



University of Pennsylvania
ScholarlyCommons

Technical Reports (CIS)

Department of Computer & Information Science

January 1988

An Adaptive Tracking Algorithm for Robotics and Computer Vision Application

Reem Bassam Safadi
University of Pennsylvania

Follow this and additional works at: https://repository.upenn.edu/cis_reports

Recommended Citation

Reem Bassam Safadi, "An Adaptive Tracking Algorithm for Robotics and Computer Vision Application", . January 1988.

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-88-05.

This paper is posted at ScholarlyCommons. https://repository.upenn.edu/cis_reports/735
For more information, please contact repository@pobox.upenn.edu.

An Adaptive Tracking Algorithm for Robotics and Computer Vision Application

Abstract

We provided a vision-controlled robotics manipulation system with a robust, accurate algorithm to predict the translational motion of a 3-D object; hence, making it possible to continuously point the video camera at the moving object. The real time video images are fed to a PVM-1 (a pyramid-based image processor) for image processing and moving object detection. The measured object coordinates are continuously fed to our algorithm for track smoothing and prediction. In this study, we examined several tracking algorithms and adopted an optimal $\alpha - \beta$ filter for tracking purposes and the $\alpha - \beta - \gamma$ filter as part of the initialization procedure. The optimum gains for these filters are based on the Tracking Index principle which in its turn is based on the measurement noise variance and the object dynamics. We derived an expression for the noise variance corresponding to our application. As for the object dynamics, we developed an adaptive method (using the $\alpha - \beta - \gamma$ filter mentioned above) for inferring object dynamics in an iterative learning process that results in an accurate estimate of the Tracking Index. The accuracy of our algorithm realizes that of the Kalman filter but is much simpler computationally.

Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-88-05.

**AN ADAPTIVE TRACKING
ALGORITHM FOR ROBOTICS
AND COMPUTER VISION
APPLICATION**

Reem Bassam Safadi

**MS-CIS-88-05
GRASP LAB 132**

**Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104**

January 1988

Acknowledgements: This research was supported in part by DARPA grant N00014-85-K-0807, NSF-CER grant MCS-8219196, USPS 104230-87-H-0001/M-0195 and U.S. Army grants DAA29-84-K-0061, DAA29-84-9-0027. Any correspondence regarding this paper should be sent to Kwangyoen Wohn, GRASP LAB.

UNIVERSITY OF PENNSYLVANIA
THE MOORE SCHOOL OF ELECTRICAL ENGINEERING
SCHOOL OF ENGINEERING AND APPLIED SCIENCE

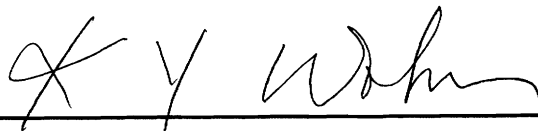
**AN ADAPTIVE TRACKING ALGORITHM FOR
ROBOTICS AND COMPUTER VISION
APPLICATIONS**

Reem Bassam Safadi

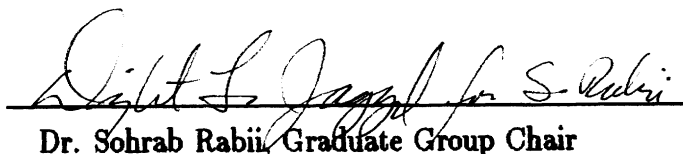
Philadelphia, Pennsylvania

December 1987

A thesis presented to the Faculty of Engineering and Applied Science of the University of Pennsylvania in partial fulfillment of the requirements for the degree of Master of Science in Engineering for graduate work in Electrical Engineering.



Dr. Kwangyoen Wohn, Supervisor



Dr. Sohrab Rabin, Graduate Group Chair

Table of Contents

Abstract	iii
Chapter 1	
Introduction and Background	1
Chapter 2	
Comparison of tracking Algorithms	6
Chapter 3	
General Implementation of the Elected Algorithm	17
Chapter 4	
Tailoring the Elected Algorithm to the PVM1-Based Robotics and Computer Vision System	27
Chapter 5	
Conclusion and Recommendations for Future Work	34
References	36
Appendix I	
Basic Equations of the Simulated Tracking Algorithms	39
Appendix II	
The Tracking Program	42

Abstract

We provided a vision-controlled robotics manipulation system with a robust, accurate algorithm to predict the translational motion of a 3-D object; hence, making it possible to continuously point the video camera at the moving object. The real time video images are fed to a PVM-1 (a pyramid-based image processor) for image processing and moving object detection. The measured object coordinates are continuously fed to our algorithm for track smoothing and prediction. In this study, we examined several tracking algorithms and adopted an optimal $\alpha - \beta$ filter for tracking purposes and the $\alpha - \beta - \gamma$ filter as part of the initialization procedure. The optimum gains for these filters are based on the Tracking Index principle which in its turn is based on the measurement noise variance and the object dynamics. We derived an expression for the noise variance corresponding to our application. As for the object dynamics, we developed an adaptive method (using the $\alpha - \beta - \gamma$ filter mentioned above) for inferring object dynamics in an iterative learning process that results in an accurate estimate of the Tracking Index. The accuracy of our algorithm realizes that of the Kalman filter but is much simpler computationally.

CHAPTER 1

Introduction and Background

A machine vision system is often considered as part of a larger system that interacts with the environment. The input to such a system, is an image, or several images, while its output is a description that must bear some relationship to what is being imaged and must contain all the needed information to carry out a designated task. In other words, the objective is not to obtain any description of what is imaged but one that allows us to take an appropriate action. Since building a "universal" vision system is still at the early stages of development, researchers have been addressing themselves to systems that perform a particular task in a controlled environment or to modules that could eventually constitute a general purpose system [1].

The work here is tied to one of those particular tasks, namely, real time analysis of images in a dynamic environment. Under this category falls the topic of motion analysis.

It is known that there are two major obstacles in computing the 3-D parameters of rigid motion from the retinal optical flow. One of these is the high dimensionality of the parameter space; the other is the nonlinearity of the constraint equations. It has been shown in [2] that these shortcomings may be eased by following two strategies. The first, and the most important of these, is to employ tracking, thereby exploiting the temporal behavior of a moving object. The second is to employ stereoscopic imaging. The results showed that tracking the object of interest is advantageous in both monocular and binocular imaging situations.

The use of a tracking filter is desirable under these circumstances in order to increase the accuracy of the image-based measurements of the position and orientation of a 3-D moving object. Our goal has been to find the most suitable tracking filter and tailor it to an existing vision-controlled robotics manipulation system that has the capability of performing object detection in real time. This capability is provided mainly by the Pyramid preprocessor [3], [4]. We will focus our attention here on tracking the translational motion of a 3-D rigid object. Tracking and prediction of this motion can be used to continuously point the video camera of the robotics system on the moving object. An extension to the problem of tracking the rotational motion of the rigid object is straightforward. The following sections give an overview of the Pyramid preprocessor and the existing moving-object detection algorithm.

1.1 The Pyramid Machine

The Pyramid Machine is a preprocessor dedicated to producing a multi-resolution representation of an image; specifically, the “pyramid like” representation. Pyramids in general, are data structures that provide successively condensed representations of the information in the input image. According to [3] there is evidence that the human visual system uses a form of multi-resolution representation which supports the concept that image processing at multi-resolution levels is very efficient and robust. The most obvious advantage of pyramid representations is that they provide a possibility for reducing the computational cost of various image operations. Many basic image operations may be performed efficiently within these pyramid structures. The first requirement for a multi-resolution image processing system is that it be able to perform a pyramid transform such as the FSD (filter,

subtract, and decimate transform), to decompose the original image into a set of different resolutions. The Pyramid machine is capable of constructing a complete low or bandpass pyramid from a 256×256 image in $1/30$ of a second (one frame time). The present design uses a single filter and decimate stage. During pyramid construction data is recirculated through this module for each pyramid level. A separable, five tap filter is used in the pyramid construction. The input data and output pyramid levels are represented with 8 bits per pixel, while internal computations are performed with 16 bit arithmetic.

When the system is used as a video preprocessor, it continuously processes incoming image data, transforming it to a more suitable representation for further processing such as eliminating certain spatial frequency bands or computing local energy measures at different scales, storing the results in a memory frame store. These results can be accessed simultaneously by the host computer for further processing. In this manner, the host processor (an IBM RT in our case) is relieved from performing extensive computations. For example, for a 256×256 pixel image, updated 30 times per second, the data rate is 2M pixels per second. By representing the video image in a suitable format, the host can perform real time or near real time image processing operations by selectively limiting the processing to 1000 to 20,000 pixels per second, rather than 2M pixels/second. This proved to be sufficient for performing numerous image processing tasks, e.g. real time motion detection and tracking, and course-fine pattern matching for robotics guidance [3].

1.2 The Existing Motion Detection Algorithm

The tracking filter (that provides this system with prediction capability) uses an existing motion detection program which detects object motion wherever it may occur within a large

field of view. The following is a brief description of how this is accomplished [4].

Let $I(T)$ be the image frame at time T . In the first step, a difference of two consecutive images is obtained $D(T) = I(T) - I(T - 1)$. Difference values that are not zero indicate that a change has occurred in the original sequence. In the second step, the difference image is decomposed into spatial frequency bands through the construction of a Laplacian pyramid (See Figure 1.1). A particular band pass level is then selected for further analysis. For example, $|L_0|$ could be chosen to generate a second series of low pass versions $G_n|L_0|$ where n ranges from 1 to 4.

G_4 which contains only 16×16 samples (and yet represents image changes corresponding to that within the original image) is examined by the host. (It is worthwhile mentioning that a typical high performance microcomputer (e.g., an IBM-RT) can examine only, on the order of, 16×16 samples per one frame interval [4].) If a change is detected in G_4 , the host examines a 16×16 subarray of G_3 centered on the change detected in G_4 . A detected change within the new subimage directs the host to examine a 16×16 subarray of G_2 . This procedure allows the system to rapidly home in on a small object anywhere within the field of view, since G_4 , which is a 16×16 array, contains the entire field of view of the 256×256 original image, but a 16×16 subarray located in G_1 covers only a small area of the original image [4].

1.3 An Adaptive Tracking Filter

We conducted a general study on the tracking filters and chose one of these filters for our application. Among the filters that were examined are: The Kalman, Wiener, $\alpha - \beta$ and $\alpha - \beta - \gamma$ filters, and the Two-Point Extrapolator. The criteria followed in our decision

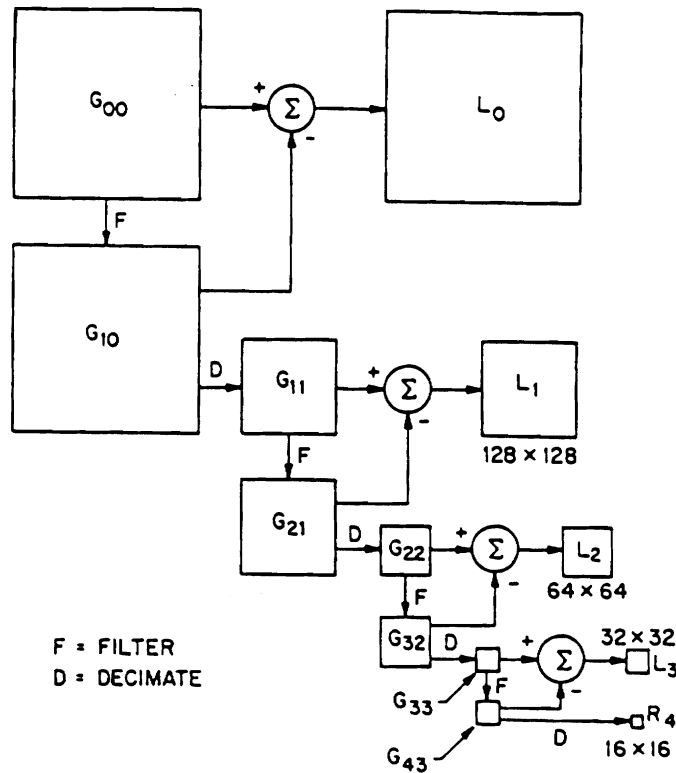


Figure 1.1 An FSD Pyramid Transform (from [3]).

G_{00} : The original image, represented by 256×256 array of pixels

$G_{n:n}$: The low pass result where the first index indicates the number of filter steps, the second index indicates the decimation steps

F : A convolution with an FIR low pass filter

D : Decimate operation, every other row and column is discarded to obtain G_{n+1}

L_n : Bandpass or Laplace pyramid level obtained by subtracting two consecutive Gaussian pyramid levels as shown

making will be discussed in the next chapter.

We adopted an optimal $\alpha - \beta$ filter for tracking purposes and the $\alpha - \beta - \gamma$ filter as part of the initialization procedure. The powerful performance of this filter (which realizes that of the Kalman) lies in the method that produces the filter gain values. The derivation of the optimum filter gains is based on the Tracking Index which is proportional to the position uncertainty due to object accelerations to that due to measurement errors (in the detection scheme) [5].

We derived an expression for the noise variance corresponding to our application. As for the object dynamics, we developed an adaptive method (using the $\alpha - \beta - \gamma$ filter mentioned previously) for inferring object dynamics in an iterative learning process that results in an accurate estimate of the Tracking Index.

The work presented here was developed on an IBM AT; and the results were tested using the position data of the moving object obtained from files produced by the Pyramid/IBM-RT system. The following chapters give a more detailed account of our work.

CHAPTER 2

Comparison of Tracking Algorithms

As we observe the position of a moving object over time using any kind of measuring instrument (a video camera, radar, etc.), almost always, the observations are cluttered by noise, errors and inaccuracies. The primary function of the tracking filter is to accept the noisy position data at its input and provide smoothed object position and velocity estimates at its output. These values are used for controlling the orientation and position of the camera as well as predicting future object positions. In addition, the smoothed position and velocity estimates can be used for track correlation and association purposes in a multi-moving-object environment.

In smoothing and prediction, two sets of equations govern the whole technique. The first set of equations is the differential equations which describe the process (object motion). The second set relates the parameters being measured to those to be estimated [6]. In addition, smoothing and prediction are related in a recursive manner, i.e. a predicted value depends on the last smoothed value, and a smoothed value takes into account the last predicted value. As an example, the prediction and smoothing relationships for an optimal mean-square-error (MSE) estimation process (Kalman filter) for object tracking is given by [5],[7]:

$$\text{prediction: } \hat{x}(k+1|k) = \phi \hat{x}(k|k)$$

$$\text{smoothing: } \hat{x}(k+1|k+1) = \hat{x}(k+1|k) + K(k+1)[z(k+1) - h\hat{x}(k+1|k)]$$

where,

$k = k$ th time interval

$\hat{x}(k+1|k) =$ predicted state value at time $k+1$

$\hat{x}(k|k) =$ smoothed state value estimate at time k

$\hat{x}(k+1|k+1) =$ smoothed state value estimate at time $k+1$

$z(k+1) =$ measured state value at time $k+1$

$K(k+1) =$ gain value at time $k+1$

$\phi =$ state transition matrix $h = 1$ when x represents position, 0 when x represents velocity or acceleration.

A state could be position, velocity, or acceleration. Here, the observation or measurement is modeled by the actual object position plus an additive noise component:

$$z(k) = x(k) + n(k)$$

where the measurement noise uncertainty $n(k)$ is assumed to be a zero-mean, white stationary random process.

The problem then is narrowed down to obtaining the “best” estimate of $\hat{x}(k|k)$. There are two solutions for extracting the “best” smoothing estimate. The first, is the Least Squares method, the other is the Minimum Variance method. Generally, the Least Squares method (the one under consideration) takes a fitting function and fits it to the data. The method calculates the residuals (differences between the observations and the fitting function), squares them, adds them, and produces a certain value. The Least Squares method

in its most general form, then minimizes that value, for any arbitrary number of dimensions. The Minimum Variance method is similar to least squares, but is more generalized. Minimum Variance takes every observation and weights each precisely according to its merits. For example, poor observations have a large variance while better observations have a smaller variance. Typically, observations are weighted by the inverse of their variance, hence poor observations are suppressed, while good observations are boosted relatively [6].

2.1 Comparison of Tracking Algorithms

A comparative study of five important real-time tracking filters was conducted in [8]. It compared these filters in tracking accuracy and computer requirements (memory and execution time).

The filters considered in this study were: the Kalman filter, simplified Kalman filter, $\alpha - \beta$ filter, Wiener filter, and the two point extrapolator. The gain vectors of the first three filters were assumed to be calculated in real time. The last two filters (Wiener and two point extrapolator) were both examples of stored gain vectors. The following is a brief overview of each filter.

In the Kalman filter, a model for measurement error has to be assumed as well as a model of the object trajectory and the disturbance of the trajectory [8]. The Kalman filter can in principle, utilize a wide variety of models for measurement noise and trajectory disturbance; however, it is often assumed that these are described by white noise with zero mean [9]. This becomes a requirement for the filter optimality and introduces the need to have two augmented state variables in order to whiten the object maneuver and adapt it to the theoretical framework necessary to make the filter optimal [8].

If the maneuver is assumed to be white, no augmentation need be performed (resulting in what is known as a simplified Kalman filter). Here it is assumed that the change in velocity of the object is uncorrelated between samples, i.e. white. Furthermore, if this filter were restricted to modeling the object path as a straight line and if the measurement noise and maneuvering noise were modeled as white gaussian with zero mean, the Kalman filter equations reduce to the Alpha-Beta filter equations with the parameter alpha and beta computed sequentially by the Kalman filter procedure [9].

The Wiener filter differs from the Kalman filter in that its gain vector being equal to the steady state gain vector of the regular Kalman filter and is calculated off-line and stored in memory. This results in considerable computational savings in addition to making it simpler than any of the preceding filters. Because it has constant gain, the Wiener filter requires no auxiliary equations to be solved and requires very little computer memory. This filter is adaptable to a variety of moving objects and can track both maneuvering and nonmaneuvering moving objects well. This is because its gain is derived from the Kalman filter, which accounts for the statistics of object maneuver directly [8].

The α - β filter comes in many different varieties. Some are designed to provide the best transient following capability for a constant velocity object, while simultaneously providing the best minimum variance estimate of position and velocity [10]. Other α - β filters that have been designed utilize the object maneuver statistics [11]. If the performance criterion is dynamic minimization of the total mean-squared filtering errors, the filter then takes the form of the Kalman filter. If the design objective is to minimize the tracking errors against a general class of trajectories, the corresponding α - β filter takes even a different form [8]. The α - β filter considered in the simulation study [8] was designed to minimize the mean-squared error in filtered (smoothed) position and velocity, under the assumption of a

constant velocity object motion.

All of the mentioned filters are recursive fading memory filters. The simplest type of filters that can be implemented is the "almost memoryless" two point extrapolator. This filter uses the last data point to determine object position and the last two data points to determine object velocity. Because this filter is essentially memoryless, its performance in tracking maneuvering and nonmaneuvering objects is quite as bad [8].

The result of the simulation study conducted in [8], showed that the most sophisticated filter, the Kalman filter, is the most accurate and the most costly to implement. Furthermore, the Kalman filter, the simplified Kalman filter, and the Wiener filter generally performed within 20 percent of each other (in terms of execution time and memory requirements). The $\alpha - \beta$ filter performed on the average about 50 percent worse than the Kalman filter with the greatest degradation occurring for maneuvering objects. The two point extrapolator, uniformly performed more than 70 percent worse than the Kalman filter.

As for implementation requirements, they increased in the following order: two point extrapolator, Wiener filter, $\alpha - \beta$ filter, simplified Kalman filter, Kalman filter. The complexity factor between successive filters was about two to one.

An attractive filter would be one that can achieve the performance of a Kalman filter, both in the transient and steady state periods; yet be easily implementable in real time. Also, it will be very desirable for such a filter to have the capability of tuning itself to the sensor and moving object characteristics.

In addition to the optimal transient and steady state performance, we would like to have a simple filter that can achieve this performance regardless of object dynamics. It is undesirable to have solutions (for filter gains) that entail recursive relationships of considerable

complexity; it would be ideal to obtain a closed form solution and eliminate the recursive process altogether at least for the steady state (even if it were simple as in [12]), without any degradation in performance. Nor would it be desirable to have a simple closed form solution that can only provide optimal performance for a particularized object motion behavior (as in [11] where it is assumed that object accelerations are exponentially correlated, and [10] where the results apply for constant velocity objects only). This means that in order to reduce the implementation cost, it is desirable to have constant gain vectors. This will result in considerable savings over filters that calculate the gain vectors in real time.

From the results of the simulation study, a probable filter that fits most of the above descriptions would be the Wiener filter. However, the performance of this filter in the transient state is not optimal, because its off-line computed vector is the Kalman steady state gain vector. Using the steady state gain in the transient state will give erroneous predictions.

2.2 The Elected Filter

A recent study [5] provided optimum parameters for the α - β filter which results in optimum tracking in both the transient and the steady states. With these parameters, the α - β filter achieves the performance of the Kalman filter without increasing its complexity or implementation cost.

The above mentioned study introduced an optimal filtering solution for the object tracking problem which depends on a parameter defined as the "Tracking Index" which is proportional to the position uncertainty due to object accelerations to that due to measurement errors. Upon evaluating this parameter, the optimal transient and steady state gains are

specified. The filtering process is as follows [6]:

Prediction:

$$x(k+1|k) = x(k|k) + Tv(k|k)$$

$$v(k+1|k) = v(k|k)$$

Smoothing:

$$x(k+1|k+1) = x(k+1|k) + \alpha[z(k+1) - x(k+1|k)]$$

$$v(k+1|k+1) = v(k+1|k) + \frac{\beta}{T}[z(k+1) - x(k+1|k)]$$

where,

$x(k|k)$ is the position estimate at time interval k

$v(k|k)$ is the velocity estimate at time interval k

$z(k)$ is the noisy position measurement

α is the position tracking parameter

β is the velocity gain tracking parameter.

T is the sampling period

In modeling the object motion, a one dimensional, linear, time invariant, ideal model is used:

$$s(k+1) = \phi s(k) + \psi w(k)$$

where $s(k)$ is the moving object state vector at time k ; ϕ is the state transition matrix; $w(k)$ is the unknown object maneuver/state transition matrix. ψ is the acceleration state

transition matrix (for a second order model $\psi = [\frac{1}{2}T^2 \quad T]^T$; for a third order model $\psi = [\frac{1}{2}T^2 \quad T \quad 1]^T$). A two state model which includes all possible maneuver accelerations [5] is sufficient and will be considered hereon; that is

$$s(k) = [x(k) \quad v(k)]^T$$

The model in our application is based on the assumption that, without maneuvering (or with smooth maneuvering in a small time interval), the moving object will generally follow a straight line constant velocity trajectory.

The noisy object measurements are modeled by the actual position of the moving object plus an additive noise component, i.e:

$$z(k) = hs(k) + n(k)$$

where for a two state model

$$h = [1 \quad 0]$$

and the measurement noise uncertainty is assumed to be a zero mean white stationary process. The values of α and β are determined by the following parameters:

T: The sampling period.

σ_n : The measurement noise standard deviation which is determined from the object detection scheme, i.e. the measurement process.

σ_a : The maneuvering accelerations standard deviation. This parameter is related to the object dynamics.

A typical maneuvering accelerations probability density [7] is shown in Fig. 2.1 In this figure, A denotes the maximum acceleration which the object can have. Values of the density between no maneuver ($a = 0$) and maximum maneuver ($a = \pm A$) are non zero because the moving object may not be accelerating at the maximum rate. The object has a probability P_1 of accelerating at this maximum level (either plus or minus), a probability P_2 of not accelerating at all, and an assumed uniform probability distribution of amplitude $(1 - (2P_1 + P_2))/2A$ of accelerating between $-A$ and $+A$. The acceleration variable, therefore has zero mean and variance $(A/3)(1 + 4P_1 - P_2)$.

The tracking index is provided below in terms of the above parameters and also in terms of the steady state gains α^* and β^* ; along with the corresponding optimal β^* relationship. As for the optimal α^* relationship, it had to be derived and will be given in the following chapter.

$$\Lambda \equiv T^2 \sigma_a / \sigma_n$$

$$\Lambda = \frac{\beta^{*2}}{1 - \alpha^*}$$

$$\beta^* = 2(2 - \alpha^*) - 4\sqrt{1 - \alpha^*}$$

The optimum transient filter gain parameters are based on following recursive approximations:

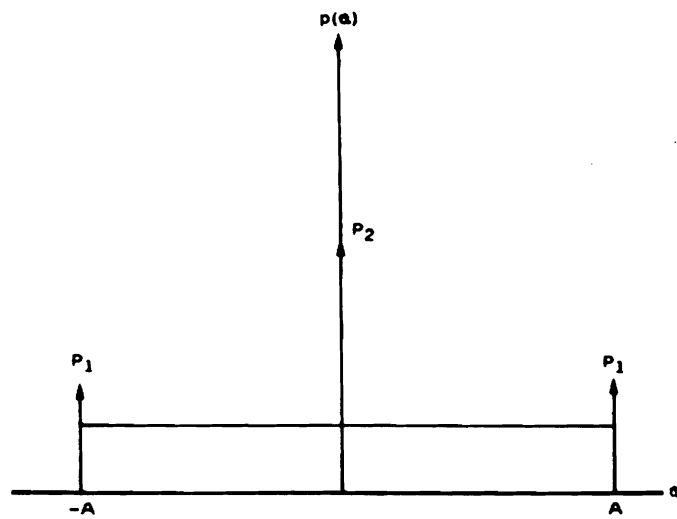


Figure 2.1 Typical Probability Density of Object Acceleration.

$$\alpha(k) = \alpha(k-1) + G_\alpha[\alpha^* - \alpha(k-1)]; \quad G_\alpha = 1 - e^{-1/k_\alpha}$$

$$\beta(k) = \beta(k-1) + G_\beta[\beta^* - \beta(k-1)]; \quad G_\beta = 1 - e^{-1/k_\beta}$$

where,

$\alpha(k)$ and $\beta(k)$ are the gains for the present state

$\alpha(k-1)$ and $\beta(k-1)$ are the gains for the previous state

α^* and β^* are the steady state optimal gain values, as defined on the previous page.

K_α , and K_β are the first order time constants of the corresponding gain excursion from its initial to its optimal steady state value. It has been shown in [5] that these time constants are strictly a function of the tracking index.

However, [5] does not provide an expression for the relationship between the tracking index and these time constants. It does, on the other hand, provide the graphs for the normalized time constants vs $\log(\Lambda^2)$, as shown in Fig. 3.

Finally, it is worthwhile to note that for small α , the $\alpha - \beta$ relationship mentioned previously approaches the classical relationship [5]:

$$\beta = \alpha^2 / (2 - \alpha)$$

When the filter parameters are adjusted according to the above equation, the filter provides the best transient following capability for a constant velocity object, while simultaneously providing the best minimum variance estimate of position and velocity of any

fixed parameter filter [10]. The following chapter will discuss implementation details of the elected filter.

CHAPTER 3

General Implementation of the Elected Algorithm

The α - β , α - β - γ filters, and the Two Point extrapolator have been implemented in the simulation program. The equations for each filter are provided in Appendix I.

These three filters were implemented as an illustrative tool during our study. Then the α - β filter was chosen to perform the tracking, while an α - β - γ filter was implemented in the learning procedure devised for inferring the dynamics of the moving object and hence optimizing the α - β filter used for the actual tracking.

The following sections will consider both filters since both are used by the simulation program.

3.1 Filters Initialization

3.1.1 Gains Initialization

The tracking index, Λ , plays a major role in determining the transient and the steady state gains of the α - β and α - β - γ filters. It is a dimensionless parameter proportional to the ratio of the position uncertainty due to object accelerations (maneuvers) and to that due to detector measurement errors [5], i.e.

$$\text{tracking index} \propto \frac{\text{position uncertainty due to acceleration}}{\text{position measurement uncertainty}}$$

It is defined by three primary object tracking modeling parameters: track period (time between position samples), object maneuverability (characterized by object accelerations), and measurement noise [5]:

$$\Lambda \equiv \frac{T^2 \sigma_a}{\sigma_n}$$

T , the track period always has a known value. As for determining σ_a and σ_n , one would need to have an a priori knowledge of the detection mechanism and the object dynamics respectively. Obtaining the values for these parameters will be discussed later on, but for now we will assume that these values are available to us and hence we have a value for the tracking index Λ . Once Λ is known, we are able to calculate the filter transient and steady state gains.

Initially, the filters gains are assigned to the following values:

For the α - β filter:

$$\alpha(0) = 1$$

$$\beta(0) = 1$$

For the α - β - γ filter:

$$\alpha(0) = 1$$

$$\beta(0) = \frac{3}{2}$$

$$\gamma(0) = 2$$

These values were obtained from the initiation of the smoothing process (shown in the following section).

The program then calculates the transient gains at each sampling interval until the optimum steady state value, corresponding to that gain, has been reached.

The transient gains for the gain parameters α , β , and γ are based on the recursive approximations that were given in Sec. 2.2 and are repeated here for convenience (with the addition of the third state parameters, i.e. acceleration):

$$\alpha(k) = \alpha(k-1) + G_\alpha[\alpha^* - \alpha(k-1)]; \quad G_\alpha = 1 - e^{-1/k_\alpha}$$

$$\beta(k) = \beta(k-1) + G_\beta[\beta^* - \beta(k-1)]; \quad G_\beta = 1 - e^{-1/k_\beta}$$

$$\gamma(k) = \gamma(k-1) + G_\gamma[\gamma^* - \gamma(k-1)]; \quad G_\gamma = 1 - e^{-1/k_\gamma}$$

where,

$\alpha(k)$, $\beta(k)$, and $\gamma(k)$ are the gains for the present state

$\alpha(k-1)$, $\beta(k-1)$, and γ are the gains for the previous state

α^* , β^* , and γ^* are the steady state optimal gain values.

k_α , k_β , and k_γ are the first order time constants of the corresponding gain excursion

from its initial to its optimal steady state value. It has been shown in [5] that these time constants are strictly a function of the tracking index. However, [5] does not provide an expression for the relationship between the tracking index and these time constants. It does, on the other hand, provide the graphs for the normalized time constants versus $\log(\Lambda)$, as shown in Figure 3.1. (The normalized time constant values were numerically approximated by using the Kalman filtering process).

In order to make these graphical relationships available to the program, one could divide the tracking index range into subranges and provide the normalized time constant values for each subrange endpoints (nodes) and estimate the values within the subranges, through interpolation. This is required because the program will need these time constant values to calculate the gains, and it needs to do so for any tracking index value.

The most suitable interpolation method for this purpose is that of Cubic Splines [13]. This is one of the most common piecewise polynomial approximations. It uses cubic polynomials to interpolate between each successive pair of nodes. There are several reasons for choosing this type of interpolation, among them:

- (1) The oscillatory nature of high degree polynomials used to approximate an arbitrary function on a closed interval, and the fact that a function over a small portion of the interval can include large fluctuations over the entire range, restrict their use when a smoother approximation is desired.
- (2) With a simple piecewise linear interpolation, there is no assurance of differentiability at each of the subinterval endpoints, i.e., the interpolating function will not be smooth at these points.

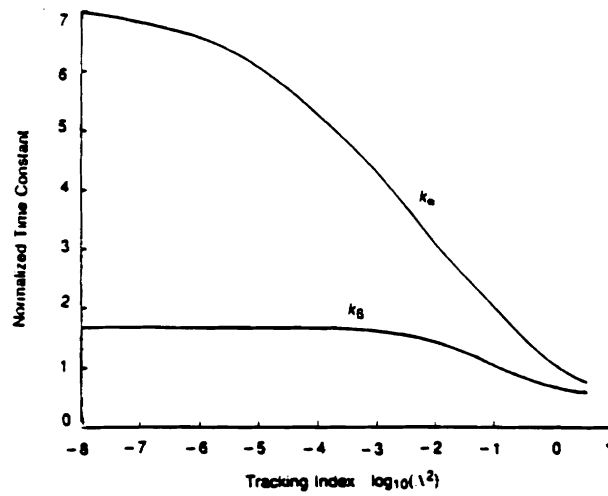


Figure 3.1a First-Order Time Constants for α - β Filter (from [5]).

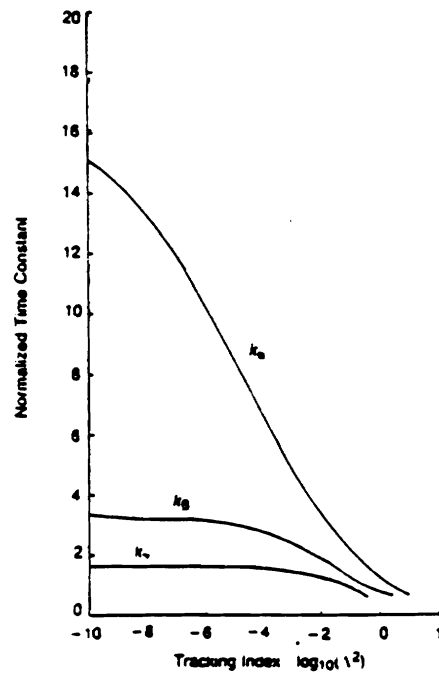


Figure 3.1b First-Order Time Constants for α - β - γ Filter (from [5]).

(3) Since the cubic splines interpolation uses a cubic polynomial at the subintervals, it will not only insure that the interpolant is continuously differentiable, but also that it has a continuous second derivative on that interval.

A final note on Cubic Splines is that either one of the following set of boundary conditions should be satisfied:

- (1) Free boundary: The second derivative at the interval end points are zero.
- (2) Clamped boundary: The first derivative of the interpolant at these end points are equal to the function first derivative at these points.

For simplicity, the free boundary condition has been used in the program.

Now, we describe the process of obtaining the optimum steady state gains. For the α - β filter, the relationship between α and the tracking index had to be derived from the following:

$$\Lambda^2 = \frac{\beta^2}{1-\alpha}, \quad \beta^2 = 2(2-\alpha) - 4\sqrt{1-\alpha}$$

The derivation gives the following result (by eliminating β):

$$\alpha^2 + \frac{\Lambda}{4}(\Lambda + 8)\alpha - \frac{\Lambda}{4}(\Lambda + 8) = 0$$

which has the following solution:

$$\alpha = -\frac{(\Lambda + 8)}{2} + \frac{1}{2}\sqrt{\frac{(\Lambda + 8)^2\Lambda^2}{16} + \Lambda(\Lambda + 8)}$$

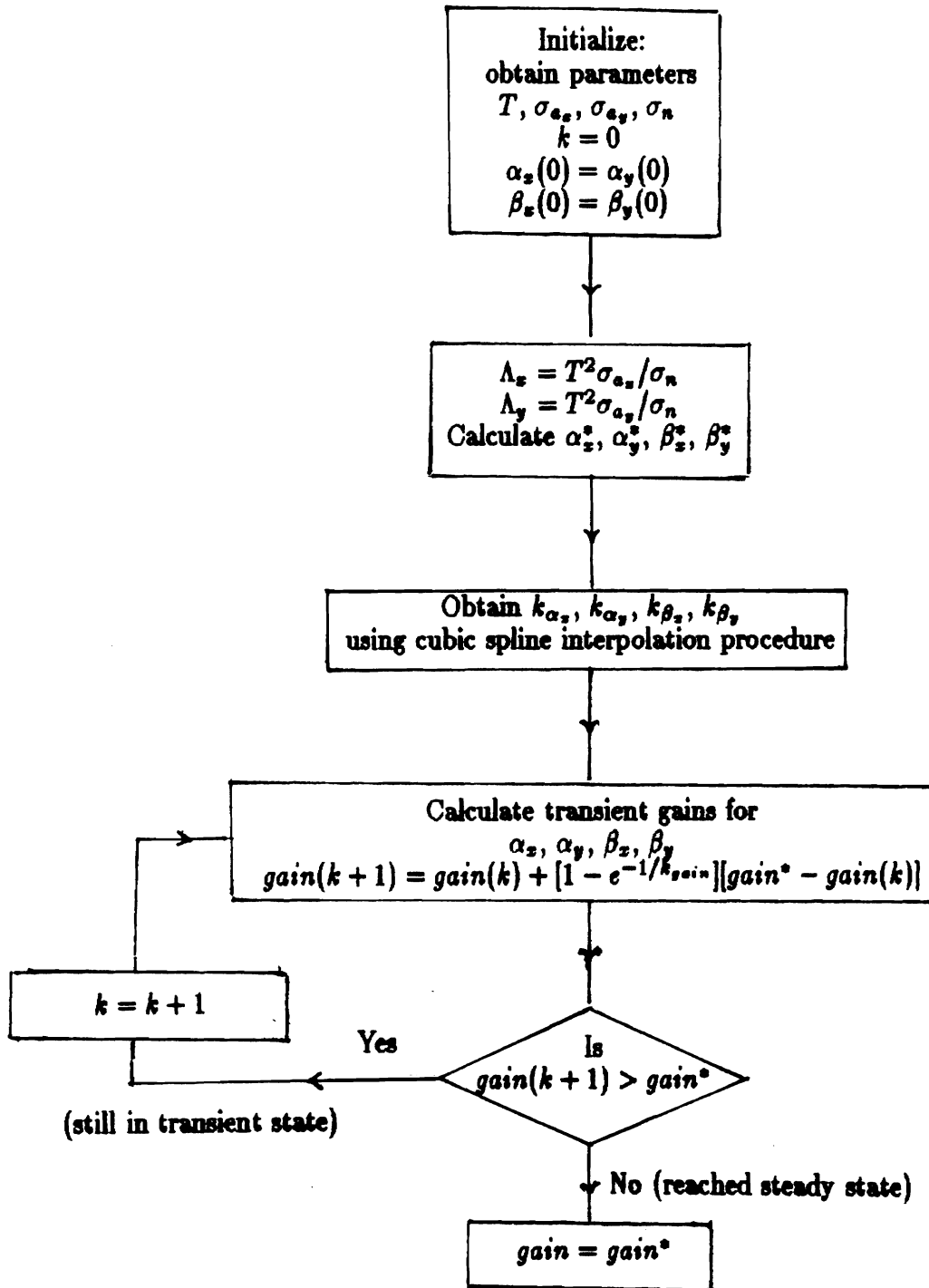


Figure 3.2 The α - β Filter Gains Calculation Procedure.

The second solution will give $\alpha < 0$; therefore, it is ignored.

As for the α - β - γ filter, the relationship between α and the tracking index was derived form:

$$\Lambda = \frac{\gamma^2}{4(1-\alpha)}, \quad \beta = 2(2-\alpha) - 4\sqrt{1-\alpha}, \quad \gamma = \frac{\beta^2}{\alpha}$$

The derivation gives the following result (by eliminating β and γ):

$$\alpha^3 + \left[\frac{\Lambda}{4}(\Lambda - 16)\right]\alpha^2 + \left[\frac{\Lambda}{4}(48 - \Lambda)\right]\alpha - 8\Lambda = 0$$

We solve the above equation numerically for α using the Newton Raphson method [13] which gives the positive real root that we are seeking (which lies between 0 and 1.)

Fig. 3.2 shows a flow diagram for obtaining the filters gains.

3.1.2 Initiation of the Smoothing Process

Each filter uses a different initial value for the smoothed estimates; the procedure for obtaining these values was shown in [5] to be MSE optimal.

The α - β filter uses a two-point measurement process for the initial position and velocity estimates:

$$\begin{aligned} x(0|0) &= z(0) \\ v(0|0) &= \frac{1}{T}[z(0) - z(-1)] \end{aligned}$$

The α - β - γ filter uses a three-point measurement initiation process:

$$\begin{aligned}x(0|0) &= z(0) \\v(0|0) &= \frac{1}{T} \left[\frac{3}{2}z(0) - 2z(-1) + \frac{1}{2}z(-2) \right] \\a(0|0) &= \frac{1}{T^2} [z(0) - 2z(-1) + z(-2)]\end{aligned}$$

where $z(i)$ is the measured position at the i th sampling period.

3.2 Case Study Simulation model

A case study was used to aid us in the understanding of the filters behavior. It also aided us in testing our developed techniques as in the case of the learning process, we developed, to be used for determining σ_a of the moving object. When we chose the simulation model for this purpose, we had two objectives in mind. The first was to have a model that tests the robustness of each filter; the second was to have a model where the primary tracking modeling parameters could be calculated in closed forms. The latter, helped us in performing the simulation under a controlled environment which is an important requirement for testing the filters and developing other techniques.

The chosen simulation model consists of sinusoidal oscillations both in the X and Y directions. The frequency of the Y oscillations was chosen to be an integral multiple of that of the X direction. Therefore, the motion in the X - Y plane will be cyclic (forming Lissajous patterns) with sinusoidal velocity and acceleration in each direction. This model will test all three filters for maneuver following capability and prediction accuracy. Given below are the model equations:

$$x(t) = \hat{x} \sin(\omega t)$$

$$y(t) = \hat{y} \cos(m\omega t) \quad m = 1, 2, \dots$$

where \hat{x} and \hat{y} are the amplitude factors for $x(t)$ and $y(t)$ respectively.

An approximation to the maneuvering standard deviation is given by [15]:

$$\sigma_{a_x} = \frac{1}{\sqrt{2}} |a_x(t)|_{max} = \frac{1}{\sqrt{2}} \omega^2 \hat{x}$$

$$\sigma_{a_y} = \frac{1}{\sqrt{2}} |a_y(t)|_{max} = \frac{1}{\sqrt{2}} m^2 \omega^2 \hat{y}$$

i.e., it is $\frac{1}{\sqrt{2}}$ of the maximum acceleration value. This is to be expected since the probability density function takes the form shown in Figure 3.3.

As for the sampling interval T , it could range from 20 to 50 samples of a full cycle of the fastest sinusoid, i.e.

$$T = \frac{1}{N_s} T_{p_s} = \frac{1}{N_s} \frac{2\pi}{m\omega_0}, \quad 20 < N_s < 50$$

The measurement noise which is assumed to be white gaussian, has a standard deviation equal to a small fraction ($\frac{1}{200} - \frac{1}{50}$) of the maximum sinusoidal amplitude, i.e.

$$\sigma_{n_x} = \frac{1}{N_n} \hat{x}, \quad \sigma_{n_y} = \frac{1}{N_n} \hat{y}, \quad 50 < N_n < 200$$

Substituting the above values of T , σ_a , and σ_n for the tracking index:

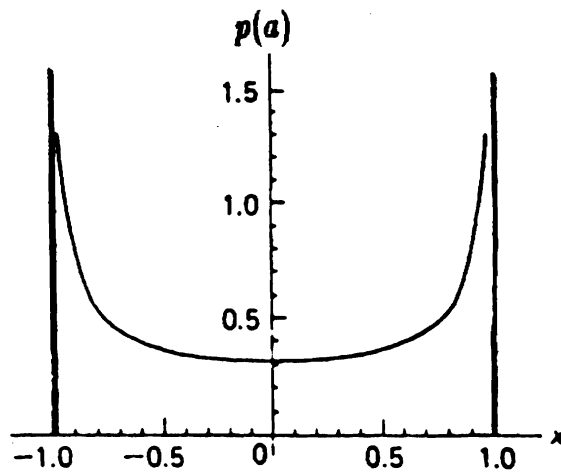


Figure 3.3 Probability Density Function of the Acceleration in the Simulation Model (where A is the maximum acceleration; from [15]).

$$\Lambda_x = \frac{1}{\sqrt{2}} \left(\frac{2\pi}{mN_s} \right)^2 N_n, \quad \Lambda_y = \frac{1}{\sqrt{2}} \left(\frac{2\pi}{N_s} \right)^2 N_n$$

3.3 Simulation Results

To test which of the three filters has the best performance, we calculated the standard deviation of the difference between the predicted and model values. Table 3.1 shows these standard deviations for some of the runs. As expected, for this continuous acceleration model, the α - β - γ filter had the error smallest standard deviation.

Except when the tracking index is relatively high, the program starts the filtering process by implementing the transient gains. Had not this been done, the steady state value of α (corresponding to a moderate-to-low tracking index), which in this case could be excessively small, would have been used to perform the tracking during the transient state. The latter would make it difficult for the filter to catch-up with the next measured position. The transient gains in the initial state proved to be very useful since they provide larger gain values (and then fall down to the optimum steady state values.) This means that initially, it is more important to catch-up with the next position rather than to filter out the noise. With each predicted position, the filter gains are further reduced (to allow for more smoothing of the measured position and to get a better accuracy) until the steady state value, corresponding to each gain, has been reached.

It was mentioned earlier that in our application, the moving object will not perform severe maneuvers; thus, the superior maneuver following capability of the α - β - γ filter will be somewhat wasted. This was one of the deciding factors for choosing the α - β filter over the α - β - γ filter. In addition, it was shown in [5] that the accuracy of the prediction/smoothing

N_n	N_s	Two Point Extrapolator		α - β Filter		α - β - γ Filter	
		σ_x	σ_y	σ_x	σ_y	σ_x	σ_y
200	20	6.66,	9.06	2.44,	3.51	1.77,	2.33
	30	3.93,	4.72	1.65,	2.05	1.44,	1.49
	40	2.77,	3.01	1.34,	1.49	1.29,	1.40
100	20	6.84,	9.08	3.72,	4.63	3.56,	2.91
	30	4.98,	5.05	2.72,	2.85	2.57,	2.24
	40	3.45,	3.45	2.50,	2.19	2.12,	1.91
50	20	9.62,	9.52	5.13,	6.13	5.37,	4.85
	30	7.29,	5.72	4.36,	4.15	3.51,	3.75
	40	6.30,	4.19	3.42,	3.23	3.32,	2.51

Table 3.1 Error Standard deviations - predicted position values form model position values.

process improves as α decreases. Since for a given Λ , the α - β filter provides a smaller optimum α than the one provided by the α - β - γ filter (for the same Λ - as can be seen from Fig. 3.4), we preferred to use the α - β filter.

In the next chapter we will discuss two major remaining points. The first is how we provided the α - β filter with an adaptive learning capability. The second, is how this filter was tailored for our application.

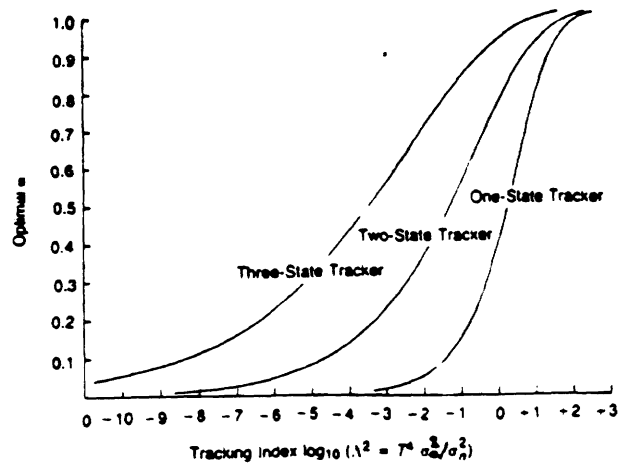


Figure 3.4 Optimal Position Tracking Gain α Versus Tracking Index Λ (from [5]).

CHAPTER 4

Tailoring the Elected Algorithm to the PVM-1 Based Robotics and Computer Vision System

In the previous chapter we assumed that the three fundamental tracking modeling parameters σ_a , σ_n , and T were available to us and we deferred the discussion on how we obtain these parameters to this chapter.

In order to make our filter practical to use, it would be necessary for us to provide the filter with these parameters. Determining the first parameter, σ_a , depends entirely on the moving object dynamics; the second parameter, σ_n , depends on the method that generates the object position values. The third parameter, T , is simply the sampling period, i.e. the time between successive position samples. This parameter is always known and in our case it is the reciprocal of the frame rate (i.e., $\frac{1}{30}$ sec).

The following sections will explain the procedures for obtaining the two parameters σ_a and σ_n .

4.1 Determining σ_a

σ_a is a quantity that reflects the uncertainty of the moving object position due to its accelerations. The degree of accelerations may vary from one moving object to another. How do we determine σ_a for any moving object that might be of interest to us? The idea of a "learning" tracker evolved when we tried to answer this question. This means that

prior to performing any tracking, an approximation to σ_a will be calculated off-line using position data obtained as a typical test run. Then the α - β filter will use this value for future tracking.

We begin our explanation of the process by introducing the underlying basic concepts behind our method:

Let a be a random variable that could take any value in the range $\pm A$; where A is the maximum acceleration that the moving object undergoes. Let us assume that we have N consecutive position samples of the moving object. We can calculate the acceleration at each point using the two preceding position points in addition to the current point, i.e.

$$a_i = \frac{1}{T^2}[x(i) - 2x(i-1) + x(i-2)]$$

where

$$3 \leq i \leq N$$

T is the sampling period

$x(i)$ is the position value at time interval i

Note that our notation $x(i)$ represents a general position point value, not necessarily the X -coordinate; i.e. $x(i)$ implicitly indicates X or Y . σ_a is derived for both directions X and Y ; each has its corresponding σ_a , i.e. σ_{a_x} and σ_{a_y} . We will use the notation σ_a to represent either direction.

The variance for this random variable is given by:

$$\sigma_a^2 = E[a_i^2] - E[a_i]^2$$

which is estimated as:

$$\sigma_a^2 \approx \frac{1}{N-2} \sum_{i=3}^N a_i^2 - \left(\frac{1}{N-2} \sum_{i=3}^N a_i \right)^2$$

The resulting σ_a value would have been used to calculate the tracking index for the tracking filter had it not been derived from the a_i values, because the a_i values were derived from the unreliable noisy object position data. However this value could be used as an initial approximation to the actual σ_a . We can obtain a better approximation if we use a filtered version of the noisy data points. Here, the α - β - γ filter comes to play.

The α - β - γ filter is run iteratively with each iteration using the smoothed position values that the filter calculated during the previous iteration. That is, at each iteration, the filter produces the object position values had there been a lesser amount of noise present in the data (than that present in the data of the previous iteration). The quality of filtering-out the noise depends on the values of the filter gains, α , β , and γ . These gains are calculated from the tracking index which in its turn is calculated from T , σ_a , and σ_n . Let us assume that σ_n is known at this point. We know the value of T and we have an initial approximation to σ_a . When the α - β - γ filter is run on the noisy position data for the first time, it uses the initial approximation of σ_a (which we derived above) to calculate an initial value of the tracking index. Then it calculates the corresponding gain values and uses them to calculate the predicted and smoothed position, velocity and acceleration values. For the second iteration, the filter uses the smoothed acceleration values from the first iteration to calculate a new improved approximation for σ_a and the process is repeated again. The

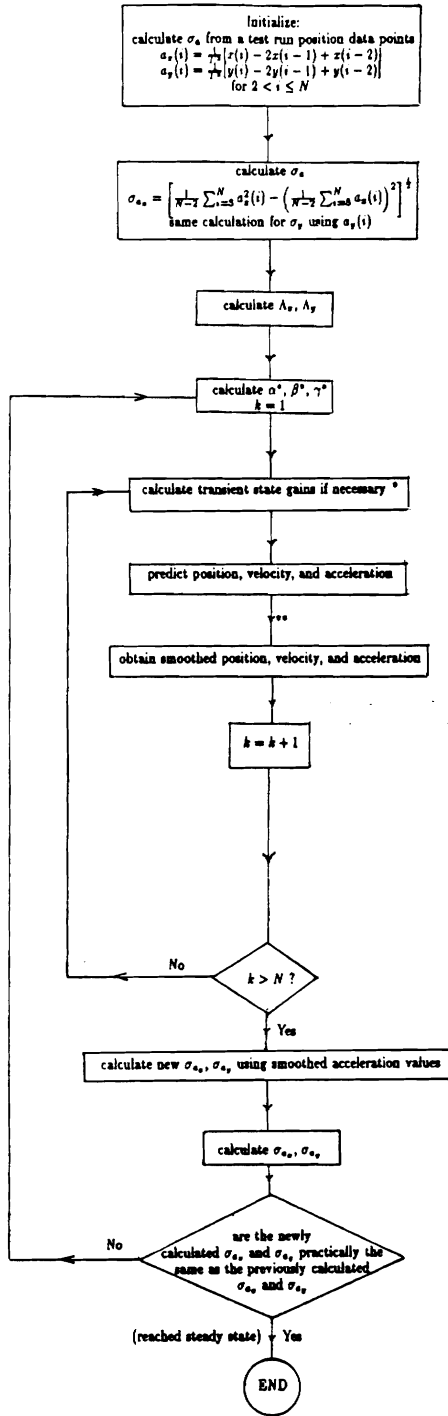
next iteration will produce even better values because the newly calculated σ_a is a closer approximation than its previous value. The whole process (see fig. 4.1) is repeated until the newly calculated σ_a asymptotes to a value that is closest to the actual σ_a .

We tested the above procedure using the simulation model (discussed in the previous chapter). The resulting σ_a was identical to the theoretically calculated value (as in sec 3.2) without the presence of noise. As we increased the noise level, the deviation from the theoretical value increased.

4.2 Determining σ_n

σ_n is a measure of the level of the uncertainty in the object position due to certain errors introduced during the position measurement process. Therefore, this quantity is entirely dependent on the measurement method. In our case, we use the existing moving object detection algorithm that was discussed in Chapter 1, to supply our tracking filter program with the moving object position; hence, σ_n corresponds to the errors introduced in this detection algorithm.

Recall that detecting the moving object may occur at any one of the five pyramid levels, (once an object has been detected, the algorithm calculates the center of the object and passes that as the position value). Each pyramid level has a corresponding σ_n . For example, one does expect the noise level in the lower resolution image to be greater than that in a higher resolution image. A safe assumption, (that was later proven to be correct), was that the noise level increases by the same factor at which the image resolution decreases, e.g. the noise level in a 32×32 resolution image is approximately 4 times that in the 128×128 resolution image. Then it is clear that once we obtain σ_n for any one of the pyramid levels,



* The procedure for calculating the transient gain is the same as that shown in Fig. 3.2 with an additional gain γ .

** No measurement need be done because we are using the same test run data points.

Figure 4.1 Calculation Procedure for σ_a .

it will be a matter of trivial scaling that will provide us with σ_n for each level.

The measurement noise present in any of these levels is mainly due to quantization noise, which is carried over to the calculation of the object center.

Let us consider σ_n for the 256×256 original image. We will perform our derivation for a one-dimensional model since, in our case, σ_n in the X -direction is identical to that in the Y -direction.

Our model is shown in Fig. 4.2. Let L be the length of a stretched wire. Let x_1 and x_2 be the position of the end points on the X -axis; and let x_{1q} and x_{2q} be the positions of the end points after quantization. Let ϵ be the difference between x_q and x (i.e., $x_q = x + \epsilon$). This is a random variable whose probability density function (pdf) is uniformly distributed between $\pm \frac{\Delta}{2}$, where Δ is the quantization step (see Fig 4.2b).

We would like to derive an expression for σ_n based on the variance of the estimate of the object center (here the center of the wire). Let \hat{x}_m represent this quantity. We proceed as follows:

$$\begin{aligned}\hat{x}_m &= \frac{x_{1q} + x_{2q}}{2} \\ &= \frac{x_1 + \epsilon_1 + x_2 + \epsilon_2}{2} = \frac{x_1 + x_2}{2} + \frac{\epsilon_1 + \epsilon_2}{2}\end{aligned}$$

Thus we can express \hat{x}_m in terms of the actual center of the object with an additive error factor $\frac{1}{2}(\epsilon_1 + \epsilon_2)$, i.e.:

$$\hat{x}_m = x_m + \frac{\epsilon_1 + \epsilon_2}{2}$$

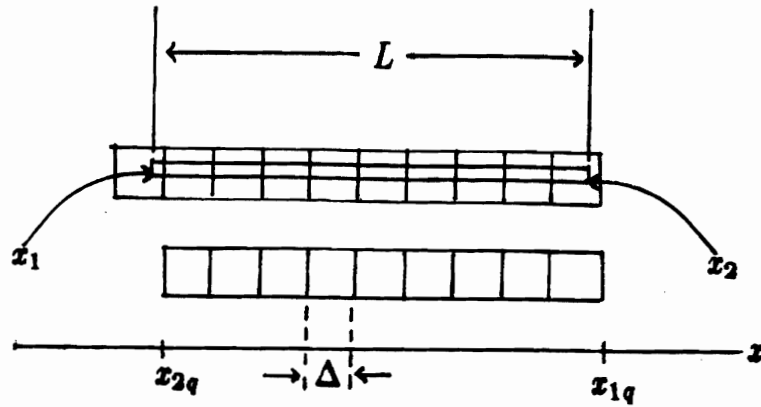


Figure 4.2a Model Used in Calculating the Quantization Noise. L Here is the Actual Length Before Quantizing.

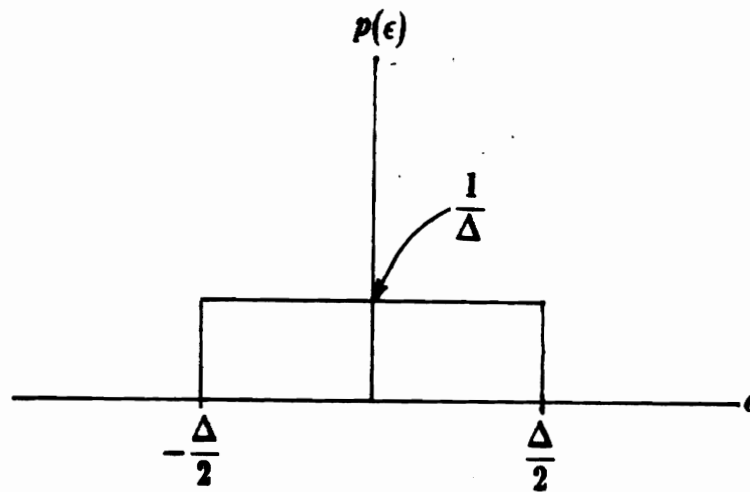


Figure 4.2b Probability Density Function of the Quantization Noise.

The expected value of \hat{x}_m is:

$$\begin{aligned} E\{\hat{x}_m\} &= E\{x_m\} + E\left\{\frac{\epsilon_1 + \epsilon_2}{2}\right\} \\ &= x_m + 0 \end{aligned}$$

therefore,

$$\begin{aligned} \text{var}[\hat{x}_m] &= \text{var}\left[\frac{\epsilon_1 + \epsilon_2}{2}\right] = \frac{1}{4}[\sigma_{\epsilon_1}^2 + \sigma_{\epsilon_2}^2] \\ &= \frac{1}{4}[2\sigma_\epsilon^2] \end{aligned}$$

hence,

$$\begin{aligned} \text{var}[\hat{x}_m] &= \frac{1}{2}\sigma_\epsilon^2 = \frac{1}{2}E\{\epsilon^2\} \\ &= \frac{1}{2} \int_{-\Delta/2}^{\Delta/2} \frac{1}{\Delta} \epsilon^2 d\epsilon \\ &= \frac{\Delta^2}{24} \end{aligned}$$

This gives us the following result:

$$\sigma_{\hat{x}_m}^2 = \text{var}[\hat{x}_m] = \frac{\Delta^2}{24}$$

and,

$$\sigma_{\hat{x}_m} = \frac{\Delta}{2\sqrt{6}}$$

Hence, the value of σ_n for the 256×256 original image is $\frac{\Delta}{2\sqrt{6}}$, where $\Delta = 1/256$. As for the other pyramid levels, σ_n is simply a 2^{n-1} multiple of that in the original image, where n is the image level. That is, σ_n for the second pyramid level is $\frac{2\Delta}{2\sqrt{6}}$; for the third level, $\frac{4\Delta}{2\sqrt{6}}$; etc.

A final note is that if the moving object is being detected in the third level, for example, our tracking algorithm uses the σ_n that corresponds to this level to calculate the tracking index. It is a simple matter for the detection algorithm to pass the pyramid level value since it is a known quantity.

These results were tested on position data files that the detection program produced (from the Pyramid Machine - IBM RT setup). Our tracking algorithm performed best when we used the proper σ_n , i.e., the one that corresponded to the pyramid level where the object was detected.

A typical image frame, obtained by our system, is shown in Figure 4.3. This is the original 256×256 image which is a part of a series of frames from which the position data of the detected moving vehicle was obtained. For this particular frame, the proper σ_n is $\frac{6\Delta}{2\sqrt{6}}$ which corresponds to the fourth (last) pyramid level. The vehicle image can be detected at this level because it occupies approximately two pixels out of 16 (in the fourth level), as seen from Figure 4.3.



Figure 4.3 A Single Frame from a Series of Frames Obtained by the Pyramid Machine (from which the position data of the detected moving vehicle was obtained).

CHAPTER 5

Conclusion and Recommendations for Future Work

We studied several tracking algorithms in order to provide a PVM1- based robotics and computer vision system with an adaptive tracking capability. The examined filters were: The Kalman, Wiener, α - β and α - β - γ filters, and the Two Point extrapolator. The decision criteria followed in choosing the appropriate algorithm was based on two objectives. The first was to obtain high performance accuracy in both transient and steady state track periods; the second was to choose a tracking algorithm with low real-time implementation cost.

The comparative study led us to choosing an optimal α - β filter that realizes the accuracy of the Kalman filter. The optimality of the adopted filter stems from the method used in producing the filter gains. This method is based on the "Tracking Index" parameter which is proportional to the ratio of uncertainties due to object accelerations (σ_a) and that due to measurement noise (σ_n).

In order to determine σ_a , we developed an adaptive method for inferring object dynamics in an iterative learning process. We used an α - β - γ filter for this purpose. The gain parameters of the latter filter are also determined by the tracking index; hence performing optimum smoothing. The resulting smoothed acceleration values are used to calculate the acceleration variance. The latter calculation is an integral part of this learning process.

As for σ_n , we learned by examining the existing moving-object detection algorithm, that the present measurement noise is mainly due to quantization effects. Hence, we were able to derive an expression for σ_n which inherently includes the quantization noise effects as they are introduced through the detection algorithm.

Having σ_a and σ_n as known quantities, we were able to use an algorithm as simple as that of the α - β filter, without increasing its complexity and implementation cost, to realize the accuracy of the Kalman filter. In addition, providing our system with an adaptive-learning tracker introduces the possibilities of using it in numerous robotics and computer vision applications.

We recommend the following for future work:

1. Provide the α - β filter with optimum on-line adaptive learning capability. This is useful in multi-moving-object environment where a priori information on the object dynamics is not available.
2. Extend the tracking algorithm to have the capability of tracking an object in a multi-moving-object environment. It has been shown in [16] that the tracking index plays a major role in the probabilistic position data -to- moving object association problem.
3. Combine the tracking algorithm with feature extraction capability [17] to single out a particular moving object out of the same assemblage of moving objects.
4. Use the tracking algorithm in conjunction with binocular imaging to compute the 3-D parameters of rigid motion.

REFERENCES

- [1] Ballard D., Brown C., *Computer Vision*, chap. 1, Prentice-Hall 1982.

- [2] Bandopadhyay A., Aloimonous J., "Active Navigation: Tracking an Environmental Point Considered Beneficial," *IEEE Motion Workshop*, 1986.

- [3] Van der Wal G., Sinniger J., "Real time pyramid transform architecture," *SPIE Vol. 579 Intelligent Robots and Computer Vision*, 1985.

- [4] Anderson C., Burt P., Van der Wal G., "Change detection and tracking using pyramid transform techniques", *SPIE Vol. 579 Intelligent Robots and Computer Vision*, 1985.

- [5] P. Kalata, "The tracking index: A generalized parameter for α - β and α - β - γ target trackers," *IEEE Trans. on Aerospace and Electronic Systems*, Vol AES-20, March 1984, pp 174-182.

- [6] Morrison N., *Tracking and Smoothing*. In Eli Brookner (Ed.), *Radar Technology*, Dedham, Mass., Artech House, 1977.

- [7] Brookner E., *The Kalman Filter*. In Eli Brookner (Ed.) *Radar Technology*, Dedham, Mass., Artech House, 1977.

- [8] Singer R., Behnke K., "Real-time tracking filter evaluation and selection for tactical

applications," *IEEE Trans. on Aerospace and Electronic Systems*, Vol. AES-7, No.1, 1971.

[9] Ziemer R., Tranter W., *Principles of Communication: Systems, Modulation, and Noise*, chap. 4, Houghton Mifflin Company, Ma., 1976.

[10] Benedict T., Bordner G., "Synthesis of an optimal set of Radar-while-scan smoothing," *IRE Trans. on Automatic Control*, Vol. AC-7, No.4 (July 1962, pp. 27-32.

[11] Fitzgerald R., "Simple Tracking Filters: Closed-Form Solutions," *IEEE Trans. on Aerospace and Electronic Systems*, Nov. 1981, pp 781-785.

[12] Acharya R., "Dynamic Scene Model: An Estimation Theory Approach," *IEEE Conference on Pattern Recognition and Image Processing*, Las Vegas, Nevada, June 1982.

[13] R. Burden, J. Faires, *Numerical Analysis*, third ed., Prindle, Weber & Schmidt publishers, MA., 1985.

[14] Skolnic M., *Introduction to Radar Systems*, second ed. McGraw Hill, N.Y., 1980.

[15] Schleher D., *Automatic Detection and Radar Data Processing*, Artech House Inc, Dedham, MA, 1980.

[16] R. Fitzgerald, "Development of practical PDA Logic for multitarget tracking by micro-processor," *Proc. 1986 American Control Conference*, June 1986, pp.889-898.

[17] Gorin Allen, "Aspect-Based aircraft classification from Dynamic Imagery," *IEEE Conference on Pattern Recognition and Image Processing*, Las Vegas, Nevada, June 1982.

APPENDIX I

Basic Equations of the Simulated Tracking Algorithms

(1) α - β Filter

Prediction:

$$x(k+1|k) = x(k|k) + Tv(k|k)$$

$$v(k+1|k) = v(k|k)$$

Correction:

$$x(k+1|k+1) = x(k+1|k) + \alpha(k+1)[z(k+1) - x(k+1|k)]$$

$$v(k+1|k+1) = v(k+1|k) + \frac{1}{T}\beta(k+1)[z(k+1) - x(k+1|k)]$$

(definitions will follow the α - β - γ filter equations)

(2) α - β - γ Filter

Prediction:

$$x(k+1|k) = x(k|k) + Tv(k|k) + \frac{1}{2}T^2a(k|k)$$

$$v(k+1|k) = v(k|k) + Ta(k|k)$$

$$a(k+1|k) = a(k|k)$$

Correction:

$$x(k+1|k+1) = x(k+1|k) + \alpha(k+1)[z(k+1) - x(k+1|k)]$$

$$v(k+1|k+1) = v(k+1|k) + \frac{1}{T}\beta(k+1)[z(k+1) - x(k+1|k)]$$

$$a(k+1|k+1) = a(k|k) + \frac{1}{2T^2}\gamma(k+1)[z(k+1) - x(k+1|k)]$$

$x(k|k)$ is the position estimate at time interval k

$x(k+1|k)$ is the position estimate at time interval $k+1$ given k samples (measurements)

$v(k|k)$ is the velocity estimate at time interval k

$a(k|k)$ is the acceleration estimate at time interval k

$\alpha(k)$ is the position tracking parameter at time k

$\beta(k)$ is the velocity tracking parameter at time k

$\gamma(k)$ is the acceleration tracking parameter at time k

T is the sampling period

$z(k+1)$ measured position at time $k+1$

(3) Two-Point Extrapolator

Prediction:

$$x(k+1|k) = x(k|k) + Tv(k|k)$$

$$v(k+1|k) = v(k|k)$$

Correction:

$$x(k+1|k+1) = z(k+1)$$

$$v(k+1|k+1) = \frac{1}{T}[z(k+1) - z(k)]$$

APPENDIX II

THE TRACKING PROGRAM

PROGRAM track;

(* This program performs tracking by using a Two Point Extrapolator, an Alpha-Beta Filter, or an Alpha-Beta-Gamma Filter. The tracking may be performed on variations of the simulation model or on a binary file of position data points. The adaptive-learning capability is also available as an option. *)

CONST

max_length = 150; (* maximum number of data points for use as a fading memory filter. This limit is for simulation purposes only. *)

m_color = 1;
pr_color = 2;
model_color = 3;

(* The following are constants related to the menu options. *)

(* cmse_opt for calculating maneuvering sigma estimate. *)

cmse_opt = 1;

(* catd_opt for continuous acceleration tracking demo. *)

catd_opt = 2;

(* spopm_opt is for tracking using series of positions obtained by the Pyramid Machine. *)

spopm_opt = 3;

exit_opt = 4;

(* sigma_n_0 is the quantization noise in the original 256x256 image *)

sigma_n_0 = 0.204;

TYPE

string_type = string[12];

array_type = array[1..max_length] of real;

VAR

(* The following arrays store the filters gains for the x and y directions. *)

gain_x, gain_y : array[2..3,1..3] of array_type;

gain_opt_x, gain_opt_y : array[2..3,1..3] of real;

gain_type : array[1..3] of string_type;

(* The following arrays are used to indicate if a particular gain of the fading memory filters has reached steady state or not. *)

steady_state_x,

steady_state_y : array[2..3,1..3] of boolean;

(* The following are first order time constants defined by the filter gain excursion from its initial state to its optimal state value. *)

k_taw_x : array[2..3,1..3] of real;

k_taw_y : array[2..3,1..3] of real;

(* The following variables are standard deviations for maneuvering accelerations and measurement noise. *)

```

sigma_a_x, sigma_a_y,
sigma_n_x, sigma_n_y : real;

tracking_index_x, tracking_index_y : real;

(* The following variable represents the "pyramid" level
   where the object is being detected. The value here is
   assigned by the user. Normally, however, this value will
   be passed by the existing Motion Detection program. Valid
   values for this variable are 0..4 *)
Pyramid_level : integer;

(* The following are the measured, predicted, and smoothed
   object position and velocity for the x and y directions. *)
m_x, m_y : array_type;
p_x, p_y,
s_x, s_y,
m_v_x, p_v_x, s_v_x,
m_v_y, p_v_y, s_v_y : array[1..3] of array_type;
s_a_x, p_a_x,
s_a_y, p_a_y : array_type;

(* The following are the simulation model position values. *)
model_x, model_y : array_type;

(* T is the sampling interval between time k and k+1. The
   sampling_factor reduces the sampling rate by the amount
   specified by this variable. *)
T : real;
sampling_factor : integer;

(* The following variables are associated with the
   continuous acceleration tracking demo, which uses the
   simulation model. *)
x_mag, y_mag : real;
w : real;
m : integer;
n_sampling, n_measurement_noise : integer;
phase : real;

(* miscellaneous *)
k, ctr, ctrl, ctr2 : integer;
max_runs : integer;
s : string_type;
option : integer;
delay_factor : integer;
filter_option : integer;
ptsfile : file of char;
file_name : string[12];
num_of_pts : integer;

(* Include the Cubic Spline interpolation procedure. *)
{$I cs_extrap.pas}

```

```
FUNCTION Gauss : real;
(* Returns a gaussian random variables by summing 12 samples of
  uniformly distributed random variable obtained from the
  predefined Random function. *)

Var
  temp : real;
  i : integer;
Begin (* gauss *)
  temp := 0.0;
  for i := 1 to 12 do
    temp := temp + random;
  Gauss := temp -6.0;
End; (* gauss *)
```

```
PROCEDURE get_position(l: integer);
```

```
(* This procedure reads the the next object position value (x and  
y) from the specified file. Then it checks to see if it is a  
valid point (based on how far the new position is from the  
previous one). If the point is invalid, the procedure  
calculates an approximation to what the proper value should  
have been using linear interpolation. Note: the first loop  
skips sampling_factor-1 points, in effect reducing the  
sampling rate by sampling_factor. *)
```

```
Const
```

```
    pos_dif_factor = 5;
```

```
Var
```

```
    position_difference : integer;
```

```
    temp_x, temp_y : char;
```

```
Begin (*get_position*)
```

```
    position_difference := pos_dif_factor * sampling_factor;
```

```
    for ctr := 1 to sampling_factor-1 do
```

```
        read(ptsfile, temp_x, temp_y);
```

```
    read(ptsfile, temp_x);
```

```
    read(ptsfile, temp_y);
```

```
    if (l > 2) then
```

```
        begin
```

```
            m_x[l] := ord(temp_x);
```

```
            m_y[l] := ord(temp_y);
```

```
            if ((abs(m_x[l] - m_x[l-1])) > position_difference) then  
                m_x[l] := 2*m_x[l-1] - m_x[l-2];
```

```
            if ((abs(m_y[l] - m_y[l-1])) > position_difference) then  
                m_y[l] := 2*m_y[l-1] - m_y[l-2];
```

```
        end
```

```
    else
```

```
        begin
```

```
            m_x[l] := ord(temp_x);
```

```
            m_y[l] := ord(temp_y);
```

```
        end;
```

```
End; (*get_position*)
```

```
PROCEDURE calc_sigma_a(first_time : boolean);
```

```
(* This procedure calculates an estimate of the maneuvering
standard deviation value. When first_time is true, the
procedure uses position points data from a test run file to
obtain the acceleration values. Otherwise, it will use the
smoothed acceleration values obtained via the alpha-beta-gamma
filter. *)
```

```
VAR
```

```
  v_x, v_y : array_type;
  sum_a_x, sum_a_y,
  sum_squared_a_x, sum_squared_a_y : real;
  i : integer;
  junk_file : text;
```

```
Begin (*calc_sigma_a*)
```

```
  sum_a_x := 0;
  sum_a_y := 0;
  sum_squared_a_x := 0;
  sum_squared_a_y := 0;
  if (first_time) then
    begin
      i := 1;
      for i := 1 to num_of_pts do
        get_position(i);
      for i := 3 to num_of_pts do
        begin
          v_x[i-1] := (m_x[i-1] - m_x[i-2])/T;
          v_x[i] := (m_x[i] - m_x[i-1])/T;
          s_a_x[i-2] := (v_x[i]-v_x[i-1])/T;

          v_y[i-1] := (m_y[i-1] - m_y[i-2])/T;
          v_y[i] := (m_y[i] - m_y[i-1])/T;
          s_a_y[i-2] := (v_y[i]-v_y[i-1])/T;
        end;
      end;
    end;
```

```
(* Calculate the estimate of the maneuvering standard
deviation *)
```

```
for i := 1 to num_of_pts-2 do
  begin
    sum_a_x := sum_a_x + s_a_x[i];
    sum_squared_a_x := sum_squared_a_x + sqr(s_a_x[i]);
    sum_a_y := sum_a_y + s_a_y[i];
    sum_squared_a_y := sum_squared_a_y + sqr(s_a_y[i]);
  end;
```

```
sigma_a_x := sqrt(sum_squared_a_x/num_of_pts -
                  sqr(sum_a_x/num_of_pts));
sigma_a_y := sqrt(sum_squared_a_y/num_of_pts -
                  sqr(sum_a_y/num_of_pts));
```

```
writeln;  
writeln('Estimated maneuvering standard deviation in the');  
writeln('x direction is : ', sigma_a_x);  
writeln;  
writeln('Estimated maneuvering standard deviation in the');  
writeln('y direction is: ', sigma_a_y);  
writeln;  
reset(ptsfile);  
End; (* calc_sigma_a*)
```


PROCEDURE Get_option;

(* This procedure displays the menus and obtains the selected options. *)

Var

answer : char;
parameter : integer;

Begin (*get_option*)

```
answer := 'y';
clrscr;
writeln('1. Calculate an estimate of maneuvering standard
          deviation');
writeln('    from a given test run. ');
writeln;
writeln('2. Continuous acceleration tracking demo');
writeln;
writeln('3. Track using series of positions obtained by
          the ');
writeln('    Pyremid machine');
writeln;
writeln('4. Exit');
writeln;
write('Please Enter option number: ');
readln(option);
if (option = catd_opt) then
begin
    clrscr;
    while (answer='y') or (answer='Y') do
    begin
        writeln(' The following parameters have the following
                values:');
        writeln;
        writeln(' 1. X-Y frequency factor = ', m);
        writeln;
        writeln(' 2. Radial frequency = ', w:5:3);
        writeln;
        writeln(' 3. Phase difference = ', phase:5:3);
        writeln;
        writeln(' 4. Number of samples of a full cycle of
                the faster');
        writeln('    sinusoid = ', n_sampling);
        writeln;
        writeln(' 5. "SNR" = ', n_measurement_noise);
        writeln;
        writeln(' 6. Display delay (ms) = ', delay_factor);
        writeln;
        write(' Do you like to change any of the parameters
              (y/n): ');
        readln (answer);
        while (answer='y') or (answer='Y') do
        begin
            writeln;
```

```

write(' Which parameter do you like to change
      (1..5): ');
readln(parameter);
writeln;
if (parameter=1) then
  begin
    write(' X-Y frequency factor = ');
    readln(m);
  end
else
  if (parameter=2) then
    begin
      write(' Radial frequency = ');
      read(w);
    end
  else
    if (parameter=3) then
      begin
        write(' Phase difference = ');
        readln(phase);
      end
    else
      if (parameter=4) then
        begin
          write(' Number of
                samples = ');
          readln (n_sampling);
        end
      else
        if (parameter=5) then
          begin
            write(' "SNR" = ');
            readln(n_measurement_
                  noise);
          end
        else
          if (parameter = 6) then
            begin
              write(' Display delay
                    (ms)= ');
              readln(delay factor);
            end
          else
            writeln(' Invalid
                    option... ');
          end
        end
      end
    end;
write('Do you like to change another
      (y/n): ');
readln(answer);
  end;
end;
end
else
  begin
    clrscr;
  end
end

```

```

writeln;
write('Please enter test-run file name: '),
readln(file_name);
assign(ptsfile, file_name);
writeln;
write('How many measured points does the file contain?');
readln(num_of_pts);
    reset(ptsfile);
writeln;
(* The following value will be normally passed by the
    existing Motion Detection Program. *)

write('Enter The pyramid level at which the object was
                                             detected: ');
readln(pyramid_level);
clrscr;
writeln(' The following parameters have the following
                                             values: ');
writeln;
writeln(' 1.    Sampling rate reduced by a factor = ',
                                             sampling_factor);
writeln;
writeln(' 2.    Display delay (ms) = ', delay_factor);
writeln;
write(' Do you like to change any of the parameters
                                             b(y/n): ');
readln (answer);
while (answer='y') or (answer='Y') do
    begin
        writeln;
        write(' Which parameter do you like to change
                                             (1..2): ');
        readln(parameter);
        writeln;
        if (parameter = 1) then
            begin
                write('Reduce the sampling rate by
                                             a factor of: ');
                readln(sampling_factor);
            end
        else
            if (parameter = 2) then
                begin
                    write('Display delay (ms)= ');
                    readln(delay_factor);
                end
            else
                writeln('Invalid option...');
        writeln;
        write('Do you like to change another
                                             (y/n): ');
        readln(answer);
    end;
end;
if (option = cmse_opt) then

```

```
begin
    filter_option := 3;
    T := 0.0333*sampling_factor;
    calc_sigma_a(true);
end;

if (option <> cmse_opt) then
begin
    clrscr;
    writeln;
    writeln('Which of the following filters you would like
            to use ? ');

    writeln;
    writeln('1. Two point extrapolator');
    writeln('2. Alpha-bata filter');
    writeln('3. Alpha-beta-gamma filter');
    writeln; write('Please enter option number: ');
    readln(filter_option);
end;
End; (*get_option*)
```

PROCEDURE calc_gains;

(* This procedure calculates the optimum steady state gains for the Alpha-Beta, and Alpha-Beta-Gamma filters. *)

Procedure calc_alpha_2(t_i: real;
var opt_gain: real);

(* This procedure calculates the optimum steady state alpha for the Alpha-Beta filter. *)

var
b: real;
root : real;

begin (*calc_alpha_2*)
b := 0.25*(t_i+8)*t_i;
root := 0.5*(-b+sqrt(sqr(b)+4*b));
if (root>0) and (root<1) then
opt_gain := root;
end; (*calc_alpha_2*)

Procedure calc_alpha_3(t_i : real;
var opt_gain : real);

(* This procedure calculates the optimum steady state alpha for the Alpha-Beta-Gamma filter. The Newton-Raphson method is used here for solving the cubic equation. *)

const
tol = 0.0001;
var
b, c, d: real;
p0, p : real;
f_p, f_d_p : real;
temp: real;
continue: boolean;

begin (*calc_alpha_3*)
b := 0.25*t_i*(t_i-16);
c := 0.25*t_i*(48-t_i);
d := -8*t_i;
p0 := 0.5;
temp := sqr(p0);
continue := true;
while continue do
begin
temp := sqr(p0);
f_p := p0*temp + b*temp + c*p0 + d;
f_d_p := 3*temp + 2*b*p0 + c;
p := p0 - f_p/f_d_p;
if (abs(p-p0)<tol) then
begin
opt_gain := p;
continue := false;
end
end

```

        else
            p0 := p;
        end;
    end; (*calc_alpha_3*)

Begin (*calc_gains*)
    if (filter_option = 2) then
        begin
            calc_alpha_2(tracking_index_x, gain_opt_x[2,1]);
            gain_opt_x[2,2] := 2 * (2 - gain_opt_x[2,1]) - 4
                * sqrt(1 - gain_opt_x[2,1]);

            calc_alpha_2(tracking_index_y, gain_opt_y[2,1]);
            gain_opt_y[2,2] := 2 * (2 - gain_opt_y[2,1]) - 4
                * sqrt(1 - gain_opt_y[2,1]);

        end
    else
        begin
            calc_alpha_3(tracking_index_x, gain_opt_x[3,1]);
            gain_opt_x[3,2] := 2 * (2 - gain_opt_x[3,1]) - 4
                * sqrt(1 - gain_opt_x[3,1]);
            gain_opt_x[3,3] := sqr(gain_opt_x[3,2]) /
                gain_opt_x[3,1];

            calc_alpha_3(tracking_index_y, gain_opt_y[3,1]);
            gain_opt_y[3,2] := 2 * (2 - gain_opt_y[3,1]) - 4
                * sqrt(1 - gain_opt_y[3,1]);
            gain_opt_y[3,3] := sqr(gain_opt_y[3,2]) /
                gain_opt_y[3,1];

        end;

        writeln('optimum alpha x = ', gain_opt_x[filter_option,1]
            :5:3);
        writeln('optimum alpha y = ', gain_opt_y[filter_option,1]
            :5:3);

        readln;
    End; (*calc_gains*)

```

```
PROCEDURE initialize_filter;
```

```
(* This procedure calculates the maneuvering, noise standard deviation and the tracking index. *)
```

```
Const
```

```
ln_10 = 2.30258;
```

```
Var
```

```
log_index_x,  
log_index_y : real;  
i, j : integer;
```

```
Begin (* initialize_filter *)
```

```
if (option=catd_opt) then  
begin
```

```
sigma_a_x := 0.707*sqr(w)*x_mag;  
sigma_a_y := 0.707*sqr(m*w)*y_mag;  
sigma_n_x := x_mag/n_measurement_noise;  
sigma_n_y := y_mag/n_measurement_noise;  
T := 6.283/(n_sampling*m*w);
```

```
end
```

```
else
```

```
begin
```

```
if (option=spopm_opt) then  
begin
```

```
write('Enter maneuvering standard deviation in  
the x direction: ');  
readln(sigma_a_x);  
writeln;  
write('Enter maneuvering standard deviation in  
the y direction: ');  
readln(sigma_a_y);
```

```
T := 0.0333*sampling_factor;
```

```
end;
```

```
(* The standard deviation value of the quantization  
noise is (2^pyramid_level)*sigma_n_0 *)
```

```
sigma_n_x := exp(pyramid_level*ln(2))*sigma_n_0;  
sigma_n_y := sigma_n_x;
```

```
end;
```

```
tracking_index_x := sqr(T)*sigma_a_x/sigma_n_x;  
tracking_index_y := sqr(T)*sigma_a_y/sigma_n_y;
```

```
writeln('Tracking_index x = ', tracking_index_x:5:3);
```

```
writeln('Tracking_index y = ', tracking_index_y:5:3);
```

```
if (filter_option <> 1) then
```

```
begin
```

```
log_index_x := ln(sqr(tracking_index_x))/ln_10;
```

```
log_index_y := ln(sqr(tracking_index_y))/ln_10;
```

```

if (filter_option=2) then
  i := 2
else
  i := 3;
for j := 1 to filter_option do
  begin
    if (log_index_x > 0) then
      steady_state_x[i,j] := true
    else
      begin
        steady_state_x[i,j] := false;
        k_taw_x[i,j] := cs_extrap(i-1,
          gain_type[j], log_index_x);
      end;
    if (log_index_y > 0) then
      steady_state_y[i,j] := true
    else
      begin
        steady_state_y[i,j] := false;
        k_taw_y[i,j] := cs_extrap(i-1,
          gain_type[j], log_index_y);
      end;
    end;
  end;

  calc_gains;

  gain_x[2,1,1] := 1.0;  gain_y[2,1,1] := 1.0;
  gain_x[2,2,1] := 1.0;  gain_y[2,2,1] := 1.0;
  gain_x[3,1,1] := 1.0;  gain_y[3,1,1] := 1.0;
  gain_x[3,2,1] := 1.5;  gain_y[3,2,1] := 1.5;
  gain_x[3,3,1] := 2.0;  gain_y[3,3,1] := 2.0;
end;

for i:= 1 to 3 do
  begin
    if (option=catd_opt) then
      begin
        model_x[i] := x_mag * sin(w*i*T);
        model_y[i] := y_mag * cos(m*w*i*T+phase);
        m_x[i] := model_x[i] + sigma_n_x * Gauss;
        m_y[i] := model_y[i] + sigma_n_y * Gauss;
      end
    else
      if (option=spopm_opt) then
        get_position(i);
      end;
    end;
  end;
end;

```

(* The following are the initial estimates of the Two Point extrapolator, Alpha-Beta, and Alpha_Beta_Gamma filters, respectively. *)


```

if (filter_option=1) then
  begin
    s_x[1,1] := m_x[3];
    s_y[1,1] := m_y[3];
    s_v_x[1,1] := (m_x[3]-m_x[2])/T;
    s_v_y[1,1] := (m_y[3]-m_y[2])/T;
  end
else
  if (filter_option=2) then
    begin
      s_x[2,1] := m_x[3];
      s_y[2,1] := m_y[3];
      s_v_x[2,1] := (m_x[3]-m_x[2])/T;
      s_v_y[2,1] := (m_y[3]-m_y[2])/T;
    end
  else
    begin
      s_x[3,1] := m_x[3];
      s_y[3,1] := m_y[3];
      s_v_x[3,1] := (3*m_x[3]-4*m_x[2]+m_x[1])/(2*T);
      s_v_y[3,1] := (3*m_y[3]-4*m_y[2]+m_y[1])/(2*T);
      s_a_x[1] := (m_x[3]-2*m_x[2]+m_x[1])/sqr(T);
      s_a_y[1] := (m_y[3]-2*m_y[2]+m_y[1])/sqr(T);
    end;
  end;
End; (*initialize_filter*)

```

```

PROCEDURE Calc_trans_gain(filter_id, gain_id : integer;
                          coordinate : char);

(* This procedure calculates the filter gains (alpha, beta, and
   gamma) during the transition period. It returns a true value
   when the steady state, optimal gain value has been reached,
   false otherwise. *)

Const
  error_percentage = 0.1;

Begin (*calc_trans_gain*)
  if (coordinate = 'x') then
    begin
      gain_x[filter_id, gain_id,k] := gain_x[filter_id,
      gain_id,k-1] + (1-exp(-1/k_taw_x[filter_id,gain_id]))
      * (gain_opt_x[filter_id,gain_id] - gain_x[filter_id,
      gain_id,k-1]);

      if (abs(100*(gain_x[filter_id,gain_id,k] -
      gain_opt_x[filter_id,gain_id])/
      gain_opt_x[filter_id,gain_id]) <
      error_percentage) then
        steady_state_x[filter_id, gain_id] := true;
      end
    else
      begin
        gain_y[filter_id, gain_id,k] := gain_y[filter_id,
        gain_id,k-1] + (1-exp(-1/k_taw_y[filter_id,gain_id]))
        * (gain_opt_y[filter_id,gain_id] - gain_y[filter_id,
        gain_id,k-1]);

        if (abs(100*(gain_y[filter_id,gain_id,k] -
        gain_opt_y[filter_id,gain_id])/
        gain_opt_y[filter_id,gain_id]) <
        error_percentage) then
          steady_state_y[filter_id, gain_id] := true;
        end;
      end;
    End; (*calc_trans_gain*)

```

PROCEDURE initialize;

Begin (* initialize *)

gain_type[1] := 'alpha';

gain_type[2] := 'beta';

gain_type[3] := 'gamma';

delay_factor := 0;

m := 2;

w := 6.28;

phase := 1.57;

n_sampling := 20;

n_measurement_noise := 256;

sampling_factor := 1;

x_mag := 125;

y_mag := 75;

get_option;

End; (* initialize *)

PROCEDURE add_measurement_noise;

(* This procedure adds noise to the ideal model location. The noise is white zero mean. *)

Begin

m_x[k] := model_x[k] + sigma_n_x * Gauss;

m_y[k] := model_y[k] + sigma_n_y * Gauss;

End;

```
PROCEDURE predict_1(k:integer);
```

```
(* The prediction in this procedure is based on that of the Two  
Point Extrapolator. *)
```

```
Begin (*predict_1*)
```

```
  p_x[1,k] := s_x[1,k-1] + T*s_v_x[1,k-1];
```

```
  p_y[1,k] := s_y[1,k-1] + T*s_v_y[1,k-1];
```

```
  p_v_x[1,k] := s_v_x[1,k-1];
```

```
  p_v_y[1,k] := s_v_y[1,k-1];
```

```
End; (*predict_1*)
```

```
PROCEDURE smooth_1(k: integer);
```

```
(* The smoothing in this procedure is based on that of the Two  
Point Extrapolator. *)
```

```
Begin (*smooth_1*)
```

```
  s_x[1,k] := m_x[k];
```

```
  s_y[1,k] := m_y[k];
```

```
  s_v_x[1,k] := (m_x[k]-m_x[k-1])/T;
```

```
  s_v_y[1,k] := (m_y[k]-m_y[k-1])/T;
```

```
End; (*smooth_1*)
```

```
PROCEDURE predict_2(k:integer);
```

```
(* The prediction in this procedure is based on that of the  
Alpha-Beta Filter. *)
```

```
Begin (*predict_2*)
```

```
  p_x[2,k] := s_x[2,k-1] + T*s_v_x[2,k-1];
```

```
  p_y[2,k] := s_y[2,k-1] + T*s_v_y[2,k-1];
```

```
  p_v_x[2,k] := s_v_x[2,k-1];
```

```
  p_v_y[2,k] := s_v_y[2,k-1];
```

```
End; (*predict_2*)
```

```
PROCEDURE smooth_2(k:integer);
```

```
(* The smoothing in this procedure is based on that of the Alpha-  
Beta Filter. *)
```

```
Var
```

```
  difference_x : real;
```

```
  difference_y : real;
```

```
Begin (*smooth_2*)
```

```
  difference_x := m_x[k] - p_x[2,k];
```

```
  difference_y := m_y[k] - p_y[2,k];
```

```
  s_x[2,k] := p_x[2,k] + gain_x[2,1,k]*difference_x;
```

```
  s_y[2,k] := p_y[2,k] + gain_y[2,1,k]*difference_y;
```

```
  s_v_x[2,k] := p_v_x[2,k] + (gain_x[2,2,k]/T)*difference_x;
```

```
  s_v_y[2,k] := p_v_y[2,k] + (gain_y[2,2,k]/T)*difference_y;
```

```
End; (*smooth_2*)
```

```
PROCEDURE predict_3(k:integer);
```

```
(* The prediction in this filter is based on that of the Alpha-  
Beta-Gamma Filter. *)
```

```
Begin (*predict_3*)
```

```
  p_x[3,k] := s_x[3,k-1] + T*s_v_x[3,k-1] + 0.5*tsqr(T)*s_a_x[k-1];  
  p_y[3,k] := s_y[3,k-1] + T*s_v_y[3,k-1] + 0.5*tsqr(T)*s_a_y[k-1];  
  p_v_x[3,k] := s_v_x[3,k-1] + T*s_a_x[k-1];  
  p_v_y[3,k] := s_v_y[3,k-1] + T*s_a_y[k-1];  
  p_a_x[k] := s_a_x[k-1];  
  p_a_y[k] := s_a_y[k-1];
```

```
End; (*predict_3*)
```

```
PROCEDURE smooth_3(k:integer);
```

```
(* The smoothing in this filter is based on that of the Alpha-  
Beta-Gamma Filter. *)
```

```
Var
```

```
  difference_x : real;  
  difference_y : real;
```

```
Begin (*smooth_3*)
```

```
  difference_x := m_x[k] - p_x[3,k];  
  difference_y := m_y[k] - p_y[3,k];  
  s_x[3,k] := p_x[3,k] + gain_x[3,1,k]*difference_x;  
  s_y[3,k] := p_y[3,k] + gain_y[3,1,k]*difference_y;  
  s_v_x[3,k] := p_v_x[3,k] + (gain_x[3,2,k]/T)*difference_x;  
  s_v_y[3,k] := p_v_y[3,k] + (gain_y[3,2,k]/T)*difference_y;  
  s_a_x[k] := p_a_x[k] + gain_x[3,3,k]/(2*sqr(T))*difference_x;  
  s_a_y[k] := p_a_y[k] + gain_y[3,3,k]/(2*sqr(T))*difference_y;
```

```
End; (*smooth_3*)
```

```
PROCEDURE get_statistics(x1, y1, x2, y2 : array_type);
```

```
(* This procedure calculates the standard deviations of the  
predicted and model values in both the x and the y directions.  
*)
```

```
Var
```

```
  zzave_x, zzave_y,  
  zave_x, zave_y : real;  
  i : integer;  
  sigma_x, sigma_y : real;  
  n : integer;
```

```
Begin (*get_statistics*)
```

```
  if (option<> catd_opt) then  
    n := num_of_pts
```

```
  else
```

```
    n := m*n_sampling-1;
```

```
  zzave_x := 0;  zzave_y := 0;
```

```
  zave_x := 0;  zave_y := 0;
```

```
  for i := 2 to n do
```

```
    begin
```

```
      zave_x := zave_x + (x1[i]-x2[i]);
```

```
      zave_y := zave_y + (y1[i]-y2[i]);
```

```
      zzave_x := zzave_x + sqr(x1[i]-x2[i]);
```

```
      zzave_y := zzave_y + sqr(y1[i]-y2[i]);
```

```
    end;
```

```
  zave_x := zave_x/n;
```

```
  zave_y := zave_y/n;
```

```
  zzave_x := zzave_x/n;
```

```
  zzave_y := zzave_y/n;
```

```
  sigma_x := sqrt(zzave_x - sqr(zave_x));
```

```
  sigma_y := sqrt(zzave_y - sqr(zave_y));
```

```
  clrscr;
```

```
  writeln;
```

```
  writeln(' The following are the values of the standard  
                                                    deviations of ');
```

```
  writeln;
```

```
  writeln(' the predicted and model position values:');
```

```
  writeln; writeln;
```

```
  writeln('Standard deviation in the x direction: ', sigma_x  
                                                    :5:3);
```

```
  writeln;
```

```
  writeln('Standard deviation in the y direction: ', sigma_y  
                                                    :5:3);
```

```
  readln;
```

```
  clrscr;
```

```
End; (*get_statistics*)
```

```
PROCEDURE drawp(x_temp, y_temp : real;
                color : integer);
Var
  x, y : integer;

Begin (*drawp*)
  if (option=catd_opt) then
    begin
      x := round(x_temp)+160;
      y := round(y_temp)+100;
    end
  else
    begin
      x := round(2240*x_temp/255) - 950;
      y := round(300*y_temp/255) - 100;
    end;

    draw(x-2, y, x+2, y, color);
    draw(x, y-2, x, y+2, color);
  End; (*drawp*)
```



```
BEGIN (*Track*)
```

```
initialize;
initialize_filter;
while (option <> exit_opt) do
  begin
    if (option <> cmse_opt) then
      begin
        graphcolormode;
        palette(2);
        textcolor(2); write(' R: predicted');
        textcolor(3); write(' y: model');
        textcolor(1); writeln(' G: measured');
      end;
    if (option=catd_opt) then
      max_runs := m*n_samplng+1
    else
      max_runs := num_of_pts;

    for k := 2 to max_runs do
      begin
        delay(delay_factor);
        if (filter_option=1) then
          begin
            predict_1(k);
            drawp(p_x[1,k], p_y[1,k], pr_color);
          end
        else
          if (filter_option=2) then
            begin
              predict_2(k);
              drawp(p_x[2,k], p_y[2,k],
                  pr_color);
            end
          else
            begin
              predict_3(k);
              if (option <> cmse_opt) then
                drawp(p_x[3,k], p_y[3,k],
                    pr_color);
              end;
            end;
          delay(delay_factor);

          if (option=catd_opt) then
            begin
              model_x[k] := x_mag * sin(w*(k+2)*T);
              model_y[k] := y_mag * cos(m*w*(k+2)*
                  T+phase);
            end
          else
            get_position(k);
          if (option = catd_opt) then
            begin
```

```

        drawp(model_x[k], model_y[k],
              model_color);
        add_measurement_noise;
        delay(delay_factor);
    end;

    drawp(m_x[k], m_y[k], m_color);

    if (filter_option <> 1) then
        begin
            if (filter_option=2) then
                ctr1 := 2
            else
                ctr1 := 3;
            for ctr2 := 1 to filter_option do
                begin
                    gain_x[ctr1,ctr2,k] :=
                        gain_opt_x[ctr1,ctr2];
                    gain_y[ctr1,ctr2,k] :=
                        gain_opt_y[ctr1,ctr2];
                end;
                if not(steady_state_x[ctr1,ctr2]) then
                    calc_trans_gain(ctr1,ctr2,'x');
                else
                    gain_x[ctr1,ctr2,k] := gain_opt_x[
                        ctr1,ctr2];
                for ctr2 := 1 to filter_option do
                    if not(steady_state_y[ctr1,ctr2])
                    then
                        calc_trans_gain(ctr1,ctr2,'y')
                    else
                        gain_y[ctr1,ctr2,k] :=
                            gain_opt_y[ctr1,ctr2];
                end;
            end;

            if (filter_option=1) then
                smooth_1(k)
            else
                if (filter_option=2) then
                    smooth_2(k)
                else
                    smooth_3(k);
            end;
        end;

    readln;

    if (option = catd_opt) then
        get_statistics(model_x, model_y, p_x[filter_option],
                      p_y[filter_option])
    else
        get_statistics(m_x, m_y, s_x[filter_option],
                      s_y[filter_option]);

    if (option = catd_opt) then
        get_option

```

```
else
  if (option = cmse_opt) then
    begin
      reset(ptsfile);
      calc_sigma_a(false);
      initialize_filter;
      option := exit_opt;

      graphcolormode;
      palette(2);
    end;
  If (option <> catd_opt) then
    close(ptsfile);
END. (*Track*)
```