fractions of outliers larger than 50%, provided that the outliers do not "conspire" between the images.)

1.2 Partial Hausdorff Distance

In this paper we consider a well-known robust measure of similarity called the *partial Hausdorff distance*.[1,2,11] Consider the set of distances resulting from taking each point in one set, and finding the nearest point to it in the other set. Rather than taking the sum or the maximum of these distances, which may be affected by outliers, we consider the median or, in general, the kth smallest distance. More formally, given two point sets A and B, and a parameter k, $1 \leq k \leq |A|$, we define the *directed partial*

Hausdorff distance from A to B to be

$$H_k(A, B) = K^{th}_{a \in A} \min_{b \in B} \text{dist}(a, b),$$

where K^{th} returns the kth smallest element of the set, and where $\text{dist}(a, b)$ is the Euclidean distance from a to b. (Readers familiar with the work of Huttenlocher *et al.* should note that we reverse the roles of A and B here.) The parameter k is typically based on *a priori* bounds on the number of points of A that are expected to have close matches in B under the optimum transformation. These are the *inliers*.

Observe that this definition is asymmetric. When matching, it is possible to consider the directed distances both from A to B and from B to A. Although we have not considered this, it would be a straightforward extension. The "one-sided" nature of the distance function is convenient when set A is expected to arise as a pattern in a much larger subject set B, but is not as important for image registration applications. It is often more natural to express k in terms of a quantile. For $0 < q \leq 1$, define $H_q(A, B)$ to be $H_k(A, B)$, where $k = \lceil q|A| \rceil$. If $q = 0.5$, then the partial Hausdorff distance is a 50% breakdown-point estimator, since the quality of the final match cannot be affected by any subset of outliers of size less than $|A|/2$.

The final issue to be considered is the space of transformations. The most common assumption is that this is some subspace of the space of affine transformations. The set of affine transformations consists of any transformation that can be expressed as a nonsingular linear transformation followed by a translation. Common subspaces include translations only, rigid motions (translation, rotation, and possibly reflection), homothetic transformations (translation, rotation, and uniform scaling). Our algorithmic methodology can be applied to any reasonable space of transformations of bounded dimensionality.

1.3 Prior Work

A large number of papers have been written on the point pattern matching problem in the fields of computer vision, pattern recognition, and computational geometry. Most of the formulations of the point matching problem in computational geometry are not suitable for noisy, cluttered images either because they require exact matches,[12] they require 1–1 matches,[13,14] or they assume that every point in one set has a close match in the other set in terms of the (standard) Hausdorff distance.[15,16,17,18] Even under these relatively restrictive assumptions, the computational complexity can be quite high. For example, the best-known algorithm for determining the translation and rotation that minimize the Hausdorff distance between sets of sizes m and n runs in $O(m^3 n^2 \log^2 mn)$ time.[15,16] This complexity may be unacceptably high for applications involving hundreds or thousands of feature points.

Numerous point matching algorithms have been proposed and used in the fields of computer vision and pattern recognition. These range from relaxation-based

methods,[19,4] to cluster detection in transformation space (by computing point-to-point correspondences[20,21,22]), to hierarchical decomposition of transformation space coupled with the application of a robust similarity measure.[2,11,23,24] Most of the techniques presented in these papers are computationally intensive (in a worst-case theoretical sense), or take long times to run in practice.

Two robust distance measures have been studied from an algorithmic perspective. The first is the method of partial Hausdorff distance matching, introduced by Huttenlocher, Klanderman and Rucklidge.[1,2,11] The other distance measure is the *absolute difference*, recently introduced by Hagedoorn and Veltkamp.[24] Both of these algorithms are based on a branch-and-bound search of transformation space, which we will discuss in greater detail below.

The Hausdorff distance measure was also used by Kedem and Yarmovski[25] to register images for stereo matching under translation. Their method is based on extracting curved segments from the image, using the partial Hausdorff distance to find local matches between these segments, and then applying an evaluate-propagate stage to extend local matches to global matches.

In an attempt to circumvent the high complexity of point pattern matching, some researchers have considered Monte Carlo algorithms, which combine the use of sampling and of transformations obtained through point-to-point alignments. The basic idea is that the images of three noncollinear points determine a unique affine transformation in the plane. Thus, by sampling a number of triples of points (or even pairs or singletons, depending on the number of degrees of freedom in the transformation space), and enumerating all possible matching triples in the other point set, one is likely to encounter a good match. This is the basis of many matching algorithms, including those of Goshtasby and others[20,21,22] and Goodrich *et al.*,[17] and also of methods based on geometric hashing.[26,27]

1.4 Contributions

As seen in the previous section, finding exact and even approximate solutions to point pattern matching problems can be computationally quite intensive. Two common methods of point pattern matching are relevant to understanding our approaches: (1) geometric branch-and-bound search of the transformation space and (2) extracting transformations from point-to-point alignments. The first method has the advantage that it can provide arbitrarily good guarantees on the accuracy of the final match, and that it naturally uses *a priori* information to bound the search. The main problem with this method is that the nature of the search leads to rather high running times. The second method is very easy to implement, but it cannot generate results with better than a fixed constant error (depending on the transformation space[17]), and does not lend itself easily to exploiting *a priori* information in the search.

In this paper we propose two ways of reducing the computational burden of point pattern matching for image registration. Our first innovation is to consider computing

an approximation to the optimum transformation. The approximation factor can be specified by the user. We also propose a new algorithm for robust point pattern matching, called *bounded alignment*, which we feel combines the stronger points of the branch-and-bound and alignment methods. It is a Monte-Carlo algorithm, but the user can make the probability of failure arbitrarily small. The algorithm is based on the branch-and-bound search framework, but it uses point alignments to speed up the search. In particular, when the search reaches a point where a significant number of point-to-point correspondences can be inferred, it starts applying alignments to these pairs. Unlike the brute-force alignment method described above, the correspondences are known, so only a small number of alignments need be checked. We provide empirical evidence that with high probability, at least one of these alignments will allow the search algorithm to "zoom-in" rapidly on the neighborhood of the optimum transformation. This acceleration can lead to significantly faster overall search times, especially when the final Hausdorff distance is small relative to the distances between the feature points.

The rest of the paper is organized as follows. In Section 2 we describe the branch-and-bound approximation algorithm. In Section 3 we present the bounded alignment algorithm. Section 4 provides empirical results, and Section 5 contains concluding remarks.

2 The Branch-and-Bound Algorithm

Both of our registration algorithms are based on a geometric branch-and-bound framework. This framework was first developed by Huttenlocher and Rucklidge for performing approximate point pattern matching using the partial Hausdorff distance,[2,28,29] and was also used by Hagedoorn and Veltkamp [24] in their algorithm for performing approximate matching using the absolute difference similarity measure. Our algorithm was developed independently of that of Hagedoorn and Veltkamp, but its overall structure is quite similar. Because the bounded alignment algorithm is based on the branch-and-bound algorithm, we will also describe the branch-and-bound algorithm here.

Recall that we are given two points sets A and B and a space of transformations $\mathcal{T}$. We are also given a distance quantile q, $0 < q \leq 1$. The problem is to find $\tau \in \mathcal{T}$ that minimizes the partial Hausdorff distance $H_q(\tau(A), B)$. Since A and B will be fixed for the remainder of the discussion, let us define $sim_q(\tau)$ as the similarity measure of τ. (Note that the better the match, the smaller the similarity measure.) Let τ_{opt} denote this optimum transformation, and let sim_{opt} be the optimum similarity.

2.1 Transformation Space

There are a number of different ways to describe the space $\mathcal{T}$ of transformations. In our implementation we have assumed that $\mathcal{T}$ is a bounded set of affine transforma-

tions. Assuming that the point sets A and B are both in two-dimensional space, any affine transformation τ applied to a point $a \in A$ can be expressed as a linear transformation M followed by a translation t, i.e.

$$\tau(a) : Ma + t,$$

where a and t are two-element column vectors, and M is a 2×2 matrix. Since six parameters are needed to define the transformation, this naturally leads to a six-dimensional *transformation space*. The algorithm can be applied to any reasonable parameterization of this space.

There are somewhat more natural ways to describe affine transformations in terms of translation, rotation, scaling, and shearing. We have implemented two forms, one which uses the six coefficients defining M and t, and another which is used for rigid transformations (without reflection), in which we model a transformation as a point in 3-space by giving its rotation angle θ and the x- and y-components of the translation. The description given here is based on the most general case of six-dimensional space; the restriction to lower dimensional spaces is straightforward.

2.2 Approximation Error

There are three natural ways to define the approximation error. The approximation may be suboptimal by some relative error factor; it may be suboptimal becausey some absolute additive error; or it may be suboptimal because too small a quantile was used in the Hausdorff distance. Rather than define three separate approximation problems, we provide three nonnegative parameters which allow the user to control these tradeoffs:

- ϵ_r, the relative error bound,
- ϵ_a, the absolute error bound, and
- ϵ_q, the quantile error bound.

Define $q' = (1 - \epsilon_q)q$ to be the *weak quantile*. Note that since $q' \leq q$, we have $sim_{q'}(\tau) \leq sim_q(\tau)$, for any $\tau \in \mathcal{T}$. We say that a transformation τ is *approximately optimal* relative to these parameters if either

$$sim_{q'}(\tau) \;\leq\; (1 + \epsilon_r) sim_{opt} \qquad \text{or} \qquad sim_{q'}(\tau) \;\leq\; sim_{opt} + \epsilon_a$$

holds. We call these conditions *approximate correctness conditions*. Thus, the approximate solution is slightly less robust, in that it considers only the weak quantile rather than the true quantile, and it may exceed the optimum similarity by a relative error of ϵ_r or an absolute error of ϵ_a. Later we show that as long as $\epsilon_a > 0$ or $\epsilon_r sim_{opt} > 0$, termination is guaranteed. The branch-and-bound search of (2) and (24) implicitly provides ϵ_a but not the other two parameters.

2.3 Exact Algorithm

We begin by describing an exact version of the algorithm, and later in Section 2.5 we describe the changes needed to yield the approximation algorithm. As mentioned earlier, the algorithm is based on a geometric branch-and-bound search of transformation space. We construct a search tree, where each node of the tree is identified with the set of transformations contained in some axis-aligned hyperrectangle in the six-dimensional transformation space. These hyperrectangles are called *cells*. Each cell T is represented by a pair of transformations, (τ_{lo}, τ_{hi}), whose coordinates are the upper and lower bounds on the transformations of the cell. Any transformation whose coordinates lie between the corresponding coordinates of τ_{lo} and τ_{hi} lies in this cell.

The algorithm begins with an initial cell, which is assumed to contain the optimum transformation. This is supplied by the user, based on *a priori* knowledge of the nature of the transformation (e.g., bounds on the possible translation and rotation). A cell is either *active*, meaning that it is a candidate to provide the optimum transformation, or *killed*, meaning that it cannot provide the optimum solution. At any time the union of the active and killed cells forms a partition of the initial cell into subcells with disjoint interiors. The algorithm proceeds by selecting one of the active cells to be *processed*. After processing, a cell is either killed or is split into two disjoint subcells. Each of the subcells is made active. We will discuss termination conditions later.

Let us consider the algorithm in somewhat greater detail. For each cell T that we process, we are interested in the transformation of this cell that has the smallest similarity measure. We compute an upper bound $sim_{hi}(T)$ and a lower bound $sim_{lo}(T)$ on this smallest similarity (discussed below). For each upper bound, there will be a transformation that serves as a witness to this upper bound. The algorithm maintains the best similarity sim_{best} that it has encountered thus far in the search, and the associated transformation τ_{best}. To process a cell, we first compute its lower bound (see details below). If the lower bound exceeds sim_{best}, we kill this cell. Otherwise, we compute the upper bound, and if it is less than sim_{best}, we update the current best.

To compute the upper bound, we may sample any transformation from within the cell. In our implementation, this is done by simply taking the midpoint τ of the cell, and then computing $sim_q(\tau)$. In particular, this involves computing the image of each point of A under τ, and then computing the nearest neighbor in B for each image point. Since the cell's optimum similarity is smaller than (or equal to) any sampled point, this is an upper bound on the minimum similarity of the cell. Nearest neighbors are computed by storing the points of B in a kd-tree data structure, and applying known efficient search techniques.[30,31,32]

To compute the lower bound, we use a technique similar to that described in references (2) and (24). Given any cell $T \subset \mathcal{T}$, and given any point $a \in A$, consider the image of a under every $\tau \in T$. It is easy to compute a bounding rectangle for

this set. For example, if the coordinates of a are nonnegative, this is the rectangle defined by the corner points $\tau_{lo}(a)$ and $\tau_{hi}(a)$. (The general formula involves a case analysis on the signs of the coordinates of a.) We call this bounding rectangle the *uncertainty region* of a relative to T. This is essentially the same concept as the traced volumes introduced by Hagedoorn and Veltkamp.[24] In this way, each cell is associated with a collection of uncertainty regions, one for each point of A. (See Fig. 1 for an illustration of uncertainty regions. The transformed points of A are shown in white, each transformed by the midpoint of the transformation cell; the points of B are shown in black; and the uncertainty regions are shown as rectangles.)

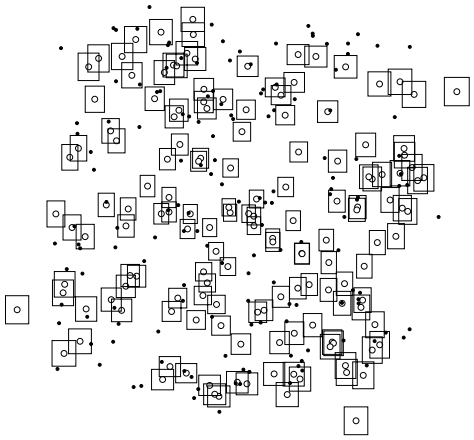


Figure 1: Uncertainty regions.

Define the distance between an uncertainty region and a point $b \in B$ to be the minimum distance between b and any part of the uncertainty region. If b lies inside the uncertainty region, then the distance is zero. To derive our lower bound for T, for each point $a \in A$, we compute the distance from the corresponding uncertainty region to its nearest neighbor in B. Observe that this distance is a lower bound on the distance from $\tau(a)$ to its nearest neighbor in B, for any $\tau \in T$. We then take the qth smallest such distance. Call this $sim_{lo}(T)$. Clearly this is a lower bound on $sim_q(\tau)$ for any $\tau \in T$, and hence is indeed a lower bound for the cell. Nearest neighbors to an uncertainty region can be computed by a straightforward generalization of the kd-tree-based nearest neighbor method described above.[33]

2.4 Cell Processing

Next we consider how cells are processed. We split each cell into two subcells (rather than $2^6 = 64$ subcells as in (2)). We choose the cell to be processed and the splitting dimension so as to reduce the size of the uncertainty region as much as possible. In particular, define the *size* of an uncertainty region to be its longest side. Define the *size* of a cell to be largest size among the associated uncertainty regions for each point

in A. (Note that cell size is *not* defined in terms of the size of the hyperrectangle in transformation space.) Our implementation computes a fast upper bound on this size, by first computing the bounding box for the points of A, and then computing the maximum size of the uncertainty region generated by any point in this bounding box. The active cells are stored in a priority queue, ordered by cell size. The largest cell, that is, the cell generating the largest uncertainty region, is chosen for processing at each stage.

If a cell survives processing, it is split along one of its sides into two smaller cells. Consider a cell T defined by a lower and upper bound range (τ_{lo}, τ_{hi}). To determine which side to split, we determine which of the six components of the transformation range contributes most to the size of the uncertainty region, and then we split the cell through its midpoint along this dimension. To make this precise, first consider the transformation $\Delta = \tau_{hi} - \tau_{lo}$, formed by taking the component-by-component differences of these transformations. In particular, Δ is described by a 2×2 matrix whose entries are Δ_{ij} for $i, j \in \{1, 2\}$ and a translation vector (t_1, t_2). Given a point $a = (a_1, a_2) \in A$, by the linearity of our representation, the size of the uncertainty region of a point a is just the longer side of the vector $\Delta(a)$. The x- and y-components of this vector are

$$\begin{aligned} x &= \Delta_{11}a_1 + \Delta_{12}a_2 + t_1, \\ y &= \Delta_{21}a_1 + \Delta_{22}a_2 + t_2. \end{aligned}$$

We define the component that contributes most to the size of a's uncertainty region to be the largest in absolute value among the six terms consisting of the $\Delta_{ij}a_j$'s and the t_i's. This is for a single point a. To define the component that contributes most to the size of an entire cell T, we take maximum component over all points $a \in A$. In our implementation, we compute a fast bound by taking the the maximum over all points lying in the bounding box of A. Because there are six dimensions that affect the size of each uncertainty region, and given the linearity of the transformation space, we have the following result:

Lemma 1

(i) *After six consecutive splits, the size of a cell (that is, the maximum side length of its largest uncertainty region) decreases by at least half.*

(ii) *If there are currently m active cells, then after at most $64m$ instances of cell processing, the size of the largest cell decreases by at least half.*

Proof: The first part follows from linearity and the fact that at each step we bisect the side of the cell that contributes the most to the size of the uncertainty region. Since each cell has six sides, with each six consecutive splits all six components must be decreased by at least one half; hence the longest side has decreased by at least half. To prove the second part, observe that six levels of splitting will generally result

in at most $2^6 = 64$ new cells. In the worst case, none of these cells is killed. Thus, if m cells are currently active, after $64m$ new cells have been generated, we must be at least six levels deeper in the search tree. From (i) it follows that the sizes of all cells, and of the largest cell in particular, decrease by at least half. □

2.5 Approximation Algorithm

Having completed the description of the exact algorithm, we now discuss how to modify it to obtain an approximation algorithm. First, we compute the upper bound $sim_{hi}(T)$ on the similarity of a cell using the weak quantile q', rather than the true quantile q. Thus we have $sim_{hi}(T) \geq sim_{q'}(\tau)$, for any $\tau \in T$. Second, the condition for killing a cell is not that the cell's lower bound be greater than the current best, but that its lower bound not be significantly lower than the current best. In particular, given a cell T with lower bound $sim_{lo}(T)$, it is killed if either

$$sim_{lo}(T) > \frac{sim_{best}}{1+\epsilon_r} \qquad \text{or} \qquad sim_{lo}(T) > sim_{best} - \epsilon_a$$

is met. The algorithm terminates when all cells have been killed or when $sim_{best} < \epsilon_a$.

Here is an overview of the approximation algorithm. The input consists of the point sets A and B, the Hausdorff quantile q, the approximation parameters ϵ_r, ϵ_a, and ϵ_q, and the initial cell T_0.

(1) Build a nearest neighbor data structure for the points of B. Initialize the priority queue to contain T_0. Set $sim_{best} = \infty$. Define the weak quantile q' to be $q(1-\epsilon_q)$. Repeat steps (2)–(5) until the priority queue is empty or until $sim_{best} \leq \epsilon_a$.

(2) Remove the largest cell T from the queue. Compute its lower and upper bounds. This involves the following steps:

   (a) Compute the uncertainty regions for every point $a \in A$ with respect to T.

   (b) For each uncertainty region, compute its nearest neighbor in B (where the distance from a point to a region is defined to be the shortest distance between the point and any part of the region).

   (c) Using any fast selection algorithm, compute the qth quantile among these distances. Call this $sim_{lo}(T)$. If $sim_{lo}(T) > sim_{best}/(1+\epsilon_r)$ or if $sim_{lo}(T) > sim_{best} - \epsilon_a$, kill this cell and return to step (2).

   (d) Otherwise, sample a transformation τ from this cell. Compute the image of each point of A under τ, and compute the nearest neighbors of these points with respect to B. Find the q'th smallest such distance. Call this $sim_{hi}(T)$.

(3) If $sim_{hi}(T) < sim_{best}$, update sim_{best} and let τ_{best} be the associated transformation.

(4) Split T into two smaller subcells T_1 and T_2, by splitting it along the dimension that contributes most to its uncertainty region size. Compute size bounds for T_1 and T_2.

(5) Enqueue T_1 and T_2 in the queue of active cells.

The final transformation is τ_{best} and its similarity is sim_{best}. The correctness of the algorithm is established by the following theorem.

Theorem 1 *If $\epsilon_a > 0$ or $\epsilon_r sim_{opt} > 0$, the approximation algorithm eventually terminates, and upon termination τ_{best} satisfies at least one of the following approximate correctness conditions:*

(i) $sim_{best} \leq (1 + \epsilon_r) sim_{opt}$,

(ii) $sim_{best} \leq sim_{opt} + \epsilon_a$.

Before presenting the proof, observe that ϵ_a, ϵ_r, and sim_{opt} are all nonnegative quantities. Thus, the only way to violate this condition is if the user sets $\epsilon_a = 0$, and either ϵ_r or sim_{opt} is equal to zero.

Proof: First we show that the algorithm eventually terminates. Suppose to the contrary that the algorithm does not terminate. Recall that the size of a cell is the longest side of all of the associated uncertainty regions. Since the initial cell is bounded, it follows from Lemma 1 that if the algorithm does not terminate, then after a sufficiently large number of cells have been processed, the maximum cell size falls below any given positive threshold. Let

$$\beta = \max\left(\epsilon_a, \frac{\epsilon_r}{1 + \epsilon_r} sim_{opt}\right).$$

From the hypotheses of the theorem, we have $\beta > 0$. We will show that when every active cell has size less than $\beta/\sqrt{2}$, every cell to be processed will be killed. Hence the algorithm must terminate.

Consider any cell T that is processed after this happens, and let τ denote the transformation that was sampled from the midpoint of T in computing $sim_{hi}(T)$. Let $\delta(a)$ denote the distance between a's uncertainty region and the nearest point in B. Let $\delta'(a)$ denote the distance between $\tau(a)$ and its nearest neighbor in B. The diameter of each uncertainty region is at most $\sqrt{2}$ times its size. Because $\tau(a)$ lies within a's uncertainty region, and the size of T is at most $\beta/\sqrt{2}$, it follows that

$$\delta(a) \leq \delta'(a) \leq \delta(a) + \beta.$$

Let δ be the qth quantile of $\delta(a)$ taken over all $a \in A$, and let δ' be the q'th quantile of $\delta'(a)$ for $a \in A$. Since the above inequality holds irrespective of a, and since $q' \leq q$, we have $\delta' \leq \delta + \beta$. By construction, $sim_{lo}(T) = \delta$ and $sim_{hi}(T) = \delta'$, from which we have

$$sim_{hi}(T) - sim_{lo}(T) < \beta.$$

Since τ was considered as a candidate for the best similarity, we have $sim_{best} \leq sim_{hi}(T)$. Combining this we have the following relationship between sim_{best} and $sim_{lo}(T)$.

$$\begin{aligned} sim_{best} - sim_{lo}(T) &\leq sim_{hi}(T) - sim_{lo}(T) \\ &< \max\left(\epsilon_a, \frac{\epsilon_r}{1+\epsilon_r} sim_{opt}\right) \\ &\leq \max\left(\epsilon_a, \frac{\epsilon_r}{1+\epsilon_r} sim_{best}\right). \end{aligned}$$

There are two cases. If the maximum is achieved by ϵ_a, we have

$$sim_{lo}(T) > sim_{best} - \epsilon_a.$$

On the other hand, if the maximum is achieved by the other term, we have

$$sim_{lo}(T) > \frac{sim_{best}}{1+\epsilon_r}.$$

In either case, T satisfies one of the two conditions needed to kill it. Thus, no cell can survive beyond this point, and the algorithm will terminate.

To establish the correctness of the algorithm upon termination, first recall that $sim_{best} = sim_{q'}(\tau_{best})$. We show that at least one of the approximation conditions stated in the theorem holds in either case. First, if the algorithm was terminated because $sim_{best} \leq \epsilon_a$, then since the optimum similarity is nonnegative, we have $sim_{best} \leq sim_{opt} + \epsilon_a$, implying that (ii) holds.

Suppose, on the other hand, the algorithm terminates because all the cells have been killed, consider the moment at which the cell T containing the optimum transformation, τ_{opt}, is killed. The lower bound for this cell satisfies $sim_{lo}(T) \leq sim_q(\tau_{opt})$. It was killed for one of two reasons. First, if $sim_{lo}(T) > sim_{best}/(1+\epsilon_r)$ we have

$$sim_{best} \;<\; (1+\epsilon_r) sim_{lo}(T) \;\leq\; (1+\epsilon_r) sim_q(\tau_{opt}) \;=\; (1+\epsilon_r) sim_{opt},$$

implying (i). On the other hand, if $sim_{lo}(T) > sim_{best} - \epsilon_a$, then we have

$$sim_{best} \;<\; sim_{lo}(T) + \epsilon_a \;\leq\; sim_q(\tau_{opt}) + \epsilon_a \;=\; sim_{opt} + \epsilon_a,$$

implying (ii). Hence when the algorithm terminates, all the cells—including the optimal cell—have been killed, implying that the reported transformation is approximately correct. □

3 Bounded Alignment

The branch-and-bound algorithm has many nice features, but its main drawback is its relatively high running time. This occurs especially when high accuracy is required and the optimum similarity is very good. The reason for this is that the algorithm must decompose a (typically 2–6 dimensional) search space into very small cells in order for the uncertainty regions to be sufficiently small so that the lower bounds become large enough to kill unpromising cells.

For this reason, we introduce an additional process called *bounded alignment* to help accelerate the search. Throughout this section we assume that the transformation space is the 6-dimensional space of affine transformations. Generalizations to other transformation spaces are straightforward.

3.1 Intuition

Before discussing what bounded alignment is, let us consider why the branch-and-bound approximation algorithm may run slowly, and how it might be speeded up. It is argued in the proof of Theorem 1 that the algorithm terminates when the cell sizes decrease to around $\max(\epsilon_a, \epsilon_r sim_{opt})$. If this quantity is small compared to the initial uncertainty bounds, then the algorithm may require a great deal of splitting before achieving this size. Of course, not all cells need to be split to this small size, but at the very least, observe that if the uncertainty region of a cell contains even a single point of B, then this region contributes a distance of zero to the set of distances used in computing the lower bound for this cell. Thus to have a nonzero lower bound, a significant fraction (at least q) of uncertainty regions must contain no points of B. As cell sizes shrink, before we arrive at a stage in which many cells have nonempty uncertainty regions, we will first reach a stage where many cells have uncertainty regions that contain at most a single point. When this happens, we can determine that there is a unique corresponding matching point for each of these uncertainty regions. It is at this point that alignment can be applied.

To illustrate the idea, consider the example shown in Fig. 2 below. (Transformed points of A are indicated in white and points of B in black.) To simplify the discussion, consider the case of translation alone. Let $k = 4$ ($q = 0.5$), and suppose that there is a match with $H_4 = 0$. There are 8 points in A, and there is a translation for which at least 4 of them (or actually 5) match exactly.

On the left we show an example where the cell contains the optimum transformation. Suppose that the search has progressed to a stage where most of the uncertainty regions associated with the points of A contain at most one point of B. Consider the points of A that have exactly one point of B in their uncertainty regions. We sample one such point at random and compute the unique transformation that maps this point to the corresponding point of B (indicated by an arrow in the figure). There is a good probability (at least 1/2) that the point of A that we picked is an inlier,

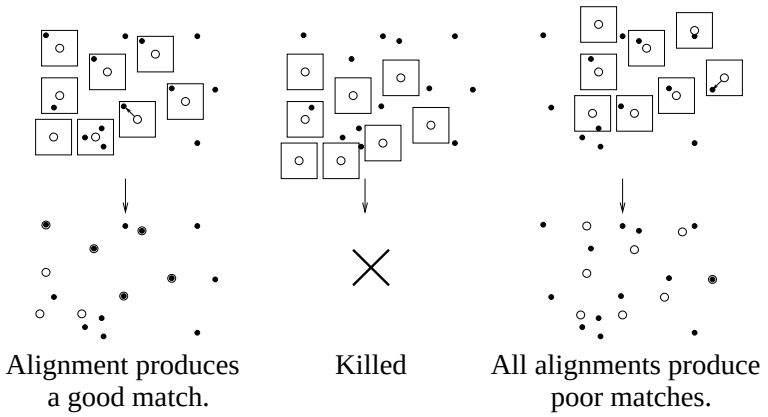


Figure 2: Alignment applied to three cells in transformation space.

implying that the point of B in its uncertainty region is its matching point. The resulting transformation is the desired optimum (or in the case of errors, should be close to the optimum).

On the other hand, if a cell does not contain the optimum (or any transformation close to the optimum), then consider the following two cases. In the middle figure we illustrate a case in which most of the uncertainty regions contain no matching point of B. In this easy case, even the branch-and-bound algorithm will report that no match exists and the cell will be killed. However, consider the case shown on the right. Here a majority of the uncertainty regions contain a point of B. As a result, the lower bound associated with this cell becomes zero, and it is not possible at this stage for the branch-and-bound algorithm to distinguish this cell from the one containing the optimum. However, if we apply the same sampling that we did to the cell on the left (one such sampling is indicated by an arrow in the figure), then we should expect to find that every sample returns a rather poor similarity measure. After taking a number of samples and witnessing repeatedly poor similarities, we may regard this as evidence that the cell in question does not contain the optimum and therefore we kill it. This is the intuitive basis for bounded alignment, i.e., of using the bounds provided by the branch-and-bound framework to guide the alignment process (rather than simply enumerating all possible alignments as in (17), and in turn using the results of these alignments to guide the branch-and-bound search.

3.2 Alignment

Let us think of the points of A as being partitioned into two classes: *inliers*, i.e., points that are mapped by the optimum transformation to lie within distance sim_{opt} of their nearest neighbors in B, and *outliers*, i.e., all the remaining points of A. By definition of sim_{opt} as a q-quantile Hausdorff distance, at least fraction q of the points of A are inliers. (Normally, the concept of an inlier is based on the underlying point

distribution, but this functional definition is sufficient for our purposes.) To simplify the explanation, suppose for a moment that the inliers are completely free of error, that is, the q-quantile Hausdorff distance is zero for the optimum transformation, τ_{opt}. If we were able to find three inliers of A and match them against the corresponding elements of B, and further, if these three points of A were affinely independent (i.e., not collinear), then these three point correspondences would uniquely determine the affine transformation that achieves a Hausdorff distance of zero. The process whereby triples from A are matched against prospective corresponding triples from B in order to determine a candidate transformation is called *alignment*.

Of course, in a noisy environment it is unreasonable to assume that points can be matched exactly. Furthermore, the numerical extraction of the affine transformation may introduce errors. To generalize this to noisy environments, suppose that for each inlier $a \in A$ there is a point of B that lies within some small distance η from its optimum image point, $\tau_{opt}(a)$. We assume that an upper bound on η, called the *noise bound*, is provided to the search algorithm. For image registration applications, such an estimate could be based on knowledge of the accuracy of the feature extraction process. In general, this parameter could be determined through binary search. The final Hausdorff distance will be no larger than η. If η is small relative to the size of the image, and if the three points are geometrically well distributed, we can expect that the match that results from alignment should be fairly close to the optimum. Intuitively, a geometrically well-distributed triple should define a large, "fat" triangle among the points of A. To make this more formal, we present the following definition.

Definition: Given a set of points A in the plane and $\alpha \geq 1$, a triple $\{a_1, a_2, a_3\} \subseteq A$ is *well-distributed* relative to α if every point $a \in A$ can be expressed as an affine combination $a = \alpha_1 a_1 + \alpha_2 a_2 + \alpha_3 a_3$ such that $|\alpha_i| \leq \alpha$.

Given this definition, we have the following result: (See (17) for similar bounds for other subspaces of affine transformations.)

Lemma 2 *Given a planar point set A, a well-distributed triple $\{a_1, a_2, a_3\}$ relative to some $\alpha > 1$, and an affine transformation τ that maps each of the points a_i to a point that lies within distance η of a_i, then for any point $a \in A$, the distance between $\tau(a)$ and a is at most $3\alpha\eta$.*

Proof: Consider any point $a \in A$. By the definition of well-distributed, we can write a as $\sum_i \alpha_i a_i$, where $|\alpha_i| \leq \alpha$. By linearity we have $\tau(a) - a = \sum_i \alpha_i(\tau(a_i) - a_i)$. By the triangle inequality we have

$$|\tau(a) - a| \leq \sum_{i=1}^{3} \alpha_i |\tau(a_i) - a_i| \leq \eta \sum_{i=1}^{3} \alpha_i \leq 3\alpha\eta.$$

□

This lemma implies that if alignment is applied to a well-distributed triple of points and if the images of the aligned points are perturbed by a small amount η, then the perturbation induced by the resulting transformation will be proportional to η.

In general, the number of subsets that have to be tested may be quite large. However, the branch-and-bound search provides a particularly simple way of identifying the most promising correspondences to try. In particular, as the search proceeds and cells are split, the search eventually arrives at a stage in which a significant fraction of the uncertainty regions will contain at most a single point of B. An uncertainty region is said to be *alignable* if there is at most one point of B in the region, or if the region is empty and there is at least one point of B within distance η of the region. If the current cell has a significant fraction of alignable uncertainty regions, we say that this cell is *eligible for alignment*. (See Fig. 3. The transformed points of A are shown in white, and points of B are shown in black. Alignable uncertainty regions are shown with solid lines, and unalignable uncertainty regions are shaded. For each alignable uncertainty region, a line segment joins the associated (i.e., transformed) point of A to its nearest neighbor in B.)

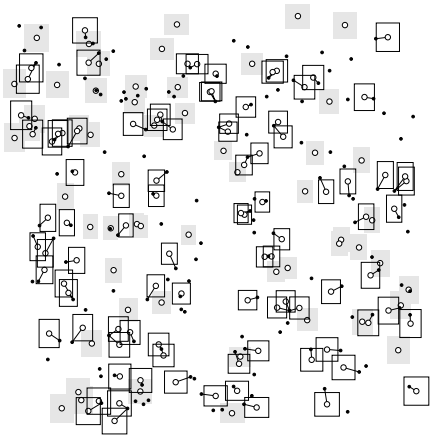


Figure 3: Alignable uncertainty regions.

Consider the cell containing the optimum transformation. If it is eligible for alignment, then for each of its alignable uncertainty regions, there is a single obvious candidate to which the corresponding point of A should be mapped. That is, if the uncertainty region is nonempty, a point of B lies within the uncertainty region; otherwise, it is the nearest neighbor of the uncertainty region. Furthermore, if $a \in A$ is an inlier, the nearest neighbor in B should lie within distance η of a, and hence within η of the uncertainty region.

To apply alignment, we need to make the assumption that the uncertainty regions of outliers are not more likely to be alignable than those of inliers. (If this assumption is violated, we should at least have a bound on the difference between these

probabilities.) Under the stated assumption, if we randomly sample three alignable uncertainty regions of A (without replacement), then with probability roughly q^3, all three points will be inliers. Thus, the three respective nearest neighbors of B will be within distance η from each of these points. Moreover, it follows from Lemma 2 that if the three sampled A are well-distributed, the alignment transformation (that results from considering the correspondence between the sampled triple from A and its counterpart from B) will be a reasonable approximation to the optimum, in the sense that the Hausdorff distance should not exceed the optimum Hausdorff distance by an amount greater than a constant c times η (as determined by Lemma 2).

Thus, under the above assumptions, and depending on q, if we take a sufficiently large number N of random samples, the probability of generating an alignment transformation whose distance from the optimum is more than $c\eta$ is at most $(1-q^3)^N$. (To see this, observe that $(1-q^3)$ is the probability that all three sampled points of A are outliers, and for failure this event must be repeated N times.) To reduce the chances of failing to find at least one good triple to some small failure probability p_f, the number of well-distributed samples should satisfy

$$N(q, p_f) \geq \frac{\ln p_f}{\ln(1-q^3)}.$$

For example, to reduce the probability of failure below 0.01, and for $q = 0.5$, roughly 35 sampled triples are sufficient. Note that this constant, while large, is independent of the numbers of feature points in the images.

If, on the other hand, we find that $N(q, p_f)$ well-distributed sampled triples all yield similarities that exceed the current best (by at least $c\eta$), we conclude that the cell in question does not contain the optimum transformation, and discard it from further consideration. This is the basis of alignment. Observe that it is a Monte Carlo process, in the sense that with probability p_f we may erroneously kill the cell containing the optimum transformation.

3.3 Processing Alignments

Let us now describe the alignment process in greater detail. First, we need to make the following assumption upon which alignment relies.

Alignment Assumption: For a cell containing the optimum transformation, the probability that an uncertainty region associated with this cell is alignable, and the probability that an alignable triple is well-distributed are not greater for outliers than for inliers.

As mentioned above, this assumption is not necessarily realistic. But if it is possible to estimate the difference in the conditional probabilities (that an uncertainty region is alignable) for inliers and outliers, and if these probabilities are not

significantly different, then the number of samples derived above can be modified accordingly. If this assumption is not expected to hold (e.g., when inlying feature points are highly clustered), then bounded alignment may not perform well.

We can incorporate alignment into the existing branch-and-bound approach as follows. The additional processing is added just after the upper and lower bounds have been computed for a cell. We supply the algorithm with three additional pieces of information: an expected perturbation η between inliers and their corresponding points in the final match, a sampling quantile q_s, and a minimum sample size N_s (the latter two are explained below in more detail). First, we augment the procedure that computes the distance from each uncertainty region to its nearest neighbor to compute the number of points of B that lie within the uncertainty region. This can be done by modifying the kd-tree nearest neighbor algorithms.[33] If this number is at most one, and the nearest neighbor is within distance η of the uncertainty region, then we flag this region as being alignable.

After processing all the uncertainty regions, if the fraction of alignable uncertainty regions exceeds q_s, we declare that the cell itself is eligible for alignment. If so, let A' denote the subset of A such that for each $a \in A'$, there exists at least one point $b \in B$ that either lies inside or within distance η of a's uncertainty region. If this cell contains the optimum, then these are the only candidates to be inliers. Sample (without replacement) N_s triples of points from A', such that each triple is well-distributed (relative to some given constant). This is done by repeatedly sampling triples, until one that is well-distributed is found. For each sampled triple we compute an alignment transformation as follows. For each point a of the triple, sample a random point $b \in B$ lying inside or within distance η of a's uncertainty region. Generate the transformation that aligns the triple from A' with these corresponding points of B, and compute its similarity. If the similarity of this transformation is better than the current best similarity sim_{best}, make it the new best. If the similarity obtained for all of the N_s transformations exceeds the current best by an additive amount $c\eta$, kill this cell. Otherwise, the cell remains active, and the normal processing described in the previous section resumes.

Testing and updating the similarity for each sampled triple as described above can be further exploited as follows. As already stated, if we find that any sampled transformation is better than the current best (and the transformation satisfies the additional allowability conditions), we take this transformation to be the current best. Thus alignment not only helps in improving lower bounds, but serves to generate better upper bounds.

Here is a summary of the steps used for the bounded alignment algorithm. (These steps are added after step (2d) in the previous description.) The algorithm is given an expected inlier perturbation η, a sampling quantile q_s, and a minimum sample size N_s.

(e) For each $a \in A$, count the number of points of B that lie within a's uncertainty region. If this at most one, and the nearest neighbor is within distance η of the

uncertainty region, flag this region as alignable.

(f) If the fraction of alignable uncertainty regions is less than q_s, return to step (2). Otherwise, let A' denote the subset of A such that for each $a \in A'$, there exists at least one point $b \in B$ that either lies inside or within distance η of a's uncertainty region. Repeat the following N_s times:

(i) Sample (without replacement) triples of points of A', until a triple that is geometrically well-distributed is found.

(ii) Compute the transformation that aligns each point in the triple with a random point of B in its associated uncertainty region. Compute the similarity of this transformation.

(iii) If the similarity of this transformation is better than the current best similarity sim_{best}, make it the new best.

If the similarity obtained for all of the N_s transformations exceeds the current best by an additive amount of η, kill this cell.

Our implementation differs slightly from this description. We do not check that the points are well-distributed. Instead, we apply a somewhat more precise test. We generate the alignment transformation and then test whether this transformation lies within or is sufficiently close to the current cell. The definition of "sufficiently close" is that the translational component should be within some fixed constant factor of η, and the entries of the matrix M should be within a constant factor of the ratio of η to the diameter of the point set A. Our reasoning is that points do not need to be well-distributed, as long as they generate a transformation that is reasonably close to the cell. Because of its Monte Carlo nature, the danger in alignment is that the cell containing the optimum may be killed accidentally. If the current cell contains the optimum transformation, we expect that the transformation generated by a triple of inliers should either lie inside or close to the cell. If not, we conclude that the triple is unacceptable. Because it is easy to test this condition once the transformation has been generated, we generate triples until this condition is satisfied. If we fail after a repeated number of tries (10 in the current implementation), we treat this sample as a failure.

3.4 Analysis

Under what circumstances does alignment offer an advantage over pure branch-and-bound? At this point we do not have a theoretical analysis or even a complete empirical analysis explaining when this happens. However, based on our empirical experiences with the algorithm on uniformly distributed point sets, in many instances alignment can offer significant improvements in running time over pure branch-and-bound search (see Section 4). In particular, we have observed that alignment seems

to offer the greatest advantage when the deviation η in the location of feature points is relatively small with respect to the expected distance δ from a random point and its nearest neighbor in B.

Here is some informal justification for this empirical observation. Assume that the points of B are uniformly distributed, and for simplicity suppose that the center of each uncertainty region is sufficiently random that the expected distance to the nearest point of B is δ. When the sizes of the uncertainty regions decrease to roughly δ, alignment becomes possible, because many uncertainty regions of the cell containing the optimum transformation should contain at most one point of B. Also note that at this point, the current best similarity is expected to be proportional to δ. This is because we cannot expect to locate the optimum much more precisely than the cell size (unless we happen to be very lucky in sampling a transformation). However, by the reasoning earlier in this section, once we reach this stage, and alignment can be applied to the optimum cell, we expect to discover a transformation whose similarity is not greater than $sim_{opt} + c\eta$. Recall that $sim_{opt} \leq \eta$, and so sim_{best} is now proportional to the optimum. Thus if the ratio δ/η is large, the alignment algorithm has a big advantage, as the tighter upper bound makes it possible to kill more unpromising cells.

Also consider the conditions needed to kill a nonoptimal cell. Assuming, as before, that sim_{best} is proportional to δ at this stage of the search, then (considering just the relative error) a cell T cannot be killed until

$$sim_{lo}(T) > \frac{sim_{best}}{1+\epsilon_r} \geq \frac{\delta}{1+\epsilon_r}.$$

The distance from an uncertainty region to its nearest neighbor in B is expected to be at most δ minus the radius of the uncertainty region. It follows from simple algebra that if the cell size is larger than $2\epsilon_r\delta$, this condition is not likely to be satisfied by most of the cell's uncertainty regions. But alignment already applies when the cell size is roughly δ. Thus the pure branch-and-bound approach must continue to split the cell until its size is reduced by an amount proportional to $2\epsilon_r$ before it can be killed. By Lemma 1, this may require $6\log_2(1/2\epsilon_r)$ additional splits, and could result in the generation of $(1/2\epsilon_r)^6$ additional cells. For even moderately small values of ϵ_r, this value may be significantly larger than the N_s samples generated by alignment; hence alignment may offer a significant improvement over pure branch-and-bound.

4 Empirical Study

We have implemented search algorithms in C++ both for standard branch-and-bound search and for as branch-and-bound with bounded alignment search. We have also implemented a driver program to run a series of tests, and to gather statistics about the performance of the two algorithms. Nearest neighbor and range queries were performed using kd-trees as generated by the `ANN` approximate nearest neighbor

library.[34] The driver program can either read input sets from a file or generate inputs randomly from a number of distributions.

We ran experiments on both synthetically generated data sets and real satellite images to test the general behavior of our algorithms, and specifically to study the performance of branch-and-bound with and without bounded alignment.

4.1 Synthetic Experiments

For the synthetic experiments, the point sets were generated as follows. First the user specifies the number of points in the sets A and B, bounding rectangles for A and B, the desired fraction of inliers of A, and the standard deviation σ of a random perturbation to be applied to each image point. An affine transformation τ is generated at random from a collection of bounds on the x and y translation, rotation, x and y scale, and shearing. The desired point sets are generated as follows. To generate the inliers, a point a is generated uniformly at random from A's bounding rectangle. For each such point, $\tau(a)$ is computed, and a Gaussian error with mean 0 and standard deviation σ is added. The point is accepted if it lies within the bounding rectangle of B, and rejected otherwise. This is repeated until the desired number of accepted inliers is generated. The remaining points of the sets A and B are distributed uniformly in each of the bounding rectangles. The given transformation τ is called the *target transformation*. In general, it may not be the optimum transformation (unless $\sigma = 0$), but it should be close enough to the optimum that we may use it for validating the results of the experiments.

The user supplies the desired quantile q for the Hausdorff distance, the error factors ϵ_r, ϵ_a, and ϵ_q, and the bounds on the initial cell. If bounded alignment is being applied, the user also supplies q_s, the fraction of alignable uncertainty regions needed to invoke alignment for a cell; N_s, the number of well-distributed triples to sample from the alignable regions; and η, the noise bound. The program can be run using either the standard branch-and-bound algorithm or the bounded alignment variant. It is also possible to select from two different parameterizations of transformation space. The first is a full six-dimensional space (although lower-dimensional searches are possible by allowing some of the parameter ranges to degenerate to 0). The other allows searches of three-dimensional space of rigid transformations by specifying a range of angles and ranges of x- and y-translations.

The program measures a number of statistics on its own performance, including CPU seconds, number of cells expanded, and number of times the Hausdorff-based similarity was computed. Statistics are also gathered on a level-by-level basis for the search tree.

We conjecture that bounded alignment should perform best when the Hausdorff distance is small relative to the nearest neighbor distance, since in this case the best alignments are generated. To test this, we generated points with varying degrees of perturbation error, keeping the number of points and the number of outliers fixed.

The perturbation error σ ranged from 0.1 to 5. The sets A consisted of 300 points, 180 of which were inliers, in a square $[-400, 400]^2$. We applied to these points a random rotation in the interval $[40°, 45°]$, and a random translation in $[-10, 10]^2$. The points for B were generated in a square $[-500, 500]^2$. (Different bounding squares were used to minimize boundary effects.) We fixed $\epsilon_q = \epsilon_r = 0.2$, and computed the median Hausdorff distance. The absolute error ϵ_a and the noise bound η were both set equal to σ, since this is a lower bound on the expected Hausdorff error.

The search was performed over rigid transformations. The initial search cell allowed 10° of variation in rotation and 40 units of translation in each of x and y. The initial cell contained the target transformation, but was randomly perturbed so that the exact location of the target transformation within the cell was unknown to the algorithm. Both branch-and-bound search and bounded alignment were invoked at least 20 times for each value of σ. Experiments were compiled using `g++ -O3`, and run on a Sun SPARC 5, running SunOS 5.5.

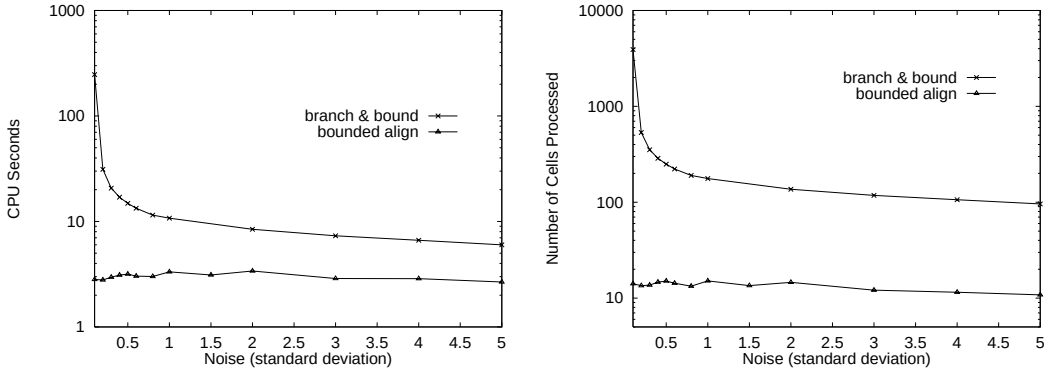


Figure 4: Execution time and number of cells processed for pure branch-and-bound and bounded alignment as a function of the perturbation standard deviation σ.

The average results are shown in Fig. 4. Observe that bounded alignment offered a significant advantage over standard branch-and-bound both in terms of CPU time and the number of cells processed when the perturbation error was small, i.e., when the final Hausdorff distance was small (typically $\sqrt{2}\sigma$). When the Hausdorff distance was large (implying that there was no obvious strong match), both algorithms performed quite efficiently. In other words, bounded alignment performed well in near exact match situations where standard alignment would do well, and also in poor match situations where branch-and-bound would do well.

We also compared the similarities of the resulting transformation with the target transformation. Out of over 400 experiments, in over half of the cases the relative error was less than 2% (while 20% relative error was tolerated), and in almost 80% of the cases the relative error was less than 10%.

To get a closer insight into the behavior of both algorithms, we computed the number of cells processed and the average best similarity for each level of the search

tree for the case $\sigma = 0.2$, averaged over 40 instances. The results are shown in Fig. 5. From the plot on the left, observe that bounded alignment tends to converge towards the optimum similarity much faster than branch-and-bound search. The fact that it has a better bound on the optimum similarity allows it to do a better job of pruning unpromising cells from the search. On the right we show the number of cells processed at each level of the search. In both cases there is an initial phase where cells are not being killed, and their numbers increase exponentially. At some point, the algorithm reaches a stage where the uncertainty is small enough that it is possible to kill cells. When this happens, there is a sudden decrease in the number of cells per level. As predicted in the analysis of the previous section, bounded alignment reaches this stage earlier than branch-and-bound. After this, bounded alignment seems to keep the number of active cells very low. In the branch-and-bound search, however, the number of cells per level starts increasing. Although we have no theoretical explanation for this phenomenon, we believe that it would be an interesting topic for further study.

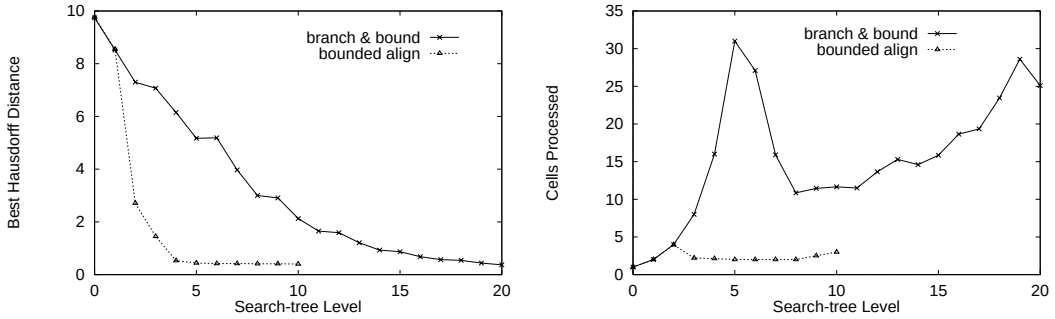


Figure 5: Best similarity and number of cells processed per level of the search tree for $\sigma = 0.2$.

4.2 Experiments on Satellite Imagery

For our second experiment we chose a number of remotely sensed data sets. Specifically, we experimented with three Landsat/TM scenes (over the Pacific northwest (Pac NW), Washington, DC (DC), and Haifa, Israel), an AVHRR scene (over South Africa (SA)), and a GOES scene (over Baja California). In each case, either two images (taken at different times) were provided, or a single image was given and a counterpart image generated by applying some transformation to the original one. Feature maps were generated by selecting the top 5 (or 10)% of the extracted features, e.g., edge magnitude using standard edge detection, wavelet coefficients using a wavelet decomposition,[35] etc.

For each data set, we ran both branch-and-bound (B&B) and bounded alignment (B. A.) on five different seeds. (All runs were done on an ULTRA-SPARC.) The median values (for each parameter of interest) are given in Table 1 as representative

results of the experiments. Most of the parameters of the algorithms, other than those specified by the user based on a priori knowledge, e.g., the initial cell, the Hausdorff distance quantile, etc. were kept essentially constant. Specifically, we ran the algorithms with the following parameter setting: $\epsilon_r = 0.1$, $\epsilon_a = 0.2$–0.4, $\epsilon_q = 0.2$, and $\eta = 0.4$–0.6. (Assuming feature extraction accuracy on the order of a pixel and a similar expected Hausdorff distance, the choice for η seems reasonable.) Also, we picked $N_s = 10$–20, and q_s such that $q_s|A|$ was on the order of 100 points.

	Alg	Cells	Sec	Angle	t_x	t_y	Sim
Pac NW	B&B	399	32	18.9	0.34	-0.49	0.42
Pac NW	B. A.	7	3	18.8	0.10	-0.54	0.47
DC	B&B	385	13	-0.08	32.1	32.0	0.30
DC	B. A.	35	4.8	0.14	32.5	31.7	0.60
Haifa	B&B	1103	55	-0.06	13.6	-1.02	0.59
Haifa	B. A.	25	7.4	-0.23	14.1	-1.13	0.73
SA	B&B	2037	81	-0.005	4.1	-5.8	0.18
SA	B. A.	29	3.9	0.0	5.0	-6.0	0.00
Baja	B&B	1164	17	-0.16	9.9	-10.0	0.17
Baja	B. A.	11	0.6	0.0	10.0	-10.0	0.00

Table 1: Results obtained using to branch-and-bound and bounded alignment on real data sets.

Figures 6–10 display the data and the results for the five experiments. In each case, (a) and (b) show, respectively, the first image associated with set A and the associated feature points; (c) and (d) show, respectively, the image associated with set B and its feature map; and (e) displays a final overlaying of feature points using a typical transformation computed by the bounded alignment algorithm.

The first TM data set consisted of a 128×128 gray-scale image over the Pacific northwest. Its counterpart was artificially generated by applying a -18° rotation. The features were extracted through wavelet decomposition and their map sizes were $|A| = 1756$ and $|B| = 1845$. The algorithms were run with an initial search cell having an angular range of 2° and a translational range of 5 pixels (in both the x and y directions). The results clearly indicate that bounded alignment outperformed branch-and-bound by a significant factor, while producing roughly the same similarity. The target similarity at exactly 18° is 0.81, worse than either of the similarities derived from the algorithm.

The data set of DC consisted of a 128×128 TM subimage, whose counterpart was generated by applying a translation of $(32.5, 32.5)$. (The translation procedure performed some averaging, resulting in slight blurring of the image.) The features were extracted by taking the top 5% of the edge pixels to yield feature maps of size $|A| = 763$ and $|B| = 766$. The initial search cell allowed 10° of rotation and 5

pixels of translation. The target similarity was 0.71, again worse than either reported similarity. Note, however, that although bounded alignment outperformed branch-and-bound as far as number of cells visited/running times are concerned, it often produced matches that were in excess of the allowable error bounds (in this case $\epsilon_r = 0.1$ and $\epsilon_a = 0.2$). This can most likely be attributed to a failure in the Monte Carlo algorithm. Recall from Section 3 that the bounded alignment algorithm assumes that there is equal probability that inliers and outliers are eligible for alignment. In the DC data set the inliers were more tightly clustered than the outliers, and this proximity reduced the likelihood of the inliers being eligible for alignment. This violation of the alignment assumption could be overcome by increasing the parameters q_s and/or N_s, at the expense of increasing running time. We feel that such problematic data sets imply that further work can be done on improving the generality of bounded alignment, and overcoming the restrictions of the alignment assumption.

The Haifa data set consisted of a pair of TM images taken on two different occasions. Feature extraction based on hierarchical wavelet decomposition was used to yield feature maps of size $|A| = 1120$ and $|B| = 1020$. The initial cell was set to an angular range of 5° and a translational range of 5 pixels in both directions, with a target similarity of 0.5. Our representative results indicate that bounded alignment ran about 8 times faster than branch-and-bound while reporting comparable similarities. However, this may not adequately reflect the situation. Examining individual runs more carefully shows that bounded alignment may take longer to run. As it turns out, the features extracted for the Haifa data set do not contain many "strong" inliers, i.e., inliers strongly matching their counterpart features. Put differently, η is relatively large in this case, which implies, according to the previous subsection, that bounded alignment may not necessarily outperform branch-and-bound. Furthermore, even the strong inliers seem to be highly clustered. As in the DC example, this suggests that for bounded alignment to overcome the violation of the alignment assumption so as to return more accurate results (Hausdorff distance-wise), the parameters q_s and/or N_s may need to be increased.

The South Africa data set consisted of a pair of AVHRR subimages taken at two different times. These subimages were cropped from the original data, so as to avoid undesired boundary effects. Using hierarchical wavelet decomposition and picking (less than) the top 5% of the features, we obtained feature maps of size $|A| = 872$ and $|B| = 927$. The algorithms were run with a target similarity of 1.0 and an initial cell of angular range 10° and translational range 5 pixels. Likewise, we generated feature maps for the GOES subimages of Baja with $|A| = 326$ and $|B| = 503$, and ran the algorithms with a similar initial cell. The target transformation in this case, $(0.0, 10.0, -10.0)$, corresponds to a target similarity of 0.0. (This may be because of the preliminary geo-registration that these images had undergone.) In both cases, bounded alignment outperformed branch-and-bound significantly, converging on a perfect match with a similarity of 0.0. (Note that branch-and-bound terminates the search once it reports a similarity smaller than than the allowed absolute error metric.)

In summary, it can be seen that in all instances where the alignment assumption holds, bounded alignment produces results comparable to or better than those of branch-and-bound, and the running times in terms of both of CPU seconds and number of cells visited, are smaller.

5 Conclusions

We have presented two algorithms for registering images in a robust manner through the use of feature point pattern matching. Both algorithms allow the user to improve running times by specifying approximate bounds on the relative, absolute, or quantile errors. The first algorithm is based on branch-and-bound search. It is simple and safe, but is relatively slow, especially when high accuracy is desired. The second algorithm, called bounded alignment, is based on combining branch-and-bound with computing point alignments to accelerate the search. It seems to be much faster than the branch-and-bound algorithm in many cases, but it is a Monte Carlo algorithm, and hence may fail with some small probability. We provide empirical evidence that by allowing approximation in the branch-and-bound, and through the use of bounded alignment, it is possible to achieve registrations of high accuracy with reasonably small running times, even with a relatively large number of feature points.

Acknowledgements

We would like to thank William J. Campbell and Robert F. Cromp of the Applied Information Science Branch, Code 935, NASA/GSFC, for their ongoing support and for sustaining the activities of the image registration group at NASA/GSFC. Also, we are grateful to Azriel Rosenfeld for his editorial comments and suggestions.

Bibliography

1. D. P. Huttenlocher and W. J. Rucklidge. A multi-resolution technique for comparing images using the Hausdorff distance. Technical Report 1321, Dept. of Computer Science, Cornell University, Ithaca, NY, 1992.
2. D. P. Huttenlocher and W. J. Rucklidge. A multi-resolution technique for comparing images using the Hausdorff distance. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 705–706, New York, June 1993.
3. L. G. Brown. A survey of image registration techniques. *ACM Computing Surveys*, 24:325–376, 1992.
4. J. Ton and A. K. Jain. Registering Landsat images by point matching. *IEEE Transactions on Geoscience and Remote Sensing*, 27:642–651, 1989.

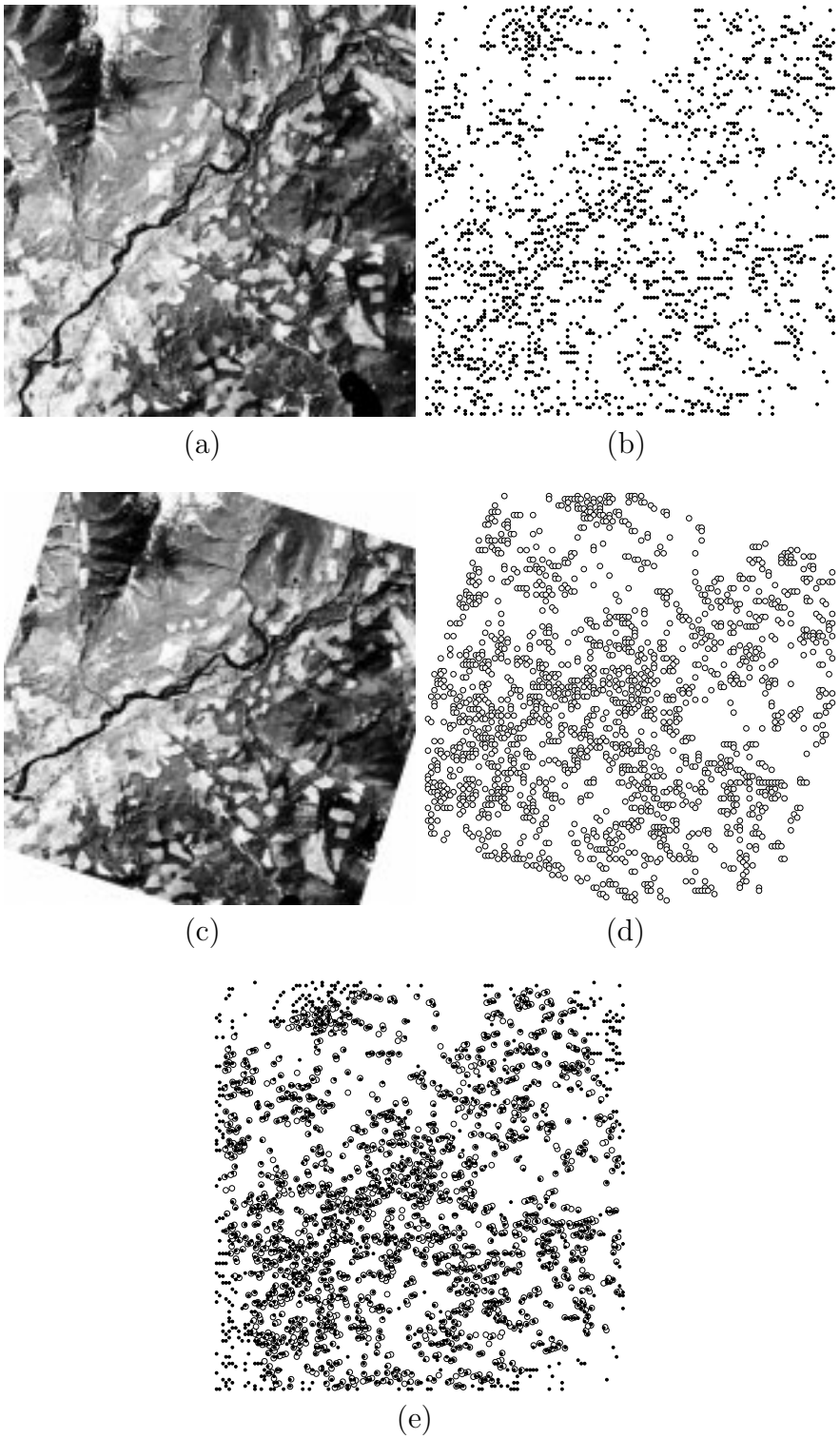


Figure 6: Matching two Landsat/TM images of the Pacific northwest.

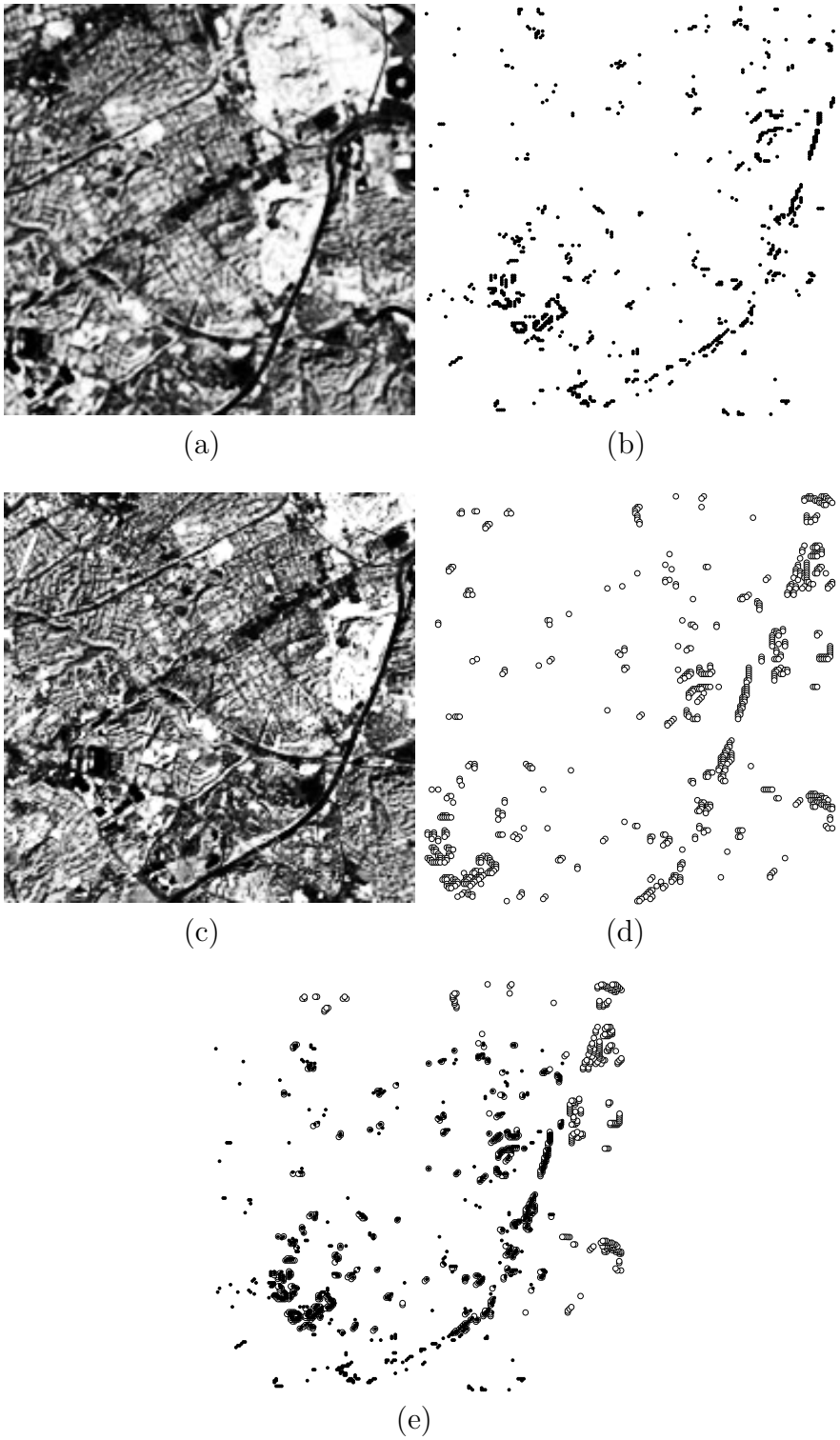


Figure 7: Matching two Landsat/TM images of Washington, DC.

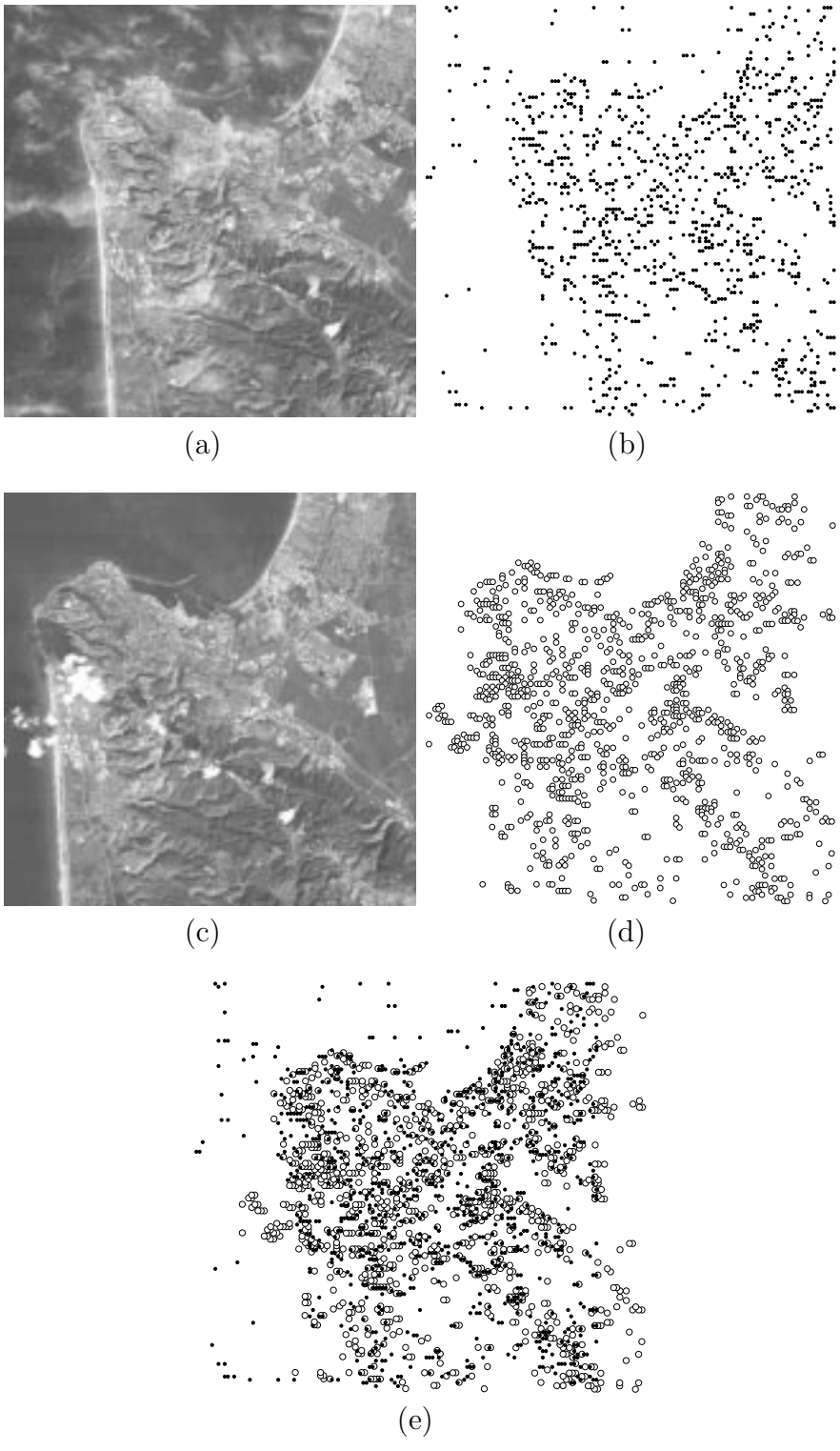


Figure 8: Matching two Landsat/TM images of Haifa, Israel.

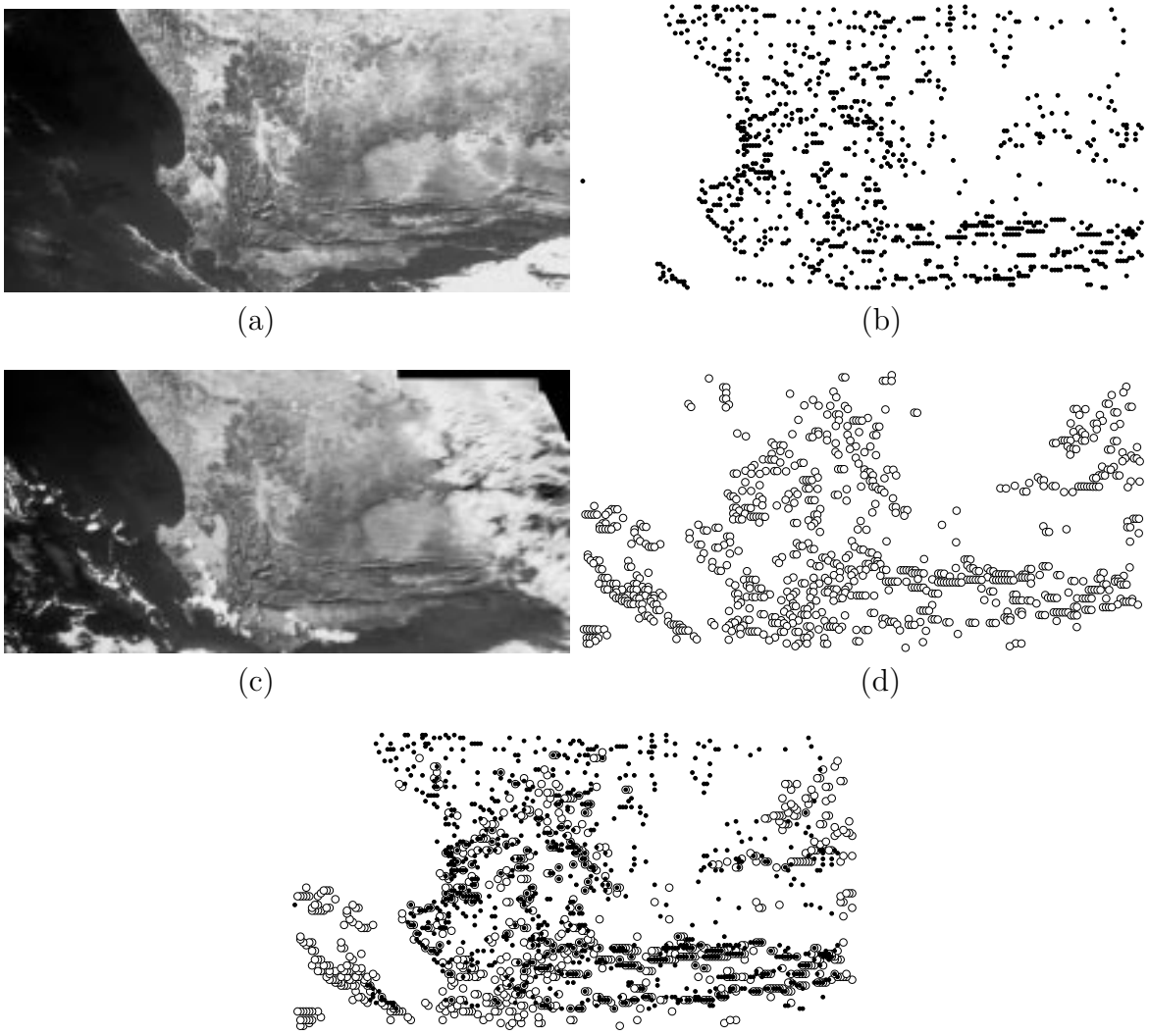


Figure 9: Matching two AVHRR images of South Africa.

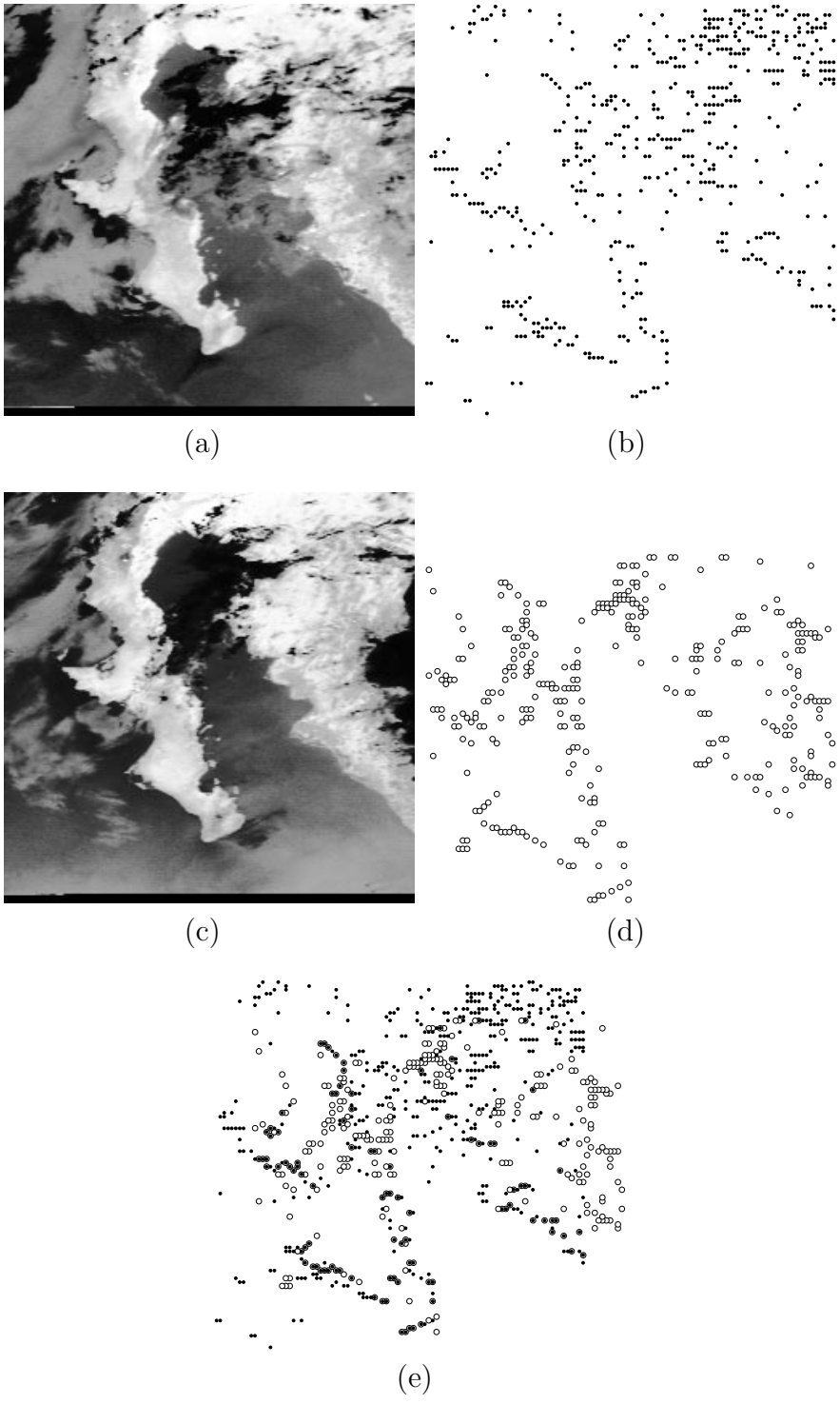


Figure 10: Matching two GOES images of Baja California.

5. H. Alt and L. J. Guibas. Discrete geometric shapes: Matching, interpolation, and approximation. Technical Report 96–11, Institut für Informatik, Freie Unversität Berlin, Berlin, Germany, 1996.
6. J. Le Moigne, W. Xia, J.C. Tilton, B-T. Lerner, E. Kaymaz, J. Pierce, S. Raghavan, S. Chettri, T. El-Ghazawi, M. Manohar, N. S. Netanyahu, W. J. Campbell, and R. F. Cromp. Towards an intercomparison of automated registration algorithms for multiple source remote sensing data. In *Proceedings of the CESDIS Image Registration Workshop*, pages 307–316, NASA Goddard Space Flight Center, November 1997. Also in NASA Publication CP–1998–206853.
7. J. Le Moigne, W. Xia, J.C. Tilton, T. El-Ghazawi, M. Manohar, N. S. Netanyahu, W. J. Campbell, and R. F. Cromp. First evaluation of automatic image registration methods. In *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium*, Seattle, Washington, July 1998.
8. M. McGuire and H. Stone. Techniques for multi-resolution image registration in the presence of occlusions. In *Proceedings of the CESDIS Image Registration Workshop*, pages 101–122, NASA Goddard Space Flight Center, November 1997. Also in NASA Publication CP–1998–206853.
9. D. L. Donoho and P. J. Huber. The notion of breakdown point. In P. J. Bickel, K. Doksun, and Jr. J. L. Hodges, editors, *A Festschrift for Erich L. Lehman*, pages 157–184. Wadsworth, Belmont, California, 1983.
10. P. J. Rousseeuw and A. M. Leroy. *Robust Regression and Outlier Detection.* Wiley, New York, 1987.
11. D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge. Comparing images using the Hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:850–863, 1993.
12. P. J. de Rezende and D. T. Lee. Point set pattern matching in d dimensions. *Algorithmica*, 13:387–404, 1995.
13. H. Alt, K. Mehlhorn, H. Wagener, and E. Welzl. Congruence, similarity and symmetries of geometric objects. *Discrete and Computational Geometry*, 3:237–256, 1988.
14. P. J. Heffernan and S. Schirra. Approximate decision algorithms for point set congruence. *Computational Geometry Theory and Applications*, 4:137–156, 1994.
15. H. Alt, O. Aichholzer, and G. Rote. Matching shapes with a reference point. In *Proceedings of the 10th Annual ACM Symposium on Computational Geometry.*, pages 85–92, Stony Brook, NY, June 1994.
16. L. P. Chew, M. T. Goodrich, D. P. Huttenlocher, K. Kedem, J. M. Kleinberg, and D. Kravets. Geometric pattern matching under Euclidean motion. *Computational Geometry Theory and Applications.*, 7:113–124, 1997.
17. M. T. Goodrich, J. S. Mitchell, and M. W. Orletsky. Practical methods for approximate geometric pattern matching under rigid motion. In *Proceedings of the 10th Annual ACM Symposium on Computational Geometry*, pages 103–112, Stony Brook, NY, June 1994.
18. D. P. Huttenlocher, K. Kedem, and M. Sharir. The upper envelope of Voronoi surfaces

and its applications. *Discrete and Computational Geometry*, 9:267–291, 1993.

19. S. Ranade and A. Rosenfeld. Point pattern matching by relaxation. *Pattern Recognition*, 12:269–275, 1980.
20. G. C. Stockman, S. Kopstein, and S. Benett. Matching images to models for registration and object detection via clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4:229–241, 1982.
21. A. Goshtasby and G. C. Stockman. Point pattern matching using convex hull edges. *IEEE Transactions on Systems, Man, and Cybernetics*, 15:631–637, 1985.
22. A. Goshtasby, G. C. Stockman, and C. V. Page. A region-based approach to digital image registration with subpixel accuracy. *IEEE Transactions on Geoscience and Remote Sensing*, 24:390–399, 1986.
23. C. F. Olson and D. P. Huttenlocher. Automatic target recognition by matching oriented edge pixels. *IEEE Transactions on Image Processing*, 6:103–113, 1997.
24. M. Hagedoorn and R. C. Veltkamp. Reliable and efficient pattern matching using an affine invariant metric. Technical Report RUU-CS-97-33, Dept. of Computing Science, Utrecht University, The Netherlands, 1997.
25. Klara Kedem and Yana Yarmovski. Curve based stereo matching using the minimum Hausdorff distance. In *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, pages C15–C18, 1996.
26. S. Irani and P. Raghavan. Combinatorial and experimental results for randomized point matching algorithms. In *Proceedings of the 12th Annual ACM Symposium on Computational Geometry*, pages 68–77, Philadelphia, Pennsylvania, May 1996.
27. Y. Lamdan and H. J. Wolfson. Geometric hashing: A general and efficient model-based recognition scheme. In *Proceedings of the 2nd International Conference on Computer Vision*, pages 238–249, Tampa, Florida, December 1988.
28. W. J. Rucklidge. *Efficient visual recognition using the Hausdorff distance*. Number 1173 in Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1996.
29. W. J. Rucklidge. Efficiently locating objects using the Hausdorff distance. *International Journal of Computer Vision*, 24:251–270, 1997.
30. J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3:209–226, 1977.
31. S. Arya and D. M. Mount. Algorithms for fast vector quantization. In J. A. Storer and M. Cohn, editors, *Proceedings of DCC '93: Data Compression Conference*, pages 381–390, Los Alamitos, California, 1993. IEEE Press.
32. S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. In *Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms*, pages 573–582, Arlington, Virginia, January 1994.
33. S. Arya and D. M. Mount. Approximate range searching. In *Proceedings of the 11th Annual ACM Symposium on Computational Geometry*, pages 172–181, Vancouver,

Canada, June 1995.

34. D. M. Mount and S. Arya. ANN: A library for approximate nearest neighbor searching. CGC 2nd Annual Fall Workshop on Computational Geometry, URL: `http://www.cs.umd.edu/~mount/ANN`, 1997.
35. J. Le Moigne, W. J. Campbell, and R. F. Cromp. An automated parallel image registration technique of multiple source remote sensing data. *IEEE Transactions on Geoscience and Remote Sensing*. To appear.