X Vision: A Portable Substrate for Real-Time Vision Applications

Gregory D. Hager and Kentaro Toyama
Department of Computer Science
Yale University, P.O. Box 208285
New Haven, CT, 06520

Phone: (203) 432-6432
Fax: (203) 432-0593
E-mail: hager@cs.yale.edu, toyama@cs.yale.edu

Abstract

In the past several years, the speed of standard processors has reached the point where interesting problems requiring visual tracking can be carried out on standard workstations. However, relatively little attention has been devoted to developing visual tracking technology in its own right.

In this article, we describe X Vision, a modular, portable framework for visual tracking. X Vision is designed to be a programming environment for real-time vision which provides high performance on standard workstations outfitted with a simple digitizer. X Vision consists of a small set of image-level tracking primitives, and a framework for combining tracking primitives to form complex tracking systems. Efficiency and robustness are achieved by propagating geometric and temporal constraints to the feature detection level, where image warping and specialized image processing are combined to perform feature detection quickly and robustly.

Over the past several years, we have used X Vision to construct several vision-based systems. We present some of these applications as an illustration of how useful, robust tracking systems can be constructed by simple combinations of a few basic primitives combined with the appropriate task-specific constraints.

To appear in Computer Vision and Image Understanding

1 Introduction

Real-time vision is an ideal source of feedback for systems that must interact dynamically with the world. Cameras are passive and unobtrusive, they have a wide field of view, and they provide a means for accurately measuring the geometric properties of physical objects. Potential applications for visual feedback range from traditional problems such as robotic hand-eye coordination and mobile robot navigation to more recent areas of interest such as user interfaces, gesture recognition, and surveillance.

One of the key problems in real-time vision is to track objects of interest through a series of images. There are two general classes of image processing algorithms used for this task: full-field image processing followed by segmentation and matching, and localized feature detection. Many tracking problems can be solved using either approach, but it is clear that the data-processing requirements for the solutions vary considerably. Full-frame algorithms such as optical flow calculation or region segmentation tend to lead to data intensive, processing which is performed offline or which is accelerated using specialized hardware (for a notable exception, see [36]). On the other hand, feature-based algorithms usually concentrate on spatially localized areas of the image. Since image processing is local, high data bandwidth between the host and the digitizer is not needed. The amount of data that must be processed is also relatively low and can be handled by sequential algorithms operating on standard computing hardware. Such systems are cost-effective and, since the tracking algorithms reside in software, extremely flexible and portable. Furthermore, as the speed of processors continues to increase, so does the complexity of the real-time vision applications that can be run on them. These advances anticipate the day when even full-frame applications requiring moderate processing can be run on standard hardware.

Local feature tracking has already found wide applicability in the vision and robotics literature. One of the most common applications is determining structure from motion. Structure from motion algorithms attempt to recover the three-dimensional structure of objects by observing their movement in multiple camera frames. Most often, this research involves observation of line segments [12, 30, 43, 45, 57], point features [41, 44], or both [14, 42], as they move in the image. As with stereo vision research, a basic necessity for recovering structure accurately is a solution to the correspondence problem: three-dimensional structure cannot be accurately determined without knowing which image features correspond to the same physical point in successive image frames. In this sense, precise local feature tracking is essential for the accurate recovery of three-dimensional structure.

Robotic hand-eye applications also make heavy use of visual tracking. Robots often operate in environments rich with edges, corners, and textures, making feature-based tracking a natural choice for providing visual input. Specific applications include calibration of cameras and robots [9, 28], visual-servoing and hand-eye coordination [10, 18, 25, 27, 56], mobile robot navigation and map-making [45, 55], pursuit of moving objects [10, 26], grasping [1], and telerobotics [23]. Robotic applications most often require the tracking of objects more complex than line segments or point features, and they frequently require the ability to track multiple objects. Thus, a tracking system for robotic applications must include a framework for composing simple features to track objects such as rectangles, wheels, and grippers in a variety of environments. At the same time, the fact that vision is in a servo loop implies that tracking must be fast, accurate, and highly reliable.

A third category of tracking applications are those which track modeled objects. Models may be anything from weak assumptions about the form of the object as it projects to the camera image (*e.g.* , contour trackers which assume simple, closed contours [8]) to full-fledged three-dimensional models with variable parameters (such as a model for an automobile which allows for turning wheels, opening doors, etc.). Automatic road-following has been accomplished by tracking the edges of the road [34]. Various snake-like trackers are used to track objects in 2D as they move across the camera image [2, 8, 11, 29, 46, 49, 54]. Three-dimensional models, while more complex, allow for precise pose estimation [17, 31]. The key problem in model-based tracking is to integrate simple features into a consistent whole, both to predict the configuration of features in the future and to evaluate the accuracy of any single feature.

While the list of tracking applications is long, the features used in these applications are variations on a very small set of primitives: "edgels" or line segments [12, 17, 30, 31, 43, 45, 49, 57], corners based on line segments [23, 41], small patches of texture [13], and easily detectable highlights [4, 39]. Although the basic tracking principles for such simple features have been known for some time, experience has shown that tracking them is most effective when strong geometric, physical, and temporal constraints from the surrounding task can be brought to bear on the tracking problem. In many cases, the natural abstraction is a multi-level framework where geometric constraints are imposed "top-down" while geometric information about the world is computed "bottom-up."

Although tracking is a necessary function for most of the research listed above, it is generally not a focus of the work and is often solved in an *ad hoc* fashion for the purposes of a single demonstration. This has led to a proliferation of tracking techniques which, although effective for particular experiments, are not practical solutions in general. Many tracking systems, for example, are only applied to pre-stored video sequences and do not operate in real time [40]. The implicit assumption is that speed will come, in time, with better technology (perhaps a reasonable assumption, but one which does not help those seeking real-time applications today). Other tracking systems require specialized hardware [1], making it difficult for researchers without such resources to replicate results. Finally, most, if not all, existing tracking methodologies lack modularity and portability, forcing tracking modules to be re-invented for every application.

Based on these observations, we believe that the availability of fast, portable, reconfigurable tracking system would greatly accelerate research requiring real-time vision tools. Just as the X Window system made graphical user interfaces a common feature of desktop workstations, an analogous "X Vision" system could make desktop visual tracking a standard tool in next generation computing. We have constructed such a system, called X Vision, both to study the science and art of visual tracking as well as to conduct experiments utilizing visual feedback. Experience from several teaching and research applications suggests that this system reduces the startup time for new vision applications, makes real-time vision accessible to "non-experts," and demonstrates that interesting research utilizing real-time vision can be performed with minimal hardware.

This article describes the philosophy and design of X Vision, focusing particularly on how geometric warping and geometric constraints are used to achieve high performance. We also present timing data for various tracking primitives and several demonstrations of X Vision-based systems. The remainder of the article is organized into four parts. Section 2 describes X Vision in some detail and Section 3 shows several examples of its use. The final section suggests some of the future directions for this paradigm, and we include an appendix which discusses some details of the software implementation.

2 Tracking System Design and Implementation

It has often been said that "vision is inverse graphics." X Vision embodies this analogy and carries it one step further by viewing *visual tracking as inverse animation.* In particular, most graphics or animation systems implement a few simple primitives, *e.g.*, lines and arcs, and define more complex objects in terms of these primitives. So, for example, a polygon may be decomposed into its polyhedral faces which are further decomposed into constituent lines. Given an object-viewer relationship, these lines are projected into the screen coordinate system and displayed. A good graphics system makes defining these types of geometric relationships simple and intuitive [15].

X Vision provides this functionality *and* its converse. In addition to stating how a complex object in a particular pose or configuration is decomposed into a list of primitive features, X Vision describes how the pose or attitude is computed given the locations of those primitives. More specifically, the system is organized around a small set of image-level primitives referred to as *basic features.* Each of these features is described in terms of a small set of parameters, referred to as a *state vector*, which completely specifies the features' positions and appearances. Complex features or objects carry their own state vectors which are computed by defining functions or constraints on a collection of simpler state vectors. These complex features may themselves participate in the construction of yet more complex features. Conversely, given the state vector of a complex feature, constraints are imposed on the state of its constituent features and the process recurses until image-level primitives are reached. The image-level primitives search for features in the neighborhood of their expected locations which produces a new state vector, and the cycle repeats.

In addition to being efficient and modular, X Vision provides facilities to simplify embedding of vision into applications. In particular, X Vision incorporates data abstraction that dissociates information carried in the feature state from the tracking mechanism used to acquire it.

2.1 Image-Level Feature Tracking

The primitive feature tracking algorithms of X Vision are optimized to be both accurate and efficient on scalar processors. These goals are met largely through two important attributes of X Vision. First, any tracking primitive operates on a relatively small "region of interest" within the image. Tracking a feature means that the region of interest retains a fixed, pre-defined relationship to the feature. In X Vision, a region of interest is referred to as a *window.* Fundamentally, the goal of low-level processing is to process the pixels within a window using a minimal number of addressing operations, bus transfer cycles, and arithmetic operations.

The second key idea is to employ *image warping* to geometrically transform windows so that image features appear in a canonical configuration. Subsequent processing of the warped window can then be simplified by assuming the feature is in or near this canonical configuration. As a result, the image processing algorithms used in feature-tracking can focus on the problem of accurate *configuration adjustment* rather than general-purpose feature detection. For example, consider locating a straight edge segment with approximately known orientation within an image region. Traditional feature detection methods utilize one or more convolutions, thresholding, and feature aggregation algorithms to detect edge segments. This is followed by a matching phase which utilizes orientation, segment length, and other cues to choose the segment which corresponds to the target [5]. Because the orientation and linearity constraints appear late in the detection process, such methods spend a large amount of time performing general purpose edge detection which in turn generates large amounts of data that must then be analyzed in the subsequent match phase. A more effective approach, as described in Section 2.1.2, is to exploit these constraints at the outset by utilizing a detector tuned for straight edges.

An additional advantage to warping-based algorithms is that they separate the "change of coordinates" needed to rectify a feature from the image processing used to detect it. On one hand, the same type of coordinate transforms, *e.g.* , rigid transformations, occur repeatedly, so the same warping primitives can be reused. On the other hand, various types of warping can be used to normalize features so that the same accelerated image processing can be applied over and over again. For example, quadratic warping could be used to locally "straighten" a curved edge so that an optimized straight edge detection strategy can be applied.

The low-level features currently available in X Vision include solid or broken contrast edges detected using several variations on standard edge-detection, general grey-scale patterns tracked using SSD (sum-of-squared differences) methods [3, 47], and a variety of color and motion-based primitives used for initial detection of objects and subsequent match disambiguation [51]. The remainder of this section describes how edge-tracking and correlation-based tracking have been incorporated into X Vision. In the sequel, all reported timing figures were taken on an SGI Indy workstation equipped with a 175Mhz R4400 SC processor and an SGI VINO digitizing system. Nearly equivalent results have been obtained for a Sun Sparc 20 equipped with a 70Mhz supersparc processor and for a 120MHz Pentium microprocessor, both with standard digitizers and cameras.

2.1.1 Warping

In the remainder of this article, we define *acquiring* a window to be the transfer *and* warping of the window's pixels. The algorithms described in this article use rigid and affine warping of rectangular image regions. The warping algorithms are based on the observation that a positive-definite linear transformation $\mathbf{A}$, can be written as a product of an upper-triangular matrix $\mathbf{U}$ and a rotation matrix $\mathbf{R}(\theta)$ as

$$\mathbf{A} = \mathbf{U}\ \mathbf{R}(\theta) = \begin{bmatrix} s_x & \gamma \\ 0 & s_y \end{bmatrix} \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

The implementation of image warping mirrors this factorization. First, a rotated rectangular area is acquired using an algorithm closely related to Bresenham algorithms for fast line rendering [15]. The resulting buffer can be subsequently scaled and sheared using an optimized bilinear interpolation algorithm. The former is relatively inexpensive, requiring about 2 additions per pixel to implement. The latter is more expensive, requiring 3 multiplies and 6 additions per pixel in our implementation. The initial acquisition is also parameterized by a sampling factor, making it possible to acquire decimated images at no additional cost. The warping algorithm supports reduction of resolution by averaging neighborhoods of pixels at a cost of one addition and $1/r$ multiplies per pixel for reduction by a factor of r. Figure 1 shows the time consumed by the three stages of warping (rotation, scale, and resolution reduction) on various size regions, and shows the effective time consumed for affine warping followed by resolution reduction to three different scales.

[1] The time taken for scaling varies with the amount of scaling done; these timings are for scaling the input image by a factor of $1/1.1$.

Size	20×20	40×40	60×60	80×80	100×100
Rotational Warping	0.11	0.40	0.94	1.77	2.83
Scale and Shear[1]	0.39	1.52	3.45	6.11	9.69
Resolution by 2	0.06	0.23	0.54	1.02	1.67
Resolution by 4	0.04	0.13	0.30	0.59	0.97
Affine	0.50	1.92	4.39	7.88	12.52
Affine by 2	0.56	2.15	4.93	8.90	14.19
Affine by 4	0.54	2.05	4.69	8.47	13.49

Figure 1. The time in milliseconds consumed image warping for various size regions. The first two lines show the time for each of the warping stages. The third and fourth lines show the time taken for reducing image resolution by a factor of 2 and 4. The final lines show the time needed for affine warping at various scales based on the component times.

For the purposes of later discussion, we denote an image region acquired at time t as $\mathcal{R}(t)$. The region containing the entire camera image at time t is written $\mathcal{I}(t)$. Warping operators operate on image regions to produce new regions. We write $\mathcal{R}(t) = \mathtt{warp}_{\mathrm{rot}}(\mathcal{I}(t); \mathbf{d}, \theta)$ to denote the acquisition of an image region centered at $\mathbf{d} = (x, y)^t$ and rotated by θ. Likewise, using the definition of $\mathbf{U}$ above, $\mathcal{R}'(t) = \mathtt{warp}_{\mathrm{ss}}(\mathcal{R}(t); \mathbf{U})$ denotes scaling the image region $\mathcal{R}(t)$ by s_x and s_y and shearing by γ. Affine warping is defined as

$$\mathtt{warp}_{\mathrm{aff}}(\mathcal{R}(t); \mathbf{A}, \mathbf{d}) = \mathtt{warp}_{\mathrm{ss}}(\mathtt{warp}_{\mathrm{rot}}(\mathcal{R}(t); \mathbf{d}, \theta); \mathbf{U})$$

where $\mathbf{A} = \mathbf{U}\,\mathbf{R}(\theta)$.

2.1.2 Edges

X Vision provides a tracking mechanism for linear edge segments of arbitrary length. The state of an edge segment consists of its position, $\mathbf{d} = (x, y)^t$, and orientation, θ, in framebuffer coordinates as well as its filter response r. Given prior state information $\mathbf{L}_t = (x_t, y_t, \theta_t, r_t)^t$, we can write the feature tracking cycle for the edge state computation at time $t + \tau$ schematically as

$$\mathbf{L}_{t+\tau} = \mathbf{L}_t + \mathtt{Edge}(\mathtt{warp}_{\mathrm{rot}}(\mathcal{I}(t+\tau); x_t, y_t, \theta_t); \mathbf{L}_t). \tag{1}$$

The edge tracking procedure can be divided into two stages: feature detection and state updating. In the detection stage, rotational image warping is used to acquire a window which, if the prior estimate is correct, leads to an edge which is vertical within the warped window. Detecting a straight, vertical contrast step edge can be implemented by convolving each row of the window with a derivative-based kernel, and averaging the resulting response curves by summing down the columns of the window. Finding the maximum value of this response function localizes the edge. Performance can be improved by noting that the order of the convolution and summation steps can be commuted. Thus, in an $n \times m$ window, edge localization with a convolution mask of width k can be performed with just $m \times (n + k)$ additions and mk multiplications. We, in fact, often use an IR filter composed of a series of -1s, one or more 0s, and a series of $+1$s which can be implemented using only $m \times (n+4)$ additions. We note that this is significantly cheaper than using, for example, steerable filters for this purpose [16].

The detection scheme described above requires orientation information to function correctly. If this information cannot be supplied from "higher-level" geometric constraints, it is estimated as follows (refer to Figure 3). As the orientation of the acquisition window rotates relative to the edge, the response of the filter drops sharply. Thus, edge orientation can be computed by sampling at several orientations and interpolating the responses to locate the direction of maximum response. However, implementing this scheme directly would be wasteful because the acquisition windows would overlap, causing many pixels to be transferred and warped three times. To avoid this overhead, an expanded window at the predicted orientation is acquired, and the summation step is repeated three times: once along the columns, and once along two diagonal paths at a small angular offset from vertical. This effectively approximates rotation by image shear, a well-known technique in graphics [15]. Quadratic interpolation of the maximum of the three curves is used to estimate the orientation of the underlying edge. In the ideal case, if the convolution template is symmetric and the response function after superposition is unimodal, the horizontal displacement of the edge should agree between all three filters. In practice, the estimate of edge location will be biased. For this reason, edge location is computed as the weighted average of the edge location of all three peaks.

Even though the edge detector described above is quite selective, as the edge segment moves through clutter, we can expect multiple local maxima to appear in the convolution output. This is a well-known and unavoidable problem for which many solutions have been proposed [38]. By default, X Vision declares a match if and only if a unique local maximum exists within an interval about the response value stored in the state. The match interval is chosen as a fraction of the difference between the matched response value and its next closest response in the previous frame. This scheme makes it extremely unlikely that mistracking due to incorrect matching will occur. Such an event could happen only if some distracting edge of the correct orientation and response moved into the tracking window just as the desired edge changed response or moved out of the tracking window. The value of the threshold determines how selective the filter is. A narrow match band implicitly assumes that the edge response remains constant over time, a problem in environments with changing backgrounds. Other possibilities include matching on the brightness of the "foreground" object or matching based on nearness to an expected location passed from a higher-level object. Experimental results on line tracking using various match functions can be found in [49].

The result of the image processing stage is to compute an offset normal to the edge orientation, δt, and an orientation offset $\delta\theta$. Given these values, the geometric parameters of the edge tracker are updated according to the following equation:

$$\begin{bmatrix} x_{t+\tau} \\ y_{t+\tau} \\ \theta_{t+\tau} \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} + \begin{bmatrix} -\delta t \sin(\theta_t + \delta\theta) \\ \delta t \cos(\theta_t + \delta\theta) \\ \delta\theta \end{bmatrix}. \tag{2}$$

Because of the aperture problem, the state vector is not fully determined by information returned from feature detection. There is nothing to keep the window from moving "along" the edge that it is tracking. For this reason, the edge tracking primitive almost always participates in a composite feature that imposes additional constraints on its state (see Section 2.2).

We note that edge tracking robustness can be increased by making edge segments as long

Figure 2. Close-up of tracking windows at two time points. Left, time t_i, where the edge tracking algorithm has computed the correct warp parameters to make an edge appear vertical (the "setpoint"). Right, the edge acquired at time t_{i+1}. The warp parameters computed for t_i were used to acquire the image, but the underlying edge has changed orientation. Figure 3 shows how the new orientation is computed.

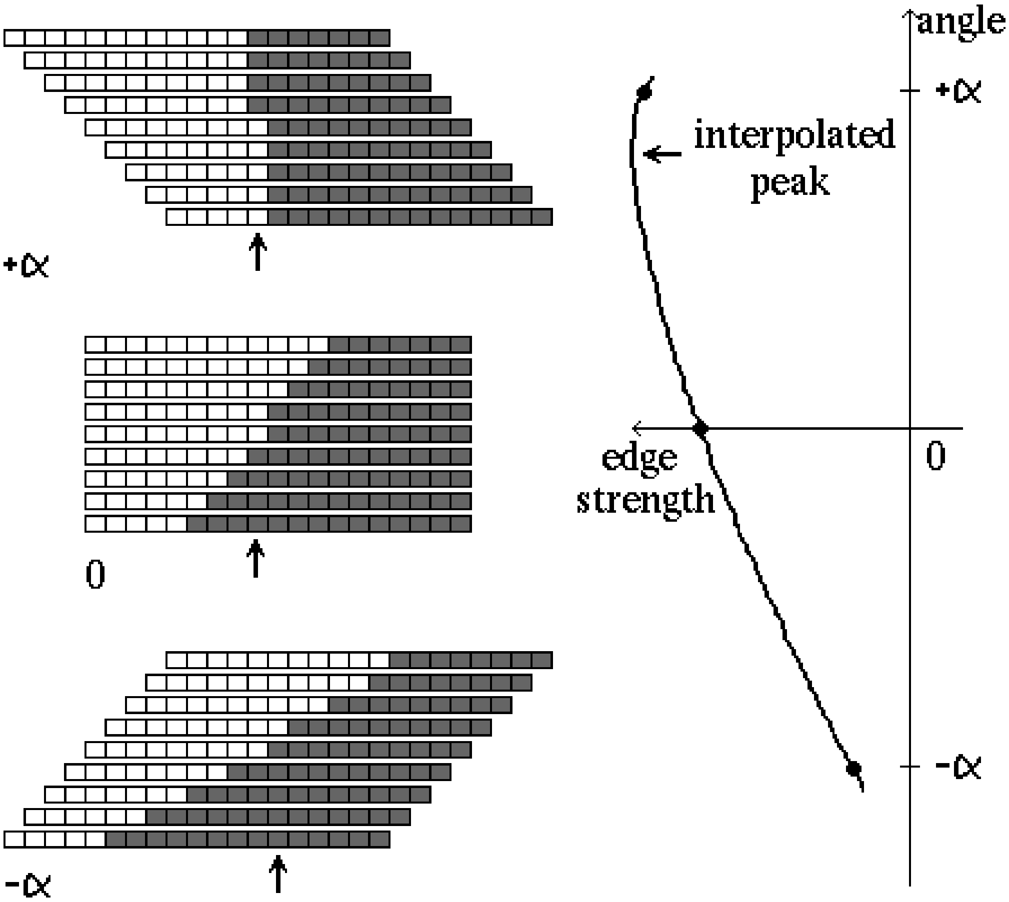

Figure 3. Schematic for computing edge orientation. The diagrams on the left show a window of pixels at three different "orientations." The middle figure displays the edge after a warped acquisition (Figure 2, right). The top and bottom figures show the effect of shifting rows to simulate orientational offset. Summing the columns for each figure and taking differences between adjacent sets of columns gives estimates for edge strength. The arrows at the bottom show where each image experiences the strongest vertical edge within the window. At the right, these values are plotted with angle offset on the vertical axis and edge strengths on the horizontal axis. The three data points are fit to a quadratic, whose peak offers an estimate for the best angular offset of the actual edge. (Both the orientation of the edge in the middle figure and the extent of the shearing in the top and bottom figures have been exaggerated for illustrative purposes.)

Line	Length Sampling		
Length, Width	Full	1/2	1/4
20, 20	0.39	0.29	0.20
40, 20	0.71	0.41	0.26
60, 20	1.13	0.59	0.35
20, 30	0.56	0.34	0.27
40, 30	0.93	0.55	0.35
60, 30	1.55	0.77	0.47
20, 40	0.65	0.43	0.32
40, 40	1.17	0.66	0.45
60, 40	2.09	0.97	0.57

Figure 4. Time in milliseconds required for one iteration of tracking an edge segment.

as possible [50]. Long segments are less likely to become completely occluded, and changes in the background tend to affect a smaller proportion of the segment with a commensurately lower impact on the filter response. On long edge segments, speed is maintained by subsampling the window in the direction of the edge segment. Likewise, the maximum edge motion between images can be increased by subsampling in the horizontal direction. In this case, the accuracy of edge localization drops and the possibility of an ambiguous match increases.

Figure 4 shows timings for simple edge tracking that were obtained during test runs. Length and width refer to the length of the tracked edge and the width of the search region normal to the edge in pixels, respectively. One interesting point is that, because superimposing the columns of the image is performed before the convolution step, processing speed is sublinear with edge length. For example, moving from 20 pixel edges to 40 pixel edges results in only a 65% increase in time. Also, tracking a 40 pixel segment at half resolution takes the same amount of time as a 20 pixel segment at full resolution because the warping operator implements decimation efficiently. Finally, if we consider tracking 20 pixel edge segments at full resolution, we see that it is possible to track up to $33.33/0.44 \approx 85$ segments simultaneously at frame rate.

2.1.3 Region-Based Tracking

In region-based tracking, we consider matching a stored reference window to a region of the image. The reference is either a region taken from the scene itself, or a pre-supplied target template. It is assumed throughout that the surface patch corresponding to the region of interest is roughly planar and that its projection is relatively small compared to the image as a whole so that perspective effects are minimal. Under these circumstances, the geometric distortions of a region are well modeled by an affine transformation consisting of a 2×1 translation vector, $\mathbf{d} = (u, v)^t$, and a positive definite 2×2 matrix, $\mathbf{A}$. The state vector for our region tracker includes these six geometric parameters and a residual value, r, which indicates how well the reference region and the current image region match in a least-squares sense. Defining $\mathbf{S}_t = (\mathbf{A}_t, \mathbf{d}_t, r_t)$ to denote the state vector parameters at time t, the computation of the state at time $t+\tau$ can be written schematically as

$$\mathbf{S}_{t+\tau} = \mathbf{S}_t + \mathtt{SSD}(\mathtt{warp}_{\text{aff}}(\mathcal{I}(t+\tau); \mathbf{A}_t, \mathbf{d}_t); \mathbf{S}_t). \tag{3}$$

As before, previous state information of the reference region is used to acquire and warp a prospective image region. Once this transformation is performed, computing the remaining geometric differences between reference and the prospective regions is posed as a sum-of-squared differences (least-squares) optimization problem similar to that of stereo matching [33]. We note that this approach to region tracking is not itself new, however previous implementations were based on computing and integrating interframe motion [48, 40], did not operate in real-time [6, 7], or computed a subset of the affine parameters [37]. Our region tracking uses the initial reference region throughout the image sequence to provide a fixed "setpoint" for the algorithm, and it computes up to full affine image deformations at or near frame rate.

Let $I(\mathbf{x}, t)$ denote the value of the pixel at location $\mathbf{x} = (x, y)^t$ at time t in an image sequence. Consider a planar surface patch undergoing rigid motion observed under orthographic projection. At time t_0, the surface projects to an image region $\mathcal{R}(t_0)$, subsequently referred to as the *target region*, with a spatial extent represented as a set of image locations, $\mathcal{W}$. At some later point, $t > t_0$, the region projects to an affine transformation of the original region. If illumination remains constant, the geometric relationship between the projections can be recovered by minimizing the following objective function:

$$O(\mathbf{A}, \mathbf{d}) = \sum_{\mathbf{x} \in \mathcal{W}} (I(\mathbf{A}\mathbf{x} + \mathbf{d}, t) - I(\mathbf{x}, t_0))^2 w(\mathbf{x}), \quad t > t_0, \tag{4}$$

where $\mathbf{A}$ and $\mathbf{d}$ are as described above, $\mathbf{x} = (x, y)^t$, and $w(\cdot)$ is an arbitrary positive weighting function.

Suppose that a solution at time t, $(\mathbf{A}_t, \mathbf{d}_t)$, is known, and the goal is to compute the solution at time $t + \tau$ for small positive τ. Since we apply affine warping to the image of time $t + \tau$, it is useful to define

$$\begin{aligned} \mathbf{A}_{t+\tau} &= \mathbf{A}_t(\mathbf{I} + \mathbf{A}') && (5) \\ \mathbf{d}_{t+\tau} &= \mathbf{d}_t + \mathbf{A}_t \mathbf{d}', && (6) \end{aligned}$$

where $\mathbf{A}'$ and $\mathbf{d}'$ represent incremental changes during the interval τ. Substituting into (4) we have

$$O(\mathbf{A}_{t+\tau}, \mathbf{d}_{t+\tau}) = \sum_{\mathbf{x} \in \mathcal{W}} (I(\mathbf{A}_t(\mathbf{I} + \mathbf{A}')\mathbf{x} + \mathbf{A}_t\mathbf{d}' + \mathbf{d}_t, t + \tau) - I(\mathbf{x}, t_0))^2 w(\mathbf{x}). \tag{7}$$

We now introduce the "warped image," $J(\mathbf{x}, t) = I(\mathbf{A}_t\mathbf{x} + \mathbf{d}_t, t + \tau)$, and write a new objective function $O'(\cdot)$ in terms of $\mathbf{J}$, $\mathbf{A}'$ and $\mathbf{d}'$:

$$O'(\mathbf{A}', \mathbf{d}') = \sum_{\mathbf{x} \in \mathcal{W}} (J(\mathbf{x} + \mathbf{A}'\mathbf{x} + \mathbf{d}', t + \tau) - I(\mathbf{x}, t_0))^2 w(\mathbf{x}). \tag{8}$$

Solving (8) proceeds by linearizing J about the point $(\mathbf{A}', \mathbf{d}') = 0$, yielding

$$O'(\mathbf{A}', \mathbf{d}') = \sum_{\mathbf{x} \in \mathcal{W}} (J(\mathbf{x}, t) \ + \ \nabla J(\mathbf{x}, t) \cdot (\mathbf{A}'\mathbf{x} + \mathbf{d}') \ - \ I(\mathbf{x}, t_0))^2 w(\mathbf{x}), \quad \tau > 0, \tag{9}$$

where $\nabla J = (J_x, J_y)^t$ is the warped image spatial gradient.

If the solution at t is nearly the correct one, then $J(\mathbf{x}, t) \approx I(\mathbf{x}, t)$, and hence $\nabla J \approx \nabla I = (I_x, I_y)^t$, where I_x and I_y are spatial gradients of the original image. With this observation, we can simplify (9) and rewrite it in terms of the spatial derivatives of the reference image yielding

$$O'(\mathbf{A}', \mathbf{d}') = \sum_{\mathbf{x} \in \mathcal{W}} (\nabla I(\mathbf{x}, t) \cdot (\mathbf{A}'\mathbf{x} + \mathbf{d}') \ + \ (J(\mathbf{x}, t) - I(\mathbf{x}, t_0)))^2 w(\mathbf{x}). \tag{10}$$

In this form, the problem can be solved by joint optimization over all six unknowns in $\mathbf{A}'$ and $\mathbf{d}'$. However, one difficulty with computing affine structure lies in the fact that many target regions do not have enough texture to fully determine all six geometric parameters [40]. Consider, for example, a window placed on a right-angle corner. A pure translation of the corner can be accounted for as translation, scaling or a linear combination of both. The solution implemented in X Vision is based on the observation that the image structure which determines translation and rotation is similar to that which determines scale and shear. In general, translation and rotation are much more rapidly changing parameters than are scale and shear. In ambiguous situations these parameters should be the preferred interpretation for image changes.

To implement this solution, we decompose $\mathbf{A}'$ into a differential rotation and an upper triangular matrix:

$$\mathbf{A}' = \begin{bmatrix} 0 & \alpha \\ -\alpha & 0 \end{bmatrix} + \begin{bmatrix} s_x & \gamma \\ 0 & s_y \end{bmatrix}$$

and solve for two parameter groups, $(\mathbf{d}, \alpha)$ and (s_x, s_y, γ), sequentially. This establishes preferences for interpreting image changes (the result being that some image perturbations result in short detours in the state space before arriving at a final state estimate). Although less accurate than a simultaneous solution, the small amount of distortion between temporally adjacent images makes this solution method sufficiently precise for most applications.

We first solve for translation and rotation. For an image location $\mathbf{x} = (x, y)^t$ we define

$$\begin{aligned} g_x(\mathbf{x}) &= I_x(\mathbf{x}, t_0)\sqrt{w(\mathbf{x})} && (11) \\ g_y(\mathbf{x}) &= I_y(\mathbf{x}, t_0)\sqrt{w(\mathbf{x})} \\ g_r(\mathbf{x}) &= (y\ I_x(\mathbf{x}, t_0) - x\ I_y(\mathbf{x}, t_0))\sqrt{w(\mathbf{x})} \\ h_0(\mathbf{x}) &= (J(\mathbf{x}, t) - I(\mathbf{x}, t_0))\sqrt{w(\mathbf{x})}, && (12) \end{aligned}$$

and the linear system for computing translation and rotation is

$$\sum_{\mathbf{x} \in \mathcal{W}} \begin{bmatrix} g_x g_x & g_x g_y & g_x g_r \\ g_y g_x & g_y g_y & g_y g_r \\ g_r g_x & g_r g_y & g_r g_r \end{bmatrix} \begin{bmatrix} \mathbf{d} \\ \alpha \end{bmatrix} = \sum_{\mathbf{x} \in \mathcal{W}} \begin{bmatrix} h_0\ g_x \\ h_0\ g_y \\ h_0\ g_r \end{bmatrix}. \tag{13}$$

Since the spatial derivatives are only computed using the original reference image, g_x, g_y, and g_r are constant over time, so those values and the inverse of the matrix on the left hand side of (13) can be computed offline.

Once $\mathbf{d}$ and α are known, the least squares residual value is computed as

$$h_1(\mathbf{x}) = h_0(\mathbf{x}) - g_x(\mathbf{x})u - g_y(\mathbf{x})v - g_r(\mathbf{x})\alpha. \tag{14}$$

If the image distortion arises from pure translation and no noise is present, then we expect that $h_1(\mathbf{x}) = 0$ after this step. Any remaining residual can be attributed to geometric distortions in the second group of parameters, linearization error or noise. To recover scale changes and shear, we define for an image location $\mathbf{x} = (x, y)^t$

$$\begin{aligned} g_{sx}(\mathbf{x}) &= x\, g_x(\mathbf{x}), & (15)\\ g_{sy}(\mathbf{x}) &= y\, g_y(\mathbf{x}), \\ g_\gamma(\mathbf{x}) &= y\, g_x(\mathbf{x}), & (16) \end{aligned}$$

and the linear system for computing scaling and shear parameters is

$$\sum_{\mathbf{x}\in\mathcal{W}} \begin{bmatrix} g_{sx}g_{sx} & g_{sx}g_{sy} & g_{sx}g_\gamma \\ g_{sy}g_{sx} & g_{sy}g_{sy} & g_{sy}g_\gamma \\ g_\gamma g_{sx} & g_\gamma g_{sy} & g_\gamma g_\gamma \end{bmatrix} \begin{bmatrix} s_x \\ s_y \\ \gamma \end{bmatrix} = \sum_{\mathbf{x}\in\mathcal{W}} \begin{bmatrix} h_1\, g_{sx} \\ h_1\, g_{sy} \\ h_1\, g_\gamma \end{bmatrix}. \tag{17}$$

As before, g_{sx}, g_{sy}, and g_γ can be precomputed as can the inverse of the matrix on the left hand side. The residual is

$$h_2(\mathbf{x}) = h_1(\mathbf{x}) - g_{sx}(\mathbf{x})s_x - g_{sy}(\mathbf{x})s_y - g_r(\mathbf{x})\gamma. \tag{18}$$

After all relevant stages of processing have been complete,

$$r = \frac{\sqrt{\sum_{\mathbf{x}\in\mathcal{W}} h_2(\mathbf{x})^2}}{|\mathcal{W}|} \tag{19}$$

is stored as the match value of the state vector.

One potential problem with this approach is that the brightness and contrast of the target are unlikely to remain constant which may bias the results of the optimization. The solution is to normalize images to have zero first moment and unit second moment. We note that with these modifications, solving (4) for rigid motions (translation and rotation) is equivalent to maximizing normalized correlation [24]. Extensions to the SSD-based region tracking paradigm for more complex lighting models can be found in [21].

Another problem is that the image gradients are only locally valid. In order to guarantee tracking of motions larger than a fraction of a pixel, these calculations must be carried out at varying levels of resolution. For this reason, a software reduction of resolution is carried out at the time of window acquisition. All of the above calculations except for image scaling are computed at the reduced resolution, and the estimated motion values are appropriately rescaled. The tracking algorithm changes the resolution adaptively based on image motion. If the computed motion value for either component of $\mathbf{d}'$ exceeds 0.25, the resolution for the subsequent step is halved. If the interframe motion is less than 0.1, the resolution is doubled. This leads to a fast algorithm for tracking fast motions and a slower but more accurate algorithm for tracking slower motions.

If we consider the complexity of tracking in terms of arithmetic operations on pixels (asymptotically, these calculations dominate the other operations needed to solve the linear system) we see that there is a fixed overhead of one difference and multiply to compute h_0. Each parameter computed requires an additional multiply and addition per pixel. Computing the residual values

Size	40 × 40		60 × 60		80 × 80		100 × 100	
Reduction	4	2	4	2	4	2	4	2
Rigid	1.5	5.6	3.2	9.5	6.1	17.7	9.4	28.3
Affine	3.7	8.4	8.1	15.6	14.4	28.5	22.5	43.1

Figure 5. The time in milliseconds consumed by one cycle of tracking for various instantiations of the SSD tracker. The first row shows the timings for rigid motion (translation and rotation), and the second row shows the time for full affine deformations.

consumes a multiply and addition per pixel per parameter value. In addition to parameter estimation, the initial brightness and contrast compensation consume three additions two multiplies per pixel. Thus, to compute the algorithm at a resolution d requires $15/d^2$ multiplies and $1 + 16/d^2$ additions per pixel (neglecting warping costs). It is interesting to note that at a reduction factor of $d = 4$, the algorithm compares favorably with edge detection on comparable sized regions.

To get a sense of the time consumed by these operations, several test cases are shown in Figure 5. The first row shows the time needed to track rigid motions (translation and rotation) and the second shows the time taken for tracking with full affine deformations. The times include both warping and parameter estimation; the times given in Figure 1 can be subtracted to determine the time needed to estimate parameter values. In particular, it is important to note that, because the time consumed by affine warping is nearly constant with respect to resolution, parameter estimation tends to dominate the computation for half and full resolution tracking, while image warping tends to dominate the computation for lower resolution tracking. With the exception of 100×100 images at half resolution, all updates require less than one frame time (33.33 ms.) to compute. Comparing with Figure 4, we observe that the time needed to track a 40×40 region at one-fourth resolution is nearly equivalent to that needed to track a comparably-sized edge segment as expected from the complexity analysis given above.

To get a sense of the effectiveness of affine tracking, Figure 6 shows several images of a box as a 100×100 region on its surface was tracked at one-fourth resolution. The lower series of images is the warped image which is the input to the SSD updating algorithm. We see that except for minor variations, the warped images are identical despite the radically different poses of the box.

2.2 Networks of Features

One goal of X Vision is to make it simple to quickly prototype tracking systems from existing components, and then to add application-specific constraints quickly and cleanly. This is accomplished by extending the state-based representation used in image-level features with additional infrastructure to support hierarchical imposition of geometric and physical constraints on feature evolution.

More specifically, we define *composite features* to be features that compute their state from other basic and composite features. We allow two types of feature composition. In the first case, information flow is purely "bottom-up." Features are combined solely to compute information from their state vectors without altering their tracking behavior. For example, given two point features it may be desirable to present them as the line feature passing through both. A feature

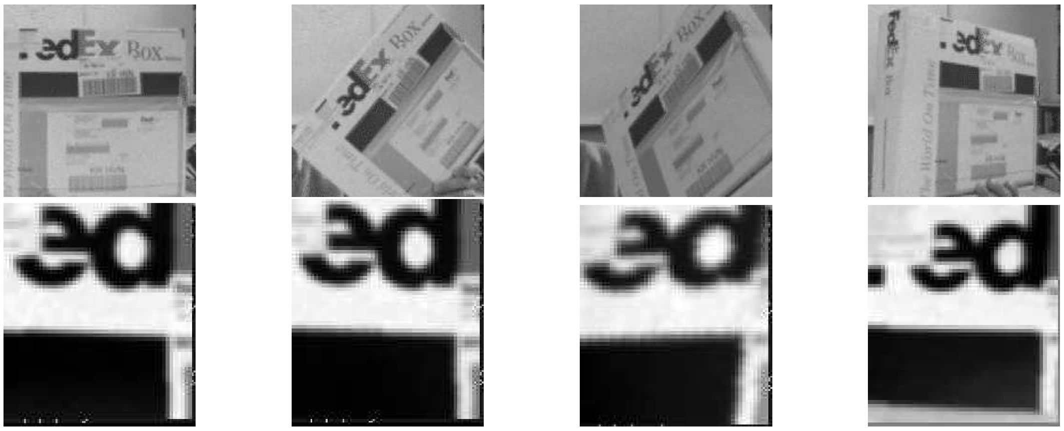

Figure 6. Several images of a planar region and the corresponding warped image used by the tracker. The image at the left is the initial reference image.

(henceforth, *feature* refers to both basic and composite features) can participate in any number of such constructions. In the second case, the point of performing feature composition is to exploit higher level geometric constraints in tracking as well as to compute a new state vector. In this case, information flows both upward and downward.

We further define a *feature network* to be a set of nodes connected by arcs directed either upward or downward (a feature and its subsidiary feature can be linked in both directions). Nodes represent features, and links represent the information dependency between a composite feature and the features used to compute its state. To implement these functions, we associate a `state-computation` procedure with the incoming links to a node, and a `constraint-propagation` procedure with the outgoing links.

A complete feature tracking cycle consists of: 1) traversing the downward links from each top-level node by executing the associated `constraint-propagation` procedure until basic features are reached; 2) performing low-level detection in every basic feature; and 3) traversing the upward links of the graph by executing the `state-computation` procedure of each node. State prediction can be added to this cycle by including it in the downward constraint propagation phase. Thus, a feature tracking system is completely characterized by the topology of the network, the identity of the basic features, and the state computation and constraint propagation functions for each non-basic feature node.

A concrete example is a feature tracker for the intersection of two non-collinear contours. This composite feature has a state vector $C = (x, y, \theta, \alpha)^T$ describing the position of the intersection point, the orientation of one contour, and the orientation difference between the two contours. The `constraint-propagation` function for corners is implemented as follows. From image edges with state $\mathbf{L}_1 = (x_1, y_1, \theta_1, r_1)^T$ and $\mathbf{L}_2 = (x_2, y_2, \theta_2, r_2)^T$, the distance from the center of each tracking window to the point of intersection the two edges can be computed as

$$\begin{aligned}
\lambda_1 &= ((x_2 - x_1)\sin(\theta_2) - (y_2 - y_1)\cos(\theta_2))/\sin(\theta_2 - \theta_1), \\
\lambda_2 &= ((x_2 - x_1)\sin(\theta_1) - (y_2 - y_1)\cos(\theta_1))/\sin(\theta_2 - \theta_1).
\end{aligned}$$

Given a known corner state vector, we can choose "setpoints" λ_1^* and λ_2^* describing where to position the edge trackers relative to the intersection point. With this information, the states of the individual edges can be adjusted as follows:

$$\begin{aligned} x_i &= x_c - \lambda_i^* \cos(\theta_i), \\ y_i &= y_c - \lambda_i^* \sin(\theta_i), \end{aligned} \tag{20}$$

for $i = 1, 2$. Choosing $\lambda_1^* = \lambda_2^* = 0$ defines a cross pattern. If the window extends h pixels along the edge, choosing $\lambda_1^* = \lambda_2^* = h/2$ defines a corner. Choosing $\lambda_1^* = 0$ and $\lambda_2^* = h/2$ defines a tee junction, and so forth.

Conversely, given updated state information for the component edges, the `state-computation` function computes:

$$\begin{aligned} x_c &= x_1 + \lambda_1 \cos(\theta_1), \\ y_c &= y_1 + \lambda_1 \sin(\theta_1), \\ \theta_c &= \theta_1, \\ \alpha_c &= \theta_2 - \theta_1. \end{aligned} \tag{21}$$

The tracking cycle for this system starts by using prior predictions of corner state to impose the constraints of (20) downward. Image-level feature detection is then performed, and finally information is propagated upward by computing (21).

Composite features that have been implemented within this scheme range from simple edge intersections as described above, to snake-like contour tracking [49], to three-dimensional model-based tracking using pose estimation [32], as well as a variety of more specialized object trackers, some of which are described in Section 3.

2.3 Feature Typing

In order to make feature composition simpler and more generic, we have included polymorphic type support in the tracking system. Each feature, basic or composite, carries a type. This type identifies the geometric or physical information contained in the state vector of the feature. For example, there are *point features* which carry location information and *line features* which carry orientation information.

Any composite feature can specify the type of its subsidiary features and can itself carry a type. In this way, the construction becomes independent of a manner with which its subsidiary nodes compute information. So, for example, a line feature can be constructed from two point features by computing the line that passes through the features and a point feature can be computed by intersecting two line features. An instance of the intersection-based point feature can be instantiated either from edges detected in images or line features that are themselves computed from point features.

3 Applications

We have used X Vision for several purposes including hand-eye coordination [19, 20, 22], a pose-based object tracking system [32], a robust face-tracking system [51], a gesture-based drawing program, a six degree-of-freedom mouse [52], and a variety of small video games. In this section, we describe some applications of X Vision which illustrate how the tools it provides—particularly image warping, image subsampling, constraint propagation, and typing—can be used to quickly prototype fast and effective tracking systems.

line length (pixels)	sampling rate	tracking speed (msec/cycle)		σ of position (pixels)	
		A	B	A	B
24	1	9.3	7.7	0.09	0.01
12	2	5.5	4.5	0.10	0.00
8	3	3.7	3.3	0.07	0.03
6	4	3.0	2.7	0.05	0.04
2	12	1.9	1.6	0.05	0.04
1	24	1.5	1.3	0.05	0.07

Figure 7. Speed and accuracy of tracking rectangles with various spatial sampling rates. The figures in column A are for a tracker based on four corners computing independent orientation. The figures in column B are for a tracker which passes orientation down from the top-level composite feature.

3.1 Pure Tracking Applications

Edge-Based Disk Tracking One important application for any tracking system is model-based tracking of objects for applications such as hand-eye coordination or virtual reality. While a generic model-based tracker for three-dimensional objects for this system can be constructed [32], X Vision makes it possible to gain additional speed and robustness by customizing the tracking loop using object-specific geometric information.

On example of this customization process is the development of a tracker for rectangular floppy disks that we use as test objects in our hand-eye experiments (described below). Given the available tools, the most straightforward rectangle tracker is a composite tracker which tracks four corners, which in turn are composite trackers which track two lines each as described in Section 2.1.2. No additional constraints are imposed on the corners.

This method, while simple to implement, has two obvious disadvantages First, in order to track quickly, only a small region of the occluding contour of the disk near the corners is processed. This makes them prone to mistracking through chance occlusion and background distractions. Second, each of the line computations is independently computing orientation from image information, making the tracking relatively slow. The first problem is handled by increasing the effective length of the edge trackers by operating them on subsampled windows. The second problem is solved by adding additional constraints to the system. The composite feature for the rectangle computes the orientation of the lines joining the corners and passes this information down to the image-level edge trackers. The edge trackers then do not need to compute orientation from image information. The net effect of these two changes is to create a highly constrained snake-like contour tracker [29].

Figure 7 contains some timing and accuracy statistics for the resulting algorithm. It shows that there is little or no loss of precision in determining the location of the corners with reasonable sampling rates. At the same time, we see a 10% to 20% speedup by not computing line orientations at the image level and a nearly linear speedup with image subsampling level.

Region-Based Face Tracking A frontal view of a human face is sufficiently planar to be successfully tracked as a single SSD region. Figure 8 shows several image pairs illustrating poses of a face and the warped image resulting from tracking. Despite the fact that the face is nonplanar, resulting for example in a stretching of the nose as the face is turned, the tracking is quite

Figure 8. Above several images of the a face and below the corresponding warped images used by the tracking system.

effective. However, tracking a face as a single region requires affine warping over a relatively large region which is somewhat slow (about 40 milliseconds per iteration). It can be confused if the face undergoes distortions which cannot be easily captured by affine deformation, and it is sensitive to lighting variations and shadowing. Also, many areas of the face contain no strong gradients, thus contributing little to the state computation.

Figure 9 shows the structure of a more specialized tracking arrangement that uses SSD trackers at the regions of highest contrast — the eyes and mouth. The result is a gain in performance as well as the ability to recognize isolated changes in the underlying features. For each of the eyes and the mouth, a `MultiSSD` composite tracker performs an SSD computation for multiple reference images. The state of the `MultiSSD` tracker is computed to be the state of the tracker with the best match value and the numeric identity of this tracker. The constraint function copies this state back down to the component SSD trackers, forcing "losing" SSD trackers follow the "winner." In effect, the `MultiSSD` feature is a tracker with an n-ary switch.

From `MultiSSD` we derive an `Eye` tracker which modifies the display function of `MultiSSD` to show an open or closed eye based on the state of the binary switch. We also derive `Mouth` which similarly displays an open or closed mouth. Two `Eye`'s compose the `Eyes` tracker. The state computation function of `Eyes` computes the orientation of the line joining the eyes, which is propagated to the lower-level `Eye` trackers, obviating the need to compute orientation from image-level information. Thus, the low-level `Eye` trackers only solve for translation, much as the disk tracker described above. Since the mouth can move independently, `Mouth` computes both translation and orientation directly from the image. Finally, the `Face` tracker comprises `Eyes` and `Mouth`. It imposes no constraints on them, but it does interpolate the position of the nose based on the `Eyes` and `Mouth`.

The tracker is initialized by indicating the positions of the eyes and mouth and memorizing their appearance when they are closed and open. When run, the net effect is a graphical display of a "clown face" that mimics the antics of the underlying human face — the mouth and eyes follow those of the operator and open and close as the operator's do as shown in Figure 10. This system

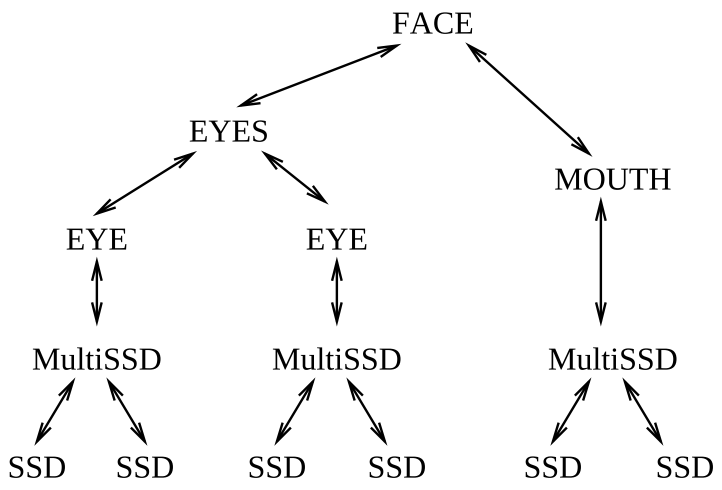

Figure 9. The tracking network used for face tracking.

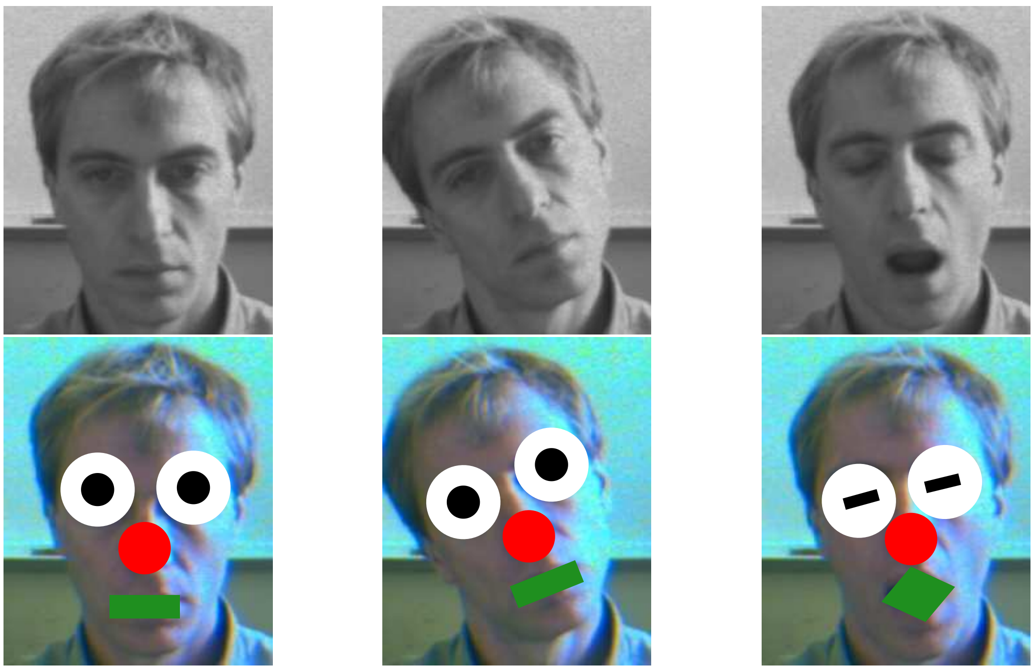

Figure 10. The “clown face” tracker.

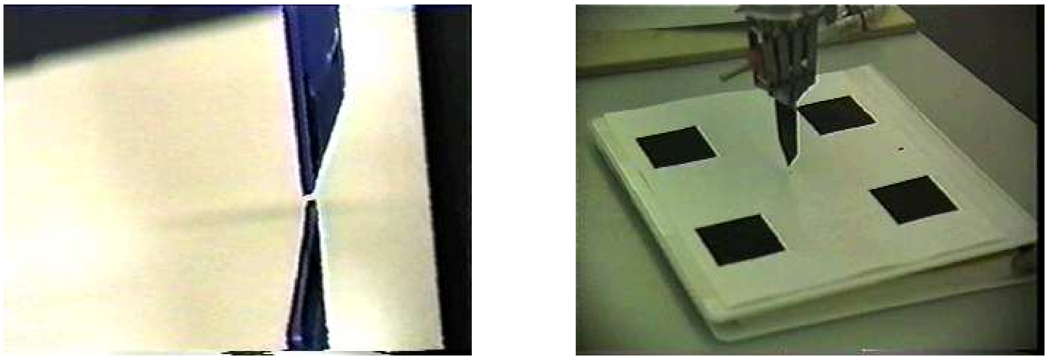

Figure 11. The results of performing point-to-point positioning to observable features (left) and to a setpoint defined in the plane (right).

requires less than 10 milliseconds per iteration without graphics.

3.2 An Embedded Application

As an illustration of the use of X Vision embedded within a larger system, we describe some results of using X Vision within a hand-eye coordination system [20, 22, 53]. The system relies on image-level feedback from two cameras to control the relative pose between an object held in a robot end-effector and a static object in the environment.

The typing capabilities of X Vision make it possible to abstract the function of the hand-eye coordination primitives from their visual inputs. The hand-eye system implements a set of *primitive skills* which are vision-based regulators for attaining a particular geometric constraint between the pose of a robot-held object and a target object. These skills are defined on corresponding features tracked simultaneously in two cameras. For example, two primitive skills are point-to-point positioning of the robot end-effector based on tracking point features, and point-to-line positioning of the end-effector using point and line features. Using X Vision's typing scheme, these feedback methods can be implemented in terms of feature type, and used with a variety of application-specific tracking configurations without change. For example, positioning or orienting the robot in a plane is practically useful for systems which use a table or other level surface as a work-space. In order to relate task coordinates to image coordinates, a planar invariant employing cross-ratios can be used [35]. This construction is implemented as a composite feature composed of several corner trackers. However, it is typed as a point feature, so it can be coupled directly with a point positioning skill in order to perform planar positioning as shown in Figure 11 (right).

It can be shown that the accuracy of primitive skills depends only on the accuracy of feature location in the image [20]. Hence, the physical accuracy of hand-eye experiments can be used to directly determine the accuracy of our feature localization algorithms. We have performed several hundred point-to-point positioning experiments with a camera baseline of approximately 30cm at distances of 80 to 100cm. Accuracy is typically within a millimeter of position. For example, Figure 11 (left) shows the accuracy achieved when attempting to touch the corners of two floppy disks. For reference, the width of the disks is 2.5mm. Simple calculations show that positioning accuracy of a millimeter at one meter of depth with a baseline of 30cm using the physical parameters of our cameras yields a corner localization accuracy of ± 0.15 pixels.

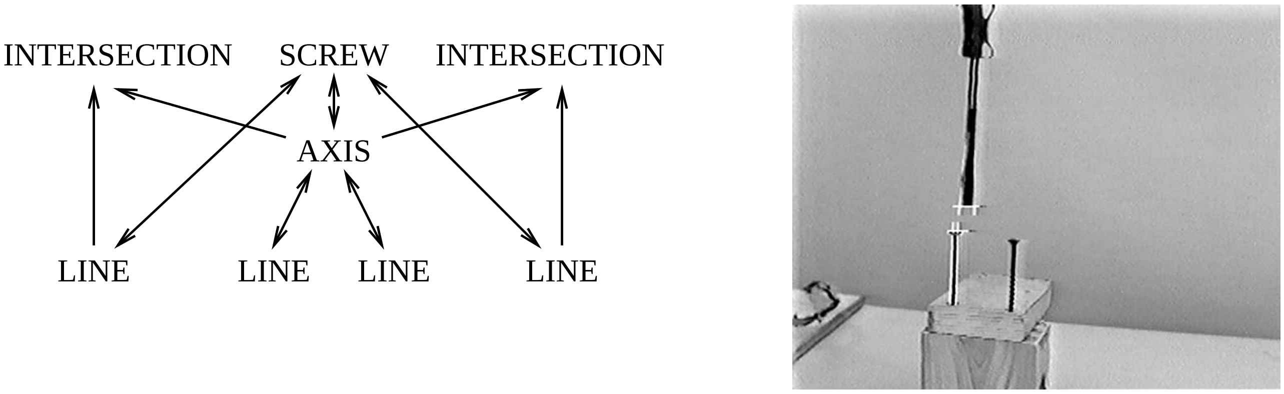

Figure 12. Left, the network of the screw tracker. Right, the system in operation.

Primitive skills are often combined to form more complex kinematic constraints. For example, two point-to-line constraints define a colinearity constraint useful for alignment. Figure 12 (right) shows an example of a problem requiring alignment—placing a screwdriver onto a screw. The diagram to the left summarizes the tracker for the screw. Computations of state based on the sides of the screw in both cameras allow the screw's central axis to be tracked. In addition, the head of the screw is tracked by two short line trackers.

For the purposes of hand-eye coordination, the screw tracker is augmented with two composite features which compute the intersection point of the central axis with each of the short line segments. These point features are used as input the visual servoing algorithm. This is a common use of such geometric constructions — to extract information from an object tracker which has its own internal structure and constraints.

4 Conclusions

We have presented the X Vision system for flexible, efficient visual tracking. The main features of this system are the strong use of image warping, highly optimized low-level tracking methods, and a simple notion of feature combination. Our experience has shown that X Vision makes the use of visual tracking in applications cheap, simple, and even fun. Beginning users tend to become proficient after a short "startup" period. Experts can easily develop and debug complex applications in a few hours' time. The modularity of the system has also made it an ideal framework for comparative studies. It is straightforward to add new tracking primitives to the system and benchmark them against existing methods on real images or on canned sequences.

This paradigm for visual tracking suggests a number of interesting questions and research directions. For example, it suggests that an extremely effective but flexible architecture for visual tracking may be hardware assisted warping, already available on many graphics workstations, combined with software-based tracking and state propagation methods. At a more abstract level, X Vision is strongly based on the notion of images as dynamical objects. Not surprisingly, the state computation and propagation cycle it uses is closely related to that found in estimation and feedback algorithms studied in systems theory. The use of systems design models and concepts to develop accurate prediction and estimation techniques for the X Vision framework is an area we are currently actively pursuing. Another related problem is that of partial and full occlusion, and target reacquisition. In recent work, we have begun to developed methods more robust to occlusion and distraction [21, 49], and techniques for automatic target localization and initialization [51].

X Vision is on ongoing software development project, versions of which are freely available. Information on the current version is can be found at

`http://www.cs.yale.edu/HTML/YALE/CS/AI/VisionRobotics/YaleAI.html` .

Acknowledgments This research was supported by 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.

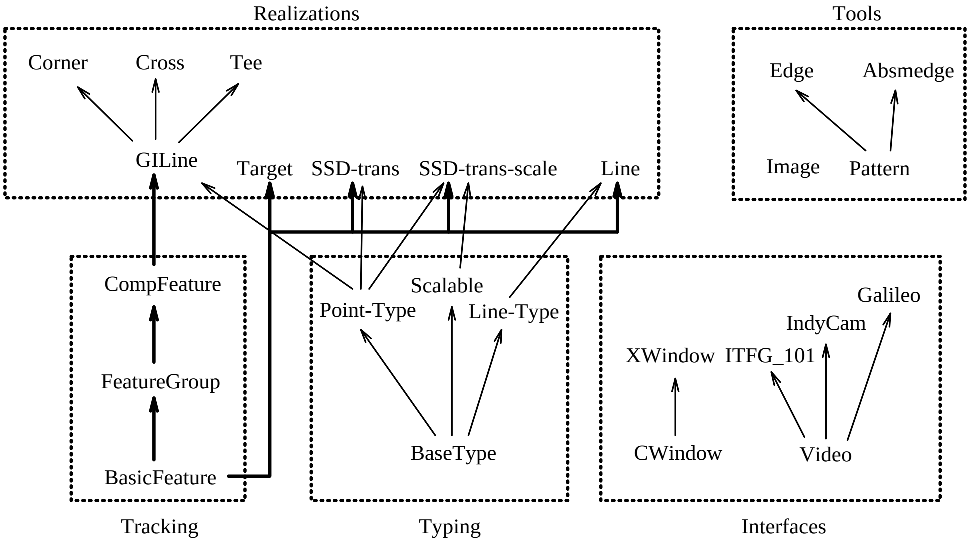

Figure 13. The organization of X Vision.

A Programming Environment

Just as graphics systems are amenable to object-oriented techniques, we have found that object-oriented programming is well-suited to tracking systems. The use of object-oriented methods allows us to hide the details of how specific methods are implemented and to interact with the system through a pre-specified set of generic interfaces. Modularization through object-oriented programming also makes it easy to conceptualize the distinct stages of multi-level tracking. The application of high-level constraints should not require an overhaul of low-level procedures, nor should low-level implementation affect the overall framework provided by higher level algorithms. Finally, such modularization also enhances the portability of the system.

We have constructed X Vision as a set of classes in C++. Briefly, all features are derived from a base class called `BasicFeature`. Basic features are directly derived from this class, and are characterized by their state vector, and functions which compute state information and display the feature graphically. There are two types of composite feature which are also derived from `BasicFeature`. `CompFeature` describes a composite feature which has both upward and downward links. `FeatureGroup` is a composite feature with only upward links—that is, it does not impose any constraints on its subsidiary features. Any feature may participate in only one `CompFeature`, but many `FeatureGroup`'s.

Both `CompFeature` and `FeatureGroup` maintain and manage an internal queue of their subsidiary features. Information is propagated up and down the feature network using two functions: `compute_state` which computes a composite feature's state from the state of its subsidiary nodes, and `state_propagate` that adjusts the state of a subsidiary nodes based on the state of their supersidiary node. The default update cycle for a `CompFeature` is to call its own `state_propagate` function, to call the update function of the children, and then to call `compute_state`. A `FeatureGroup` is similar, except there is no `state_propagate` function. The tracking cycle is combined into a single function `track()` callable only from a top-level feature. Calling it sweeps information down the network to the set of basic features, updates of the state of all basic features, and sweeps updated

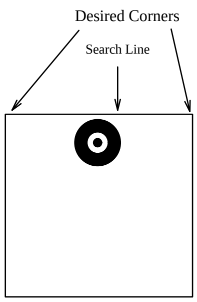

Figure 14. Schematic of the initialization example.

state information back up the network.

We have found that this programming environment greatly facilitates the development of tracking applications, and leads to clear compact program semantics. As an example, consider a simple program to locate and track the corners of the disk shown in Figure 14 using the fiducial marks located near one edge.

```
  Video v(1);                         // A video device
  Edge e;                             // The pattern to track
  Target t1(Sig1);                    // A specialized pattern finder
                                      // with a known signature
  Line l1(&e,&v),                     // Two line trackers
       l2(&e,&v);                     // tracking edges  in v

  Corner c1(&e, &v,UL),               // Two corner trackers operating in v
         c2(&e, &v,UR);

  if (!(t1.search()))  exit(1);                  // Search globally for the target
                                                 // This takes about 0.5 seconds

  l1.set_state(t1.x(), t1.y() + t1.sizey()/2, 0);    // Initialize two
  l2.set_state(t1.x(), t1.y() + t1.sizey()/2, M_PI); // search lines

  If (!(l1.search(c1) && l2.search(c2)))       // Search along the lines for
     exit(1);                                  // the corners

  CompFeature p;                               // A generic composite feature
  p += c1;  p += c2;                           // Add the corners to it.

  while (...) {                                // Go into a main tracking loop
   p.track();                                  // which combines tracking with
   ... other user code ...                     // other useful functions.
}
```

After picking a video device and edge pattern to track, the remainder of the code locates the specialized target, uses it as a basis to find the horizontal line, searches along the line to find the specified corners, and then goes into a basic tracking cycle to which other user code can be added. Naturally, any other user code impacts the speed of tracking and so must be limited to operations that can be performed in a small fraction of a second. In most of our applications, this is a feedback control computation, or a broadcast of information to another processor.

References

[1] P. Allen, A. Timcenko, B. Yoshimi, and P. Michelman. Automated tracking and grasping of a moving object with a robotic hand-eye system. *IEEE Trans. on Robotics and Automation*, **9**(2):152–165, 1993.

[2] A. A. Amini, S. Tehrani, and T. E. Weymouth. Using dynamic programming for minimizing the energy of active contours in the presence of hard constraints, in *Proc., 2nd Int. Conf. on Computer Vision*, pp. 95–99, 1988.

[3] P. Anandan. A computational framework and an algorithm for the measurement of structure from motion. *Int. J. Computer Vision*, **2**:283–310, 1989.

[4] R.L. Anderson. Dynamic sensing in a ping-pong playing robot. *IEEE Trans. on Robotics and Automation*, **5**(6):723–739, 1989.

[5] N. Ayache and O. D. Faugeras. Building, registrating, and fusing noisy visual maps. *Int. J. Robotics Research*, **7**(6):45–65, 1988.

[6] M. J. Black and Y. Yacoob. Tracking and recognizing rigid and non-rigid facial motions using local parametric models of image motions, in *Proc., Int. Conf. on Computer Vision*, Cambridge, MA, pp. 374–381, 1995.

[7] M.J. Black and A.D. Jepson. Eigentracking: Robust matching and tracking of articulated objects using a view-based representation, in *Proc., European Conf. on Computer Vision*, Cambridge, UK, 1996.

[8] A. Blake, R. Curwen, and A. Zisserman. Affile-invariant contour tracking with automatic control of spatiotemporal scale, in *Proc., Int. Conf. on Computer Vision*, pp. 421–430, Berlin, Germany, May 1993.

[9] Y. L. Chang and P. Liang. On recursive calibration of cameras for robot hand-eye systems, in *Proc., IEEE Int. Conf. on Robotics and Automation*, **2**, pp. 838–843, 1989.

[10] F. Chaumette, P. Rives, and B. Espiau. Positioning of a robot with respect to an object, tracking it, and estimating its velocity by visual servoing, in *Proc., IEEE Int. Conf. on Robotics and Automation*, pp. 2248–2253, Sacramento, CA, April 1991.

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

[12] J. L. Crowley, P. Stelmaszyk, T. Skordas, and P. Puget. Measurement and integration of 3-D structures by tracking edge lines. *Int. J. Computer Vision*, **8**(1):29–52, 1992.

[13] M. W. Eklund, G. Ravichandran, M. M. Trivedi, and S. B. Marapane. Adaptive visual tracking algorithm and real-time implemenation, in *Proc., IEEE Int. Conf. on Robotics and Automation*, pp. 2657–2662, Nagoya, Japan, May 1995.

[14] O. D. Faugeras, F. Lustaman, and G. Toscani. Motion and structure from point and line matches, in *Proc., Int. Conf. on Computer Vision*, pp. 25–33, June 1987.

[15] J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes. *Computer Graphics.* Addison Wesley, 1993.

[16] W. T. Freeman and E. H. Adelson. Steerable filters for early vision, image analysis, and wavelet decomposition, in *Proc., Int. Conf. on Computer Vision*, Osaka, Japan, 1990.

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

[18] G. D. Hager. Calibration-free visual control using projective invariance, in *Int. Conf. on Computer Vision*, Cambridge, MA, pp. 1009-1015, 1995.

[19] G. D. Hager. Real-time feature tracking and projective invariance as a basis for hand-eye coordination, in *Computer Vision and Pattern Recognition*, pp. 533–539. IEEE Computer Society Press, 1994.

[20] G. D. Hager. A modular system for robust hand-eye coordination. DCS RR-1074, Yale University, New Haven, CT, June 1995.

[21] G. D. Hager and P.N. Belhumeur. Real-time tracking of image regions with changes in geometry and illumination, in *Computer Vision and Pattern Recognition*, pp. 403–410. IEEE Computer Society Press, 1996.

[22] G. D. Hager, W-C. Chang, and A. S. Morse. Robot hand-eye coordination based on stereo vision. *IEEE Control Systems Magazine*, **15**(1):30–39, February 1995.

[23] G. D. Hager, G. Grunwald, and K. Toyama. Feature-based visual servoing and its application to telerobotics, in *Intelligent Robots and Systems* (V. Graefe, Ed.), Elsevier, 1995.

[24] R. M. Haralick and L. G. Shapiro. *Computer and Robot Vision: Volume II.* Addison Wesley, 1993.

[25] T. Heikkila, Matsushita, and Sato. Planning of visual feedback with robot-sensor co-operation, in *Proc., IEEE Int. Workshop on Intelligent Robots and Systems*, Tokyo, 1988.

[26] E. Huber and D. Kortenkamp. Using stereo vision to pursue moving agents with a mobile robot, in *Proc., IEEE Int. Conf. on Robotics and Automation*, pp. 2340–2346, Nagoya, Japan, May 1995.

[27] I. Inoue. Hand eye coordination in rope handling, in *1st Int. Symp. on Robotics Research*, Bretten Woods, USA, 1983.

[28] A. Izaguirre, P. Pu, and J. Summers. A new development in camera calibration: Calibrating a pair of mobile cameras. *Int. J. Robot. Res.*, **6**(3):104–116, 1987.

[29] H. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *Int. J. Computer Vision*, **1**:321–331, 1987.

[30] Y. Liu and T. S. Huang. A linear algorithm for motion estimation using straight line correspondences, in *Int. Conf. on Pattern Recognition*, pp. 213–219, 1988.

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

[32] C.-P. Lu. *Online Pose Estimation and Model Matching.* PhD thesis, Yale University, 1995.

[33] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. pp. 674–679, 1981.

[34] A. D. Morgan, E. L. Dagless, D. J. Milford, and B. T. Thomas. Road edge tracking for robot road following: a real-time implementation. *Image and Vision Computing*, **8**(3):233–240, August 1990.

[35] J. Mundy and A. Zisserman. *Geometric Invariance in Computer Vision.* MIT Press, Cambridge, MA, 1992.

[36] H. K. Nishihara. Teleos AVP, in *CVPR Demo Program*, San Francisco, 1996.

[37] N. Papanikolopoulos, P. Khosla, and T. Kanade. Visual tracking of a moving target by a camera mounted on a robot: A combination of control and vision, in Proc., IEEE Int. Conf. on Robotics and Automation, **9**(1), 1993.

[38] D. Reynard, A. Wildenberg, A. Blake, and J. Marchant. Learning dynamics of complex motions from image sequences. pp. I:357–368, 1996.

[39] A. A. Rizzi and D. E. Koditschek. An active visual estimator for dexterous manipulation. Paper presented at the 1994 Workshop on Visual Servoing, 1994.

[40] J. Shi and C. Tomasi. Good features to track, in *Computer Vision and Pattern Recognition*, pp. 593–600. IEEE Computer Society Press, 1994.

[41] D. Sinclair, A. Blake, S. Smith, and C. Rothwell. Planar region detection and motion recovery. *Image and Vision Computing*, **11**(4):229–234, May 1993.

[42] M. E. Spetsakis. A linear algorithm for point and line-based structure from motion. *CVGIP: Image Understanding*, **56**(2):230–241, September 1992.

[43] M. E. Spetsakis and J. Aloimonos. Structure from motion using line correspondences. *Int. J. Computer Vision*, **4**:171–183, 1990.

[44] T. N. Tan, K. D. Baker, and G. D. Sullivan. 3D structure and motion estimation from 2D image sequences. *Image and Vision Computing*, **11**(4):203–210, May 1993.

[45] Camillo J. Taylor and D. J. Kriegman. Structure and motion from line segments in multiple images, in *Proc., IEEE Int. Conf. on Robotics and Automation*, pp. 1615–1620, 1992.

[46] D. Terzopoulos and R. Szeliski. Tracking with Kalman snakes, in *Active Vision* (A. Blake and A. Yuille, Ed.), MIT Press, Cambridge, MA, 1992.

[47] C. Tomasi and T. Kanade. Shape and motion from image streams: a factorization method, full report on the orthographic case. CMU-CS 92-104, CMU, 1992.

[48] C. Tomasi and T. Kanade. Shape and motion without depth, in *Proc., Image Understanding Workshop*, pp. 258–271, September 1990.

[49] K. Toyama and G. D. Hager. Keeping your eye on the ball: Tracking occluding contours of unfamiliar objects without distraction, in *Proc., Int. Conf. on Intel. Robotics and Sys.*, pp. 354–359, Pittsburgh, PA, August 1995.

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

[51] K. Toyama and G. D. Hager. Incremental focus of attention for robust visual tracking, in *Computer Vision and Pattern Recognition*, pp. 189–195. IEEE Computer Society Press, 1996.

[52] K. Toyama and G. D. Hager. Vision-based 3D "Surfball" input device, 1996. Provisional patent application filed through Yale Office of Cooperative Research, OCR 770.

[53] K. Toyama, J. Wang, and G. D. Hager. SERVOMATIC: a modular system for robust positioning using stereo visual servoing, in *Proc., Int. Conf. Robotics and Automation*, pp. 2636–2643, 1996.

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

[55] Y. Yagi, K. Sato, and M. Yachida. Evaluating effectivity of map generation by tracking vertical edges in omnidirectional image sequence, in *Proc., IEEE Int. Conf. on Robotics and Automation*, pp. 2334–2339, Nagoya, Japan, May 1995.

[56] B. H. Yoshimi and P. K. Allen. Active, uncalibrated visual servoing, in *Proc., IEEE Int. Conf. on Robotics and Automation*, **4**, pp. 156–161, San Diego, CA, May 1994.

[57] Z. Zhang and O. Faugeras. Determining motion from 3D line segment matches: a comparative study. *Image and Vision Computing*, **9**(1):10–19, February 1991.