Keeping One's Eye on the Ball: Tracking Occluding Contours of Unfamiliar Objects without Distraction

Kentaro Toyama and Gregory D. Hager

Research Report YALEU/DCS/RR-1060
January 1995

Keeping One's Eye on the Ball: Tracking Occluding Contours of Unfamiliar Objects without Distraction

Kentaro Toyama and Gregory D. Hager

Department of Computer Science

Yale University*

December 6, 1994

Abstract

Visual tracking is prone to distractions, where features similar to the target features guide the tracker away from its intended object. Global shape models and dynamic models are necessary for completely distraction-free contour tracking, but there are many cases when component feature trackers alone can be expected to avoid distraction. We define the tracking problem in general and devise a method for local, window-based, feature trackers to track accurately in spite of background distractions. The algorithm is applied to a generic line tracker and a snake-like contour tracker which are then experimentally analyzed with respect to previous contour-trackers. We discuss the advantages and disadvantages of our approach and suggest that existing model-based trackers can be improved by incorporating similar techniques at the local level.

*P.O. Box 208285, New Haven, CT, 06520. E-mail: toyama@cs.yale.edu, hager@cs.yale.edu

1 Introduction

Given a series of images, the problem of visual tracking is to identify the apparent motion of a target from frame to frame. Many applications use assumptions about a motion model or the rigidity of the target to perform their function. In this way, the tracking problem is reduced to a combination of prediction and localized search: first, tracking algorithms guess the new configuration of a visual feature, then they search near the guess to determine the actual state of the feature.

By narrowing the search area, tracking applications try to avoid both computational complexity and mistracking. Unfortunately, limited search area does not imply that the search is trivial or even uniquely solvable. In fact, the searching stage of tracking is often a small-scale problem in pattern recognition.

Most search algorithms rely on one of two pattern detection methods: edge detection or correlation-based pattern matching. Generic edge detection has the advantage that it is relatively insensitive to many types of changes in the environment, such as illumination and contrast changes across the contour. However it suffers from a lack of specificity. In many circumstances, a generic edge-based tracker is easily "distracted" by background features. Successful tracking with edge operators depends on having an extremely localized search through the use of geometric or temporal constraints. Pattern matching techniques such as correlation suffer from exactly the opposite problem — they are often too specific. Small changes, for example, in the local brightness or contrast can easily ruin or bias the match.

We have begun to use visual tracking in a variety of applications where the environment is unstructured [7], implying that specific geometric and dynamic information is not available. Hence, our tracking algorithms must rely on local feature detection to perform properly. In this article, we address the question of what can be tracked locally in a distraction-free fashion. We first delineate a set of situations where local tracking can be performed robustly. Then, based on these observations, we develop algorithms which are more robust than either

correlation-based or edge-based tracking and offer experimental evidence for this claim.

The remainder of this article is structured as follows. In the next section, we describe a generic model for feature tracking and review related work in the area of visual tracking. Next, we consider what kind of contours can be tracked accurately based on local information alone. We then describe our algorithm; and finally, we implement the algorithm for the specific problems of tracking edge segments and tracking occluding contours. The performance of our distraction-proof trackers is analyzed and compared with existing trackers for similar objects.

2 Background and Tracking Problem Definition

In this section, we define trackers as state-based observers with minimal dynamics and introduce the notions of model-based and temporal constraints.

2.1 Problem Formalization

We view tracking as a state-based control system. At the image level, pattern recognition algorithms compute the state of a feature assumed to fall within a local search area, commonly referred to as a *window*, in an image. Windows are subregions of the entire camera image with position, length, width, scale, orientation, and possibly other parameters. The use of small windows to track isolated features makes tracking faster. For example, a contour tracker could focus on small, isolated regions near the contour edge, instead of processing entire image frames.

We formalize feature trackers here as functions which return states or state offsets of features given the pixel values from a window. The state vector of a feature may include position and orientation, as well as a variety of other geometric parameters. In the sequel, a state space for a feature x is denoted F_x. The notation $\mathcal{F}^n$ is shorthand for $\prod_{i=1}^{n} F_i$ where

F_i, $1 \leq i \leq n$, are arbitrary feature state spaces. The variable $t \in [0, \infty]$ represents time. The notation $f_i(t)$ denotes the state of feature i at time t. Systems evolve continuously, but are sampled at a fixed but arbitrary time increment (which we normalize to 1). The notation t_k is equivalent to the instant k time intervals following t_0.

Local feature detectors attempt to minimize objective functions, which may be temporally dependent or independent. Time-dependent objective functions try to solve some kind of temporal correspondence problem. That is, they are trying to correlate features between two images taken at different time instances. We model this process as an optimization as follows:

$$f(t) = \hat{f}(t) + \arg \min_{f(t) \in \mathcal{F}(t)} O(f(t), \mathcal{W}_{t'}, \mathcal{W}_t)$$

where $\mathcal{F}(t)$ is a set of potential state offsets, $\mathcal{W}_t$ is a window of an image acquired at time t, $\hat{f}(t)$ is a prediction of the feature state at time t, and t' is commonly $t-1$ or t_0.

Temporally dependent objective functions are usually variations on auto-correlation methods. In particular, sum-of-squared-difference (SSD) methods have become quite popular [10, 13]. Tracking based on correlative methods works well for matching specific patterns, yet it becomes unreliable in many situations. If a feature is not visible in its entirety in a particular frame, correlation fairs poorly; when shears and rotations are expected, computing time increases drastically; and if the target image is almost constant or if it contains repetitive elements, the matches are susceptible to minute effects of noise.

At the other end of the spectrum lie simple edge detection methods [2, 4, 6]. These methods are temporally independent in that they make decisions about a feature's state by observing only the current window.

We define $\mathbf{s} = (s_x, s_y, s_\theta)^T$ where s_x and s_y denote the center of a finite line segment, and θ denotes the direction of the line in image coordinates. $\mathcal{S}$ is a set of potential line state offsets. Edge-based methods can then be formalized as:

$$\mathbf{s}(t) = \hat{\mathbf{s}} + \arg\min_{\mathbf{s}\in\mathcal{S}} \frac{DW_t(s_x, s_y)}{Ds_\theta}.$$

where $W_t(\cdot,\cdot)$ denotes the window brightness function, and D denotes directional derivative. Note that no information about the prior image is used in this matching process; it simply finds the "strongest" edge.

Edge-based methods can be made robust to noise, to varying brightness, and to other abrupt changes in the image region surrounding a feature. Yet while they excel at finding features quickly, their emphasis on "edgeness" over sameness makes them especially prone to distraction, which we define presently.

A feature is *visible* at time t if it lies entirely within its window, and nothing physically interferes between it and the camera. A feature is *observable* if the set of allowable states at time t, $\mathcal{F}(t)$, contains the true feature state.

We define *distraction* as an event that occurs when other objects move into a window of an observable target component, and the feature tracker's objective function returns a value for its state, $f(t)$, other than the one sought by the user. We note that distraction is a subjective term, where whether a feature tracker is distracted depends upon the intent of the algorithm and the intent of the task. It is of interest to note that an objective function that is adequately designed for its feature in an expected environment should never be distracted. Unless the feature tracker finds more than one minimum, distraction occurs only when the objective function is not accurately defined.

2.2 High-Level Constraints

Three other types of information are commonly used in tracking applications. In *model-based tracking,* information about the geometry of features is supplied, as in [9]. Here, the configurations of physical features are expected to fit some shape model of the object, and visual tracking of those features is enough to track the state of the modeled object. The

principle advantage of model-based tracking is that the model overconstrains the possible configuration of features, so incorrect feature tracking information can often be detected and discarded. Furthermore, the location of these errant features can be predicted from the model state. In unstructured situations, we do not have prior information about the object, so strong model-based constraints usually cannot be used.

Likewise, when information about the object dynamics is available, temporal constraints can be used localize the search for features [4, 5, 11]. The object dynamics can be expressed in terms of the image features [5], or in terms of a model [11]. Again, in unstructured situations, it is unlikely that highly reliable dynamic information is available, except on the camera system itself. Even when dynamic information is known, it may not be precise enough to prevent distraction.

In *constraint-based tracking,* no explicit model is used. However, the feature states are forced to respect one or more constraints $C : \mathcal{F}^m \rightarrow \Re$. An example is found in [2]. Their work supposes constant velocity motion and an affine motion constraint on the set of features being tracked. Kalman filters applied both to the motion and the affine model allow temporary distactions to be eventually disregarded as noise. However, because their methods see distraction merely as noise to be filtered out over time, they still fail during initial phases of distraction or in cases of slow distraction. We seek to avoid even such temporary distraction, and expect to do so without explicit models for the object or dynamics. Including model constraints clearly enhances the overall reliability of the tracking algorithm, but tracking errors may still cause estimation errors and bias. We expect that robust feature detection will benefit these algorithms by significantly reducing such estimation errors.

3 Robust Local Tracking

In many cases, particularly those involving the occluding contours of object, correct tracking performance is difficult to guarantee. We have often watched with dismay when a tracker

meant to track a screwdriver finds the light switch on the wall more interesting or cringed helplessly when a casual bystander unwittingly passed behind an experiment, only to drag our trackers away.

There are two ways to decrease the likelihood of distraction. The first is to decrease the size of the search region through better temporal or spatial constraints. However, this alternative is only available in structured environments where prior knowledge about the object's spatial or temporal properties are given; and even with prior models, the models themselves may not be precise enough to eliminate the possibility of local distractions. The second way is to increase the local information retained about the feature's appearance and to develop algorithms that use this information in a robust fashion. This section considers the latter alternative, specifically, as it applies to contour tracking. Similar analysis reveals identical results for edge tracking, as well.

3.1 What Can't Be Tracked Locally

Some tracking problems have no solutions at a local, window level. For instance, if we anticipate partial foreground occlusion of a contour, no algorithm can be expected to know the location of a hidden edge simply by observing single windows or even rows of adjacent windows. What is observed as an occludor enters a small window could be mimicked by different combinations of background motion and object motion, making it impossible for individual windows to determine even if the edge is visible or occluded.

If we assume that foreground occlusion will not occur, we know that the contour will, at least, be visible to appropriately placed windows. Some analysis reveals, however, that local window information will still not suffice in certain situations. For example, if we consider a soda can viewed from its side and rotating about it's longitudinal axis, we notice that small windows observing its contour edge will not be able to distinguish between new patterns rotating into view and similar background patterns appearing from behind. Thus, a single

window (or even a series of windows) cannot unambiguously determine the location of the occluding contour without using model information.

More complex issues arise if local windows are not guaranteed to be tracking the same part of the object contour. If the foreground object appears to change from frame to frame within a window, it is not clear how such a difference can be understood as a change in the foreground object rather than additional background motion.

There are quite a few other situations where local information alone is insufficient to track occluding contours without distraction, leading us to suspect, therefore, that there will never be distraction-free, context-independent contour tracking. Algorithms which incorporate global shape models or environments that fit narrower assumptions are necessary to track occluding contours correctly.

3.2 What *Can* Be Tracked Locally

There are many generic instances where a little local processing can make contour-tracking much more robust to background distractions. The principle requirement is that the foreground immediately within the contour must have enough spatial and temporal stability so that local trackers can recognize it from one frame to the next.

In particular the contours of the following types of objects can be tracked accurately, using only local knowledge, regardless of background distraction:

- Any kind of connected object with monotone brightness, moving in any way.
- Any connected, rigid, planar object with relatively large patches, where each patch is of monotone brightness, moving in any way such that the line normal to the plane is never perpendicular to the optical axis of the camera.
- Any connected, rigid, curved object with relatively large patches of monotone brightness, translating in any direction and/or rotating only about the optical axis of the

camera.

Some additional constraints which must be fulfilled in order for these objects to be tracked include reasonable sampling assumptions, rarity of the background appearing like the foreground with respect to brightness, and spatial/temporal smoothness in ambient lighting changes.

In the discussion above, "brightness" can be replaced by other qualities, providing that there exist reliable algorithms for detecting those qualities and distinguishing them from things without those qualities. Such qualities might include color, certain textures, etc. Depending upon the quality or qualities chosen, additional assumptions may be necessary and others may become disposable. For example, color-based tracking may eliminate the need for some ambient lighting assumptions, while requiring that any lighting changes vary smoothly with respect to color.

We now search for an algorithm which can track parts of these objects in spite of background distraction.

4 Feature Tracking

Our high-level trackers (see Section 5) expect the individual low-level feature trackers to return a 1-dimensional offset for the new location of a contour edge. Thus, search windows are a thin line of pixels, where we assume that every tracked edge component has a foreground on one side and a background on the other, where the foreground falls within the assumptions given in Section 3.2, and the background is continually, and unpredictably, changing. Locally tracking an edge under these assumptions requires some mixture of edge detection and foreground pattern matching.

Several objective functions were considered as candidates for determining the location of the new contour edge:

SSD:

$$j^* = \arg\min_{j=0}^{w-k} \sum_{i=0}^{k-1} (p_{i+j} - p'_i)^2, \tag{1}$$

where $p_i = p_i(t)$ is the image value of the ith pixel at time t, $p'_i = p'_i(t')$ is an image value for the previous foreground ($t' = t - 1$) or the original foreground ($t' = 0$), w is the width of the window, and j^* is the location of the new edge ($w/2$ should be subtracted to find the offset). The SSD method searches for the best location to match the k values nearest the edge of the previous foreground with the new image, where "best" is defined as where the sum of squared differences is least.

Gain-adjusted SSD:

$$j^* = \arg\min_{j=0}^{w-k} \frac{\sum_{i=0}^{k-1} (p_{i+j} - p'_i)^2}{\gamma_j}, \tag{2}$$

where p_i, p'_i, j^*, and k are as in normal SSD, and γ_j is a factor that normalizes the intensity values of a strip of k pixels such that the mean intensities of attempted matches are the same.

Edge-biased SSD:

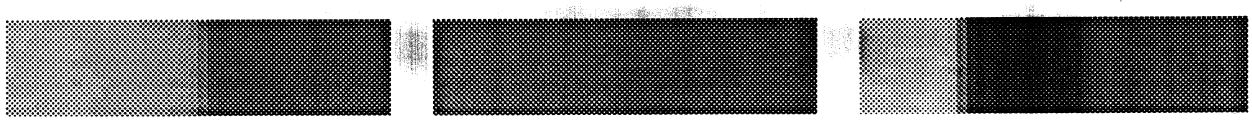
$$j^* = \arg\min_{j=0}^{w-k} \sum_{i=0}^{k-1} (p_{i+j} - p'_i)^2 - \beta \text{edge}(j), \tag{3}$$

where β is a weight that biases the SSD near edges, and edge(j) is a function that returns 1 if an edge exists at location j. This performs an SSD calculation similar to that of Equation 1, but prefers matches that coincide with an observable edge.

Coarse SSD:

$$\hat{j} = \arg\min_{j=1}^{|m|} \sum_{m=1}^{k} (m_{i+j} - m'_i)^2, \tag{4}$$

where m's are mode values of regions between edges, and $\hat{j}$ is the edge number (not location) of the best edge. This computes SSD values on modes of gray-scale values between edges. We use "mode" in the standard statistical sense of "the most frequently occurring event."



(a) (b) (c)

Figure 1. (a) A simple edge, (b) in a shadow, and (c) with background distraction. In (c), the patch of gray on the far right is the foreground object. The patch in the middle is the distractor which has crept into the window from behind the foreground object.

Mode values were used in this computation to stabilize otherwise noisy input — outlying pixel values are ignored, and the rest are tallied such that the dominant intensity value is returned as the region intensity.

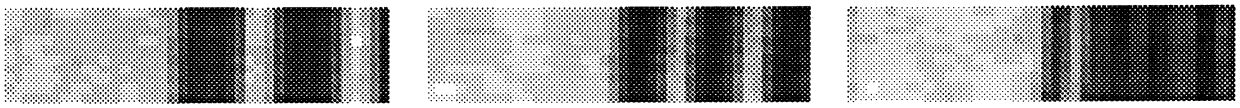
Strongest Edge:

$$j^* = \arg \max_{j=k/2}^{w-k/2} \left| \sum_{m=j-k/2}^{j} i_m - \sum_{m=j+1}^{j+k/2} i_m \right|, \tag{5}$$

where j^* is the location of the strongest edge and w is the width of the image. This convolves a discrete gradient mask (of size $k < w$) with the image values and finds the strongest edge in the new image without considering previous frames.

Our desiderata for a reliable tracker are that it can track in spite of gradual illumination changes, avoid distraction, deal with translations, and be reasonably fast. Thus, experiments were run to test the ability to track simple edges and textured edges, each with distraction, ambient lighting changes, scale changes, and varying speeds. For these experiments tracking was considered successful if an algorithm continued to track the correct edge as the target was smoothly displaced. In general, algorithms either succeeded or failed consistently on any specific target, so algorithms were rated on their ability to track under these conditions with slightly different targets and distractors, different suddenness or degrees of lighting changes, and different degrees of scale changes.

Tracking speed is a function of the width of the window, since wider windows require



(a) (b) (c)

Figure 2. (a) A periodically textured foreground, (b) slanted for scale change, and (c) an irregularly textured foreground.

greater processing time. Algorithms were standardized with widths of 40 pixels. The remaining two components that determine speed are the number of arithmetic operations computed each frame and the width of the foreground that needs to be visible for accurate tracking to take place. We simply measured the speed at which the target could be moved without mistracking and without leaving the tracker behind.

Examples of the types of images seen by component windows are shown in Figures 1 and 2. The right half of each image is cast in the experiments as the foreground object.

In Figure 3, we summarize how well each algorithm performed on several different examples of each kind of task. Fractional numbers indicate the number of successful tracking tests out of the total number of trials, where each trial is of a different foreground/background combination.

These results are as expected. Valuing the existence of an edge adds robustness to pure SSD methods. Thus, coarse SSD and edge-biased SSD both performed excellently overall, since coarse SSD does not allow a contour to be found except where a detectable edge exists, and edge-biased SSD prefers detectable edges. Simple edge detection, while robust in simple cases, fails immediately when any kind of texture or distraction is considered. Pure SSD fared poorly for constant-intensity foregrounds because matches to previous foregrounds often floated away from the contour as negligible changes in pixel values made one constant-valued region preferable to another. On the other hand, including part of the background

Method:	1a	1b	2a	2b	3a	3b	4a	4b	5
Simple edges	2/5	1/5	0/5	0/5	4/5	4/5	5/5	5/5	4/5
Distraction free	0/5	0/5	0/5	0/5	4/5	4/5	5/5	4/5	2/5
Robust to lighting changes	0/5	0/5	0/5	0/5	4/5	5/5	5/5	5/5	4/5
Robust to scale changes	0/5	0/5	0/5	0/5	4/5	4/5	5/5	5/5	4/5
Textured foreground	3/5	3/5	2/5	2/5	3/5	2/5	2/5	2/5	2/5
Distraction free	2/5	2/5	1/5	1/5	3/5	3/5	2/5	2/5	1/5
Robust to lighting changes	2/5	2/5	1/5	1/5	2/5	1/5	1/5	2/5	2/5
Robust to scale changes	1/5	2/5	1/5	1/5	0/5	1/5	2/5	2/5	2/5
Speed (pixels/sec)	250	250	220	220	350	350	500	500	500

Figure 3. Method numbers refer to equations given above. Methods suffixed by an "a" match frames with the original frame, "b" methods match frames with the last frame.

for the SSD match resulted in a tracker that did not avoid distraction reliably.

Texture proved to be difficult to track overall for three reasons. First, small motions in the direction parallel to the contour often result in great changes in the image. Secondly, periodic textures create a type of aperture problem for the foreground matcher — it is difficult to know where the foreground begins, when several matches seem equally likely. Lastly, because texture often contains pixel-width features, even SSD matches were poor when such features moved sub-pixel distances.

In the end, we selected Method 4a (Coarse SSD) with $k = 1$ as our algorithm for finding the foreground edge because it applies best to the assumptions given in Section 3.2. Coarse SSD corresponds to matching a constant valued region immediately inside of the contour edge to the new image, and it allows for scaling and shearing effects by performing the SSD computation not on the image itself, but rather on the mode values of entire regions. This approach provides a convenient compromise between simple edge tracking and correlation-

based tracking, giving the algorithm the robustness of edge tracking with the specificity of correlation methods. Greater values of k seemed to degrade the value of Method 4 because the existence of edges was not very reliable from frame to frame. In particular, soft changes in intensity resulted in occasionally detected edges, ruining potential matches. On the other hand, raising the threshold for edge detection weakened the algorithm when ambient illumination was low.

We stress, ultimately, that the type of algorithm used should depend on the environment and the types of object tracked, and that at a purely local level, it is difficult to arrive at one, good, general-purpose tracking algorithm.

5 Implementation and Results

We now describe how Coarse SSD is employed in two real tracking applications. These applications are separated into two levels, where the lower level deals with local tracking and feature matching, and the upper level directs window placement and computes the state of the entire object. See [6] for more information on layered feature tracking.

5.1 Low Level Tracking

The algorithm chosen above is incorporated into a 1-dimensional edge detector. Each window is simply a single line of pixels which monitors a part of the contour. The position and orientation of the windows (the $\mathbf{w}$'s) are chosen by the top level (see Section 5.2), and the width was determined empirically.

Equation 4 is implemented in a straight-forward manner. Edges are found by convolving a 1-dimensional discrete derivative mask with the line of pixels in a window and looking for points with gray-scale gradient above a threshold (See [6]). Regions between edges and window boundaries are identified by computing the mode of gray-scale values for each region

to within 2 percent of the range of possible values. Our foreground-finding computation then finds the minimum SSD between the mode value of the region immediately within the previous edge and the new image.

During successful contour-tracking, the strip of pixels will be split into two halves, where one half is the image of the foreground object, the other half is part of the background, and the boundary between the halves is the edge itself.

In extreme cases, when there seems to be no continuity from frame $\mathbf{b}_{i-1}$ to frame $\mathbf{b}_i$ (when the minimum correlation is above a certain threshold), the window gives up trying to interpret the new image and signals that its return value is unreliable. This is an indication to the top level that the bottom level is unable to offer any advice about the location of the contour component. Such a situation should not occur if sampling assumptions are not violated, but it occurs fairly frequently in practice.

5.2 High Level Tracking

We believe that the low-level tracker will provide additional robustness to many existing constraint- or model-based trackers. In order to show how easily it can be incorporated into existing trackers, we use it as the basis for distraction-free line trackers and snake-like contour trackers.

The top level makes two choices. The first choice is to select the parameters for the window which the bottom level processes. The second is to guess, when the bottom level provides insufficient knowledge about the location of a tracked component, where the edge lies. In some applications, other tracked components may give strong hints about the location of the unknown component. In other instances, global constraints may be sufficient.

In our implementations, we make only minimal global assumptions so that the power and the limitations of wholly local tracking are evident.

5.2.1 Lines

We use many low-level trackers arranged linearly to track a whole line segment. The constraint in line or edge tracking is simply that the expected feature is a line segment. Thus the top level must balance the competing forces of individual low-level trackers with the straightness of the line.

The tracking scheme is simple: First, windows are placed by the top level; then, the low-level components search independently for their edges; and finally, line-fitting constraints are applied, components are adjusted to lie on the fit, and the cycle is repeated.

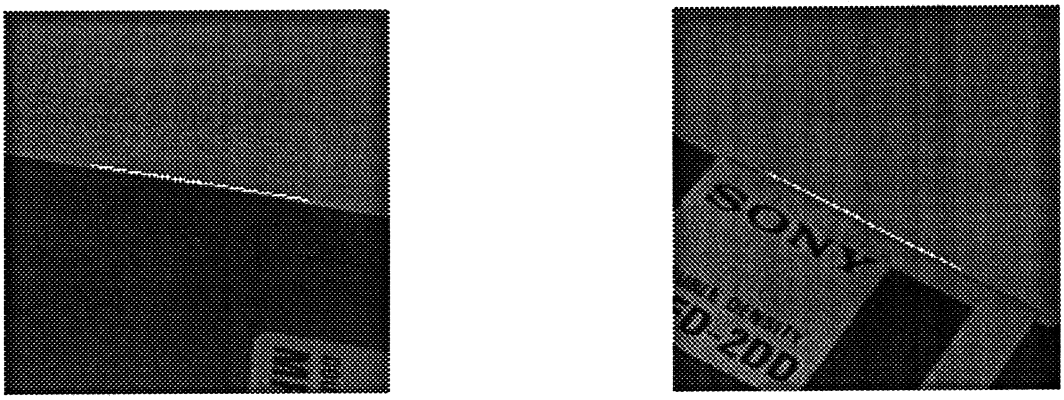
We use approximately one third as many component trackers as the length in pixels of the desired line segment. Components are indexed in order from one endpoint to the other. The center of the line is determined by the geometrical center of mass of all of the component edges. The orientation of the line is determined by taking the average slope of a set of lines determined by two valid feature trackers:

$$\theta = \frac{\sum_{i=1}^{3k/4} \text{atan}_2(y_{i+k/4} - y_i, x_{i+k/4} - x_i)\text{valid}(i)}{\sum_{i=1}^{3k/4} \text{valid}(i) \cdot \text{valid}(i + k/4)}$$

where k is the total number of components, $\text{atan}_2(y, x)$ returns a signed inverse tangent value for y/x, and $\text{valid}(i)$ returns 1 if the low-level tracker's output is reliable and 0 otherwise. Component window positions are placed on equispaced points along a line with slope equal to $\tan(\theta)$ and centered on the previously computed center of mass. Orientations of all components are set at θ. These new tracker window parameters are used as the initial parameters for the next round of tracking.

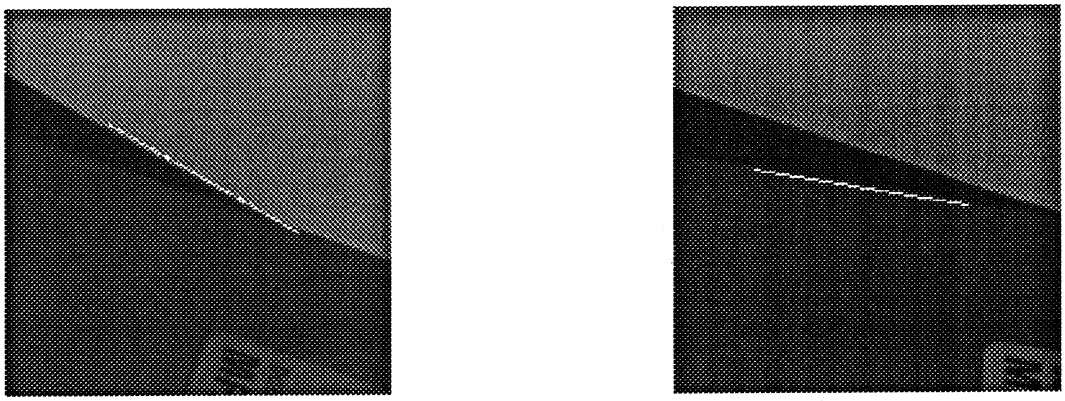
Figure 4a shows an initialized line. The diskette at the bottom is the foreground object we are trying to track. In Figure 5a, we see how a simple edge-tracker is easily distracted by another, stronger edge. Figure 5b shows how our tracker remains unperturbed by the apparent distraction.

In a different set of examples, we show how an SSD-based edge-tracker fares well for simple distraction, but not as well when shearing effects are added. Figure 4b shows the



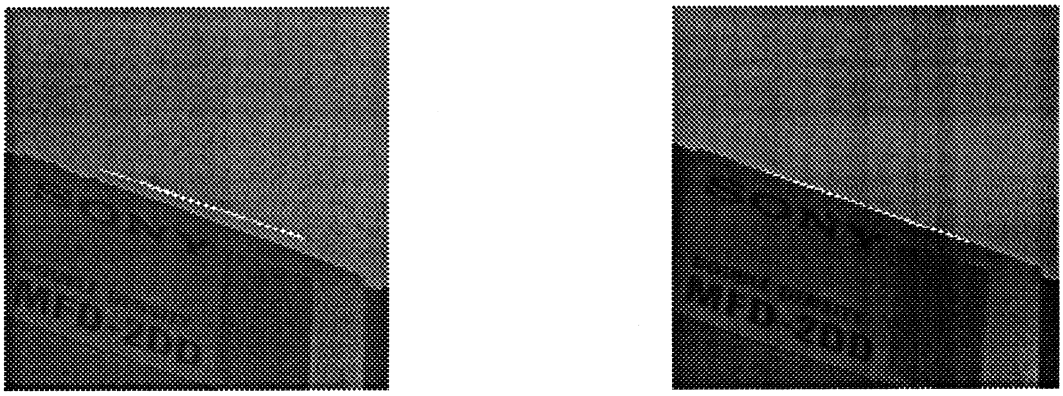
(a) (b)

Figure 4. Initialized lines for use with comparison against (a) simple foregrounds and (b) slightly textured foregrounds.



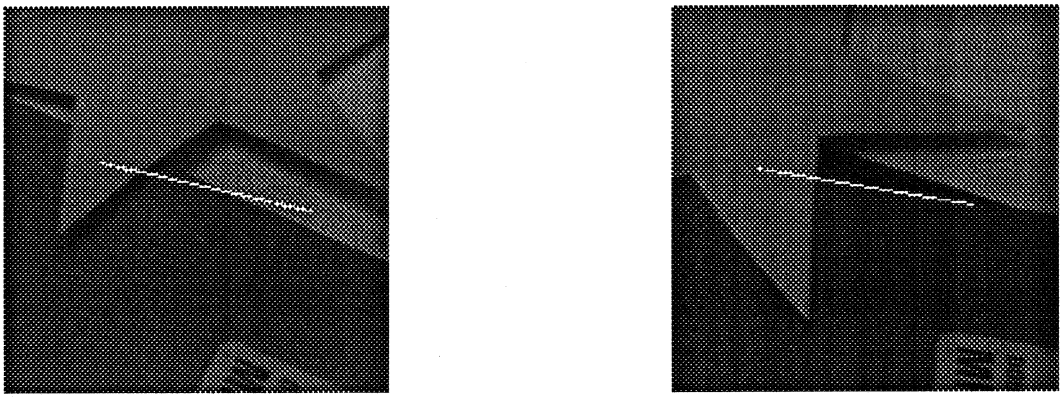
(a) (b)

Figure 5. (a) A distracted line and (b) an undistracted line.



(a) (b)

Figure 6. (a) A mistracked line and (b) a correctly tracked line.



(a) (b)

Figure 7. Partially occluded line (a) mistracked by a simple edge detector and (b) successfully tracked.

initial configuration. Figure 6a shows the SSD algorithm failing for slight shearing and rotation. In Figure 6b, our algorithm performs well in the same situation. We note also that SSD-based trackers cannot track roughly constant-valued foregrounds, such as that seen in Figure 4a, at all.

The method we use to compute the slope of the line actually enables the line to deal well with partial foreground *occlusions* as well as distractions. In Figure 7b, we see a part of the line segment occluded without any adverse effect on line tracking. A simple edge-tracker becomes confused by such an image, because of it's rudimentary objective function (Figure 7a).

5.2.2 Contours

Our contour tracker is slightly more complex, but uses no shape models, either. The only geometric constraint is that the tracked object is a simple, closed loop. Our snake-like tracker takes elements of many previous contour trackers [1, 3, 8, 12, 14].

To determine the position of new search windows, we compute a weighted combination of predicted positions and spatially interpolated positions to calculate the parameters, $\mathbf{w}$, of the search window for the lowest level. The coordinate and orientation elements of this vector also serve as the best prediction for the new location of the contour component edge. The top level chooses between the coordinate and orientation of $\mathbf{w}$ and the suggestion made by the bottom level. If the bottom level is sure of its recommendation, the edge position is updated to the recommended position. If the bottom level can give no hints for the component position, then $\mathbf{w}$ is used as the location of the edge.

The initial position is computed as follows: Three independent estimates are computed first. Given the prior history of a tracked component, where a history is an array of two previous positions and orientations, the new position is then extrapolated for the next time instant using linear extrapolation (this is an instantiation of a minimal dynamic model

that assumes smooth velocity transitions — it is unnecessary for distraction-proof tracking). Because we make no concrete assumptions about the motion, we do not use a Kalman filter for position or velocity. Rather, we discard completely the information previous to the third frame before the current frame. Next, a spatial interpolation process occurs for each tracking component. Neighboring components are fit with a least-squares algorithm to a second-order spline, and a middle position is interpolated. The temporally extrapolated position (which is independent of spatial neighbors) and the spatially interpolated position (which is independent of history), are combined as a weighted average, and this final value is used as the position of the center of the search window for the bottom level.

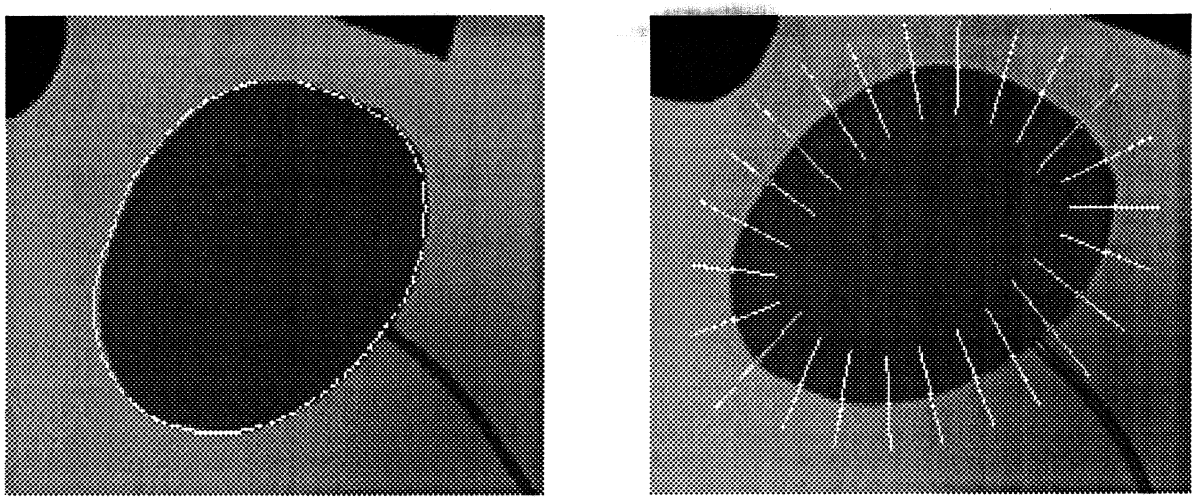
The final position is either the initial position estimated by this last procedure, or the best recommendation by the bottom level, given the initial search window.

Lastly, given the final positions of each feature, cubic splines connect the features to give a final estimate for the entire contour.

Figure 8 displays images immediately after a contour tracker is initialized. Figure 8(a) shows the cubic spline determined by the knot points shown in Figure 8(b). The silhouette of the gray ovals are the intended targets for Figures 9 through 12.

We show in Figures 9 and 10 the problem with many edge-based contour trackers. In each instance, edges more prominent than the original edges enter into one or more search windows, effectively distracting them from their intended targets. Distraction occurs because individual components are simply high-gradient edge finders, and depend upon high-level models to correct them in the case that they stray. While many model- or template-based contour trackers, such as those described in [2], might not be distracted by the high-contrast edges in Figures (a) and (b), they would still fail in cases (c) and (d), where the change in the shape of the contour is small and/or gradual.

Figures 11 and 12 illustrate distraction-free tracking. By simply maintaining some information about the kind of edges that are tracked (i.e., what intensities are observed inside



(a) (b)

Figure 8. Initial tracking showing, (a) a cubic spline contour, and (b) the search windows. The short line segments represent the search window of each tracking component, with estimated edge location (and consequently, spline knot points) at their midpoints.

the contour), tracking is improved greatly. Without recourse to any global models for shape, individual search windows can remain on target, even when what might be considered more "attractive" edges enter the search space.

As expected, our trackers fail when our assumptions are not valid. Sudden illumination changes cause each local tracker to be in doubt about the new foreground, at which point our tracker deteriorates to a simple edge-based tracker and becomes prey to future distraction (performance, however, is no worse than for standard edge-based trackers). General rotations of objects with non-constant brightness cause local trackers to maintain track of specific patterns on the surface, but not of the entire occluding contour. And finally, distraction by objects that have the same intensity as the inside of the contour cause local trackers to lock onto the distracting edge. These problems cannot be avoided by local tracking schemes, and in some contrived instances, probably cannot be avoided at all.

Finally, in Figure 13, we show how a book remains accurately tracked even as significant distraction occurs in the background. We note that no specific dynamic models or shape models for the book have been used. The images were taken approximately 15-20 frames apart, at a tracking rate of 20Hz for 35 search windows of 40 pixel widths each. Similar results can be obtained no matter how quickly distractions in the background occur.

6 Discussion

Our original intent was to create generic contour trackers, which could track occluding contours without predefined models and without being prone to background distraction. Focusing on what individual feature trackers could do to avoid distraction, we arrived at the conclusion that although local information is surprisingly limited and that some sort of global perspective greatly enhances tracking, there is also much that can be done at the local level to improve reliability.

There seem to be several different avenues for subsequent research. One is to explore the

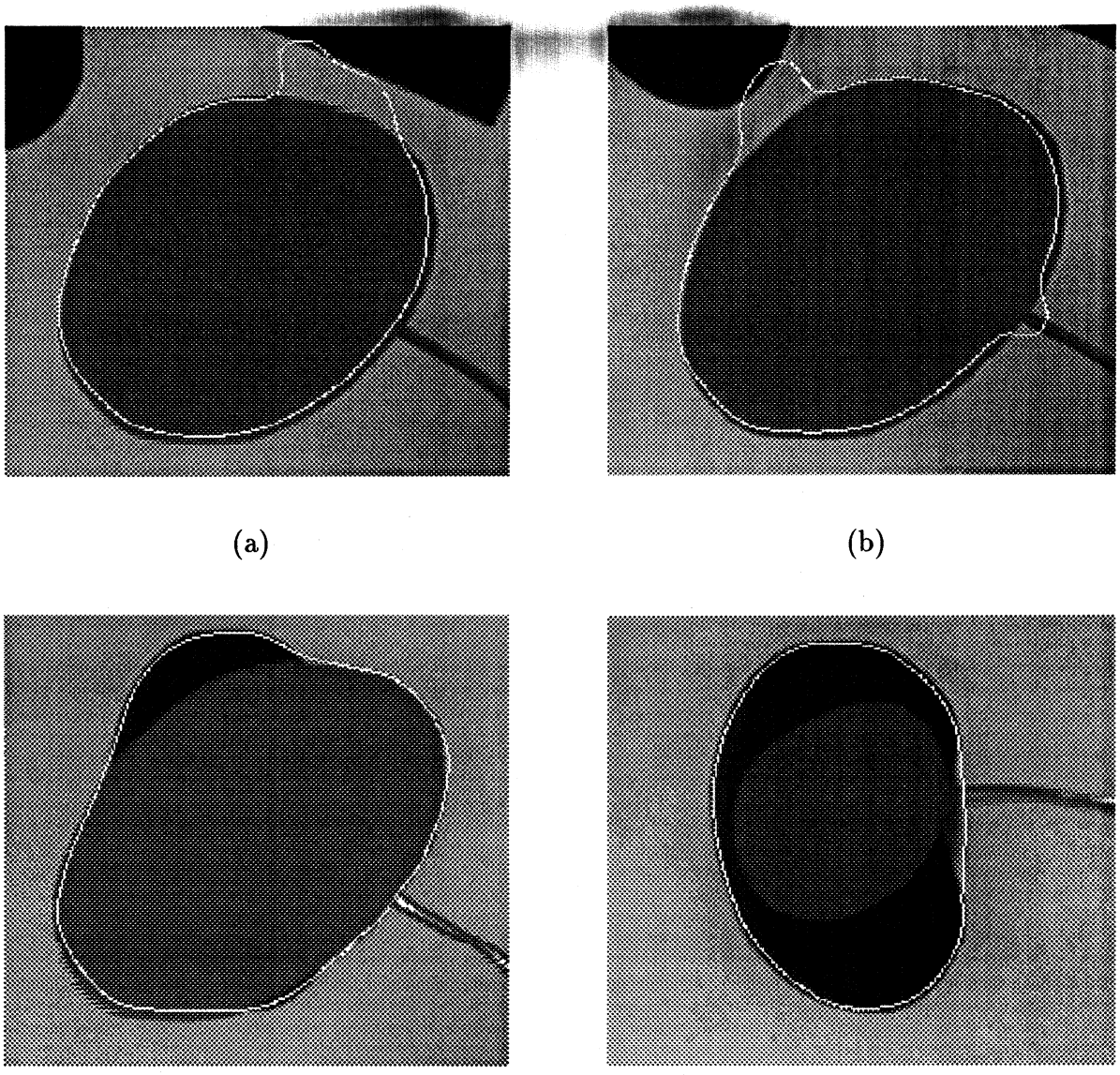


Figure 9. Instances of distraction and consequent mistracking.

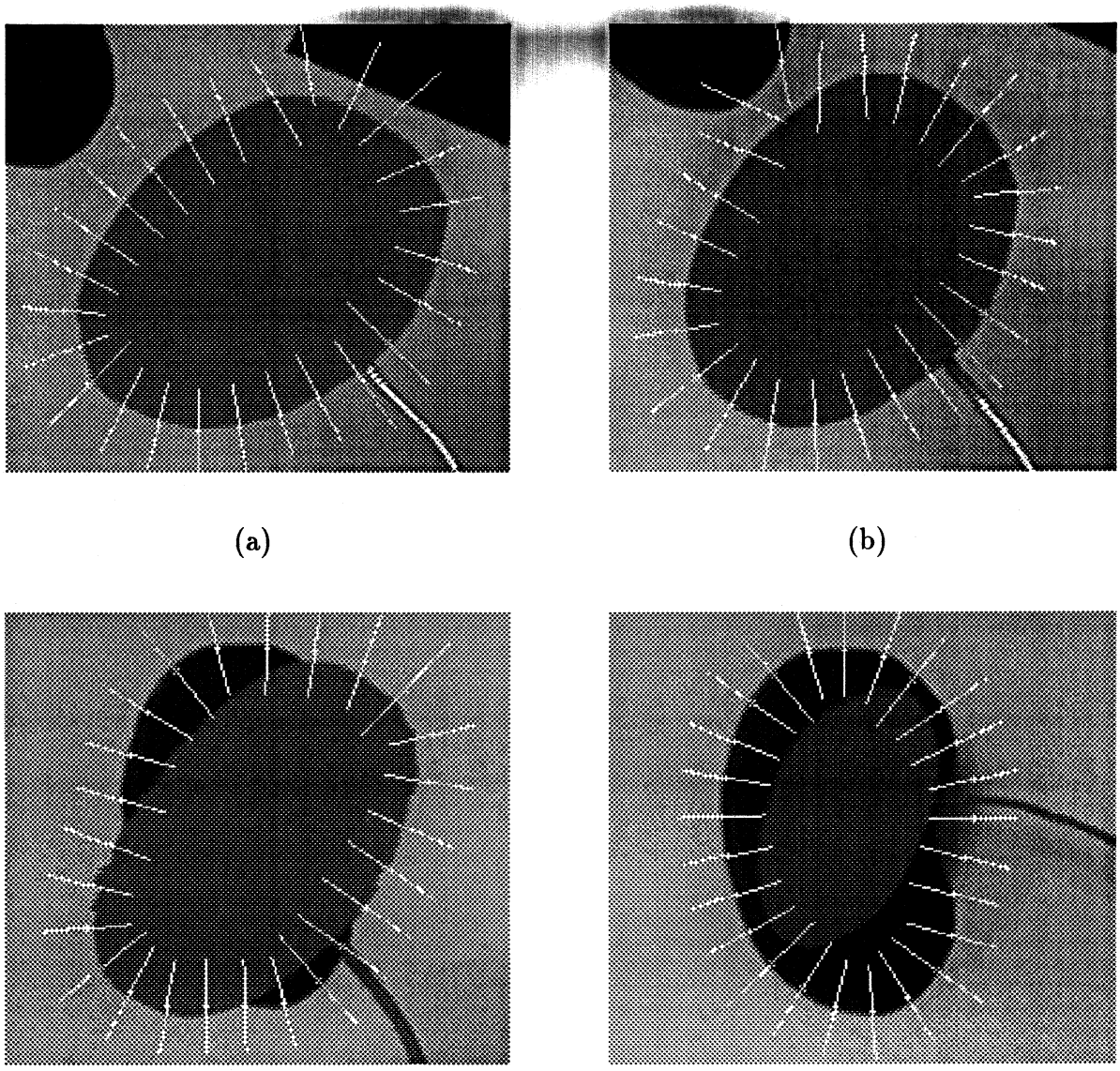


Figure 10. What's really happening during distraction: some windows discover more "attractive" edges.

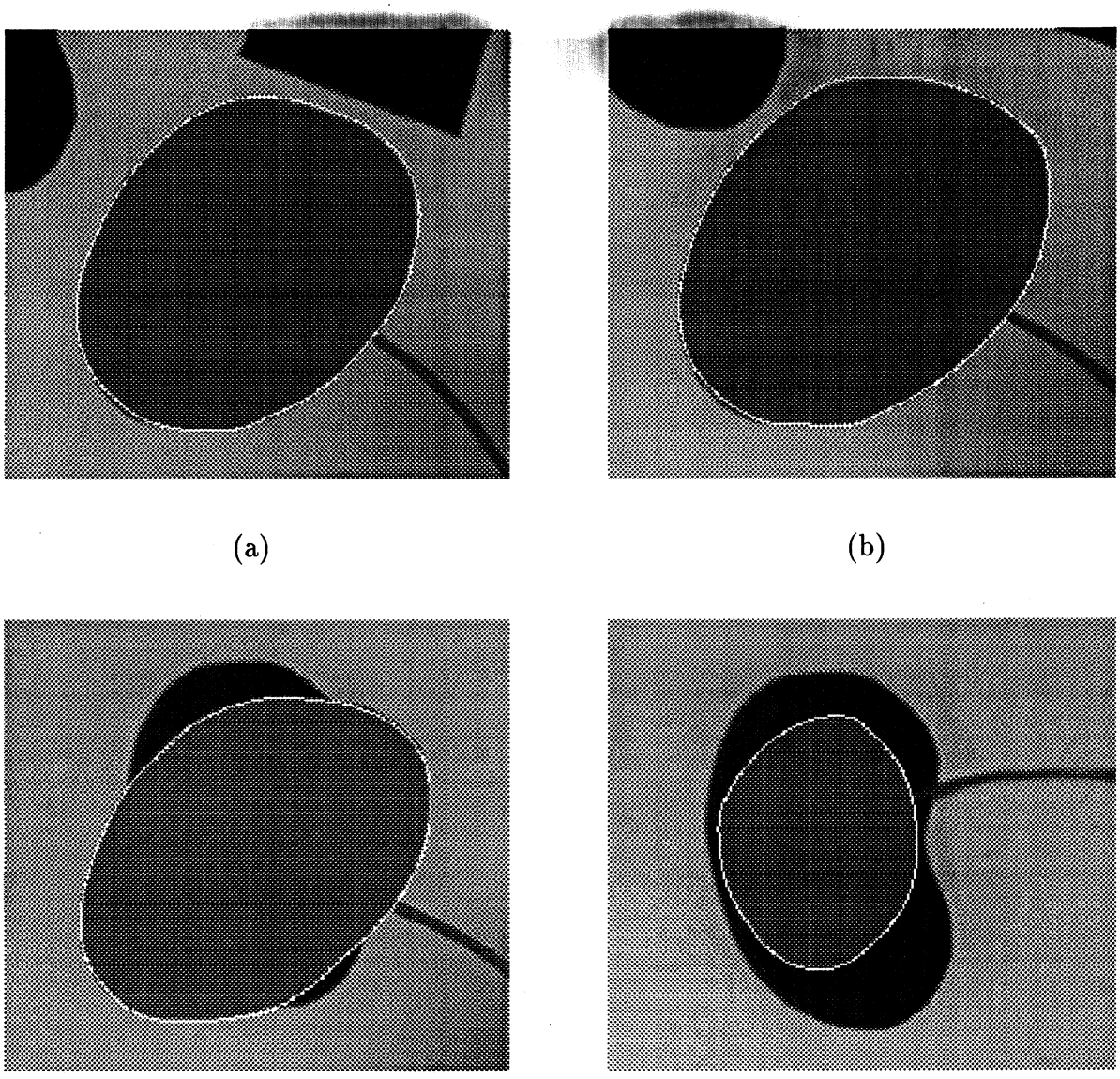
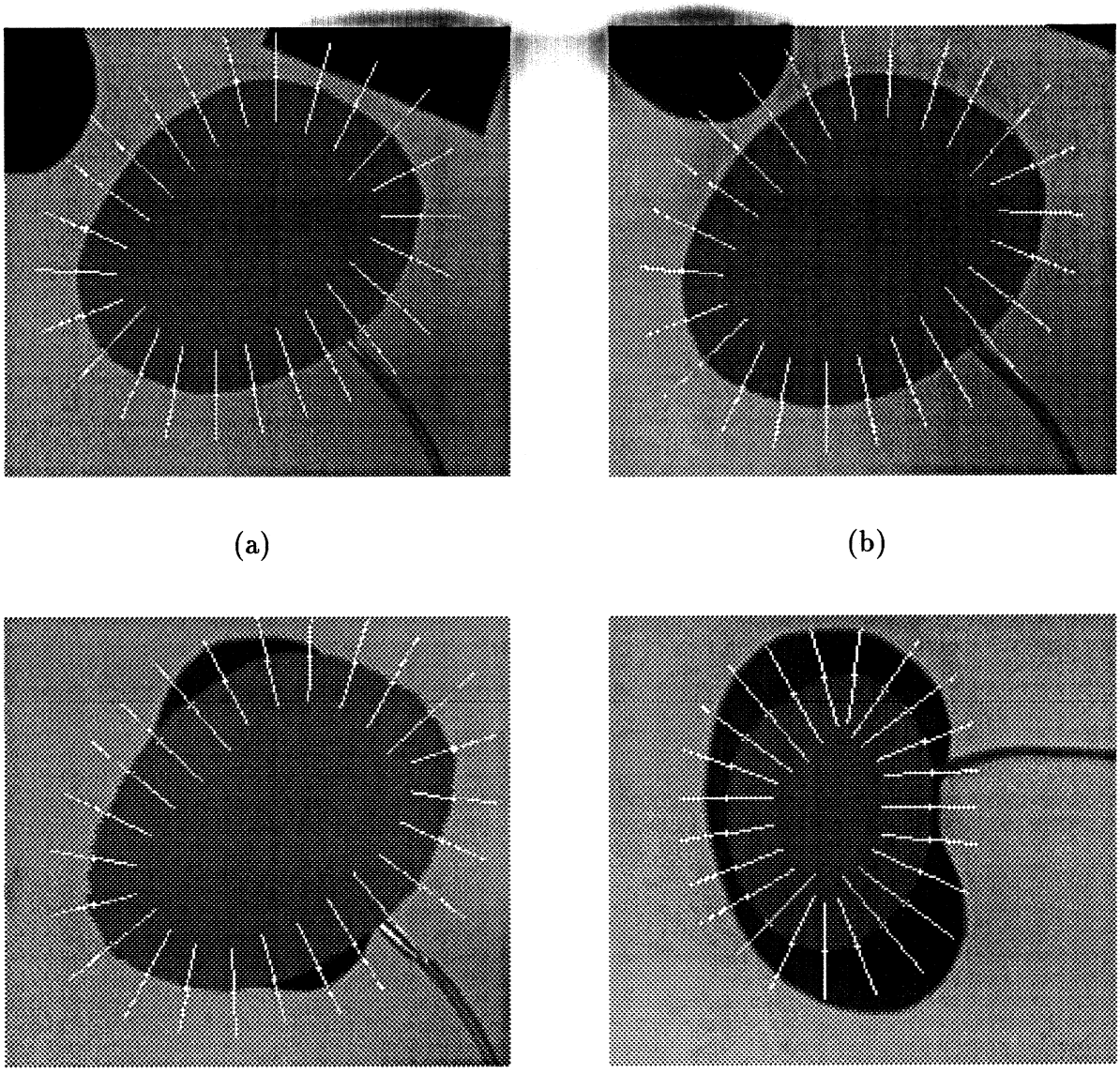


Figure 11. The fix.



(a) (b)

(c) (d)

Figure 12. Some search windows include potential distractors, but they are ignored.

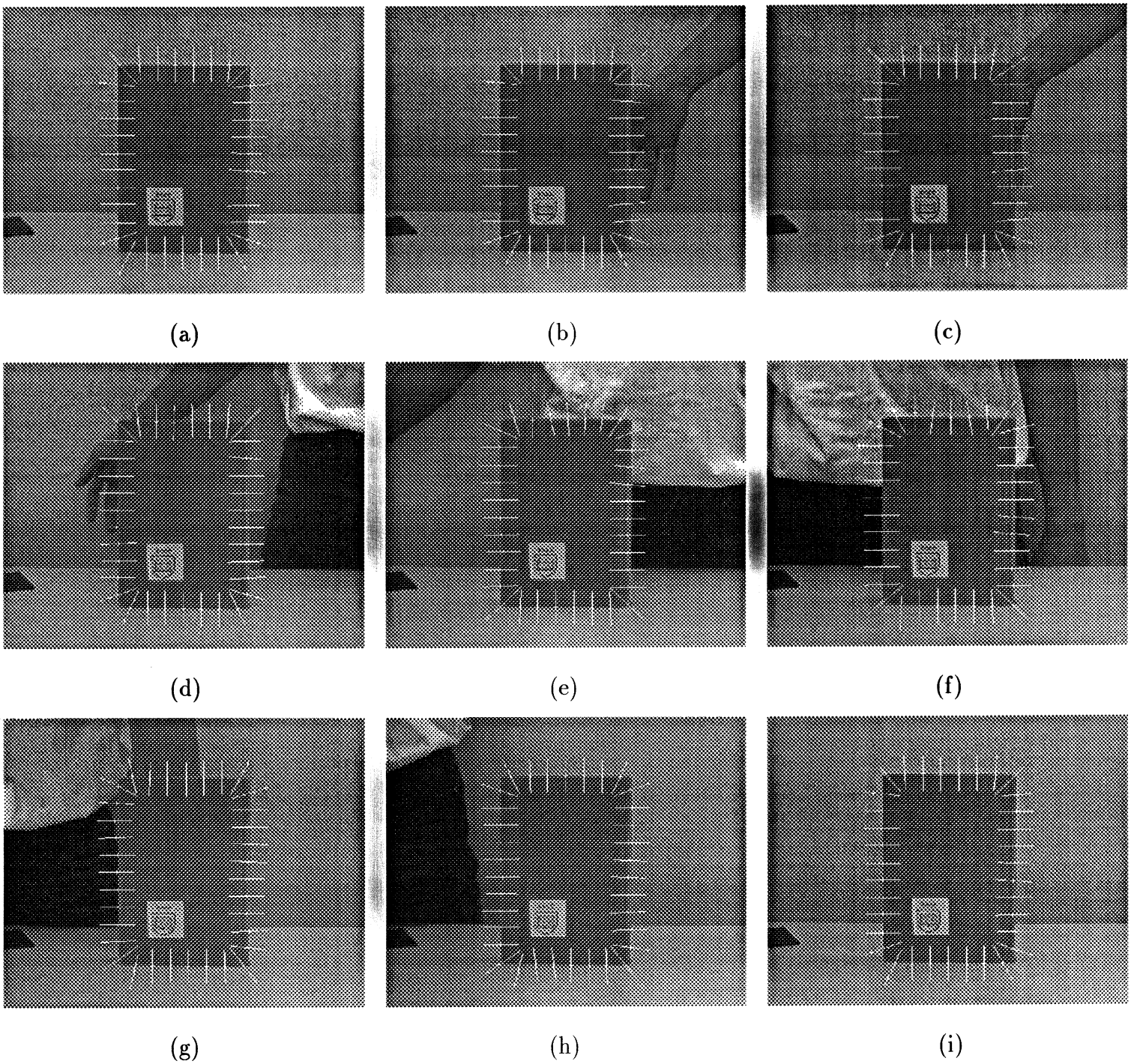


Figure 13. A contour of a book undistracted by movement of strong edges in background.

possibilities of distraction-free tracking using more information from a broader perspective: Can generic contour trackers be improved with additional high-level knowledge, yet without models? Another avenue is to combine model-based tracking with the work presented in this paper. We anticipate that local methods incorporated into model-based trackers will result in very robust trackers. Yet another consists of doing more careful work on how best to match foregrounds with new images; our treatment of this problem barely scratches the surface, because it ignores the possibility of tracking small 2-dimensional patches of foreground. Finally, we are interested in the theoretical limitations of local feature tracking — can we prove that isolated window information alone is insufficient to solve various tracking problems?

Visual distraction, though difficult to define objectively, is a common problem for most motion trackers. In some sense, tracking accurately in the face of distraction is really only better tracking with a keener objective function. Yet, specifying the target of a feature tracker too narrowly results in a significant decrease in robustness, and overly generic descriptions of the target result in a tendency to be distracted by similar features. We have presented a method that combines the specificity of correlation methods with the robustness of edge-based trackers. Our method, without the use of global shape models or dynamic models, performs well in all cases where local, window tracking alone can be expected to track without distractions. Integration of our algorithm with model-based tracking should result in highly robust, distraction-free tracking of occluding contours.

Acknowledgements: This research was supported by DARPA grants N00014-91-J-1577 and N00014-93-1-1235, and by National Science Foundation grants IRI-9109116 and DDM-9112458.

References

[1] A. A. Amini, S. Tehrani, T. E. Weymouth. Using dynamic programming for minimizing

the energy of active contours in the presence of hard constraints. In *Proceedings, 2nd Int'l Conf. on Comp. Vision*, pages 95–99, 1988.

[2] A. Blake, R. Curwen, and A. Zisserman. Affine-invariant contour tracking with automatic control of spatiotemporal scale. In *Proceedings, Int'l Conf. on Comp. Vision*, Berlin, Germany, May 11-14, 1993, pp. 421–430.

[3] L. D. Cohen. On active contour models and balloons. In *CVGIP: Image Understanding*, 53(2):211-218, March 1991.

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

[5] O. D. Faugeras. Three-Dimensional Computer Vision. Cambridge, MA: MIT Press, 1993.

[6] G. Hager, S. Puri, and K. Toyama. A framework for real-time window-based tracking using off-the-shelf hardware. Yale Tech Report: Yale-TR-988, 1993.

[7] J. Huang and G. Hager. Tracking tools for vision-based navigation. Submitted to *Int'l Conf. on Intel. Robot and Sys.*, 1995.

[8] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: active contour models. In *Int'l J. of Comp. Vision*, 1:321–331, 1987.

[9] D. G. Lowe. Robust model-based motion tracking through the integration of search and estimation. In *Int'l J. of Comp. Vision*, 8(2):113–122, 1992.

[10] N. P. Papanikolopoulos, P. K. Khosla, T. Kanade. Visual tracking of a moving target by a camera mounted on a robot: a combination of control and vision. In *IEEE Trans. on Robotics and Automaton*, 9(1):14–35, February, 1993.

[11] A. A. Rizzi and D. E. Koditschek. An active visual estimator for dexterous manipulation. In *IEEE Int'l Conf. on Robotics and Automaton*, 1994.

[12] D. Terzopoulos and Szeliski. Tracking with Kalman snakes In *Active Vision*, ed. A. Blake and A. Yuille. Cambridge, MA: MIT Press, 1992.

[13] C. Tomasi and T. Kanade. Detection and tracking of point features. Carnegie-Mellon Tech Report, CMU-CS-91-132, April, 1991.

[14] D. J. Williams and M. Shah. A fast algorithm for active contours and curvature estimation. In *CVGIP: Image Understanding*, 55(1):14–26, January 1992.