Incremental Focus of Attention for Robust Vision-Based Tracking

Kentaro Toyama and Gregory Hager
Department of Computer Science
Yale University
P. O. Box 208285
New Haven, CT 06520-8285
Phone: (203) 432-6432
E-mail: toyama@cs.yale.edu, hager@cs.yale.edu

Abstract

We present the *Incremental Focus of Attention* (IFA) architecture for robust, adaptive, real-time motion tracking. IFA systems combine several visual search and vision-based tracking algorithms into a layered hierarchy. The architecture controls the transitions between layers and executes algorithms appropriate to the visual environment at hand: When conditions are good, tracking is accurate and precise; as conditions deteriorate, more robust, yet less accurate algorithms take over; when tracking is lost altogether, layers cooperate to perform a rapid search for the target in order to recover it and continue tracking.

Implemented IFA systems are extremely robust to most common types of temporary visual disturbances. They resist minor visual perturbances and recover quickly after full occlusions, illumination changes, major distractions, and target disappearances. Analysis of the algorithm's recovery times are supported by simulation results and experiments on real data. In particular, examples show that recovery times after lost tracking depend primarily on the number of objects visually similar to the target in the field of view.

Submitted to IJCV special edition on vision at Yale.

Keywords: visual tracking, real-time vision, face tracking, robust tracking.

1 Introduction

Biological visual systems exhibit an amazing robustness to complex visual events. The human visual system, for example, is able to adapt to or recover from many unexpected visual circumstances. On one hand, it can acquire partial information about an object if it is at all visible; on the other hand, it can reacquire an object that is temporarily lost from view. Thus, athletes can still catch, hit, or kick a ball by knowing its approximate position even when it is spinning rapidly, and motorists can quickly recover track of the vehicle ahead of them, even after a glance in the mirrors.

As we move vision systems from the structured laboratory to the "real world," we must endow them with this ability to cope with unexpected or unmodeled situations. Robotic hand-eye systems must be able to track grasped objects even if they are dropped, vision-based human-computer interfaces should not be bothered by a sneezing subject, and automated driving vehicles must remain aware of the road even if it becomes momentarily obscured by snow or dirt. In biological systems, this kind of robustness is taken for granted, yet in computer vision it is an issue that has received relatively little attention.

Incremental Focus of Attention (henceforth, IFA) begins to fill this gap by providing a design methodology for developing robust motion tracking systems. Conceptually, IFA is a framework for organizing multiple tracking algorithms and search heuristics into robust systems. During execution, IFA efficiently focuses the "attention" of the tracking system onto relevant parts of the image. The result is robustness in two senses. First, if an IFA system is composed of several trackers of varying accuracy, failure of any specific tracking algorithm usually means that another, less precise algorithm takes over. Second, when visual perturbations temporarily cause a tracking system to lose its target, IFA provides the possibility of reinitialization and recovery. In this way, IFA tracking systems gracefully degrade as visual conditions deteriorate and recover when conditions improve.

This article describes the development of the IFA architecture for efficient and robust vision-based tracking. In the next section, we formulate the tracking problem in more precise terms and develop necessary terminology. In Section 3, we give an overview of related work. Section 4 describes the construction of IFA systems. Their correctness and robustness properties are discussed in Section 5. Finally, Section 6 discusses the implementation of several IFA tracking systems and offers experimental results which justify IFA's claims to robustness.

2 The Robust Tracking Problem

In *vision-based object tracking*, the goal is to determine specific measurable attributes of a *target* from a sequence of temporally ordered images. Tracking problems can be as simple as determining the vertical position in image coordinates of a bouncing ball or as complex as computing the instantaneous 3-D kinematic configuration of a walking person. Throughout this article, we assume that the value of these quantities at a given time point t is given by a d-dimensional *state* vector, $\mathbf{x}(t) \in \mathcal{X} \subset \Re^d$. We further assume that $\mathcal{X}$, the *state space* of the system, is bounded and therefore of finite volume. Then, the *vision-based tracking problem* is defined to be the calculation of a series of estimates $\hat{\mathbf{x}}(t)$ of the actual state, $\mathbf{x}^*(t)$ of the target at time t. For the remainder of the article, we assume the dependence of $\mathbf{x}^*$ and $\hat{\mathbf{x}}$ on t without explicitly noting it.

We view vision-based tracking as a repeated search and estimation process, where the search occurs *in the state space of the target, not in the image.* The input to a tracking algorithm is an

input configuration set, or a set of candidate target states, $\mathbf{X}^{\text{in}} \subseteq \mathcal{X}$, together with an image, I. The output at each time step consists of an *output configuration set*, $\mathbf{X}^{\text{out}} \subset \mathcal{X}$, such that $\mathbf{X}^{\text{out}} \subset \mathbf{X}^{\text{in}}$, where $\mathbf{X}^{\text{out}}$ should include $\hat{\mathbf{x}}$. The output set, $\mathbf{X}^{\text{out}}$, of an algorithm is also a representation of the algorithm's *margin of error*. Thus, we have chosen the margin of error to be a set of states rather than, for example, a statistical distribution over states. Note that for actual implementation, neither input sets nor output sets are necessarily explicit. An algorithm may return a single state, for example, but such output together with the known margin of error can be interpreted as a set of states.

By introducing an explicit margin of error, we can precisely define several relevant terms. We say that tracking is *accurate* if and only if $\mathbf{x}^* \in \mathbf{X}^{\text{out}}$. *Mistracking*, or tracking *failure* occurs when $\mathbf{x}^* \notin \mathbf{X}^{\text{out}}$. *Precision* is related to a measure of the size of the error margin, denoted $|\mathbf{X}^{\text{out}}|$. The simplest formulation of precision is that it relates inversely with the volume of states occupied by the error margin. Under this formulation, algorithms which return large margins of error are less precise than algorithms with smaller margins of error. In addition, the dimensionality of the state information computed by an algorithm affects its precision. For example, a tracking algorithm which can determine the position *and* orientation of a face is more precise than an algorithm which determines only the position: the former has a smaller output set than the latter, which includes sets ranging over all possible orientations.

As noted above, we concentrate on the problem of *robustness*. Robustness is the ability of a vision-based tracking system to track accurately and precisely during or after visual circumstances that are less than ideal. The open-ended nature of these situations is what makes robustness elusive: Changes in ambient illumination disrupt intensity-based tracking schemes; distractions in the background draw attention away from the true target; foreground occlusions obscure a target's appearance; erratic motion makes prediction unreliable; and the target itself may undergo visual changes not taken into account by the algorithm designer. The *robust vision-based tracking problem* is, therefore, a vision-based tracking subproblem – the problem of coping with a complex environment.

3 Previous Work in Robust Tracking

The existing literature on robust tracking can be broadly categorized into research that contributes to either *ante-failure* or *post-failure* robustness [42]. Ante-failure robust systems seek to avoid tracking failure altogether through specialized algorithms that anticipate visual disturbances and attempt to track despite them. Post-failure systems, on the other hand, accept the inevitability of mistracking and are designed to recover from failure once it happens. It seems clear that both ante-failure and post-failure means are necessary for reliable tracking, but research thus far has been heavily weighted towards the former.

Advances in ante-failure robustness are usually achieved through robust statistics, temporal filtering, or *ad hoc* methods for handling specific types of visual perturbations. For instance, distractions – objects which are close to the target both in appearance and state – can be handled in several ways. One method is to consider only a small set of states surrounding the target [15, 47]. This requires some predictability in the target trajectory but eliminates the need to examine the entire image. Another way to handle distraction is through foveation, effectively blurring the image region around the target [6, 37]. Finally, a tracking system may filter output estimates based on expected motion and noise [1]. Each technique avoids distraction by ignoring or filtering out

competing object states which are unlikely to be the target.

Other visual disturbances have other solutions. Changes in ambient lighting have been handled by concentrating on color cues [35, 50], by tracking based on edges [13, 24, 26], or by explicitly modeling illumination parameters [20]. Fast or unpredictable motion requires combinations of faster hardware, full-frame processing [5, 30], or probabilistic dynamics [22]. Occlusions, where opaque objects intercept the camera's line of sight to the target, can be handled by robust matching [13, 14, 26, 25] or by dynamic state prediction [1, 36, 38].

Ante-failure work generally seeks to handle problems one at a time, but some researchers have sought to design systems that are ante-failure robust to many types of visual perturbations simultaneously. Notable among these are probabilistic methods, which sample and interpolate likelihoods over the state space [22], and sensor fusion techniques, which track based on multiple cues [9, 23, 32, 34, 41].

None of these efforts are error-free. As in other domains, ante-failure methods cannot eliminate failure entirely. Most often, the limitations are a reflection not of poor algorithmic design, but rather of the impossibility of perfect vision-based tracking in complex circumstances.

To deal with such situations, some researchers have incorporated post-failure schemes to recover tracking when mistracking occurs. Typically, post-failure techniques execute different algorithms depending on tracking success. Much of this work is inspired by *focus of attention* in biological systems. Cognitive science research in focus of attention suggests that biological vision systems are broadly organized into *pre-attentive* and *post-attentive* stages: The pre-attentive stage rapidly finds image subregions of interest on which to focus the attention of a post-attentive stage, which examines the attended region more closely [29, 43, 44, 49].

Among those interested in vision-based tracking by computer, many have incorporated knowledge gained from cognitive science and built tracking systems that manage two separate algorithms, where the first algorithm rapidly finds relevant candidate regions in an image and the second performs tracking. Most of this work focuses on a single means to find and track a target, using various cues: intensity [33], color [35], or motion [21, 28]. These methods are efficient, but do not take advantage of the multiple cues which identify real-world objects.

Some recent work extends this notion to using more than two stages or multiple cues for tracking. In face tracking, for example, some systems are able to robustly determine such things as the facial expression of the target in real-time [9, 32]. Another system is able to detect and track a limited set of vehicles using a three-stage focus of attention scheme [7]. Others, focusing on the information contained in multiple cues focus on fusing different types of image information to identify a unique target [27, 46].

IFA differs from prior work in several ways. First, rather than trying to build computational models of biological visual systems [11, 45], we are primarily concerned with synthesizing methods for vision-based tracking which are computationally efficient and robust. Second, IFA is not restricted to computing positional or directional information about a target, as are many active tracking systems which emphasize the control aspects of keeping an object in view [2, 8, 31, 33, 46]. Rather, IFA search may recover rotation, scaling, shear, and even full 3D pose information. Third, we pose tracking itself as a flexible multi-stage focus of attention rather than remaining within the standard two-stage paradigm [27, 31] or even a fixed k-stage architecture [7, 9, 32]. Lastly, our framework is not restricted to particular targets or specific cue types. Techniques such as eye-blink detection [9] or mouth localization [32] are specifically designed for frontal views of faces only and do not generalize for other target types. IFA, in contrast, allows utilization of a variety

of algorithms and thus confers robustness to many different target types.

Finally, we note that recent work in control of robotic systems lays a possible theoretical foundation for IFA. "Dynamical pick and place" tasks, in contrast to static pick and place tasks, require a robot to control the dynamics of an object in addition to controlling its position [3]. An example is the problem of using a flat, gripper-less, 3-DOF arm to catch a ball thrown into the robot's workspace, where "catching" means bringing the ball to eventual standstill on the bat. By *sequential composition* of locally convergent controllers, one for each subregion in the state space of the ball, the ball can be caught from almost any initial state [3, 4]: When the ball is falling at high speeds, the controller switches to bouncing control to slow the ball down. At lower speeds, the controller may switch to "palming" behavior, where the bat is able to maintain contact with the ball continuously without bouncing. Ultimately, the ball is caught and held still. A kind of sequential composition occurs in IFA, where different tracking algorithms take control depending on the dynamics of the target.

There are two reasons, however, why sequential composition is not adequate for the tracking problem. First, in tracking, the state of the tracked object is also the quantity to be determined, whereas in robot control, sensing is taken for granted and the result of computation is a command to take action. A circularity arises here – the very state that needs to be computed is what determines the method of computation. Secondly, as explained above, tracking can fail. The state of the target can at times be unknown, requiring a system that recognizes such situations and takes steps to recover from them.

IFA handles exactly these problems. It composes multiple tracking algorithms in a hierarchical scheme such that each algorithm acts as both a fallback procedure and a reinitialization process for the layers above (see Figure 1). Furthermore, the principles used to construct an IFA system are general and apply to a wide variety of tracking tasks.

4 System Construction

The design of an IFA system begins with an inventory of the tracking algorithms that are available for tracking a particular target. Algorithms are broadly classified as attention-focusing mechanisms (*selectors*) or as tracking mechanisms (*trackers*). The former can be thought of as heuristics which select regions of the state space to search while the latter track the target once it is found. These algorithms, called *layers* in the framework, are ordered by decreasing precision, with more precise algorithms at the top. At any given moment, processing occurs in a single layer. Successful tracking at a layer pushes control upward while unsuccessful tracking pulls control downward. Because one layer's output set is used as the next layer's input set, ascending layers gradually increase the precision of the system's state estimate. Finally, the system operates in either of two modes, indicating its "awareness" of tracking success. During `search` mode, the system has no definite state information about the target but actively searches the state space to find it. During `track` mode, the system asserts that it is tracking the target and can estimate part or all of the desired state information.

4.1 Layer Elements

The raw materials for IFA layers are tracking algorithms and visual search heuristics. In general, the more algorithms are available for finding and tracking a target, the better the final system, so

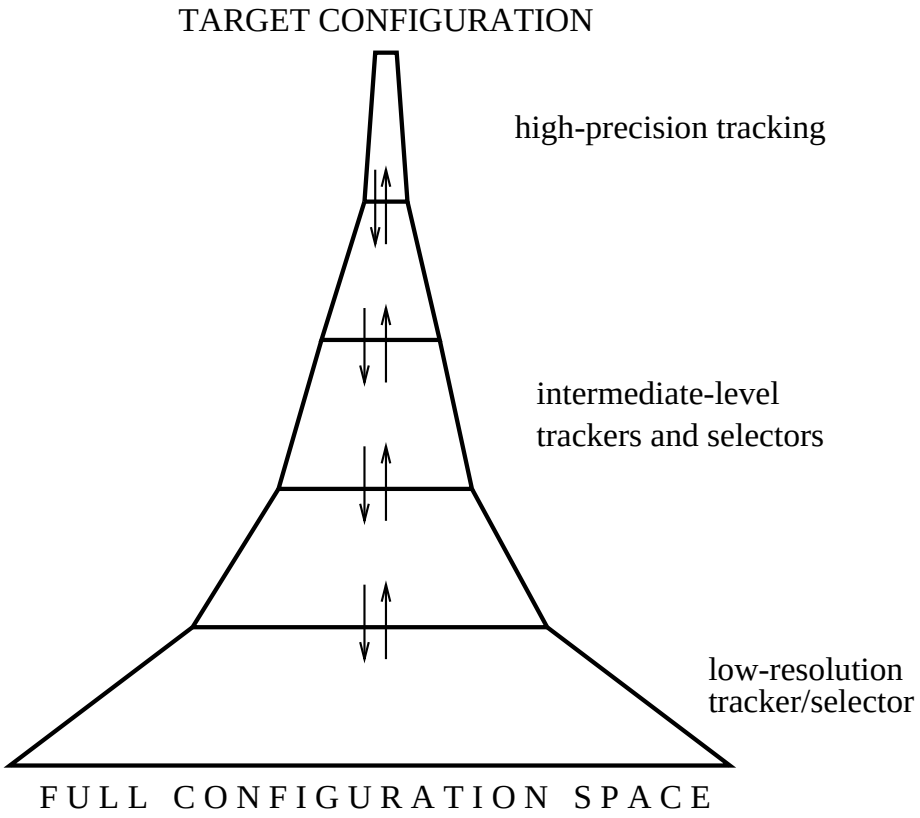


Figure 1. Schematic of a five-layer Incremental Focus of Attention (IFA) system. Tracking algorithms are represented as layers, where information flow (input and output sets) between algorithms is indicated by the arrows.

it is advantageous to explore and exploit every visual uniqueness of the target.

Given a set of visual searching and tracking algorithms, they must first be classified as either potential trackers or potential selectors. For the sake of correctness a clear distinction between trackers and selectors must be enforced. Trackers should be algorithms which almost always generate an output set that contains the target state given an input set which contains the target. If a tracking algorithm often tracks non-target objects, it may be more suitable as a selector. Conversely, if an attentional heuristic is based on a cue that is known to be unique within its input sets, it may be better suited as a tracker. An attention algorithm focusing on flesh-colored "blobs" could be a successful tracking algorithm if guaranteed that its input will only contain one flesh-colored object. The sensitivity of the surrounding task to mistracking and accuracy will also come into consideration.

In the following, let $\overline{\mathcal{X}}$ denote the set of all subsets of $\mathcal{X}$ and let $\mathcal{I}$ denote the set of all images. We model trackers and selectors as functions from state sets and images to state sets. For a given function f, $\mathrm{Dom}(f) \subseteq \overline{\mathcal{X}}$ and $\mathrm{Rng}(f) \subseteq \overline{\mathcal{X}}$ denote the domain of input state sets and range of output state sets of f, respectively. In all cases, we assume that $\mathrm{Dom}(f)$ covers $\mathcal{X}$.

4.1.1 Trackers

Formally, a tracker should fit the following definition as closely as possible (in the sections to follow, we discuss the effect of non-ideal trackers):

Definition 1. *An* idealized tracker *is a function,* $f : \overline{\mathcal{X}} \times \mathcal{I} \mapsto \overline{\mathcal{X}}$*, such that for all* $\mathbf{X} \in \mathrm{Dom}(f)$ *and* $I \in \mathcal{I}$*,*

1. $f(\mathbf{X}, I) \subset \mathbf{X}$.
2. If $\mathbf{x}^* \in \mathbf{X}$, then either $\mathbf{x}^* \in f(\mathbf{X}, I)$ or $f(\mathbf{X}, I) = \emptyset$.
3. If $\mathbf{x}^* \notin \mathbf{X}$, then $f(\mathbf{X}, I) = \emptyset$.

Together, Properties 2 and 3, which we refer to as the *filter criterion*, are a formalization of partial correctness. They state that trackers may return occasional false negatives, where a target which is present goes undetected, but never produces false positives, where a non-target is *hallucinated* although none exists. Thus, trackers must monitor their performance and report tracking failure. Usually, geometric constraints or thresholds can be set so that a tracker will report failure when appropriate. A tracker based on a wire-frame object model, for example, might report failure when a certain percentage of its model edges lack correspondences in the image. Although precise self-assessment is optimal, for the correctness of the algorithm, it is better for trackers to err on the side of conservativeness in their own estimation of success.

4.1.2 Selectors

Selectors are attention-focusing algorithms which are heuristic in nature and hence prone to returning sets not containing the target state:

Definition 2. *An* idealized selector *is a randomized function,* $g : \overline{\mathcal{X}} \times \mathcal{I} \mapsto \overline{\mathcal{X}}$*, such that for all* $\mathbf{X} \in \mathrm{Dom}(f)$ *and* $I \in \mathcal{I}$*,*

1. $g(\mathbf{X}, I) \subset \mathbf{X}$.
2. There is some $\epsilon_g > 0$ such that if $\mathbf{x} \in \mathbf{X}$, there is finite probability ϵ_g that $\mathbf{x} \in g(\mathbf{X}, I)$.

The purpose of selectors is to output manageable state sets for higher layers with a possible bias toward certain geometric or image-based constraints. For instance, a selector for face tracking will prefer *but not insist on* returning output sets which include states consistent with detected motion or flesh color. Thus, selectors return different output sets over time such that different portions of the state space are passed up the hierarchy with each call.

Finally, associated with each selector, g_i, are an *iteration index*, σ_i, and an iteration threshold, σ_i^{MAX}. The iteration index counts the number of times a selector is executed and elicits selector *failure* when it surpasses the iteration threshold (Section 4.4 describes when indices are incremented or reset). Monitoring of the iteration index prevents repeated, unsuccessful attempts to find the target in a region of the state space not containing the target. Actual σ_i^{MAX} values will depend on the reliability of the attention heuristic.

4.1.3 Set Dilation

One last adjustment is made to both trackers and selectors. Since targets are moving even as computation in layers takes place, the state sets output by layers must be adjusted in order to accommodate target movement. The union of potential adjusted output sets must be guaranteed to include the target state if the input set included it. In practice, this seemingly strict requirement is satisfied by *set dilation*, in which the initial output sets of a layer are simply expanded to include neighboring states as well. For the remainder of this article, $\mathbf{X}^{\text{out}'}$, will be taken to mean the output set after set dilation.

4.2 Layer Composition

In order to construct the system, all layers are sorted by output precision. Because greater precision often requires more processing, this means that faster algorithms will tend to occur at the bottom layers. Algorithms which are superseded by others in both speed and precision should be discarded.

Next, additional selectors are inserted wherever one layer's output is not precise enough to satisfy the input constraints of the layer above it. These selectors may be specially designed for the system and the environment, or they may be brute-force search algorithms which systematically partition a state space into subsets of the appropriate size. If the bottommost layer does not take the full configuration set as its input, a selector is added at the bottom. Any selectors that are more precise than the most precise tracker are discarded or converted into trackers. The appendix offers descriptions of some generic selectors.

Layers are then labeled from 0 to $n-1$ such that the topmost layer is $n-1$, and the bottommost is 0.

4.3 State Transition Graph

At the heart of an IFA system lies a deterministic finite state automaton (Figure 2(right)), in which transitions occur based on the success of layers. This automaton is constructed as follows:

- For each layer k ($0 \leq k < n$), we create a node N_k^{s}.
- Let m be the index of the first selector that occurs above a tracker. We create nodes N_j^{t}, for each layer j ($m+1 \leq j < n$). In Figure 2, $n = 5$, $m = 2$, layers 0 and 2 are selectors, layers 1, 3, and 4 are trackers, `search` nodes are on the left, and `track` nodes are on the right. The superscripts indicate the obvious correspondence of states with the two modes.
- For each k, $0 < k < n$, we create a *success link* (a directed edge) from node N_{k-1}^{s} to N_k^{s} and a *failure link* from node N_k^{s} to node N_l^{s}, where l is the first layer below layer k that is a selector. One more failure link is created from node N_0^{s} to itself.
- From N_{n-1}^{s}, the `search` node for the topmost tracker, we add a success link to N_{n-1}^{t}.
- For each j, $m < j < n$, we add a failure link from node N_j^{t} to N_{j-1}^{t}; and similarly, we add a success link from node N_{j-1}^{t} to N_j^{t}. One more success link is created from node N_{n-1}^{t} to itself.
- Finally, we add a failure link from node N_{m+1}^{t} to N_m^{s}.

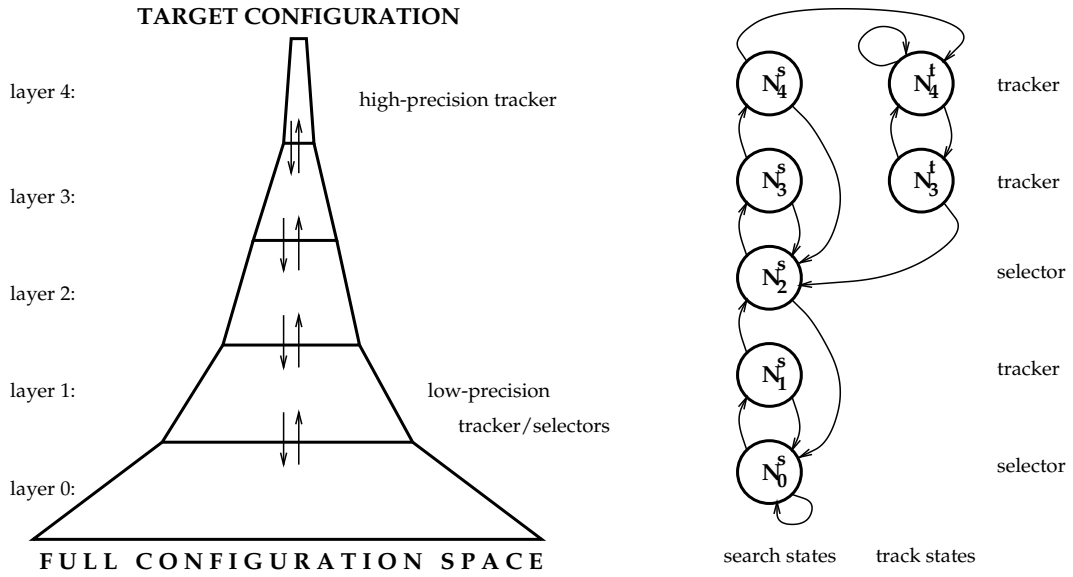


Figure 2. The correspondence between layers and states. On the left, a five-layer IFA system. On the right, the corresponding finite state transition graph for an example with three trackers and two selectors. Subscripts indicate corresponding layer number; superscripts indicate whether the state is in `search` or `track` mode. Outbound edges at the top left of a state are taken when a layer is successful; outbound edges at the bottom right are taken when a layer fails.

Since each state represents a unique combination of tracking layer and operating mode (i.e., `search` or `track`), the internal knowledge of the system in terms of tracking mode and precision is represented in its entirety by the current state of the automaton.

4.4 Algorithm

In describing the IFA algorithm (Figure 3), we use the following variables: N represents the current node in the state transition graph, $i = L(N)$ represents the layer associated with N, l_i represents the algorithm at layer i, and $\mathbf{X}_i^{\text{in}}$ and $\mathbf{X}_i^{\text{out}}$ denote the input and output sets of layer i. The functions $S(N)$ and $F(N)$ return the destination nodes of transition edges out of node N for success and failure, respectively.

Once initialized, the algorithm spends its time in the main loop, repeatedly executing the currently active layer and evaluating the results. Layers cause a straightforward transition based on their success, moving control up a layer when successful and down to the next tracker or selector, if unsuccessful.

Some additional bookkeeping happens for these transitions, both to keep iteration indices updated and to send the proper state sets to the next executing layer. A selector's iteration index is incremented each time its corresponding node is visited and reset to 0 either when the selector fails or when top layer tracking is achieved. Output state sets at a particular layer are only used as input sets of the next processing layer when the layer in question is successful. If a layer is unsuccessful, then the selector to which control falls uses the same input set as it did the previous time it was called. In Figure 3, Steps 1 and 2 are concerned with tracker and selector failure, respectively, and

Initialization: Set all iteration indices, σ_i, to 0, set N to N_0^{s} ($i = 0$), and set $\mathbf{X}_0^{\mathrm{in}} \leftarrow \mathcal{X}$, the entire state space.

Loop: Compute $\mathbf{X}_i^{\mathrm{out}} = l_i(\mathbf{X}_i^{\mathrm{in}}, I(t))$, perform set dilation (resulting in $\mathbf{X}_i^{\mathrm{out}'}$), and evaluate the following conditional:

1. If layer i is a tracker and $\mathbf{X}_i^{\mathrm{out}} = \emptyset$, update $N \leftarrow F(N)$, update $i \leftarrow L(N)$, and increment σ_i if Layer i is a selector. Go to Loop.
2. If layer i is a selector and $\sigma_i > \sigma_i^{\mathrm{MAX}}$, then reset σ_i to 0, reset the selector, set $N \leftarrow F(N)$, and set $i \leftarrow L(N)$. Go to Loop.
3. Otherwise, set $N \leftarrow S(N)$, $i \leftarrow L(N)$, and set $\mathbf{X}_i^{\mathrm{in}} \leftarrow \mathbf{X}_{i-1}^{\mathrm{out}'}$. If $N = N_{n-1}^{\mathrm{t}}$, reset all selectors, set all iteration indices to 0, and set $\mathbf{X}_i^{\mathrm{in}} \leftarrow \mathbf{X}_i^{\mathrm{out}'}$. Go to Loop.

Figure 3. The IFA algorithm.

Step 3 handles success for both trackers and selectors.

5 Properties of IFA Systems

Under some idealized assumptions about both algorithms and environment, IFA is easily seen to be correct and robust. In this section, we first make these idealized assumptions explicit and then discuss how the system is expected to perform under weakened assumptions.

5.1 Idealized IFA

If we make the single assumption that the top layer does not admit false positives, then the correctness of the algorithm follows immediately:

Property 1. *(correctness) The return of* $\mathbf{X}^{\mathrm{out}} \neq \emptyset$ *by Node* N_{n-1}^{t} *implies that* $\mathbf{x}^* \in \mathbf{X}^{\mathrm{out}}$.

The robustness properties of IFA also follow immediately when layers are composed of ideal trackers and selectors. The following definition will be useful:

Definition 3. *A state-image pair* $(\mathbf{x}, I)$ *is said to be in the* attractive region, $\mathcal{A}(f) \subseteq \mathcal{X} \times \mathcal{I}$, *of a tracker,* f, *if and only if for any* $\mathbf{X} \in \mathrm{Dom}(f)$, *if* $\mathbf{x} \in \mathbf{X}$ *then* $f(\mathbf{X}, I) \neq \emptyset$.

In short, the attractive region of a tracker is the set of inputs on which the tracker operates perfectly, without false negatives.

Now, consider an arbitrary IFA system, $\mathcal{S}$ that is composed of selectors and trackers. Define *ideal visual conditions*, $\mathcal{A}^*(\mathcal{S})$, to be the set of state-image pairs which fall in the attractive region of all trackers in the system. Let us assume the following:

Assumptions:

1. $\mathcal{A}^*(\mathcal{S})$ is non-empty, and only state-image pairs, $(\mathbf{X}, I)$, in $\mathcal{A}^*(\mathcal{S})$, occur.
2. Set dilation preserves containment of the target. That is, if $\mathbf{X}^{\text{in}}$ for a layer contains $\mathbf{x}^*$, then the union of all potential $\mathbf{X}^{\text{out}'}$ after set dilation contains $\mathbf{x}^*$.
3. For all selectors other than the bottom one, the iteration index associated with the selector is finite.

Under these conditions, the following property is straightforward to show:

Property 2. *(robustness) Assume that a system is begun at some node, N_k^{s} ($0 \leq k < n-1$, where n is the number of layers), and that $\mathbf{x}^* \in \mathbf{X}^{\text{in}}$. Then, under ideal visual conditions, $P(N = N_{n-1}^{\text{t}}) \to 1$ as $t \to \infty$.*

A special case of Property 2 affirms that when the system starts at the bottom layer, high precision tracking will eventually take place.

5.2 IFA in Practice

In reality, the assumptions made in the previous section do not hold perfectly. Since we are making claims about the robustness of systems, it is vital to our analysis to examine the effects of violated assumptions. We show that even though system performance deteriorates with weakened assumptions, the system as a whole nevertheless retains robustness and partial correctness.

5.2.1 Non-ideal Visual Conditions

Assumption 1 ("ideal visual conditions") is required for recovery after tracking is lost. The systematic violation of this assumption suggests that some permanent visual perturbation has taken place. Any visual situation that does not fall in $\mathcal{A}^*(\mathcal{S})$ for a particular tracker will prevent recovery, so a permanent perturbation will prevent recovery forever. On the other hand, the intent is to recover quickly, so ideal conditions should not need to remain for very long. In Section 5.3, we will consider the issue of expected recovery times.

5.2.2 Imperfect Trackers

Real trackers may return false positives. This can happen even with excellent visual criteria for judging whether an image contains the target object, because it is always possible that there are multiple objects visually identical to the target (henceforth, these objects will be dubbed *twins*).

Twins can exist for any single tracker. Thus, a tracker which uses only color to determine its target will see twins in every object with the same color as the target. Twins may violate Definition 1(Item 2) because the output set of a tracker may contain the state of a twin only, even if the input set contains the actual target. Twins can also violate Definition 1(Item 3) of trackers,

since they will cause a tracker to output a non-empty set, even when only a twin, and not the actual target state, is in their input state set. Therefore, the correctness and robustness properties in Section 5.1 depend on the fact that at the very least, the top layer tracker does not admit twins.

Twins can cause one other form of tracking error, which we call errors due to *sleight of hand*. Sleight of hand occurs when the system is in one of the non-top-layer track nodes (N_i^{t}, where $i \neq n-1$), and an object that is a twin for that layer occludes and then moves away from the target object. An example of this is when the system is tracking a target's partial state based solely on color, and another object of similar color passes in front of the target. It is possible in this instance that the system will begin tracking the second object instead of the target, without recognizing its error. This situation can be completely avoided by excluding all but the top layer from `track` mode. The cost of choosing this conservative option, however, is to lose the possibility of tracking at lower precisions under perturbed conditions, *i.e.*, to lose the quality of graceful degradation.

5.3 Recovery Times

Since the correctness and robustness of IFA algorithms in practice does not deviate significantly from that of the ideal model, the key performance issue is a quantitative one – the question of recovery time after a failure event.

It will be useful to consider a compact representation of an IFA system, in which selectors are always followed by trackers and trackers (except the top layer tracker) are followed by selectors. Formally this is not a problem since the composition of any number of trackers is a tracker and the composition of any number of selectors is a selector. Label the $2m$ new layers g_0, f_0, g_1, f_1, ..., g_{m-1}, f_{m-1}, where g_0 represents the bottommost selector and f_{m-1} is the topmost tracker.

In the following sections, we will let $\mathbf{E}[\kappa_i]$ represent the expected number of times that selector g_i will be called before its output set contains the target. Let T_i be the total time to execute both selector g_i and tracker f_i. We will additionally make some independence assumptions which will make the analyses tractable:

1. State-image pairs are static between visits to the bottom layer.
2. Each time the bottom-layer is visited, state-image pairs are generated independently from a uniform probability density over the attractive region.
3. The operation and associated probabilities of each layer is independent of the operation of layers preceding it.

These assumptions are intended to be consistent with intuitive notions of object motion, particularly for those targets which require post-failure robustness: While most objects exhibit apparent continuity over small time scales, they are more likely to appear to move "randomly" when sampled over larger time scales.

5.3.1 Ideal IFA

Under ideal conditions, the filter criterion for tracker f_i ensures that g_{i+1}'s input set always includes the target state. Thus, we can set $\sigma_{g_{i+1}}^{\mathrm{MAX}}$ to ∞ and still expect recovery.

The expected value of the sum of random variables is simply the sum of their expected values, so the total expected time to complete recovery from selector layer i, is given by

$$\mathbf{E}[T_i^{m-1}] = \sum_{j=1}^{m-1} \mathbf{E}[\kappa_j] T_j. \tag{1}$$

In this case, better selector heuristics at any Layer i will decrease $\mathbf{E}[\kappa_i]$ and contribute directly to shorter expected recovery times. We note that the impact of the improvement at any layer is roughly equal to improvement of any other layer, because most branches in the implicit search tree are immediate dead ends (see Figure 4(a)).

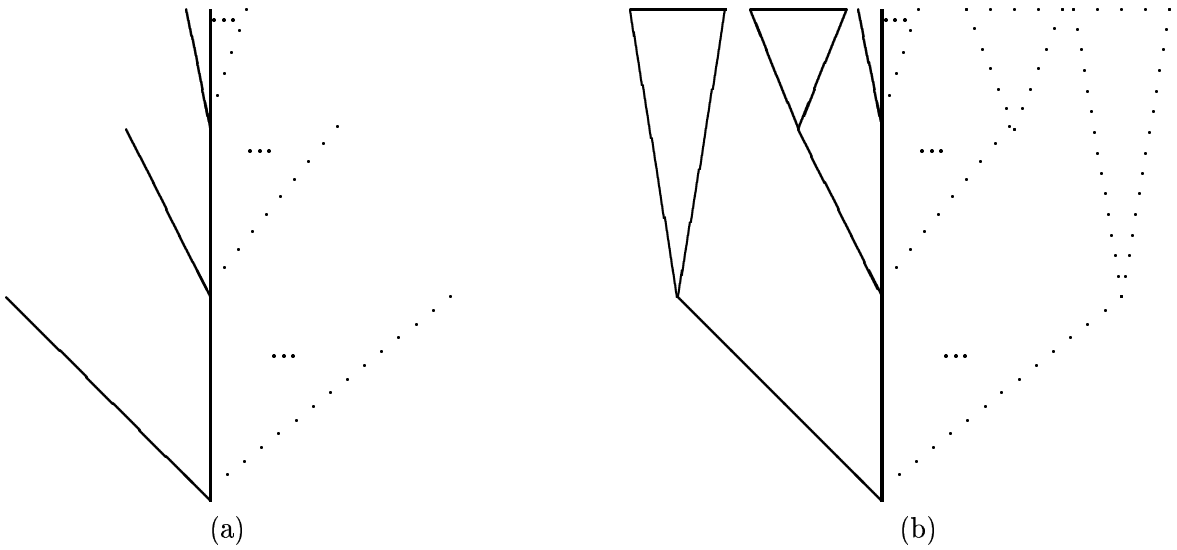


Figure 4. Schematics for the IFA search process. Solid lines represent branches or subtrees which actually were explored until target state localization. Dotted lines represent branches or subtrees which were not explored before state determination. Triangles represent entire subtrees. (a) IFA systems with trackers which fulfill the filter criterion prune all but one subtree at each depth. (b) Systems whose trackers do not fulfill the filter criterion may have to search every subtree to maximum depth.

5.3.2 IFA in Practice

The existence of false positives for trackers complicates the recovery time analysis. With false positives, iteration thresholds must be finite (except at the bottom layer), in order to ensure exit from upper layers that are given input sets not containing the target.

To make the analysis concrete, we make the following modeling decisions: (1) there is a fixed probability, p_{f_i}, of a false positive at Layer f_i for each potential branch of the search at Layer g_i; (2) a false positive at one layer does not affect the probability of false positives at higher layers; (3) there are fixed probabilities, $\rho_{ij} < (1 - p_{f_i})$, which represent the probability that a given selector, g_i, receives the target state in its input state, it will return an output set containing the target on the jth iteration.

We now consider the time required to fall out of a layer, should that layer be called with an input set not containing the target. We merely add up the number of visits to each layer (multiplied

by the respective times it takes to do so), given that all iteration thresholds must be reached before the algorithm reverts to layer $i-1$ below. Below, M_i represents the time it takes to revert to Layer $i-1$, starting at Layer i, when Layer i receives a set without the target:

$$\mathbf{E}[M_i] = \sigma_i^{\mathrm{MAX}} T_i + p_{f_i} \sigma_i^{\mathrm{MAX}} \mathbf{E}[M_{i+1}] \tag{2}$$

Note that only false positives generated at higher layers cause additional search deeper in the subtree. Letting $\mathbf{E}[M_{m-1}] = T_{m-1}$, this recurrence expands to

$$\mathbf{E}[M_i] = \sum_{j=i}^{m-1} \Big(\prod_{k=i}^{j-1} p_{f_k} \sigma_k^{\mathrm{MAX}} \Big) \sigma_j^{\mathrm{MAX}} T_j. \tag{3}$$

Next, we compute H_i^{m-1} the time required to find a target from Layer i, if the system does in fact find the target. First, we compute $\mathbf{E}[\kappa_i']$, the expected number of times Layer i itself is called, by normalizing the probabilities of κ_i for values less than or equal to σ_i^{MAX} and then computing their expected values. We then sum the expected time it takes to execute Layer i, the expected time spent exploring false positives, and the expected time spent ascending the remaining layers:

$$\mathbf{E}[H_i^{m-1}] = T_i \mathbf{E}[\kappa_i'] + p_{f_i} (\mathbf{E}[\kappa_i'] - 1) \mathbf{E}[M_{i+1}] + \mathbf{E}[H_{i+1}^{m-1}] \tag{4}$$

$$= \sum_{j=i}^{m-1} T_j \mathbf{E}[\kappa_j'] + p_{f_j} (\mathbf{E}[\kappa_j'] - 1) \mathbf{E}[M_{j+1}], \tag{5}$$

where M_{j+1} is as in Equation (3) and $\mathbf{E}[H_{m-1}^{m-1}] = T_{m-1}$. For purposes of recovery after failure, we are interested primarily in the case when $i = 0$.

Letting $\mathbf{E}[\kappa_{\mathrm{all}}]$ be the expected number of times the bottom layer will be called before the algorithm converges to the top layer, we can expect the total recovery time to be

$$\mathbf{E}[T_0^{m-1}] = \mathbf{E}[\kappa_{\mathrm{all}}] \mathbf{E}[M_0] + \mathbf{E}[H_0^{m-1}]. \tag{6}$$

The value of $\mathbf{E}[\kappa_{\mathrm{all}}]$ can be estimated as follows. The probability that selector-tracker pair i will succeed, given it receives the target in its input set is given by the sum of a geometric series:

$$P_i^{\mathrm{hit}} = \sum_{j=0}^{\sigma_i^{\mathrm{MAX}}} \rho_{ij} \prod_{k=0}^{j-1} (1 - \rho_{ik}). \tag{7}$$

Taking advantage of our independence assumptions, the probability of reaching the top layer from the bottom layer is always

$$P_{0,m-1}^{\mathrm{hit}} = \prod_{i=0}^{m-1} P_i^{\mathrm{hit}}. \tag{8}$$

The number of times the bottom layer must be visited to reach the top layer is geometrically distributed, with initial probability $P_{0,m-1}^{\mathrm{hit}}$. The expected value of this distribution is simply $1/P_{0,m-1}^{\mathrm{hit}}$, so $\mathbf{E}[\kappa_{\mathrm{all}}]$ is $1/P_{0,m-1}^{\mathrm{hit}} - 1$.

The complex interaction of several parameters (selector heuristic effectiveness, number of distractors, layer execution times, and iteration indices) makes these expressions difficult to grasp, and in any case, most of the probabilities are unknown *a priori*. In the next section, we perform Monte Carlo simulations of an IFA system to get a better sense for how design choices affect time to recovery.

5.3.3 Simulation

Using the assumptions made in Section 5.3.2 and approximate probability models based on our face tracking implementation (presented in Section 6), we performed a series of Monte Carlo simulations to predict recovery times under various situations.

The face tracker can be compactly represented as four alternating selector and tracker layers. Empirical data show that the bottom two layers take approximately 0.66 seconds to find and determine whether a blob of color is of sufficient size to be a face; the top two layers take mean time 0.33 seconds to confirm or reject a candidate state set.

Under good visual conditions, the hit rate and the rate of false positives at the bottom layers are determined entirely by the number of distractors present in the image. Letting d be the number of flesh-colored distractors in the image, we approximate p_{f_0}, the rate of false positives, with $\frac{d}{d+1}$. So, for example, with 9 distractors, $p_{f_0} = 0.9$. We eliminate the possibility of false positives at the top layer and set $p_{f_1} = 0.0$. The top layer iteration threshold is set to 1 (for reasons to follow), so that fixes $\mathbf{E}[\kappa_1']$ for the top selector to 1, as well.

This particular set of values induces the following terms for Equation (6):

$$\begin{aligned} \mathbf{E}[M_0] &= 0.66 + \frac{d}{d+1}0.33 \\ \mathbf{E}[H_0^1] &= 0.99 \\ \mathbf{E}[\kappa_{\text{all}}] &= \frac{1}{1/(d+1)} - 1 = d. \end{aligned}$$

Thus,

$$\mathbf{E}[T_{\text{face}}] = 0.66d + 0.33\frac{d^2}{d+1} + 0.99.$$

$\mathbf{E}[T_{\text{face}}]$ versus d becomes asymptotically linear as d grows larger, as seen in Figure 5(the dotted line). The solid line shows simulated results of how expected recovery times change with number of distractors (mean over 1000 trials). The simulated results are almost exactly as expected by analysis, with more distractors leading linearly to greater recovery times.

Figure 6 shows the effect of the top layer's iteration threshold on expected recovery time. The graph shows two different cases. The solid line indicates the case when there is one distractor and the dotted line indicates the case for two distractors. For these particular cases, clearly a smaller iteration threshold is better (and thus, for all of the experiments on real data, we used frustration indices of 1, except for the bottom layer).

We caution that other simulations with different parameters show that the trends noted here do not necessarily hold in general. Some simulations with 3 layers, for example, display best performance at iteration thresholds of 2 or 3. We have found, however, that except in the case of extreme cases where low layers take significantly longer to execute than higher layers, small iteration thresholds appear optimal.

6 Implementation and Results

In this section, we demonstrate several implemented IFA systems which robustly track various objects. These systems were implemented both on a Sun Sparc 20 equipped with a standard

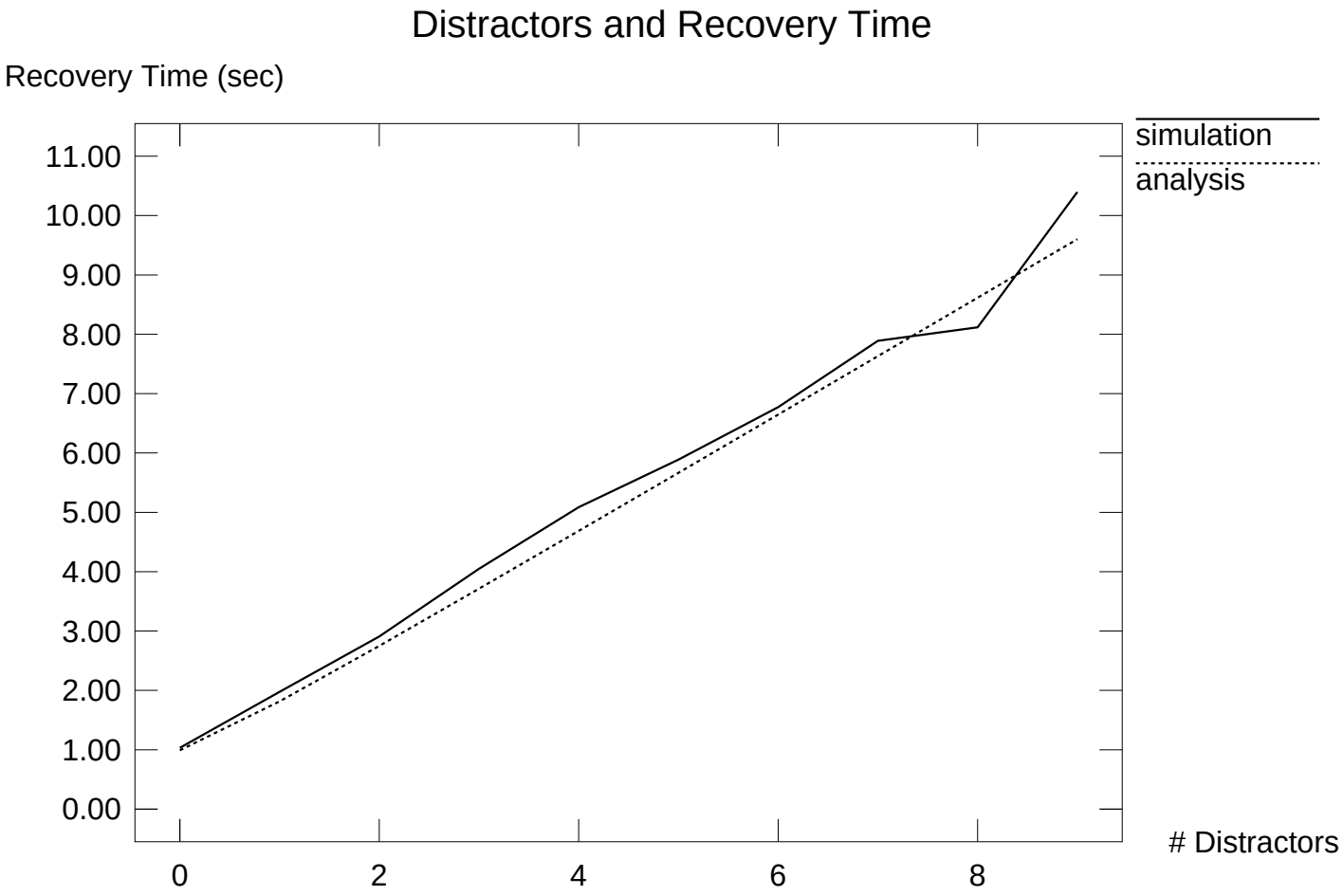


Figure 5. Recovery time as a function of the number of distractors.

framegrabber connected to a single-chip color CCD camera and on a Silicon Graphics Indigo with VINO color camera.

The appendix gives detailed descriptions of the trackers and selectors used by the systems.

6.1 Robust Face Tracking

The face tracking system presented here models faces as planar objects with a few special properties (it can, therefore, be easily modified to track other uniquely textured planar objects, *e.g.*, the envelope in Section 6.2). The state information recovered is the 2-dimensional position of the center of a pre-initialized face in image coordinates and the angular orientation of the face about the camera's optical axis.

There are 7 IFA layers for face tracking. Three selectors form the base of the system. Layer 0 is random selector. Layer 1 is biased towards regions of the state marked by flesh color in the image. Layer 2 is a position- and motion-based selector that tends towards regions that are closest to the last known position of the object. Layer 3 uses a homogeneous region tracker to track the

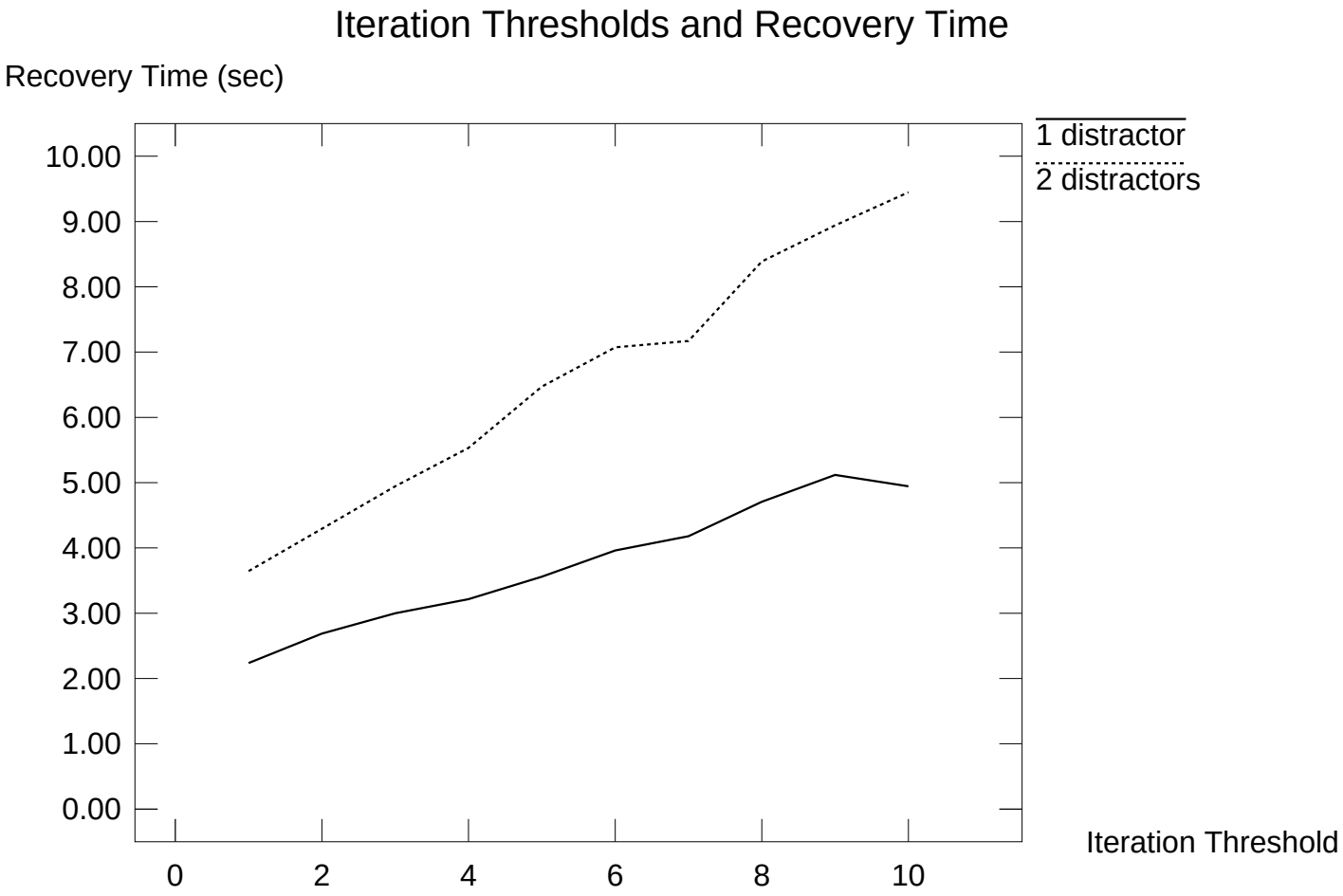


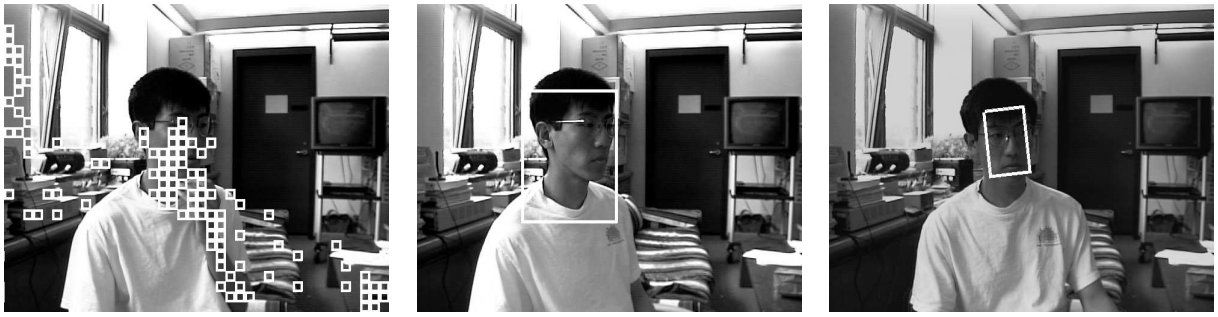
Figure 6. Recovery time as a function of the bottom-layer iteration threshold: with one distractor (solid), and two distractors (dotted).

position and size of "blobs" of color. Layer 4 is a selector which uses the principal axes of the rough ellipsoid formed by the face, and selects likely orientations of the face based on the axes. Layer 5 tracks state more precisely using an SSD-based pattern tracker, and Layer 6 uses the same SSD algorithm at high resolution to track with sub-pixel accuracy. All trackers determine failure in a unique way. Layer 3, color blob tracking, fails when too few of its pixels fall within the target color range, or when the blob of color is too small; Layer 7, SSD tracking, fails when the SSD residual value rises above an empirically set threshold. In addition, all trackers signal failure when their estimate of the speed of the target approaches their maximum tracking speeds.

Two sets of data about the specific face to be tracked must be collected before the system is operational. First, the range of colors of the face must be initialized. This is currently performed by a manual initialization in which various parts of the face are sampled and the RGB values are stored in memory. All color-based algorithms simply compare input pixels to stored RGB values. (More sophisticated color models could be implemented, but our simple technique offers the advantage of speed during execution, and we have found that it performs adequately under common lighting

conditions.) The second set of data sampled is an actual intensity-based image of the face. This image, and its spatial derivatives, are used as templates for the SSD trackers at the top two layers. For a more detailed account of how SSD tracking operates, see [20].

The system operates exactly as expected – it resists failure for minor disturbances and recovers when mistracking occurs. Figure 7 shows face tracking at different layers.



(Layer 1) (Layer 3) (Layer 6)

Figure 7. Face tracking at different layers. On the left, tracking has temporarily reverted to Layer 1, a color selector. Small squares mark areas of the image in which flesh-color is detected. In the middle, tracking proceeds at Layer 3, color blob tracking. On the right, full state with position and orientation is computed by Layer 6, precise SSD tracking.

6.1.1 Experiments

We have tested the system extensively under various types of visual perturbations. Four sets of experiments were performed for each type of perturbation event: (TT) a temporary event during `track` mode, (TS) a temporary event during `search` mode, (PT) a permanent event in `track` mode, and (PS) a permanent event in `search` mode. All experiments for `track` mode were performed with the tracking system initially at the top layer. All experiments for `search` mode were begun at the bottommost layer. Table 8 shows the results of the experiments.

Briefly, the results can be summarized as follows: With a temporary perturbation, the system, if it is affected at all, recovers to full-precision tracking of the target no matter what type of disturbance occurs and regardless of the layer then executing. Recovery never occurs for permanent perturbations where the disturbance prevents target confirmation in any tracking layer. Overall, the system is highly robust to any type of temporary disturbance. The same conclusions were informally drawn from repeated demonstrations to skeptical, well-informed audiences [16].

6.1.2 False Positives

The worst error a tracking system can make is to track the wrong object, believing it to be the correct target. As mentioned in Section 5.2.2 such false positives should only occur in an IFA framework when there are twins or when sleight of hand occurs.

The case of twins is all but ruled out in face tracking by the exacting nature of the SSD tracking algorithm which expects a strict match with the stored template. Over the course of hundreds of experiments, the face tracking system has never locked on to a face other than the one on which it was initialized.

Mode:	TT	TS	PT	PS
Change in hue	none	slows recovery	none	prevents recov
Change in intensity	none	none	none	none
Lights out	loss & recovery	slows recovery	loss	prevents recov
Medium speed	drop(1) & recov	slows recovery	drop(1)	prevents recov
Fast speed	drop(2) & recov	slows recovery	drop(2)	prevents recov
Very fast speed	loss & recovery	slows recovery	loss	prevents recov
Flesh-toned distract	none	slows recovery	none	slows recovery
Other faces distract	none	slows recovery	none	slows recovery
Twin distract	none	distracts & recov	none	confuses 50%
Partial Occlusion	varies	none	varies	none
Full Occlusion	loss & recovery	slows recovery	loss	prevents recov
Target Distortion	varies	slows recovery	varies	prevents recov
Noise	varies	slows recovery	varies	prevents recov
Scale Changes	drop(3) & recov	slows recovery	drop(3)	prevents recov
Out-of-plane Rotation	drop(3) & recov	slows recovery	drop(3)	prevents recov
Disappearing Target	loss & recovery	slows recovery	loss	prevents recov

Figure 8. The effect of various visual disturbances on face tracking performance. Numbers after "drop" entries indicate the number of layers descended. Some disturbances affect the system differently based on the extent of the disturbance. In these cases, the effect varies from no effect at all to mistraking.

Sleight of hand is a rare, but real problem. An example of this is when the system is tracking the face based solely on color (Layer 3, `track` mode), and another flesh-toned object passes in front of the face. It is possible in this instance that the system will begin tracking the second object instead of the face, without recognizing its error. This situation can be contrived, and the system does in fact mistrack, recovering only if the second object itself undergoes a significant visual disturbance.

In practice, this scenario has never occurred even during hour-long demonstrations, primarily because it is rare for anything flesh-toned and larger than the target face to occlude the target completely, and then remain in the image. Occlusions with smaller flesh-toned objects, such as the subject's hands, are not distracting enough to the color tracking algorithm to pull tracking away, and other occlusions simply cause complete mistracking (during which time the system acknowledges its temporary inability to track).

6.1.3 Actual Recovery Times

Figure 9 shows where the tracking systems spends it time during some example runs. The horizontal axis represents time and the vertical axis represents the IFA layer executed by the system. The three separate graphs represent three different background conditions which will be explained below. All trials began with a fully initialized system at Layer 7.

Time instants marked by an '**a**' indicate temporary partial occlusions (scratching the chin). The head turned approximately 10 degrees to the left at points marked by a '**b**'. In both instances, top-layer tracking was disturbed, causing the system to fall to a coarse SSD tracker (Layer 5) momentarily, but as the occlusion ceased or as the head turned to face forward, the system quickly

climbed back up to the top-layer tracker. At times marked by '**c**,' a more significant occlusion (scratching the nose) causes tracking to descend to color blob tracking at Layer 3. At instants marked '**d**,' the target face moved quickly and erratically for approximately 2 seconds. During this interval, tracking alternates between layers 3 and 5, where tracking takes place with varying degrees of precision. Tracking resumes as soon as the face settles to a slow speed. At '**e**,' a quick, complete occlusion of the face occurs. The system descends all the way to Layer 2, where the algorithm searches for a moving object near the last known location of the face. Since the occlusion ends quickly, the first candidate output set of Layer 2 (a selector) contains the target and tracking recovers within 2 seconds in all cases. Finally, at times marked '**f**,' the camera was jerked away so that tracking was completely lost as the face disappeared from view. While the camera was readjusted with the target object in a different location in the field of view, the system oscillated back and forth among the low layers as it tried to find the target object among various candidates. Eventually, the color selector chose an output set containing the target object and tracking returned to the top layer.

Because our top level selectors use heuristics based on color and motion, the more cluttered the background is with respect to these attributes, the greater the time required for tracking to recover. Thus, in Figure 9(top), where there was only one other flesh-colored object in the background, experiment **f** exhibits a very quick recovery from lost tracking, even after the face temporarily disappears completely from view. With a few more flesh-toned objects in the background (Figure 9(middle)), recovery takes somewhat longer; the bottom graph shows trials with even more potential distractors. Figure 10 shows images containing different levels of clutter. As more clutter is added to the background, the tracking system must examine more disjoint subsets of the state space before finding a subset that contains the actual target.

The data correspond closely with the analysis of expected recovery times from Section 5.3.2 and the simulations from Section 5.3.3. The expected recovery time should increase linearly with the number of flesh-toned objects in the scene. Repeated experiments (40 trials) confirm these figures (see Figure 11). This trend suggests analogies with pop-out and camouflage effects in biological vision systems.

6.1.4 Tradeoff Management

Part of the robustness of an IFA system arises from its ability to dynamically manage tradeoffs between different tracking needs. For example, consider the tradeoff between allowable tracking speed and precision. IFA systems are constructed so that more precise algorithms occupy higher layers. Since no one would choose to use a slow, imprecise algorithm when a faster, more precise one is available, a consequence of this organization is that layers tend to be sorted in order of decreasing speed. In particular, those layers which have corresponding `track` nodes in the state transition graph show an increase in precision together with a decrease in speed.

In Figure 12, we plot accuracy (the negative log of translational error $\times$ orientational error) versus target translational speed in pixels. The four graph segments represent the top four layers in the face tracker, which have track nodes. The graph is continuous within each layer but disjoint at the boundaries because of the qualitative difference in the tracking algorithms across layers. Each segment represents the inherent precision-speed tradeoff within a single layer. Layer 6 shows the greatest accuracy but can only track up to a certain target speed. Layer 3 shows the opposite behavior. Together, the segments show IFA's ability to integrate several different tracking

algorithms and manage the tradeoff between precision and maximum allowable target speed, given limited computational resources.

Similar tradeoffs between the extent of other visual perturbations (extent of occlusion, deviation from object description, etc.) and precision are also handled by IFA in the same manner.

6.2 Other IFA Systems

In the past, we have often used 3.5 inch diskettes as a convenient target for various experiments with visual servoing [18, 40]. These experiments, while successful in demonstrating hand-eye coordination, were nevertheless extremely sensitive to disturbances. To make this system more robust to visual disturbances, we developed a robust disk tracker which consists of the following components (in order of increasing output accuracy): a color-, motion-, intensity-, and position-based combination selector (layer 0), a homogeneous region tracker (layer 1), an edge-of-polygon tracker (layer 2), and a feature-based rectangle tracker (layer 3).

The results for disk tracking are qualitatively similar to the results for face tracking [39]. The major difference in performance was that more disturbances caused tracking to fail completely. The poorer performance can be explained by the lack of robustness in the high-layer tracking for rectangles. But, as with face tracking, the system was able to rapidly recover from all temporary perturbations.

In another illustration of IFA applied to robotics, we developed a robust doorknob tracking module for use with mobile robots [12]. This particular implementation used an *intermediate object* [48], where a search for the door takes place to facilitate finding the doorknob itself (characterized as an "edge-rich" object on doors approximately at waist height). Restriction of attention to doors helps both to speed up the search as well as to constrain the search in an environment full of edge-rich objects.

Finally, we constructed a two-sided object tracker which can be seen as a precursor to a robust 3-D object tracker. The two-sided object tracker is based on the idea of tracking an object by matching to multiple views.

The implementation is identical to that for face tracking, except that the SSD trackers are replaced by SSD-based multi-pattern trackers, and the color-based heuristic for the bottom layer selector seeks to find color matches based on the union of predominantly occurring colors in all stored figures.

In Figure 13, we see how two images were used to track an envelope. The front and back of the envelope were initialized as the images to track. In the left and right figures, tracking proceeded at the top layer, since one of the envelope's faces was completely visible. However, during the time the envelope was being flipped over, no match was made to either image; thus, tracking temporarily reverts to a lower layer where color-based selection takes place.

Performance of two-sided tracking was similar to that of face tracking, although recovery times were greater due to the need to compare the image with two stored templates.

7 Conclusion

The best argument for IFA systems is the subjective experience of watching them in action. Where previously, entering the camera field of view was taboo during experiments which use tracking, now, it is almost a pleasure to deliberately cause mistracking, just to watch the system recover its

target. Tracking applications can be left running even as people walk in front of the camera, lights are turned off, or target equipment is moved, because soon after the environment settles again, tracking will have resumed. In a word, tracking is tenacious.

One of the advantages of the IFA framework from a programmer's point of view is the relative ease with which existing trackers can be incorporated into it. The bottom layers with some variation suit almost all tracking applications. Then, given existing trackers, it is usually easy to decide when they mistrack based on geometric and visual constraints, especially because there is room to err on the side of being overly conservative. Putting them together is a matter of nesting tracking loops with entrance and exit conditions that depend on the layers. Placing the most precise tracker in the innermost loop fashions the top of the framework and added robustness is the result.

Further work with IFA proceeds along several avenues. One interesting problem is the automatic construction of IFA systems, given some *a priori* knowledge about the desired target. It should be possible to design and initialize an IFA system given statistical models for what faces look like, how often they appear in an image, and so on. Another direction for future work lies in creating dynamic IFA systems which swap layers in and out depending upon visual conditions. Finally, we are attempting to further formalize the notions of "robustness" and "graceful degradation" with the goal of providing a theoretical framework for evaluating the robustness of IFA and other real-time systems.

A Descriptions of Sample Layers

Layers are based on well-known tracking algorithms or focus of attention cues. Below are some that we incorporate into IFA systems.

- The **linear exhaustive selector** takes any set of configurations as its input and returns a set that is appropriate for the tracker above it. Repeated application of this selector on the same input set will return different output sets such that the union of output sets contains the input set within a finite number of applications.
- A **random exhaustive selector** takes any set of configurations as its input and randomly returns a set that is appropriate for the tracker above it. Repeated application of this selector returns different output sets such that in the limit of infinite applications, the entire input set is covered.

Either of the selectors above provide a "last resort" mechanism for any heuristic-based selector and are used as the basis for the selectors described below.

- **Position-based selectors** bias their output according to preset geometric constraints which may be relative to the camera coordinate system or absolute with respect to a physical coordinate system. If, for example, it is known *a priori* that there will be two objects of similar appearance and that the target is the one to the left, the selector begins by returning output configurations which are consistent with this information. Position-based selectors may also incorporate prediction models such that selection is biased towards the expected location of the object. As with any selector, multiple calls to this selector will eventually cover the entire input set.

- A **color-based selector** marks the regions in a image which contain certain range of hues. The selector will find the pixels in the image which are within the range of desired hues and will tend to return a set of possible configurations for its target which are consistent with those pixels. See Figure 10 for an example of an color-based selector choosing only parts of the image which coincide with the target object's hue.

- An **intensity-based selector** performs exactly like the color-based selector, except that its image processing occurs on the intensity values of pixels rather than on hue.

- Given any input set, a **motion-based selector** will tend to return output sets that include configurations whose camera projections intersect with regions of non-zero image differences between two recent time instances. Simply put, it is most likely to focus its attention on regions where motion is detected.

- A **combination selector** combines some of the search heuristics described above. For example, a color- and motion-based selector would most likely return output sets whose configurations correspond to projections which match a given hue and exhibit motion.

Many of the selectors above are appropriate for processing an entire image. We employ them on subsampled, low-resolution images which take only a fraction of the time it would take to acquire and process complete images. Because they produce relatively low-precision output, we have found that applying them to low-resolution images does not significantly affect their performance.

- If the target object has pixel values (hue or intensity) which are predominantly within a certain range, then a **homogeneous region tracker** (a "blob" tracker) will produce a set of configurations which correspond roughly to the center of the region. Tracking is performed by computing spatial moments of the pixels which exhibit the RGB values sought.

- Intuitively, an **edge-of-polygon tracker** tracks a single edge of a polygon. In terms of the tracking formalism, it expects a set of target configurations which would project the target object onto a small region of the image, but for which no orientation information is known. By tracking a single edge, the output configuration set includes only those configurations of the polygon which would be consistent with the object projecting the tracked edge onto the camera image.

- A **feature-based polygon tracker** tracks polygonal shapes as multiple corners, which are in turn tracked as two intersecting line segments (a rectangle tracker is described in [19]). The polygonal tracker will narrow a compact set of input configurations, into an smaller set of configurations which corresponds to the image projection of the polygon together with some allowance for noise. Mistracking is detected when the line segments of adjacent corners are out of alignment or if distinct corners merge together.

- A **feature-based rectangle tracker** is a specific instance of the polygon tracker, where four corners are used to track an approximately rectangular shape.

- The **SSD-based pattern tracker** matches a stored template to a region of the current image and computes a residue value. It returns an output set whose elements are configurations that correspond to affine transformations of the template which match with the current camera

image. It fails if the computed sum-of-squared-difference (SSD) residue is above a given threshold (see [17]). The residue of an SSD computation is the actual pixel-by-pixel sum of squared differences. Variations of this tracker may perform the same function with a subset of the affine transformations.

- An **SSD-based multi-pattern tracker** performs as above, but it tracks successfully if the current image contains a match to any of several stored images.
- A **3-D model tracker** (such as the ones described in [10, 13, 26]) tracks the 3-D pose of an object with respect to the camera. The input configuration set would be the approximate pose of the object with an error margin, and the output set could be a smaller set of poses. Mistracking is detected when the image does not provide enough correspondences with the model.

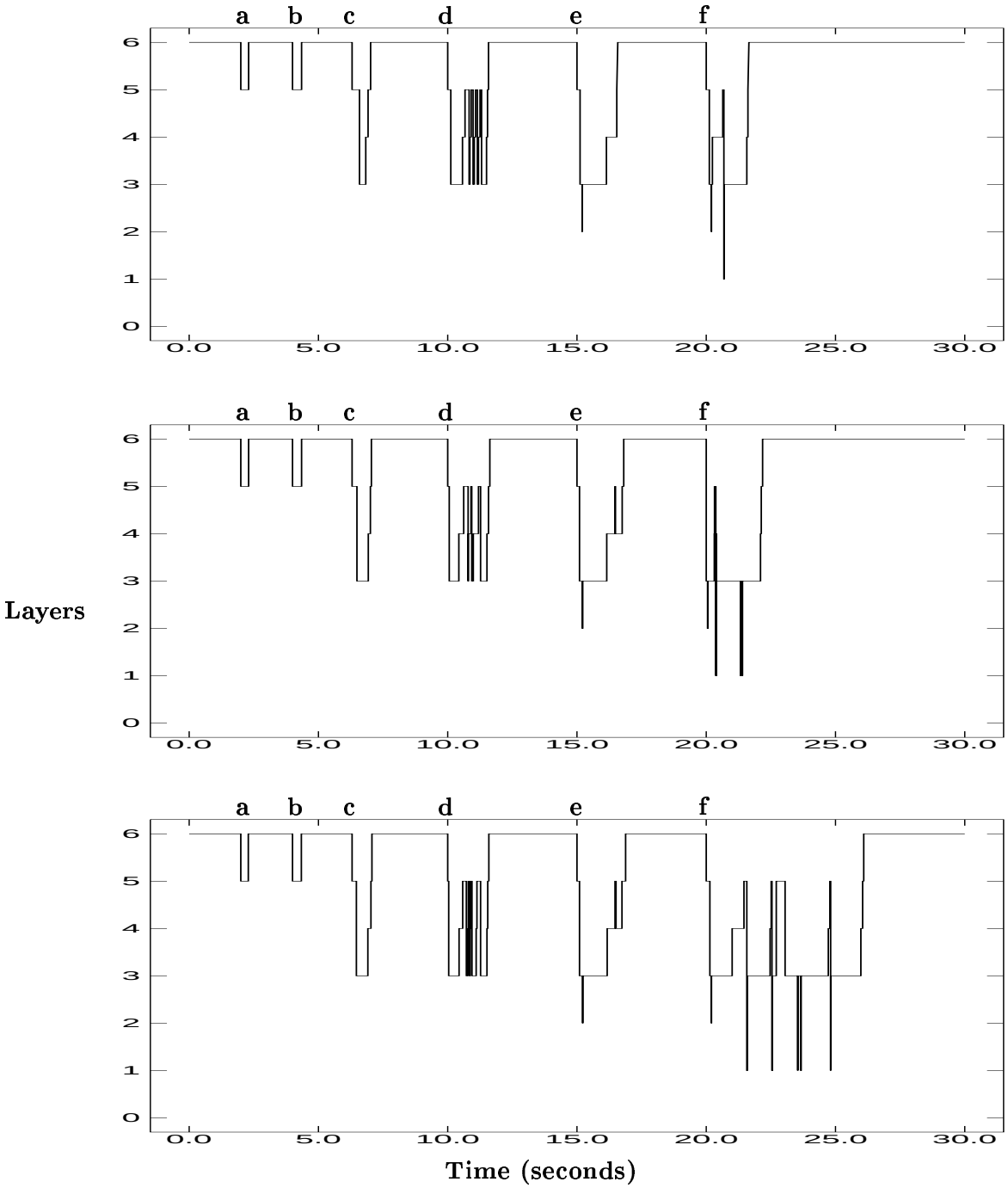
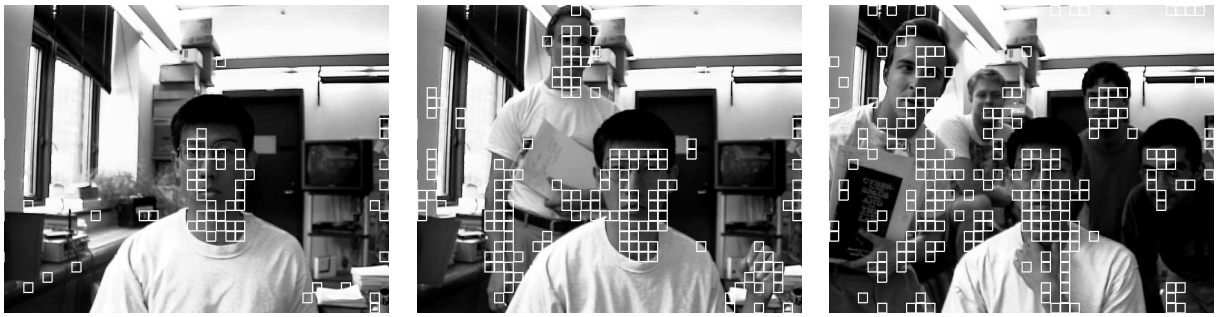


Figure 9. SSD tracking layers and recovery time. Low clutter (top), medium clutter (middle), and high clutter (bottom).



(low) (medium) (high)

Figure 10. Experimental setup for various levels of clutter. Regions considered flesh-toned are marked. Note that as clutter increases, more regions are marked, and consequently, the number of candidate states increases.

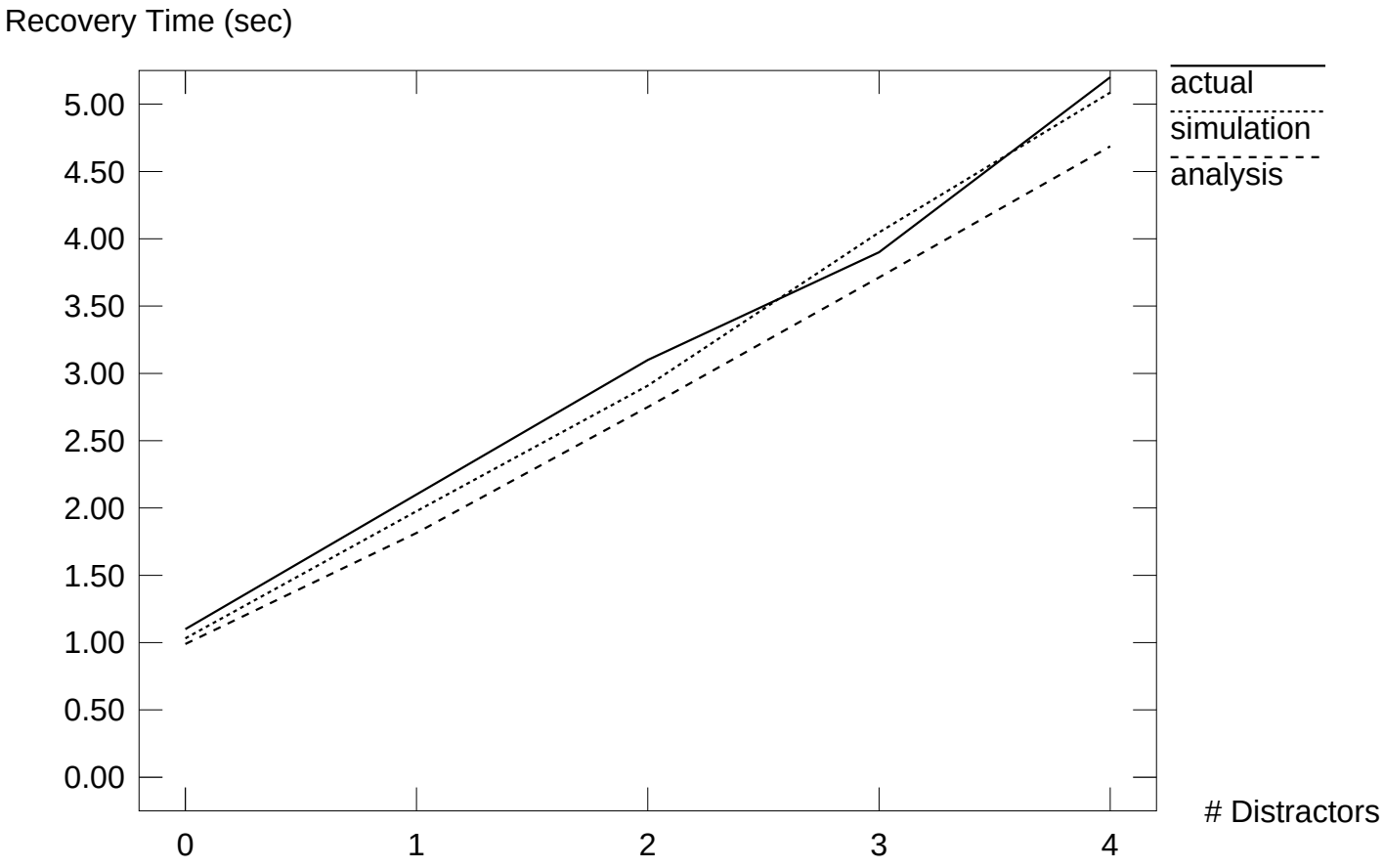


Figure 11. Actual (solid, 40 trials), simulation (dotted, 200 trials), and computed (dashed) mean recovery times for zero to four distractors.

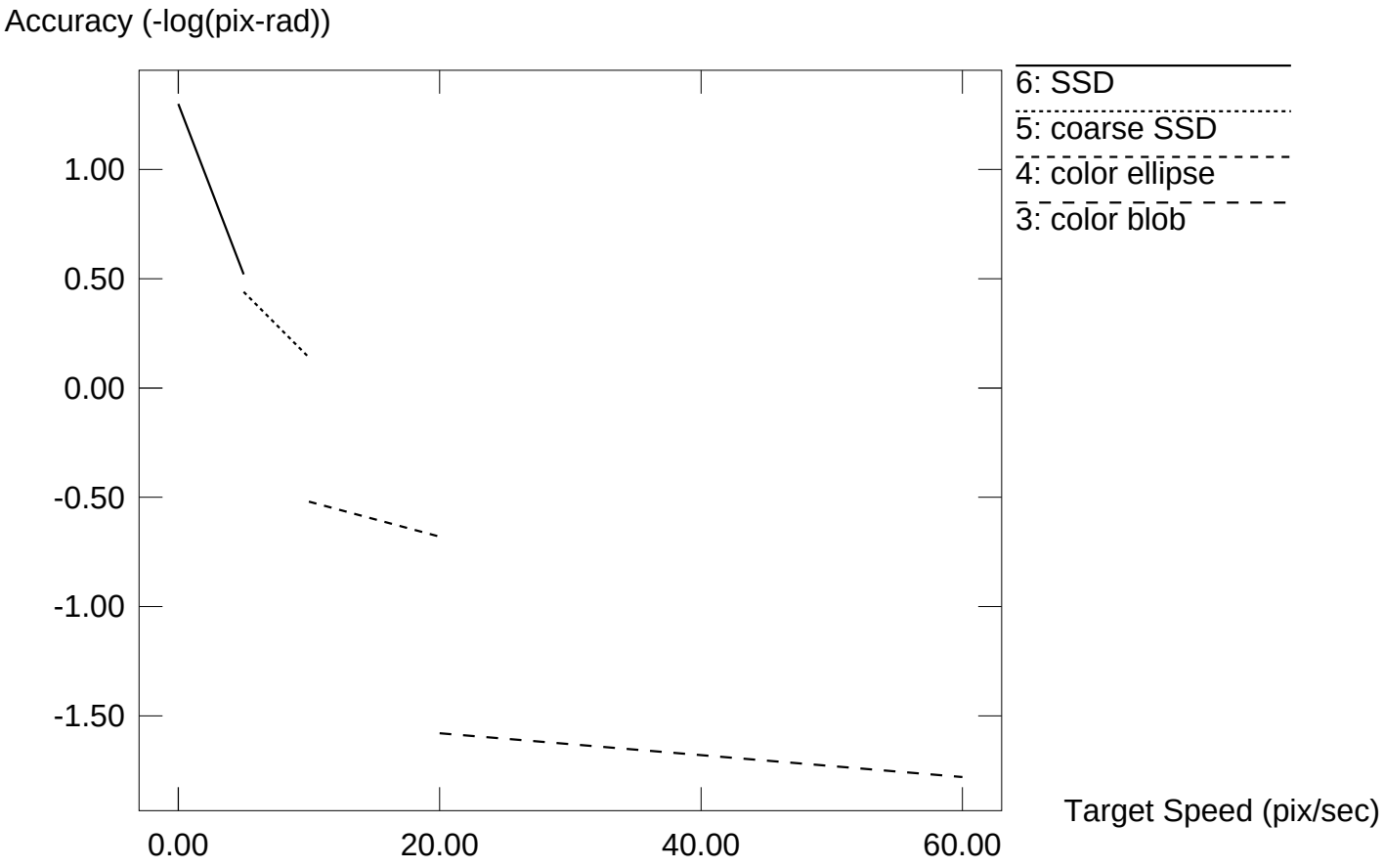
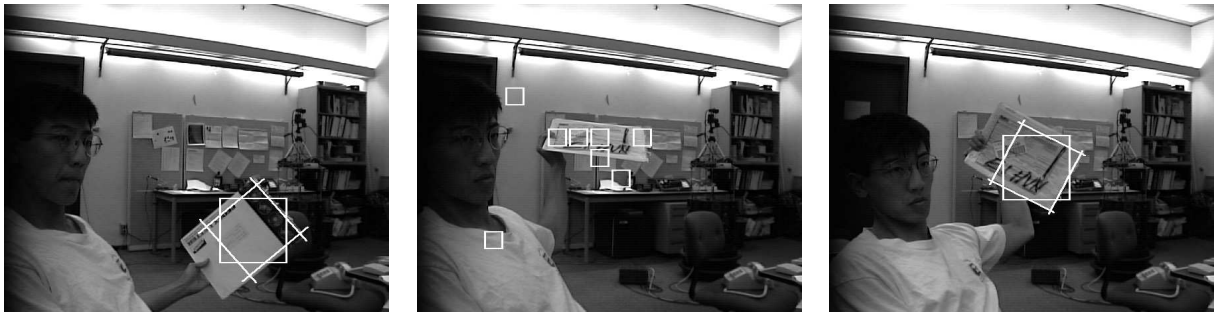


Figure 12. Layers exhibit a tradeoff between speed and precision.



(front) (no match) (back)

Figure 13. Tracking an envelope with two different sides. In the center image, the envelope is tilted too much to match with either stored image, so tracking has reverted to color tracking.

This research was supported by Siemens AG, ARPA grant N00014-93-1-1235, Army DURIP grant DAAH04-95-1-0058, by National Science Foundation grant IRI-9420982, and by funds provided by Yale University.

References

[1] A. Blake, R. Curwen, and A. Zisserman. Affine-invariant contour tracking with automatic control of spatiotemporal scale. In *Proc. Int'l Conference on Computer Vision*, pages 421–430, Berlin, Germany, May 1993.

[2] K.J. Bradshaw, P.F. McLauchlan, I.D. Reid, and D.W. Murray. Saccade and pursuit on an active head/eye platform. *Image and Vision Computing*, 12(3):155–163, April 1994.

[3] R. Burridge, A. Rizzi, and D. Koditschek. Towards a dynamical pick and place. In *Proc. Int'l Conf. Intel. Rob. and Sys.*, volume 2, pages 292–297, August 1995.

[4] R. Burridge, A. Rizzi, and D. Koditschek. Sequential composition of dynamically dexterous robot behaviors. *Int'l J. Robot. Res.*, 1997. To appear.

[5] P.J. Burt. Attention mechanisms for vision in a dynamic world. In *Int'l Conference on Patt. Recog.*, pages 977–987, 1988.

[6] P.J. Burt and G.S. van der Wal. An architecture for multiresolution, focal, image analysis. In *Int'l Conference on Patt. Recog.*, pages 305–311, 1990.

[7] V. Concepcion and H. Wechsler. Detection and localization of objects in time-varying imagery using attention, representation and memory pyramids. *Patt. Recog.*, 29(9):1543–1557, September 1996.

[8] D. Coombs and C.M. Brown. Real-time binocular smooth-pursuit. *Int'l Journal of Computer Vision*, 11(2):147–165, October 1993.

[9] J. Crowley and F. Berard. Multi-modal tracking of faces for video communication. In *Computer Vision and Patt. Recog.*, 1997.

[10] J. L. Crowley, P. Stelmaszyk, T. Skordas, and P. Puget. Measurement and integration of 3-D structures by tracking edge lines. *Int'l Journal of Computer Vision*, 8(1):29–52, 1992.

[11] S. M. Culhane and J. K. Tsotsos. An attentional prototype for early vision. In *Computer Vision – ECCV '92*, pages 551–560, Italy, May 1992.

[12] W. Feiten, G. D. Hager, J. Bauer, B. Magnussen, and K. Toyama. Modeling and control for mobile manipulation in everyday environments. In *Int'l Symp. on Robot. Res.*, 1997.

[13] D. B. Gennery. Visual tracking of known three-dimensional objects. *Int'l Journal of Computer Vision*, 7(3):243–270, 1992.

[14] G. Hager and P.N. Belhumeur. Occlusion insensitive tracking of image regions with changes in geometry and illumination. Technical Report DCS-TR-1122, Yale University, 1996.

[15] G. Hager and K. Toyama. XVision: A portable substrate for real-time vision applications. In *Proc. European Conf. on Computer Vision*, volume 2, pages 507–512, Cambridge, UK, 1996.

[16] G. Hager and K. Toyama. XVision: Interaction through real-time visual tracking. In *CVPR Demo Program*, San Francisco, CA, 1996.

[17] G. Hager and K. Toyama. XVision: A portable substrate for real-time vision applications. *Computer Vision and Image Understanding*, 1998.

[18] G. D. Hager. Calibration-free visual control using projective invariance. In *Proc. 5th Int. Conf. Comp. Vision*, pages 1009–1015, 1995.

[19] G. D. Hager. The "X-Vision" system: A general purpose substrate for real-time vision-based robotics. In *Proceedings of the Workshop on Vision for Robots*, pages 56–63, 1995. Also available as Yale CS-RR-1078.

[20] G. D. Hager and P.N. Belhumeur. Real-time tracking of image regions with changes in geometry and illumination. In *Computer Vision and Patt. Recog.*, pages 403–410. IEEE Computer Society Press, 1996.

[21] E. Huber and D. Kortenkamp. Using stereo vision to pursue moving agents with a mobile robot. In *Proc. Int'l Conf. on Robot. and Autom.*, pages 2340–2346, Nagoya, Japan, May 1995.

[22] M. Isard and A. Blake. Contour tracking by stochastic propagation of conditional density. In *Proc. European Conf. on Computer Vision*, pages I:343–356, 1996.

[23] R.E. Kahn, M.J. Swain, P.N. Prokopowicz, and R.J. Firby. Gesture recognition using Perseus architecture. In *Computer Vision and Patt. Recog.*, pages 734–741, 1996.

[24] H. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *Int'l Journal of Computer Vision*, 1:321–331, 1987.

[25] A. Kosaka and G. Nakazawa. Vision-based motion tracking of rigid objects using prediction of uncertainties. In *Proc. Int'l Conf. on Robot. and Autom.*, pages 2637–2644, Nagoya, Japan, May 1995.

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

[27] A. Maki, P. Nordlund, and J. Eklundh. A computational model of depth-based attention. In *Proc. Int'l Conf. on Patt. Recog.*, page D9E.1, 1996.

[28] D. Murray and A. Basu. Motion tracking with an active camera. *IEEE Trans. Patt. Anal. and Mach. Intel.*, 16(5):449–459, May 1994.

[29] U. Neisser. *Cognitive Psychology.* Appleton-Century-Crofts, New York, 1967.

[30] H. K. Nishihara. Teleos AVP. In *CVPR Demo Program*, 1996.

[31] P. Nordlund and T. Uhlin. Closing the loop: Detection and pursuit of a moving object by a moving observer. *Image and Vision Computing*, 14:265–275, May 1996.

[32] N. Oliver, A. Pentland, and F. Berard. LAFTER: Lips and face real time tracker. In *Proc. Computer Vision and Patt. Recog.*, 1997.

[33] K. Pahlavan and J.-O. Eklundh. A head-eye system – analysis and design. *CVGIP: Image Understanding*, 56:41–56, July 1992.

[34] P. Prokopowicz, M. Swain, and R. Kahn. Task and environment-sensitive tracking. Technical Report 94-05, University of Chicago, March 1994.

[35] C. Rasmussen, K. Toyama, and G. D. Hager. Tracking objects by color alone. Technical Report DCS-TR-1114, Yale University, June 1996.

[36] A.A. Rizzi and D. E. Koditschek. Further progress in robot juggling: The spatial two-juggle. In *Proc. Int'l Conf. on Robot. and Autom.*, pages 919–924, 1993.

[37] D. Terzopoulos and T. F. Rabie. Animat vision: Active vision in artificial animals. In *Int'l Conf. on Comp. Vision*, pages 801–808, June 1995.

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

[39] K. Toyama and G. Hager. Incremental focus of attention for robust visual tracking. In *Computer Vision and Patt. Recog.*, pages 189–195, 1996.

[40] K. Toyama, G. Hager, and J. Wang. SERVOMATIC: a modular system for robust positioning using stereo visual servoing. In *Proc. Int'l Conf. on Robot. and Autom.*, pages 2636–2643, 1996.

[41] K. Toyama and G. D. Hager. Tracker fusion for robustness in visual feature tracking. In *SPIE Int'l Sym. Intel. Sys. and Adv. Manufacturing*, volume 2589, Philadelphia, PA, October 1995.

[42] K. Toyama and G. D. Hager. If at first you don't succeed... In *Proc. Nat'l Conf. on AI*, Providence, RI, 1997.

[43] A. Treisman. Preattentive processing in vision. *CVGIP: Image Understanding*, 31:156–177, 1985.

[44] J. K. Tsotsos. An inhibitory beam for attentional selection. In *Spatial Vision for Humans and Robots*. Cambridge University Press, 1993.

[45] J. K. Tsotsos. Towards a computational model of visual attention. In T. Papathomas, C. Chubb, A. Gorea, and E. Kowler, editors, *Early Vision and Beyond*, pages 207–218. MIT Press, 1995.

[46] T. Uhlin, P. Nordlund, A. Maki, and J.-O. Eklundh. Towards an active visual observer. In *Proc. Int'l Conf. on Computer Vision*, pages 679–686, Cambridge, MA, June 1995.

[47] M. Vincze. Optimal window size for visual tracking. In *Int'l Conference on Patt. Recog.*, page A91.1, 1996.

[48] L.E. Wixson and D.H. Ballard. Using intermediate objects to improve the efficiency of visual-search. *Int'l J. of Computer Vision*, 12(2-3):209–230, April 1994.

[49] J. Wolfe. Guided search 2.0: A revised model of visual search. *Psychonomic Bulletin and Review*, 1(2):202–238, 1995.

[50] C.R. Wren, A. Azarbayejani, T. Darrell, and A. Pentland. Pfinder: Real-time tracking of the human body. In *Vismod*, 1995.