60
81

EFFECTS OF TARGET'S ACCELERATION ON ALPHA-BETA TRACKING FILTERS

by

Leo Henry Hoffman, II

Project report submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING

in

Aerospace Engineering

APPROVED:

Dr. Eugene M. Cliff, Chairman

Dr. William H. Mason

Dr. Rakesh K. Kapania

June, 1990
Blacksburg, Virginia

EFFECTS OF TARGET'S ACCELERATION ON ALPHA-BETA TRACKING FILTERS

by
Leo Henry Hoffman, II
Dr. Eugene M. Cliff, Chairman
Aerospace Engineering

(ABSTRACT)

This paper examines the effect of a target's acceleration on the Fire Control System (FCS) $\alpha-\beta$ tracking filters used on the AEGIS cruisers. A single inbound target model was used to test the response of the tracking filter to an accelerating target. This target would begin to approach the AEGIS cruiser from a variety of distances ranging from 40,00 yards to 200,000 yards. The target model would begin its approach starting with an initial velocity of 200 yards/sec. and after a preselected time, the target would undergo an acceleration for a time duration of 5 seconds. The target's acceleration ranges from 1g to 6g's. For target's acceleration of 1g or greater, the difference between the actual and filtered velocity increases linearly with increasing acceleration and is fairly independent of range and the noise present in the measurement data. For target's acceleration less than 1g, the difference between the actual and filtered velocity is a strong function of acceleration, noise and range.

Acknowledgements

I would like to express my appreciation to Dr. Cliff for chairing my committee and to Dr. Mason and Dr. Kapania for serving as committee members and to all of the above for their encouragement and advice during the course of my project and graduate career here at Virginia Tech. I would like to thank Dr. Grossman for his guidance during my graduate studies. Finally, I would like to thank my mother and father who have made this opportunity possible and for their encouragement over the years. They will always be part of my life, dreams and acomplishments.

Table of Contents

1.0 Introduction 1

2.0 Filtering Equations 3

3.0 Results and Conclusions 18

4.0 Recommendations 30

Appendix A. Program TAFEP 31

Appendix B. Program TAFEV 39

Appendix C. Program Subroutines 47

Appendix D. Program DRW 57

Appendix E. Program DRWF 64

References 71

Vita 72

List of Tables

Table 1. Velocity Difference at 200,000 yards 28

Table 2. Velocity Difference at 160,000 yards 28

Table 3. Velocity Difference at 100,000 yards 29

Table 4. Velocity Difference at 40,000 yards 29

List of Figures

Figure 1. Flow chart diagram 17

Figure 2. Velocity difference at 200,000 yards 20

Figure 3. Velocity difference at 160,000 yards 21

Figure 4. Velocity difference at 100,000 yards 22

Figure 5. Velocity difference at 40,000 yards 23

Figure 6. Velocity difference at 200,000 yards without accel 24

Figure 7. Velocity difference at 160,000 yards without accel 25

Figure 8. Velocity difference at 100,000 yards without accel 26

Figure 9. Velocity difference at 40,000 yards without accel 27

Nomenclature

$x_f(t)$	filtered position estimate at time t
$x_p(t)$	predicted position at time step t
$x_m(t)$	measured position at time step t
$v_f(t)$	filtered velocity at time step t
Δt	time step interval
x,y,z	cartesian coordinates of target relative to ship
R	radial distance from ship to target

GREEK LETTERS

α	weighting coefficient for the position estimate
β	weighting coefficient for the velocity estimate
Γ	tracking index
σ_m	assumed target acceleration variance

σ_r assumed position measurement variance

θ azimuth angle

ϕ elevation angle

1.0 Introduction

This paper presents results from a study of the effects of a target's acceleration on $\alpha-\beta$ tracking filters. The radar used on the AEGIS cruisers, TAS AN/SPY-1A radar, can measure position directly during the search dwell, but it can not measure the target's velocity directly. Therefore, the radar system must use a differencing method to estimate the target's velocity. Once the target's velocity has been estimated, the next search coordinates for the radar can be predicted. But the measured position contains a large amount of noise which distorts the target's true position.

Some of the noise sources that are present in the measurements are: the radar equipment, ship motion, weather conditions and electronic jamming. The $\alpha-\beta$ filter reduces the effect of these noises as much as possible so that a reasonable estimate of the target's position can be used to approximate the target's velocity and thus predict the target's new position with an accuracy such that, upon the next search dwell, new targets will not be confused with existing targets. The $\alpha-\beta$ filter works satisfactorily when simulating targets traveling at a constant rate of speed, but the effects of simulating an accelerating or

maneuvering target are not known. It is the purpose of this paper to study these effects on the filter using several target models with different noise patterns integrated with a filtering system similar to the FCS found on AEGIS cruisers.

2.0 Filtering Equations

The $\alpha-\beta$ filter is used to track targets when there is a limited amount of computational time available. Unlike the Kalman filter, the $\alpha-\beta$ filter is a stationary noise process filter while the Kalman filter is a non-stationary noise process filter. If the Kalman filter is in a steady-state condition, stationary noise process, then it is equivalent to the $\alpha-\beta$ filter. An optimal relationship, in the sense that the effect of noise on the measured data is minimized, between α and β was originally derived by Benedict and Bordner using a calculus of variation approach. The Benedict-Bordner relationship may be derived using the Kalman filter equations assuming steady-state performance [5].

Another useful quantity called the tracking index, Γ, is completely specified by three parameters: the assumed target maneuverability, σ_m, the assumed target range variance, σ_r and the time increment, Δt. To show what assumptions went into the derivation of the Benedict-Bordner relationship, the equations for the Kalman filter are reviewed. The Kalman filter is a means of estimating information recursively, based on prior information obtained from

observations that are coupled by noise. It is a maximum likelihood estimator where the errors are assumed to be uncorrelated and Gaussian. The target kinematics are modeled as a Gauss-Markov random sequence [7],

$$\underline{X}(t+\Delta t) = \underline{U} \bullet \underline{X}(t) + \underline{W}(t) \tag{2.0-1}$$

while the measurement model is represented by

$$\underline{Z}(t) = \underline{H} \bullet \underline{X}(t) + \underline{n}(t) \tag{2.0-2}$$

where

$\underline{X}(t)$ is the (nx1) state vector at time t,

$\underline{U}$ is the (nxn) transition matrix relating $\underline{X}(t)$ to $\underline{X}(t+\Delta t)$,

$\underline{W}(t)$ is the (nx1) zero-mean, white-noise sequence with known covariance,

$\underline{Z}(t)$ is the (mx1) measurement vector at time t,

$\underline{H}$ is the (mxn) matrix defining the ideal connection between measurement and state vector,

$\underline{n}(t)$ is the (mx1) zero mean measurement vector error with known covariance and uncorrelated with the $\underline{W}(t)$ sequence,

$\underline{G}$ is the (nx1) matrix which relates the noise model to the zero-mean white acceleration noise,

The correlation of the zero-mean noise sequence is

$$E[\underline{W}(t1)\underline{W}(t2)^T] = \underline{Q} \text{ for } t1 = t2, \qquad (2.0\text{-}3)$$
$$= 0 \text{ for } t1 \text{ not equal to } t2$$

$$E[\underline{n}(t1)\underline{n}(t2)^T] = \underline{R} \text{ for } t1 = t2, \qquad (2.0\text{-}4)$$
$$= 0 \text{ for } t1 \text{ not equal to } t2$$

$$E[\underline{W}(t1)\underline{n}(t2)^T] = 0 \text{ for all } t1 \text{ and } t2 \qquad (2.0\text{-}5)$$

where (^{T}) indicates the transpose of the matrix.

The optimal, in the minimum mean squared error sense, estimate of $\underline{X}(t+\Delta t)$, (prediction of target's position), is obtained by weighting previous estimates, $\underline{X}(t)$, (filtered radar measurements), with current measurements, which gives

$$\underline{X}(t+\Delta t) = \underline{X}(t) + \underline{K}(t)\bullet(\underline{Z}(t)-\underline{H}\bullet\underline{X}(t)) \qquad (2.0\text{-}6)$$

with the definitions

$$\hat{\underline{X}}(t) \equiv \underline{X}(t+\Delta t) - \underline{X}(t),$$

the estimation error can now be written as

$$\hat{\underline{X}}(t) = (\underline{I}-\underline{K}(t)\underline{H})\underline{X}(t) + \underline{K}(t)\underline{n}(t) \tag{2.0-7}$$

The matrix $\underline{K}(t)$ is refered to as the Kalman gain matrix. The error covariance matrix, $\underline{P}(t+\Delta t) = E[\underline{X}(t+\Delta t)\underline{X}^T(t+\Delta t)]$, which updates the error covariance at time t, is obtained by taking the expected value of the estimation error outer product matrix.

$$\underline{P}(t+\Delta t) = (\underline{I}-\underline{K}(t)\bullet\underline{H})\bullet\underline{P}(t)\bullet(\underline{I}-\underline{K}(t)\bullet\underline{H})^T + \underline{K}(t)\bullet\underline{R}\bullet\underline{K}(t)^T \tag{2.0-8}$$

where $P(t) = E[\underline{X}(t)\underline{X}^T(t)]$ is the extrapolated error covariance at time t. The optimal Kalman gain matrix is obtained by minimizing the trace of $\underline{P}(t+\Delta t)$ with respect to $\underline{K}(t)$. This gives

$$\underline{K}(t) = \underline{P}(t)\bullet\underline{H}^T\bullet[\underline{R} + \underline{H}\bullet\underline{P}(t)\bullet\underline{H}^T]^{-1} \tag{2.0-9}$$

by using the matrix inversion lemma [7],

$$[\underline{P}(t+\Delta t)^{-1} + \underline{H}^T\bullet\underline{Q}\bullet\underline{H}]^{-1} = \underline{P}(t) - \underline{P}(t)\bullet\underline{H}^T\bullet[\underline{H}\bullet\underline{P}(t)\bullet\underline{H}^T + \underline{Q}^{-1}],$$

the form of the matrix can be simplified to obtained

$$K(t) = P(t+\Delta t)\bullet H^T\bullet[R^{-1}] \tag{2.0-10}$$

Substituting equation 2.0-10 into 2.0-9 gives the optimized value of the updated error covariance matrix as

$$\underline{P}(t+\Delta t) = (\underline{I}-\underline{K}(t)\bullet\underline{H})\bullet\underline{P}(t). \tag{2.0-11}$$

The extrapolation of the values between measurement (i.e., predicted values) are

$$\underline{X}(t+\Delta t) = \underline{U}\bullet\underline{X}(t), \tag{2.0-12a}$$

$$\underline{P}(t+\Delta t) = \underline{U}\bullet\underline{P}(t+\Delta t)\bullet\underline{U}^T + \underline{Q}. \tag{2.0-12b}$$

A more detailed discussion on the Kalman filter can be covered in Bryson and Ho in chapter 12 section 5. Now, with some background in Kalman filters covered, we can apply the Kalman filter equations to our model equations and begin to derive the Benedict-Bordner equation. Beginning with the one-dimensional, scalar equation of motion

$$x(t+\Delta t) = x(t) + u(t)\bullet\Delta t \tag{2.0-13a}$$

$$u(t+\Delta t) = u(t) + w(t)\bullet\Delta t \tag{2.0-13b}$$

where w(t) is the gaussian white noise sequence with known covariance with a

measurement model represented by

$$\underline{Z}(t) = \underline{X}(t) + \underline{n}(t) \tag{2.0-13c}$$

Our state vector, along with the other vectors mentioned at the beginning of this section, are now written as

$$\underline{X}(t) = \begin{bmatrix} x(t) \\ u(t) \end{bmatrix}, \ \underline{U} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}, \ \underline{G} = \begin{bmatrix} \frac{\Delta t^2}{2} & \Delta t \end{bmatrix}, \ \underline{H} = [\,1 \ \ 0\,], \ \underline{W}(t) = \begin{bmatrix} 0 \\ w_v \end{bmatrix}$$

along with $\underline{R} = \sigma^2{}_r$ and $\underline{Q} = \underline{G} \bullet \sigma^2{}_m \bullet \underline{G}^T$ and where n(t) is the measurement error with known covariance. Kalata has noted that a more realistic assumption is to assume a maneuver uncertainty, (angular uncertainty), in the position state. So the $\underline{W}(t)$ vector is modified to

$$\underline{W}(t) = \begin{bmatrix} w_x \\ w_v \end{bmatrix}$$

So now, the Gauss-Markov ramdom process [7] can now be written in vector form as

$$\underline{X}(t+\Delta t) = \underline{U} \bullet \underline{X}(t) + \underline{G} \bullet w_a \tag{2.0-14}$$

which is the same as equation 2.0-1. Also the measurement vector $\underline{Z}(t)$ is the same as equation 2.0-2.

For a stationary noise process, the gains in the Kalman filter will settle down to constant values. For ease of exposition we define α as K_{11} and β as $K_{12} \bullet \Delta t$. Combining equations 2.0-6 and 2.0-12a gives the filter equations in the form of the two-state Kalman filter as

$$\underline{X}(t+\Delta t) = \begin{bmatrix} 1-\alpha & \frac{1-\alpha}{\Delta t} \\ \frac{-\beta}{\Delta t} & 1-\beta \end{bmatrix} \underline{X}(t) + \begin{bmatrix} \alpha \\ \frac{\beta}{\Delta t} \end{bmatrix} \underline{Z}(t) \qquad (2.0\text{-}15)$$

As noted above, for stationary noise processes, the estimation error covariance and hence, the Kalman gains will approach steady-state values. Then, the matrices $P(t-\Delta t)$ and $P(t+\Delta t)$ will each have constant values which can be solved for the optimal relationship between α and β. Letting, then the Kalman filter equations with $\underline{P}(t-\Delta t) = \underline{P}$, $\underline{P}(t+\Delta t) = \underline{S}$, becomes

$$\underline{S} = (\underline{I} - \underline{K} \bullet \underline{H}) \bullet \underline{P} \qquad (2.0\text{-}16a)$$

$$\underline{P} = \underline{U} \bullet \underline{S} \bullet \underline{U}^T + \underline{G} \bullet \sigma^2_m \bullet \underline{G}^T \qquad (2.0\text{-}16b)$$

$$\underline{K} = \underline{S} \bullet \underline{H}^T / \sigma^2_r \qquad (2.0\text{-}16c)$$

Assuming that the elements of $\underline{S}$ are $S_{11} = a$, $S_{12} = S_{21} = b$ and $S_{22} = c$, gives

the equation 2.0-16c as

$$\begin{bmatrix} \alpha \\ \frac{\beta}{\Delta t} \end{bmatrix} = \begin{bmatrix} \frac{a}{\sigma^2_r} \\ \frac{b}{\sigma^2_r} \end{bmatrix}$$

(2.0-17)

Substituting the matrix $\underline{S}$ into 2.0-16b yields

$$\begin{bmatrix} p_{11} \\ p_{12} \\ p_{22} \end{bmatrix} = \begin{bmatrix} 1 & \Delta t & \Delta t^2 \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} + \begin{bmatrix} \frac{\Delta t^4}{4} \\ \frac{\Delta t^3}{2} \\ \Delta t^2 \end{bmatrix} \sigma^2_m \qquad (2.0\text{-}18)$$

While substituting the matrix $\underline{S}$ in 2.0-16a gives

$$a = (1-\alpha) \bullet p_{11} \qquad (2.0\text{-}19a)$$

$$b = (1-\alpha) \bullet p_{12} \qquad (2.0\text{-}19b)$$

$$b = -(\beta p_{11}/\Delta t) + p_{12} \qquad (2.0\text{-}19c)$$

$$c = -(\beta p_{12}/\Delta t) + p_{22} \qquad (2.0\text{-}19d)$$

Substituting equation 2.0-19d into the third equation in 2.0-18 gives

$$p_{12} = (\Delta t)^3 \bullet \sigma_m^2/\beta \tag{2.0-20}$$

Then substituting equations 2.0-20 and 2.0-19a into the first equation in 2.0-18 gives equation 2.0-21a and substituting equations 2.0-20 and 2.0-19b into the second equation in 2.0-18 gives equation 2.0-21b

$$\alpha \bullet a = (1-\alpha) \bullet (2 \bullet b \bullet \Delta t + c \bullet (\Delta t)^2 + (\beta \bullet \Delta t \bullet p_{12})/4) \tag{2.0-21a}$$

$$\alpha \bullet b = (1-\alpha) \bullet (c \bullet \Delta t + \beta \bullet p_{12}/2) \tag{2.0-21b}$$

Then substracting Δt times equation 2.0-21b from 2.0-21a yields

$$\alpha \bullet a - \alpha \bullet b \bullet \Delta t = (1-\alpha) \bullet (2 \bullet b \bullet \Delta t - (\beta \bullet \Delta t \bullet p_{12})/2) \tag{2.0-22}$$

From equation 2.0-17

$$\Delta t \bullet (\alpha/\beta) = a/b \tag{2.0-23a}$$

$$b \bullet (\alpha^2 + \beta \bullet (\alpha-2)) = -\beta^2 \bullet (1-\alpha) \bullet p_{12}/4 \tag{2.0-23b}$$

are obtained. Then substituting equation 2.0-19b into equation 2.0-23b and solving for β yields the result that is the Benedict-Bordner relationship.

$$\beta = 2\bullet(2-\alpha) - 4\bullet(1-\alpha)^{1/2} \tag{2.0-24}$$

As mentioned earlier, the $\alpha-\beta$ tracking filter can be written as a function of σ_r, σ_m and Δt. A relationship was developed by P. R. Kalata in 1976 [2] which relates the forementioned. This relationship is known as the tracking index, Γ, and is the following equation

$$\Gamma^2 = (\Delta t)^4\bullet(\sigma_m/\sigma_r)^2 \tag{2.0-25}$$

The tracking index can be derived from the equations used previously to derive the optimal relationship between α and β. Substituting the first equation of 2.0-17 into equation 2.0-19a gives

$$\alpha\bullet\sigma_r^2/(1-\alpha) = p_{11}. \tag{2.0-26}$$

Then noting from equation 2.0-19a and 2.0-19b that

$$p_{11}/p_{12} = a/b \tag{2.0-27}$$

and substituting equation 2.0-23a into equation 2.0-27 gives

$$\alpha \bullet \sigma_r 2/(1-\alpha) = \Delta t \bullet \alpha \bullet B/\beta \tag{2.0-28}$$

Then substituting equation 2.0-20 into equation 2.0-28 gives the result that

$$\beta^2/(1-\alpha) = \Gamma^2 \tag{2.0-29}$$

where Γ^2 is defined as

$$\Gamma^2 = (\Delta t)^4 \bullet (\sigma_m/\sigma_r)^2 \tag{2.0-30}$$

Notice that certain combinations of Δt, σ_m and σ_r will yield the same Γ^2. For example, doubling both noise variances will cause no change in the tracking index. There are many other combinations that can be thought of that will cause no change in the tracking index. Now, combining equation 2.0-30 with 2.0-24 gives an algebraic equation relating α to Γ. α has one real root on the interval [0,1]. Once α has been computed, β can be computed and hence, the data can now be filtered. So, with the input parameters known, Δt, σ_m and σ_r, α and β can be calculated. Note that the input parameters constant, thus α and β will remain constant.

$$((2(2-\alpha) - 4(1-\alpha)^{1/2})^2)/(1-\alpha) = \Gamma^2. \tag{2.0-31}$$

Now with the α-β filter derived, and the coefficients related to the input parameters, the tracking equations themselves can be developed and are presented as the following in the x direction and the equations are scalar:

$$diff = x_m(t) - x_p(t) \tag{2.0-32a}$$

$$x_f(t) = x_p(t) + \alpha(\sigma_r,\sigma_m,\Delta t)\bullet diff \tag{2.0-32b}$$

$$u_f(t) = u_f(t-\Delta t) + (\beta(\alpha)/\Delta t)\bullet diff \tag{2.0-32c}$$

$$x_p(t+\Delta t) = x_f(t) + u_f(t)\bullet\Delta t \tag{2.0-32d}$$

With $x_m(t)$ representing the target's measured position in x space, $x_p(t)$ is the target's predicted position in x space, $x_f(t)$ is the target's filtered position in x space and $u_f(t)$ is the target's filtered velocity in x space. The above equations are simply a weighted averaging of the actual target's position and the radar's predicted target's position. These equations require a unique initialization scheme and there are several ways to do the initialization.

For the purpose of this study, the simplest initialization scheme was implemented since the the scheme used on AEGIS cruisers are complex and classified. The scheme used in this study is as follows:

(1) The coefficients α is set equal to 1 and β is set equal to 0 for the first time step. This sets the target's predicted coordinates equal to the measured coordinates and the velocity is set to zero.

(2) The coefficients α is set equal to 1 and β is set equal to 1 for the second time step. This will set the position for the second time step two times the initial position and the velocity will be the input velocity.

(3) For the third time step, the coefficients α and β are calculated from the tracking index once and then these coefficients are used for the duration of the track.

Now the task of designing a simulation is the last part necessary to conduct this study. A flow chart diagram of the program's structure has been included in this report and is under Figure 1 at the end of this section. The simulation programs are included in this report and are listed under appendixes A thru C.

The primary goal for this study to be a success is the realistic simulation of the noise that an AEGIS cruiser's radar system would encounter in the environment. This was accomplished through utilizing a random number generator as input for a Gaussian distribution in which the standard deviations could be input by the user. Now the process of emulating actual radar data is straight foward. The path of the target can be generated through a series of target models. The coordinates, which are in Cartesian, are then converted into Spherical coordinates and the noise is then added, (σ_r is added to the target's range and σ_m is added to ϕ and θ), to these coordinates and then converted

back to Cartesian. This modeling yields an accurate representation of real measured radar data.

This simulation program, however, was written on the basis of a few assumptions. All of the radar data gathered by radar met a quality criteria. There were no system failures or lost of the target's track and the noise in the θ and ϕ directions were the same.

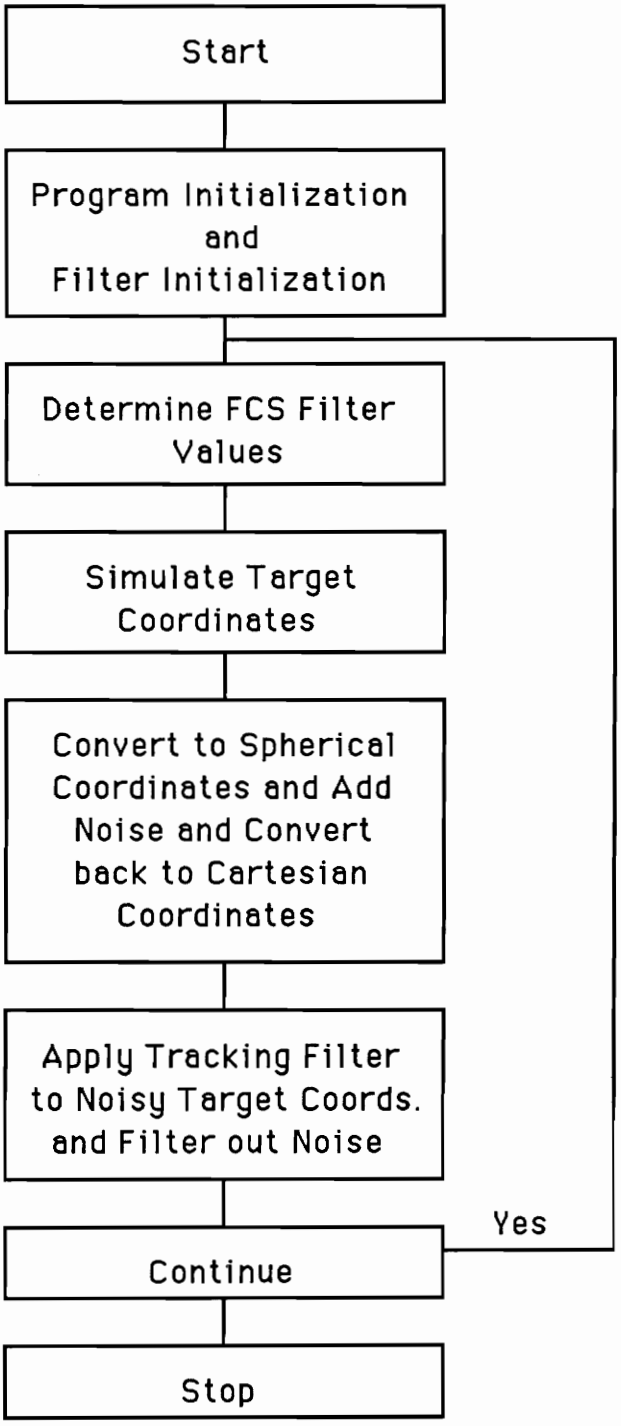


Figure 1. Flow chart of main computer simulation program

3.0 Results and Conclusions

Using a simple "side-direction" target model where the target approaches the AEGIS cruiser from its starboard side, a simple study of the reaction of the $\alpha-\beta$ tracking filter to an accelerating target can be carried out for different cases of range, noise and acceleration. Overall, the $\alpha-\beta$ filter takes approximately 10 seconds to settle down from the transient caused by the initialization scheme implemented in the main simulation program. For target's accelerations of 1g or greater, the maximum difference between the actual and filtered velocity increases linearly with acceleration and is fairly independent of noise and range. So one can draw a conclusion that for a target's accelerations of 1g and greater, the velocity difference is a linear function of acceleration. This can be observed in Tables 1 thru 4 and Figures 2 thru 5.

For target's accelerations less than 1g, the difference between the actual and filtered velocity is a strong function of acceleration, range and noise. Thus, the conclusion that can be drawn from this observation is that for a target's accelerations less than 1g, the velocity difference is a strong function of acceleration, noise and range. This can also be observed in Tables 1 thru 4

and Figures 6 thru 9.

So reviewing the stated observations and conclusions, the effect a target's acceleration on an $\alpha-\beta$ tracking filter causes the tracking filter to underpredict the target's position. As the target's acceleration increases linearly, the difference in the actual and filtered position also increases linearly.

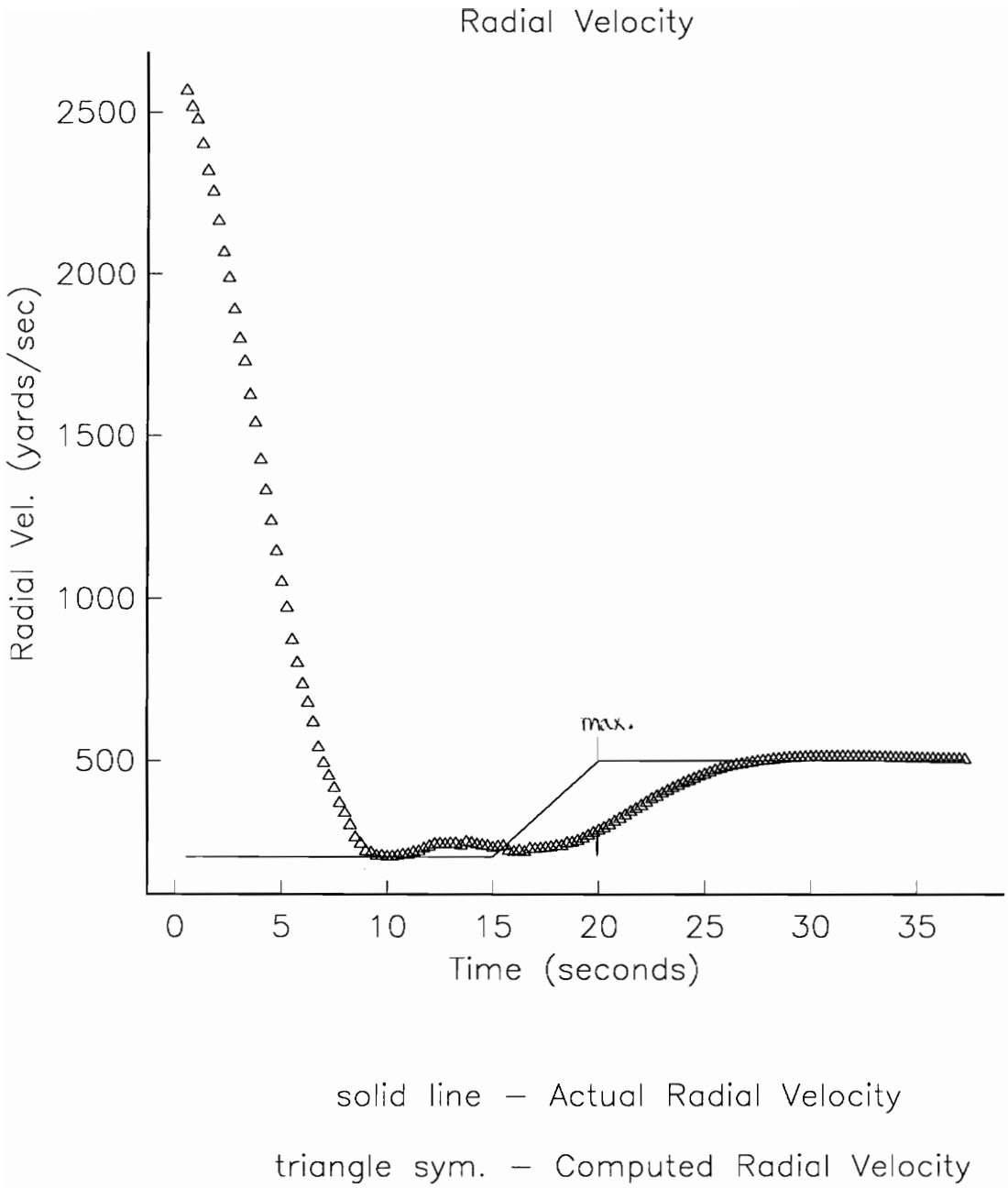


Figure 2. Velocity profiles beginning at 200,000 yards, 6g's, σ_r = 5.0 yards and σ_m = 1.5 mr

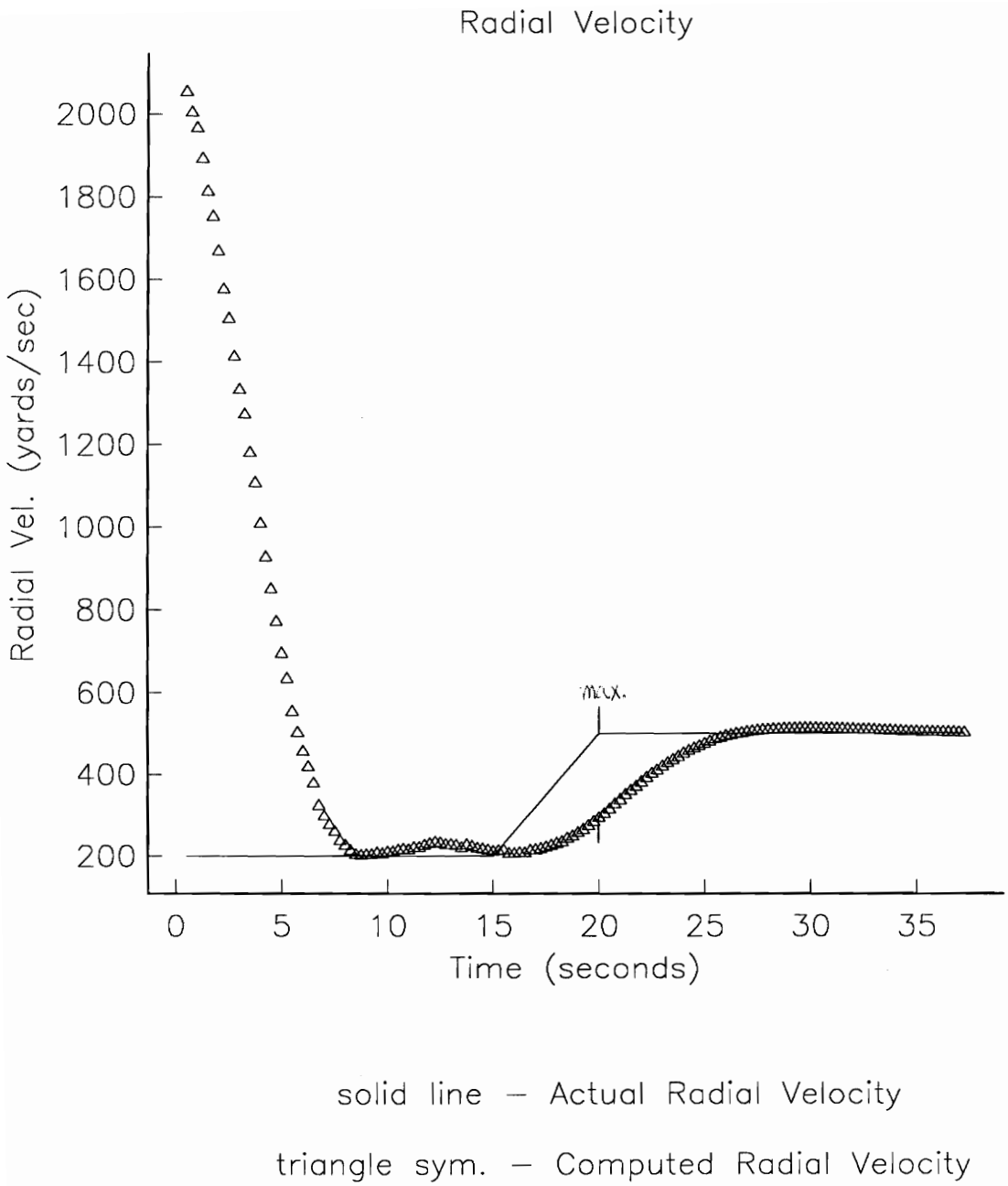


Figure 3. Velocity profile beginning at 160,000 yards, 6g's, σ_r = 5.0 yards and σ_m = 1.5 mr

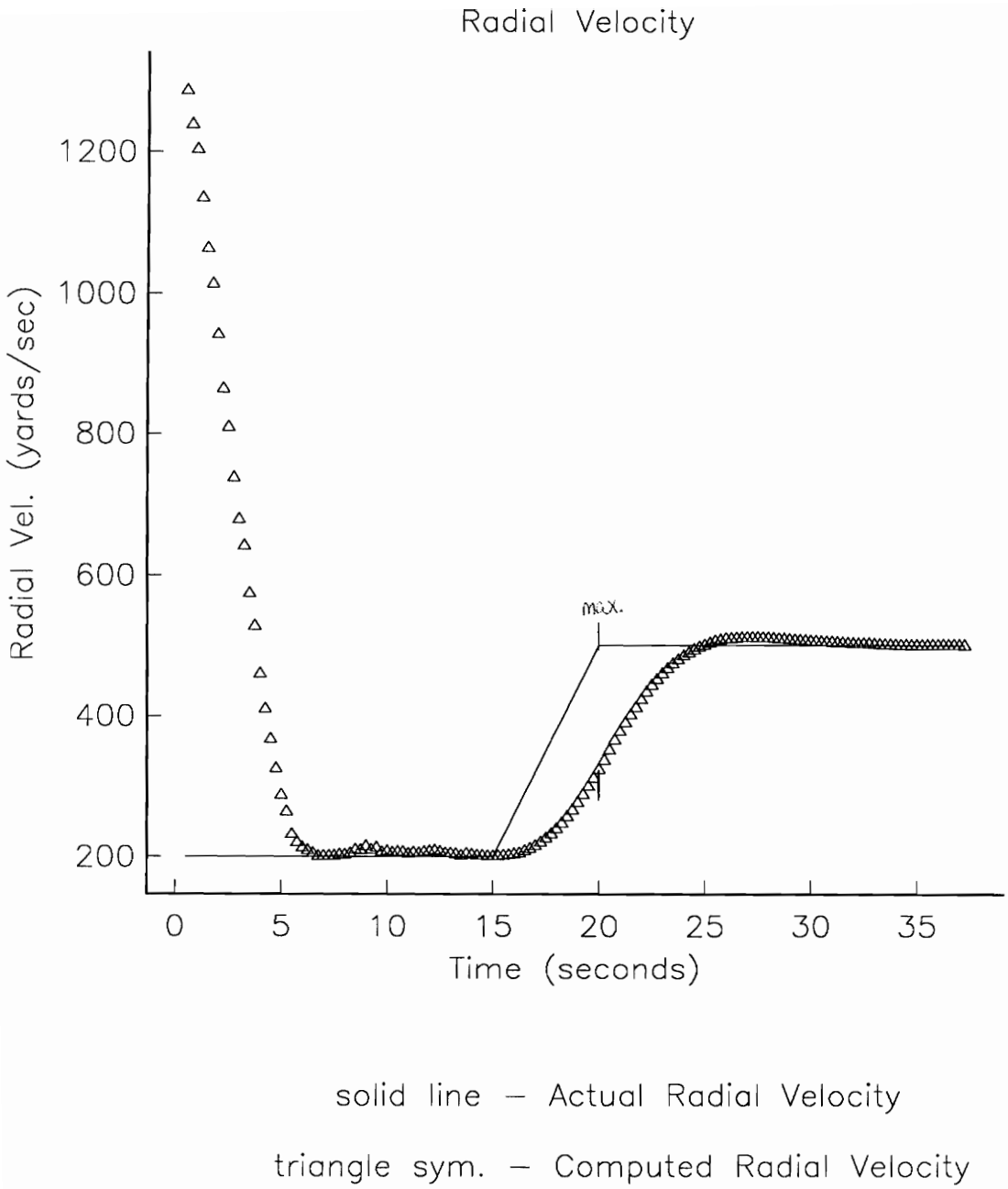
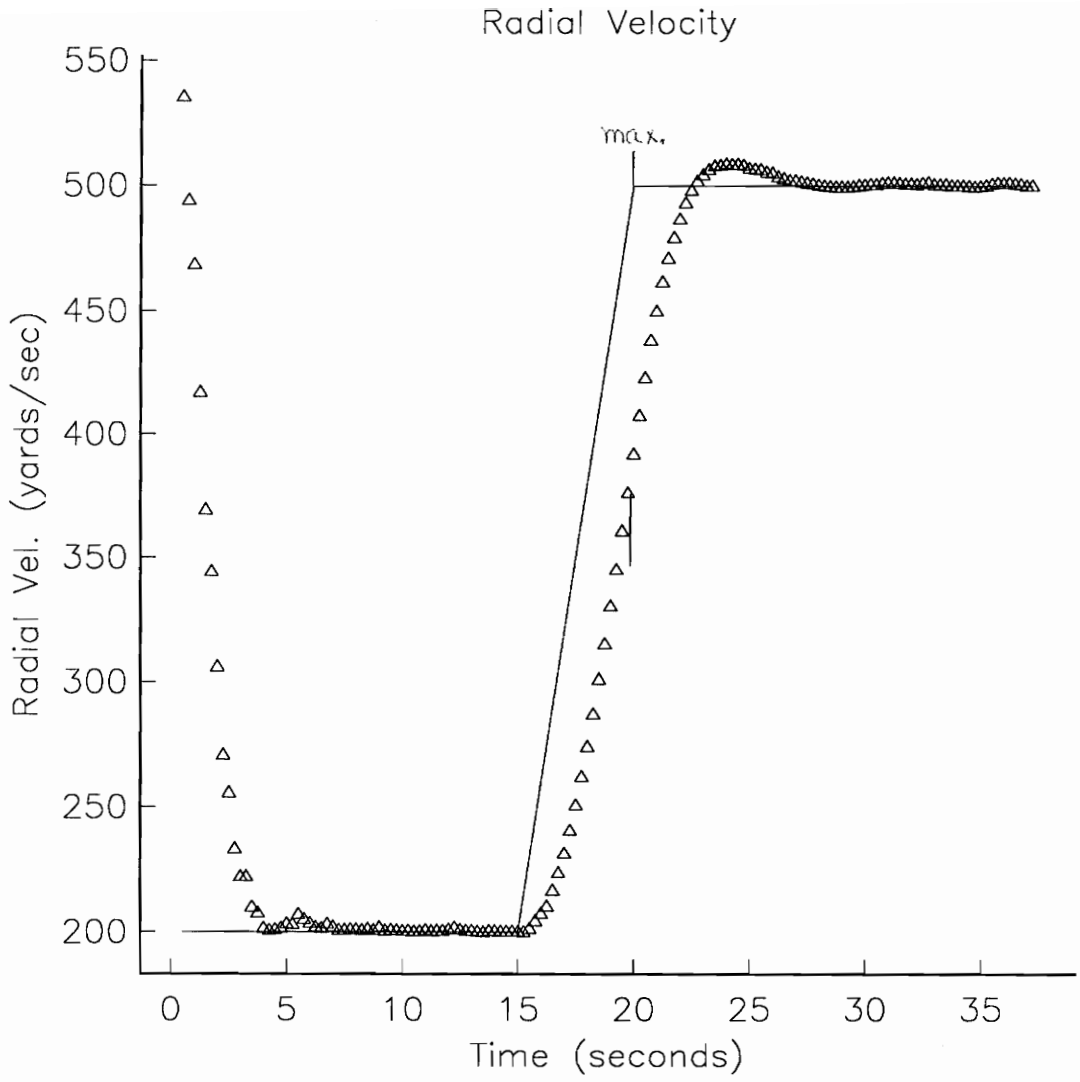


Figure 4. Velocity profile beginning at 100,000 yards, 6g's, σ_r = 5.0 yards and σ_m = 1.5 mr



solid line – Actual Radial Velocity

triangle sym. – Computed Radial Velocity

Figure 5. Velocity profile beginning at 40,000 yards, 6g's, σ_r = 5.0 yards and σ_m = 1.5 mr

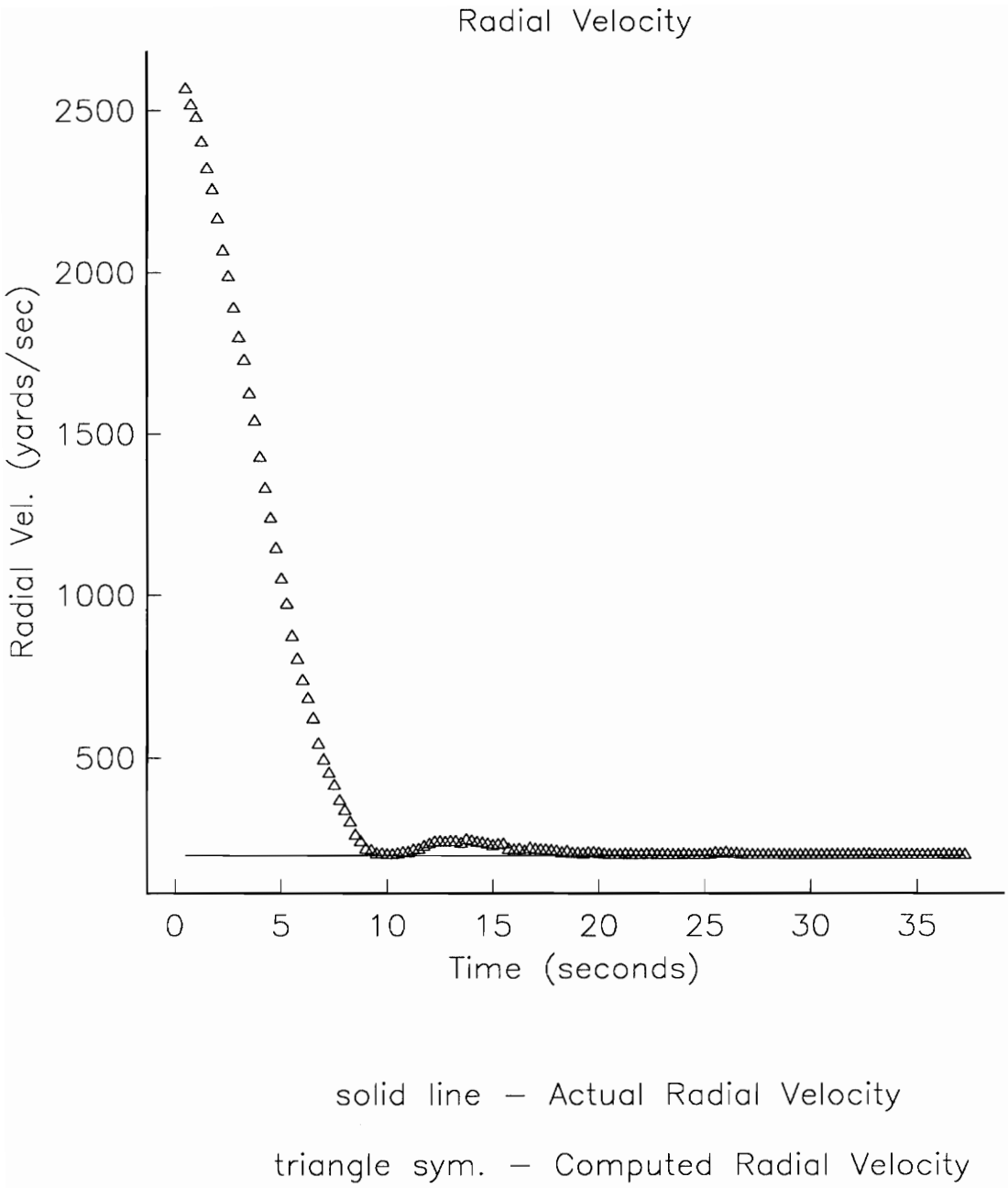


Figure 6. Velocity profile beginning at 200,000 yards, 0g's, σ_r = 5.0 yards and σ_m = 1.5 mr

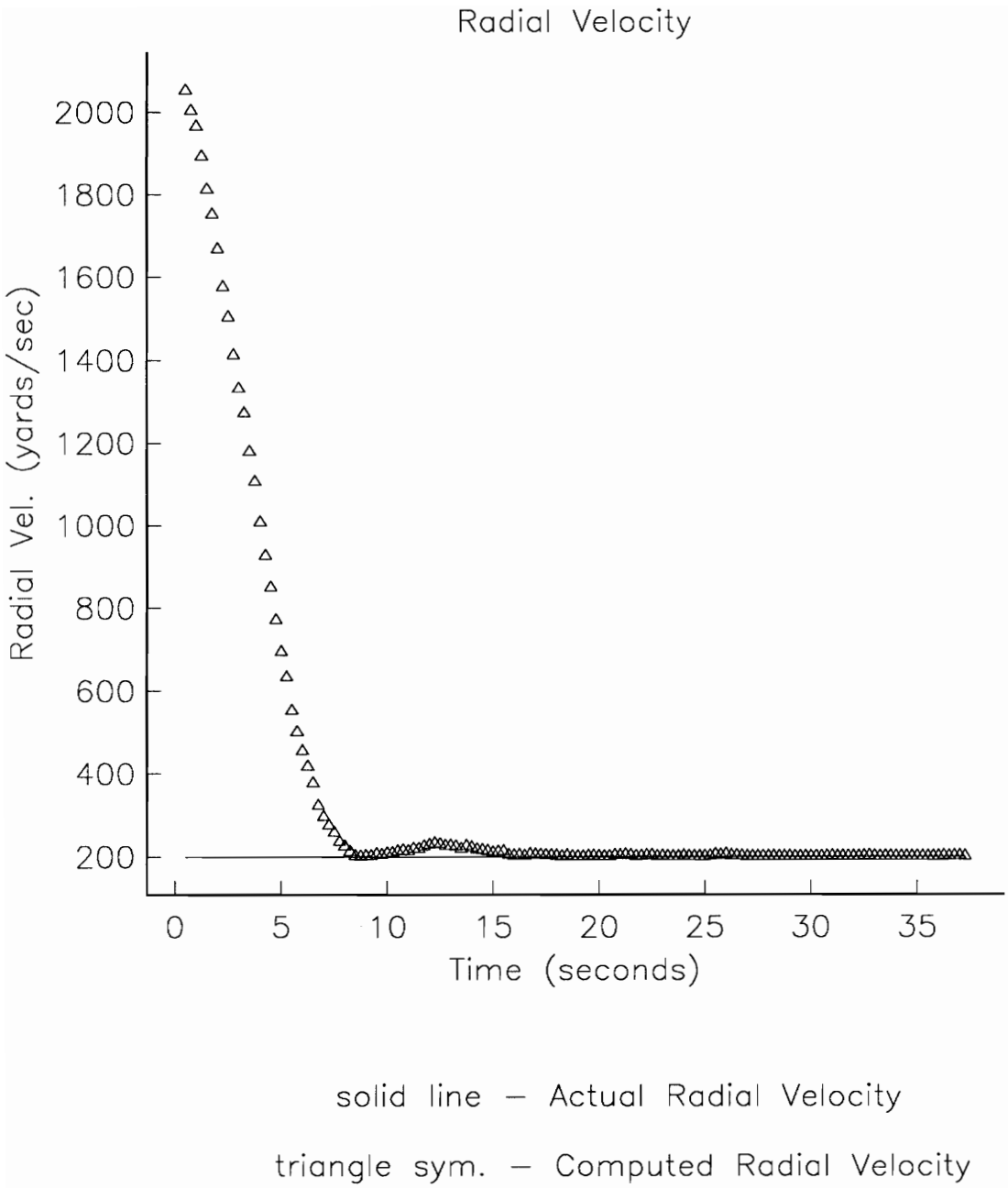


Figure 7. Velocity profile beginning at 160,000 yards, 0g's, σ_r = 5.0 yards and σ_m = 1.5 mr

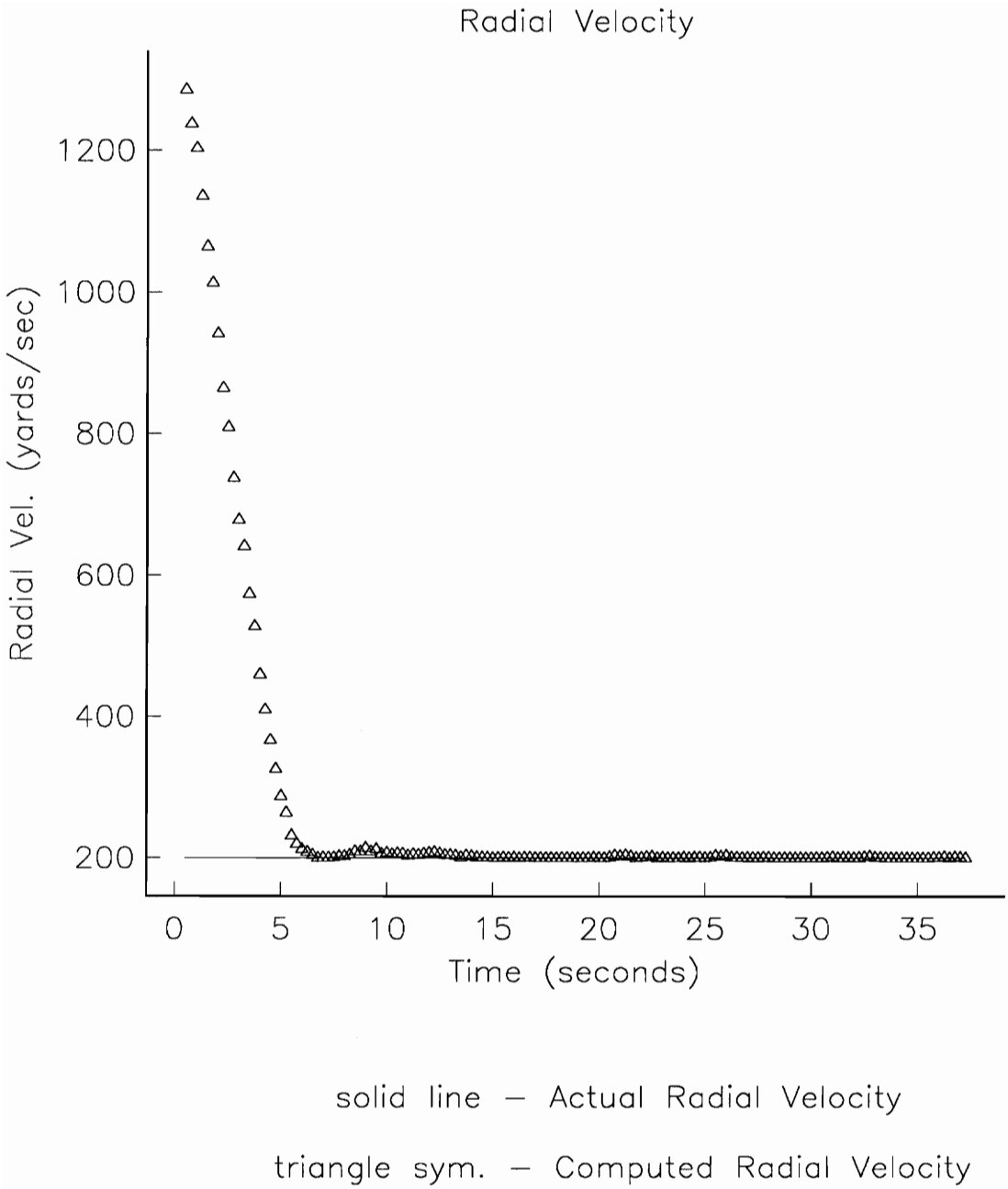


Figure 8. Velocity profile beginning at 100,000 yards, 0g's, σ_r = 5.0 yards and σ_m = 1.5 mr

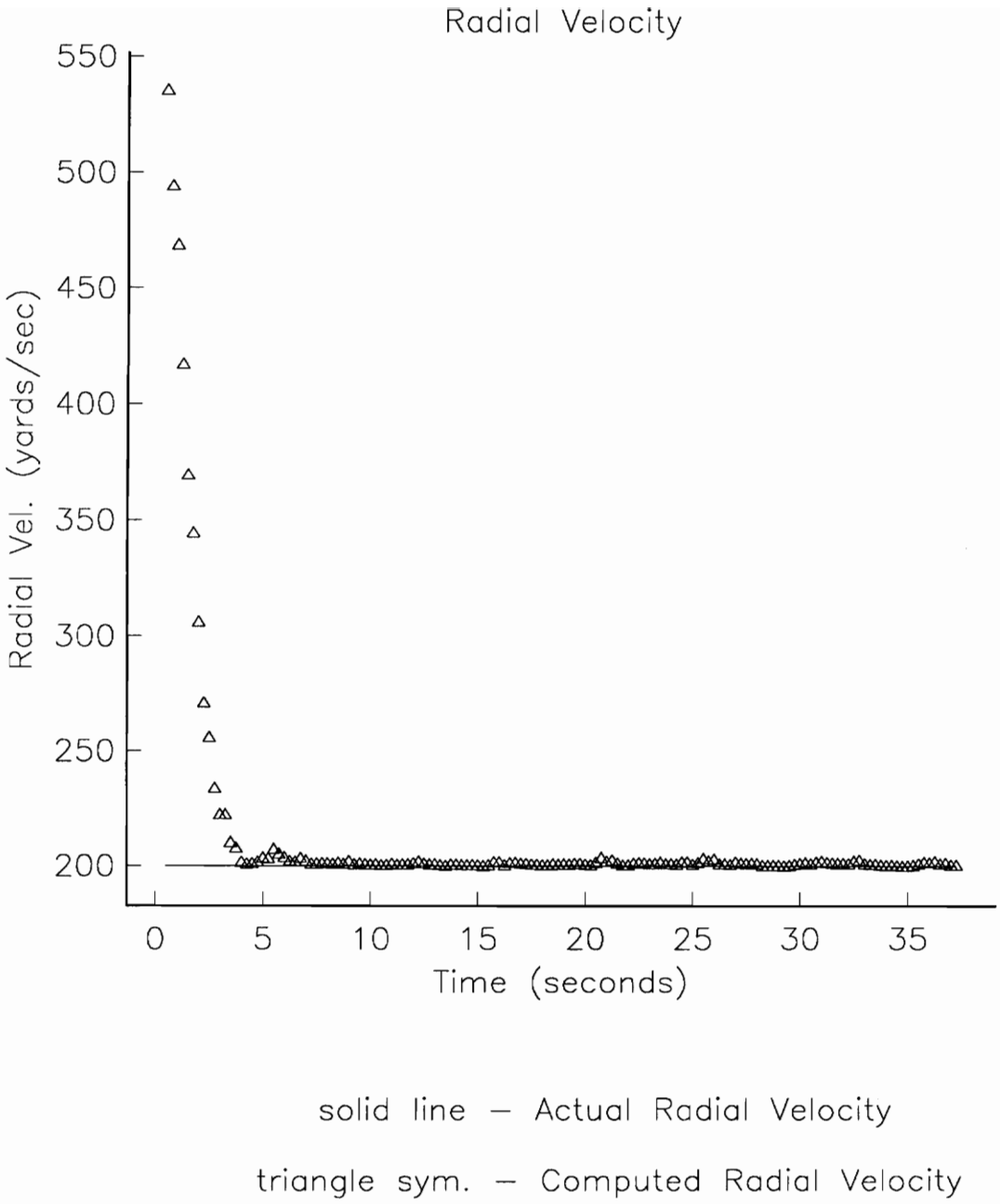


Figure 9. Velocity profile beginning at 40,000 yards, 0g's, σ_r = 5.0 yards and σ_m = 1.5 mr

Table 1. Velocity Difference at 200,000 yards

	Target's Acceleration				
Noise	6 g's	3 g's	2 g's	1 g's	0 g's
1.5 mr	134.375	68.75	43.75	34.37	37.5
1.0 mr	134.375	67.18	43.75	21.87	14.0
0.5 mr	135.937	67.5	45	22.5	3.75
0.0 mr	135.937	67.5	45	22.66	0.0

Table 2. Velocity Difference at 160,000 yards

	Target's Acceleration				
Noise	6 g's	3 g's	2 g's	1 g's	0 g's
1.5 mr	137.5	65.62	43.75	21.87	25
1.0 mr	134.375	67.81	43.75	21.87	17.5
0.5 mr	135.937	68.12	45	22.5	2.5
0.0 mr	135.937	67.5	45.5	22.66	0.0

Table 3. Velocity Difference at 100,000 yards

	Target's Acceleration				
Noise	6 g's	3 g's	2 g's	1 g's	0 g's
1.5 mr	135	67.5	45	22.5	7.5
1.0 mr	136.875	67.5	45	22.5	3.75
0.5 mr	136.875	68.75	45.62	22.75	0.875
0.0 mr	135	67.5	45.5	22.75	0.0

Table 4. Velocity Difference at 40,000 yards

	Target's Acceleration				
Noise	6 g's	3 g's	2 g's	1 g's	0 g's
1.5 mr	138.75	76.25	45.62	22.81	1.875
1.0 mr	136.5	76.25	45.62	23.125	1.25
0.5 mr	136.875	68.75	46.25	22.8	0.9
0.0 mr	135	67.5	45.62	22.5	0.0

4.0 Recommendations

An evaluation and comparison of other tracking filters is recommended for future work. Possible study of ocean swells during strong storms for better modelling. Also, the data quality acceptance criteria should be included for future study along with multiple targets and different approach tarjectories should be studied. Possible study of non-constant accelerating targets should also be given attention. Also, an examination of other target models along with other various initial velocities should be studied.

Appendix A. Program TAFEP

```
      program tafep

      parameter(n=800)

      real xm(n),ym(n),zm(n),xs(n),ys(n),zs(n)
      real us(n),vs(n),ws(n),xp(n),yp(n),zp(n),px(50),py(50),pz(50)
      real alpha,beta,t,load,r,theta,phi,mr,stime(50)
      integer ndec(50),count

      t = 0.0
      theta = 0.0
      phi = 0.0
      count = 1
      pi = 3.141593
      con = 0.0
      ztime = 0.0

c
c     This program is a simulation of a tracking situtation.  The modules of this
c      program solves for optimal alpha and beta, predicts new scanning
c      coordinates, adds in noise and converts between coordinates along with
c      processing radar data through the tracking filtering equations.  Note: This
c      program is valid for only one target!

      open(4,file='xt.dat',status='new')
      open(5,file='yt.dat',status='new')
      open(6,file='xt.dat',status='new')
      open(7,file='xs.dat',status='new')
      open(8,file='ys.dat',status='new')
      open(9,file='zs.dat',status='new')
```

```
      open(10,file='difxxm.dat',status='new')
      open(11,file='difyym.dat',status='new')
      open(12,file='difzzm.dat',status='new')
      open(13,file='vrs.dat',status='new')
      open(14,file='vrp.dat',status='new')
      open(15,file='difvr.dat',status='new')

c
c     Read in initial radar data to begin program execution.
c

      write(6,*) '          '
      write(6,*) 'Enter initial x, y and z coordinates & initial u, v and w velocities'
      read(5,*) xo,yo,zo,u,v,w

      uuo = u
      vvo = v
      wwo = w

      write(6,*) '          '
      write(6,*) 'Enter the acceleration Ax, Ay and Az.'
      read(5,*) ax,ay,az

      write(6,*) '          '
      write(6,*) 'Enter the number of the target model you wish to use.'
      write(6,*) '    Crossing target model with acceleration          (1)'
      write(6,*) '    Side direction target model with acceleration    (2)'
      write(6,*) '    Radially inbound target model                    (3)'
      write(6,*) '    Maneuvering target model                         (4)'
      write(6,*) '    3-D Approach target model with acceleration      (5)'
      write(6,*) '    Crossing target model w/o acceleration           (6)'
      write(6,*) '    Side direction target model w/o acceleration     (7)'
      write(6,*) '    3-D Approach target model with acceleration      (8)'
      read(5,*) idec

      if(idec.eq.3.or.idec.eq.4) then
        write(6,*) '          '
        write(6,*) 'Enter theta and phi.'
        read(5,*) theta,phi
        theta = theta*(pi/180)
        phi = phi*(pi/180)
      else
```

```
      endif

      write(6,*) '          '
      write(6,*) 'Enter the number of the filtering system to use.'
      write(6,*) '     Benedict-Bordner filter              (1)'
      write(6,*) '     Alpha-Beta-Gamma filter              (2)'
      write(6,*) '     FCS filter                           (3)'
      read(5,*) jdec

      if(jdec.eq.2) then
        write(6,*) '          '
        write(6,*) 'Enter the tracking index GAMMA.'
        read(5,*) 'GAMMA
      else
      endif

      write(6,*) '              '
      write(6,*) 'Enter the desired time increments in seconds.'
      read(5,*) step
      write(6,*) '              '

      write(6,*) 'Enter the number of switches between target models.'
      read(5,*) nswtch
      write(6,*) '          '

      stime(nswtch+1) = 1.0e+12
      ndec(nswtch+1) = 1

      if(nswtch.le.0) goto 13

      do 12 m = 1,nswtch
        write(6,*) 'Enter the switch over time in seconds.'
        read(5,*) stime(m)
        write(6,*) '          '
        write(6,*) 'Enter the number of the target model to switch over to.'
        read(5,*) ndec(m)
        write(6,*) '          '
        write(6,*) 'Enter the new accelerations, if any, Ax, Ay and Az.'
        read(5,*) px(m),py(m),pz(m)
        write(6,*) '          '
   12 continue
```

```
13  continue

      write(6,*) 'Enter the load factor; if not maneuvering then any no. will do.'
      read(5,*) load

      write(6,*) '          '
      write(6,*) 'Enter the standard deviation for Theta & Phi in milliradians.'
      read(5,*) mr
      write(6,*) '        '
      write(6,*) 'Enter the standard deviation for Range in yards.'
      read(5,*) rr
      write(6,*) '              '

      write(6,*) 'Do you want to add this noise? y=1, n=0'
      read(5,*) ijk
      write(6,*) '              '

      write(6,*) 'Enter the number of data steps.'
      read(5,*) length
      write(6,*) '        '
      write(6,*) '         '

c
c     Filter Pre-Initialization Step #1
c

c     k = 1 (Step 1)

c     alpha = 1.0
c     beta = 0.0

      call convert(xo,yo,zo,xm,ym,zm,rb,mr,rr,1,n,ijk)

      xs(1) = xm(1)
      ys(1) = ym(1)
      zs(1) = zm(1)

      xp(2) = xs(1)
      yp(2) = ys(1)
      zp(2) = zs(1)

      us(1) = 0.0
```

```
      vs(1) = 0.0
      ws(1) = 0.0

      write(6,1) xs(1),ys(1),zs(1),1
      write(6,2) us(1),vs(1),ws(1)

c
c     Filter Pre-Initialization Step #2
c

c    k = 2

      alpha = 1.0
      beta = 1.0
      t = step

      r = sqrt(xs(1)**2+ys(1)**2+zs(1)**2)

      call target(xt,yt,zt,xo,yo,zo,u,v,w,ax,ay,az,r,theta,phi,idec,t,load,1,n,con,
     $ztime)

      call convert(xt,yt,zt,xm,ym,zm,rc,mr,rr,2,n,ijk)

      xs(2) = xp(2) + alpha*(xm(2)-xm(1))
      ys(2) = yp(2) + alpha*(ym(2)-ym(1))
      zs(2) = zp(2) + alpha*(zm(2)-zm(2))

      us(2) = us(1) + (beta/step)*(xm(2)-xm(1))
      vs(2) = vs(1) + (beta/step)*(ym(2)-ym(1))
      ws(2) = ws(1) + (beta/step)*(zm(2)-zm(1))

      xp(3) = xs(2) + step*us(2)
      yp(3) = ys(2) + step*vs(2)
      zp(3) = zs(2) + step*ws(2)

      write(6,1) xs(2),ys(2),zs(2),2
      write(6,2) us(2),vs(2),ws(2)

c
c     Begin main program loop.
c
```

```
      do 10 k = 3,length
        inct = k
        t = t+step

c
c     After target data has smoothed out , switch over to new target model once
c     time criteria has been satisfied and continue the switching process as
c     required.
c

        if(t.ge.stime(count)) then
              idec = ndec(count)
              con = stime(count)

              u = u + ax*del
              v = v + ay*del
              w = w + az*del

              uuo = u
              vvo = v
              wwo = w

              ax = px(count)
              ay = py(count)
              az = pz(count)

              if((count/2)*2.eq.count) then
                ztime = con-step
                xo = xt
                yo = yt
                zo = zt
              else
              endif

              count = count + 1
        else
              del = abs(t-con+step)
        endif

c
c     Solve for optimal Alpha and Beta.
c
```

```
      range = sqrt(xp(k)**2+yp(k)**2+zp(k)**2)

      call fcsfil(jdec,range,alpha,beta,gamma,step,mr,rr)

c
c     Predict mew radar scanning coordinates from current data.
c

      r = sqrt(xs(k-2)**2+ys(k-1)**2+zs(k-1)**2)

      call target(xt,yt,zt,xo,yo,zo,u,v,w,ax,ay,az,r,theta,phi,idec,t,load,k-1,n,
     $con,ztime)

c
c     Convert from Cartesian to Spherical, add noise and convert back.
c

      call convert(xt,yt,zt,xm,ym,zm,ra,mr,rr,k,n,ijk)

c
c     Process radar data through track filtering equations for smoothing and
c     updating of the velocities.
c

      xs(k) = xp(k) + alpha*(xm(k)-xp(k))
      ys(k) = yp(k) + alpha*(ym(k)-yp(k))
      zs(k) = zp(k) + alpha*(zm(k)-zp(k))

      us(k) = us(k-1) + (beta/step)*(xm(k)-xp(k))
      vs(k) = vs(k-1) + (beta/step)*(ym(k)-yp(k))
      ws(k) = ws(k-1) + (beta/step)*(zm(k)-zp(k))

      xp(k+1) = xs(k) + step*us(k)
      yp(k+1) = ys(k) + step*vs(k)
      zp(k+1) = zs(k) + step*ws(k)

      uuu = uuo + ax*(t-con)
      vvv = vvo + ay*(t-con)
      www = wwo + az*(t-con)

      vrs = sqrt(us(k)**2+vs(k)**2+ws(k)**2)
```

```
      vrp = sqrt(uuu**2+vvv**2+www**2)

      write(6,1) xs(k),ys(k),zs(k),inct
      write(6,2) us(k),vs(k),ws(k)
      write(4,3) t,xs(k)
      write(5,3) t,ys(k)
      write(6,3) t,zs(k)
      write(7,3) t,xt
      write(8,3) t,yt
      write(9,3) t,zt
      write(13,3) t,vrs
      write(14,3) t,vrp

      difx = xt-xs(k)
      dify = yt-ys(k)
      difz = zt-zs(k)
      difvr = vrs-vrp

      write(10,3) t,difx
      write(11,3) t,dify
      write(12,3) t,difz
      write(15,3) t,difvr

1  format(1x,'Xs = ',f13.5,'Ys = ',f13.5,'Zs = ',f13.5,'Count = ',i6)
2  format(1x,Xus = ',f13.5,'Yvs = ',f13.5,'Zws = ',f13.5//)
3  format(1x,f13.5,2x,f13.5)

10  continue
11  continue

    stop
    end
```

Appendix B. Program TAFEV

```
      program tafep

      parameter(n=800)

      real xm(n),ym(n),zm(n),xs(n),ys(n),zs(n)
      real us(n),vs(n),ws(n),xp(n),yp(n),zp(n),px(50),py(50),pz(50)
      real alpha,beta,t,load,r,theta,phi,mr,stime(50)
      integer ndec(50),count

      t = 0.0
      theta = 0.0
      phi = 0.0
      count = 1
      pi = 3.141593
      con = 0.0
      ztime = 0.0

c
c    This program is a simulation of a tracking situtation.  The modules of this
c     program solves for optimal alpha and beta, predicts new scanning
c     coordinates, adds in noise and converts between coordinates along with
c     processing radar data through the tracking filtering equations.  Note: This
c     program is valid for only one target!

      open(4,file='ut.dat',status='new')
      open(5,file='vt.dat',status='new')
      open(6,file='wt.dat',status='new')
      open(7,file='us.dat',status='new')
      open(8,file='vs.dat',status='new')
      open(9,file='ws.dat',status='new')
```

```
      open(10,file='difuum.dat',status='new')
      open(11,file='difvvm.dat',status='new')
      open(12,file='difwwm.dat',status='new')
      open(13,file='prs.dat',status='new')
      open(14,file='prp.dat',status='new')
      open(15,file='difpr.dat',status='new')

c
c     Read in initial radar data to begin program execution.
c

      write(6,*) '          '
      write(6,*) 'Enter initial x, y and z coordinates & initial u, v and w velocities'
      read(5,*) xo,yo,zo,u,v,w

      uuo = u
      vvo = v
      wwo = w

      write(6,*) '          '
      write(6,*) 'Enter the acceleration Ax, Ay and Az.'
      read(5,*) ax,ay,az

      write(6,*) '          '
      write(6,*) 'Enter the number of the target model you wish to use.'
      write(6,*) '    Crossing target model with acceleration          (1)'
      write(6,*) '    Side direction target model with acceleration    (2)'
      write(6,*) '    Radially inbound target model                    (3)'
      write(6,*) '    Maneuvering target model                         (4)'
      write(6,*) '    3-D Approach target model with acceleration      (5)'
      write(6,*) '    Crossing target model w/o acceleration           (6)'
      write(6,*) '    Side direction target model w/o acceleration     (7)'
      write(6,*) '    3-D Approach target model with acceleration      (8)'
      read(5,*) idec

      if(idec.eq.3.or.idec.eq.4) then
        write(6,*) '          '
        write(6,*) 'Enter theta and phi.'
        read(5,*) theta,phi
        theta = theta*(pi/180)
        phi = phi*(pi/180)
      else
```

```
      endif

      write(6,*) '          '
      write(6,*) 'Enter the number of the filtering system to use.'
      write(6,*) '    Benedict-Bordner filter              (1)'
      write(6,*) '    Alpha-Beta-Gamma filter              (2)'
      write(6,*) '    FCS filter                           (3)'
      read(5,*) jdec

      if(jdec.eq.2) then
        write(6,*) '          '
        write(6,*) 'Enter the tracking index GAMMA.'
        read(5,*) 'GAMMA
      else
      endif

      write(6,*) '              '
      write(6,*) 'Enter the desired time increments in seconds.'
      read(5,*) step
      write(6,*) '              '

      write(6,*) 'Enter the number of switches between target models.'
      read(5,*) nswtch
      write(6,*) '          '

      stime(nswtch+1) = 1.0e+12
      ndec(nswtch+1) = 1

      if(nswtch.le.0) goto 13

      do 12 m = 1,nswtch
        write(6,*) 'Enter the switch over time in seconds.'
        read(5,*) stime(m)
        write(6,*) '          '
        write(6,*) 'Enter the number of the target model to switch over to.'
        read(5,*) ndec(m)
        write(6,*) '          '
        write(6,*) 'Enter the new accelerations, if any, Ax, Ay and Az.'
        read(5,*) px(m),py(m),pz(m)
        write(6,*) '          '
   12 continue
```

```
13  continue

      write(6,*) 'Enter the load factor; if not maneuvering then any no. will do.'
      read(5,*) load

      write(6,*) '          '
      write(6,*) 'Enter the standard deviation for Theta & Phi in milliradians.'
      read(5,*) mr
      write(6,*) '        '
      write(6,*) 'Enter the standard deviation for Range in yards.'
      read(5,*) rr
      write(6,*) '              '

      write(6,*) 'Do you want to add this noise? y=1, n=0'
      read(5,*) ijk
      write(6,*) '              '

      write(6,*) 'Enter the number of data steps.'
      read(5,*) length
      write(6,*) '        '
      write(6,*) '         '

c
c    Filter Pre-Initialization Step #1
c

c    k = 1 (Step 1)

c    alpha = 1.0
c    beta = 0.0

      call convert(xo,yo,zo,xm,ym,zm,rb,mr,rr,1,n,ijk)

      xs(1) = xm(1)
      ys(1) = ym(1)
      zs(1) = zm(1)

      xp(2) = xs(1)
      yp(2) = ys(1)
      zp(2) = zs(1)

      us(1) = 0.0
```

```
      vs(1) = 0.0
      ws(1) = 0.0

      write(6,1) xs(1),ys(1),zs(1),1
      write(6,2) us(1),vs(1),ws(1)

c
c     Filter Pre-Initialization Step #2
c

c    k = 2

      alpha = 1.0
      beta = 1.0
      t = step

      r = sqrt(xs(1)**2+ys(1)**2+zs(1)**2)

      call target(xt,yt,zt,xo,yo,zo,u,v,w,ax,ay,az,r,theta,phi,idec,t,load,1,n,con,
     $ztime)

      call convert(xt,yt,zt,xm,ym,zm,rc,mr,rr,2,n,ijk)

      xs(2) = xp(2) + alpha*(xm(2)-xm(1))
      ys(2) = yp(2) + alpha*(ym(2)-ym(1))
      zs(2) = zp(2) + alpha*(zm(2)-zm(2))

      us(2) = us(1) + (beta/step)*(xm(2)-xm(1))
      vs(2) = vs(1) + (beta/step)*(ym(2)-ym(1))
      ws(2) = ws(1) + (beta/step)*(zm(2)-zm(1))

      xp(3) = xs(2) + step*us(2)
      yp(3) = ys(2) + step*vs(2)
      zp(3) = zs(2) + step*ws(2)

      write(6,1) xs(2),ys(2),zs(2),2
      write(6,2) us(2),vs(2),ws(2)

c
c     Begin main program loop.
c
```

```
      do 10 k = 3,length
        inct = k
        t = t+step

c
c     After target data has smoothed out , switch over to new target model once
c     time criteria has been satisfied and continue the switching process as
c     required.
c

        if(t.ge.stime(count)) then
              idec = ndec(count)
              con = stime(count)

              u = u + ax*del
              v = v + ay*del
              w = w + az*del

              uuo = u
              vvo = v
              wwo = w

              ax = px(count)
              ay = py(count)
              az = pz(count)

              if((count/2)*2.eq.count) then
                ztime = con-step
                xo = xt
                yo = yt
                zo = zt
              else
              endif

              count = count + 1
        else
              del = abs(t-con+step)
        endif

c
c     Solve for optimal Alpha and Beta.
c
```

```
      range = sqrt(xp(k)**2+yp(k)**2+zp(k)**2)

      call fcsfil(jdec,range,alpha,beta,gamma,step,mr,rr)

c
c     Predict mew radar scanning coordinates from current data.
c

      r = sqrt(xs(k-2)**2+ys(k-1)**2+zs(k-1)**2)

      call target(xt,yt,zt,xo,yo,zo,u,v,w,ax,ay,az,r,theta,phi,idec,t,load,k-1,n,
     $con,ztime)

c
c     Convert from Cartesian to Spherical, add noise and convert back.
c

      call convert(xt,yt,zt,xm,ym,zm,ra,mr,rr,k,n,ijk)

c
c     Process radar data through track filtering equations for smoothing and
c     updating of the velocities.
c

      xs(k) = xp(k) + alpha*(xm(k)-xp(k))
      ys(k) = yp(k) + alpha*(ym(k)-yp(k))
      zs(k) = zp(k) + alpha*(zm(k)-zp(k))

      us(k) = us(k-1) + (beta/step)*(xm(k)-xp(k))
      vs(k) = vs(k-1) + (beta/step)*(ym(k)-yp(k))
      ws(k) = ws(k-1) + (beta/step)*(zm(k)-zp(k))

      xp(k+1) = xs(k) + step*us(k)
      yp(k+1) = ys(k) + step*vs(k)
      zp(k+1) = zs(k) + step*ws(k)

      uuu = uuo + ax*(t-con)
      vvv = vvo + ay*(t-con)
      www = wwo + az*(t-con)

      vrs = sqrt(us(k)**2+vs(k)**2+ws(k)**2)
```

```
      vrp = sqrt(uuu**2+vvv**2+www**2)

      write(6,1) xs(k),ys(k),zs(k),inct
      write(6,2) xs(k),ys(k),zs(k)
      write(4,3) t,us(k)
      write(5,3) t,vs(k)
      write(6,3) t,ws(k)
      write(7,3) t,ut
      write(8,3) t,vt
      write(9,3) t,wt
      write(13,3) t,prs
      write(14,3) t,prp

      difu = ut-us(k)
      difv = vt-vs(k)
      difw = wt-ws(k)
      difpr = prs-prp

      write(10,3) t,difu
      write(11,3) t,difv
      write(12,3) t,difw
      write(15,3) t,difpr

1  format(1x,'Xs = ',f13.5,'Ys = ',f13.5,'Zs = ',f13.5,'Count = ',i6)
2  format(1x,Xus = ',f13.5,'Yvs = ',f13.5,'Zws = ',f13.5//)
3  format(1x,f13.5,2x,f13.5)

10  continue
11  continue

    stop
    end
```

Appendix C. Subroutines

```
      subroutine convert(xt,yt,zt,xm,ym,zm,r,mr,rr,k,n,ijk)

      real xt,yt,zt,xm(n),ym(n),zm(n),r,theta,phi,mr

c
c     This subroutine converts Cartesian coordinates to Spherical coordinates
c     ,adds noise to the Spherical coordinates and then converts back to
c     Cartesian coordinates.
c

      if(ijk.eq.1) then

c
c     Converts from Cartesian to Spherical coordinates.
c

        r = sqrt(xt**2+yt**2+zt**2)
        theta = atan2(yt,xt)
        phi = acos(zt/r)

c
c     Addition of white noise to spherical coordinates.
c

        call ran1(-1,value)
        call gasdev(-1,gas)

c
c     This adds noise to the R, Theta and Phi in spherical coordinates.
c     The noise is standard noise with standard deviation in milliradians.
```

```
c
          r = rr*value
          theta = theta + gas*mr/1000.
          phi = phi + gas*mr/1000.

c
c     Convert Spherical coordinates back to Cartesian coordinates.
c

          xm(k) = r*cos(theta)*sin(phi)
          ym(k) = r*sin(theta)*sin(phi)
          zm(k) = r*cos(phi)

        else

c
c     No noise addition.
c

          xm(k) = xt
          ym(k) = yt
          zm(k) = zt

        endif

        write(*,1) xm(k),ym(k),zm(k)
    1   format(1x,'Xn = ',f13.5,'Yn = ',f13.5,'Zn = ',f13.5)

        return
        end



        subroutine fcsfil(jdec,range,alpha,beta,gamma,step,mr,rr)

        real alpha,beta,gamma,increm,range,rrr,rr,mr,stem
        integer count,jdec

        if(jdec.eq.1) goto 1000
        if(jdec.eq.2) goto 2000
        if(jdec.eq.3) goto 3000
```

```
c
c     This section selects alpha from a range criteria
c

 1000 increm = 0.01
      count = 0
      alpha = 0.0

      rrr = sqrt(rr*2+range**2*(mr/1000.)**2)
      gamma = step**2*32.2/rrr

   11 continue

      beta = alpha*2/(2.0-alpha)
      test = sqrt(beta**2/(1.0-alpha))

      if(count.ge.4) then
        alpha = alpha-increm
        increm = increm/10.0
        count = count+1
        goto 11
      else
        alpha = alpha+increm
        goto 11
      endif

   21 continue

      write(6,2) alpha,beta
    2 format(1x,'Alpha = ',f13.5,'Beta = ',f13.5)

      goto 6000

c
c     This section solves for alpha and beta using a bracketting method once
c     given the tracking index "gamma".
c

 2000 increm = 0.01
      count = 0
      alpha = 0.0
```

```
 10  continue

     beta = alpha**2/(2.0-alpha)
     test = sqrt(beta**2/(1.0-alpha))

     if(count.ge.4) goto 20

     if(test.gt.gamma) then
       alpha = alpha-increm
       increm = increm/10.0
       count = count+1
       goto 10
     else
       alpha = alpha+increm
       goto 10
     endif

 20  continue

     write(6,1) alpha,beta
  1  format(1x,'Alpha = ',f13.5,'Beta = ',f13.5)

     goto 6000

c
c    This section selects Alpha & Beta from a range criteria
c

3000 if(range.ge.0.0.and.range.le.15000.0) alpha = .41
     if(range.gt.15000.0.and.range.le.30000.0) alpha = .31
     if(range.gt.30000.0.and.range.le.250000.0) alpha = .21

     beta = alpha**2/(2.0-alpha)

     write(6,3) alpha,beta
  3  format(1x,'Alpha = ',f13.5,'Beta = ',f13.5)

     goto 6000

6000 continue
```

```
      return
      end



      subroutine gasdev(ineg1,gas)

c
c     This program converts a normal distribution to a Gaussian distribution.  This
c     program utilizies the random number generator ran1(idum).  Two values
c     of the random number generator are required.  This used the Box-Jenkins
c     algorithim.
c

      data iset /0/

      if(iset.eq.0) then
 1      call ran1(ineg1,value1)
        call ran1(ineg1,value2)
        v1 = 2.0*value1-1.0
        v2 = 2.0*value2-1.0
        r = v1**2+v2**2

c
c     Prevents round off error giving numb >= 1
c

        if(r.ge.1.0) goto 1

        fac = sqrt(-2.0*alog(r)/r)
        gset = v1*fac
        gas = v2*fac
        iset = 1
      else
        gas = gset
        iset = 0
      endif

      return
      end
```

```
      subroutine ran1(idum,value)

c
c   This program is a random number generator that generates numbers
c   between 0.0 and 1.0.  The parameter idum is set to any negative value.
c

      real r(97)
      integer idum

      m1 = 259200
      ia1 = 7141
      ic1 = 54773
      rm1 = 1.0/float(m1)
      m2 = 134456
      ia2 = 8121
      ic2 = 28411
      rm2 = 1.0/float(m2)
      m3 = 243000
      ia3 = 4561
      ic3 = 51349

      data iff /0/

      if(idum.lt.0.or.iff.eq.0) then
        iff = 1
        ix1 = mod(ic1-idum,m1)
        ix1 = mod(ia1*ix1+ic1,m1)
        ix2 = mod(ix1,m2)
        ix1 = mod(ia1*ix1+ic1,m1)
        ix3 = mod(ix1,m3)

        do 11 j = 1,97
              ix1 = mod(ia1*ix1+ic1,m1)
              ix2 = mod(ia2*ix2+ic2,,m2)
              r(j) = (float(ix1)+float(ix2)*rm2)*rm1
   11   continue

      idum = 1

      else
```

```
      endif

      ix1 = mod(ia1*ix1+ic1,m1)
      ix2 = mod(ia2*ix2+ic2,m2)
      ix3 = mod(ia3*ix3+ic3,m3)
      j = 1+(97*ix3)/m3

      if(j.gt.97.or.j.lt.1) pause

      value = r(j)
      r(j) = (float(ix1)+float(ix2)*rm2)*rm1

      return
      end


      subroutine target(xt,yt,zt,xo,yo,zo,u,v,w,ax,ay,az,r,th,ph,idec,t,load,k,n,con,
     $ztime)

      real xt,yt,zt,xo,yo,zo,u,v,w,t,load,r,th,ph,ax,ay,az,con

c
c     This program uses a target model to predict the next set of radar pointing
c     coordinates to track the target.  Note:  "g" is in yards/sec/sec
c

c
c     Selection of target model for calculations.
c

      if(idec.eq.1) goto 100
      if(idec.eq.2) goto 200
      if(idec.eq.3) goto 300
      if(idec.eq.4) goto 400
      if(idec.eq.5) goto 500
      if(idec.eq.6) goto 520
      if(idec.eq.7) goto 540
      if(idec.eq.8) goto 560

c
c     Crossing target model with acceleration
c
```

```
 100  xt = xo+u*(t-ztime)+0.5*ax*(t-con)**2
      yt = yo+v*(t-ztime)+0.5*ay*(t-con)**2
      zt = zo

      goto 600

c
c    Side direction target model with acceleration
c

 200  xt = xo
      yt = yo+v*(t-ztime)+0.5*ay*(t-con)**2
      zt = zo

      goto 600

c
c    Radially inbound target model
c

 300  ro = sqrt(xo**2+yo**2+zo**2)
      vr = (xo*u+yo*v+zo*w)/ro

      xt = (ro+vr*t)*sin(ph)*cos(th)
      yt = (ro+vr*t)*sin(ph)*sin(th)
      zt = (ro+vr*t)*cos(ph)

      goto 600

c
c   Maneuvering target model
c

 400  ro = sqrt(xo**2+yo**2+zo**2)
      vr = (xo*u+yo*v+zo*w)/ro
      ww = load*10.73333/vr

      xt = (ro+vr*t)*sin(ph)*cos(ww*t)
      yt = (ro+vr*t)*sin(ph)*sin(ww*t)
      zt = (ro+vr*t)*cos(ph)
```

```
      goto 600

c
c   3-D Approach target model with acceleration
c

 500  xt = xo+u*(t-ztime)+0.5*ax*(t-con)**2
      yt = yo+v*(t-ztime)+0.5*ay*(t-con)**2
      zt = zo+w*(t-ztime)+0.5*az*(t-con)**2

      goto 600

c
c    Crossing target  model
c

 520  xt = xo+u*(t-ztime)
      yt = yo+v*(t-ztime)
      zt = zo

      goto 600

c
c    Side direction target model
c

 540  xt = xo
      yt = yo+v*(t-ztime)
      zt = zo

      goto 600

c
c   3-D Approach target model
c

 560  xt = xo+u*(t-ztime)
      yt = yo+v*(t-ztime)
      zt = zo+w*(t-ztime)

      goto 600
```

```
600  continue

     write(6,1) xt,yt,zt
   1 format(1x,'Xp = ',f13.5,'Yp = ',f13.5,'Zp = ',f13.5)

     return
     end
```

Appendix D. Program DRW

```
      program drw
c
c     This program is a low level driver which will generate a plot using QCAL
c     subroutines from higher level drivers.  This particular program will graph
c     2-D plots of one or two lines on the screen.
c
      real xarray(1002),yarray(1002)
      real xarra1(1002),yarra1(1002)

      character*34 xaxis,yaxis,yaxis2,figure
      character*34 subt

c
c     Data initialization
c

      write(6,*) '          '
      write(6,*) 'Enter 0 to print to the screen or 50 to print to a file for printing
     $on a LaserJet series II printer.'
      read(5,*) ldev
      write(6,*) '          '

      write(6,*) 'Do you want data symbols? y=1 n=0'
      read(5,*) l
      write(6,*) '          '

      if(l.eq.1) then
        write(6,*) 'The first line will have square symbols.'
```

```
      write(6,*) 'The second line (if you have one) will have triangle symbols.'
      write(6,*) '                '
      write(6,*) 'How often do you want data symbols?'
      read(5,*) lll
      write(6,*) '                '
    else
    endif

    write(6,*) 'Enter the number of lines to plot: 1 or 2'
    write(6,*) 'NOTE: More than three points/line must be used!'
    read(5,*) num
    write(6,*) '            '

    write(6,*) 'Enter the number of data points for line 1.'
    read(5,*) numb1

    if(num.eq.2) then
      write(6,*) '          '
      write(6,*) 'Enter the number of data points for line 2.'
      read(5,*) num2
    else
    endif

    write(6,*) '            '
    write(6,*) 'Enter the name of the figure.'
    read(5,'(a34)') figure

    write(6,*) '            '
    write(6,*) 'Enter the name of the x-axis.'
    read(5,'(a34)') xaxis

    write(6,*) '            '
    write(6,*) 'Enter the name of the y-axis.'
    read(5,'(a34)') yaxis

    write(6,*) '          '
    write(6,*) 'Enter a subtitle for the plot.'
    read(5,'(a34)') subt

c
c   Data prescaling routine
c
```

```
      write(6,*) '              '
      write(6,*) 'Enter the data in x, y pairs for line number 1.'
      write(6,*) '              '

      n = 2
      read(5,*) xarray(2),yarray(2)
      ymax1 = yarray(2)
      ymin1 = yarray(2)
      xmax1 = xarray(2)
      xmin1 = xarray(2)

      do 1 i = 3,numb1+1
        read(5,*) xarray(i),yarray(i)
        if(ymax1.le.yarray(i)) ymax1 = yarray(i)
        if(ymin1.ge.yarray(i)) ymin1 = yarray(i)
        if(xmax1.le.xarray(i)) xmax1 = xarray(i)
        if(xmin1.ge.xarray(i)) xmin1 = xarray(i)
        n = n+1
    1 continue
 1000 continue

      if(num.eq.2) then

        write(6,*) '              '
        write(6,*) 'Enter the data in x, y pairs for line number 1.'
        write(6,*) '              '

        n1 = 2
        read(5,*) xarra1(2),yarra1(2)
        ymax2 = yarra1(2)
        ymin2 = yarra1(2)
        xmax2 = xarra1(2)
        xmin2 = xarra1(2)

        do 2 i = 3,numb2+1
          read(5,*) xarra1(i),yarra1(i)
          if(ymax2.le.yarra1(i)) ymax2 = yarra1(i)
          if(ymin2.ge.yarra1(i)) ymin2 = yarra1(i)
          if(xmax2.le.xarra1(i)) xmax2 = xarra1(i)
          if(xmin2.ge.xarra1(i)) xmin2 = xarra1(i)
          n1 = n1+1
```

```
    2  continue
 1000  continue

       else
       endif

       if(num.eq.2) then
         if(ymax1.ge.ymax2) then
               ymax = ymax1
         else
               ymax = ymax2
         endif
         if(xmax1.ge.xmax2) then
               xmax = xmax1
         else
               xmax = xmax2
         endif
       else
         ymax = ymax1
         xmax = xmax1
       endif

       if(num.eq.2) then
         if(ymin1.le.ymin2) then
               ymin = ymin1
         else
               ymin = ymin2
         endif
         if(xmin.le.xmin2) then
               xmin = xmin1
         else
               xmin = xmin2
         endif
       else
         ymin = ymin1
         xmin = xmin1
       endif

       xarray(n+1) = xmax
       xarray(1) = xmin
       xarra1(n1+1) = xmax
       xarra1(n1) = xmin
```

```
      xarray(n+2) = 0.0
      xarray(n+3) = 0.0
      xarra1(n1+2) = 0.0
      xarra1(n1+3) = 0.0

      yarray(n+1) = ymax
      yarray(1) = ymin
      yarra1(n1+1) = ymax
      yarra1(1) = ymin

      yarray(n+2) = 0.0
      yarray(n+3) = 0.0
      yarra1(n1+2) = 0.0
      yarra1(n1+3) = 0.0

c
c    Initialize QCAL with logical number of output device
c

      if(ldev.eq.50) then
        call plots(-4,9600,ldev)
      else
        call plots(0,0,ldev)
      endif

c
c    Establish orgin
c

      if(ldev.eq.0) then
        call plot(0.5,0.5,-3)
      else
        call plot(0.7,1.5,-3)
      endif

c
c     Scale all following graph dimensions by a factor of one.
c

      call factor(1.0)
```

```
c
c    Compute scale factors so that x:y data pairs will plot within a 9" by 5" area.
c

        call scale(xarray,9.0,n+1,1)
        call scale(yarray,5.0,n+1,1)

        if(num.eq.2) then
          call scale(xarra1,9.0,n1+1,1)
          call scale(yarra1,9.0,n1+1,1)
        else
        endif

c
c    Draw x and y axes.
c

        call axis(0.0,0.0,xaxis,-34,9.0,0.0,xarray(n+2),xarray(n+3))
        call axis(0.0,0.0.,yaxis,34,5.0,90.0,yarray(n+2),yarray(n+3))

c
c    Plot y vs. x, line connecting data points and with a box symbol at every other
c    point.
c

        xarray(1) = xarray(2)
        xarray(n+1) = xarray(n)
        yarray(1) = yarray(2)
        yarray(n+1) = yarray(n)

        xarra1(1) = xarra1(2)
        xarra2(n1+1) = xarra1(n1)
        yarra1(1) = yarra1(2)
        yarra1(n1+1) = yarra1(n1)

        call line(xarray,yarray,n+1,1,lll,0)

        if(num.eq.2) call line(xarra1,yarra1,n1+1,1,lll,2)

c
c    Plot graph title
c
```

```
call symbol(0.8,5.6,0.21,figure,0.0,34)
call symbol(0.8,5.3,0.14,subt,0.0,34)

end
```

Appendix E. Program DRWR

```
      program drwr

c
c     This program is a low level driver which will generate a plot using QCAL
c     subroutines from higher level drivers.  This particular program will graph
c     2-D plots of one or two lines on the screen.
c

      real xarray(1002),yarray(1002)
      real xarra1(1002),yarra1(1002)

      character*34 xaxis,yaxis,yaxis2,figure
      character*34 subt

c
c     Data initialization
c

      write(6,*) '          '
      write(6,*) 'Enter 0 to print to the screen or 50 to print to a file for printing
     $on a LaserJet series II printer.'
      read(5,*) ldev
      write(6,*) '          '

      write(6,*) 'Do you want data symbols? y=1 n=0'
      read(5,*) l
      write(6,*) '          '

      if(l.eq.1) then
        write(6,*) 'The first line will have square symbols.'
```

```
      write(6,*) 'The second line (if you have one) will have triangle symbols.'
      write(6,*) '                '
      write(6,*) 'How often do you want data symbols?'
      read(5,*) lll
      write(6,*) '                '
    else
    endif

    write(6,*) 'Enter the number of lines to plot: 1 or 2'
    write(6,*) 'NOTE: More than three points/line must be used!'
    read(5,*) num
    write(6,*) '                '

    write(6,*) 'Enter the number of data points for line 1.'
    read(5,*) numb1

    if(num.eq.2) then
      write(6,*) '                '
      write(6,*) 'Enter the number of data points for line 2.'
      read(5,*) num2
    else
    endif

    write(6,*) '                '
    write(6,*) 'Enter the name of the figure.'
    read(5,'(a34)') figure

    write(6,*) '                '
    write(6,*) 'Enter the name of the x-axis.'
    read(5,'(a34)') xaxis

    write(6,*) '                '
    write(6,*) 'Enter the name of the y-axis.'
    read(5,'(a34)') yaxis

    write(6,*) '                '
    write(6,*) 'Enter a subtitle for the plot.'
    read(5,'(a34)') subt

c
c     Data prescaling routine
c
```

```
      n = 2
      read(3,*,end=1000) xarray(2),yarray(2)
      ymax1 = yarray(2)
      ymin1 = yarray(2)
      xmax1 = xarray(2)
      xmin1 = xarray(2)

      do 1 i = 3,1000
        read(3,*,end=1000) xarray(i),yarray(i)
        if(ymax1.le.yarray(i)) ymax1 = yarray(i)
        if(ymin1.ge.yarray(i)) ymin1 = yarray(i)
        if(xmax1.le.xarray(i)) xmax1 = xarray(i)
        if(xmin1.ge.xarray(i)) xmin1 = xarray(i)
        n = n+1
    1 continue
 1000 continue

      if(num.eq.2) then

        n1 = 1
        read(4,*,end=1001) xarra1(2),yarra1(2)
        ymax2 = yarra1(2)
        ymin2 = yarra1(2)
        xmax2 = xarra1(2)
        xmin2 = xarra1(2)

        do 2 i = 2,1000
          read(4,*,end=1001) xarra1(i),yarra1(i)
          if(ymax2.le.yarra1(i)) ymax2 = yarra1(i)
          if(ymin2.ge.yarra1(i)) ymin2 = yarra1(i)
          if(xmax2.le.xarra1(i)) xmax2 = xarra1(i)
          if(xmin2.ge.xarra1(i)) xmin2 = xarra1(i)
          n1 = n1+1
    2   continue
 1000   continue

      else
      endif

      if(num.eq.2) then
        if(ymax1.ge.ymax2) then
              ymax = ymax1
```

```
  else
        ymax = ymax2
  endif
  if(xmax1.ge.xmax2) then
        xmax = xmax1
  else
        xmax = xmax2
  endif
else
  ymax = ymax1
  xmax = xmax1
endif

if(num.eq.2) then
  if(ymin1.le.ymin2) then
        ymin = ymin1
  else
        ymin = ymin2
  endif
  if(xmin1.le.xmin2) then
        xmin = xmin1
  else
        xmin = xmin2
  endif
else
  ymin = ymin1
  xmin = xmin1
endif

dif1 = abs(xarray(2)-xmax)
dif2 = abs(xarray(2)-xmin)
dif3 = abs(xarra1(2)-xmax)
dif4 = abs(xarra1(2)-xmin)

if(dif2.le.dif1) then
  xarray(n+1) = xmax
  xarray(1) = xmin
else
  xarray(1) = xmax
  xarray(n+1) = xmin
endif
```

```
      if(dif4.le.dif3) then
        xarra1(n1+1) = xmax
        xarra1(1) = xmin
      else
        xarra1(1) = xmax
        xarra1(n1+1) = xmin
      endif

      xarray(n+2) = 0.0
      xarray(n+3) = 0.0
      xarra1(n1+2) = 0.0
      xarra1(n1+3) = 0.0

      dif1 = abs(yarray(2)-ymax)
      dif2 = abs(yarray(2)-ymin)
      dif3 = abs(yarra1(2)-ymax)
      dif4 = abs(yarra1(2)-ymin)

      if(dif2.le.dif1) then
        yarray(n+1) = ymax
        yarray(1) = ymin
      else
        yarray(1) = ymax
        yarray(n+1) = ymin
      endif

      if(dif4.le.dif3) then
        yarra1(n1+1) = ymax
        yarra1(1) = ymin
      else
        yarra1(1) = ymax
        yarra1(n1+1) = ymin
      endif


      yarray(n+2) = 0.0
      yarray(n+3) = 0.0
      yarra1(n1+2) = 0.0
      yarra1(n1+3) = 0.0

c
c     Initialize QCAL with logical number of output device
```

```
c
      if(ldev.eq.50) then
        call plots(-4,9600,ldev)
      else
        call plots(0,0,ldev)
      endif

c
c    Establish orgin
c

      if(ldev.eq.0) then
        call plot(0.5,0.5,-3)
      else
        call plot(0.7,1.5,-3)
      endif

c
c     Scale all following graph dimensions by a factor of one.
c

      call factor(1.0)

c
c    Compute scale factors so that x:y data pairs will plot within a 9" by 5" area.
c

      call scale(xarray,9.0,n+1,1)
      call scale(yarray,5.0,n+1,1)

      if(num.eq.2) then
        call scale(xarra1,9.0,n1+1,1)
        call scale(yarra1,9.0,n1+1,1)
      else
      endif

c
c    Draw x and y axes.
c

      call axis(0.0,0.0,xaxis,-34,9.0,0.0,xarray(n+2),xarray(n+3))
```

```
      call axis(0.0,0.0.,yaxis,34,5.0,90.0,yarray(n+2),yarray(n+3))

c
c    Plot y vs. x, line connecting data points and with a box symbol at every other
c    point.
c

      xarray(1) = xarray(2)
      xarray(n+1) = xarray(n)
      yarray(1) = yarray(2)
      yarray(n+1) = yarray(n)

      xarra1(1) = xarra1(2)
      xarra2(n1+1) = xarra1(n1)
      yarra1(1) = yarra1(2)
      yarra1(n1+1) = yarra1(n1)

      call line(xarray,yarray,n+1,1,lll,0)

      if(num.eq.2) call line(xarra1,yarra1,n1+1,1,lll,2)

c
c   Plot graph title
c

      call symbol(0.8,5.6,0.21,figure,0.0,34)
      call symbol(0.8,5.3,0.14,subt,0.0,34)

      end
```

References

1. Benedict, T. R., and Bordner, G. W., "Systhensis of an Optimal Set of Radar Track-While-Scan Smoothing Equations," IRE Transaction on Automatic Control, Vol. AC-7, No. 4, July 1962, pp. 27-32

2. Kalata, P. R., "Optimal $\alpha-\beta$ Filtering," Internal Correspondance, Nov. 1962

3. Castella, F. R., and Dunnebacke, F. G., "Analytical Results for the X-Y Kalman Tracking Filter," IEEE Transaction on Aerospace and Electronic Systems, Vol. AES-10, No. 6, Nov. 1974, pp. 891-895

4. Trunk, G. V., and Wilson, J. D., "Tracking Filters for Multiple-Platform Radar Integration," NRL Report 8087, Dec. 1976, pp. 1-15

5. Gray, J. E., and Murray, W. J., "Analysis of Alpha-Beta Filters for Potential Application in Antiair Kill Evaluation Improvements and as Acceleration Dectors," NSWC Report TR 89-269, Jan, 1990

6. Bar-Shalom, Yaakov, and Fortmann, Thomas E., "Tracking and Data Association," Academic Press, INC., 1988

7. Bryson, Jr., Arthur E., and Ho, Yu-Chi, "Applied Optimal Control," Bliadell Publishing Company, 1969

Vita

The author was born on August 22, 1966 in Martinsville, Virginia to Mr. and Mrs. Leo H. Hoffman. At Martinsville High School, he developed an interest in math and science which directed him to the field of engineering. He was accepted into the College of Engineering in 1984 and completed his undergraduate studies in 1988. Upon graduation, he was accepted into the graduate program of Master of Engineering in Aerospace Engineering to continue his studies specializing in the field of aerodynamics and computational fluid dynamics. Upon completion of his degree, the author will begin employment with Naval Air Development Center in Warminster, PA.

Leo Henry Hoffman, II