International Journal of Computer Vision, 7:3, 243–270 (1992)

Visual Tracking of Known Three-Dimensional Objects

DONALD B. GENNERY
Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA 91109

Received June 13, 1991. Revised August 15, 1991.

Abstract
A method is described of visually tracking a known three-dimensional object as it moves with six degrees of freedom. The method uses the predicted position of known features on the object to find the features in images from one or more cameras, measures the position of the features in the images, and uses these measurements to update the estimates of position, orientation, linear velocity, and angular velocity of the object model. The features usually used are brightness edges that correspond to markings or the edges of solid objects, although point features can be used. The solution for object position and orientation is a weighted least-squares adjustment that includes filtering over time, which reduces the effects of errors, allows extrapolation over times of missing data, and allows the use of stereo information from multiple-camera images that are not coincident in time. The filtering action is derived so as to be optimum if the acceleration is random. (Alternatively, random torque can be assumed for rotation.) The filter is equivalent to a Kalman filter, but for efficiency it is formulated differently in order to take advantage of the dimensionality of the observations and the state vector which occur in this problem. The method can track accurately with arbitrarily large angular velocities, as long as the angular acceleration (or torque) is small. Results are presented showing the successful tracking of partially obscured objects with clutter.

1 Introduction

A robot for performing assembly work may often have to grasp a moving rigid object. Such tasks may occur in future activities in space, both for assembly of structures in orbit and for retrieval of satellites. It will be necessary to determine the pose (position and orientation) and velocities (linear velocity and angular velocity) of the object accurately as it moves in three-dimensional space, so that it can be grasped in a proper manner. In many such tasks, the object being handled is known, and an accurate model of it can be prepared beforehand. This model enables known features on the object to be searched for in images of the scene, and thus facilitates tracking the object and grasping it using grasp points that are built into the model.

The tracking task can be conveniently divided into two portions: acquisition and tracking proper. In the acquisition portion, which is similar in some respects to recognition, the object must be located in the scene, and its approximate pose and velocities must be determined. Then this information can be used to initiate the tracking-proper phase, in which the object pose and velocities are refined for greater accuracy and are rapidly updated, by robust techniques especially suited to this task. We have done some work on the acquisition problem, and a preliminary report has been made elsewhere [Gennery 1986].

This article deals only with the tracking-proper task, and is the third in a series of reports that deal with this problem. (For the examples herein, acquisition was performed manually with the aid of a visual display, starting from an approximately known position and orientation when the object is stationary or moving slowly.)

Originally, a fairly simple method was developed [Saund et al. 1981] that compared the measured positions of features in images to their predicted positions in order to update the state of the object. It used only a simple object; it used only the corners of the object as features in the adjustment; it represented orienta-

tion by means of Euler angles, which limited the rotation because of the singularities contained in that representation; and it filtered, independently in each coordinate, the results of separate least-squares adjustments for each time.

Then a much more elaborate method was developed and briefly described [Gennery 1982]. That method, which is subsumed by the method of this article, also compared the measured and predicted feature positions in order to update the object state. However, it used polyhedral object models; looked for edge elements all along the predicted object edges; represented orientation by means of quaternions in order to avoid singularities; used infinitesimal rotation vectors to adjust orientation, in order to simplify the computations; combined filtering optimally in the adjustment, enabling stereo depth information to be extracted when more than one camera is used, even if the different cameras take their pictures at different times; used an alternative to the usual Kalman fitler, saving computation time in the type of situaton encountered in tracking complicated rigid objects; and used an accurate error propagation through rotation, allowing optimal filtering even when the object rotates by an arbitrarily large amount between successive frames.

Since then, several improvements to that tracking method have been implemented. This article is an updated, more detailed description of the method, including those improvements and others that have been devised for possible future use. The improvements include the following: a more general object model including reflectivity of faces, which allows coplanar faces, different weights for different edges according to their expected contrast, and the use of edge polarity; a correction for raster scan delay; the computation of performance indicators; the use of the moment of inertia tensor so that prediction can be based on angular momentum instead of angular velocity, if appropriate; more robust detection of features, which uses variable weights depending on the measured properties of the detected features; and the inclusion of lens distortion. Also, better examples of the performance of the method are presented.

The object models now used consist of planar surfaces, which can be specified for any particular polyhedral object with possible polygonal markings. The features searched for in the images usually are the edges formed by the intersections of the planar faces or by the boundaries between coplanar surfaces of differing reflectivity. Alternatively, point features, corresponding to lights on the object, can be used.

Before 1981, the work that had been done on the visual tracking of objects did not have much in common with the work described here. It mostly had been two-dimensional tracking [Nagel 1978; Martin & Aggarwal 1978; Gilbert et al. 1980] or had dealt with restricted domains in which only partial spatial information is extracted [Roach & Aggarwal 1979]. Here we are concerned with determining the three-dimensional position and orientation of a solid object as it moves in an arbitrary way. Also, some work had dealt with objects that are labeled with obvious features that unambiguously determine the desired information [Pinkney 1978], whereas here the features can be those occurring naturally on the object, such as differences in illumination across boundaries of planar faces. Such features often can be missed because of the conditions of illumination, and extaneous features may be detected. Therefore, the algorithm must be able to handle such imperfections.

Since our earlier tracker report [Gennery 1982], some other work has appeared that is similar in some ways, but that work does not contain all of the features described in the 1982 report. Broida and Chellappa [1986] deal with only two dimensions and use only (simulated) points (not edge features). Young and Chellappa [1990] use only points, and they assume that the three-dimensional position of each point has been previously obtained independently (instead of having the tracking program find the features itself in the two-dimensional images). Verghese and Dyer [1988] do not use a filter with velocities for prediction, and thus more searching would be required to find the features if large velocities were to occur. Dickmanns and Graefe [1988a,b] deal with several different problems with different types of motion, none of which has the full three-dimensional rotation of the problem here. Wünsche [1986] considers rotation only in a plane, but includes a way of selecting those features that contribute the most to the solution. Wu et al. [1989] use only points; and they represent rotation by means of roll, pitch, and yaw (similar to Euler angles), which contain singularities, instead of quaternions. Furthermore, apparently none of these other methods use the alternative to the usual Kalman filter, infinitesimal rotation vectors, or the accurate error propagation through three-dimensional rotation. (Also, these other methods do not include most of the improvements described here.)

The *tracker* described herein can use any number of cameras. If only one camera is used, the distance to the object will be obtained as part of the solution because of the known size of the object. If more than

one camera is used, stereo triangulation will produce more accurate distance information, especially if the angle between the cameras as seen from the object is large. (This wide-angle stereo condition is feasible because correlation between pictures is not done; instead, each feature is searched for at its expected position in each picture independently.) Triangulation to individual features is not done. Instead, the information from each camera is entered independently into an overall solution. Even if some features are seen by one camera and different features are seen by another camera, they still produce stereo depth information (not only for object distance but also for relative distance to different portions of the object to aid in the determination of orientation), because of the known spatial relationships between features in the object model. In fact, the different pictures do not have to be coincident in time for this stereo information to be useful. The constraints of the filtering included in the adjustment allow the information obtained at different times to be combined optimally. The *tracker* of the earlier report [Gennery 1982] included this noncoincident stereo ability, although it was artificially limited to one or two cameras. This ability is also inherent in the methods of some of the other authors [Dickmanns & Graefe 1988a,b; Wu et al. 1989], although they did not utilize it, since they used only one camera.

The mathematics used here involve least-squares adjustments, covariance matrixes, error propagation, and matrix algebra. A good text on these subjects is provided by Mikhail [1976]. Quaternions also are used; their relevant properties are described in appendix A.

In order to aid in keeping track of the different types of mathematical entities, the following system of symbols will be used: scalars will be denoted by lower-case letters; quaternions will be denoted by capital letters; physical vectors in three-dimensional space will be denoted by boldfaced lower-case letters, and they will be considered to be equivalent to 3-by-1 matrixes; and other matrixes (including 6-vectors) will be denoted by boldfaced capital letters. Where needed, a quaternion will be expressed in terms of its scalar part s and its vector part $\mathbf{v}$ by the notation $(s, \mathbf{v})$. The transpose of a matrix $\mathbf{A}$ will be denoted by $\mathbf{A}^T$ and its inverse by $\mathbf{A}^{-1}$. The vector product (cross-product) of two vectors $\mathbf{a}$ and $\mathbf{b}$ will be denoted by $\mathbf{a} \times \mathbf{b}$, and their scalar product (dot product, or inner product) will be denoted by $\mathbf{a} \cdot \mathbf{b}$ (equivalent to $\mathbf{a}^T\mathbf{b}$).

2 Representation of Orientation

Ideally, one would like to represent the orientation of the object by means of a set of parameters having the following properties: the number of parameters is three, since there are only three degrees of freedom to a rotation in three-dimensional space; the representation contains no singularities, so that the partial derivatives of the parameters with respect to any small rotation angle are always finite; and the parameters are continuous (that is, a continuous motion of the object never produces a discontinuity in the parameters). (In two-dimensional space, where there is only one rotational degree of freedom, the rotation angle meets these criteria, although it is multivalued.) Unfortunately, such a set of parameters for three-dimensional space does not exist.

The Euler angles are often used to represent orientation, but they contain a singularity. For example, if the Euler angles are defined such that the first and third rotations are about the same (rotated) axis [Goldstein 1980], a zero value for the second angle causes the first and third angles to become indeterminate. An object motion which passes arbitrarily close to this condition can produce arbitrarily large derivatives of these angles. This is usually not a problem in analytical studies, but in numerical adjustments it can cause the solution to fail.

Since by Euler's theorem any rotation in three-dimensional space can be considered to be a rotation about a single fixed axis, one possibility is to represent the orientation by means of a "vector" whose direction is this axis of rotation and whose magnitude is the angle of rotation. However, if the angle is a multiple of 2π radians, the direction of the vector is indeterminate, and thus so are its components (except when the angle is zero, when the components are zero also). Thus singularities are still present, unless the angle is restricted to the range from $-\pi$ to π, in which case a discontinuity has been introduced. In addition, this entity is not a physical vector (see Goldstein [1980]). The practical consequences of this latter fact seem to be that the partial derivatives of interesting quantities with respect to the three components are complicated to compute analytically (see Ayache & Faugeras [1988]). However, if the magnitude of the rotation is infinitesimal, this representation actually is a physical vector, as explained by Goldstein [1980], and partial

derivatives with respect to this vector are extremely simple, as we shall see in section 5. Of course, an infinitesimal vector cannot represent the current orientation, which may require a large rotation from the reference orientation, but it can be used for small corrections.

If the requirement that the number of parameters be three is removed, there are many possibilities that meet the other criteria. One of these is the rotation matrix, which has nine components. It is especially convenient for rotating vectors and for the associated error propagation. Another possibility is a quaternion with unit norm, which has four components. Not only are there fewer parameters to deal with, but the normalization problem is easier when a quaternion rather than a matrix is used as the primary representation of orientation. Through a long sequence of corrections applied directly to a matrix or quaternion, numerical errors will cause the matrix to depart from orthonormality or the quaternion to depart from unity norm. A quaternion is normalized simply by dividing by the square root of the sum of the squares of its components. It is also very easy to update a quaternion when extrapolating to a predicted orientation based on an estimated angular velocity, as will be shown in section 4.

Because of the above conflicting requirements, the implemented tracker uses a combination of representations for orientation. A unit quaternion is used to represent the current estimate of object orientation. However, the parameters used in the adjustment are the three components of an infinitesimal rotation vector which represents the correction needed to the current estimate of orientation. Of course, the actual correction needed in the adjustment is finite. However, the adjustment is based on a linearization of the actual problem anyway (which is accurate as long as the needed corrections are small), and assuming that the orientation corrections are infinitesimal is equivalent to assuming that their effects on the measurable quantities are linear. The fact that only three rotation parameters are used in the adjustment keeps the computation time small and avoids the complication of adding a constraint to the adjustment because of an excess of parameters over the true degrees of freedom. The solution for the infinitesimal correction rotation vector is used to correct the quaternion (by a simple process described in section 7), and an extrapolation to a new time is done by a quaternion product (as described in section 4), so that a new orientation estimate expressed in a fixed coordinate system is produced. (The correction in the adjustment must be small for linearity, but the extrapolation can be arbitrarily large.) A rotation matrix computed from the rotation (unit) quaternion is used for rotating vectors, for efficiency.

3 Overview of Tracker

Within the tracking program, the current estimate of the state of the object consists of the vector **p**, which represents the position of the origin of the object-fixed coordinate system; the quaternion R, which represents the orientation of the object; the vector **v**, which represents the linear velocity of the object (derivative of **p** with respect to time for the true, unknown values); and the vector ω, which represents the angular velocity of the object. (If the moment of inertia tensor of the object were used in the extrapolation of orientation, angular momentum λ would be used instead of ω.) All of these are expressed in a fixed-reference coordinate system. The uncertainty in these quantities is represented by the 12×12 covariance matrix **S**, in which the fourth, fifth, and sixth rows and columns refer to an infinitesimal rotation correction vector ϕ instead of to R, as explained in section 2. The **S** matrix is partitioned as needed into 6×6 covariance matrixes and 3×3 covariance matrixes as follows:

$$\mathbf{S}=\begin{bmatrix}\mathbf{S_{PP}} & \mathbf{S_{PV}}\\ \mathbf{S_{PV}^T} & \mathbf{S_{VV}}\end{bmatrix}=\begin{bmatrix}\mathbf{S_{pp}} & \mathbf{S_{p\phi}} & \mathbf{S_{pv}} & \mathbf{S_{p\omega}}\\ \mathbf{S_{p\phi}^T} & \mathbf{S_{\phi\phi}} & \mathbf{S_{\phi v}} & \mathbf{S_{\phi\omega}}\\ \mathbf{S_{pv}^T} & \mathbf{S_{\phi v}^T} & \mathbf{S_{vv}} & \mathbf{S_{v\omega}}\\ \mathbf{S_{p\omega}^T} & \mathbf{S_{\phi\omega}^T} & \mathbf{S_{v\omega}^T} & \mathbf{S_{\omega\omega}}\end{bmatrix} \quad (1)$$

where the lower-case subscripts refer to the vectors described above, **P** denotes the pose (position and orientation), and **V** denotes the linear and angular velocities. With a circumflex ($\hat{}$) over it, any of these symbols refers to predicted values rather than adjusted values. (For a covariance matrix, logically the circumflexes should be on the subscripts instead of on the **S**, since it represents the covariance matrix of predicted values and not a predicted covariance matrix of adjusted values. However, for simplicity, the circumflex is put over the **S**.)

The tracker operates in a loop with the following major steps: Prediction, Projection, Measurement, and Adjustment, as shown in figure 1. These steps are described briefly in the following paragraphs and in detail in sections 4–7.

Prediction receives the values of **p**, R, **v**, ω, and **S** (and the time for which these are valid) produced by a previous adjustment when tracking or available

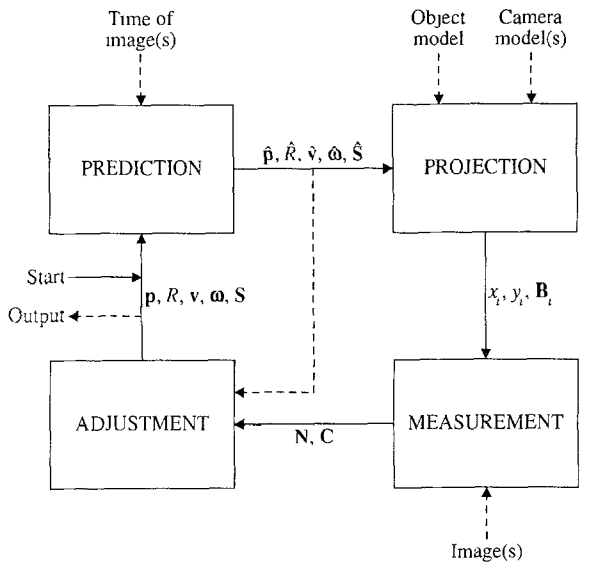


Fig. 1. Block diagram of the tracker. Solid arrows denote program flow and data; dashed arrows denote data only. Mathematical symbols are defined in the text.

as initial conditions when starting. The time at which a new image is obtained is noted, and the object state is extrapolated to this time to produce the predicted values $\hat{\mathbf{p}}$, $\hat{R}$, $\hat{\mathbf{v}}$, $\hat{\boldsymbol{\omega}}$, and their uncertainties as represented by the covariance matrix $\hat{\mathbf{S}}$. (It is not necessary for the new image to be available yet; only its time is needed at this point. The image is used only in Measurement. However, because of the hardware, the current tracker freezes the image in an image buffer, notes the time, and then performs Prediction, Projection, and Measurement while the image is waiting.)

Projection uses this predicted information, the given object model, and the given camera model to compute the visibility of the object vertexes and edges, and, for vertexes on edges predicted to be visible, to transform the vertex positions from object coordinates to fixed coordinates. These predicted vertex (or other point-feature) positions are then projected into the image plane to produce their image coordinates x_i and y_i and the partial derivative matrices $\mathbf{B}_i$ of these image coordinates relative to the object pose.

Measurement uses the projected vertex information and the edge-vertex connectivity information in the object model to search the new image for brightness edges near the predicted positions of object edges. (The vertexes themselves or other point features could be used here instead of or in addition to the edges. The implemented version currently provides a choice between edges or lights.) The discrepancies between the predicted and measured positions, along with the partial derivatives, are used to compute the 6×6 matrix $\mathbf{N}$ and the 6×1 matrix $\mathbf{C}$, which consist respectively of the coefficients and the "constants" in the (partially reduced) normal equations [Mikhail 1976] which would produce a linearized least-squares adjustment of object pose $\mathbf{P}$ based on information obtained at this time only. (The corrections would be $\mathbf{N}^{-1}\mathbf{C}$, with covariance matrix $\mathbf{N}^{-1}$. However, in general $\mathbf{N}$ might be singular, so that there would not be a solution for this time only; but the actual computations in Adjustment use whatever information $\mathbf{N}$ and $\mathbf{C}$ contain, combined with the information from other times in the filtering action.)

Adjustment combines the information from Measurement and Prediction to produce new values of $\mathbf{p}$, R, $\mathbf{v}$, $\boldsymbol{\omega}$, and $\mathbf{S}$ valid for the time of the image just used in Measurement. Giving the predicted values appropriate weight in this adjustment (according to their covariance matrix) produces the filtering action. The new adjusted values are used as output and as input to Prediction to repeat the process.

If more than one camera is used, the pictures from the different cameras could be taken simultaneously. In this case, each time through the above loop, Projection would project the predicted data into all of the image planes and Measurement would process each image and collect the results into one $\mathbf{N}$ matrix and one $\mathbf{C}$ matrix. These matrixes would contain the stereo information resulting from the use of multiple cameras. However, if the different cameras take their pictures at different times, then on different times through the loop different cameras would be used, with the appropriate camera model being used in Projection and the appropriate image being used in Measurement each time. In this case, the $\mathbf{N}$ and $\mathbf{C}$ matrixes on each time through the loop would contain information from only one camera. However, because of the memory of old information caused by the inclusion of the predicted data in the adjustment, stereo depth information would still be produced. (This fact is discussed further in section 9).

4 Prediction

4.1 Extrapolation Based on Angular Velocity

When a new picture is taken during tracking, the predicted object pose for the time of the picture must be computed from the previous data (from the previous

adjustment or from input initialization data). In the current tracker, this extrapolation (over a time interval τ) is based on the assumption that the linear acceleration and angular acceleration are random (that is, they consist of white noise). (This random signal driving the filter is often called "excitation" or "plant noise.") Specifically, they are assumed to have constant power spectra denoted by a and α, respectively, in each dimension. These denote mean squared value per unit frequency band, considering both positive and negative frequencies. (If only positive frequencies are considered, the power spectra are $2a$ and 2α.) This assumption cannot be literally true, since a constant power spectrum implies an infinite variance. However, all that is required for reasonable accuracy is that the power spectrum be constant to a frequency considerably higher than the rate at which the pictures are taken. An assumption more closely aligned with reality could be devised for most actual situations, but it would be quite different for different situations. (For example, if there tended to be extended periods of fairly high acceleration, it would be better to assume random third derivatives of pose with respect to time, instead of second derivatives. The accelerations would then have to be included in the object state, and **S** would be an 18×18 matrix instead of 12×12.) The assumption used here at least leads to simple results. (The filtering action that it produces is discussed in section 9).

Since acceleration is random, its expected value at all times is zero. Thus the extrapolation of pose itself (ignoring for a moment its accuracy) uses the previous estimates of the velocities as constants over the time τ elapsed since the previous picture. Therefore, the predicted values of position, velocity, and angular velocity are obtained from the old adjusted values as follows:

$$\hat{\mathbf{p}} = \mathbf{p} + \tau\mathbf{v} \tag{2}$$

$$\hat{\mathbf{v}} = \mathbf{v} \tag{3}$$

$$\hat{\omega} = \omega \tag{4}$$

The orientation is extrapolated by first computing a quaternion H corresponding to the rotation during the elapsed time τ as follows (from equation (A3)):

$$H = \left(\cos\frac{\omega\tau}{2},\ \mathbf{u}\sin\frac{\omega\tau}{2}\right) \tag{5}$$

where ω is the magnitude of the vector ω, and $\mathbf{u}$ is the unit vector in the direction of ω. Then the predicted orientation is obtained by the following quaternion product:

$$\hat{R} = HR \tag{6}$$

Now we consider the accuracy of the above predicted quantities, as represented by their covariance matrix $\hat{\mathbf{S}}$. This can be extrapolated from **S** by considering three effects: the effect on pose of the uncertainty in the velocities used to do the extrapolation in (2)–(6), the effect of the rotation (represented by H) that occurs during the extrapolation interval, and the effect of the random acceleration that occurs during the extrapolation interval. The first two of these effects can be summarized by a 12×12 transition matrix **J** that shows how the extrapolated values of the object state (pose and velocities) depend on the previous state. (By the usual rule of covariance propagation, the covariance matrix resulting from this transition then would be $\mathbf{JSJ}^{\mathrm{T}}$.) **J** is as follows:

$$\mathbf{J} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \tau\mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{H} & \mathbf{0} & \tau\mathbf{G} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \tag{7}$$

where **I** is the identity matrix (here 3×3), **0** is the zero matrix (here 3×3), **H** is the rotation matrix corresponding to the quaternion H (computed according to equation (A5) or (A6)), and **G** will be defined below. The off-diagonal terms in this expression for **J** take care of the first effect mentioned above, and the presence of the **H** and **G** matrixes instead of identity matrixes takes care of the second effect. In the case of **H**, this simply rotates any orientation error vector into the new orientation. However, the effect of an error in the angular-velocity vector on the new orientation, as given by $\tau\mathbf{G}$, is more complicated and will now be discussed.

An error in angular velocity at any time during the extrapolation interval τ produces an error in orientation at that time, which is then rotated (assuming it is an infinitesimal orientation error) by the amount of rotation occurring in the remainder of the extrapolation interval, in order to produce the effect on the final orientation. Therefore, the total effect on the final orientation of a constant angular velocity error over the entire interval is produced by the matrix product of **G** and the angular velocity error vector, times τ, where **G** is the average value (over the extrapolation interval) of the rotation matrix that represents the rotation over a portion of the interval. That is,

$$\mathbf{G} = \frac{1}{\tau}\int_0^{\tau} \mathbf{H}(\tau),\, dt \tag{8}$$

where $\mathbf{H}(\tau)$ represents the value of $\mathbf{H}$ computed according to equation (5) for any particular value of τ, and the integration is over the actual extrapolation interval τ. Since the extrapolation process assumes that ω is constant over this interval, the integral in (8) can be evaluated in a straightforward manner (by using (A6)) to produce the following:

$$\mathbf{G} = \begin{bmatrix} (1-c)u_1^2 + c & (1-c)u_1u_2 - su_3 & (1-c)u_1u_3 + su_2 \\ (1-c)u_1u_2 + su_3 & (1-c)u_2^2 + c & (1-c)u_2u_3 - su_1 \\ (1-c)u_1u_3 - su_2 & (1-c)u_2u_3 + su_1 & (1-c)u_3^2 + c \end{bmatrix} \tag{9}$$

where

$$s = \frac{2}{\omega\tau}\sin^2\frac{\omega\tau}{2}$$

$$c = \frac{2}{\omega\tau}\sin\frac{\omega\tau}{2}\cos\frac{\omega\tau}{2}$$

and where u_1, u_2, and u_3 are the components of $\mathbf{u}$ (the unit vector in the direction of ω). Note that by taking limits, when $\omega\tau = 0$, $\mathbf{G} = \mathbf{I}$.

Now the third effect mentioned above (the effect of the random acceleration that occurs during the extrapolation interval) will be discussed. At first, consider only scalar position x, velocity $\dot{x}$, and acceleration $\ddot{x}$, where x and $\dot{x}$ represent only the changes from the beginning of the extrapolation interval, for an arbitrary variable x. Thus the values at the end of the extrapolation interval τ are

$$\dot{x} = \int_0^{\tau} \ddot{x}\, dt \tag{10}$$

$$\begin{aligned} x &= \int_0^{\tau} \dot{x}\, dt \\ &= \dot{x}\tau - \int_0^{\tau} t\ddot{x}\, dt \\ &= \tau\int_0^{\tau} \ddot{x}\, dt - \int_0^{\tau} t\ddot{x}\, dt \\ &= \int_0^{\tau} (\tau - t)\ddot{x}\, dt \end{aligned} \tag{11}$$

From the above two equations the covariance matrix of position and velocity can be derived as the expected products of these values, since their expected values are zero. In this process we use the fact that the expected squared value of $\ddot{x}\, dt$ is $a\, dt$ (because the expected squared value of the mean of $\ddot{x}$ over any interval Δt is $a/\Delta t$, since a is assumed to be the constant power spectral density of $\ddot{x}$). Thus,

$$\begin{aligned} \begin{bmatrix} \sigma_x^2 & \sigma_{x\dot{x}} \\ \sigma_{x\dot{x}} & \sigma_{\dot{x}}^2 \end{bmatrix} &= \int_0^{\tau} \begin{bmatrix} (\tau - t)^2 & (\tau - t) \\ (\tau - t) & 1 \end{bmatrix} a\, dt \\ &= \begin{bmatrix} 1/3a\tau^3 & 1/2a\tau^2 \\ 1/2a\tau^2 & a\tau \end{bmatrix} \end{aligned} \tag{12}$$

Generalizing this to three dimensions means using a 3×3 matrix instead of a. However, in the present tracker it is assumed that the different components of the random acceleration vector are uncorrelated and tend to be equally large. Therefore, the above components of the covariance matrix are just multiplied by the identity matrix to produce the 3×3 terms to be added to the covariance matrix of $\mathbf{p}$ and $\mathbf{v}$. Similarly, the same is done for orientation ϕ and angular velocity ω, except that α (the power spectrum of angular acceleration) is used instead of a. Strictly speaking, the effect of the rotation during the interval τ should be included in the orientation and orientation-angular-velocity correlation. However, the main additive terms driving the filtering action are the terms affecting velocities, which then affect pose through the off-diagonal terms of $\mathbf{J}$ on subsequent iterations, and the other additive terms usually have only a minor effect. Also, usually the rotation during the interval is small, and thus its effect through these terms is doubly small. Therefore, this effect is neglected for simplicity. (Such other additive terms are usually omitted altogether in Kalman filter applications. The effect of leaving them out is discussed elsewhere [Gennery 1990].)

Therefore, combining all three effects on the covariance matrix of the predicted quantities produces the following result:

$$\hat{\mathbf{S}} = \mathbf{J}\mathbf{S}\mathbf{J}^{\mathsf{T}} + \begin{bmatrix} \frac{1}{3}a\tau^3\mathbf{I} & \mathbf{0} & \frac{1}{2}a\tau^2\mathbf{I} & \mathbf{0} \\ \mathbf{0} & \frac{1}{3}\alpha\tau^3\mathbf{I} & \mathbf{0} & \frac{1}{2}\alpha\tau^2\mathbf{I} \\ \frac{1}{2}a\tau^2\mathbf{I} & \mathbf{0} & a\tau\mathbf{I} & \mathbf{0} \\ \mathbf{0} & \frac{1}{2}\alpha\tau^2\mathbf{I} & \mathbf{0} & \alpha\tau\mathbf{I} \end{bmatrix} \tag{13}$$

However, because $\mathbf{J}$ is so sparse, the above matrix product is expanded in terms of 3×3 matrixes, according to the definitions of $\mathbf{S}$ in (1) and $\mathbf{J}$ in (7), to produce the following:

$$\hat{\mathbf{S}}_{\mathbf{pp}} = \mathbf{S}_{\mathbf{pp}} + \tau(\mathbf{S}_{\mathbf{pv}} + \mathbf{S}_{\mathbf{pv}}^{\mathsf{T}}) + \tau^2\mathbf{S}_{\mathbf{vv}} + \tfrac{1}{3}a\tau^3\mathbf{I} \tag{14}$$

$$\hat{\mathbf{S}}_{\mathbf{p}\phi} = \mathbf{S}_{\mathbf{p}\phi}\mathbf{H}^{\mathsf{T}} + \tau\mathbf{S}_{\mathbf{p}\omega}\mathbf{G}^{\mathsf{T}} + \tau(\mathbf{H}\mathbf{S}_{\phi\mathbf{v}})^{\mathsf{T}} + \tau^2\mathbf{S}_{\mathbf{v}\omega}\mathbf{G}^{\mathsf{T}} \tag{15}$$

$$\hat{\mathbf{S}}_{\phi\phi} = \mathbf{H}\mathbf{S}_{\phi\phi}\mathbf{H}^T + \tau[\mathbf{H}\mathbf{S}_{\phi\omega}\mathbf{G}^T + (\mathbf{H}\mathbf{S}_{\phi\omega}\mathbf{G}^T)^T] + \tau^2\mathbf{G}\mathbf{S}_{\omega\omega}\mathbf{G}^T + \frac{1}{3}\alpha\tau^3\mathbf{I} \quad (16)$$

$$\hat{\mathbf{S}}_{pv} = \mathbf{S}_{pv} + \tau\mathbf{S}_{vv} + \frac{1}{2}a\tau^2\mathbf{I} \quad (17)$$

$$\hat{\mathbf{S}}_{\phi v} = \mathbf{H}\mathbf{S}_{\phi v} + \tau(\mathbf{S}_{v\omega}\mathbf{G}^T)^T \quad (18)$$

$$\hat{\mathbf{S}}_{vv} = \mathbf{S}_{vv} + a\tau\mathbf{I} \quad (19)$$

$$\hat{\mathbf{S}}_{p\omega} = \mathbf{S}_{p\omega} + \tau\mathbf{S}_{v\omega} \quad (20)$$

$$\hat{\mathbf{S}}_{\phi\omega} = \mathbf{H}\mathbf{S}_{\phi\omega} + \tau\mathbf{G}\mathbf{S}_{\omega\omega} + \frac{1}{2}\alpha\tau^2\mathbf{I} \quad (21)$$

$$\hat{\mathbf{S}}_{v\omega} = \mathbf{S}_{v\omega} \quad (22)$$

$$\hat{\mathbf{S}}_{\omega\omega} = \mathbf{S}_{\omega\omega} + \alpha\tau\mathbf{I} \quad (23)$$

These equations are used in the actual numerical computations, for efficiency.

4.2 Changes for Extrapolation Based on Angular Momentum

If it is assumed that torque instead of angular acceleration is random, the following changes are needed. (These have not yet been implemented in the tracker.) Angular momentum λ would be used instead of angular velocity ω in equation (4), in equation (69) in section 7, and in the subscripts of $\mathbf{S}$ (and similarly for the predicted quantities). The meaning of α would be the power spectrum of torque instead of the power spectrum of angular acceleration. The origin of the object coordinate system should be at the object center of mass in this case. To do the extrapolation of orientation, the computations in section 4.1 need to be changed, as described below.

The relationship between angular momentum and angular velocity is

$$\lambda = \mathbf{M}\omega \quad (24)$$

and thus

$$\omega = \mathbf{M}^{-1}\lambda \quad (25)$$

where $\mathbf{M}$ is the moment of inertia matrix of the object (a tensor of the second rank), expressed in the fixed coordinate system. In the object coordinate system, the moment of inertia matrix $\mathbf{M}'$ is constant and is assumed to be known. From it, $\mathbf{M}$ can be computed as follows, by using the rotation matrix $\mathbf{R}$ corresponding to the quaternion R:

$$\mathbf{M} = \mathbf{R}\mathbf{M}'\mathbf{R}^T \quad (26)$$

and thus

$$\mathbf{M}^{-1} = \mathbf{R}(\mathbf{M}')^{-1}\mathbf{R}^T \quad (27)$$

since $\mathbf{R}^{-1} = \mathbf{R}^T$.

One way of proceeding is to integrate Euler's equations (see Goldstein [1980]). However, if the rotation that occurs in the interval τ is small, a numerical integration with a step size of τ can be done with reasonable accuracy, and things simplify considerably. (Even though the angular velocity would then be assumed to be constant over extrapolation interval τ, the filtering action usually extends over a considerably longer interval, as explained in section 9, and the angular velocity will vary over this interval because of the effects of a changing $\mathbf{R}$ acting through (27) and (25).) All that needs to be done is to use ω from (25) in (5), and to use $\mathbf{G}\mathbf{M}^{-1}$ instead of $\mathbf{G}$ in (7) and (14)–(23). This substitution for $\mathbf{G}$ does the error propagation correctly from the angular momentum, since from (25) $\mathbf{M}^{-1}$ is the transformation matrix from angular momentum to angular velocity. In order to achieve greater accuracy with this method, the interval τ could be broken into smaller intervals, with the above process repeating (producing a more accurate numerical integration). A compromise might be to do this (or to use Euler's equations) only for the computation of $\hat{R}$ and $\hat{\omega}$, and to use the one-step (or an intermediate-step) error propagation as above to compute $\hat{\mathbf{S}}$, in order to save time. In this case, better accuracy in the error propagation could be achieved by using the average of ω and $\hat{\omega}$ instead of ω in (5).

The use of random torque instead of random acceleration would be appropriate for a satellite freely tumbling in space. The torque on it would be very small, and thus α could be made very small, with this formulation. (Perhaps it even could be zero, although some allowance probably should made for inaccuracies in $\mathbf{M}'$ and for the effect of nonlinearities on a possibly large error in initial conditions.) Even though the angular momentum of the satellite would be nearly constant, the angular velocity would not be, unless the satellite were rotating about one of its principal axes.

5 Projection

The predicted object pose, the object model, and the camera model are used to compute which features are expected to be visible from the camera position. At

present, for the purpose of the visibility computation only, the tracking program assumes that the object is convex. Therefore, this computation simply notes which faces are turned toward the camera; and all vertexes, other point features, and edges associated with these faces are assumed to be visible (except for faces that are seen nearly edge on, which are considered questionable and thus are not used). However, in the future we may implement a process for determining visibility of features, which will work in the general case including concave objects. This probably will be derived from Hedgley's hidden-line algorithm [Hedgley 1982], but with some improvements to take advantage of the fact that not everything needs to be computed every time while tracking. For example, since the object is rigid, only certain faces can hide certain other faces, and the corresponding information can be precomputed as part of the object model. Also, while the object is moving, not much changes from one image to the next with a given camera, and thus some things would not have to be recomputed. When the features used are edges, for any edge which is partially visible the corresponding vertexes will be used below.

The position of vertex (or other point feature) i in object coordinates $\boldsymbol{\rho}_i'$ is given as part of the object model. By using the predicted data, this is rotated to produce the vector from the object origin to the vertex in fixed coordinates $\boldsymbol{\rho}_i$, and is translated to produce the vertex position in fixed coordinates $\mathbf{r}_i$, as follows:

$$\boldsymbol{\rho}_i = \hat{\mathbf{R}}\boldsymbol{\rho}_i' \tag{28}$$

$$\mathbf{r}_i = \hat{\mathbf{p}} + \boldsymbol{\rho}_i \tag{29}$$

where $\hat{\mathbf{R}}$ is the rotation matrix corresponding to the quaternion $\hat{R}$.

Then each vertex is projected into the image plane to produce its image coordinates x_i and y_i, as follows in the currently implemented tracker:

$$x_i = \frac{(\mathbf{r}_i - \mathbf{c}_c) \cdot \mathbf{c}_h}{(\mathbf{r}_i - \mathbf{c}_c) \cdot \mathbf{c}_a} \tag{30}$$

$$y_i = \frac{(\mathbf{r}_i - \mathbf{c}_c) \cdot \mathbf{c}_v}{(\mathbf{r}_i - \mathbf{c}_c) \cdot \mathbf{c}_a} \tag{31}$$

where $\mathbf{c}_c$, $\mathbf{c}_a$, $\mathbf{c}_h$, and $\mathbf{c}_v$ constitute the camera model as defined by Yakimovsky and Cunningham [1978], and are assumed to be known from a previous calibration, as described by Gennery et al. [1987]. (This camera model includes the central projection and a general affine transformation in the image plane.) Here $\mathbf{c}_c$ is the camera position, $\mathbf{c}_a$ is a unit vector perpendicular to the image plane (towards the scene), and the $\mathbf{c}_h$ and $\mathbf{c}_v$ vectors combine information that specifies the directions in the image plane, the scales, and the zero offsets of the x (horizontal) and y (vertical) axes, respectively. Additional terms for lens distortion can be included in (30) and (31), and we now can include them in our camera calibration [Gennery 1991]. But, since their effect is small for reasonable lenses, these lens distortion terms can be omitted from the partial derivatives below with no appreciable loss of accuracy.

In order to do the adjustment, the partial derivatives of the image coordinates with respect to the object pose will be needed. The partial derivatives with respect to the position of the point in space can be obtained by differentiating (30) and (31), to produce the following:

$$\frac{\partial x_i}{\partial \mathbf{r}_i} = \left(\frac{\mathbf{c}_h - x_i \mathbf{c}_a}{(\mathbf{r}_i - \mathbf{c}_c) \cdot \mathbf{c}_a} \right)^T \tag{32}$$

$$\frac{\partial y_i}{\partial \mathbf{r}_i} = \left(\frac{\mathbf{c}_v - y_i \mathbf{c}_a}{(\mathbf{r}_i - \mathbf{c}_c) \cdot \mathbf{c}_a} \right)^T \tag{33}$$

where the transpose is indicated because the derivative of a scalar with respect to a vector is conventionally considered to be a row matrix. The derivatives with respect to object position $\mathbf{p}$ are identical to these, because of (29). The derivatives with respect to orientation can be found by considering the effect of an infinitesimal object rotation $\boldsymbol{\phi}$. It will cause a point on the object to move by the vector $\boldsymbol{\phi} \times \boldsymbol{\rho}_i$. The dot product of this with (32) produces the effect of $\boldsymbol{\phi}$ on x_i,

$$\Delta x_i = \boldsymbol{\phi} \times \boldsymbol{\rho}_i \cdot \left(\frac{\partial x_i}{\partial \mathbf{r}_i} \right)^T = \boldsymbol{\rho}_i \times \left(\frac{\partial x_i}{\partial \mathbf{r}_i} \right)^T \cdot \boldsymbol{\phi} \tag{34}$$

and similarly for y_i, where we have used the fact that the vectors in the scalar triple product may be cyclicly permuted. The expression dotted with $\boldsymbol{\phi}$ then consists of the partial derivatives of x_i with respect to $\boldsymbol{\phi}$. These results can now be combined to produce the 1×6 matrixes of partial derivatives of x_i and y_i with respect to position $\mathbf{p}$ and orientation increment $\boldsymbol{\phi}$ (together denoted $\mathbf{P}$), which can be assembled into a 2×6 matrix $\mathbf{B}_i$, as follows:

$$\mathbf{B}_i = \begin{bmatrix} \dfrac{\partial x_i}{\partial \mathbf{P}} \\ \dfrac{\partial y_i}{\partial \mathbf{P}} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial x_i}{\partial \mathbf{r}_i} & \left(\boldsymbol{\rho}_i \times \left(\dfrac{\partial x_i}{\partial \mathbf{r}_i} \right)^T \right)^T \\ \dfrac{\partial y_i}{\partial \mathbf{r}_i} & \left(\boldsymbol{\rho}_i \times \left(\dfrac{\partial y_i}{\partial \mathbf{r}_i} \right)^T \right)^T \end{bmatrix} \tag{35}$$

If the images are obtained from a television camera that is raster scanned in real time with continuous exposure, the different parts of the changing image will be sampled at different times. (It is assumed here that noninterlaced scanning is used.) Since the normal scan is from top to bottom, the first line of the picture corresponds to a time almost one frame time (except for the vertical blanking time) earlier than that of the last line. (The image also is blurred over the exposure time of approximately one frame time. The time referred to here is the center of this exposure time.) If the time between sampled images (τ) is much larger than the frame time, this time shift probably is not important. However, if τ is small and the object is moving rapidly, it could cause significant error. An approximate correction for this effect can be done by incrementing the x_i and y_i values from (30) and (31) according to their projected velocities times the amount of the time offset. This can be done by using the matrix of partial derivatives $\mathbf{B}_i$ defined above, to produce the following corrections to be added to x_i and y_i:

$$\begin{bmatrix} \delta x_i \\ \\ \delta y_i \end{bmatrix} = \mathbf{B}_i \begin{bmatrix} \hat{\mathbf{v}} \\ \\ \hat{\omega} \end{bmatrix} \delta t \tag{36}$$

where δt is the time offset from the nominal time for the frame (preferably the center of the integration time of the center line, which is approximately the time of the previous vertical sync pulse) to the time of this scan line as predicted by y_i from equation (31). Doing only this ignores the nonlinear effects of the propagation from 3D pose to 2D position, the fact that the discrepancies computed in section 6 are now a function of the velocity vectors in addition to the object pose, and the fact that the corrected y_i instead of the uncorrected y_i should be used in computing δt. However, if the corrections were so large that these effects became important, the image probably would be blurred so much that the feature detectors wouldn't work anyway. The corrections from (36) are used in the current tracker, although there is a provision to omit them. (The corrections would be omitted if the camera is shuttered or the scene is illuminated by strobe lights, so that the exposure is effectively instantaneous. The nominal time for the frame then would be the time of the open shutter or strobe flash, which should occur during the previous vertical blanking interval.)

6 Measurement

6.1 Point Features

If the features to be used consist of points (such as vertexes or lights), then their predicted positions x_i and y_i and their partial derivatives, as computed in Projection, are used in a straightforward way, as in a standard linearized weighted least-squares adjustment [Mikhail 1976], to produce the 6×6 matrix **N** (coefficients of the normal equations) and the 6×1 matrix **C** (constants in the normal equations), as follows:

$$\mathbf{N} = \Sigma \mathbf{B}_i^T \mathbf{W}_i \mathbf{B}_i \tag{37}$$

$$\mathbf{C} = \Sigma \mathbf{B}_i^T \mathbf{W}_i \begin{bmatrix} x - x_i \\ y - y_i \end{bmatrix} \tag{38}$$

where x and y are the measured position of the feature which has been found near the predicted position (for lights, currently obtained by thresholding and computing the centroid), $\mathbf{W}_i$ is a 2×2 weight matrix (which should be the inverse of the covariance matrix of the measured feature position), and the summations are over all of the features.

6.2 Preliminary Computations for Edges

The tracker usually uses information all along the predicted edges of the object, derived from brightness edges detected in the image. Thus **N** and **C** need to be computed in a different manner from the above. (It would of course be possible to use both point and edge information, in which case, **N** and **C** would be computed as the sum of the values from equations (37) and (38) above and (60) and (61) below.)

First, the predicted edge must be computed. The pair of predicted vertex positions x_i and y_i that are at the opposite ends of each object edge (according to the object model) will be indicated here by replacing the subscript i with the subscripts 1 and 2. Thus the length of the predicted edge and its direction cosines are

$$l = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{39}$$

$$c_x = \frac{x_2 - x_1}{l} \tag{40}$$

$$c_y = \frac{y_2 - y_1}{l} \tag{41}$$

Next, a coordinate system is defined aligned with the predicted edge, such that g is the distance parallel to the edge from point 1 and h is the perpendicular distance from the edge. Thus, for any point x, y:

$$g = c_x(x - x_1) + c_y(y - y_1) \tag{42}$$

$$h = c_x(y - y_1) - c_y(x - x_1) \tag{43}$$

Note that g and h form a rectangular coordinate system in pixel space, and not in the physical space of the actual image plane, since x and y are measured in pixels, which may not be square. But that is ordinarily what is wanted, since the edge detector is defined in terms of the sampled data, and thus its x and y measurements in pixel space will tend to be equally accurate and to have uncorrelated errors.

The partial derivatives of h with respect to the parameters (object pose) will be needed. At the ends of the edge, the negative of these will be denoted by $\mathbf{A}_1$ and $\mathbf{A}_2$, which can be obtained by differentiating (43) to produce (since at end 1, $x - x_1 \approx 0$ and $y - y_1 \approx 0$, and thus the derivatives of c_x and c_y can be ignored, and similarly at end 2)

$$\mathbf{A}_1 = c_x \frac{\partial y_1}{\partial \mathbf{P}} - c_y \frac{\partial x_1}{\partial \mathbf{P}} \tag{44}$$

$$\mathbf{A}_2 = c_x \frac{\partial y_2}{\partial \mathbf{P}} - c_y \frac{\partial x_2}{\partial \mathbf{P}} \tag{45}$$

where the derivatives with respect to $\mathbf{P}$ are obtained from $\mathbf{B}_i$ (see (35)). (The reason for changing the sign is that h represents an observed value minus an adjusted value, and the partial derivatives in the usual least-squares adjustment formulation are of the adjusted value.) If needed, the derivatives of h at any point along the line could be obtained by a linear interpolation of the values at the end points, as follows:

$$\frac{\partial h}{\partial \mathbf{P}} = -\frac{l - g}{l}\mathbf{A}_1 - \frac{g}{l}\mathbf{A}_2 \tag{46}$$

If lens distortion is included in the camera model, long object edges should be broken into segments short enough to project accurately as straight lines, for use in computing h from (43). However, the other quantities, including the partial derivatives, do not need high accuracy, and thus for speed they can be computed without considering this segmentation, unless the distortion is very large.

6.3 Simple Edge Measurement

The edge information can be measured and collected in a variety of ways, differing in their sophistication in the use of the available information and in the amount of computation required. In the currently implemented tracker, brightness edges in the image are detected by using a hardware edge detector called IMFEX [Eskenazi & Wilf 1979], which is similar to the Sobel operator with thresholding and thinning and which runs at the usual video rate. The tracker uses this edge information in the following very simple manner.

The portions of the predicted line within about five pixels from the end points (vertexes) are ignored, in order to avoid areas where the edge detector will be less accurate because of conflicting edge information. For the remaining portion, points spaced at about three-pixel intervals are selected, since the edge operator is three pixels wide. For each of these points, a search out to five pixels away is made for the nearest detected edge element, with the search being done either in the x or in the y direction, whichever is more nearly perpendicular to the predicted edge. If an edge element is detected on a particular search, its x and y image coordinates are converted to g and h by (42) and (43). The values needed below in (57) and (58) for measurement u and weight w are

$$u = h \tag{47}$$

$$w = \frac{w_a}{\sigma^2} \tag{48}$$

where σ is the assumed accuracy of the measured edge positions, and where w_a is the a priori weight for this edge. Each a priori weight can have a value from 0 to 1. Currently, for edges that form the boundary between the object and the background as seen from the camera, the a priori weight is an input constant, and for each interior edge the a priori weight is computed as the maximum of a color weight and a shape weight. The color weight varies from 0 to 1 linearly over a specified range of difference in reflectance of the two faces as specified in the object model. The shape weight varies from 0 to 1 linearly over a specified range of the negative of the cosine of the angle between the two faces as specified in the object model. (Therefore, more weight is given to edges that should be easier to detect because of different reflectances or orientations of the faces that meet at them.)

6.4 Advanced Edge Measurement

One obvious improvement would be to use the edge direction information from the edge detector, in addition to the edge strength information. If the measured direction differs too much from the predicted direction, the edge element should not be used. (If the edge arises from a reflectivity difference between coplanar nonspecular faces in the object model, it would be desirable to require also that the polarity of the edge be correct.) Instead of an all-or-nothing choice here, the edge element could be given a variable weight according to how closely the directions agreed. Similarly, the edge strength could influence a variable weight, instead of simply being thresholded. A search accurately approximating (to the nearest pixel) the perpendicular to the predicted edge could be used. Also, the portion rejected at the end points could be made variable, according to the angle at which predicted edges meet at the vertex.

Some further possible improvements concern the effect of the distance of the detected edge element from the predicted position (h) and of the number of edge elements that might be found on a given search (along a perpendicular from a particular point on the predicted line). (The present program uses only the nearest one and gives it constant weight unless it is too far away, in which case it is ignored completely, as previously stated.) One way of including this extra information that is in some sense optimum will now be described.

First, the variance of h, considering both the measurement errors (whose standard deviation is σ) and the prediction errors (due to the uncertainty in object pose, whose covariance matrix is $\hat{\mathbf{S}}_{\mathbf{PP}}$), is

$$\sigma_h^2 = \sigma^2 + \frac{\partial h}{\partial \mathbf{P}} \hat{\mathbf{S}}_{\mathbf{PP}} \left(\frac{\partial h}{\partial \mathbf{P}} \right)^{\mathsf{T}} \tag{49}$$

By using (46), this can be rewritten as follows:

$$\sigma_h^2 = \sigma^2 + \mathbf{A}_1 \hat{\mathbf{S}}_{\mathbf{PP}} \mathbf{A}_1^{\mathsf{T}} + 2(\mathbf{A}_1 \hat{\mathbf{S}}_{\mathbf{PP}} \mathbf{A}_2^{\mathsf{T}} - \mathbf{A}_1 \hat{\mathbf{S}}_{\mathbf{PP}} \mathbf{A}_1^{\mathsf{T}}) \frac{g}{l} + (\mathbf{A}_1 \hat{\mathbf{S}}_{\mathbf{PP}} \mathbf{A}_1^{\mathsf{T}} - 2\mathbf{A}_1 \hat{\mathbf{S}}_{\mathbf{PP}} \mathbf{A}_2^{\mathsf{T}} + \mathbf{A}_2 \hat{\mathbf{S}}_{\mathbf{PP}} \mathbf{A}_2^{\mathsf{T}}) \frac{g^2}{l^2} \tag{50}$$

which is a second-degree polynomial in g/l, whose three coefficients can be precomputed before the object edge is examined.

Then, for each edge element detected at a distance h from the predicted edge, the relative weight β, which is equivalent to a probability density function assuming that the errors have the normal (Gaussian) distribution, can be computed as follows:

$$\beta = \frac{\gamma w_a}{\sqrt{2\pi}\,\sigma_h} \exp\left(-\frac{h^2}{2\sigma_h^2} \right) \tag{51}$$

where γ is a factor to take care of the effects of edge direction and magnitude previously discussed (= 1 for a strong edge in exactly the right direction), and where w_a is the a priori weight defined in section 6.3. (The Gaussian function in (51) could be approximated by a table lookup, for speed.) These values would be used to compute the following weighted moments:

$$m_0 = \Sigma\beta \tag{52}$$

$$m_1 = \Sigma\beta h \tag{53}$$

$$m_2 = \Sigma\beta h^2 \tag{54}$$

where the summations are over all edge elements found in a particular search (along a perpendicular to the predicted edge). The search would extend sufficiently far to make β negligibly small (perhaps $4\sigma_h$ in each direction).

Finally, these moments would be used to compute the combined measurement and its absolute weight, as follows:

$$u = \frac{m_1}{m_0} \tag{55}$$

$$w = \frac{m_0}{(f + m_0)\left(\sigma^2 + \frac{m_2}{m_0} - u^2 \right)} \tag{56}$$

where f is a given quantity that represents the a priori probability density of false edge elements being detected (due to extraneous markings or shadows on the object or noise in the image). The justification of (55) and (56) is as follows. The expression for u is just the weighted average of the detected edge-element positions, and thus produces a reasonable value to use for the combined measurement. In the expression for w, the second expression in parentheses in the denominator denotes the accuracy (variance) of the measurement. This consists of two parts: the a priori variance σ^2, and the variance about the mean of the detected edge elements $m_2/m_0 - u^2$. (This latter effect is included because, if several elements have been detected spread over a wide area, it becomes very uncertain where the true edge is.) The reciprocal of this total variance is then the appropriate weight to use in a least-squares

adjustment. However, the factor $m_0/(f + m_0)$ is included so that, if the only elements detected are so far away from the predicted position that it is more likely that they are extraneous features rather than the desired object edge, the weight is reduced accordingly. (For example, if $f = 0.01$, which means that an extraneous edge would be expected about once every 100 pixels on a one-dimensional scan, the Gaussian function is equal to f at $h = 2.72\sigma_h$. Thus, anything beyond this point would not get much weight; but if an edge element were detected well inside this point, it would get nearly full weight, since it likely would be the true edge.)

This improved method of collecting the edge information (according to (51)–(56)) would slow down the tracker somewhat if done in the same computer. However, a parallel or pipelined image-processor device such as PIFEX [Gennery & Wilcox 1985] would be able to implement, at high speed, improved edge detectors and an approximation to the above improved edge-collection technique. Implementing these changes in such a device would make the tracker more robust and also slightly faster, since the search in effect would be done in this device also.

6.5 Combining Measurements from All Edges

The measurements u and weights w obtained by one of the methods in sections 6.3 and 6.4 must be collected into an overall solution. One way to do this would be to use them directly in the usual equations for **N** and **C**, considering u to be the observations and $\partial h/\partial \mathbf{P}$ to be the partial derivatives of the observed quantities with respect to the parameters. However, this would require computing the 6×6 components of **N** and the 6×1 components of **C** for every u, which currently is every third pixel along the predicted object edges, and summing over all of these values. A much faster way is to collect the information from each predicted edge, which requires summing only a 2×2 matrix and a 2×1 matrix, and then combining this information into the overall solution by computing the larger matrixes only for each complete predicted object edge and summing over them. In effect, what is done is to fit a straight line to the measurements along each predicted object edge, and then to combine all of these line fits into an overall solution. Under the linear approximation, which is valid if the fitted line is close to the predicted line, these methods are mathematically equivalent. The two intermediate parameters to be adjusted are the values of h at the two end points, that is, by how much the fitted line misses the predicted vertex at each end. The weighted least-squares fit for these is

$$\mathbf{W} = \Sigma \begin{bmatrix} 1 - \frac{g}{l} \\ \\ \frac{g}{l} \end{bmatrix} w \begin{bmatrix} 1 - \frac{g}{l} & \frac{g}{l} \end{bmatrix} \tag{57}$$

$$\mathbf{U} = \Sigma \begin{bmatrix} 1 - \frac{g}{l} \\ \\ \frac{g}{l} \end{bmatrix} wu \tag{58}$$

$$\begin{bmatrix} h_1 \\ \\ h_2 \end{bmatrix} = \mathbf{W}^{-1}\mathbf{U} \tag{59}$$

where the summations are over all the points along the predicted edge for which u and w were computed. **W** is the weight matrix (inverse of the covariance matrix) of the resulting h_1 and h_2 values.

These results can now be used in the overall solution, by using the facts that $\mathbf{A}_1$ and $\mathbf{A}_2$ as defined in equations (44) and (45) represent the partial derivatives of h_1 and h_2 with respect to the parameters, and $\mathbf{W}[h_1 \quad h_2]^T = \mathbf{U}$. Thus, in the usual normal equations of a weighted least-squares adjustment $\mathbf{ND} = \mathbf{C}$, where **D** represents the corrections needed to the parameters (without filtering), **N** (6×6) and **C** (6×1) are as follows:

$$\mathbf{N} = \Sigma [\mathbf{A}_1^T \quad \mathbf{A}_2^T]\mathbf{W} \begin{bmatrix} \mathbf{A}_1 \\ \\ \mathbf{A}_2 \end{bmatrix} \tag{60}$$

$$\mathbf{C} = \Sigma [\mathbf{A}_1^T \quad \mathbf{A}_2^T]\mathbf{U} \tag{61}$$

where the summations are over all of the predicted edges. Notice that h_1 and h_2 themselves are not used in (61), but only their weighted values **U**. Therefore, even if only one point is found along an entire predicted edge, **W** and **U** still contain useful information, although **W** is singular and thus h_1 and h_2 are undefined in this case. It is not necessary to compute (59), unless it is desired to check h_1 and h_2 for reasonable values. (The present tracker makes a crude check of this sort, but doing so is not very important, since most wildly erroneous measurements are prevented by not

using edge elements far from the predicted edge. However, h_1 and h_2 are used in computing one of the diagnostic quantities described in section 8.)

Some comments about efficiency in computing equations (57), (58), (60), and (61) can be made. Since small matrixes are involved here, efficiency can be gained without too much effort by multiplying out the matrix products and dealing with scalar quantities. Further savings can be made by using the fact that **W** and **N** are symmetrical matrixes. For example, in the summation in (60), only the 21 unique elements of **N** need to be accumulated. After the summations are complete, the 15 elements on one side of the main diagonal can be copied to the other side to complete the 36 elements of the matrix.

7 Adjustment

The **N** matrix computed in Measurement represents the combined weights of adjusted values of pose, and **C** represents the weighted adjusted values. That is, if no filtering were desired, the weighted least-squares solution for the six parameters, ignoring the predicted values except as initial approximations to be corrected, would apply the correction $\mathbf{N}^{-1}\mathbf{C}$ to the predicted values, and the accuracy of the results would be represented by the covariance matrix $\mathbf{N}^{-1}$.

The velocities are included in the adjustment by considering there to be twelve adjusted parameters, consisting of the column marixes **P** and **V**, where **P** is composed of the three components of position and the three components of incremental orientation, and **V** is composed of the three components of linear velocity and the three components of angular velocity. The measurements which produce **N** and **C** above contribute no information directly to **V**. However, the predicted values $\hat{\mathbf{P}}$ and $\hat{\mathbf{V}}$ can be considered to be additional measurements directly on **P** and **V** with covariance matrix $\hat{\mathbf{S}}$, and thus weight matrix $\hat{\mathbf{S}}^{-1}$. Therefore, by computing the vector mean [Mikhail 1976] (least-squares with measurements directly on the parameters) of the corrections to $\hat{\mathbf{P}}$ and $\hat{\mathbf{V}}$, the adjustment including the information contained in the predicted values in principle could be obtained as follows:

$$\mathbf{S} = \left(\begin{bmatrix} \mathbf{N} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} + \hat{\mathbf{S}}^{-1} \right)^{-1} \tag{62}$$

$$\begin{bmatrix} \mathbf{P} \\ \mathbf{V} \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{P}} \\ \hat{\mathbf{V}} \end{bmatrix} + \mathbf{S} \begin{bmatrix} \mathbf{C} \\ \mathbf{0} \end{bmatrix} \tag{63}$$

However, using (62) and (63) is inefficient and may present numerical problems, since the two matrixes are to be inverted are 12×12 and may be nearly singular. If **S** and $\hat{\mathbf{S}}$ are partitioned into 6×6 matrixes according to (1), a mathematically equivalent form can be produced by using the following equation (an identity if **N** and $\hat{\mathbf{S}}_{\mathbf{PP}}$ are square and the same size):

$$\left(\begin{bmatrix} \mathbf{N} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} + \begin{bmatrix} \hat{\mathbf{S}}_{\mathbf{PP}} & \hat{\mathbf{S}}_{\mathbf{PV}} \\ \hat{\mathbf{S}}_{\mathbf{PV}}^{\mathsf{T}} & \hat{\mathbf{S}}_{\mathbf{VV}} \end{bmatrix}^{-1} \right)^{-1} =$$

$$\begin{bmatrix} (\mathbf{I} + \hat{\mathbf{S}}_{\mathbf{PP}}\mathbf{N})^{-1}\hat{\mathbf{S}}_{\mathbf{PP}} & (\mathbf{I} + \hat{\mathbf{S}}_{\mathbf{PP}}\mathbf{N})^{-1}\hat{\mathbf{S}}_{\mathbf{PV}} \\ [(\mathbf{I} + \hat{\mathbf{S}}_{\mathbf{PP}}\mathbf{N})^{-1}\hat{\mathbf{S}}_{\mathbf{PV}}]^{\mathsf{T}} & \hat{\mathbf{S}}_{\mathbf{VV}} - \hat{\mathbf{S}}_{\mathbf{PV}}^{\mathsf{T}}\mathbf{N}(\mathbf{I} + \hat{\mathbf{S}}_{\mathbf{PP}}\mathbf{N})^{-1}\hat{\mathbf{S}}_{\mathbf{PV}} \end{bmatrix} \tag{64}$$

which is proved in appendix B.

Substituting (64) into (62) and (63), using the definitions in (1), and replacing **P** and **V** with their component vectors produces

$$\mathbf{S}_{\mathbf{PP}} = (\mathbf{I} + \hat{\mathbf{S}}_{\mathbf{PP}}\mathbf{N})^{-1}\hat{\mathbf{S}}_{\mathbf{PP}} \tag{65}$$

$$\mathbf{S}_{\mathbf{PV}} = (\mathbf{I} + \hat{\mathbf{S}}_{\mathbf{PP}}\mathbf{N})^{-1}\hat{\mathbf{S}}_{\mathbf{PV}} \tag{66}$$

$$\mathbf{S}_{\mathbf{VV}} = \hat{\mathbf{S}}_{\mathbf{VV}} - \hat{\mathbf{S}}_{\mathbf{PV}}^{\mathsf{T}}\mathbf{N}(\mathbf{I} + \hat{\mathbf{S}}_{\mathbf{PP}}\mathbf{N})^{-1}\hat{\mathbf{S}}_{\mathbf{PV}} \tag{67}$$

$$\begin{bmatrix} \mathbf{d} \\ \boldsymbol{\phi} \end{bmatrix} = \mathbf{S}_{\mathbf{PP}}\mathbf{C} \tag{68}$$

$$\begin{bmatrix} \mathbf{v} \\ \boldsymbol{\omega} \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{v}} \\ \hat{\boldsymbol{\omega}} \end{bmatrix} + \mathbf{S}_{\mathbf{PV}}^{\mathsf{T}}\mathbf{C} \tag{69}$$

where **d** and $\boldsymbol{\phi}$ are the corrections needed to the predicted position and orientation to produce the adjusted values. (These are shown in this manner because orientation must be handled differently from the others, as shown below.) Not only is this form more efficient computationally, but the matrix to be inverted $(\mathbf{I} + \hat{\mathbf{S}}_{\mathbf{PP}}\mathbf{N})$ is guaranteed to be nonsingular (unless $\hat{\mathbf{S}}_{\mathbf{PP}}$ or **N** is infinite), because both $\hat{\mathbf{S}}_{\mathbf{PP}}$ and **N** are nonnegative definite. Discussion of the relative efficiencies of these two forms and the usual form of Kalman filters is provided in section 9.

Since $\mathbf{S}_{\mathbf{PP}}$ and $\mathbf{S}_{\mathbf{VV}}$ are symmetrical matrixes, a

small amount of time could be saved by utilizing this fact in the above computations. Even if this is not done, because of numerical error these matrixes must still be forced to be symmetrical, perhaps by averaging each matrix and its transpose as produced by (65) and (67). (These statements would apply to the whole **S** matrix if (62) were used.) The reason this is necessary is that otherwise numerical error would gradually accumulate over many iterations, causing the matrixes to depart from symmetry and producing large errors. (Analytically, if **S** starts symmetrical, it will always remain symmetrical through the computations in the tracking loop. But with finite precision, this does not happen.)

The adjusted velocities were obtained above in (69). The adjusted position is

$$\mathbf{p} = \hat{\mathbf{p}} + \mathbf{d} \tag{70}$$

The orientation correction vector $\boldsymbol{\phi}$ must be used to produce the adjusted orientation quaternion R from the predicted orientation quaternion $\hat{R}$. Since $\boldsymbol{\phi}$ is considered to represent an infinitesimal rotation, the quaternion corresponding to it can be derived from equation (A3) by using the small-angle approximations for the trigonometric functions, to produce

$$F = (1, \frac{1}{2}\boldsymbol{\phi}) \tag{71}$$

Then the quaternion product $F\hat{R}$ gives the adjusted orientation, except for normalization. The normalization is needed both to correct for the fact that F is not exactly normalized (because of the small-angle approximation) and to correct numerical error that otherwise would accumulate over many iterations. Thus,

$$R = \frac{F\hat{R}}{\sqrt{\text{norm}\,(F\hat{R})}} \tag{72}$$

(The denominator in (72) is simply the square root of the sum of the squares of the four components of the numerator.)

The results of the adjustment are **p** from (70), R from (72), **v** and $\boldsymbol{\omega}$ from (69), and **S** from (65)–(67) according to (1). These (along with the time associated with the image from which they were derived) are used as output from the program, and they are used as input to Prediction for another iteration.

8 Performance Indicators

The tracker computes a few quantities that indicate how closely things are conforming to expectations. Currently, these quantities are used only as output for diagnostic purposes. However, in the future they may be used to change some of the parameters within the tracker so that it can adapt to changing conditions.

One such indication is how well the detected edge elements can be fit by straight lines corresponding to individual object edges, according to (59) in section 6. The measure of goodness of fit in a weighted least-squares adjustment is the quadratic form of the residuals and weight matrix, which is the quantity being minimized and which has an expected value equal to the number of observations minus the number of parameters being adjusted (the degrees of freedom of the adustment), if the weight matrix is the inverse of the covariance matrix of observation errors. However, this is equal to the quadratic form of the observations (or discrepancies in a nonlinear adjustment) and weight matrix minus the inner product of the adjusted parameters (or corrections in a nonlinear adjustment) and the vector of "constants" in the normal equations [Mikhail 1976]. This latter form does not require computing the residuals.

Therefore, the quadratic form that indicates the goodness of fit to an object edge is

$$q' = \Sigma wu^2 - h_1u_1 - h_2u_2 \tag{73}$$

where the summation is over all points used along this object edge, and where u_1 and u_2 are the elements of **U** (provided that there are at least two points along the edge so that h_1 and h_2 are determined). The number of degrees of freedom b' associated with this is the number of points minus two if the simple method of (47) and (48) is used. However, if the more elaborate method of (50)–(56) is used, this should be corrected by multiplying by the average of the probability $m_0/(f + m_0)$ that each detected feature is genuine. Thus,

$$b' = \left(1 - \frac{2}{k}\right) \Sigma \frac{m_0}{f + m_0} \tag{74}$$

instead of $k - 2$, where k is the number of points summed along the edge. Then q' and b' are summed over all edges in the object to obtain the total effective quadratic form q and the total effective degrees of freedom b (not necessarily an integer, as a true number of degrees of freedom would be), q and b are smoothed over time with a first-order recursive filter to obtain $\bar{q}$ and $\bar{b}$, and the ratio of the quadratic form to degrees of freedom (for both unsmoothed and smoothed values) is computed. The expected value of this ratio is unity. If it is considerably larger than unity, it indicates that perhaps the data is noisier than expected. Therefore, it might be possible to use the value of $\bar{q}/\bar{b}$ to adjust

the value of σ. (At present, q, b, q/b, and $\bar{q}/\bar{b}$ are used as output.)

Another type of indication is whether the corrections needed to object pose at each time are about what would be expected from the assumed nature of the measurement noise and the object acceleration. (If the corrections are unexpectedly large, this might indicate that the accelerations are higher than expected, if the noise level has been verified by the above method.) This information can be obtained by comparing the correction vectors $\mathbf{d}$ and $\boldsymbol{\phi}$ obtained from (68) in section 7 to their total covariance matrix, which represents the entire range of values and not just their uncertainty. This covariance matrix can be obtained as follows.

First, we can consider (68) to be obtained from an equivalent least-squares adjustment in which the measurements are $\mathbf{P}_o$ (the pose that would be obtained by fitting the object model to points obtained at this time only) with weight $\mathbf{N}$ (and thus covariance matrix $\mathbf{N}^{-1}$). In this equivalent adjustment, the discrepancies are $\mathbf{P}_o - \hat{\mathbf{P}}$, since the predicted values $\hat{\mathbf{P}}$ are the initial approximation for this adjustment. Therefore, in (68) we can replace $\mathbf{C}$ with $\mathbf{N}(\mathbf{P}_o - \hat{\mathbf{P}})$, so that the vector of corrections is $\mathbf{S_{PP}N}(\mathbf{P}_o - \hat{\mathbf{P}})$. Then, doing covariance propagation in the usual way produces the following for the total covariance matrix of the corrections, since $\mathbf{N}^{-1}$ is the covariance matrix of $\mathbf{P}_o$, $\hat{\mathbf{S}}_{\mathbf{PP}}$ is the covariance matrix of $\hat{\mathbf{P}}$, and it is assumed that the current measurements (leading to $\mathbf{P}_o$) are uncorrelated with anything previous (leading to $\hat{\mathbf{P}}$):

$$\begin{aligned}\mathbf{E} &= \mathbf{S_{PP}N}(\mathbf{N}^{-1} + \hat{\mathbf{S}}_{\mathbf{PP}})(\mathbf{S_{PP}N})^{\mathsf{T}} \\ &= \mathbf{S_{PP}N}(\mathbf{N}^{-1} + \hat{\mathbf{S}}_{\mathbf{PP}})\mathbf{NS_{PP}} \\ &= \mathbf{S_{PP}N}(\mathbf{I} + \hat{\mathbf{S}}_{\mathbf{PP}}\mathbf{N})\mathbf{S_{PP}} \\ &= \mathbf{S_{PP}N}(\mathbf{I} + \hat{\mathbf{S}}_{\mathbf{PP}}\mathbf{N})(\mathbf{I} + \hat{\mathbf{S}}_{\mathbf{PP}}\mathbf{N})^{-1}\hat{\mathbf{S}}_{\mathbf{PP}} \\ &= \mathbf{S_{PP}N}\hat{\mathbf{S}}_{\mathbf{PP}} \end{aligned} \tag{75}$$

where in the second line we have used the fact that $\mathbf{S_{PP}}$ and $\mathbf{N}$ are symmetrical, and in the fourth line we have substituted an expression for $\mathbf{S_{PP}}$ from (65).

A suitable indicator might be the quadratic form $[\mathbf{d}^{\mathsf{T}} \ \boldsymbol{\phi}^{\mathsf{T}}]\mathbf{E}^{-1}[\mathbf{d}^{\mathsf{T}} \ \boldsymbol{\phi}^{\mathsf{T}}]^{\mathsf{T}}$, whose expected value is 6; or separate quadratic forms for the position and orientation parts (each with expected value 3) might be desired so that the effects of linear acceleration and angular acceleration can be seen separately. However, to save computation time, the program currently uses only the main diagonal elements of $\mathbf{E}$ and computes the following two quantities:

$$\xi_p = \frac{d_1^2 + d_2^2 + d_3^2}{e_{11} + e_{22} + e_{33}} \tag{76}$$

$$\xi_\phi = \frac{\phi_1^2 + \phi_2^2 + \phi_3^2}{e_{44} + e_{55} + e_{66}} \tag{77}$$

where the numerical subscripts refer to individual elements of the matrix and vectors. The expected values of ξ_p and ξ_ϕ are unity. They are smoothed by a first-order recursive filter and used as output. In the future, it might be possible to use them to adjust the values of the acceleration parameters a and α.

9 Discussion of Filtering

The filtering action in the tracker represents an application of Kalman fitlering. However, in the usual formulation of the Kalman filter, the general equations corresponding to (62) and (63) are transformed by means of the "matrix inversion lemma" [Maybeck 1979] into mathematically equivalent but computationally different form in which the size of the matrix to be inverted is equal to the number of new observations instead of the number of parameters in the state vector, although several matrix multiplications also are then needed. Indeed, Kalman originally derived the filter in this latter form (but somewhat more general), directly from first principles [Kalman 1960]. Since the number of new observations usually is considerably less than the size of the state vector, the latter form usually is more efficient. However, here that is not the case, except for very simple objects.

In order to make a simple quantitative comparison of the efficiency of the various approaches, the number of multiplications and divisions, called "operations" below, will be used. This will be taken to be n^3 for inverting an $n \times n$ matrix, and lmn for multiplying an $l \times m$ matrix by an $m \times n$ matrix. (Including additions and subtractions would not change the relative performance of the different approaches by much).

Since (62) involves two inversions of 12×12 matrixes, it requires 3456 operations, and (63) requires 72 (not counting multiplying by the zeros), for a total of 3528 for this approach derived directly from the usual least-squares formulation.

The amount of computation when using the Kalman formulation in the usual way directly on the measured quantities depends on the number of features. For example, consider a cube (a fairly simple object). Typically nine edges are visible, and even the efficient technique described in section 6 of collecting all of the elements along each edge into two observations would

produce 18 observations, which is greater than the size of the state vector (12 here, as far as the adjustment is concerned, even though the actual state contains 13 quantities because of the redundancy contained in the quaternion used to represent orientation). The usual Kalman formulation (see [Maybeck 1979] or Appendix B) then would require about 16,000 operations (not counting multiplying by the zero derivatives with respect to the velocities). Running the Kalman filter over the edges in sequence (which is equivalent to recursive estimation [Mikhail 1976]) would also be inefficient (since of the nine solutions produced at each time point, only the last would be of any use). It would required about 5000 operations (not counting multiplying by zero). (These approaches would be efficient with only one or two features.)

However, by using the results of Measurement here, we can consider the observations to be direct measurements of the 6 incremental pose parameters. Therefore, the observations would be $\mathbf{N}^{-1}\mathbf{C}$, with covariance matrix $\mathbf{N}^{-1}$. With only 6 observations, the usual Kalman equations ((B5)–(B7) for the case here), including the computation of the inverse of $\mathbf{N}$, would require 3096 operations (independently of the complexity of the object). Furthermore, as discussed in appendix B, in this case the measurement matrix would consist entirely of ones and zeros. Eliminating the multiplications involving it would reduce the number of operations to 1800 (by using equations (B8), (B9), and (B7)). This is more efficient than using (62) and (63) (requiring 3528 operations), but it is not as good as using (65)–(69), as we shall see. (Since the different forms are mathematically equivalent, it is of course possible to transform algebraically the Kalman equations into (65)–(69), as discussed in appendix B.)

Now consider the formulation actually used here, in which (62) was transformed into (65)–(67). The latter involve one 6×6 inversion and five multiplications of 6×6 matrixes, requiring 1296 operations, and (68) and (69) require 72 operations, for a total of 1368. This is only 39% as much as using (62) and (63) (and 76% as much as the fastest Kalman formulation described above). This saving is possible because of the sparseness of the matrix containing $\mathbf{N}$ in (62), which in turn is caused by the fact that, although velocity is included in the state vector, the observations are independent of velocity. This efficiency and the numerical considerations mentioned in section 7 are the reasons for using this formulation.

The fact that in the adjustment corrections are applied to the predicted values in order to obtain the adjusted values causes the solution to converge to the optimum solution just as it would in a standard linearized least-squares adjustment. Since the prediction is usually close to the actual pose of the object, the nonlinearities usually are small, and thus the convergence is very rapid here, if a and α are large. Therefore, in the absence of noise and high accelerations (since the prediction uses velocity), the tracker very closely tracks the actual movement of the object, and separate iterations at each time point are not needed. However, the fact that the predicted values are given some weight in the adjustment produces the filtering action, because of the memory of previous measurements contained in the predicted data. This filtering action determines the way that the tracker responds to noise and to acceleration, and it allows the use of stereo with noncoincident pictures. These two effects now will be discussed.

As in any Kalman filter, the amount of smoothing depends on the amount of plant noise that is assumed, here represented by the acceleration parameters a and α, and the weight given to the measurements, here represented by $\mathbf{N}$. If the acceleration parameters are small (or $\mathbf{N}$ is small), old information is given relatively high weight in the adjustment, and as a result there is a large amount of smoothing. If the acceleration parameters are large (or $\mathbf{N}$ is large), the effect of old information rapidly decays, and as a result there is not much smoothing. However, the precise nature of the smoothing depends on the particular prediction model used here.

In order to obtain a simple quantitative analysis of the filter, some approximations must be made. In particular, let us assume that the filter is linear and stationary. The linearity assumption has been made already in the design of the filter and is accurate when the deviation between the predicted values and actual values are small. Stationarity requires that the same input points with the same accuracy and same geometry are present at every time point, the time interval between these is constant, and sufficient time has elapsed for these conditions to cause the covariance matrix $\mathbf{S}$ to become constant. This assumption is accurate only when the angular velocity is so small that the object does not rotate appreciably during the time constant of the filter and when the set of detected edge points does not change appreciably from one time to the next.

When an edge comes into view or disappears from view, the assumption of stationarity can be grossly violated (both because of the differing data and because of the fact that the time interval may change considerably because of the differing amount of computing needed), and noncoincident stereo violates it in a way discussed below.

In general, the weights as stated by **N** are different in different dimensions. Thus the amount of smoothing will be different in different dimensions, with the directions of lower accuracy being smoothed more, as they should be. The directions for which different smoothing is produced in general are not aligned with the coordinate axes, but are aligned with the eigenvectors of **N**, provided that the different dimensions are scaled so that a and α are numerically the same in all six dimensions. (We have previously assumed that a is the same in all three spatial dimensions, and similarly for α. If the distance unit is chosen so that a and α are numerically equal, the stated condition is achieved, so that the only thing that causes different amounts of smoothing is **N**.) When things are scaled in this way, the eigenvalues of **N** control the amount of smoothing along each eigenvector, as if separate one-dimensional filters were being used in each of these six directions (as long as the linearity and stationarity assumptions are accurate). (This same condition holds in the more general case of arbitrary acceleration parameters, except that it is not just **N** that determines these directions.)

When the filtering action is analyzed quantitatively [Gennery 1990] for a linear, stationary, one-dimensional, second-order recursive filter of the type used here, one result is that the time constant of the filter is

$$c_t \approx \left(\frac{4\tau}{an}\right)^{1/4} \tag{78}$$

where n is the eigenvalue of **N** for this direction, and where a represents either a or α. This approximation is reasonably accurate when a is sufficiently small so that $c_t \gtrsim \tau$. The time constant c_t is the time required for the amplitude of oscillation of the filter output to decay by a factor of $1/e$ or for the phase of the oscillation to change by one radian, after an initial input disturbance. (Within the accuracy of the approximation, these two are equal for this filter.) The time constant thus gives a rough indication of the amount of smoothing.

When only one camera is used, the depth information comes only from the known size of the object and is not very accurate, since the object usually subtends a small angle at the camera. Therefore, the covariance matrix $\mathbf{S}_{pp}$ represents a long thin ellipsoid aligned with the camera-object line. (Similar remarks apply to the other portions of **S**.) If two or more cameras were used in an ordinary least-squares adjustment, the resulting $\mathbf{S}_{pp}$ matrix would be the inverse of the sum of the inverses of the $\mathbf{S}_{pp}$'s that would result from each camera alone. If the ellipsoids from each camera intersect at an appreciable angle, the amount of uncertainty in the depth direction is greatly reduced, and the resulting ellipsoid is not so elongated. (This is just ordinary stereo action.) If the multiple cameras are used at different times, the error ellipsoid will grow between the times of successive pictures. If the acceleration parameters are small, it will not grow very much, and thus the result is nearly the same as in the ordinary stereo case and the approximation in (78) above is accurate, provided that for **N** and τ the average of their values over all of the cameras is used. However, if the acceleration parameters are large, the error ellipsoid will grow by a large amount between pictures, and thus combining two of them may not reduce the depth uncertainty much. This is reasonable, since, if the object could randomly accelerate that much, its position could change unpredictably so much from one picture to the next that the two pictures could not be combined to produce reliable stereo information. In this case, the filter is not stationary.

The choice between the two ways of using multiple cameras (simultaneous or sequential) depends on the available hardware and timing considerations. If only one image can be stored at a time, then the sequential method (one camera on each time through the loop) must be used. If the object model is complicated and there are many features to find in the picture, Projection and Measurement will occupy most of the computing time, and thus it makes sense to use the information from each of the cameras immediately in Adjustment, to produce updated results reasonably rapidly. Thus the sequential method would be recommended. On the other hand, if the object is very simple and has few features to find, Projection and Measurement won't require much time. Therefore, they might as well be done for all images at once (if these can be stored), so that more complete results will be available from the adjustment, without the prediction uncertainty that

would enter if a wait until the next iteration (through another prediction and adjustment) were introduced. Such a simple case might seldom occur. However, the same situation exists if the hardware is sufficiently fast that the computations can be completed in one frame time of the television cameras. (The present version of the tracker uses only the sequential method.)

10 Results

The tracker has been tested extensively in the JPL Robotics Lab. A few examples are shown here. Our old setup [Gennery 1982] had two cameras 0.56 meter apart, each with 240 lines by 188 pixels per line. The tracker program ran on a General Automation SPC-16/85 computer.

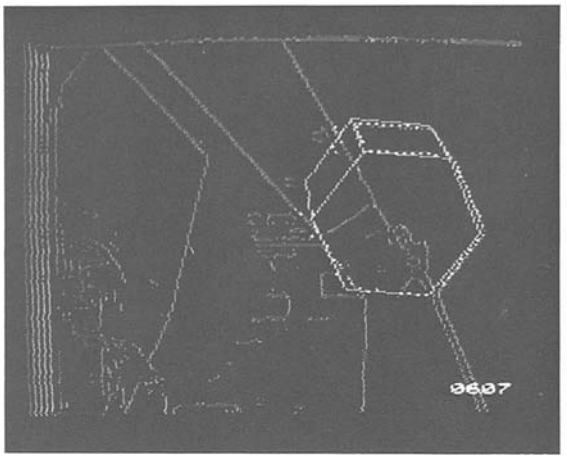
Using this setup, the tracker was tested by using a Unimation Puma arm under manual control to move an object in an arbitrary way as the program tried to track it. The tracker was tested using both one camera and two cameras, under various lighting conditions, with various background clutter, and with the object translating, rotating, and both. With only one camera, the program was fairly sensitive to clutter and lighting, but with two cameras the presence of the redundant information and of the accurate depth information from stereo caused much more robust operation.

The object used in most of these early experiments is a hexagonal prism 0.203 meter tall with side faces 0.128 meter wide, painted flat white. A rod protrudes from the center of one of the hexagonal faces for the purpose of grasping the object, but this rod is not included in the object model. In the test described below, this object was about 1.7 meters from the cameras. Pictures from the two cameras were taken alternately, and typically about 0.4 second of processing was required for each picture. In addition, in this case there was a deliberate delay of 0.1 second to prolong the display of the results (so that the total time through the main loop shown in figure 1 was around 0.5 second). The values used for the acceleration parameters were a = 1 mm^2/sec^3 and α = 0.0001 radian^2/sec^3. The assumed standard deviation σ of the edge measurements was one pixel.

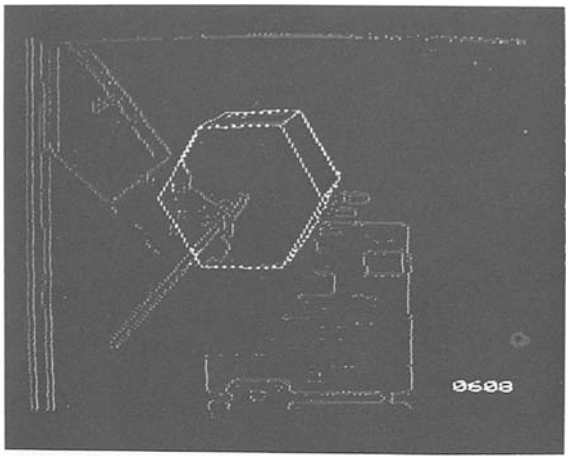
Figure 2 shows the results from six successive frames (of those used by the program), alternating between views from the left camera and the right camera. In the figure, the brightness edges found by the IMFEX edge detector are indicated by faint lines, the predicted visible object edges are indicated by brighter lines, and the edge elements used by the tracker are shown as yet brighter dots. (Detected edges from the object, some background clutter, the Puma arm, and some shadows can be seen in the figure.) In this figure, the object is rotating approximately at 0.27 radian/sec about an axis that is horizontal and perpendicular to the line of sight from the cameras (not one of the body axes). (It reached this velocity from zero in about 14 seconds, while the program was tracking it.) In the process of rotating the object, the Puma arm passed partly in front of the object, as can be seen in the figure. Nevertheless, the program continued to track the object, as can be seen from the fact that the bright lines (predicted object) are close to the faint lines (detected edges) produced by the object. (If a predicted edge lies exactly on a detected edge, hopefully the bright dots can be seen on the bright line in the figure, indicating that the detected edge is there.) The adjusted data probably would be even closer to the detected data, but were not displayed by the old version of the program.

Essentially this same test was repeated many times. When two cameras were used, it almost always was successful. When only one camera was used, the program often lost track, no matter which camera was used, although it always tracked through at least part of the run. The computed accuracy (combined standard deviation in three dimensions, equal to the square root of the sum of the three diagonal elements of the appropriate covariance matrix) of the adjusted data in tests such as this (with two cameras) typically was about 2 mm in position, 0.01 radian in orientation, 2 mm/sec in velocity, and 0.02 radian/sec in angular velocity. The filter time constants ranged approximately from one to two seconds.

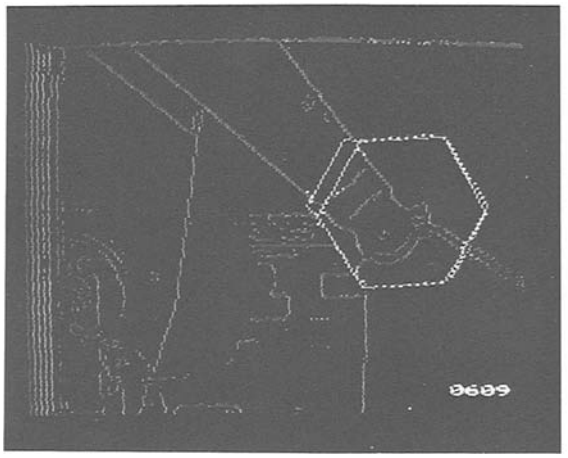
The new setup has three cameras, with two cameras at the same elevation 2.8 meters apart and another camera 1.3 meters above their midpoint, for the examples below. The camera focal length is 12.5 mm, each camera has 240 lines by 320 pixels per line (of which only 238 by 245 are useful from IMFEX), and the image size is 6.6 mm by 8.8 mm, resulting in 0.0275-mm square pixels. (Two other cameras, with 25 mm focal length, are mounted on an arm for closeup views and can be seen in figure 4, but they were not used in the examples presented here.) The cameras were calibrated as described by Gennery et al. [1987]. The tracker program runs on a Digital Equipment Corporation MicroVAX II.



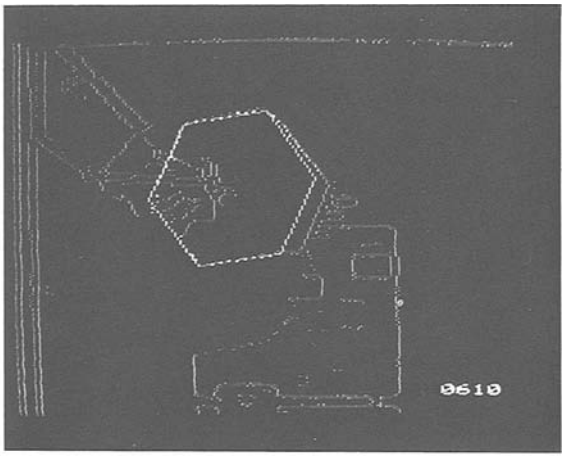
(a)



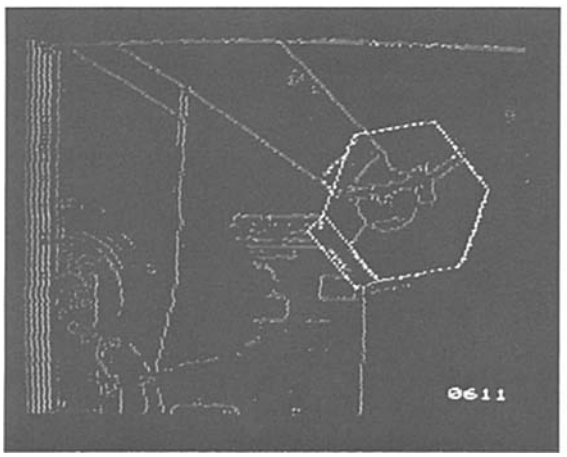
(b)



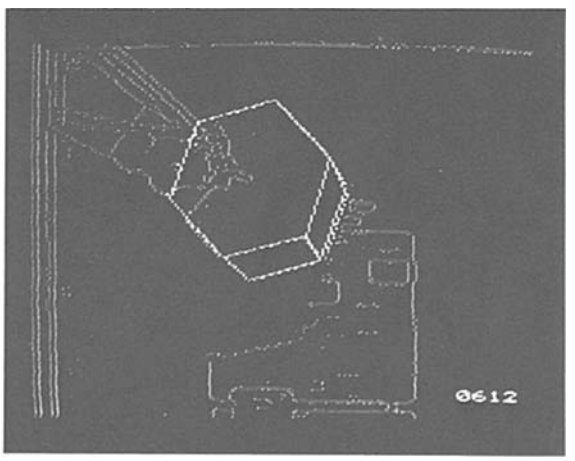
(c)



(d)



(e)



(f)

The new version of the program displays the adjusted position of the object superimposed on the raw digitized image from one of the cameras, in a separate image buffer from that used for the input edge data, so that the correctness of track can easily be seen. Usually only the visible vertexes and corners in the object model are shown (as white dots at the nearest pixel), instead of the edges, for speed. Figure 3 shows such results from tracking the same object (in the foreground) used in the example above, when it was about three meters from the cameras. (In this case, the program was able to process about six frames per second.)

Fig. 3. New sample track with vertexes overlaid on raw digitized picture.

Most of the new experiments used an imitation satellite, shown in figure 4 (and faintly in the background of the digitized image in figure 3). It has the approximate shape of a hexagonal prism 1.22 meters tall with side faces 0.76 meter wide. Each of the six side faces consists of a panel surrounded by an aluminum frame. The frame was covered with thermal protective foil (as shown in figure 4) for the later experiments. Five of the panels are real solar panels; the sixth is an aluminum panel containing a smaller white panel and a fluid coupling. The top of the object is open for suspension but is lined with black cloth to appear solid. The aluminum surfaces are partially specular; the foil is highly specular but somewhat crinkled. Two handles are attached to the white panel, and two grappling fixtures are attached to the frame beside the aluminum,

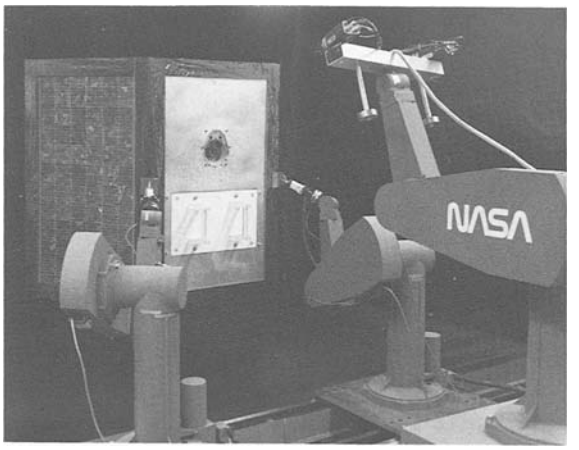


Fig. 4. Imitation satellite (held by arms).

panel. The object model used for the experiments includes the outline of the hexagonal prism, the six panels on the six faces, and the white panel. Each of the surfaces so defined was modeled as a planar region of constant reflectivity, with the shapes being hexagons for the top and bottom, rectangles with rectangular holes for the frame and the aluminum panel, and rectangles for the other panels. The wire-frame representation of the model is shown in figure 5. The coupling, handles,

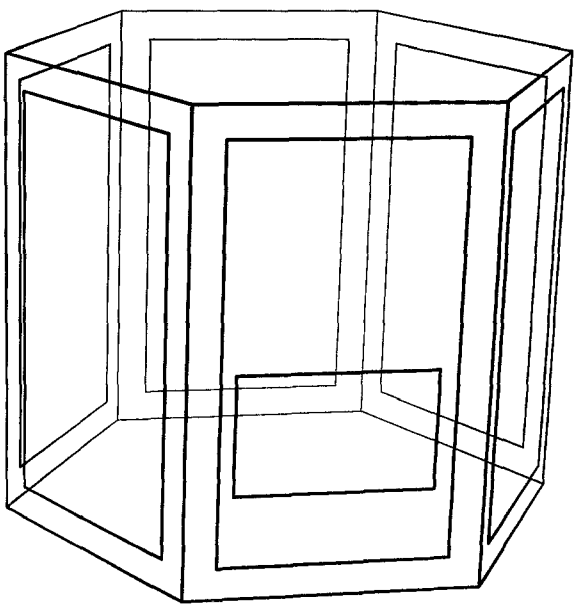


Fig. 5. Perspective view of object model of imitation satellite, showing all edges.

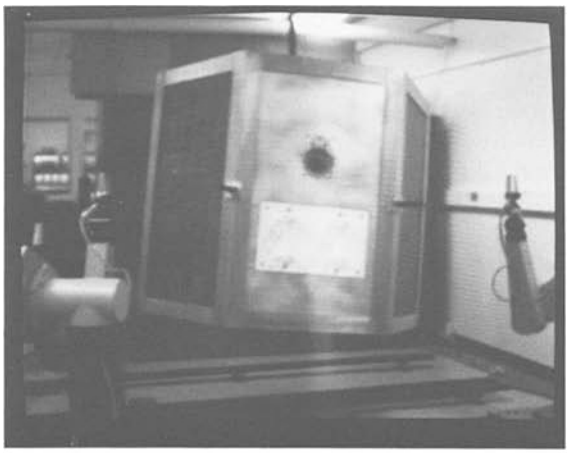
Fig. 2. Consecutive sample results from old version of program, for (a) left camera at time t_a, (b) right camera at t_a + 0.52 sec, (c) left camera at t_a + 0.97 sec, (d) right camera at t_a + 1.42 sec, (e) left camera at t_a + 1.80 sec, (f) right camera at t_a + 2.23 sec.

and other minor features are not included in the model. The grappling fixtures are included in the model for grappling purposes, but are not used by the vision system. The object was suspended from the ceiling so that it could move with six degrees of freedom, including unlimited rotation about the vertical axis. Its center was about four meters from the cameras; therefore, each pixel represents about nine millimeters on the object.

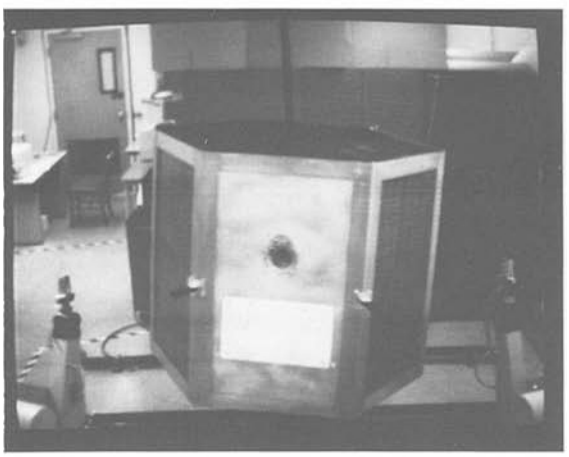
Figure 6 shows typical digitized pictures of the stationary imitation satellite without the foil, from the three cameras. Figure 7 shows the corresponding edge maps produced by IMFEX, with the portions not processed by IMFEX (roughly equal on the left and right) deleted. Figure 8 shows one frame of the results projected into the right picture, while tracking the moving object under these conditions. White dots, as before, show the computed vertex positions projected into the right picture. The object was rotating at 1.02 radian/sec about the vertical axis at the time of this frame. (It reached this velocity from zero in about 45 seconds, while the program was tracking it.) About 0.5 seconds was required for each frame (so that about 1.5 seconds was required to loop through the three cameras). For this test, $a = 10$ mm^2/sec^3, $\alpha = 0.0001$ radian^2/sec^3, and $\sigma = 1$ pixel. At the time of figure 8, the resulting filter time constants were around one second, and the computed total standard deviations of the adjusted data were 4.7 mm in position, 7.6 mradian in orientation, 5.7 mm/sec in velocity, and 14.0 mradian/sec in angular velocity. From the closeness of the white dots to the vertexes in the picture, it can be seen that the program was accurately tracking, in spite of the poor quality of the edge maps.

Similarly, figure 9 shows digitized pictures of the imitation satellite with the thermal protective foil (and with a different background and different arm positions), and figure 10 shows the corresponding edge maps produced by IMFEX. Figure 11 shows one frame of the results projected into the top picture, while tracking the object as it was rotating at 0.24 radian/sec about the vertical axis.

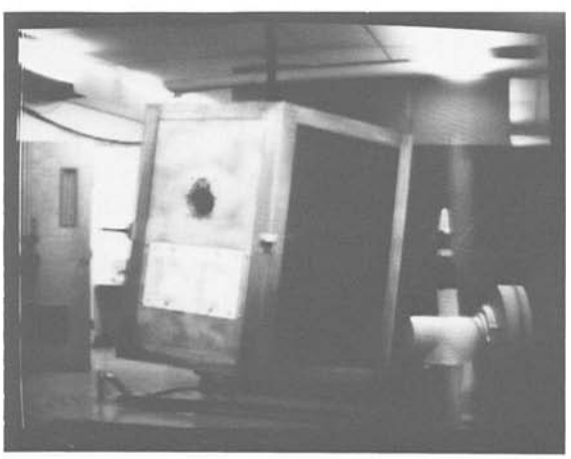
A test of absolute accuracy was done by having the program track an 8½×11-inch sheet of paper taped to the calibration fixture in one of its calibration positions (3.5 meters from the plane of the cameras). The difference between the three-dimensional positions computed by the tracker and accurately measured by hand was 6.5 mm (roughly in the direction from the fixture to the cameras, as one would expect). Some of this error



(a)



(b)



(c)

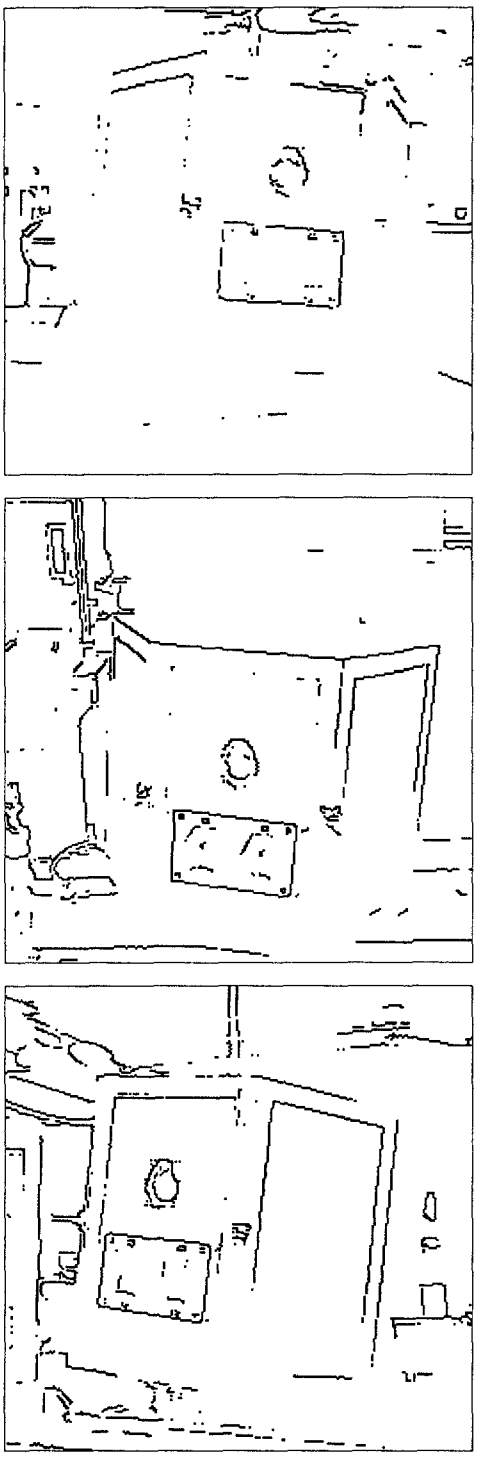


Fig. 7. Edge pictures corresponding to figure 6.

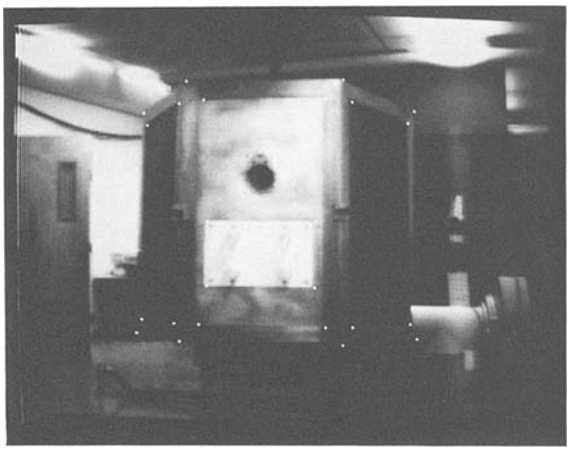


Fig. 8. Sample track of object in figure 6.

probably is due to miscalibration of the cameras, caused by the fact that lens distortion was not included in the camera model and possibly by errors in the multiple positions of the calibration fixture.

The tracker has been used in successful laboratory grappling experiments [Wilcox et al. 1989]. Figure 4 shows a typical position of the two Puma arms and the imitation satellite after the arms have grabbed it under autonomous control of the tracker and brought it to rest. Typical angular velocities in these experiments have been about 0.2 radian/sec, and typical position errors in the points of contact of the arms relative to the object have been around one centimeter.

It would be possible to model the arms, so that the tracker will know not to look for features in portions of the scene where an arm obscures the tracked object. However, at present this is not done. Therefore, edges detected on the arms act as spurious data to the tracker.

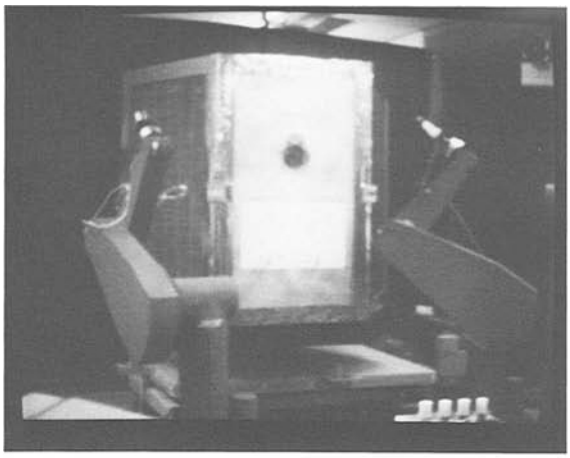
11 Conclusions

The method described here can track (in six degrees of freedom) a rapidly translating and rotating rigid three-dimensional object for which an object model is known, after being started approximately on track. (Examples were shown with rotations up to about 30° per frame.) With the aid of a hardware feature detector,

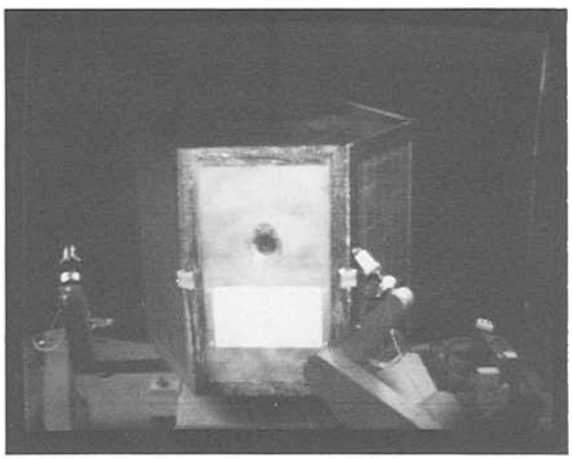
Fig. 6. Digitized pictures of imitation satellite (without foil) from the (a) left, (b) top, and (c) right cameras.

computational speeds of around two to six frames per second were achieved on fairly slow computers, when tracking objects of moderate complexity. Currently available hardware should allow the computations to be done at the usual video-frame rate for such objects.

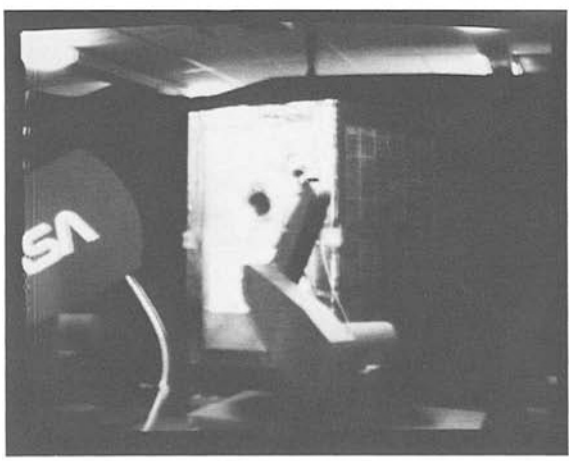
The method is able to tolerate a considerable amount of missing and spurious features, especially when stereo is used. This is because it looks for features (usually edge elements) only near their expected positions, because the typical abundance of features produces considerable overdetermination in the adjustment, and because of the smoothing produced by the filtering. Because of the spatial coherence produced by the object model and the temporal coherence produced by the prediction model used in the filter, finding the features by looking near their predicted positions in each image is more robust than tracking individual features in a sequence of noisy images.



(a)



(b)



(c)

Appendix A: Quaternions

The properties of quaternions will be briefly reviewed. Proofs and further information can be found in the references [Brand 1947; Corben & Stehle 1960; Goldstein 1980].

A quaternion Q can be defined to be a quadruple of real quantities q_0, q_1, q_2, and q_3. A quaternion also can be considered to be the combination of a scalar (corresponding to q_0) and a vector (whose components are q_1, q_2, and q_3). The notation used here for this representation is $Q = (s, \mathbf{v})$, for a quaternion Q with scalar part s and vector part $\mathbf{v}$. (Another way of looking at quaternions, as a generalization of complex numbers, with one real part and three imaginary parts, will not be used here.)

The product $R = PQ$ of two quaternions is defined as follows:

$$\begin{aligned} r_0 &= p_0q_0 - p_1q_1 - p_2q_2 - p_3q_3 \\ r_1 &= p_0q_1 + p_1q_0 + p_2q_3 - p_3q_2 \\ r_2 &= p_0q_2 + p_2q_0 + p_3q_1 - p_1q_3 \\ r_3 &= p_0q_3 + p_3q_0 + p_1q_2 - p_2q_1 \end{aligned} \quad \text{(A1)}$$

The equivalent definition using the scalar-vector notation is

$$(a, \mathbf{u})(b, \mathbf{v}) = (ab - \mathbf{u} \cdot \mathbf{v}, a\mathbf{v} + b\mathbf{u} + \mathbf{u} \times \mathbf{v}) \quad \text{(A2)}$$

Note that this is not commutative. However, quaternion multiplication is associative.

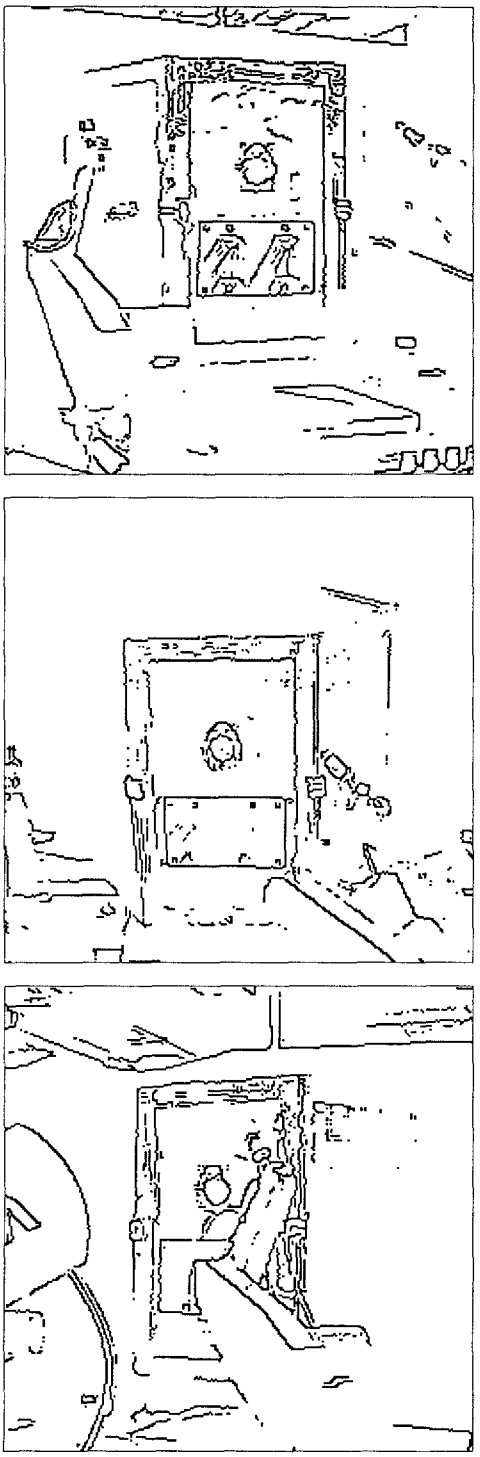


Fig. 10. Edge pictures corresponding to figure 9.

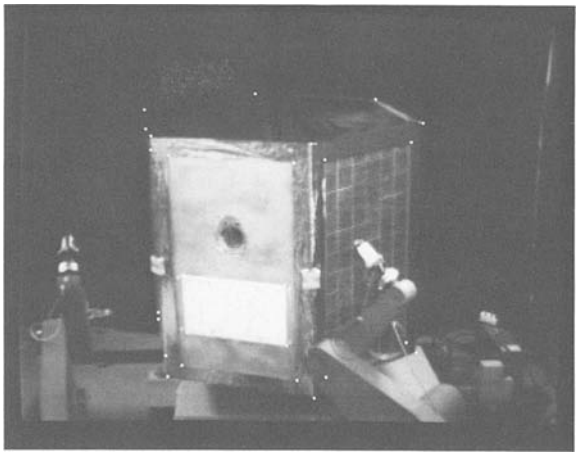


Fig. 11. Sample track of object in figure 9.

The conjugate of a quaternion Q (denoted by Q^*) is obtained by changing the sign of the vector portion (that is, the signs of q_1, q_2, and q_3). The conjugate of a product can be changed to a product of conjugates by reversing the order of multiplication (that is, $(PQ)^* = Q^*P^*$). The norm of a quaternion is the sum of the squares of its four components.

Any rotation in three-space can be represented by a quaternion whose norm is unity (known as a unit quaternion). By Euler's theorem, any such rotation can be represented by a single rotation angle θ about some unit vector $\mathbf{u}$. The quaternion R representing this rotation is

$$R = \left(\cos\frac{\theta}{2}, \mathbf{u}\sin\frac{\theta}{2}\right) \tag{A3}$$

In this case the four components of R are known as the Euler parameters. Successive rotations can be combined by multiplying the corresponding quaternions from right to left. Note that R and $-R$ represent rotations differing by one revolution, and thus represent the same orientation. (The quaternion representing the negative of the rotation represented by R is R^*.)

A vector can be represented by a quaternion by setting the scalar part to zero and setting the vector part to the vector. A vector represented by the quaternion V can be rotated by a rotation represented by the quaternion R as follows:

$$V' = RVR^* \tag{A4}$$

Fig. 9. Digitized pictures of imitation satellite (with foil covering frame) from the (a) left, (b) top, and (c) right cameras.

where the quaternion conjugate and quaternion product have been used. However, if there are several vectors all to be rotated the same, it is more efficient to compute the rotation matrix and to compute the matrix product of this matrix times each vector in the usual way ($\mathbf{v}' = \mathbf{R}\mathbf{v}$). The rotation matrix $\mathbf{R}$ corresponding to a unit quaternion R is

$$\mathbf{R} = \begin{bmatrix} r_0^2 + r_1^2 - r_2^2 - r_3^2 & 2(r_1r_2 - r_0r_3) & 2(r_1r_3 + r_0r_2) \\ 2(r_1r_2 + r_0r_3) & r_0^2 + r_2^2 - r_1^2 - r_3^2 & 2(r_2r_3 - r_0r_1) \\ 2(r_1r_3 - r_0r_2) & 2(r_2r_3 + r_0r_1) & r_0^2 + r_3^2 - r_1^2 - r_2^2 \end{bmatrix} \quad \text{(A5)}$$

Since $r_0^2 + r_1^2 + r_2^2 + r_3^2 = 1$ for a unit quaternion, (A5) is equivalent to the following alternative form:

$$\mathbf{R} = \begin{bmatrix} 2(r_0^2 + r_1^2) - 1 & 2(r_1r_2 - r_0r_3) & 2(r_1r_3 + r_0r_2) \\ 2(r_1r_2 + r_0r_3) & 2(r_0^2 + r_2^2) - 1 & 2(r_2r_3 - r_0r_1) \\ 2(r_1r_3 - r_0r_2) & 2(r_2r_3 + r_0r_1) & 2(r_0^2 + r_3^2) - 1 \end{bmatrix} \quad \text{(A6)}$$

Appendix B: Proof of Equation (64)

The equation

$$\left(\begin{bmatrix} \mathbf{N} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} + \begin{bmatrix} \hat{\mathbf{S}}_{\mathbf{PP}} & \hat{\mathbf{S}}_{\mathbf{PV}} \\ \hat{\mathbf{S}}_{\mathbf{PV}}^{\mathsf{T}} & \hat{\mathbf{S}}_{\mathbf{VV}} \end{bmatrix}^{-1} \right)^{-1} =$$

$$\begin{bmatrix} (\mathbf{I} + \hat{\mathbf{S}}_{\mathbf{PP}}\mathbf{N})^{-1}\hat{\mathbf{S}}_{\mathbf{PP}} & (\mathbf{I} + \hat{\mathbf{S}}_{\mathbf{PP}}\mathbf{N})^{-1}\hat{\mathbf{S}}_{\mathbf{PV}} \\ [(\mathbf{I} + \hat{\mathbf{S}}_{\mathbf{PP}}\mathbf{N})^{-1}\hat{\mathbf{S}}_{\mathbf{PV}}]^{\mathsf{T}} & \hat{\mathbf{S}}_{\mathbf{VV}} - \hat{\mathbf{S}}_{\mathbf{PV}}^{\mathsf{T}}\mathbf{N}(\mathbf{I} + \hat{\mathbf{S}}_{\mathbf{PP}}\mathbf{N})^{-1}\hat{\mathbf{S}}_{\mathbf{PV}} \end{bmatrix} \quad \text{(B1)}$$

can be derived by applying the usual formulas for inverting a matrix by partitioning [Mikhail 1976] to the two indicated inverses. However, a simpler proof of (B1) can be obtained by manipulating it according to the rules of matrix algebra until an obvious identity is obtained. First, by the definition of the matrix inverse, (B1) is equivalent to

$$\left(\begin{bmatrix} \mathbf{N} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} + \begin{bmatrix} \hat{\mathbf{S}}_{\mathbf{PP}} & \hat{\mathbf{S}}_{\mathbf{PV}} \\ \hat{\mathbf{S}}_{\mathbf{PV}}^{\mathsf{T}} & \hat{\mathbf{S}}_{\mathbf{VV}} \end{bmatrix}^{-1} \right) \times$$

$$\begin{bmatrix} (\mathbf{I} + \hat{\mathbf{S}}_{\mathbf{PP}}\mathbf{N})^{-1}\hat{\mathbf{S}}_{\mathbf{PP}} & (\mathbf{I} + \hat{\mathbf{S}}_{\mathbf{PP}}\mathbf{N})^{-1}\hat{\mathbf{S}}_{\mathbf{PV}} \\ [(\mathbf{I} + \hat{\mathbf{S}}_{\mathbf{PP}}\mathbf{N})^{-1}\hat{\mathbf{S}}_{\mathbf{PV}}]^{\mathsf{T}} & \hat{\mathbf{S}}_{\mathbf{VV}} - \hat{\mathbf{S}}_{\mathbf{PV}}^{\mathsf{T}}\mathbf{N}(\mathbf{I} + \hat{\mathbf{S}}_{\mathbf{PP}}\mathbf{N})^{-1}\hat{\mathbf{S}}_{\mathbf{PV}} \end{bmatrix}$$

$$= \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \quad \text{(B2)}$$

where the identity matrix is written in terms of 6×6 matrixes for uniformity. Premultiplying both sides of (B2) by $\hat{\mathbf{S}}$ (expressed in terms of 6×6 matrixes according to (1)) produces

$$\begin{bmatrix} \mathbf{I} + \hat{\mathbf{S}}_{\mathbf{PP}}\mathbf{N} & \mathbf{0} \\ \hat{\mathbf{S}}_{\mathbf{PV}}^{\mathsf{T}}\mathbf{N} & \mathbf{I} \end{bmatrix}$$

$$\times \begin{bmatrix} (\mathbf{I}+\hat{\mathbf{S}}_{\mathbf{PP}}\mathbf{N})^{-1}\hat{\mathbf{S}}_{\mathbf{PP}} & (\mathbf{I}+\hat{\mathbf{S}}_{\mathbf{PP}}\mathbf{N})^{-1}\hat{\mathbf{S}}_{\mathbf{PV}} \\ \hat{\mathbf{S}}_{\mathbf{PV}}{}^{\mathsf{T}}\,(\mathbf{I}+\hat{\mathbf{N}}\mathbf{S}_{\mathbf{PP}})^{-1} & \hat{\mathbf{S}}_{\mathbf{VV}} - \hat{\mathbf{S}}_{\mathbf{PV}}^{\mathsf{T}}\mathbf{N}(\mathbf{I}+\hat{\mathbf{S}}_{\mathbf{PP}}\mathbf{N})^{-1}\hat{\mathbf{S}}_{\mathbf{PV}} \end{bmatrix}$$

$$= \begin{bmatrix} \hat{\mathbf{S}}_{\mathbf{PP}} & \hat{\mathbf{S}}_{\mathbf{PV}} \\ \hat{\mathbf{S}}_{\mathbf{PV}}^{\mathsf{T}} & \hat{\mathbf{S}}_{\mathbf{VV}} \end{bmatrix} \quad \text{(B3)}$$

Expanding the product on the left side of (B3) produces

$$\begin{bmatrix} \hat{\mathbf{S}}_{\mathbf{PP}} & \hat{\mathbf{S}}_{\mathbf{PV}} \\ \hat{\mathbf{S}}_{\mathbf{PV}}^{\mathsf{T}}\mathbf{N}(\mathbf{I} + \hat{\mathbf{S}}_{\mathbf{PP}}\mathbf{N})^{-1}\hat{\mathbf{S}}_{\mathbf{PP}} + \hat{\mathbf{S}}_{\mathbf{PV}}^{\mathsf{T}}(\mathbf{I} + \mathbf{N}\hat{\mathbf{S}}_{\mathbf{PP}})^{-1} & \hat{\mathbf{S}}_{\mathbf{VV}} \end{bmatrix}$$

$$= \begin{bmatrix} \hat{\mathbf{S}}_{\mathbf{PP}} & \hat{\mathbf{S}}_{\mathbf{PV}} \\ \hat{\mathbf{S}}_{\mathbf{PV}}^{\mathsf{T}} & \hat{\mathbf{S}}_{\mathbf{VV}} \end{bmatrix} \quad \text{(B4)}$$

Finally, some matrix manipulation (bringing $\hat{\mathbf{S}}_{\mathbf{PP}}$ into $(\mathbf{I} + \hat{\mathbf{S}}_{\mathbf{PP}}\mathbf{N})^{-1}$ from the right and factoring it out to the left, factoring $(\mathbf{I} + \hat{\mathbf{S}}_{\mathbf{PP}}\mathbf{N})^{-1}$ out to the right from the entire expression, and canceling $\mathbf{I} + \hat{\mathbf{S}}_{\mathbf{PP}}\mathbf{N}$ and its inverse) shows that the lower-left element of the left side of (B4) reduces to $\hat{\mathbf{S}}_{\mathbf{PV}}^{\mathsf{T}}$. Thus (B4) reduces to an identity, and, since all of the steps performed above are reversible (if $\hat{\mathbf{S}}$ and $\hat{\mathbf{S}}_{\mathbf{PP}}$ are nonsingular), this verifies (B1).

Another way of proving (B1) is to derive it (actually, (65)–(67) derived from it and (62)) from the usual Kalman filter equations. If we consider $\mathbf{N}^{-1}\mathbf{C}$ to be the measurements (relative to the predicted values), with covariance matrix $\mathbf{N}^{-1}$, as mentioned in section 9 as one possibility, then the matrix of partial derivatives

of the observations relative to the state vector is $[\mathbf{I}\ \ \mathbf{0}]$ (in terms of the 6×6 identity and zero matrixes), since the observations are direct measurements of **P** (pose) but are independent of **V** (velocities). Therefore, the Kalman update equations [Maybeck 1979] in this case become the following:

$$\mathbf{K} = \hat{\mathbf{S}}[\mathbf{I}\ \ \mathbf{0}]^{\mathrm{T}}([\mathbf{I}\ \ \mathbf{0}]\hat{\mathbf{S}}[\mathbf{I}\ \ \mathbf{0}]^{\mathrm{T}} + \mathbf{N}^{-1})^{-1} \tag{B5}$$

$$\mathbf{S} = \hat{\mathbf{S}} - \mathbf{K}[\mathbf{I}\ \ \mathbf{0}]\hat{\mathbf{S}} \tag{B6}$$

$$\begin{bmatrix}\mathbf{P}\\ \mathbf{V}\end{bmatrix} = \begin{bmatrix}\hat{\mathbf{P}}\\ \hat{\mathbf{V}}\end{bmatrix} + \mathbf{K}\mathbf{N}^{-1}\mathbf{C} \tag{B7}$$

(**K** is the Kalman gain matrix.) Partitioning $\hat{\mathbf{S}}$ into 6×6 matrixes according to (1) and expanding the matrix products involving $[\mathbf{I}\ \ \mathbf{0}]$ simplifies (B5) and (B6) in terms of the amount of computing involved, since it eliminates multiplying by ones and zeros, to produce

$$\mathbf{K} = \begin{bmatrix}\hat{\mathbf{S}}_{\mathbf{PP}}\\ \hat{\mathbf{S}}_{\mathbf{PV}}^{\mathrm{T}}\end{bmatrix}(\hat{\mathbf{S}}_{\mathbf{PP}} + \mathbf{N}^{-1})^{-1} \tag{B8}$$

$$\begin{bmatrix}\mathbf{S}_{\mathbf{PP}} & \mathbf{S}_{\mathbf{PV}}\\ \mathbf{S}_{\mathbf{PV}}^{\mathrm{T}} & \mathbf{S}_{\mathbf{VV}}\end{bmatrix} = \begin{bmatrix}\hat{\mathbf{S}}_{\mathbf{PP}} & \hat{\mathbf{S}}_{\mathbf{PV}}\\ \hat{\mathbf{S}}_{\mathbf{PV}}^{\mathrm{T}} & \hat{\mathbf{S}}_{\mathbf{VV}}\end{bmatrix} - \mathbf{K}[\hat{\mathbf{S}}_{\mathbf{PP}}\ \ \hat{\mathbf{S}}_{\mathbf{PV}}] \tag{B9}$$

with (B7) unchanged. Then, a considerable amount of matrix manipulation can simplify (B8), (B9), and (B7) further into (65)–(69).

Acknowledgments

The research described here was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration. The programming for the original version of the tracker was done primarily by Eric Saund, with portions by Doug Varney and Bob Cunningham. Melinda Yin and Todd Litwin converted the tracker program to run on the MicroVAX. Todd Litwin made several recent improvements to the program, developed the manual acquisition software, and assisted in running the recent tracking experiments.

References

Ayache, N., and Faugeras, O.D. 1988. Building, registrating, and fusing noisy visual maps. *Intern. J. Robotics Res.* 7:45–65.

Brand, L. 1947. *Vector and Tensor Analysis*. Wiley: New York.

Broida, T.J., and Chellappa, R. 1986. Estimation of object motion parameters from noisy images. *IEEE Trans. Patt. Anal. Mach. Intell.* PAMI-8:90–99.

Corben, H.C., and Stehle, P. 1960. *Classical Mechanics* (2nd ed.). Wiley: New York.

Dickmanns, E.D., and Graefe, V. 1988a. Dynamic monocular machine vision. *Mach. Vis. App.* 1:223–240.

Dickmanns, E.D., and Graefe, V. 1988b. Applications of dynamic monocular machine vision. *Mach. Vis. App.* 1:241–261.

Eskenazi, R., and Wilf, J.M. 1979. Low-level processing for real-time image analysis. Jet Propulsion Laboratory, Pasadena, CA, JPL Publication 79-79.

Gennery, D.B. 1982. Tracking known three-dimensional objects. *Proc. AAAI 2nd Natl. Conf. Artif. Intell.*, Pittsburgh, PA, pp. 13–17.

Gennery, D.B. 1986. Stereo vision for the acquisition and tracking of moving three-dimensional objects. In *Techniques for 3-D Machine Perception* (A. Rosenfeld, ed). Elsevier: Amsterdam.

Gennery, D.B. 1990. Properties of a random-acceleration recursive filter. Jet Propulsion Laboratory, Pasadena, CA, JPL internal report D-8057.

Gennery, D.B. 1991. Camera calibration including lens distortion. Jet Propulsion Laboratory, Pasadena, CA, JPL internal report D-8580.

Gennery, D.B., Litwin, T., Wilcox, B., and Bon, B. 1987. Sensing and perception research for space telerobotics at JPL. *Proc. IEEE Intern. Conf. Robot. Autom.*, Raleigh. NC, pp. 311–317.

Gennery, D.B., and Wilcox, B. 1985. A pipelined processor for low-level vision. *Proc. IEEE Comput. Soc. Conf. Comput. Vision Patt. Recog.*, San Francisco, CA, pp. 608–613.

Gilbert, A.L., Giles, M.K., Flachs, G.M., Rogers, R.B., and U, Y.H. 1980. A real-time video tracking system. *IEEE Trans. Patt. Anal. Mach. Intell.* PAMI-2:47–56.

Goldstein, H. 1980. *Classical Mechanics* (2nd ed.). Addison-Wesley: Reading, MA.

Hedgley, D.R. 1982. A general solution to the hidden-line problem. Ames Research Center, Dryden Flight Research Facility, Edwards, CA, NASA Reference Publication 1085.

Kalman, R.E. 1960. A new approach to linear filtering and prediction problems. *Trans. ASME*, series D, *J. Basic Engin.* 82:35–45.

Martin, W.N., and Aggarwal, J.K. 1978. Dynamic scene analysis. *Comput. Graph. Image Process.* 7:356–374.

Maybeck, P.M. 1979. *Stochastic Models, Estimation, and Control*, vol. 1. Academic Press: New York.

Mikhail, E.M. (with contributions by F. Ackermann) 1976. *Observations and Least Squares*. Harper & Row: New York.

Nagel, H.-H. 1978. Analysis techniques for image sequences. *Proc. 4th Intern. Conf. Patt. Recog.*, Tokyo, pp. 186–211.

Pinkney, H.F.L. 1978. Theory and development of an on-line 30 Hz video photogrammetry system for real-time 3-dimensional control. *Proc. ISP Symp. Photogramm. Industry*, Stockholm, Sweden.

Roach, J.W., and Aggarwal, J.K. 1979. Computer tracking of objects moving in space. *IEEE Trans. Patt. Anal. Mach. Intell.* PAMI-1:127–135.

Saund, E., Gennery, D.B., and Cunningham, R.T. 1981. Visual tracking in stereo. *Joint Autom. Cont. Conf.*, sponsored by ASME, University of Virginia.

Verghese, V., and Dyer, C.R. 1988. Real-time, model-based tracking of three-dimensional objects. University of Wisconsin, Madison, WI, Computer Sciences Tech. Rept. #806.

Wilcox, B., Tso, K., Litwin, T., Hayati, S., and Bon, B. 1989. Autonomous sensor-based dual-arm satellite grappling. *Proc. NASA Conf. Space Telerobotics*, Pasadena, CA (JPL Publication 89-7), vol. III, pp. 307–316.

Wu, J.J., Rink, R.E., Caelli, T.M., and Gourishankar, V.G. 1989. Recovery of the 3-D location and motion of a rigid object through camera image (An extended Kalman filter approach). *Intern. J. Comput. Vis.* 2:373–394.

Wünsche, H.-J. 1986. Detection and control of mobile robot motion by real-time computer vision. In *Mobile Robots*. W.J. Wolfe and N. Marquina, eds., *Proc. SPIE*, 727, Cambridge, MA, pp. 100–109.

Yakimovsky, Y., and Cunningham, R.T. 1978. A system for extracting three-dimensional measurements from a stereo pair of TV cameras. *Comput. Graph. Image Process.* 7:195–210.

Young, G.-S.J., and Chellappa, R. 1990. 3-D motion estimation using a sequence of noisy stereo images: Models, estimation, and uniqueness results. *IEEE Trans. Patt. Anal. Mach. Intell.* PAMI-12: 735–759.