International Journal of Computer Vision 24(3), 251–270 (1997)

Efficiently Locating Objects Using the Hausdorff Distance*

WILLIAM J. RUCKLIDGE
rucklidge@parc.xerox.com

Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304

Received September 1995; Revised June 1996; Accepted August 1996

Abstract. The Hausdorff distance is a measure defined between two point sets, here representing a model and an image. The Hausdorff distance is reliable even when the image contains multiple objects, noise, spurious features, and occlusions. In the past, it has been used to search images for instances of a model that has been translated, or translated and scaled, by finding transformations that bring a large number of model features close to image features, and vice versa. In this paper, we apply it to the task of locating an affine transformation of a model in an image; this corresponds to determining the pose of a planar object that has undergone weak-perspective projection. We develop a rasterised approach to the search and a number of techniques that allow us to locate quickly all transformations of the model that satisfy two quality criteria; we can also efficiently locate only the best transformation. We discuss an implementation of this approach, and present some examples of its use.

Keywords: Hausdorff distance, object recognition, feature-based matching, efficient search, model-based recognition, affine transformation, distance transform, guaranteed search

1. Introduction

In this paper, we present a method for efficiently searching a space of transformations of a model to find transformations that minimise the Hausdorff distance between the transformed model and an image. The Hausdorff distance is a shape-comparison metric which performs well even when the image contains many features, multiple objects, noise, spurious features, and occlusions. A detailed, and more general, description of the work presented here can be found in (Rucklidge, 1995a).

The Hausdorff distance is a distance defined between two point sets. With suitable modifications, described in (Huttenlocher et al., 1993) and briefly summarised below, it tolerates errors in the positions of the points, as well as the presence of extra points (outliers) and missing points (due to occlusion or failure of some feature detector). In this work, one of the sets of points represents a model of some object, and the other represents an image to be searched for that object. These point sets are generally the output of some feature detector, such as a Canny edge detector: every pixel in the image where an edge was detected is considered to be a point of the image set; other feature detectors can be used as well.

Previous investigations of the Hausdorff distance studied locating an object that has been translated (Huttenlocher et al., 1993), or translated and scaled (Huttenlocher and Rucklidge, 1993). In this paper, we investigate locating an object after it has undergone an affine transformation. We have also applied the techniques presented here to searching those other transformation groups. This has produced a significant improvement in search performance for those problems over the previous methods.

*This work was supported in part by National Science Foundation PYI grant IRI-9057928 and matching funds from General Electric, Kodak and Xerox, and in part by Air Force contract AFOSR-91-0328.

Suppose that an object consists of a number of points lying on a plane in space. We project (using weak-perspective projection) each point of the object onto an image plane, creating one image point set. We then change the pose of the model (or, equivalently, the location of the image plane) and repeat the projection. Then there exists an affine transformation that maps every point in the first image point set onto the corresponding point in the second image point set; also, every affine transformation of the first image point set can be generated by appropriate choice of the second pose. Thus, given a (two-dimensional) model of a planar object, and an image containing that object, we search for an affine transformation of the model that brings it into close correspondence with a portion of the image; finding this two-dimensional transformation is essentially equivalent to finding the three-dimensional pose of the object. The "close correspondence" just mentioned is measured using the Hausdorff distance. The object does not need to be perfectly planar, as the change in its appearance as its pose changes is close to an affine transformation as long as the object is relatively shallow (its thickness is small relative to its distance from the camera). This type of search problem arises in many visual tasks, such as model-based recognition, image registration, motion tracking, mobile robot navigation and so on.

Since the space of affine transformations is much larger than the spaces searched in previous work involving the Hausdorff distance, we must increase the efficiency of the search techniques used to locate the model. In this paper, we present a few such efficiency-increasing extensions. We have found that, with these techniques, the affine transformation defining the pose of the model in the image can be found efficiently by examining in detail only a small part of the space of affine transformations, and ruling out the majority of the space once it is determined that it cannot contain a good enough pose. It is also possible to restrict the type of transformations being searched (for example, to the space of two-dimensional rigid motions); doing this reduces the search time. This efficient search allows us to use this powerful matching technique (the Hausdorff distance) on an interesting search space; we can guarantee that the search is equivalent to brute-force search but many times more efficient.

Previous work (e.g., (Ayache and Faugeras, 1986; Huttenlocher and Ullman, 1990; Cass, 1990; Olson, 1994; Bruel, 1992)) on feature-based model-based recognition has mostly been concerned with correspondence-based matching: building a one-to-one correspondence between model and image feature sets. This has often led to algorithms that have poor computational complexity: as the number of features increases beyond a few hundred, the time taken quickly becomes unreasonable. The Hausdorff distance builds no explicit correspondences, and we have been able to use it with model and image sets containing thousands of points.

The most similar previous work is that in (Borgefors, 1988; Paglieroni, 1992; Paglieroni et al., 1994). These papers describe systems that also do not build correspondences; they use chamfer matching, which is closely related to the Hausdorff distance, and also use related search techniques. We describe the connections between our work and these other systems in more detail in the following sections.

An earlier, shorter, version of this paper has appeared as (Rucklidge, 1995b).

The remainder of the paper is organised as follows. In Section 2, we describe the Hausdorff distance. In Section 3, we show how to locate the model by imposing a grid on the space of affine transformations, and (in principle) evaluating the Hausdorff distance at every transformation on this grid. We discuss the maximum error that is introduced by this discretisation of the continuous problem. Section 4 then describes some search techniques that greatly increase the speed with which the grid can be scanned, and Section 5 presents some examples.

2. The Hausdorff Distance

In this section, we briefly describe the Hausdorff distance, and the modified versions of it that we use in practice. More details can be found in (Huttenlocher et al., 1993; Rucklidge, 1995a).

The Hausdorff distance between two (finite) point sets I (representing an image), and M (representing a model of some object that we want to locate in that image) in the plane is defined as

$$H(M, I) = \max(h(M, I), h(I, M)) \qquad (1)$$

where

$$h(M, I) = \max_{m \in M} \min_{i \in I} \|m - i\| \qquad (2)$$

and $\|\cdot\|$ is some norm in the plane, which we here restrict to being the L_2 norm. $h(M, I)$ can be computed by taking each point of M, computing the distance from that point to the nearest point of I, and reporting the largest distance; $h(I, M)$ can be computed similarly, and the largest of these two distances is $H(M, I)$. $h(M, I)$, the *directed distance* from M to I, is small exactly when *every* point of M is near *some* point of I. Similarly, $h(I, M)$ is small when every point of I is near some point of M, and the *undirected distance* $H(M, I)$ is small when both of these are true. We call $h(M, I)$ the *forward distance*, and $h(I, M)$ the *reverse distance*.

If M and I are quite similar, except that a single point of M is far from every point of I, then $h(M, I)$ and thus $H(M, I)$ will be quite large. This sensitivity to outliers is not acceptable in practical recognition tasks. Our solution to this problem is to replace Eq. (2) with

$$h^f(M, I) = \underset{m\in M}{f\text{th}} \min_{i\in I} \|m - i\| \tag{3}$$

where $f\text{th}_{x\in X}\, g(x)$ denotes the fth quantile value of $g(x)$ over the set X, for some value of f between zero and one. For example, the 1th quantile value is the maximum and the $\frac{1}{2}$th quantile value is the median. This modified Hausdorff distance is called the *partial directed Hausdorff distance*. When $f = 1$, it is the same as the unmodified directed Hausdorff distance.

The *partial undirected Hausdorff distance* is now naturally defined as

$$H^{f_F f_R}(M, I) = \max(h^{f_F}(M, I), h^{f_R}(I, M)) \tag{4}$$

where f_F and f_R control what fraction is used when evaluating the forward and reverse distances respectively; these are called the *forward fraction* and *reverse fraction*. From now on, we will use only the partial distances because the partial distance is a more general concept.

In many instances, the model is considerably smaller than the image: a single image can contain several objects, so a single instance of the model might occupy only a small portion of the image, and so account for only a small fraction of the image's points. Ideally, we would prefer the image points that do not belong to the model object to be ignored in the computation of the reverse Hausdorff distance. We can do this by modifying the computation of the reverse (image to model) distance so that it considers only those points of the image lying near the target object. Suppose that the model is contained in a box $(x_{\min} \cdots x_{\max}, y_{\min} \cdots y_{\max})$ in the plane. The reverse Hausdorff distance from the image to the model is then computed based only on the points that lie inside this box; this defines the *box-reverse Hausdorff distance*:

$$h_{\text{box}}(I, M) = \max_{\substack{(i_x, i_y)\in I \\ x_{\min}\le i_x\le x_{\max} \\ y_{\min}\le i_y\le y_{\max}}} \min_{m\in M} \|m - (i_x, i_y)\|\,. \tag{5}$$

Thus, any points in the image that are not close to the model's position are ignored. As before, it is possible to extend this to a partial distance by replacing the max operation in Eq. (5) with a fth operation. We call this the *partial box-reverse Hausdorff distance* and denote it by $h_{\text{box}}^{f_R}(I, M)$.

The Hausdorff distance measures the mismatch between two sets (or portions of sets) at *fixed* positions with respect to each other. We are interested in comparing models to images, where the models can be transformed by members of some transformation group; in this paper, we consider only the group of affine transformations, though the techniques we present here can be applied to other groups (Rucklidge, 1995a).

Let t be an affine transformation. Define

$$d(t) = h^{f_F}(t(M), I) \tag{6}$$
$$d'(t) = h_{\text{box}}^{f_R}(I, t(M)) \tag{7}$$
$$D(t) = \max(d(t), d'(t)) \tag{8}$$

where $t(M)$ is the set obtained by applying t to every point $m \in M$. These are respectively the partial forward distance, partial box-reverse distance, and undirected distance as a function of the transformation t; they are implicitly parameterised by the fractions f_F and f_R. We want to find transformations t that make $d(t)$ and $d'(t)$ as small as possible; these transformations are then the locations in the image of the model object that we are trying to find.

A somewhat similar mismatch measure is used in *chamfer matching*, which was first described in (Barrow et al., 1977). In chamfer matching, the mismatch between the transformed model and a portion of the image is computed by taking the mean or RMS (Borgefors, 1988) distance from each transformed model point to the nearest image point. We instead take some quantile value of these distances rather than their mean or RMS value. This has the advantage that it is not affected at all by a moderate number of outliers, whereas a single poorly-matched outlier can significantly alter the mean distance value; such an outlier might be generated by a section of the model that was occluded in the image, and so was far from any image point. The chamfer matching methods also do not have any equivalent to the reverse distance. The reverse distance is crucial in eliminating transformations where the model is erroneously matched to a portion of the image with a high density of image edges.

3. Rasterisation

In many applications, the input image and model point sets do not contain points at arbitrary locations, but instead contain only points located on some integer grid. Such a point set is essentially a binary array: every location on the integer grid, within some bounds, is either a member of the set or not; any location outside these bounds is not a member of the set. For example, a thresholded scanned document is such a binary array, as is the output of many commonly used edge detectors. We assume that all the image and model points have integral, non-negative x and y coordinates. Since an affine transformation includes a translation, this assumption can be made without loss of generality.

Recent work (Rucklidge, 1995a) has shown that finding the affine transformation that *exactly* minimises the Hausdorff distance between a transformed model and an image is likely to be computationally difficult; for example, there are instances where the Hausdorff distance between two point sets of size n, considered as a function of affine transformation, has $\Omega(n^9)$ local minima, making any combinatorial search extremely expensive. We have therefore investigated algorithms that approximate the search for the exact minimising transformation. Since the model and image point sets are rasterised, one approach to finding an approximate minimum Hausdorff distance algorithm is to extend this rasterisation. We impose a grid on the space of affine transformations; this grid's fineness (or *pitch*) is determined by the pitch of the image and model grid. This section discusses the details of such a rasterisation of transformation space.

3.1. How to Rasterise

Given the image and model arrays, we want to determine how to rasterise the transformation space: what the basis vectors and pitch of the grid will be. The idea here is to make this as uniform a quantisation as possible. Suppose that two transformations t and $\hat{t}$ are directly adjacent on the grid of transformations: they have identical parameters, except in one dimension, in which they differ by only one grid step. This small change in the transformation parameters should produce only a small change in the transformed model. Thus, for any $m \in M$, $t(m)$ and $\hat{t}(m)$ should be neighbours on the image grid: they must differ by at most one grid step, in one dimension;

In the case of the group of affine transformations, we can accomplish this in the following manner. Let $x_{\max} = \max_{(m_x,m_y)\in M} m_x$ and $y_{\max} = \max_{(m_x,m_y)\in M} m_y$. The points in the model are bounded by $0 \le m_x \le x_{\max}$ and $0 \le m_y \le y_{\max}$ for all $(m_x, m_y) \in M$. An affine transformation can be considered to be a 2×2 matrix A together with a translation (t_x, t_y), representing a mapping of the plane

$$(x, y) \to (a_{00}x + a_{01}y + t_x, a_{10}x + a_{11}y + t_y) \qquad (9)$$

when A is the matrix

$$A = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix}.$$

This transformation can be represented as a six-tuple $(a_{00}, a_{01}, a_{10}, a_{11}, t_x, t_y)$. We now define six basis vectors e_1 through e_6,

$$\begin{aligned} e_1 &= \left(\frac{1}{x_{\max}}, 0, 0, 0, 0, 0\right) \\ e_2 &= \left(0, \frac{1}{y_{\max}}, 0, 0, 0, 0\right) \\ e_3 &= \left(0, 0, \frac{1}{x_{\max}}, 0, 0, 0\right) \\ e_4 &= \left(0, 0, 0, \frac{1}{y_{\max}}, 0, 0\right) \\ e_5 &= (0, 0, 0, 0, 1, 0) \\ e_6 &= (0, 0, 0, 0, 0, 1) \end{aligned}$$

Let t be an affine transformation and $m \in M$ be a point of the model. $t(m)$ and $(t + e_1)(m)$ differ only in their x coordinates, and differ by no more than one pixel; the same holds for e_2 and e_5. These three basis vectors are thus called x-*linked*. Similarly, e_3, e_4 and e_6 are y-linked. Now, a transformation whose coordinates are $[i_1, i_2, i_3, i_4, i_5, i_6]$ in this raster basis (formed by e_1 through e_6) is the same as the natural basis transformation $(i_1/x_{\max}, i_2/y_{\max}, i_3/x_{\max}, i_4/y_{\max}, i_5, i_6)$; to avoid confusion, we will denote transformations represented in the raster basis using square brackets. Adding any x-linked basis vector to a transformation adds some value between 0 and 1 to the x coordinate of every transformed model point, without affecting the y coordinate, and similarly for adding any y-linked basis vector. These constraints allow us to prove some very useful results about the effects of restricting transformations to those lying on the grid. These are presented in Subsection 3.2.

We now introduce some notation for dealing with both the rasterised and continuous versions of transformations. The basic notation is that angle brackets, $\langle\cdot\rangle$, represent a rounding function. Also, square brackets will be used to distinguish between two versions of some functions: the function notated with square brackets has some implicit rounding property, which is not present in the original version.

Suppose that t is a transformation and $m \in M$ is a point of the model. Then $\langle t(m)\rangle$ denotes the result of applying t to m and rounding this to the nearest point on the image grid, by rounding in each dimension. This will also be written as $t[m]$, making the rounding after the application of t implicit. $t[M]$ denotes the result of rounding each point of $t(M)$. Note that this is a multiset: several distinct points in M can transform to the same point in $t[M]$, and (for counting purposes) should be kept distinct. The Hausdorff distance requires no modifications to be extended to multisets.

Analogously to Eqs. (6) through (8), we can now define

$$d[t] = h^{f_F}(t[M], I) \tag{10}$$

$$d'[t] = h^{f_R}_{\text{box}}(I, t[M]) \tag{11}$$

$$D[t] = \max(d[t], d'[t]) \tag{12}$$

which are rounding versions of the forward, box-reverse, and undirected distances. $d[t]$ is thus computed by taking each point of m, transforming it by t, rounding the result, and determining the distance to the closest point of I. The f_Fth quantile of these distance values (one value for each point of M) is then the value of $d[t]$. Similarly, $d'[t]$ is computed by determining the distance from each point of I lying under the transformed model box (recall the definition of the box-reverse distance) to the closest point of $t[M]$; the f_Rth quantile value of these distances is then $d'[t]$.

This process has at its heart the *evaluation* of transformations: for each transformation t, we compute the forward and reverse distances $d[t]$ and $d'[t]$. Each of these transformation evaluations involves computing, for each point of $t[M]$, the closest point in I, and vice versa. Since all the points of $t[M]$ have integral coordinates, we can compute the *distance transform* of I

$$\Delta[x, y] = \min_{i\in I} \|(x, y) - i\|\,. \tag{13}$$

Each point of m then simply probes $\Delta[t[m]]$ to determine the distance to the closest point of A. Algorithms to compute distance transforms can be found in (Danielsson, 1980; Paglieroni, 1992; Breu et al., 1995) and are not discussed here.

Other transformation groups than affine transformations can also be rasterised using the general concepts presented in this subsection. The most important principle to keep in mind when choosing a parameterisation and a rasterisation of a transformation space is that a small (one grid step) change in transformation parameters should yield only a small change in the location of any transformed point. If possible, the parameters should be separated into x-linked and y-linked parameters. As we shall see in Subsection 4.2, this allows us to do some pre-computation to help accelerate the search.

In (Borgefors, 1988), an essentially identical criterion is used to choose parameter quantisation: the parameter step length should be the smallest length that changes the position of one transformed model point by one pixel. However, the transformation groups considered in that paper are interlinked, in that the change in a model point's position as one parameter varies depends on the values of the other parameters. The parameterisation of affine transformation space we have chosen does not have this property, which allows us to use a uniform grid in parameter space. Transformation groups with inextricably interlinked parameters would require a much more complex grid, as was described by Borgefors.

3.2. Bounds on Rasterisation Error

Rasterising transformation space, as described in Subsection 3.1, involves replacing the continuous space of transformations with a discrete one. This process introduces error into the computation of the transformation minimising the Hausdorff distance, as almost all transformations are no longer considered. Rounding the coordinates of the transformed model points also introduces error. In this subsection, we determine how large the effects of these errors can be.

The following lemma shows that a small perturbation of the points in one set has only a small effect on the (continuous, unrounded) Hausdorff distance.

Lemma 1. *Suppose that $h^{f_F}(M, I) = d$. Let M' be obtained by perturbing each point of M by some displacement of magnitude at most δ. Then the value of $h^{f_F}(M', I)$ is bounded by $d \pm \delta$.*

Proof: (Sketch) $h^{f_F}(M, I)$ is computed by finding the closest point of I to each point of M, and reporting the f_Fth quantile of these values. Perturbing each point of M by at most δ changes its distance to the closest point of I by at most δ; the f_Fth quantile therefore changes by at most δ. □

Lemma 1 also applies to the partial reverse distance, $h^{f_R}(I, M)$, but not to the partial box-reverse distance $h^{f_R}_{\text{box}}(I, M)$.

We can now show a bound on the maximum error in the minimum Hausdorff distance caused by rasterising transformation space and rounding transformed points.

Theorem 1. *Let t^* be a transformation of M that minimises $d(\cdot)$. Let $\hat{t}^*$ be a transformation of M, lying on the grid we have imposed on the space of affine transformations, which minimises $d[\cdot]$. Then $d[\hat{t}^*]$ exceeds $d(t^*)$ by no more than $2\sqrt{2}$.*

Proof: (Sketch) Rounding each point of $t(M)$, for some transformation t, perturbs each point by at most $\sqrt{2}/2$; $d[t]$ and $d(t)$ therefore differ by at most $\sqrt{2}/2$ by Lemma 1. Let $\langle t^* \rangle$ be the result of rounding t^* to the nearest point which lies on the grid of transformations, by rounding it in each of the transformation parameter dimensions. Each transformation parameter of $\langle t^* \rangle$ differs from the corresponding parameter of t^* by at most $\frac{1}{2}$. Corresponding points in $t^*(M)$ and $\langle t^* \rangle(M)$ therefore differ in their x coordinates by at most $\frac{3}{2}$, and in their y coordinates by at most $\frac{3}{2}$. $d(t^*)$ and $d(\langle t^* \rangle)$ therefore differ by at most $3\sqrt{2}/2$, and therefore $d[\hat{t}^*] \leq d[\langle t^* \rangle] \leq d(t^*) + 2\sqrt{2}$. □

Thus the minimum value of the rasterised approximation $d[\cdot]$ specifies the minimum Hausdorff distance under transformation to an accuracy of a few units of quantization. These bounds are for the worst-case error; in practice, errors are typically no more than half this worst-case error.

3.3. Computing the Minimum Hausdorff Distance

Once the space of transformations has been discretised, we need to determine the grid transformation t that minimises the Hausdorff distance between the image I and the (rounded) transformed model $t[M]$. In many cases, however, we do not wish to find simply the best (minimising) transformation, but instead find all transformations that bring the image and transformed model close to one another. More precisely, we want to find all transformations t that satisfy two criteria:

Forward Criterion. The partial forward distance, $d[t]$ (implicitly parameterised by f_F) must be no more than the *forward threshold* τ_F.

Reverse Criterion. The partial box-reverse distance, $d'[t]$, (parameterised by f_R) must be no more than the *reverse threshold* τ_R.

f_F, f_R, τ_F and τ_R are set by the user to control which transformations are reported. A transformation satisfying both the forward and reverse criteria is called a *match*.

These criteria can also be viewed in a different way. Let t be some transformation. The forward distance $d[t]$ is below τ_F exactly when at least $\lceil f_F \#(M) \rceil$ of the transformed and rounded model points lie within τ_F of some image point. We can thus compute a new function, $f[t]$, defined by

$$f[t] = \frac{\#(\{m \in M \mid \Delta(t[m]) \leq \tau_F\})}{\#(M)}. \tag{14}$$

This function, which is implicitly parameterised by τ_F, is the *fraction* of all the model points that, when transformed by t and rounded, lie within τ_F of some image point. $f'[t]$ can be defined similarly: it is the fraction of the image points that lie within τ_R of some model point, out of the image points lying underneath the transformed model box (i.e., out of the image points contributing to the box-reverse distance).

The forward and reverse criteria can now be expressed in terms of $f[t]$ and $f'[t]$:

Forward Criterion. $f[t]$ must be at least f_F.
Reverse Criterion. $f'[t]$ must be at least f_R.

These new formulations are exactly equivalent to the original ones: from the definition of the partial Hausdorff distance, $d[t] \leq \tau_F$ exactly when $f[t] \geq f_F$, and similarly for d' and f'.

Locating Matches. Since the image and model are essentially binary arrays, the sizes of these arrays can be used to determine what portion of the space of affine transformations must be searched. We eliminate from consideration all those transformations where the transformed model does not lie inside the bounds of the image array. Finding all matches is now simple: enumerate all the transformations on the grid of transformations, where the transformed model lies within the image bounds. Determine, for each one, if it satisfies the forward and reverse criteria. Report all those that do.

The reverse Hausdorff distance from the image I to the transformed model $t[M]$ is computed using the distance transform of $t[M]$. Note that this distance transform may have to be recomputed for each different transformation t, while the distance transform of the image does not.

The similarity measure $f[t]$ is similar to the first-level verification criterion described used in alignment matching (Huttenlocher and Ullman, 1990). In that work, a number of candidate transformations are generated by hypothesising alignments between triples of image points and triples of model points. These are then evaluated by determining how many of the transformed model points lie close enough to a compatible image point (the points have associated alignments which must match for the model and image point to be compatible). The transformation is evaluated in more detail (by matching edge segments) only if the fraction of model points matching some image point is high enough. This is similar to the forward criterion. However, the alignment method considers only transformations generated by aligning a model point triple with an image point triple; in such a transformation, at least three transformed model points are *directly* superimposed on three image points. Even if the correspondences between the three model points and the three image points are correct, this “pinning” can result in the other model points being too far away from the corresponding image points. The best transformation may be one where no model point is directly on top of some image point, but where the error is spread around more evenly, rather than being forced to zero in one place, causing unpredictable increase in another place. As we are searching transformation space, rather than correspondence space, we are not restricted to such pinned transformations.

4. Efficient Computation

The exhaustive search of transformation space described in Subsection 3.3 locates all transformations satisfying the forward and reverse criteria (i.e., all matches of the transformed model to the image). However, there are many transformations to be considered, making evaluating all of them very costly.

This section presents our approach to increasing the efficiency of this search. We want to modify the search only in ways that do not miss anything: if a transformation satisfying our criteria exists, it must be found. Thus, we must manage to eliminate most of the transformations from consideration, and process those that remain efficiently, while ensuring that a good transformation can never be accidentally eliminated.

We apply our approach only to the forward distance and the forward criterion, and not to the reverse distance, because the forward distance is more constrained than the reverse distance. Specifically, Lemma 1 applies to the forward distance, but does not apply to the box-reverse distance, so it is more difficult to limit the value of the reverse distance of a transformation than the forward distance. In practice, we have found the strategy of searching (efficiently) for all transformations that satisfy the forward criterion then rejecting those transformations that do not satisfy the reverse criterion to be successful. Most of the total time is spent in the first, efficient search, phase; the second, verification, phase is almost never a bottleneck.

Several efficient search techniques have previously been presented in (Huttenlocher et al., 1993; Huttenlocher and Rucklidge, 1993). In this paper, we present some enhancements and refinements of these techniques, as well as some new techniques.

4.1. Cell Decomposition

As in (Huttenlocher and Rucklidge, 1993), we search transformation space using a cell decomposition (divide and conquer) strategy. A *cell* is a rectilinear axis-aligned region of transformation space.

- *Step 1.* Initialise a list of *possibly interesting* cells to a list of cells that are known to contain all transformations of interest. This is done by tiling the transformation space with cells of a certain size, so that all valid transformations (where the transformed model lies inside the image bounds) are covered.
- *Step 2.* Scan over the current list of cells. For each cell, determine if it is possible that the cell contains a transformation t for which $d[t] \leq \tau_F$ (we will discuss how this determination is made in Subsection 4.2). If so, mark this cell as *interesting*.

- *Step 3.* Once the entire list has been scanned, make up a new list of smaller cells (all the same size) by subdividing those cells found to be interesting. Cells found to be uninteresting need not be considered any further. Repeat Steps 2 and 3 with this new list of (smaller) possibly interesting cells. Terminate when the current cell size becomes small enough (this will be made more formal below).

This algorithm is essentially an evaluate-subdivide cycle. Each cell is evaluated, and accepted (labeled interesting) or rejected (labeled uninteresting). Interesting cells are subdivided and their children in turn evaluated. All the cells of a certain size (i.e., all the cells in one level in this process) are evaluated, then all the cells at the next level generated by subdividing the interesting ones, and so on. The final result is a list of transformations of the model that bring it into close correspondence with (a subsection of) the image.

The subdivision step takes all those cells that were labeled interesting and determines a set of smaller (finer resolution) cells that completely cover them. These finer-resolution cells are obtained by taking the coarser-resolution cell and dividing it in each dimension by a fixed constant r (we have been using $r = 2$); one coarser-resolution cell is therefore divided into r^6 finer-resolution cells. Any dimension that would be smaller than a single grid step is held at one grid step. We want this refinement to have the following properties:

- The finer-resolution cells should all be the same size.
- Each finer-resolution cell should be completely contained inside a single coarser-resolution cell; it should not overlap multiple coarser-resolution cells.
- The finer-resolution cells should not overlap each other.
- The finer-resolution cells should be as square as possible: their side lengths should be roughly equal in all dimensions.

Having all the cells at a certain level in the search be the same size makes evaluating a list of them more regular. The second and third properties make sure that a region of transformation space is not scanned twice. The fourth property is for efficiency: the number of grid transformations covered by a cell depends on its volume, whereas (as we will see) cells are eliminated using a criterion based on their edge lengths. The maximum ratio of volume to edge lengths is achieved when the cell has edge lengths equal in all dimensions.

The simplest way to ensure that these properties always hold is to set the cell sizes so that the lengths of the cell edges, in each dimension, are always a power of r, and are equal to each other. This is done by choosing the sizes of the coarsest-resolution cells appropriately.

If the cells at the current level are a single grid step wide in each dimension (i.e., each cell includes only one grid transformation), then no further refinement is possible. At this point, the list of interesting cells from the current level contains all those grid transformations t for which $d[t] \le \tau_F$. We then consider each of these individually. For each one, we compute the distance transform of the transformed model, and use this to compute the box-reverse distance $d'[t]$. We reject all those transformations for which this is greater than τ_R. Thus, we are left with a list of all the transformations that meet both the forward and reverse criteria.

Borgefors (1988) also uses a hierarchical search of a discretised transformation space. A number of start positions are seeded on a low-resolution grid in transformation space. At each level of the search, the positions from the next coarser level are improved by a hillclimbing algorithm, and pruned using some quality-of-match heuristics. These heuristics are evaluated with respect to the current level of an edge pyramid, formed by taking the input image and repeatedly decimating it; a decimated model is also used. In contrast to this, no matter what level of the search is being performed, our method uses a full-resolution model and image. This allows us to guarantee that our cell pruning step is computing the exact values it needs, not values that are approximate because the complete image is not being used. Borgefors' system is not guaranteed, in that it might fail to locate the model when it is actually present in the image, and would have been located by an exhaustive search.

Paglieroni et al. describe in (Paglieroni et al., 1994) a chamfer matching system that uses a somewhat similar search strategy (which is also similar to that in (Huttenlocher et al., 1993)). They search the (discretised) space of two-dimensional rigid motions by computing the chamfer mismatch function for a transformation, and if that value exceeds a threshold (or the best mismatch value so far seen), they eliminate that transformation, and also some surrounding transformations. They then go on to another transformation, skipping all those that have previously been eliminated directly or indirectly. The order in which translations are evaluated samples them first on a coarse grid, then fills in the gaps with a finer grid, and so on. However, this approach is unsuitable for high-dimensional parameter spaces, as it is necessary to keep track of all the transformations that have been eliminated; the book-keeping overhead is unacceptable for transformation grids containing many million transformations.

4.2. The Box Distance Transform

The key to the cell subdivision method is Step 2, where we evaluate each cell and eliminate those that cannot possibly contain a transformation that satisfies the forward criterion, without inadvertently eliminating any cells containing such a transformation. We decide whether to subdivide a cell by computing a bound on the maximum possible value attained by $f[t]$ for any t in the current cell R, and rejecting the cell if this bound is below f_F. This is equivalent to rejecting the cell if we can prove that it cannot possibly contain any transformation satisfying the forward criterion.

Let R be a cell in transformation space whose sides are aligned with the grid. Call the top left transformation (the transformation in R whose parameters all have their lowest values) t^l and the bottom right transformation (whose parameters all have their highest values) t^h. Let $t \in R$ be a transformation, and $m \in M$ a model point. Then $t(m)$ must lie within a box in the image whose top left corner is $t^l(m)$ and whose bottom right corner is $t^h(M)$ (where the origin of the image is considered to be its top left corner). This is because increasing any parameter of a transformation cannot decrease the coordinates of any transformed model point, so the minimum and maximum values for the coordinates of the transformed point are achieved when the transformation parameters achieve their maximum or minimum values, at t^h and t^l respectively. In some sense, the projection of a box in transformation space (via a point of M) is a box in (x, y) space.

Suppose that the parameters of the transformations t^l and t^h (expressed in the integer transformation basis) are

$$\begin{aligned} t^l &= \left[a^l_{00}, a^l_{01}, a^l_{10}, a^l_{11}, t^l_x, t^l_y\right] \\ t^h &= \left[a^h_{00}, a^h_{01}, a^h_{10}, a^h_{11}, t^h_x, t^h_y\right]. \end{aligned}$$

Since these are expressed in the grid basis, all the coordinates are integers. Let $w = (a^h_{00} - a^l_{00}) + (a^h_{01} - a^l_{01}) + (t^h_x - t^l_x)$ and $h = (a^h_{10} - a^l_{10}) + (a^h_{11} - a^l_{11}) + (t^h_y - t^l_y)$. w is the sum of R's edge lengths in the three x-linked dimensions, and h is the sum in the y-linked dimensions. Note that w and h do not depend on the location of R but only on its size: moving R changes both t^l and t^h but not w or h. Let $\mu = t^l[m]$ for some model point $m \in M$. As t varies within the cell R, $t[m]$ varies within a $w+1$-by-$h+1$ rectangle whose top left corner is μ.

Define

$$\Delta'_{wh}[x, y] = \min_{\substack{0 \le x' \le w \\ 0 \le y' \le h}} \Delta[x + x', y + y']. \tag{15}$$

Any portions of $\Delta[x, y]$ outside the boundary of the array it is stored in can be treated as being infinite, as they lie outside the image.

Δ'_{wh} is called the *box distance transform* of the image A, using a box of size w by h. If R contains only a single grid transformation, then $t^l = t^h$, $w = h = 0$, and $\Delta'_{wh}[x, y] = \Delta[x, y]$: the box distance transform is equal to the regular distance transform when only a single transformation is being considered. Given the array Δ, Δ'_{wh} can be computed in $O(\log(w) + \log(h))$ passes through Δ, using standard prefix techniques.

Now, $\Delta'_{wh}[\mu]$ is the minimum value that could be achieved by $\Delta[t[m]]$ for any $t \in R$: as t varies, $t[m]$ also varies, but the range of its variation is bounded by a box corresponding to the area of Δ examined to compute $\Delta'_{wh}[\mu]$, so $t[m]$ cannot possibly get closer than $\Delta'_{wh}[\mu]$ to some point of A.

In order to evaluate a cell, and determine if it is interesting or not, we compute Δ'_{wh} according to the size of the cell (i.e., according to w and h), transform each model point $m \in M$ by the top left transformation t^l and probe Δ'_{wh} at the (rounded) locations of the transformed points. We count the number of these values that are τ_F or below. If the fraction of such small values is at least f_F, then we label the cell as interesting. If this value is below f_F, then no transformation can bring enough model points close enough to some image point for the forward criterion to be satisfied, and so we reject the cell. This can never reject any cell that contains a grid transformation t satisfying the forward criterion: any such transformation must map at least $\lceil f_F \#(M) \rceil$ points of M onto locations where $\Delta[t[m]] \le \tau_F$, and so for each of those points, $\Delta'_{wh}[t^l[m]]$ can be no more than τ_F.

We can increase the efficiency of this cell evaluation by checking the count of small values against the number of values yet to be determined: if too many large values have already been seen, so that no matter how small the remaining, unseen, probe values are, the fraction of small values for the entire evaluation cannot be large enough (at least f_F), then we can immediately stop the evaluation. A similar observation can be applied when checking the reverse criterion.

Since all the cells of any given level of the multi-resolution search are the same size, we can use a single set of w and h values to compute a single array Δ'_{wh} that is used for the entire level. Each level of the search, evaluating cells of a certain size, must compute a different Δ'_{wh} array.

The only way in which the values of Δ'_{wh} are used is comparing them against τ_F, so we may threshold the array Δ'_{wh} at τ_F. This new boolean array is much smaller (only one bit per pixel is required to store it), so less memory is required; this reduction in memory usage also generally results in a slight speed improvement.

It might not be possible to construct an exact equivalent to the box distance transform for other transformation groups than the group of affine transformations: a box in transformation space might not project to a box in the image. In this case, it is best to compute a worst-case bounding box for the projection of a cell of a certain size and use that to compute the box distance transform. An adaptation of the zoning technique, described in the next subsubsection, could also be used to reduce the size of the worst-case box.

The effectiveness of this technique is greatly affected by the degree of clutter in the image. If the image is very dense, then for any moderate box size, almost every box contains an image point, and so almost every entry in the box distance transform is zero, making it difficult to prune the search. Experimentally, we have observed, as expected, that the search takes longer to examine dense areas of the image than it does to examine sparser areas.

Zoning. Different model points are affected differently by changes in the transformation parameters. For example, suppose that $m = (m_x, m_y) \in M$ is a model point, and that $m_x = 0$. Changing the a_{00} or a_{10} parameters of a transformation t does not affect $t[m]$ at all. In general, model points with smaller x coordinates are less affected by changes in these two parameters than model points with larger x coordinates. The same holds for y coordinates and a_{01} and a_{11}. Thus, the box in (x, y) space over which $t[m]$ can move as t is varied within a cell R is not the same for all $m \in M$; the w and h computed for R specify the largest such box for any model point, and Δ'_{wh} is computed using this worst-case box size.

We could compute a different Δ'_{wh} for each point of M, using the correct w and h (computed using this point's particular x and y coordinates) in Eq. (15), and have each transformed point probe the corresponding Δ'_{wh}. This would ensure that the correct-sized box was always used. However, in practice, this approach is infeasible, as it requires far too much storage: one copy of a boolean array the size of the image for each point of M, for each level of the multi-resolution search. We therefore compromise: instead of computing a different Δ'_{wh} for each point $m \in M$, we divide the model M up into several zones, typically by dividing it into equal-sized rectangles and labeling all the points inside each rectangle as members of a single zone. Each zone then determines, based on the current cell size and the coordinates of the model points within that zone, a value for w and h and thus a Δ'_{wh} array. When cells are evaluated, each transformed model point probes the box distance transform that was constructed for its zone, and determines if the probed value is greater than τ_F or not. In the upper limiting case, each point of M is in its own zone, and in the lower limiting case, all of M is in a single zone. Search efficiency increases as the number of zones increases, but so do memory requirements, as does the cost of computing all the Δ'_{wh} arrays. We currently divide the model into 16 zones, by quartering it in both x and y. As before, each box distance transform can be replaced with a binary array.

4.3. Directed Search Methods

The search method described in Subsection 4.1 attempts to find all transformations that satisfy both the forward and reverse criteria. This task is that of "find all matches". In some cases, however, this is not what the application performing the search actually requires: it might be satisfied with any match, or might want only the best match. For the moment, let us consider only the forward criterion. Three typical requirements then might be:

1. Determine if there are any transformations that meet the forward criterion and return one if so ("find any match").
2. Find the largest possible value of f_F for which there is some transformation that meets the forward criterion and return such a match ("find the best match, ranked by $f[t]$").
3. Find the smallest possible value of τ_F for which there is some transformation that meets the forward criterion and return such a match ("find the best match, ranked by $d[t]$").

These searches can be performed more efficiently than the full search. They can also be performed using a single modified search framework.

Find Any Match. Recall that the cell decomposition search method takes a list of cells, subdivides each, and determines whether or not each of the subdivided cells is interesting (might contain a match); the list of interesting cells is then used for the next level of subdivision. This method corresponds to a breadth-first search of the tree of cells. If we are interested in finding only a *single* transformation that satisfies the forward criterion, we can terminate the search once such a transformation is found. However, this eliminates only a small portion of the tree search: we might be able to avoid scanning some of the finest-resolution cells, but the rest of the work will already have been done. Searching the cells in another order than breadth-first could decrease the time to find such a match, by reaching some promising finest-resolution cells more quickly. One way of doing this is by performing a best-first search: instead of expanding all the cells in the current list, we instead expand only the *best* ones, according to some measure that attempts to quantify which cells are most likely to contain a match. This is a standard beam search, which may lead us to a match quickly, if the measure we are using to rank the cells at all levels is accurate enough. However, it might not do so, in which case we have to backtrack. If there are no matches, the entire tree must be searched, but if a match does exist it should be found more quickly than by breadth-first search. Once a match is found, of course, the search can be immediately terminated; any as-yet-unevaluated cells need never be evaluated.

The beam width used (the number of cells on the current list that are expanded at one time) affects the performance of the search. Setting the beam width too high creates a breadth-first search, as we had before; setting it very low makes this procedure a ranked depth-first search, which may not perform well. In practice, we adjust the beam width dynamically: it starts out quite small, but is increased if the search finds itself exhausting its lists and backtracking too much.

The decision as to whether or not a cell should be rejected is made by counting the results of probes generated by the transformed model points: enough probes into the appropriate Δ'_{wh} must have values not exceeding τ_F. Thus, one measure that can be used for ranking cells is simply the fraction of the probe values that were below τ_F. This is generated as a consequence of the evaluation of the cell in any case, so using it to rank the cells involves no extra cost other than that of actually sorting the list of cells. Cells that contain matches are more likely to have high values of this fraction than cells that do not. Any ranking heuristic could be used without affecting the correctness of the search; this one has the advantage that it is free.

Find the Best Match. As stated above, we may wish to find the maximum possible value of f_F for which some transformation meets the forward criterion, along with such a transformation (this is equivalent to finding the transformation t maximising $f[t]$). We perform the same form of search as in the previous subsubsection: trying to find a transformation satisfying the forward criterion as early in the search as possible. Whenever such a transformation is found, we evaluate $f[t]$ rather than terminate the search. This value is always at least f_F, since t satisfies the forward criterion. t would also have satisfied the forward criterion if f_F had any larger value up to and including $f[t]$. This means that, once t has been found, we can increase f_F by setting it to $f[t]$, and still be certain that the search will find a transformation satisfying the forward criterion (since t satisfies this modified criterion). Further, no transformation t' for which $f[t'] < f[t]$ will be found after f_F has been increased.

The best-first search can therefore be modified in the following manner: instead of terminating the search as soon as a transformation satisfying the forward criterion is found, we increase f_F whenever one is found, and continue with the search. This strengthening of the forward criterion causes the search to terminate more quickly, because more cells are considered uninteresting, and so the search space is exhausted more quickly than with the original value of f_F; we prune out cells that cannot contain anything better than what we have already found. At the end of the search, we are certain that we have found the transformation t^* that has the maximum value of $f[\cdot]$; the final value of f_F is equal to $f[t^*]$, as it was raised to that value when t^* was found. Only t^* and other transformations having the same $f[\cdot]$ value need to be reported.

We can also use this parameter update technique to search for the minimum possible value for τ_F for which some transformation meets the forward criterion. In this case, f_F is held constant, and whenever a transformation t satisfying the current forward criterion is found, τ_F is lowered to $d[t]$, strengthening the forward criterion.

These optimising search methods are essentially searching for one of the method's parameters (f_F or τ_F) in parallel with the usual search for matches. This reduces the number of parameters that need to be specified for the search process.

Incorporating the Reverse Criterion. In describing these directed search techniques, we have so far been concerned only with the forward criterion. The reverse criterion can be incorporated in one of several manners. If only a single transformation satisfying both the forward and reverse criteria is required, as in Subsection 4.3, then we may simply evaluate every transformation that satisfies the forward criterion to see if it also satisfies the reverse criterion; when some transformation satisfies both, then the search can be terminated.

The search modifications described in Subsection 4.3 can also have the reverse criterion incorporated in this manner: any transformation satisfying the current forward criterion must also pass the reverse criterion before f_F or τ_F can be updated. However, it might be desirable to have f_R (or τ_R) track f_F (or τ_F) as it is increased (lowered). This means that as the forward criterion is strengthened, the reverse criterion is also strengthened. If the initial forward and reverse criteria were both relatively weak (in order to ensure that at least one match was found), it might not make sense to report matches that satisfy a strong forward criterion, but only a weak reverse criterion.

We first consider the case where we are searching for the highest possible f_F. We introduce two new parameters, $p^\times$ and p^+, and tie f_R to f_F by ensuring that, at all times,

$$f_R = p^\times f_F + p^+ . \tag{16}$$

When we are evaluating a transformation t, we therefore compute $f[t]$ and $f'[t]$. If $f[t] \geq f_F$ and $f'[t] \geq f_R$, using the current values of f_F and f_R, then this transformation satisfies the current forward and reverse criteria, and may be used to strengthen these criteria. We therefore update f_F and f_R by setting

$$f_F = \min\left(f[t], \frac{f'[t] - p^+}{p^\times} \right)$$
$$f_R = \min(f'[t], p^\times f[t] + p^+) .$$

This ensures that t still satisfies the forward and reverse criteria with the new f_F and f_R, and that the constraint in Eq. (16) continues to hold. We can also modify the "find best, ranked by $d[t]$" search using similar update rules for τ_F and τ_R.

Incorporating the reverse criterion into the directed search method in this manner eliminates the need to specify a value for f_R (or τ_R); the search-for-best methods themselves remove the need to specify f_F (or, respectively, τ_F). However, this happens at the expense of having to specify two new parameters, $p^\times$ and p^+. In general, though, these new parameters are less variable than the old parameters: while the quality of the best match is variable, the relationship between the forward and reverse components of a single match is less so.

4.4. Parallelising

In this subsection, we discuss the possibility of parallelising the search strategies described in the previous subsection. The brute-force search described in Subsection 3.3 is clearly massively parallel: every transformation lying on the grid of transformations is evaluated, and the ones that satisfy the forward and reverse criteria are reported. However, unless an enormous number of processors are available, this will still be far too slow to be practical; we therefore consider how to parallelise the cell subdivision search method.

One way to parallelise this search method is to split the initial list of cells into groups, assign one group to each processor, and have the searches proceed independently. This approach suffers from load-balancing problems: some areas of the transformation space are easier to search than others (this, after all, is the whole premise of the cell subdivision technique), and so some processors will complete their search before others, possibly much before. Since it is not possible to *a priori* determine which areas of transformation space are easier or harder to search, static load-balancing does not work. However, dynamic load-balancing is not too difficult: whenever a processor completes its portion of the search, it sends out a broadcast to other processors, and collects some as-yet-unevaluated cells from each (or from the one with the most work remaining), and then proceeds with evaluating and subdividing these cells. The overhead of this communication is relatively low, as the time taken to subdivide and search a list of cells is much greater than the time taken to send the list over an inter-processor channel. We have implemented this method on a four processor shared-memory machine.

The directed search techniques from Subsection 4.3 can be incorporated into this framework. Each processor independently searches the region of transformation space that it has been assigned, evaluating and ranking cells. Suppose that only one transformation satisfying the forward and reverse criteria is desired. Once any processor finds such a transformation, it broadcasts that fact; the other processors can then terminate their search. The adaptive parameter update strategies of Subsection 4.3 can also be included in a similar fashion: a processor finding a match and subsequently updating its parameters broadcasts these updated parameters; processors receiving such a broadcast update their own parameters.

A simpler method takes advantage of the parallelism in evaluating the lists of cells: these computations are independent, and so can be done in parallel. Thus, the main flow of the search can proceed as in the single-processor case, except that it uses multiple processors to perform the cell evaluations. This uses less of the available parallelism, but is simpler to implement.

5. Implementation and Examples

5.1. Parameter Range Restrictions

One problem that arises in practice is that of degenerate, or near-degenerate, transformations. For example, a transformation of the model that greatly reduces its size matches just about any feature in the image, as everything in the transformed model is near the image feature, making the forward distance small, and everything in the image, near the transformed model's position, is close to something in the model, making the box-reverse distance small. The search has discovered that anything looks like a dot if it is far enough away; while this is correct, it is not useful. The transformed model could also become a straight line, with similar problems. To eliminate this problem, we allow the user to specify restrictions on the space of affine parameters, so as to avoid such degenerate or near-degenerate poses.

Equation (9) characterises an affine transformation t as a matrix A and a translation (t_x, t_y), having six parameters $(a_{00}, a_{01}, a_{10}, a_{11}, t_x, t_y)$. Here, these parameters are specified in the natural (continuous) basis, rather than the grid basis. We can restrict the range of the parameters individually, by supplying parameters $a_{00}^{\min}$, $a_{00}^{\max}$ and so on. In the following examples, unless specified otherwise, each of these a_{ij} parameters is restricted to lie between -1 and 1. This reflects the idea that the model bitmap should have been acquired at sufficient resolution to capture its full detail, and should be searched for at a smaller scale. Unless specified otherwise, we also restrict, *a priori*, the translational parameters: only translations where the top left corner of the transformed model lies inside the image bounds are allowed (though this restriction can be weakened if desired).

Restricting the a_{ij} parameters directly this does not directly address the problem of the model collapsing to a line or a point. Thus, we also consider the determinant of A, $d = a_{00}a_{11} - a_{01}a_{10}$. This value is the relative area of the transformed model: how much larger or smaller it is than the original model. We then limit this by insisting that every transformation under consideration must have $d^{\max} \geq d \geq d^{\min}$, for some parameters $d^{\max}$ and $d^{\min}$; this eliminates transformations that make the model too large or too small. We also insist that $d > 0$: transformed models whose relative area d is negative represent reflections of the model.

Consider the effect of the transformation t on a unit square. It performs the following mappings:

$$\begin{aligned}
(0,0) &\to (t_x, t_y)\\
(1,0) &\to (a_{00} + t_x, a_{10} + t_y)\\
(0,1) &\to (a_{01} + t_x, a_{11} + t_y)\\
(1,1) &\to (a_{00} + a_{01} + t_x, a_{10} + a_{11} + t_y)\,.
\end{aligned}$$

In other words, this unit square has been transformed into an arbitrary parallelogram. We can restrict the range of valid transformations by considering the parallelograms that they produce. For example, if the side lengths are too different, then we might want to reject the transformation, as the aspect ratio of the transformed model is too skewed. We can thus reject any transformations where

$$\max\left(\frac{\sqrt{a_{00}^2 + a_{10}^2}}{\sqrt{a_{01}^2 + a_{11}^2}}, \frac{\sqrt{a_{01}^2 + a_{11}^2}}{\sqrt{a_{00}^2 + a_{10}^2}}\right) > \alpha^{\max}$$

where $\alpha^{\max}$ is a parameter specifying the maximum aspect ratio skew.

A third possible transformation restriction relies on the observation that the dot product of the sides of this parallelogram (normalised for the scaling of the edges) determines how greatly shearing the transformation t is: when it is small, the parallelogram is still roughly a (possibly rotated) square. As we will see below, restricting this shear value can be useful. Thus, we also reject any transformations where

$$\frac{|a_{00}a_{01} + a_{10}a_{11}|}{\sqrt{a_{00}^2 + a_{10}^2}\sqrt{a_{01}^2 + a_{11}^2}} > s^{\max}.$$

As each cell is evaluated during the search, it is also checked to make sure that it contains some valid transformation; if every transformation inside the cell is eliminated by one or another of these restrictions, then the cell is rejected.

Once these parameters are in place, we can restrict the range of transformations in interesting ways. For example, if $s^{\max} = 0$ and $\alpha^{\max} = 1$, then the only valid transformations consist of a uniform scaling of the model, followed by a rotation and a translation. If we increase the transformation restriction by also setting $d^{\min} = d^{\max} = 1$, then we are essentially searching only the group of two-dimensional rigid motions (translations and rotations).

5.2. Examples

We now present some examples of the operation of this method. Figure 1 shows the first example (the "box" example). The parameters used were $\tau_F = \tau_R = \sqrt{5}$, $f_F = 1$, $f_R = 0.8$, $d^{\min} = 0.3$, $d^{\max} = 1$, $\alpha^{\max} = 2$, $s^{\max} = 0.4$. The wall-clock time taken to locate the model, on an eight-processor Sun SPARCServer 1000, was 7 : 33 (minutes : seconds). The speedup due to parallel processing (total CPU time divided by wall-clock time) was 6.99, indicating that there is more parallelism present than is being exploited, and that close-to-linear speedups would be expected from adding more processors. While this time is not as good as might be hoped for, it should be noted that the number of points

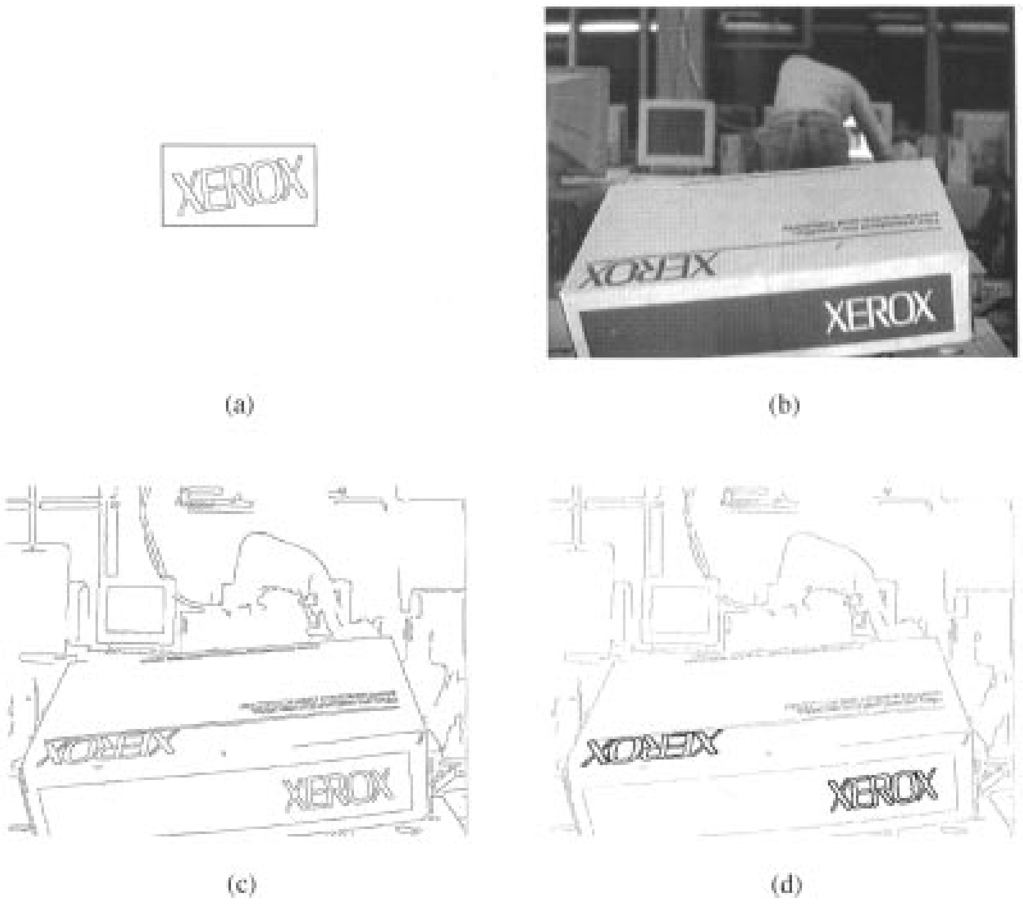

Figure 1. Locating a planar object (the "box" example). (a) The model. (b) The image in which the model is to be located. The image is 640 × 480 pixels. (c) The edges extracted from the image. (d) The located model overlaid on the image edges.

in the image and model are many times larger than can be handled by other methods that search under affine transformation: 14834 points and 1473 points, respectively. Also, the directed search modes could not be used, since there were two instances of the model to be located; using these modes would have increased the efficiency of the search.

Our second example (the "logo" example), shown in Fig. 2 shows how a partially occluded model can be located. The parameters used were $\tau_F = \tau_R = 1$, $f_F = 0.8$, $f_R = 0.6$, $a^{\min} = 0.7$, $a^{\max} = 1$, $\alpha^{\max} = 1.5$, $d^{\max} = 0.6$; the directed search for best fraction was also used, with parameters $p^\times = 0.75$, $p^+ = 0$. In other words, as better and better matches were found, and the forward fraction f_F was increased as a consequence, f_R was also increased so $\tau_R = 0.75\tau_F$ at all times during the search. The wall-clock time was 31 : 05 (minutes : seconds), and the speedup was 7.90. Since the model is nearly three-way rotationally symmetric, the search had to try many slight variations on the two wrong rotations, to see if they could be made to fit better than the right rotation, approximately tripling the time required.

We have also tested the performance of our method on smaller feature sets. Figure 3 shows one such test (the "book" example). The parameters used for this test were $\tau_F = \tau_R = 2\sqrt{2}$, $f_F = 1$, $f_R = 0.8$, $d^{\min} = 0.2$, $d^{\max} = 1$, $\alpha^{\max} = 3$, and $s^{\max} = 0.6$. The directed search for best distance was used. A number of transformations, all very similar and of the same quality, were found; one of these was picked arbitrarily and is shown in the figure. The computation took 3.69 seconds wall-clock time, and 8.13 seconds CPU time; this indicates that on such sparse sets, quite good times can be achieved.

Above, we claimed that by setting some of the range restriction parameters appropriately, we could restrict the search to the subgroup of rigid motions. Figure 4

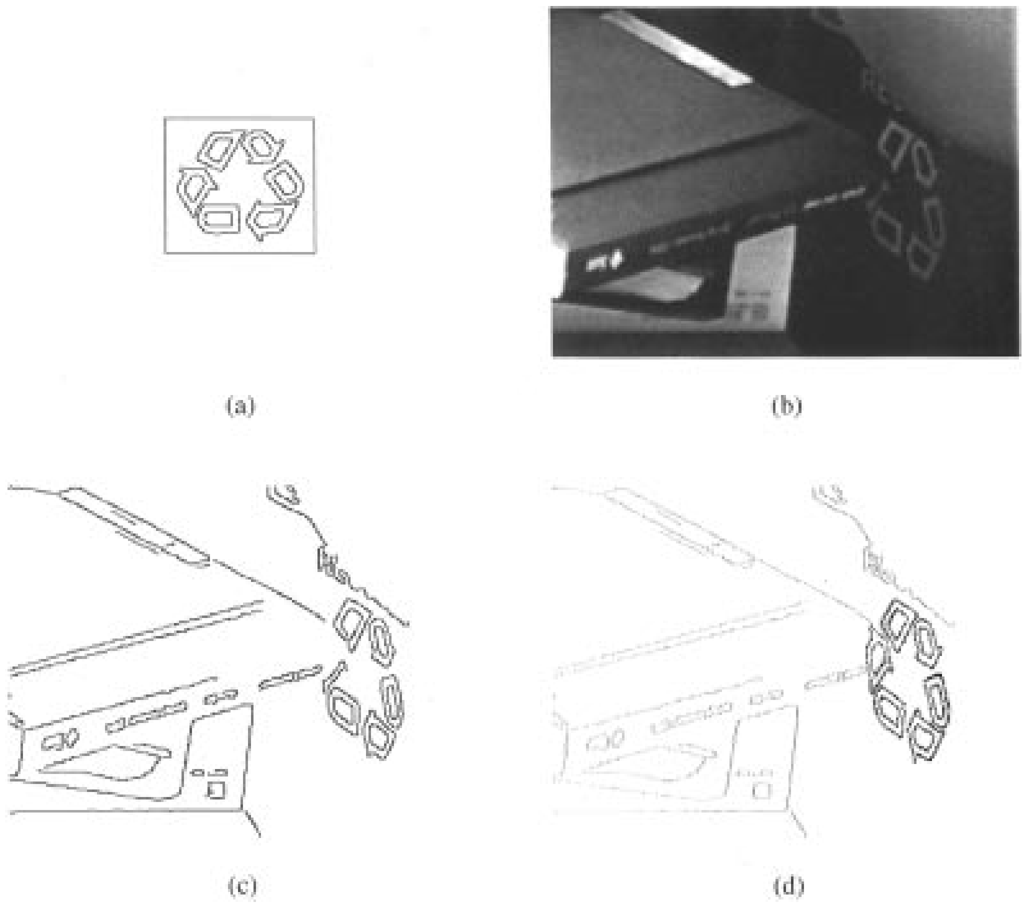

Figure 2. Locating a partially occluded object (the "logo" example). (a) The model. (b) The image in which the model is to be located. The image is 320 × 240 pixels. (c) The edges extracted from the image. (d) The located model overlaid on the image edges.

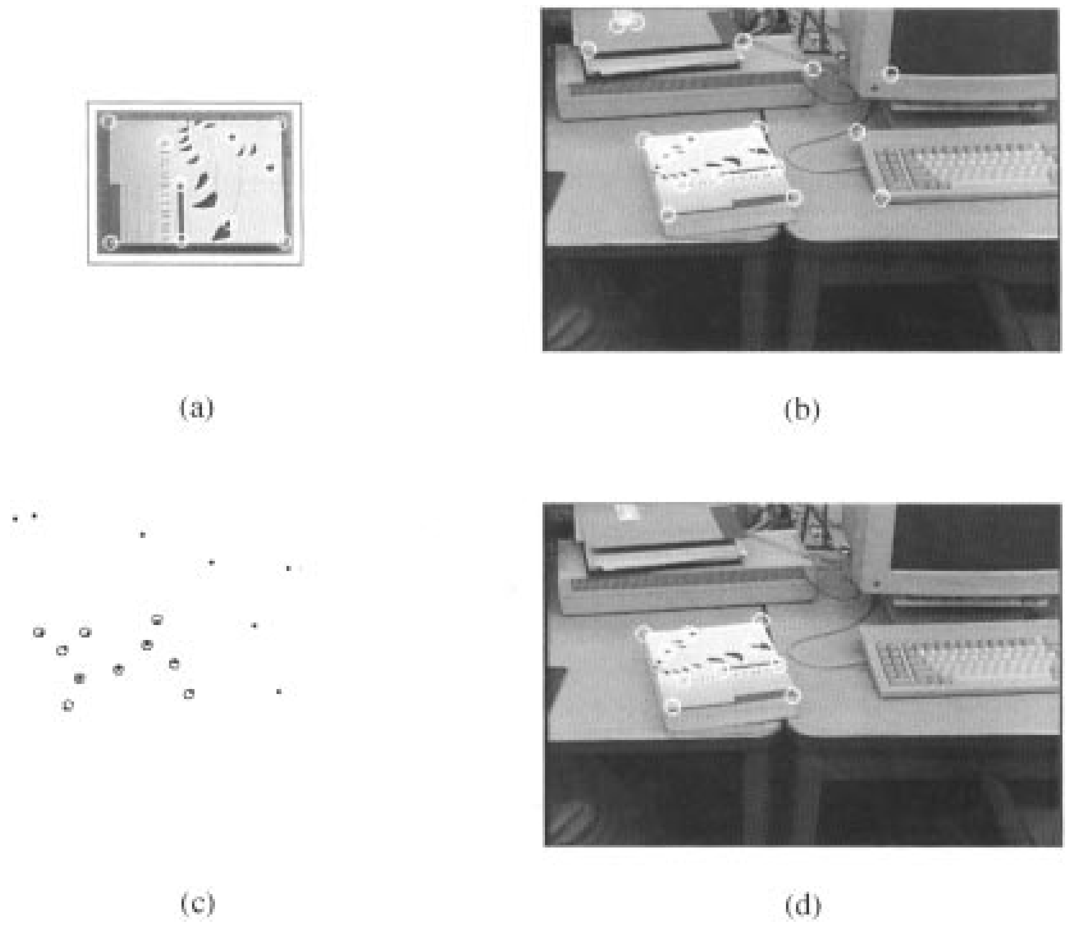

Figure 3. Locating a sparse model (the "book" example). (a) The model point set overlaid (as white circles) on the original model object. The features were extracted by hand from the grey-level image. (b) The image in which the model is to be located. The image features are overlaid (as white circles) on the original grey-level image. (c) The located model overlaid (as black circles) on the image point set (shown as black squares). (d) The located model overlaid (as white circles) on the original image.

presents an example of such a restricted search (the "blocks" example). We restricted the search by setting $d^{\min} = 0.98$, $d^{\max} = 1.02$, $\alpha^{\max} = 1.02$ and $s^{\max} = 0.02$ (these parameters should not be set to their exact limiting values because of the rasterisation of transformation space). We also set $\tau_F = \sqrt{5}$ and $f_F = 0.9$. The reverse distance was not used because of the markings on the faces of the blocks, which are not present in the model. Since the model is four-way rotationally symmetric, we also restricted the transformation range to include rotations between 0 and $\pi/2$ only. A number of matches were found; these were grouped into connected components and the best match from each component was reported. The time taken for this search was 2.72 seconds wall-clock, 8.15 seconds CPU time.

We have also used our system to perform whole-frame registration. Figure 5 shows this application (the "motion" example). We begin with two grey-level images, taken by the same camera a second or so apart. There is significant depth in the scene, the camera has moved between the frames, and there is a large object that has moved in the world between the two frames. A large number of edge points in each image have no corresponding point in the other image; the moving object also occludes a significant (and different) part of each image. We want to find the affine transformation that best aligns the two frames. Our image point set consists of the edges from the first frame; our model point set is formed by decimating (by 95%) the edges from the second frame.

We first find the pure translation of the model that brings it into best alignment with the first image. This is done using a special-case translation-only search engine (designed using the same set of principles as

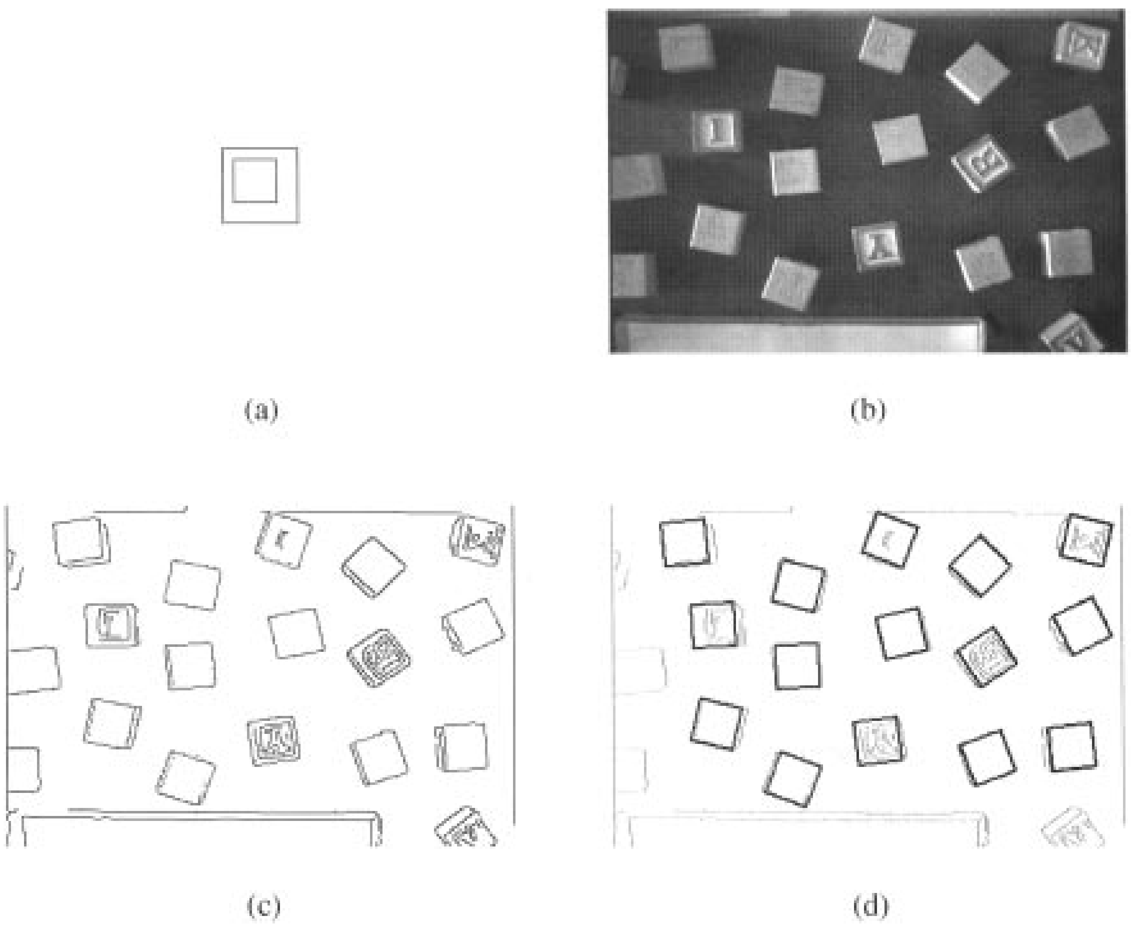

Figure 4. Locating an model undergoing rigid motion (the "blocks" example. (a) The model point set. (b) The image in which the model is to be located. The image is 360 × 240 pixels. (c) The edges extracted from the image. (d) The results of the search.

the more general search engine). This first step uses parameters $\tau_F = \sqrt{2}$, $f_F = 0.5$, and the directed search for best fraction. It searches a translational range of ±50 pixels in x and y. The usual restriction that the transformed model must lie inside the image bounds is removed for this search and for the subsequent affine search. The translation it finds, $t = (-41, 5)$, is shown in Fig. 5(e). The forward fraction $f[t]$ is about 0.65. In other words, when the model point set is translated by $(-41, 5)$, about 65% of its points land within $\sqrt{2}$ pixels of some point of the first image. The alignment is good in some places, but poor in others. This translational alignment takes 2.3 seconds.

Once we have this translation, we search for a slight affine distortion of it, in order to improve the match quality. We search a translational range of ±5 pixels about $(-41, 5)$; the other parameters are also bounded by

$$\begin{bmatrix} 1 \pm 0.03 & \pm 0.03 \\ \pm 0.03 & 1 \pm 0.03 \end{bmatrix}.$$

The range of transformations is further restricted by $d^{\min} = 0.95$, $d^{\max} = 1.05$, $\alpha^{\max} = 1.02$, $s^{\max} = 0.02$. Thus, we are searching only a small range of the affine transformation space around the identity transformation. We use $\tau_F = \sqrt{2}$, as before. We also use the directed search for best fraction as before, but with f_F initially set to 0.65 (since the translation found before is within the range being searched, and so we know that the best transformation must be at least as good as it). This search took 36 seconds wall-clock, 226 seconds CPU time.

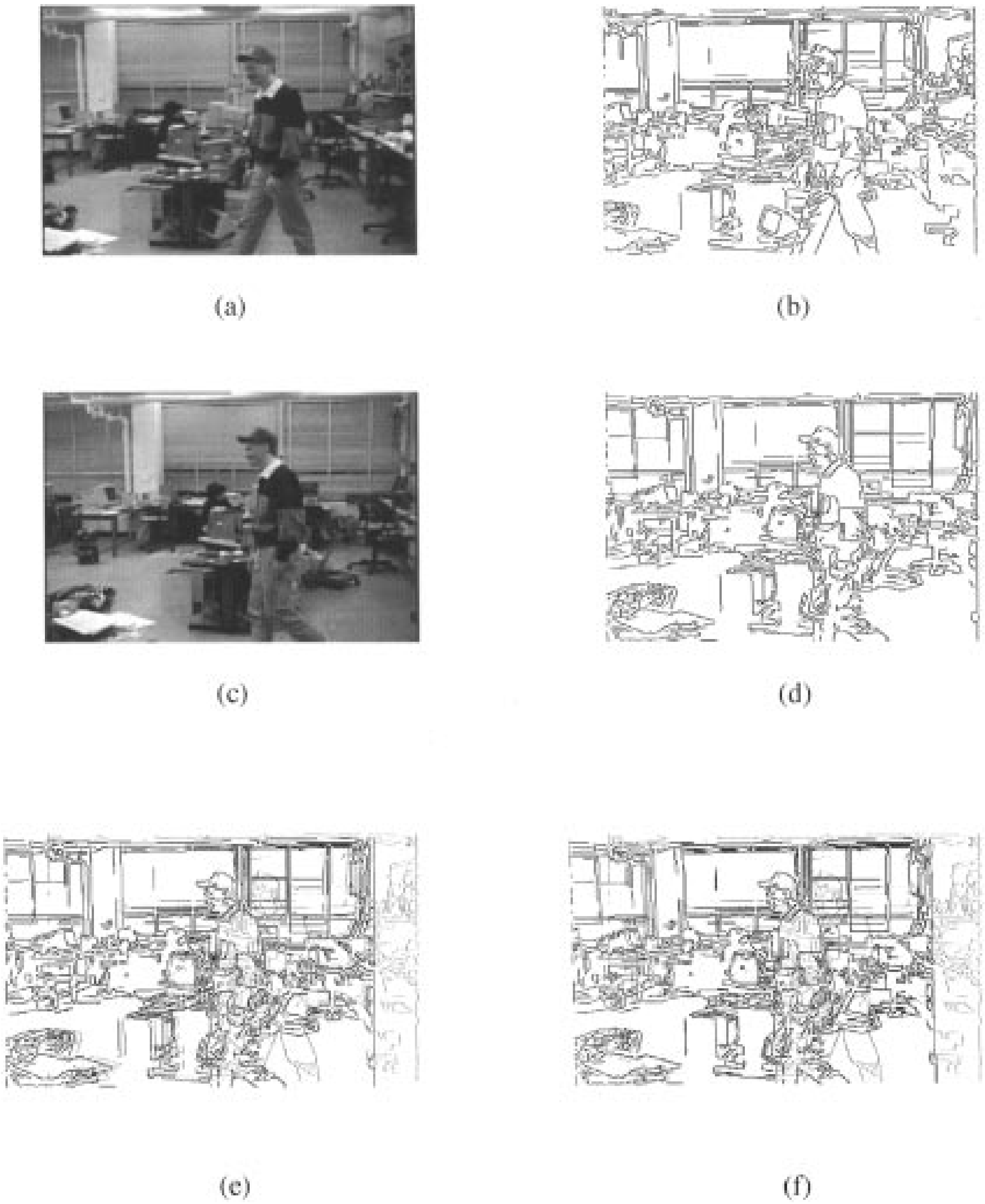

Figure 5. Compensating for camera motion (the "motion" example). (a) The first frame. (b) The edges extracted from the first frame. (c) The second frame. (d) The edges extracted from the second frame. (e) The translation best aligning the two frames. (f) The affine transformation best aligning the two frames.

Table 1. Some counts of the number of transformations involved in the examples.

	Transformations			Cells evaluated		Time (sec)	
Example	Within bounds	Inside borders	Satisfying restrictions	4×4 zoning	Without zoning	CPU	Wall
Box	1.3×10^{15}	3.5×10^{14}	4.9×10^{13}	1.0×10^{8}	3.9×10^{8}	3166.51	452.58
Logo	1.1×10^{14}	2.1×10^{13}	1.2×10^{12}	1.4×10^{8}	5.5×10^{8}	14725.78	1864.61
Book	2.8×10^{14}	4.4×10^{13}	1.2×10^{13}	5.6×10^{5}	2.6×10^{6}	8.13	3.69
Blocks	5.1×10^{11}	1.8×10^{11}	2.1×10^{6}	1.4×10^{5}	2.4×10^{5}	8.15	2.72
Motion	1.6×10^{7}	1.6×10^{7}	4.5×10^{6}	1.8×10^{6}	2.4×10^{6}	226.00	35.62

Finally, we ensure that this transformation is also a good match for the entire second edge image (not just the decimated subset we have been using). We search a very small region of transformation space around it (one grid step in each direction) and pick the best transformation. This takes only a second or two, dominated by overhead. The results of this search are shown in Fig. 5(f). The overall quality is significantly improved from the initial (purely translational) transformation. $f[t]$ for this transformation is 0.693 ($f[t]$ for the decimated model only is 0.725). The reverse fraction $f'[t]$ is 0.743.

5.3. *The Effects of Efficient Search*

In this subsection, we use these examples to quantify the effect of some of the algorithmic techniques presented in the previous sections.

For the "box" example, there are approximately 1.3×10^{15} affine transformations where the affine and translational parameters lie within the *a priori* bounds discussed in Subsection 5.1. Of these, approximately 3.5×10^{14} transform the model so that it lies within the image bitmap, and whose determinants are positive (they do not correspond to a reflection of the model bitmap). The additional scale, skew and shear restrictions leave about 4.9×10^{13} transformations still to be considered; thus, these parameter restrictions have eliminated about seven-eighths of the remaining transformations, including many degenerate or near-degenerate transformations. The number of cells actually evaluated by the search was approximately 1×10^{8}—over 99.999% of the remaining transformations were eliminated by the hierarchical search, without being explicitly considered. If no zoning had been used (if the model was considered to be a single zone), then 3.9×10^{8} cells would have been evaluated, a fourfold increase.

Table 1 gives similar numbers for the other examples. In each of the examples, the restriction that the transformed model must lie inside the image has eliminated a substantial fraction of the transformations; the exception is the "motion" example, where this restriction was not enforced. The restrictions on the overall transformation scale, skew and shear again reduced the number of transformations to be considered; in the "blocks" example, this reduction was especially pronounced, because of the restriction to essentially just rigid motions. The efficient search then only examines a small fraction of these. This is more effective in the cases where the transformations were less restricted. It is especially effective in the "book" example, where the image and model were both sparse; in general, sparser images lead to more effective pruning, as the thresholded box distance transforms are correspondingly less dense. The next column in the table shows how many transformations would have been evaluated if zoning had not been used. Increasing the number of zones above 4×4 reduces the number of transformations examined somewhat, but generally not enough to balance out the increased overhead and memory use. Finally, the last two columns summarise the CPU and wall-clock times required for these searches.

Both the "logo" and "motion" examples used the directed search for best fraction. If this had not been used, then the "logo" example would have examined 2.8×10^{8} transformations, approximately doubling its time, and the "motion" example would have examined 3.2×10^{6} transformations, again nearly double.

6. Conclusion

We have presented a method for determining affine transformations that bring a model into close correspondence with a portion of an image (both represented as finite point sets in the plane). This close correspondence is determined via modified versions of the Hausdorff distance. This mismatch measure is reliable in the presence of occlusion, spurious points, positional uncertainty, and the presence of multiple objects. We have discretised the space of affine transformations, in such a manner that the error introduced into the modified Hausdorff distance as a function of transformation is small, and so that the grid quantisation is tied to the model and image quantisation. We have introduced a hierarchical search method that is guaranteed to produce the same results as an exhaustive search, but that does not explicitly consider every transformation on the grid, and does not require excessive bookkeeping. The search uses a set of box distance transforms derived from the image in order to perform the hierarchical pruning. It also divides the model into zones, as this exploits the non-uniform manner in which the model points move as the transformation varies. The search techniques can be modified to locate the best transformation quickly, and can be performed in parallel. We have shown how we can add parameters to the search in order to restrict it to reasonable transformations, and also to restrict it to interesting subgroups of the group of affine transformations. The search techniques can also be applied to other transformation groups.

We have presented a number of examples of the operation of this method, showing that it is accurate, locating the object or object correctly in each case. With small numbers of image and model points, or with strong restrictions on the transformation range, the time taken is quite low; with large numbers of points and looser restrictions, the times are higher, but other current methods would not be able to handle such large sets. We believe that it is possible to develop further efficient search techniques that will reduce these times considerably.

References

Ayache, N. and Faugeras, O. 1986. HYPER: A new approach for the recognition and positioning of two-dimensional objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(1):44–54.

Barrow, H.G., Tenenbaum, J.M., Bolles, R.C., and Wolf, H.C. 1977. Parametric correspondence and chamfer matching: Two new techniques for image matching. In *Proc. Fifth International Joint Conference on Artificial Intelligence*, Cambridge, MA, pp. 659–663.

Borgefors, G. 1988. Hierarchical chamfer matching: A parametric edge matching algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(6):849–865.

Breu, H., Gil, J., Kirkpatrick, D., and Werman, M. 1995. Linear time Euclidean distance transform algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(5):529–533.

Bruel, T.M. 1992. Fast recognition using adaptive subdivision of transformation space. In *Proc. Computer Vision and Pattern Recognition*, Champaign-Urbana, Illinois, pp. 445–451.

Cass, T.A. 1990. Feature matching for object localization in the presence of uncertainty. In *Proc. Third International Conference on Computer Vision*, Osaka, Japan, pp. 360–364.

Danielsson, P.E. 1980. Euclidean distance mapping. *Computer Graphics and Image Processing*, 14:227–248.

Huttenlocher, D.P. and Ullman, S. 1990. Recognizing solid objects by alignment with an image. *International Journal of Computer Vision*, 5(2):195–212.

Huttenlocher, D.P. and Rucklidge, W.J. 1993. A multi-resolution technique for comparing images using the Hausdorff distance. In *Proc. Computer Vision and Pattern Recognition*, New York, NY, pp. 705–706.

Huttenlocher, D.P., Klanderman, G.A., and Rucklidge, W.J. 1993. Comparing images using the Hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):850–863.

Olson, C.F. 1994. Time and space efficient pose clustering. In *Proc. Computer Vision and Pattern Recognition*, Seattle, Washington, pp. 251–258.

Paglieroni, D.W. 1992. Distance transforms: Properties and machine vision applications. *Computer Vision, Graphics and Image Proc.: Graphical Models and Image Processing*, 54(1):56–74.

Paglieroni, D.W., Ford, G.E., and Tsujimoto, E.M. 1994. The position-orientation masking approach to parametric search for template matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(7):740–747.

Rucklidge, W.J. 1995a. *Efficient Computation of the Minimum Hausdorff Distance for Visual Recognition*. Ph.D. thesis, Cornell University.

Rucklidge, W.J. 1995b. Locating objects using the Hausdorff distance. In *Proc. Fifth International Conference on Computer Vision*, Cambridge, MA, pp. 457–464.