# Tracking Non-Rigid Objects in Complex Scenes*

Daniel P. Huttenlocher
Jae J. Noh
William J. Rucklidge

TR 92-1320
December 1992

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

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

# Tracking Non-Rigid Objects in Complex Scenes *

Daniel P. Huttenlocher, Jae J. Noh
and William J. Rucklidge

Computer Science Department
Cornell University
Ithaca NY 14853 USA

**Abstract**

We consider the problem of tracking non-rigid objects moving in a complex scene. We describe a model-based tracking method, in which two-dimensional geometric models are used to localize an object in each frame of an image sequence. The basic idea is to decompose the image of a solid object moving in space into two components: a two-dimensional motion and a two-dimensional shape change. The motion component is factored out, and the shape change is represented by explicitly storing a sequence of two-dimensional models, one corresponding to each image frame. The major assumption underlying the method is that the two-dimensional shape of an object will change slowly from one frame to the next. There is no assumption, however, that the two-dimensional image motion in successive frames will be small. Thus the method can track objects that move arbitrarily far in the image from one frame to the next.

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

# 1 Introduction

One of the most difficult problems for a dynamic vision system is tracking non-rigid objects moving in a cluttered environment. This problem is important for processes such as shape reconstruction, recognition, navigation and manipulation, which rely on the ability to efficiently and accurately select objects from an image (i.e., to solve the figure-ground problem). The problem of object tracking has received a good deal of attention in the computer vision research community over the last several years (e.g., [3, 4, 9, 10, 11, 12]). Many methods, however, do not address the problem of tracking non-rigid objects in complex scenes, particularly when there are other moving objects present.

The recent work of [13] is most similar to ours in its ability to track non-rigid objects in natural scenes. Our approach differs from theirs in that we use a model-based method, whereas they use a method based on bottom-up measurement of local differential image properties. Because a model-based approach is able to exploit global attributes of the object being tracked, it can provide significant advantages over purely local methods for situations in which the environment is cluttered, there are multiple moving objects, or there may be a large motion of an object from one frame to the next. A number of other researchers have taken a model-based approach to motion tracking (e.g., [8, 9, 10, 11]). What characterizes our approach is that there is no constraint on where in the image an object may have moved from one frame to the next, and there may be multiple moving objects in the scene. Moreover, the model of an object is acquired dynamically from the image sequence, rather than being provided a priori.

The central observation underlying our method is the fact that the two-dimensional image of an object moving in three-space can be decomposed into two parts

- a two-dimensional shape change, corresponding to a different aspect of the object becoming visible, or an actual change of shape in the object, and
- a two-dimensional motion (of some restricted class) in the image, corresponding to the motion of the visible aspect of the object.

We explicitly represent an object in terms of its two-dimensional image shape at each point in time, and store a set of two-dimensional geometric models that capture the evolution of this shape across time. Our definition of two-dimensional image 'shape' is made more precise below, but intuitively it is the change in the image of an object that cannot be accounted for by a two-dimensional motion.

The main condition imposed by our method is that the two-dimensional shape of an object not change greatly between two successive frames of an image sequence. In particular, there is no assumption that the two-dimensional motion from one frame to the next be small. The method can track objects that move anywhere in the image between two successive frames. This is a much less restrictive assumption regarding the nature of the motion than is made by local differential methods. Such methods must assume that the entire change in the object from one image to the next is small (local), whereas we only assume that the *shape* change is small, and not the motion.

In our method an object is represented as a sequence of binary images, one corresponding to each frame of the input image sequence. Each frame of the model specifies a set of pixels in a given sub-region of the corresponding image frame (i.e., the model at time $t$ is a subset of the image features at time $t$). These pixels constitute the appearance, or shape, of the object being tracked at that frame. All of the frames of the model are represented in a canonical coordinate frame, with a mapping from each model frame to the corresponding image frame. Thus a model evolves from one time step to the next, capturing the changes in the image shape of an object as it moves. In the current implementation, the binary images are based on intensity edges extracted from the images. However, any other means of deriving binary features from the image (or even multiple different sets of binary features) could easily be used in conjunction with the technique.

The basic tracking method operates by comparing the model at a given frame, $M_t$, to the image at the next frame, $I_{t+1}$, in order to find the transformation specifying the best location of that model in that image (where the model and image are binary images derived by some feature extraction mechanism). Then a new model, $M_{t+1}$, is formed by selecting the subset of $I_{t+1}$ that is 'near' the transformed model $M_t$. This new model, $M_{t+1}$, represents the shape of the object at the next time frame. The shape change from one time frame to the next is required to be small, because $M_{t+1}$ is constructed from parts of the image that are 'near' the transformed $M_t$ (this will be made more precise in the following section). The method of tracking an object is thus based entirely on comparing two-dimensional geometric structures, as represented by binary image models, between successive frames. There is no computation of local differential image quantities such as the optical flow or motion field.

Our experiments with the method find that it successfully tracks non-rigid objects (such as people walking), even in relatively cluttered scenes, with other moving objects, poor lighting conditions, specular reflections, camera panning, and large displacements in the image (e.g., the object moving from one side of the image to the other in two successive frames). The current implementation of the method requires about 5 seconds per frame for a half-resolution NTSC video image ($320 \times 240$ pixels) on a SPARCstation-2 (not including edge detection, which is about another second per frame).

In order to illustrate the method, before explaining its operation in any detail, Figure 1 shows the images and the corresponding models for six selected frames from a 100 frame sequence (frames number 1,20,40,60,80 and 100). Each row of the figure contains an image frame ($320 \times 240$ pixels), its intensity edges, and the model extracted for that frame. Recall that the model is a subset of the corresponding image frame. Each model is shown with a bounding box; note that the size of the models changes dynamically (as will be explained in Section 4). The third row of the figure illustrates a situation where the object was changing shape very rapidly from one frame to the next, so while the object is still being tracked successfully, the representation of the two-dimensional shape is relatively degraded.

Figure 2 shows six successive frames from the same image sequence (numbers 14-19). This illustrates the degree of change from one frame to the next that can be handled by the method. Despite the fact that the object changes two-dimensional shape substantially in successive frames, and even more so during the entire sequence, it is tracked successfully through all 100 frames. (Note that the models shown in the figures are the result of tracking the entire sequence of 100 frames.) As further examples will illustrate below, the method can track objects such as this even when there are other similar-shaped objects moving in the scene.

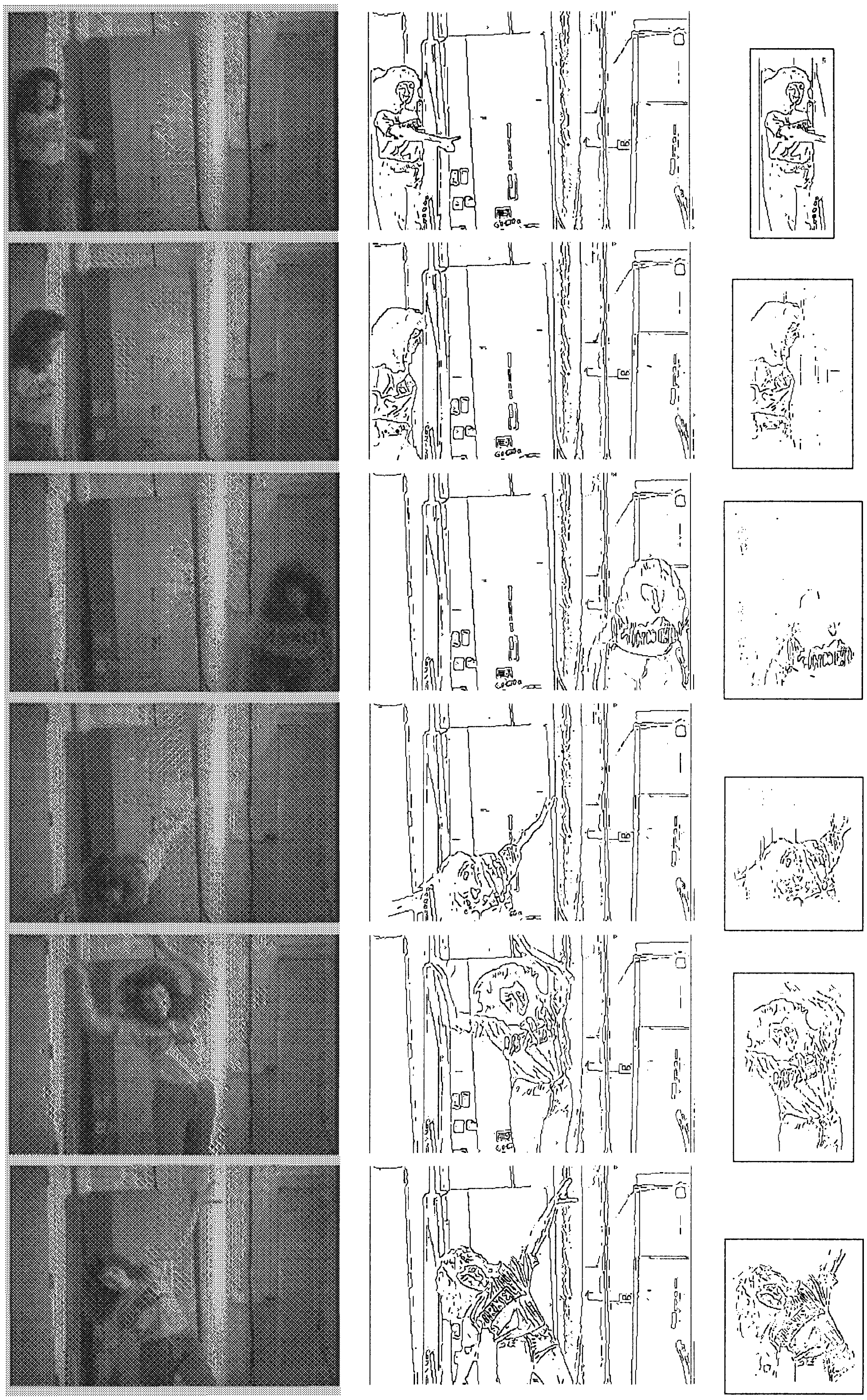

Figure 1: Six selected frames (numbers 1,20,40,60,80 and 100) from a 100 frame motion sequence, and the corresponding models of the object being tracked. Each model is a subset of the edge pixels from that image frame.

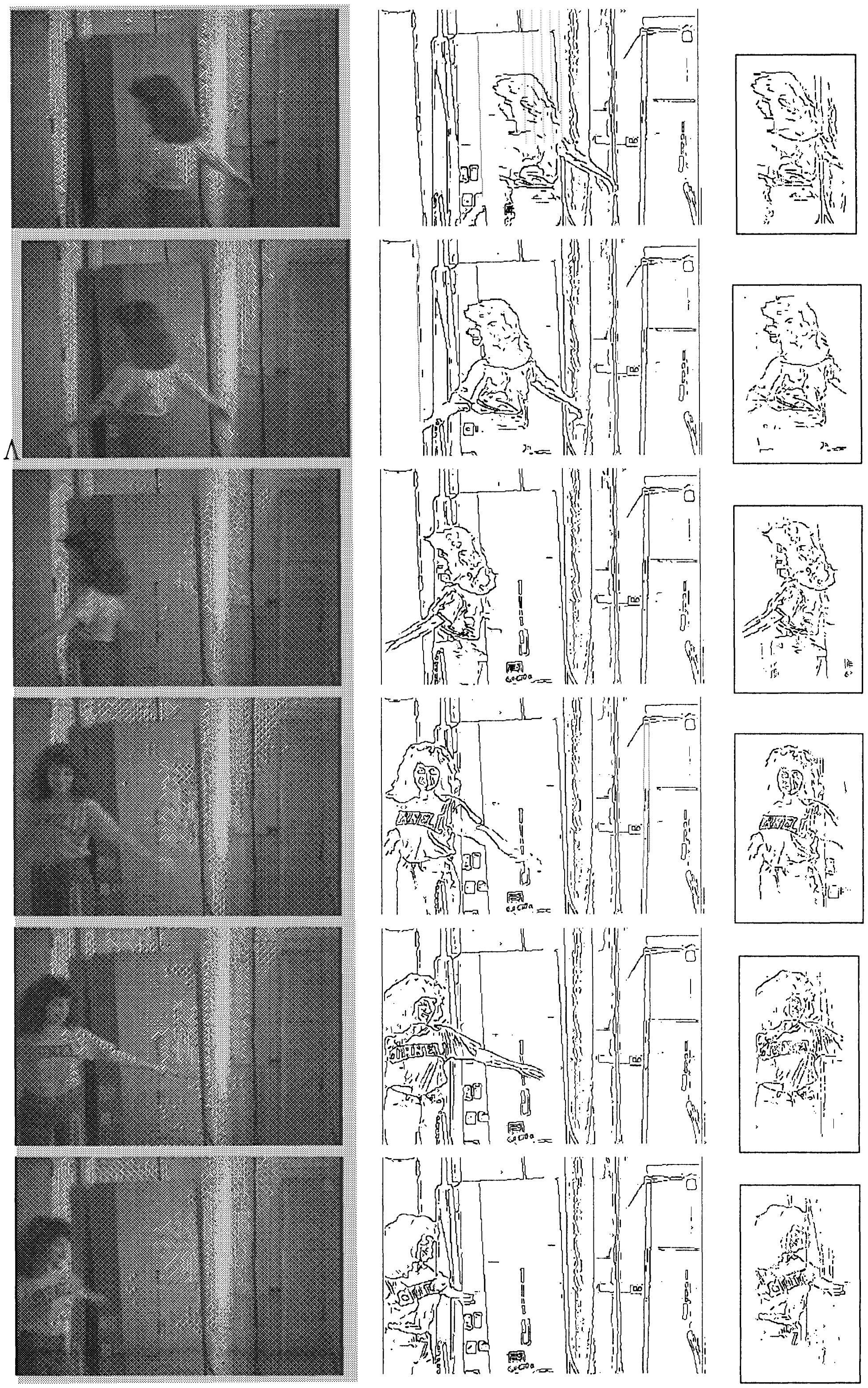

Figure 2: Six successive frames (numbers 14-19) from the same 100 frame motion sequence, and the corresponding models of the object being tracked.

The major aspects of our approach are:

1. Decomposing the image of a moving 3D object into two parts: a 2D motion and a 2D shape change. The shape change is assumed to be relatively small from one frame to the next, but the motion can be arbitrarily large (i.e., there is no 'local search window' in the image).

2. Capturing the 2D shape change between successive images with 2D geometric models that evolve across time. The models provide global geometric constraints for tracking an object.

3. Fast 2D model matching using the minimum Hausdorff distance (a min-max-min distance which is described below). This distance measures proximity without computing an explicit correspondence.

In the next section we make precise the notion of a 'small' change in two-dimensional shape, and define the measure that we use to decompose the change in the 2D image of an object into a 2D motion and a 2D shape change. In Section 3 we then describe how to use this notion of shape change to do tracking. In Section 4 we describe an implementation of the tracking method, and in Section 5 we discuss extensions of the method to handle multiple moving objects and objects that can disappear from view.

# 2 Comparing 2D shapes

We define a shape to be a two-dimensional geometric object which may be transformed by the elements of some given group of transformations. For example, a square oriented horizontally and a square oriented diagonally could be considered to be the same shape or they could not, depending on the allowable transformations. If the shapes are allowed only to translate, then they are different: one is a square and the other is a diamond. If they are allowed to undergo Euclidean motion (translation and rotation) then the two shapes are the same, since one can be transformed into the other. More formally, given two geometric objects $A$ and $B$, we say that they have the same shape exactly when there exists some $g \in G$ such that $g(A) = B$, where $G$ is the allowable group of transformations.

Having defined a shape as a geometric object under the action of some transformation group, we now need to measure the difference (or change) between two shapes. Clearly if there is some $g \in G$ such that $g(A) = B$, then the difference should be zero — the shapes are identical. What about shapes that are not identical? In order to measure shape differences we use the minimum Hausdorff distance, under the action of the transformation group $G$. The Hausdorff distance is a max-min distance for comparing sets. In the form that we use it here, we limit ourselves to finite point sets (although the measure defines a distance metric for any closed, bounded sets).

For two point sets $P$ and $Q$ the Hausdorff distance between the sets is defined as

$$H(P,Q) = \max(h(P,Q), h(Q,P)) \tag{1}$$

where

$$h(P,Q) = \max_{p \in P} \min_{q \in Q} \|p - q\|, \tag{2}$$

and $\|\cdot\|$ is some norm for measuring the distance between two points $p$ and $q$ (which we will take to be the $L_2$ norm, or Euclidean distance). Thus this distance measures the degree to which each point of $P$ is near some point of $Q$ and vice versa. The distance is small when every point of one set is near some point of the other and vice versa. In contrast with most ways of measuring the distance between two sets of points, the Hausdorff distance is not based on a matching, or correspondence, of the points in one set with points in the other. The distance simply measures the proximity of points in the two sets.

The Hausdorff distance measures the difference between fixed point sets, whereas we are interested in measuring the difference between *shapes* of point sets. Given our definition of a shape as a point set modulo the action of some transformation group $G$, the natural definition of a distance is simply the minimum with respect to that group action,

$$D_G(P,Q) = \min_{g \in G} H(g(P), Q) \ . \tag{3}$$

In other words, the distance between two shapes is the minimum difference between them under all possible transformations of one shape with respect to the other. This measure obeys the metric properties (identity, symmetry and triangle inequality) as long as the distance function $\|\cdot\|$ is invariant with respect to the group $G$ (i.e. when for all $g \in G$ and any points $x, y$, we have $\|x - y\| = \|g(x) - g(y)\|$). This is true for the group of rigid motions when used with the $L_2$ norm, and the group of translations when used with any norm (see [5, 6]). The distance, $D_G(P,Q)$, is zero when two shapes are the same, and gets larger the more 'different' the shapes. Intuitively, this measure of the difference between two shapes depends on the most mismatched point of one object with respect to the other, and vice versa. There must be a transformation in $G$ that brings all of one object near some part of the other object, and vice versa, in order for the distance to be small.

In practice, one problem with the Hausdorff distance as defined in equations (1) and (2) is that a single outlying point will cause the distance to be large. Sensing errors and other sources of noise can easily cause one (or several) outliers. To reduce the effect of outliers, we instead use a rank order 'distance', which replaces the maximization operation in equation (2) with a rank operation (i.e., selection of the median value or some other quantile). The median or other quantile is a more robust measure than the maximum, as is commonly known in statistics. This rank order, or 'partial distance' is defined as,

$$h_K(P,Q) = \underset{p \in P}{K^{\text{th}}} \min_{q \in Q} \|p - q\|, \tag{4}$$

where ${K^{\text{th}}}_{p \in P} f(p)$ denotes the $K$-th ranked value of $f(p)$ over the set $P$. That is, if we consider the points in $P$ to be in sequence ordered by their values $f(p_1) \leq \ldots \leq f(p_n)$, the $K$-th element in this sequence, $f(p_K)$, is the $K$-th ranked value. For example, the $n$-*th* ranked value is the maximum (the largest element in the sequence), and the $n/2$-*th* ranked value is the median.

This partial distance, $h_K(P,Q)$, identifies the subset of $P$ of size $K$ which has the smallest directed Hausdorff distance to the set $Q$. Intuitively, $h_K(P,Q) = d$ when there is some subset of $P$ of size at least $K$ such that each point in this subset is within distance $d$ of some point in $Q$. This definition allows for a portion of $P$ to not correspond to anything in $Q$ (as occurs, for example, when an object $P$ is partly occluded from view in some image $Q$). For instance, if we compute $h_K(P,Q)$ with $K = \lfloor .75n \rfloor$ (i.e., using the 75-th percentile distance), then up to 25% of the points of $P$ need not be near any points of $Q$ (see [7] for more details on this rank order distance).

One of the interesting properties of the Hausdorff distance, and the partial 'distance', is the asymmetry inherent in the computation. The fact that every point of $P$ (or some given-sized subset of $P$) is near some point of $Q$ says nothing about whether every point of $Q$ (or some given-sized subset) is near some point of $P$. In other words, $h_K(P,Q)$ and $h_L(Q,P)$ can attain very different values. The maximum of these two values defines the partial Hausdorff distance, $H_{K,L}(P,Q)$. As mentioned above, this measure differs from most methods that are used to compare shapes in computer vision (from correlation to model-based matching) which are based on finding a correspondence of points in two sets.

# 3 Tracking using the Hausdorff distance

With the above-mentioned asymmetry of the Hausdorff distance in mind, let us now return to the tracking problem. The idea is to exploit the asymmetry in the distance to perform two different functions: (i) finding where the model at a given time, $M_t$, moved to in the image at the next time, $I_{t+1}$, and (ii) computing the new model, $M_{t+1}$, from $M_t$ and $I_{t+1}$. For the remainder of the paper we will assume that the model $M_t$ has $m$ points (i.e., in the binary image representation of $M_t$ there are $m$ nonzero pixels).

## 3.1 Locating the object in a new image

The directed rank order 'distance' from the model $M_t$ to the image $I_{t+1}$, which we will refer to as the *forward* distance, measures the degree to which some portion of the model resembles the image. The minimum value of this distance identifies the best position of $M_t$ in $I_{t+1}$, under the action of some group $G$. This minimum value of the forward distance is given by

$$d = \min_{g \in G} h_K(g(M_t), I_{t+1}) = \min_{g \in G} \operatorname*{K^{th}}_{p \in M_t} \min_{q \in I_{t+1}} \|g(p) - q\|, \tag{5}$$

which identifies the transformation $g^* \in G$ of $M_t$ that minimizes (4). That is, at least $K$ of the $m$ points of $g^*(M_t)$ are all within distance $d$ of some point of $I_{t+1}$, and this is the minimum such distance (there is no $g \in G$ for which at least $K$ points of $g(M_t)$ are less than $d$ from some image point).

Intuitively, the forward distance identifies the best 'position', $g^*$, of $M_t$ in the image $I_{t+1}$ (the position bringing at least $K$ of the points of $g^*(M_t)$ closest to points in the image $I_{t+1}$). Note that this measure imposes no requirement on points of the image $I_{t+1}$ being near those of $g^*(M_t)$, just on points of $g^*(M_t)$ being near those of $I_{t+1}$. Thus it works fine when the model $M_t$ is just some small subpart of a larger image. Figure 3 illustrates the forward distance computation, where the transformation group $G$ is the group of translations. The figure shows a model, an image, and the translation of the model that minimizes the partial

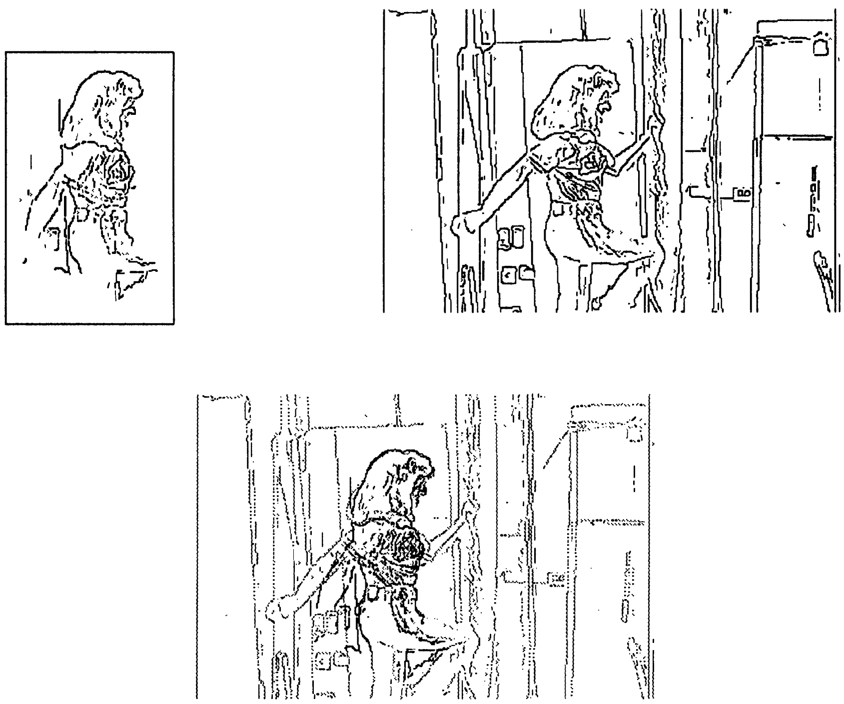

Figure 3: Using the forward distance to locate a model in an image.

directed Hausdorff distance for $K = \lfloor .8m \rfloor$ (i.e., 80% of the model points must lie near image points). In this case the distance is $\sqrt{2}$, meaning that at least 80% of the model pixels lie within one pixel (diagonally) of some image pixel. Note that the appearance of the object in the image is quite different from the model, as illustrated by superimposing the model on the image.

## 3.2 Updating the object model

Having used equation (5) to identify the best location $g^*$ of the model $M_t$ in the subsequent image frame $I_{t+1}$, it now remains to build $M_{t+1}$ by determining which pixels of $I_{t+1}$ are part of the new model. We are interested in finding some subset of the image, $I_{t+1}$ that 'agrees well' with the transformed model $g^*(M_t)$. We do this by using the distance from each point of $I_{t+1}$ to the nearest point of $g^*(M_t)$ as a criterion for selecting the subset of image points that belong to $M_{t+1}$. That is we define,

$$M_{t+1} = \{q \in I_{t+1} | \min_{p \in M_t} \|g^*(p) - q\| \leq \delta\}$$

for some distance $\delta$. In other words, $M_{t+1}$ is all those points of the image that are within distance $\delta$ of some point of $g^*(M_t)$.

The choice of the distance $\delta$ controls the degree to which the method is able to track objects that change shape. For instance, if $\delta = 0$ then only those pixels of $I_{t+1}$ that are directly superimposed on $g^*(M_t)$ will be included in $M_{t+1}$, and thus the method will not track shape changes at all. Any changes from one frame to the next that are not captured by the group of motions, $G$, will cause parts of the model to be lost. In practice, setting $\delta = 0$ will cause the tracker to lose an object after several frames even if the object does not actually change, due to noise and uncertainty in the locations of pixels. The larger the value of $\delta$, the more that the tracker is able to follow non-rigid motion, and to 'pick up' new parts of an object that have come into view since the previous frame. The parameter $\delta$ should be thought of as reflecting how much of the change from one frame to the next will be incorporated into the model (versus being left in the background). Thus, as $\delta$ becomes larger the ability to track non-rigid motion increases, at the cost of also possibly 'tracking' the background.

With some additional processing, it is often possible to compensate for the fact that larger values of $\delta$ result in the background being incorporated into the model. The idea is to take advantage of the fact that when an object moved from one frame to the next, often the background did not move; the technique that we use is described fully in Subsection 5.1.

If an object has several components which are moving about joints, and a central component to which they all are joined which is moving rigidly, the method will tend to track the central component and 'lose sight of' the peripheral ones, since it will not be able to match them reliably from one frame to the next when they exceed the shape change tolerance $\delta$. This is often the case for tracking people walking: the body and head change shape very little from one frame to the next, and so are included in the models $M_t$; the arms and legs move rapidly compared with the overall motion, and so even if an arm, say, is in $M_t$, it is not likely to be included in $M_{t+1}$ since it will have moved more than $\delta$ pixels. However, if an arm is not initially in $M_t$ but is held stationary (or moves only slowly) relative to the body, it will gradually be reacquired over the next few frames (at a rate of about $\delta$ pixels per frame). This phenomenon can be seen in the models shown in Figures 1 and 2.

The models in Figures 1 and 2 were generated by running the tracker with $K$ set to be 80% of the model pixels for the forward match (i.e., $K = \lfloor .8m \rfloor$), and with a $\delta$ of 8 pixels for constructing the models. The group of transformations, $G$, was the translation group. These same parameters were used in all the examples shown in the paper, and in many other sequences which we have analyzed with this tracker. Each frame of the sequence is $320 \times 240$ pixels, and the total running time for the 100 frame sequence was about 7 minutes on a SPARCstation-2. In the following section we describe the tracker in more detail. We then consider some other techniques that are used to enhance the basic method. At a high level, however, the method is simple:

- Use the minimum directed partial Hausdorff distance from the model to the image to find where an object moved to. This tracks the two-dimensional motion of the object.

- Use the distance from the image to the transformed model as a criterion to select a subset of the image pixels that form part of the next model. This tracks the two-dimensional shape change of the object.

# 4 An implementation of the tracker

We begin by describing how the basic tracking method described above is implemented. We then discuss modifications that allow for highly cluttered scenes, for scenes containing multiple moving objects with similar shapes, and for objects that change shape quickly or disappear and then reappear.

Recall that the basic observation underlying the method is to decompose the image of an object moving in space into a two-dimensional motion and a two-dimensional shape change. In the current system, we use the group of translations as the group of allowable two-dimensional motions. That is, any change in the image of an object other than a translation is encoded as a change in the two-dimensional shape of that object (where a change in shape is simply a change in the model from one frame to the next). The translations must be integer values such that the model $M_t$ is positioned overlapping with the image $I_{t+1}$ (this is not in practice a restriction since both the model and image are binary arrays).

Allowing only translational motion means that rotation or scaling in the image are treated as changes in shape. While it is possible to allow larger groups of image motions (e.g., rotation and scale in addition to translation) there is a significant computational cost to doing so. It is also interesting to note that there is some evidence that the human visual system similarly treats translational image motion specially. For example, apparent motion phenomena (a perceived motion when an object moves from one position to another in a pair of images) occur primarily for translational motion, and not for rotation or change of scale.

Each frame of the image sequence is initially processed in order to extract a binary feature map. Currently we use an edge operator similar to that of [1]. Denote the image edges at the current time by $I_t$, and denote the model at the current time by $M_t$. This model is also a binary edge map, and is always a subset of the edge pixels of the image, $I_t$. In the general case, the model $M_{t+1}$ is derived from $M_t$ and $I_{t+1}$ as outlined in Section 3. Of course the initial frame of a sequence must be treated specially, as is described below.

## 4.1 Finding the model's new location

Possible locations of the model $M_t$ are identified in the image at the next time frame, $I_{t+1}$, by finding the translation giving the minimum forward distance as in (5). However, rather than computing simply the single translation giving the minimum distance, we identify the set of translations of $M_t$, call it $X$, such that the partial directed Hausdorff distance is no larger than some value $\tau$.

$$X = \{x | h_K(M_t \oplus x, I_{t+1}) \leq \tau\}, \tag{6}$$

where $\oplus$ is the Minkowski sum notation, $A \oplus x = \{a + x | a \in A\}$ for any set $A$ and vector $x$. Intuitively, $X$ is the set of all translations of $M_t$ such that at least $K$ of the $m$ points (nonzero pixels) of $M_t \oplus x$ are within distance $\tau$ of some nonzero pixel of $I_{t+1}$. The value $\tau$ is updated dynamically; it will always lie in the range $\sqrt{2} \leq \tau \leq \tau_{\max}$, where $\tau_{\max}$ is a parameter of the algorithm. We describe this updating of $\tau$ in Subsection 5.2.

There are two reasons for finding all the translations where the distance is small, rather than the single best translation. First, there may be multiple translations that are essentially the same quality, in which case simply finding the best translation would not allow us to detect the presence of multiple matches. Second, there are efficient methods for finding all the translations such that the Hausdorff distance is less than some threshold. We describe these methods briefly here; see [7] for a full presentation.

The basic method for finding the translations such that the partial directed Hausdorff distance is less than $\tau$ (i.e., computing the set $X$) is as follows. First compute the distance transform $D_{t+1}$ of the image, $I_{t+1}$. $D_{t+1}$ is the array specifying for each location in the image the distance to the nearest nonzero pixel of $I_{t+1}$. This distance transform is the digitized version of the Vornoi surface, specifying the distance to the nearest point of a given set. There are a number of efficient methods for computing distance transforms (cf. [2]). From the distance transform array, it is simple to compute the directed Hausdorff distance, $h_K(M_t \oplus x, I_{t+1})$, for a given translation $x$ of $M_t$. This distance is simply the $K$-th largest of the $m$ values of $D_{t+1}$ specified by the nonzero points of $M_t \oplus x$. That is, for a given translation $x$, for each nonzero pixel $p$ of $M_t$, probe the location $p+x$ of $D_{t+1}$. The $K$-th largest of these $m$ probe values is the partial directed Hausdorff distance $h_K(M_t \oplus x, I_{t+1})$ (because each probe value is the distance to the nearest point, and by equation 4) the distance is simply the $K$-th largest such value). In order to find all translations where this distance is less than $\tau$, the computation can be repeated for each translation $x$ of the model with respect to the image.

This basic method can be sped up by several orders of magnitude, using two simple observations. First, for a given translation $x$, as soon as $m - K + 1$ of the probed values are seen to be greater than $\tau$, we know that there cannot be a match at this translation (because at least $K$ of the $m$ probe values must be no more than $\tau$ in order for the distance to be no more than $\tau$). Second, when the $K$-th ranked probe value is substantially greater than $\tau$ at some translation $x$, this rules out translations in a region around $x$ because the distance transform has linear slope (cannot decrease quickly). In particular, if the $K$-th ranked probed value at translation $x$ is $v$ (and $v > \tau$), then the distance must be larger than $\tau$ for all translations in a circle of radius $v - \tau$ around $x$.

These facts, and other similar observations (see [7]), can be used to prune many of the possible translations of a model with respect to an image without explicitly considering these translations. With these speedups, the bulk of the time spent computing the set of translations $X$ is in the computation of the distance transform array $D_{t+1}$, not in the consideration of the possible translations. This is because most translations are ruled out a priori, or after a relatively small number of probes. The ability to quickly compute the set of translations $X$ is central to the success of the tracking method. This allows us to find an object wherever it has moved in the image (as long as its two-dimensional shape has not changed substantially), without limiting the search to some local window of the image around the object's previous location.

Once the set of possible translations, $X$, has been computed, it is partitioned into equivalence classes based on connected components in the grid of translations. This is because there may be numerous adjacent translations in $X$; these will specify approximately the same position of $M_t$ with respect to $I_{t+1}$. In other words, one instance of the object in the image may result in a number of translations that are below the threshold $\tau$, and these are likely to all be neighboring translations. Thus we break $X$ into sets of neighboring translations (i.e., connected components in the translation-space where the distance is below threshold). Each of these sets of neighboring translations corresponds to a possible instance of the object, and is processed separately. Denote each such set of connected translations by $X_i$. We now proceed to find the best translation in each $X_i$, and use that translation as the representative possible location of the model corresponding to the set $X_i$. This means that if $X$ has more than one equivalence class $X_i$, the method will find multiple possible positions of the model in the image. In such cases, the tracking program uses additional information to disambiguate these matches. The process of selecting among multiple possible matches is described in Subsection 5.3, and involves using simple trajectory information.

For each translation $x \in X_i$ we know the actual distance from the model to the image at this translation, $d = h_K(M_t \oplus x, I_{t+1})$. This distance can be used to rate the quality of the translations in $X_i$, those with lower distance being better. Moreover, for each translation $x \in X_i$ we know the actual number of model points, $s$, $K \leq s \leq m$, that are within distance $d$ of image points. In other words, the distance $d$ tells us that at translation $x$ at least $K$ of the $m$ model points are within distance $d$ of image points, but there may be more than $K$ such points (the actual number of such points being $s$). Therefore, each position $x \in X_i$ of the model is 'scored' by a pair $(d, s/m)$, where $d$ is the forward distance $h_K(M_t \oplus x, I_{t+1})$ and $s/m$ is the fraction of the model points that are within distance $d$ of some image point.

We thus define the best match in each equivalence class $X_i$ as the translation that minimizes $d$, which is only natural as we are seeking the minimum Hausdorff distance as given in equation (5). If there are multiple translations with the same (minimal) value of $d$ then we select the one with the largest fraction $s/m$. This match is the 'representative' position of the model in the image for the equivalence class $X_i$. For the remainder of this section, we assume that there is just one equivalence class of translations, $X_1 = X$. In general this is the case, because multiple matches only occur when there are several objects of nearly the same two-dimensional shape in the image. In Section 5 we handle the case of multiple matches, and of no match. The translation $x \in X_1$ with the best match score specifies the location of $M_t$ in $I_{t+1}$. We call this translation $x^*$.

## 4.2 Updating the model

Having found the best translation, $x^*$ of $M_t$ with respect to $I_{t+1}$, the new model $M_{t+1}$ is constructed by selecting those nonzero pixels of $I_{t+1}$ that are within distance $\delta$ of nonzero pixels of $M_t \oplus x^*$. This is done by dilating $M_t$ by a disk of radius $\delta$, shifting this by $x^*$, and then computing the logical **and** of $I_{t+1}$ with the dilated and translated model.

In order to allow for models that may be changing in size (for example objects stretching or getting closer or farther away), the size of the array in which $M_{t+1}$ is stored is increased whenever there are 'many' nonzero pixels near the boundary, and is decreased whenever there are 'few' nonzero pixels near the boundary. The height and width of the model array are adjusted separately, in the following manner: if more than 5% of the nonzero pixels of the model are within distance $\tau_{\max}$ of the left or right boundary, and at least one pixel is on that boundary, then the width of the array is grown by $\tau_{\max}$ on both the left and right. If less than 5% of the nonzero pixels are within distance $\tau_{\max}$ of the left or right boundary, and no pixels are on the boundary then the width of the array is shrunk by $\tau_{\max}$. The height is adjusted using the analogous set of rules.

The initial model, corresponding to the first frame of the image sequence, must be computed specially because there is no previous model. The user specifies a rectangle in the first frame that contains the initial model. The image is then processed to select a subset of the edge pixels in this rectangle. This is done by assuming that the camera does not move between the first two frames, and using this fact to filter the first image frame based on the second image frame (with the filtering and shot noise removal operation described below in Subsection 5.1). Those edge pixels in the user-selected window that moved between the first and second frames are then used as the initial model. Thus the only input from the user is a rectangle in the first frame of the image sequence that contains the moving object (and as little else as possible).

# 5 Extensions to the basic method

In this section we describe three extensions to the basic tracking method presented above. The first of these is a filtering process, in which the stationary parts of the image frame $I_{t+1}$ are removed before the model $M_t$ is matched to it. This improves the performance of the method in cluttered scenes. The second extension is searching for a 'lost object'. When the tracker cannot find $M_t$ in $I_{t+1}$, it first tries raising the threshold $\tau$ within some limit, and if that fails it then tries matching previous models to the image. The third extension deals with situations where the image may contain multiple objects of similar shape. In this case, there will be multiple best matches of the model to the image (i.e., multiple equivalence classes $X_i$). Simple trajectory information is used to choose among several possible matches.

## 5.1 Filtering stationary background

For cluttered scenes, the basic tracking method can be quite sensitive to the choice of the value of $\delta$ (where recall that $\delta$ is the threshold for determining whether or not a point of $I_{t+1}$ is part of $M_{t+1}$). The problem is that if $\delta$ is too small then the model will not tolerate much change in shape from one frame to the next. On the other hand if $\delta$ is even moderately large, the model will start to pick up clutter in the background as part of the object. One means of dealing with this problem is to eliminate certain forms of clutter from the image and from the model. In particular, things that did not change from one frame to the next are not worth tracking, and can be removed from $I_{t+1}$. When the object moved but the camera stayed still, such a filtering process will remove much of the background in an image.

This filtering process is implemented as follows. Any pixel of $I_{t+1}$ that is within distance $c$ of some pixel of $I_t$ is removed from $I_{t+1}$ as a pre-processing step (prior to matching $M_t$ or constructing $M_{t+1}$). Currently the value of $c$ is zero — so any pixel of $I_{t+1}$ that is directly superimposed on a pixel of $I_t$ is discarded. Small values such as $\sqrt{2}$ (one diagonal pixel) are also reasonable as a value for $c$. This filtering tends to leave small one or two-pixel connected components (e.g., pixels in $I_{t+1}$ that extend just beyond the structures of $I_t$). These do not contain any useful information, so we remove them by applying a shot noise filter. Currently this is implemented by placing a $5 \times 5$ window (i.e. $\pm 2$ pixels) centered about each nonzero pixel of the filtered image. If there are fewer than $v$ nonzero pixels in this window, then the pixel is discarded. Values of $v$ equal to 2 or 3 pixels are reasonable (we currently use 2).

Note that if nothing moves from frame $I_t$ to $I_{t+1}$, then the filtering will remove the entire image. This is fine, however, because if nothing moved then no tracking is required; we simply let $M_{t+1} = M_t$. Note that this is not the same as losing the object; in that case there is change in the image from one frame to the next but the model cannot be located.

The filtering process will do very little when the camera moves between $I_t$ and $I_{t+1}$, because in general nearly all of the image will have changed (i.e., different parts of the scene are generally somewhat dissimilar). However, if we assume that the camera motion is under the control of the tracking system, in highly cluttered environments it may be advantageous that every few frames the camera stop moving long enough to get two successive frames at the same position. Another possibility here would be to predict, based on the camera motion in previous frames, where the background would have moved to in $I_{t+1}$, and use this predicted frame rather than $I_t$ for filtering the image. We have not yet experimented with methods based on such predictive processing.

As an illustration of how the method performs for a scene with a relatively small object model and a cluttered background, Figure 4 shows a sequence where an aluminum can is tracked as it is moved around in the image. Over time, the object model incorporates the hand of the person holding the can. With no a priori 3D model, however, this is a reasonable interpretation. The hand moves more or less rigidly with the can, and thus is "part of the same object" in terms of tracking a moving two-dimensional projected shape. There are significant specular highlights on the can, and there is a high density of background edges in the images. Note that the same parameter values were used for this sequence as for the sequences in Figures 1 and 2.

As an illustration of how the method performs when there is camera motion, Figure 5 shows a sequence where the camera pans every few frames (the panning was done manually). There are subsequences of several frames during which the camera is moving, but every few frames the camera stops moving for at least two adjacent frames. This enables the filtering process to keep the models from acquiring too much background. It should be noted that there is substantial motion blur in certain frames due to camera motion which distorts the object shape, but not enough to cause it to be lost. All of the parameter settings were again the same as for the previous sequences.

## 5.2 Finding a lost object

Thus far we have assumed that the same constant threshold $\tau$ is used in computing the set of translations $X$ in equation (6). In practice, however, it is better to set this value dynamically because the Hausdorff distance computation is much more efficient for smaller values of $\tau$ (due to the fact that more translations can be ruled out a priori when $\tau$ is smaller). We thus use the following method to set $\tau$ in the range $[\sqrt{2}, \tau_{\max}]$, for some user-specified maximum value $\tau_{\max}$. At a given time frame $t$, the model $M_t$ is matched to $I_{t+1}$ using the current value of the threshold $\tau$. If no matching translations of the model are found (the set $X$ is empty), then $\tau$ is doubled and the matching operation is repeated with this new value. As long as no match is found, the threshold doubling continues until it reaches the specified maximum value $\tau_{\max}$ (we typically set $\tau_{\max}$ to 10 pixels).

The value of $\tau$ is also decreased when it is reasonable to do so. This is done by reducing

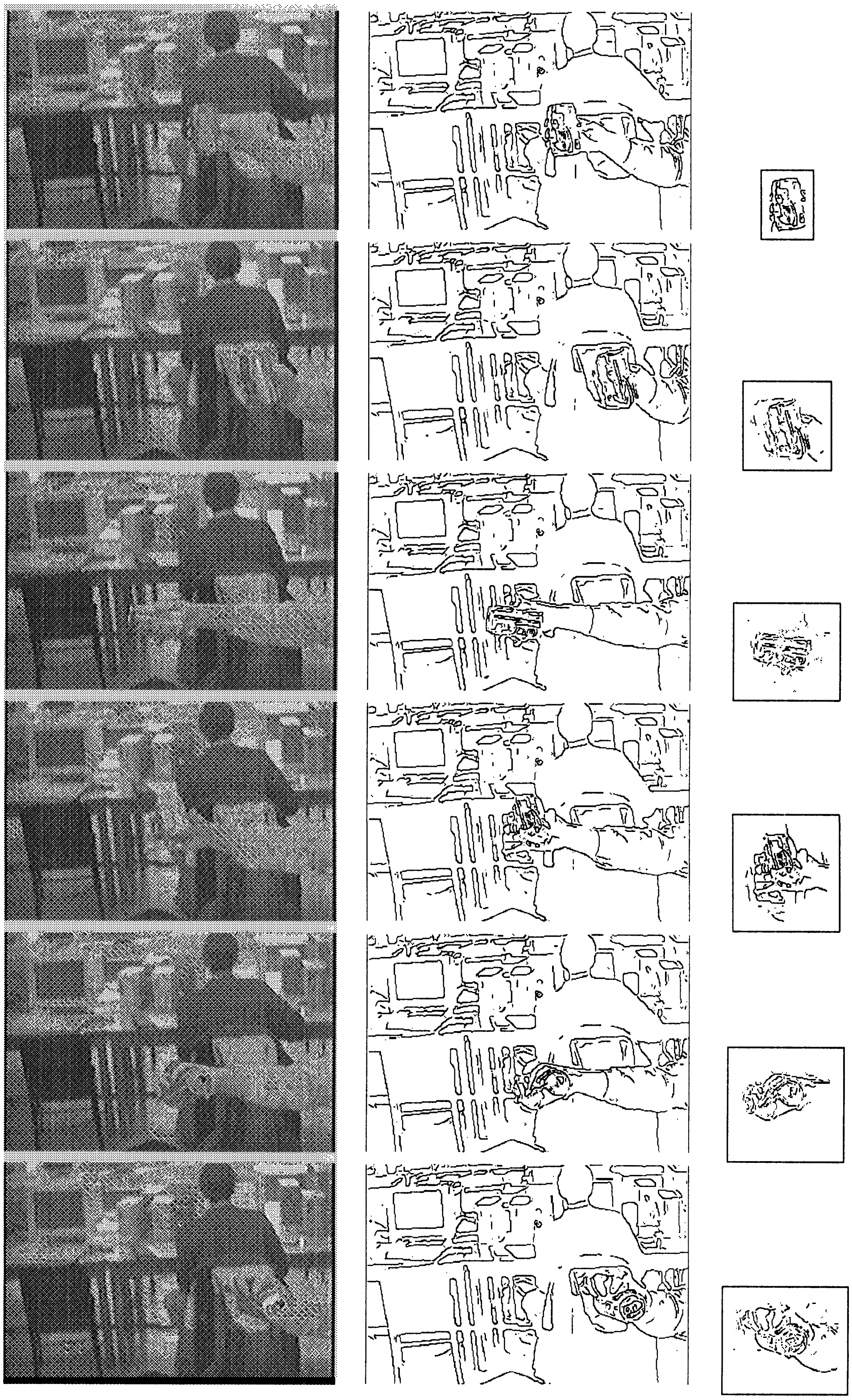

Figure 4: Six selected frames (numbers 1,20,40,60,80 and 100) from a 100 frame motion sequence, and the corresponding models of the object being tracked.

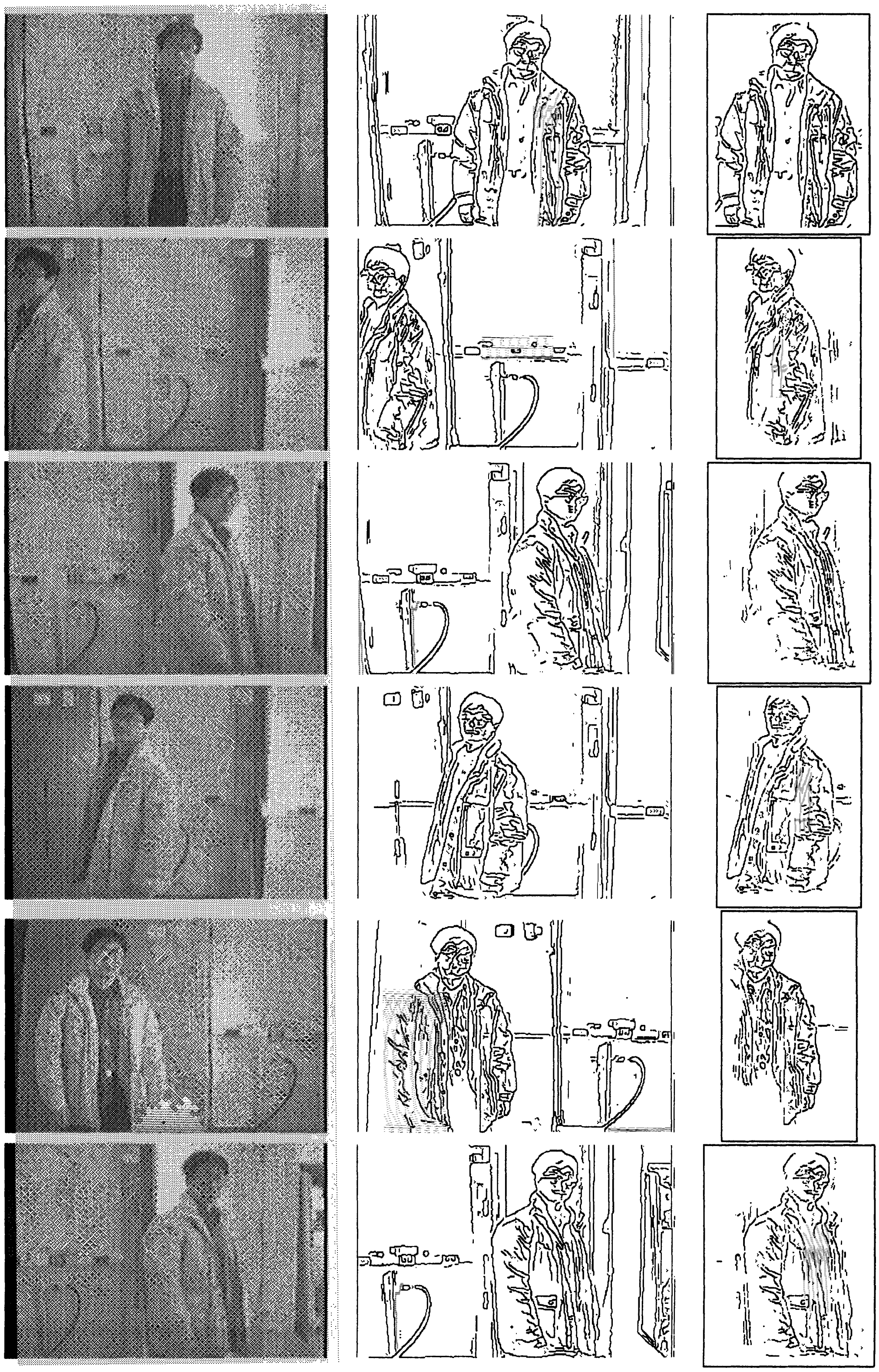

Figure 5: Six selected frames (numbers 1,20,40,60,80 and 100) from a 100 frame motion sequence, and the corresponding models of the object being tracked.

the value of $\tau$ on each frame where a match is found and the distance $d$ for that match is smaller than $\tau$. The decreases in $\tau$ are made by decrementing rather than halving the value. Note that the value of $\tau$ affects only the efficiency of the method, *not* its ability to track; the effective threshold may be considered to be $\tau_{\text{max}}$, as it is only this parameter that affects the ability to track objects. (As an aside, there is a potential interaction between the value of $\tau$ and the number of equivalence classes, $X_i$, that are found. When $\tau$ is smaller than $\tau_{\text{max}}$ the method could in principle split the set of translations corresponding to one instance of the object in the image into more than one equivalence class, and thus report multiple matches when in fact there is only one. This has not happened for any image sequence that we have used. Moreover, even if it does happen, all that needs to be done is to pick the better of any two matches that overlap substantially in the image — and clearly these matches must overlap because they are from the same underlying instance of the object in the image.)

The tracking method as described thus far does not work when the object being tracked temporarily disappears from view or changes shape suddenly from one frame to the next. In such cases the pair of frames where there is a large change in image shape will result in no matching being found of the model $M_t$ to the image $I_{t+1}$ at the distance $\tau_{\text{max}}$. When this happens, we compare additional models to the image frame, $I_{t+1}$, in which the object no longer was found. These additional models are the models from certain previous time frames, $t_1, t_2, \ldots$ $(1 \leq t_i \leq t)$ which were chosen as 'canonical' views of the object. We refer to this as 'hunt down mode', because the past history of the system is used to hunt for the missing object.

In this mode, the 'canonical' models are matched to the image, until either a good match is found, or the set of canonical models, $\mathcal{C} = \{M_{t_1}, M_{t_2}, \ldots\}$, has been exhausted. If one of the canonical models matches the image, then this model is used to form the model $M_{t+1}$. The first matching model that is found is used. More specifically, each model $M_{t_i} \in \mathcal{C}$ is compared with $I_{t+1}$ using the forward distance computation in equation (6). If for a given model $M_{t_i}$ the set $X$ of translations is nonempty (i.e., there are matches where the forward distance is less than $\tau$), then that model $M_{t_i}$ is used to construct $M_{t+1}$ in the normal manner: the best translation $x^* \in X$ is identified (as described in the previous section) and $M_{t+1}$ is formed by selecting those pixels of $I_{t+1}$ that are within distance $\delta$ of $M_{t_i} \oplus x^*$. (Note the canonical model $M_{t_i}$ does not replace $M_t$, the model at time $t$, but is simply used to form $M_{t+1}$ in the normal manner.)

If none of the models in $\mathcal{C}$ match $I_{t+1}$ then the tracker has lost the object at that time frame. No model array is constructed for such a frame. Rather than giving up when no match is found, the tracker continues trying to match $M_t$ and the set of models $\mathcal{C}$ to each successive image frame until a matching image frame is found. No model is built for such frames, because the object has either disappeared from the image or has changed shape so radically that it can no longer be identified.

The set of canonical models, $\mathcal{C}$, is constructed by selecting models from certain frames of the image sequence. After each model $M_{t+1}$ is created, it is compared with the models in $\mathcal{C}$. If it is sufficiently different from all of these models, then it constitutes a new view of the object (a new 2D shape) and is added to the set. This comparison is done using the *bidirectional* minimum Hausdorff distance, because in order for two models to be similar to one another we want to ensure that many points of one are near the other and vice versa.

This differs from the use of the Hausdorff distance to find a model in an image, where we use just the forward distance to find possible locations of the model with respect to the image. Thus we say that the model $M_{t+1}$ is similar to some other model $M_{t_i}$ when there exists at least one translation $x$ such that

$$h_K(M_{t+1} \oplus x, M_{t_i}) \leq \delta \wedge h_{K'}(M_{t_i}, M_{t+1} \oplus x) \leq \delta \ . \tag{7}$$

In this equation, $K = \lfloor fm \rfloor$, and $K' = \lfloor fm' \rfloor$, where $m$ is the number of nonzero pixels in $M_{t+1}$ and $m'$ is the number of nonzero pixels in $M_{t_i}$.

If $M_{t+1}$ is not similar to any model $M_{t_i} \in \mathcal{C}$ then it is added to the set. We use the same values of $f$ and $\delta$ as are used in tracking the object and forming the model $M_{t+1}$. This process of comparing each model with the set $\mathcal{C}$ is relatively fast, because the different models are in general approximately the same size, and thus there are not many translations, $x$, to consider.

The set of canonical models built in this way thus represents the distinctive views, or two-dimensional shapes, of a given object. The tracker learns what shapes correspond to a given object, and then uses those shapes to track the object when necessary. For the image sequences in this paper, there are generally around 10 or 12 canonical models formed for a given object (out of the 100 frames). The process of matching multiple models to $I_{t+1}$ is not much slower than just using the single model $M_t$. This is because, as noted in Subsection 4.1, the bulk of the time in finding the set of possible translations of the model (the set $X$) is in computing the distance transform of the image. The distance transform is only computed once for a given image, even when multiple models are being compared to the image.

The sequence in Figure 6 shows two people walking around, and contains several frames where one person goes behind the other. In some of these cases, the tracking method temporarily loses the object being tracked (neither $M_t$ nor any of the models in $\mathcal{C}$ match the image). For such frames, no model is generated. After a few frames, the object comes back into view, and it is found again. In some of the cases that one person disappears behind the other, the tracker incorrectly matches the other person. This situation is discussed in the following subsection.

## 5.3 Disambiguating multiple matches

The tracking method as described thus far does not work for sequences containing multiple moving objects of approximately the same 2D shape. In such cases several matches of the model $M_t$ will be found to the image $I_{t+1}$. In this subsection we discuss some extensions to the tracker in order to handle this type of situation. Note that while in principle it is possible to get multiple matches of a model to an image due to erroneous matching of the model to the background, in practice this does not happen unless the the model is very sparse compared to the background. Thus if the model has very few points, this can be a problem. In all the image sequences in the paper, and many others, multiple matches occur only when either (i) the scene really contains two or more objects of a similar 2D shape or (ii) there is dense background that "appears" when the object moves. Both of these cases are addressed here by using simple trajectory information.

We augment the tracking method to use simple trajectory information (whereas the basic tracking method described above does not use the trajectory of an object in the image). The idea is to use trajectory information in order to identify frames where the object being tracked might be incorrect. For instance, frames in which the trajectory of the object being tracked has changed suddenly are candidates for places where the tracker might have made an error and started to track the wrong thing in the image. In such a case, the tracking program uses additional models in order to see if any of them match the image as well as $M_t$ does. If any of these models do match well (or there are multiple matches of $M_t$) then the match that best meets the trajectory constraints is used. In effect, when another model is used in place of $M_t$ it is assumed that $M_t$ corresponds to some impostor in the image. For example, this happens when the object being tracked disappears from view and there is some other similar shaped thing in the image. The initial object might later reappear, but if we were using $M_t$ alone we would probably not pick it up, since $M_t$ would represent the shape of the other (incorrect) object. This situation occurs in the sequence shown in Figure 6, where there are two people walking around who disappear behind one another on several occasions. The two shapes are similar enough that sometimes when the person being tracked disappears behind the other person, the incorrect person initially matches well enough to become the new model. In such cases, however, there is a sudden trajectory change, which in effect causes the tracker to be 'suspicious' about whether it is actually following the correct object.

In the current implementation of the tracker we use very simple trajectory constraints, in order to detect situations in which the tracker may have started to follow the wrong object. The 'trajectory' of the object is defined to be the translation from frame $I_{t-1}$ to $I_t$. That is, the model $M_{t-2}$ matched $I_{t-1}$ at some translation $x^*_{t-1}$ and the model $M_{t-1}$ matched $I_t$ at some translation $x^*_t$. The vector $v = x^*_t - x^*_{t-1}$ is defined to be the current trajectory of the object. The motion of the object is simply $w = x^*_{t+1} - x^*_t$, where $x^*_{t+1}$ is the location of the match of $M_t$ to $I_{t+1}$. When there is a significant change between $v$ and $w$, the tracker enters 'suspicious mode' in which it tries certain additional models in order to make sure that the correct object is actually being tracked. Currently a significant change in trajectory is when $v \cdot w < 0$, which means that the trajectory direction changed by more than 90 degrees. In practice, it may be desirable to set the trajectory, $v$, based on more than a single previous frame.

When the tracker goes into suspicious mode, it matches the models from the previous several frames (currently the previous 5 models). Given all the matches of these models and $M_t$, the tracker selects the match that has the closest trajectory to the previous frames. Once the tracker goes into suspicious mode, it stays that way for several frames (currently 5 frames or until one of the other models matches better than $M_t$, whichever comes first). This is because the object being tracked may be temporarily hidden from view, so the tracker gives it a few frames to re-appear (and to be matched by one of the previous views). If this does not happen, then the tracker assumes that its suspicions were unfounded and continues tracking as usual.

More sophisticated methods of using trajectory information, and matching multiple models to the image are possible — we are just beginning to explore the simultaneous use of trajectory information with two-dimensional model matching.

Figure 6 shows an example where two objects were tracked through an image sequence. Frames 1,20,40,60,80 and 100 are shown. During the sequence, each object passed in front

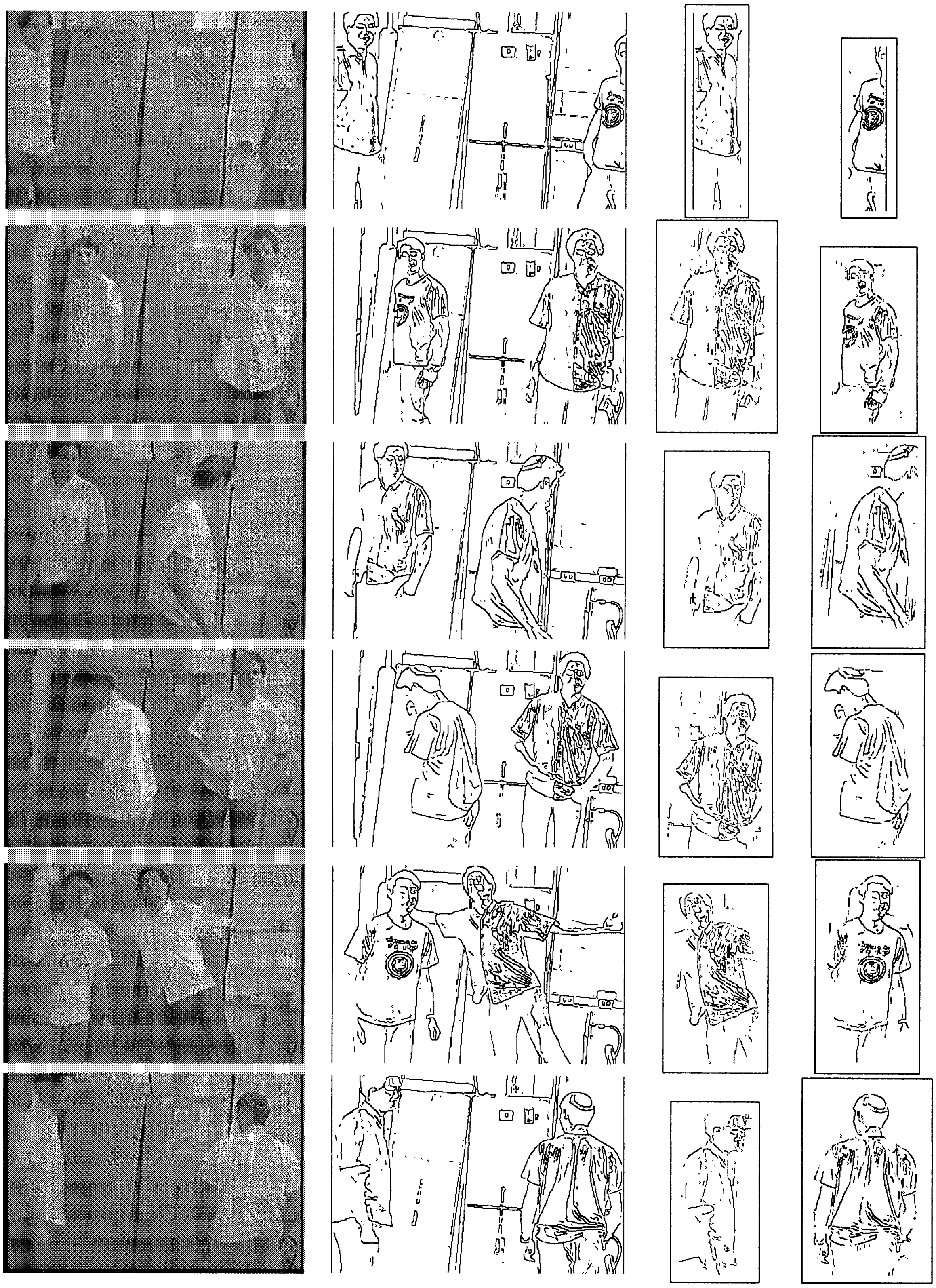

Figure 6: Six selected frames (numbers 1,20,40,60,80 and 100) from a 100 frame motion sequence, and the corresponding models of the two objects being tracked.

of the other several times, and so was obscured from view. The objects also changed their two-dimensional shape quite significantly. Nonetheless, each one was successfully tracked through the entire sequence, and the tracker did not misidentify one object as the other (although it did temporarily track the wrong object for a frame or two in both sequences of models). This sequence really illustrates the potential power of model-based methods in which two-dimensional shape is used to track an object. Without such additional global constraints (or overly strong trajectory constraints), it is very difficult to track objects that pass behind other similar objects.

# 6 Summary

The steps of the tracking method are thus as follows.

For each frame of the image sequence we assume that some feature detector has been run. Currently we use a variant of the edge detector described in [1]. We form our initial model based on those pixels that moved between the first two frames, $I_1$ and $I_2$. Then for each additional binary image frame $I_{t+1}$, we do the following:

1. Filter the image by removing those nonzero pixels that are greater than distance $c$ from nonzero pixels of the previous image frame $I_t$ (currently $c = 0$), and apply a shot-noise filter to the result. Call this resulting image $I'_{t+1}$.

2. Find the best matching position of the model from previous time, $M_t$, with respect to the filtered image $I'_{t+1}$, using the partial directed Hausdorff distance as in equation (6) with $K = \lfloor fm \rfloor$ (where we have used $f = .8$ and $\tau_{\max} = 10$ pixels).

   - Trajectory information is used, as described in subsection 5.3, to disambiguate multiple matches.
   - The set of of canonical models, $\mathcal{C}$, is used to try to locate the object if no match of $M_t$ to $I'_{t+1}$ is found.

3. Construct the new model, $M_{t+1}$, by selecting those pixels of $I'_{t+1}$ that are within distance $\delta$ of pixels of the translated $M_t$ (where we have used $\delta = 8$ pixels). Update the size of the model array if required.

There are three parameters for the matching method: the maximum forward distance $\tau_{\max}$, the forward fraction $f$ (where $K = \lfloor fm \rfloor$), and the model construction threshold $\delta$. These parameters have relatively simple meanings. The first two parameters control the amount of shape change that is allowed from one frame to the next — the distance $\tau_{\max}$ controls how much non-translational motion is tolerated and the fraction $f$ controls what portion of the model can remain unaccounted for in the image. The third parameter controls the amount of non-translational shape change that is incorporated into the model at the next time frame. (We have used the same values of these parameters in all the examples in this paper, and in many other cases.) In addition, there are several constants in the tracking method: the fraction of model pixels that is used in expansion/contraction of the model array, the distance that is used in image filtering, and the size of the shot-noise filter. These parameters were always set to the same values for all the sequences shown in the paper (and for many other sequences as well).

The tracking method thus consists solely of matching two-dimensional geometric models to two-dimensional edge images, finding the position of the model at one time frame in the image at the next time frame, and updating the model to reflect the change in its shape. There is no limitation on the search in the image — the model can translate anywhere from one frame to the next. The non-translational motion, however, must be small from one frame to the next, as this is considered to be a change in the two-dimensional shape of the object. Such shape changes are limited to a distance of $\delta$ pixels in two subsequent frames.

## 6.1 Conclusions

We have described a method for tracking three-dimensional objects which are moving in space and may be altering their three-dimensional shape. The method decomposes this motion and shape change into two components: a two-dimensional motion (which in we take to be a translation in the image plane), and a two-dimensional shape change (which we take to be all the remaining change in the two-dimensional image of the object — i.e., both the non-translational component of the image motion and the three-dimensional shape change). The major assumption is that this second component generates only a small change between two consecutive frames, though the cumulative change may be large. There is no such restriction on the two-dimensional translation between frames; objects are not restricted to move locally. We have shown extensions which eliminate stationary background information to improve the quality of the tracking, and which use information from previous frames (both shape and trajectory information) to ensure that a correct match is found in subsequent frames.

The method successfully tracks objects through image sequences in which they change overall shape substantially, they move large distances from one frame to the next, and they may be partially or fully occluded in several successive frames.

# References


[1] J.F. Canny. A computational approach to edge detection. *IEEE Trans. Pat. Anal. and Mach. Intel.*, 8(6):34–43, 1986.

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

[3] E. Dickmanns and V. Graefe. Dynamic monocular machine vision. *Machine Vision Applications*, 1:223–240, 1988.

[4] D. Gennery. Tracking known three dimensional objects. In *Second National Conf. on Artificial Intelligence*, pages 13–17, 1982.

[5] D.P. Huttenlocher, K. Kedem, and J.M. Kleinberg. On dynamic Voronoi diagrams and the minimum Hausdorff distance for point sets under Euclidean motion in the plane. In

*Proceedings of the Eighth ACM Symposium on Computational Geometry*, pages 110–119, 1992.

[6] D.P. Huttenlocher, K. Kedem, and M. Sharir. The upper envelope of Voronoi surfaces and its applications. In *Proceedings of Seventh ACM Symposium on Computational Geometry*, pages 194–293, 1991. To appear in *Discrete Computational Geometry*.

[7] D.P. Huttenlocher, G.A. Klanderman, and W.J. Rucklidge. Comparing images using the Hausdorff distance. *IEEE Trans. Pat. Anal. and Mach. Intel.* To appear.

[8] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(3):321–331, 1988.

[9] D. Koller, K. Daniilidis, T. Thórhallsson, and H.-H. Nagel. Model-based object tracking in traffic scenes. In *Proc. 2nd European Conf. on Computer Vision*, pages 437–452, Santa Margherita Ligure, Italy, May 1992.

[10] D.G. Lowe. Robust model-based motion tracking through the integration of search and estimation. *International Journal of Computer Vision*, 8(2):113–122, 1992.

[11] N. Ueda and K. Mase. Tracking moving contours using energy-minimizing elastic contour models. In *Proc. 2nd European Conf. on Computer Vision*, pages 453–457, Santa Margherita Ligure, Italy, May 1992.

[12] G. Verghese, K. Gale, and C.R. Dyer. Real-time, parallel motion tracking of three-dimensional objects from spatiotemporal image sequences. In Kumar et. al., editor, *Parallel Algorithms for Machine Intelligence and Vision*, pages 340–359, New York, 1990. Springer-Verlag.

[13] J. Woodfill and R.D. Zabih. An algorithm for real-time tracking of non-rigid objects. In *Proc. American Association for Artificial Intelligence Conference*, 1991.