IDA PAPER P-2935

# FLEXIBLE ARCHITECTURES FOR SENSOR FUSION IN THEATER MISSILE DEFENSE

Gabriel Frenkel



April 1994



*Prepared for*
The Ballistic Missile Defense Organization



INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, Virginia 22311-1772

IDA Log No. HQ 93-44777

## DEFINITIONS

IDA publishes the following documents to report the results of its work.

### Reports

Reports are the most authoritative and most carefully considered products IDA publishes. They normally embody results of major projects which (a) have a direct bearing on decisions affecting major programs, (b) address issues of significant concern to the Executive Branch, the Congress and/or the public, or (c) address issues that have significant economic implications. IDA Reports are reviewed by outside panels of experts to ensure their high quality and relevance to the problems studied, and they are released by the President of IDA.

### Group Reports

Group Reports record the findings and results of IDA established working groups and panels composed of senior individuals addressing major issues which otherwise would be the subject of an IDA Report. IDA Group Reports are reviewed by the senior individuals responsible for the project and others as selected by IDA to ensure their high quality and relevance to the problems studied, and are released by the President of IDA.

### Papers

Papers, also authoritative and carefully considered products of IDA, address studies that are narrower in scope than those covered in Reports. IDA Papers are reviewed to ensure that they meet the high standards expected of refereed papers in professional journals or formal Agency reports.

### Documents

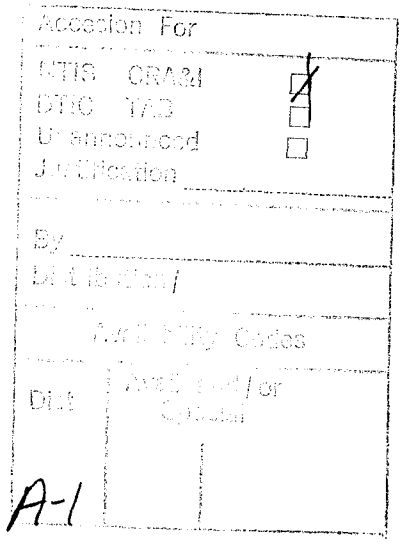
IDA Documents are used for the convenience of the sponsors or the analysts (a) to record substantive work done in quick reaction studies, (b) to record the proceedings of conferences and meetings, (c) to make available preliminary and tentative results of analyses, (d) to record data developed in the course of an investigation, or (e) to forward information that is essentially unanalyzed and unevaluated. The review of IDA Documents is suited to their content and intended use.

The work reported in this document was conducted under contract MDA 903 89 C 0003 for the Department of Defense. The publication of this IDA document does not indicate endorsement by the Department of Defense, nor should the contents be construed as reflecting the official position of that Agency.

IDA PAPER P-2935

# FLEXIBLE ARCHITECTURES FOR SENSOR FUSION IN THEATER MISSILE DEFENSE

Gabriel Frenkel




April 1994



INSTITUTE FOR DEFENSE ANALYSES
Contract MDA 903 89 C 0003
Task T-R2-597.12

# PREFACE

This work was sponsored by the Ballistic Missile Defense Organization (BMDO) in support of an overall effort to define a feasible ballistic missile/command, control, and communications ($BM/C^3$) defense architecture and to recommend architectural extensions that will accommodate mid- and long-term $BM/C^3$ needs.

The extremely helpful comments, suggestions, and review by Dr. Oliver Drummond of the Hughes Missile Systems Company and Dr. Amnon Dalcher of the Institute for Defense Analyses (IDA) are gratefully acknowledged.

# CONTENTS

# FIGURES



# TABLES

# I. INTRODUCTION

Designing an effective Theater Ballistic Missile Defense (TBMD) Command and Control structure is one of the key ongoing Ballistic Missile Defense Office (BMDO) activities. At the onset, timely and accurate tracking were recognized as key requirements. To satisfy these requirements, sensor fusion and the associated flexible control structure become paramount. At present, efforts on sensor fusion, associated Operational Control and Combat Command elements design, and procedures and algorithms for tracking and intercept support are based on the substantial past achievements of a series of efforts sponsored by Strategic Defense Initiative Office (SDIO). Nevertheless, some key questions need further clarification, and this report attempts to shed light on topics in a number of broad areas:

- Methods of sensor fusion and associated algorithms: their advantages and disadvantages and their dynamic utilization in the battle scenario. Issues of track fusion vs. data fusion, which appear regularly in the community, are of particular interest.
- Communication loads of particular sensor fusion architectures and associated operational control.
- Effective utilization of sensor fusion for intercept support and for burnout point estimation.

This paper addresses the simplest possible case: a single missile on a ballistic trajectory tracked by two radar sensors in a flat earth model. Later, we will demonstrate that this case is particularly conducive for providing valuable answers in the broad areas listed above. This paper is based on a number of simplifying assumptions, and some of them might warrant special attention in other studies. For example,

- Neither false signals nor persistent clutter is considered.
- Since the scenarios contain only a single target, issues about closely spaced objects and data association do not arise.
- All analyses and modeling are based on a linearized model, i.e., an Extended Kalman Filter (EKF), but errors caused by linearization are not considered. [The author's opinion is that the errors incurred as a result of this

approximation are minimal because of the smooth (ballistic) trajectories considered.]

- A pure ballistic target trajectory is used. Tumbling, gravitational anomalies, and other similar effects are not included.

Extending the results obtained in this paper to multiple maneuvering targets and infrared (IR) sensors is a natural next step. A valuable tool for accomplishing this is the Sensor Fusion Architecture Model (SFAM) developed for this effort. SFAM, which is extremely flexible and easy to operate, generated the graphs in this paper. The Appendix contains the complete code for SFAM, including input data for a typical run.

# II. DISCUSSION OF SENSOR FUSION

## A. THE USES AND DIFFICULTIES OF SENSOR FUSION

The first electronic sensor for tracking was the radar developed for monitoring enemy aircraft during the Battle of Britain in 1940. For many years, its basic function remained unchanged: a self-contained unit that performed the dual actions of active sensing and signal processing (tracking) at a single location. The resultant habit of perception of basic unified sensor/tracker elements had a major impact on our formulation of coordinated theater operations, led to all kind of problems such as the prolonged controversy over data fusion vs. track fusion, and, in general, generated a rigidity of thinking that hindered the development of truly creative and flexible control structures.

To enlarge the field of possibilities, we must dissociate the sensor from the tracker. Figure 2-1 illustrates this new perception. The scenario consists of a number of sensors, S, and filters or trackers, F, which may or may not be collocated.

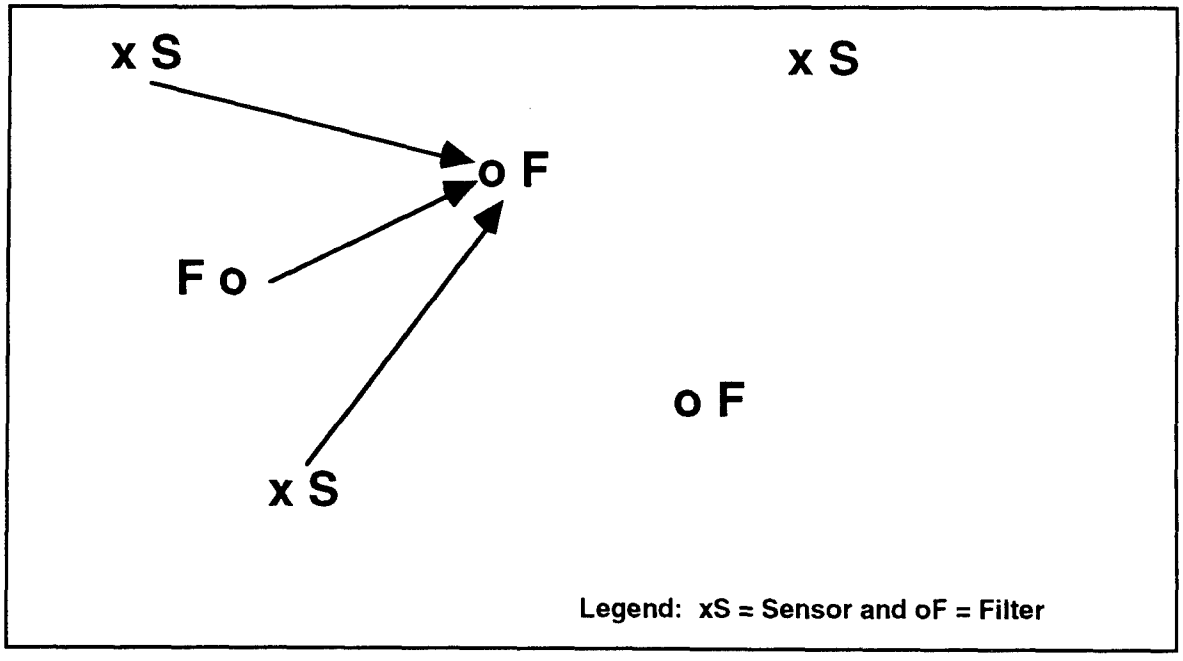



**Figure 2-1. Flexible Sensor Fusion**

*To have effective, dynamically flexible and adaptable sensor fusion, each tracking filter must be capable of receiving at any given time either a measurement update from any sensor or a State Vector (SV) i.e., track update from any other filter. This capability is vital to the implementation of flexible sensor fusion architectures.*

At a first glance, this flexibility might not seem attractive. Complex correlation problems seem to lead to complex calculations, to predefined structures in which the flexibility is lost and to heavy communications loads. However, the benefits of such flexible operations are numerous. Some of the more important ones are enhanced tracking accuracy from additional measurements averaging and from geographic diversity, efficient resource utilization, more effective intercept support and reduced communication load.

## B. FLEXIBLE SENSOR FUSION ALGORITHMS

In Figure 2-1, we assume that the tracker is a recursive filter, which is updated from either a sensor measurement or a track from another filter. An EKF is assumed, i.e., the operation consists of the alternation of time updates (i.e., prediction) and measurement updates. Until now, we have distinguished between two seemingly mutually exclusive alternatives: data fusion, which occurs when a filter input consists of measurements from more than one sensor, and track fusion, which occurs when the estimates of two filters, each operating with its own sensor and measurements, are combined. In the mode of operation shown in Figure 2-1, this distinction disappears.

The effect of a track and measurement on the EKF is fundamentally different. A track contains all the information of the past measurements from which it was derived. In statistical terminology, a track is a sufficient statistic for those measurements. Thus, when another EKF is updated with that track, the result is exactly what it would have been if it were updated by all the measurements from which it arose. By contrast, a measurement input has the value of only that particular data element. This difference will be amply illustrated in the computer runs presented in Section II.

To understand how the EKF input will be handled, we must look at the different considerations depending on whether it is a track or a measurement.

### 1. EKF Update With a Sensor Measurement

Figure 2-2 shows the algorithm for updating the EKF from a sensor. Two important departures from the situation that is usually encountered are as follows:

$Z(t_{k+1})$ = measurement received from a sensor
$t_{k+1}$ = time at which measurement was performed
$t_k$ = time of last prior filter update
$\Phi$ (1) = process transition matrix over unit time
$\Phi$ (k+1/k) = process transition matrix from time $t_k$ to $t_{k+1}$

Define the process model for the update interval
$X(k+1) = \Phi(1)^d * X(k) + g(d)$
where
$d = t_{k+1} - t_k$
and
g(d) = gravity vector function = $g*t^2/2$

Predict X at time $t_{k+1}$

$$\hat{X}(k+1/k) = \Phi(1)^d * \hat{X}(k/k) \quad (1)$$

Compute the measurement mapping matrix $H(t_{k+1})$. Its terms are given by
$H_{ij}(t_{k+1}) = dZ(t_{k+1})_i/dx_j$

evaluated at $X = \hat{X}(k+1/k)$.

The estimate updated by the measurement is given by

$$\hat{X}(k+1) = \hat{X}(k+1) + K(k+1)*(Z(t_{k+1}) - H(t_{k+1})* \hat{X}(k+1/k)) \quad (2)$$

where K(k+1) is the Kalman gain given by
$K(k+1) = P(k+1/k)*H(t_{k+1})'*inv(P(k+1/k)*H(t_{k+1})'+Theta)$
Theta is the sensor measurement error covariance in sensor coordinates

The a-priori and a-posteriori error covariance matrices are computed recursively:

$$P(k+1/k) = \Phi(1)^d * P(k/k) * \Phi(1)^{d'} \quad (3)$$

$$P(k+1) = (I - K(k+1)*H(t_{k+1}))*P(k+1/k) \quad (4)$$

**Figure 2-2. Algorithm for Updating an EKF With a Sensor Measurement**

- Because the measurement may come from any one of a number of unsychronized sensors, the update interval is variable from update to update.
- Because of target and perhaps sensor motion, the matrix describing how target position is mapped into measurements varies from update to update and must be recomputed.

In Figure 2-2, the process model for the variable time interval "d" is given by Eq. (1). The process transition matrix over unit time is known and constant. The new update time is available as a time tag that is appended to the new measurement by the sensor from which it is sent. The measurement mapping matrix H is derived from the equations of

the transformation from the sensor coordinate system to the common coordinate system. The components of the measurement Z (e.g., azimuth, elevation and range) are functions of the Cartesian coordinates $x_i$, i = 1,2,3. This function is evaluated in Eq. (4) at the predicted target position. If the sensor is moving, its position must be available with the measurement. Even if the sensor is static, the measurement mapping matrix H must be computed using Eq. (4) at each update, since the target is moving. This computation could be performed either at the sensor or at the recipient. A common coordinate system is implicit in any sensor fusion architecture, and a number of decisions must be made (i.e., the type of coordinate system and where to perform the coordinate transformations). Different measurements may come from different sensors, and their position is needed in Eq. (4) as they are interleaved.

In summary, measurement updates involve two key points: (1) updating an EKF with a measurement requires knowledge of the sensor position, which is to be used for the appropriate transformations and (2) for asynchronous operations, the 1-second transition matrix is raised to a power equal to the update interval.

## 2. EKF Update With a Track From Another EKF

As mentioned previously, a truly flexible fusion architecture mandates the capability of handling a track from another EKF as just another measurement. However, we must then consider the correlation of the various random quantities involved.

The operation of an EKF requires that the errors in subsequent measurements be uncorrelated. If this condition is fulfilled, the error in a new measurement is also uncorrelated with the estimate before update. However, when a track from another EKF is handled as a measurement, this condition, in general, does not apply.

### a. Correlation From Plant Noise

This situation arises when two EKFs track the same target, whose motion is assumed to have a random component. The outputs of two filters having the same random input will be correlated. The correlation is in general quite complex because it depends on the geometry and times of all past updates of both filters.

### b. Correlation From Common Sensor Data

If the two filters were updated from the same measurement, the error in that measurement will propagate in both filters. Thus, the source of this type of correlation is the set of measurements common to both filters.

### c. Correlation From Common Initialization Source

If the EKFs were started from the same source, the effect is the same as that of a common measurement.

### d. Correlation From Repeated Updates From the Same EKF

The errors in subsequent estimates of an EKF are correlated. This violates the condition that the sequence of measurements used for update be white.

We might be tempted to ignore the correlation and update the filter as if the error were uncorrelated. This is not advisable, however, as shown by the simple example of Figure 2-3, where the weighted sum of two estimates is taken as if they were uncorrelated and the resultant variance is compared with the better of the two estimates. When the correlation is high and one of the variances is much larger, the deterioration is appreciable. The possibility of ending up with a poorer track after fusion advises against ignoring the correlation.

The design of a flexible architecture in which any sequence of sensor data and tracks may be used for updates must consider the following:

- If the mission is limited to defense against a ballistic trajectory, plant noise is absent as a source of correlation.
- Since the measurement errors from different sensors as well as from the same sensor at different times are independent, if two tracks are derived from two separate sensors, the tracks are independent and easy to combine as a weighted sum.
- Suppose that two EKFs (KF1 and KF2) are present and have errors that are independent of one another. If KF1 is updated with an SV from KF2, the errors in the two KFs then become correlated, and subsequent updates from KF2 must account for this correlation.
  - The error correlation between KF1 and KF2 before the second update is a complex function of the geometry at all intervening times at which KF2 was updated with sensor measurements.

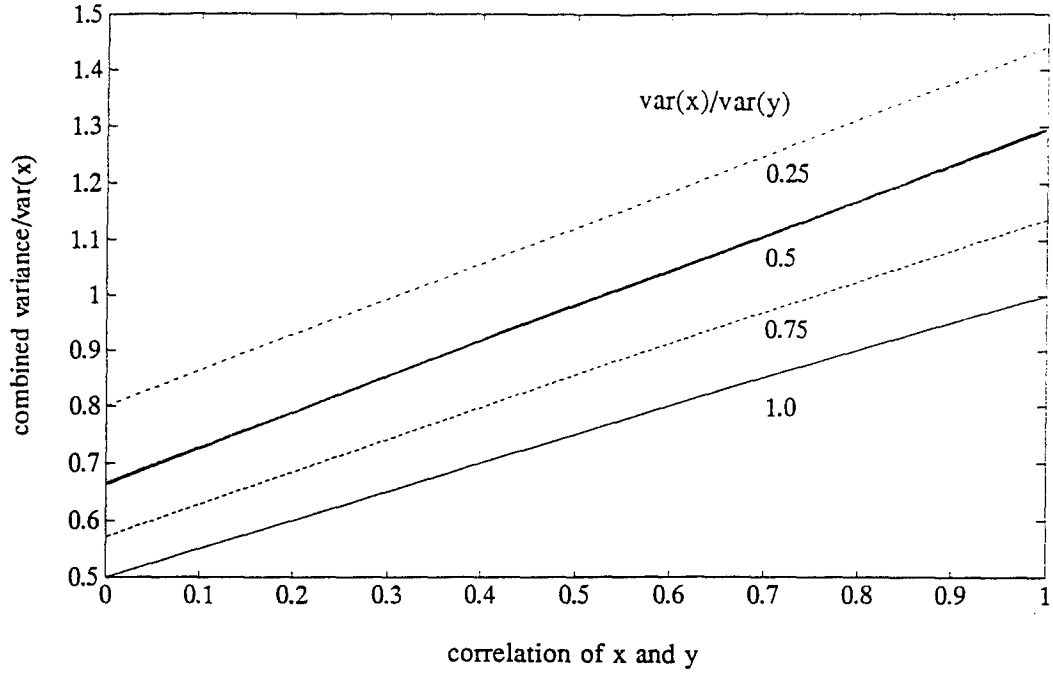



**Figure 2-3. Effects of Ignored Correlation**

Computing the correlation could be quite burdensome both in computation load and in the communication requirements. The algorithm presented in Figure 2-4 circumvents these difficulties and yields an optimum estimate in a simple, straightforward manner. This algorithm is based on the following idea: given a track and its covariance sent to KF1 from KF2, an equivalent uncorrelated measurement and its covariance can be computed if the prior such track/covariance pair is known. These are equivalent in the sense that, when used as a measurement input and its covariance with the priors for an update, the estimate and error covariance matrix are actually received. They are used for updating the recipient EKF, as if they were actual measurements. As will be shown presently, the measurement mapping matrix is taken as the identity matrix.

The algorithm shown in Figure 2-4 presupposes that no plant noise is present and that the two filters were initialized from independent sources. The proof is straightforward. Using the standard EKF recursion, with the measurement mapping matrix as the identity matrix [**Note:** Equations (5), (6), (7), and (8) are in Figure 2-4]:

$$P_{new} = P_{pre} - P_{pre} * inv( P_{pre} + Theta) * P_{pre} \quad , \tag{9}$$

$$K = P_{pre} * inv(Theta + P_{pre}) \quad , \tag{10}$$

$$X_{new} = X_{pre} + K * (Z - X_{pre}) \quad . \tag{11}$$

EKF1, EKF2 = filter to be updated and filter which is the source of the update

$\hat{X}_{old}$, $P_{old}$ = SV estimate and its error covariance at time $t_{old}$ sent to KF1 from KF2

$\hat{X}_{new}$, $P_{new}$ = next state vector estimate and its error covariance from KF2 to KF1

When a track and its covariance are sent from KF2 to KF1 for the first time use them as a measurement and error covariance matrix in the EKF equations. At all subsequent times:

$$\hat{X}_{pre} = \Phi(1)^{d} * \hat{X}_{old} + g(d) \quad (5)$$

$$P_{pre} = \Phi(1)^{d} * P_{old} * (\Phi(1)^{k})' \quad (6)$$

where

$\Phi(1)$ = Transition matrix of the process for unit time

$d = t_{new} - t_{old}$

$g(d)$ = gravity vector function

$$\text{Theta} = P_{pre} * \text{inv}(P_{pre} - P_{new}) * P_{pre} - P_{pre} \quad (7)$$

$$Z = (\text{Theta} + P_{pre}) * \text{inv}(P_{pre}) * (\hat{X}_{new} - \hat{X}_{pre}) + \hat{X}_{pre} \quad (8)$$

Use Z as a measurement with mapping matrix H = I and measurement error covariance matrix Theta in the standard EKF equations to update KF1

**Figure 2-4. Algorithm for Updating an EKF From Another EKF**

Solving for Theta and Z yields Eq. (7) and Eq. (8) which concludes the proof. We must remember that this algorithm is valid—in the sense of yielding an optimum estimate—only if the SVs to be combined do not share any measurements. Two notable exceptions are as follows: (1) if both filters were originally updated from the same source or (2) if at some previous time these filters were updated using an SV from the same external source. An extension of these results to these important cases is presently under study and will be presented in a subsequent report.

Figure 2-5 offers proof of the correctness of the algorithm in Figure 2-4. Two filters, KF1 and KF2, receive inputs from two different sensors, S1 and S2, respectively. A third filter, KF3, receives the same measurements from sensors S1 and S2, and a fourth filter, KF4, receives the inputs from S1, and at 95, 175, and 235 sec and a track from KF2 that is used as an update based on the algorithm just described. As indicated in Figure 2-5, after KF3 and KF4 are updated from KF2, their error covariances are identical. This happens because the track sent over is a sufficient statistic for all past measurements from S2; hence, once processed, KF3 and KF4 have the same information. For example, at 175 sec, KF4 should have the output it would have had if it had it received all measurements up to that time from both S1 and S2. Subsequently, KF4 deteriorates relative to KF3. However, after each update from KF2, the tracks of KF3 and KF4 become identical.

Table 2-1 illustrates the details of the algorithm's implementation. At each EKF, a table (the "lastfrom table") is maintained. This table has one entry for each EKF from which a track was used in the past as an update. The entry contains the identification (ID) of the sender, the time tag, track (SV), and its covariance. For example, $P_i(T_j/T_k)$ is the covariance of $X_i$, namely the SV sent from EKF No.i, which was an estimate of the track at time $T_j$ based on data up to time $T_k$. When a new track is received from an EKF with the same ID, the appropriate line of the table is extracted and used in Eq. (7) and Eq. (8) , and the new track replaces the previous one in the table.

## C. THE SENSOR FUSION ARCHITECTURE MODEL (SFAM)

SFAM was developed to evaluate various candidate approaches to sensor fusion. The MATLAB® software package, which is used for coding, is an interactive software package for scientific and numeric computations. MATLAB® was selected because it has strong capabilities for matrix manipulations and graphics. SFAM has a number of desirable features:

- It is modular and easy to modify or extend.
- It is user friendly.
- It is versatile and flexible in terms of the number of filters, the sequence of inputs (sensors and other EKFs) and timing, and various geometries and parameters.
- It has good screen graphics and plotting capabilities.

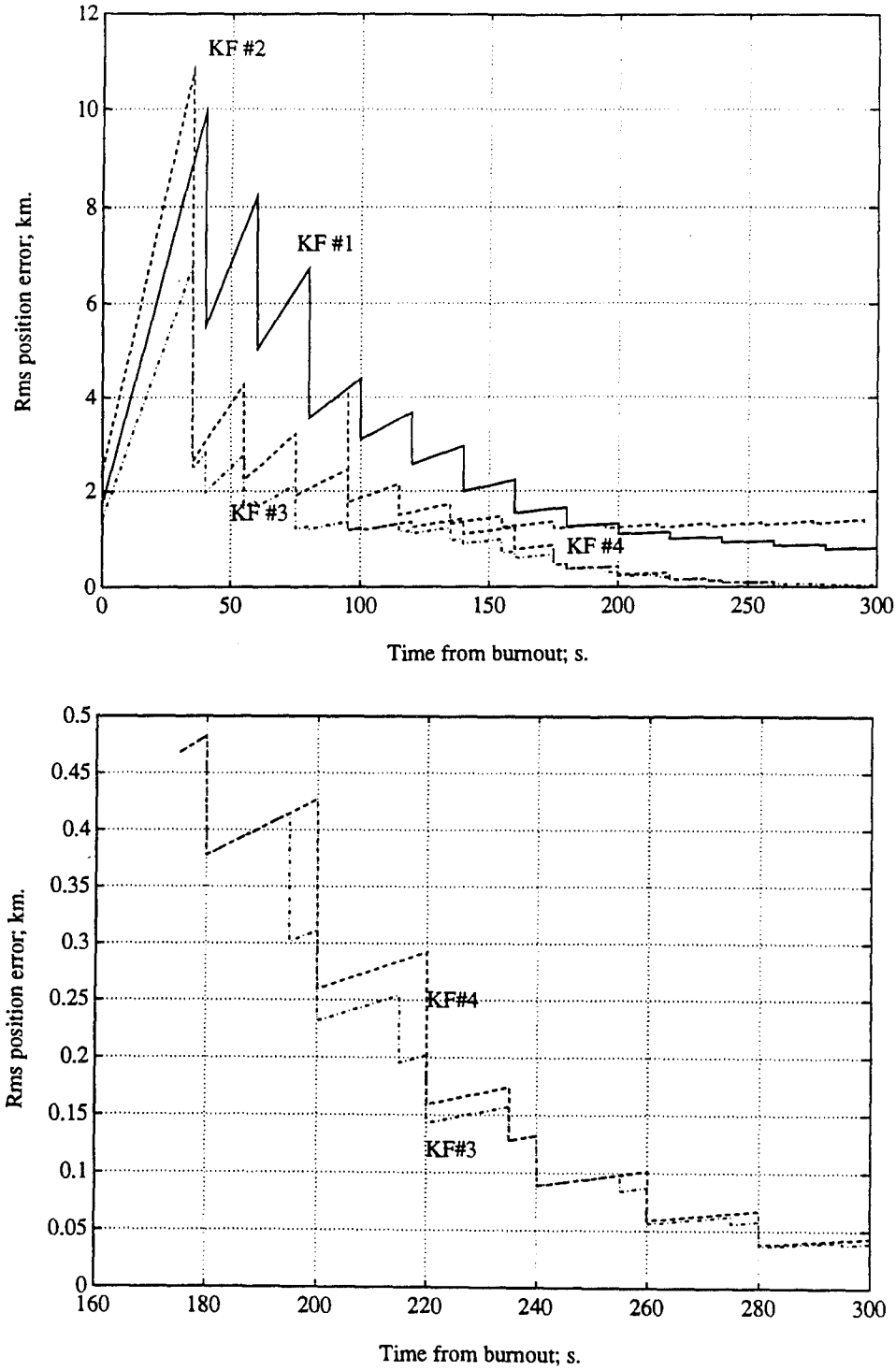



**Figure 2-5. Position Error of Measurement Updates (KF3) and Track Updates (KF4)**

Table 2-1. Lastfrom Table at KF1

| From | Time | X | P |
|---|---|---|---|
| KF2 | Tlast2 | X2(Tlast2/Tlast2) | P2(Tlast2/Tlast2) |
| • | • | • | • |
| • | • | • | • |
| • | • | • | • |
| KFn | Tlastn | Xn(/Tlastn/Tlastn) | Pn(Tlastn/Tlastn) |
| • | • | • | • |

Figure 2-6 shows the hierarchy of subroutines. Each subroutine, except the disk operating system (DOS) files XLATE.EXE and MATPRINT.BAT, is in a separate American Standard Code for Information Interchange (ASCII) file with the extension .M. This extension is needed so that the subroutine can be executed in MATLAB®. Three file names end with a one or two-digit number, for example PRMTRS13.M. These files must be edited for each run, and the run number, e.g., 13, must be appended to the file name. In Figure 2-6, XX stands for the run number.

FUSESHXX is the shell of SFAM. It calls first PRMTRSXX to read in the parameters of run No. XX. Subsequently, it calls FOREMAN once for each update of an EKF. After the updates are completed, FUSESHXX calls STAT, which processes the results and saves them in the temporary file RUN.M. While still in MATLAB®, STAT calls on XLATE.EXE, which is a DOS routine that formats the data and saves this data in INTFCEXX.M. Finally, STAT calls PLTCOMXX.M, which displays the results graphically on the screen and saves the graphic file in TEMP.MET. This completes the run. While still in MATLAB® or after leaving MATLAB®, MATPRINT.BAT can be called as a DOS command. This routine sends the contents of TEMP.MET to the printer.

**PRMTRSXX** is edited for each run for the desired parameters. These parameters fall into five categories:

- Configuration (number of sensors, number of EKFs, dimension of SVs)
- Sensors (coordinates, measurement error covariance expressed in sensor coordinates)
- Process model (initial SV of each EKF, transition matrix)
- Initial conditions (initial SV of each EKF, initial error covariance)
- Initial values of some run parameters.

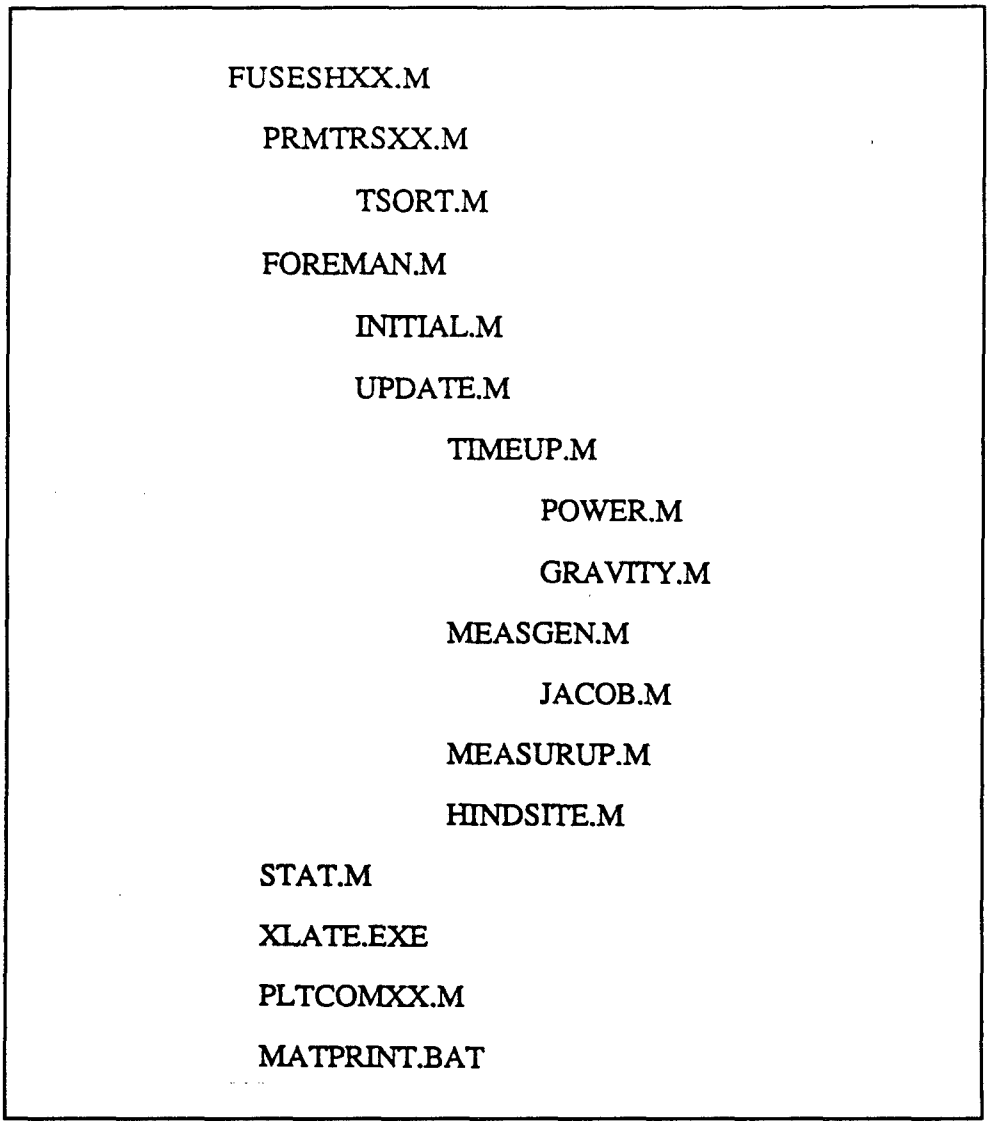



**Figure 2-6. SFAM Hierarchy**

The Ttable (see Table 2-2 as an example) has one row for each update event. The first column is the time at which it occurs, the second column is the ID of the EKF being updated, and the third column is a code denoting the source of the update, which could be a sensor, another EKF, or an external source for initialization. When the table is prepared, the rows can be in random time order. Calling TSORT rearranges the rows in chronological order.

FOREMAN takes the next line of Ttable and determines whether an EKF must be initialized (in which case INITIAL is called) or updated (in which case UPDATE is called).

Table 2-2. Ttable

| Times | KF ID | Source | Times | KF ID | Source |
|---|---|---|---|---|---|
| 0 | 1 | 21 | 175 | 2 | 2 |
| 0 | 2 | 22 | 180 | 1 | 1 |
| 35 | 2 | 2 | 180 | 3 | 1 |
| 40 | 1 | 1 | 195 | 2 | 2 |
| 55 | 2 | 2 | 200 | 1 | 1 |
| 60 | 1 | 1 | 200 | 3 | 1 |
| 75 | 2 | 2 | 215 | 2 | 2 |
| 80 | 1 | 1 | 220 | 1 | 1 |
| 80 | 3 | 31 | 220 | 3 | 1 |
| 95 | 2 | 2 | 235 | 2 | 2 |
| 95 | 3 | 42 | 240 | 1 | 1 |
| 100 | 1 | 1 | 240 | 3 | 1 |
| 100 | 3 | 1 | 255 | 2 | 2 |
| 115 | 2 | 2 | 260 | 1 | 1 |
| 120 | 1 | 1 | 260 | 3 | 1 |
| 120 | 3 | 1 | 275 | 2 | 2 |
| 135 | 2 | 2 | 280 | 1 | 1 |
| 140 | 1 | 1 | 280 | 3 | 3 |
| 140 | 3 | 1 | 295 | 2 | 2 |
| 155 | 2 | 2 | 300 | 1 | 1 |
| 160 | 1 | 1 | 300 | 3 | 1 |
| 160 | 3 | 1 | | | |

In the latter case, the relevant variables, such as the appropriate covariance or prior estimate, are identified. First, UPDATE calls TIMEUP for a time update of the filter. Subsequently, UPDATE calls MEASGEN, which generates the appropriate measurement parameters. These parameters are next used by MEASURUP for a measurement update. UPDATE also calls on HINDSITE, which estimates the target burnout point, i.e., the SV at $t = 0$.

Three subroutines shown in Figure 2-6 but not mentioned yet are POWER, which raises the transition matrix for the process for 1 second to the appropriate power (e.g., d) to

compute the transition matrix for "d" sec; GRAVITY, which computes the effect of gravity for any time interval; and JACOB, which computes the Jacobian of the polar to Cartesian transformation involved, as in Eq. (4).

## D. SENSOR FUSION ARCHITECTURES AND STRATEGIES

The flexibility achieved through the data fusion and track fusion algorithms described previously can be exploited in the design of a command center and associated architecture. Some key elements and considerations will be illustrated with examples obtained through the SFAM.

Figure 2-7 shows the scenario used in all computer runs. The two sensors (Sensor #1 and Sensor #2) are located on the ground, 340 km apart. The target trajectory is in a plane parallel to the sensors' baseline at a distance of about 115 km. The burnout is at 50-km altitude at a point nearer to Sensor #2. The flight time is approximately 300 sec. During the second half of the flight, the target is closer to Sensor #1. In all runs, the sensor measurement error covariance matrix is diagonal, with angular errors in both dimensions of 1.7E-4 $rad^2$ and range error of 1E-4 $m^2$.

### 1. Improving Accuracy

Figure 2-8 shows the performance of three EKFs in the scenario of Figure 2-7. Both sensors generate measurements that are unsynchronized, 20 sec apart. KF #1 (KF1) uses all measurements of Sensor #1 (S1), and KF2 uses all measurements of Sensor #2 (S2). KF3 receives all measurements of S1 and S2 as well as the two independent initialization inputs from external sources used by KF1 and KF2. Thus, KF3 has all the data used by KF1 and KF2. The root mean squared (rms) position error of all three KFs exhibits the typical sawtooth function. After each update, the error increases linearly with time as we would expect when future position is computed through prediction from present position estimates. As soon as a new measurement is processed, the error decreases.

Figure 2-8 is a good illustration of a number of general conclusions:

- Data fusion always helps, at least conceptually, although it is not necessarily practical. Theoretically, obtaining data from additional sensors always improves performance, even if the quality of the additional data is poor.
- If the sensors are unsynchronized, the time interval between updates is reduced and this results in improved accuracy between updates. In fact, having the data interleaved is preferable.

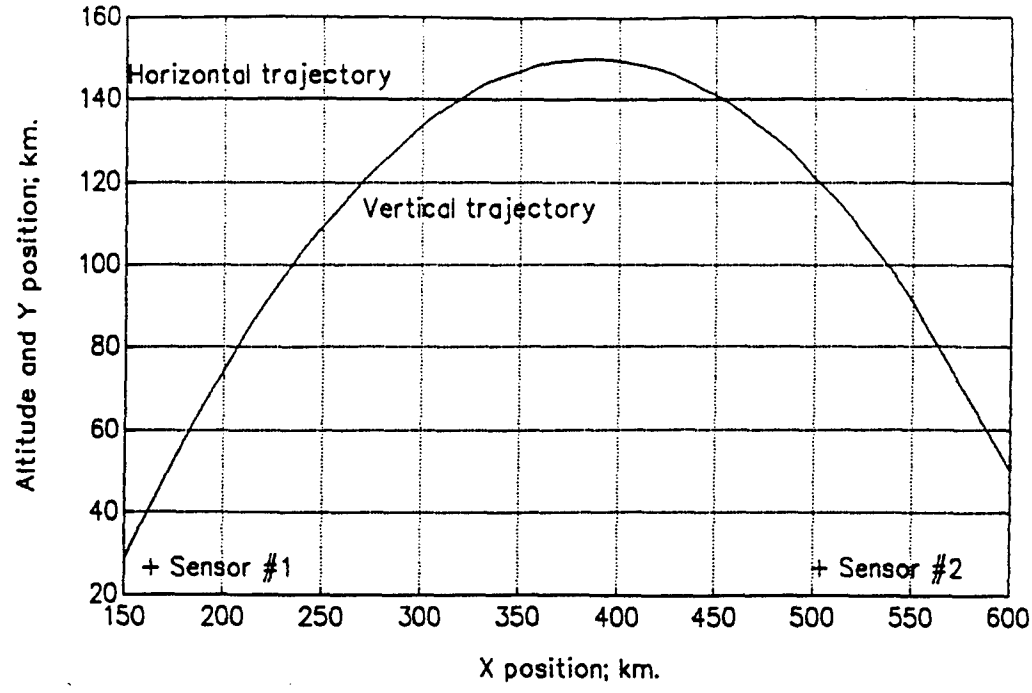



**Figure 2-7. Target Trajectory**

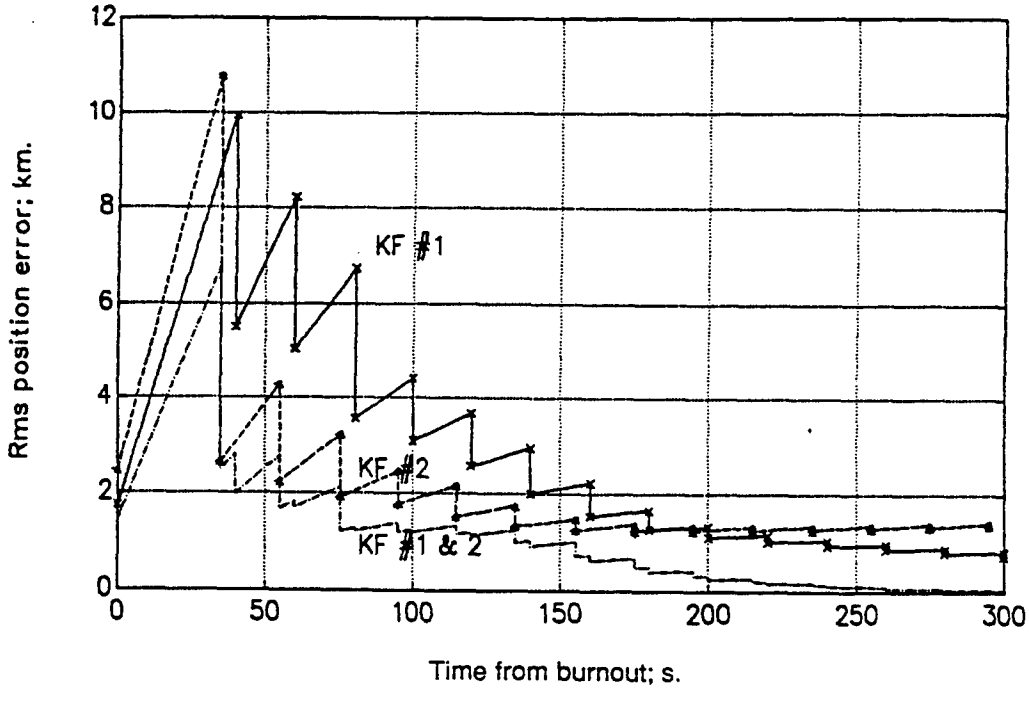



**Figure 2-8. Position Error in Combining All Measurements**
***(Inputs: KF1 from S1; KF2 from S2; KF3, all measurements from S1 and S2)***

- Geographic diversity is an important concept that deserves further elaboration because it offers major advantages. The impact of the spatial configuration on position estimation is well known. For example, in some systems using sensors for location, a concept known as Geographic Dilution of Position (GDOP) is used to designate some deleterious effects, such as error blowup when the target is on the baseline of the two sensors.

Figure 2-8 shows the merits of geographic diversity, and a numerical example will illustrate these merits further. We will assume two estimates, (1) and (2), with the following total mean squared error, $\sigma^2_t$, caused by independent errors in the Cartesian coordinates x, y, and z.

$$\sigma^2_x(1) = 10 \quad \sigma^2_y(1) = 1 \quad \sigma^2_z(1) = 1 \quad \sigma^2_t(1) = 12$$

$$\sigma^2_x(2) = 1 \quad \sigma^2_y(2) = 10 \quad \sigma^2_z(2) = 1 \quad \sigma^2_t(2) = 12$$

A weighted average reduces the total error to 2.3. To simplify this example, we will assume a suboptimum combiner. Instead of a weighted average, the suboptimum combiner will take—for each component—the estimate with the smaller variance, yielding

$$\sigma^2_x = 1 \qquad \sigma^2_y = 1 \qquad \sigma^2_z = 1 \qquad \sigma^2_t(1) = 3$$

In either case, the substantial improvement is because the large errors are in different components in the two estimates.

The effect is exploited in Synthetic Aperture Radar (SAR), where a moving radar becomes equivalent to looking at the target from more than one direction. The basic principle, which holds in our case, is that a sensor usually has directional error characteristic, namely, a direction in which the error is greatest and the principal contributor to sensor errors. Combining two sensors whose directions of maximum error are different improves substantially the weighted average of the two estimates. This improvement results because in any direction there is an estimate that is good. This is evident in Figure 2-8 where at 200 sec, for example, the two sensors produce rms position errors on the order of 1 km, but their combined measurements produce an error on the order of 0.1 km.

Similar conclusions can be reached when investigating the effects of track fusion (see Figure 2-5 ). An important point is that the track sent as an update does not have to be better than the track already in KF4. In fact, the total rms position error could be substantially larger and still improve the accuracy dramatically. What matters is the distribution of the error components between the two tracks that are to be combined.

These examples have some important implications concerning the reporting rules. They suggest that using only the rms position error as a criterion for reporting is not advisable. As an alternative, we should create a better rule on which to base the decision to report. However, since such a rule would require some calculation, we might as well compute the weighted average of the local estimate and the estimate available to the network as a whole. Then, we would discover whether making the local estimate available would result in sufficient improvement.

## 2. Flexible Counterattack Against Launchers

When the objective is to destroy the launcher by estimating the launch point from the track, we must be able to respond before the launcher is moved. Obtaining the best results entails a delicate balance between a swift response and accuracy in pinpointing the launcher. Accuracy, in turn, requires some time delay. An obvious strategy is to retaliate at the earliest possible time at which a reasonably good estimate is available. This strategy is illustrated in Figure 2-9. For simplicity, we eliminated the sawtooth between updates. Rather, the curve shows at any given time the error after an update at that time. As before, KF1 and KF2 receive updates at 20-sec intervals from S1 and S2, respectively. The geometry is the same as that in Figure 2-3. KF3, KF4, and KF5 receive the same inputs as KF1, and each one receives an additional measurement from S2 at 95 sec, 175 sec, and 255 sec, respectively.

We can draw a number of important conclusions from Figure 2-9. First, even a single measurement from a second sensor improves the accuracy of the estimate dramatically. Geographic diversity is the source of this improvement. Second, of lesser importance is how recently the update from the second sensor occurred. For example, if the goal is to shoot when the estimate of burnout position has an error that is less than 0.1 km rms, two things are necessary: (1) one must wait until 260 sec and (2) at some previous time, one measurement had to be incorporated from S1. This measurement should be dated as close as possible to 260 sec.

The second example, shown in Figure 2-10, has the same parameters as those in Figure 2-5. KF4 receives additional help in the form of SVs used for update from KF2. After every such input, an abrupt decrease occurs in the error of the estimate of burnout point, the times and resultant rms position errors being 0.42 km at 95 sec, 0.13 km at 175 sec, and 0.10 km at 225 sec. This phenomenon suggests a shoot-look-shoot strategy,

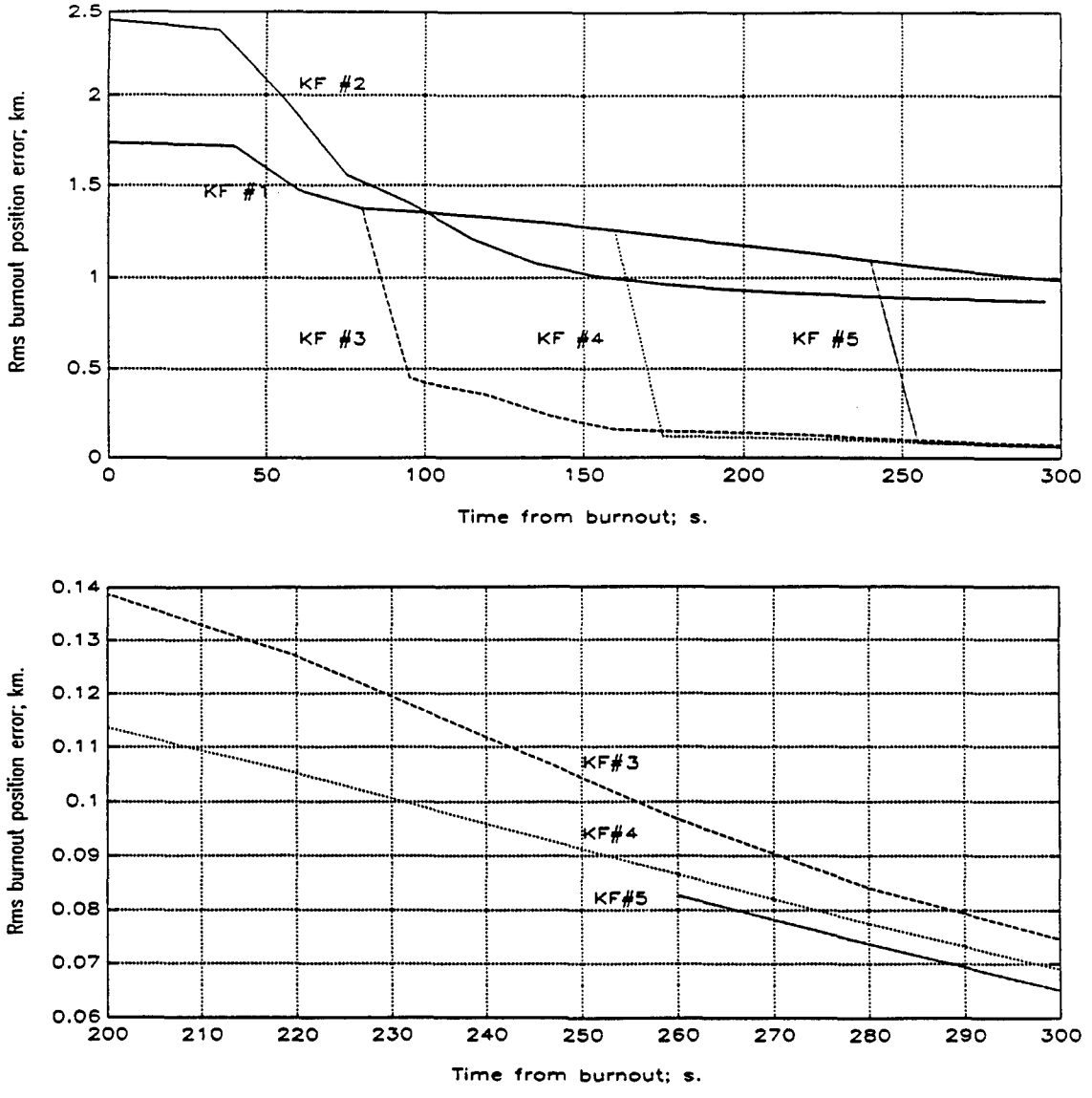



**Figure 2-9. Effect of Single Measurement From a Second Sensor on Burnout Position Estimate**

***(Inputs: KF1 from S1; KF2 from S2; KF3, 4, 5 from S1 and one input from S2)***

namely, the estimate is successively improved for one more shot. The extraneous help could come from more than one source. While the additional delay with each try diminishes the chance of success because the launcher might move, the increased accuracy enhances the chance of success.

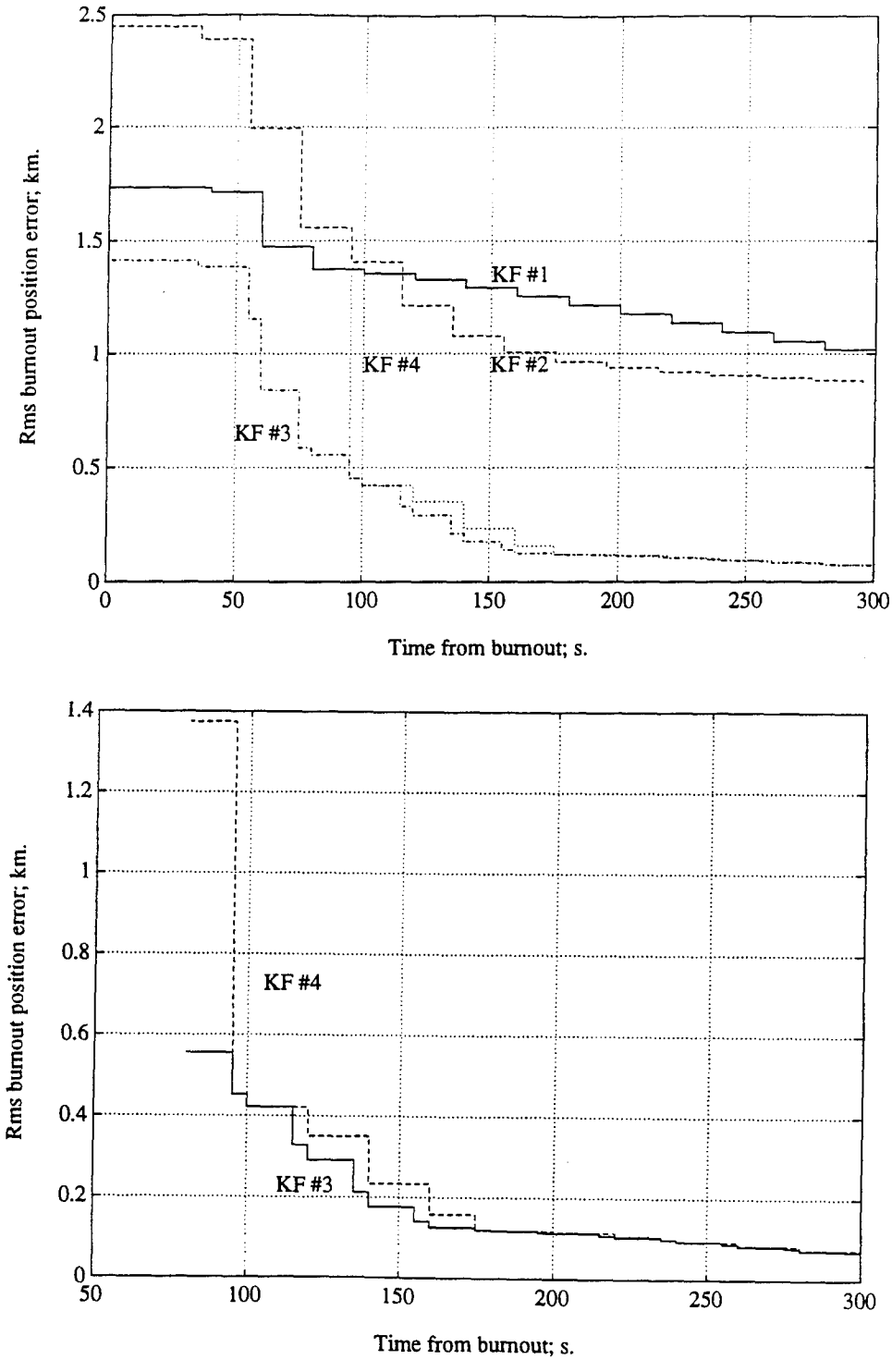



**Figure 2-10. Impact of Repeated Updates From a Track on Burnout Position Error**

***(Inputs: KF1 from S1; KF2 from S2; KF3 from S1 and S2; KF4 from S1 and three Inputs from KF2)***

## 3. Efficient Sensor Management

For the conditions of this study, the three potential benefits from the dynamic allocation of sensor resources are as follows:

- Enhanced performance
- Freeing resources for other tasks or even eliminating some sensors
- Enhanced reliability because of redundancy.

A tradeoff between the first two benefits usually exists. The sensor resources are used either for improving performance or are released for other uses. The third benefit results from the involvement of more than one sensor. If one sensor fails, tracking can still continue, even though the performance deteriorates.

Figures 2-11 and 2-12 were generated on the same computer run. KF1 and KF2 receive data from S1 and S2, respectively. KF3 receives data from S1 first and switches to S2 halfway through the scenario. All three filters are updated at the same rate of one measurement each 20 sec. As evidenced from the figures, KF3 achieves substantial improvement in performance compared to the other two filters because of geographic diversity, with no increase in the number of measurements. Presumably, during the first and second half of the scenario, S2 and S1, respectively, do not have to attend to this target and are freed for other tasks.

Since geographic diversity is so effective, could it be used to preserve the performance achievable with a single sensor at reduced measurement input rate? The answer is illustrated in Figure 2-13 where KF3 received interleaved data from S1 and S2 but at a total rate of only half the rate of KF1 and KF2. Even at this reduced rate, KF3 performs as well as the other two filters and toward the end of the run substantially better.

Thus, by mixing sensors, the data rates can be reduced for the same performance. To use such an approach effectively, we must develop resource allocation algorithms.

## 4. Dynamic Intercept Support

Successful intercept requires accurate prediction of future target position; therefore, the accuracy of the velocity estimate is important. For tracking requirements, the process of ballistic missile (BM) intercept can be divided into six steps:

1. The launch is detected, and track is initiated. At this point, the EKF is initialized.

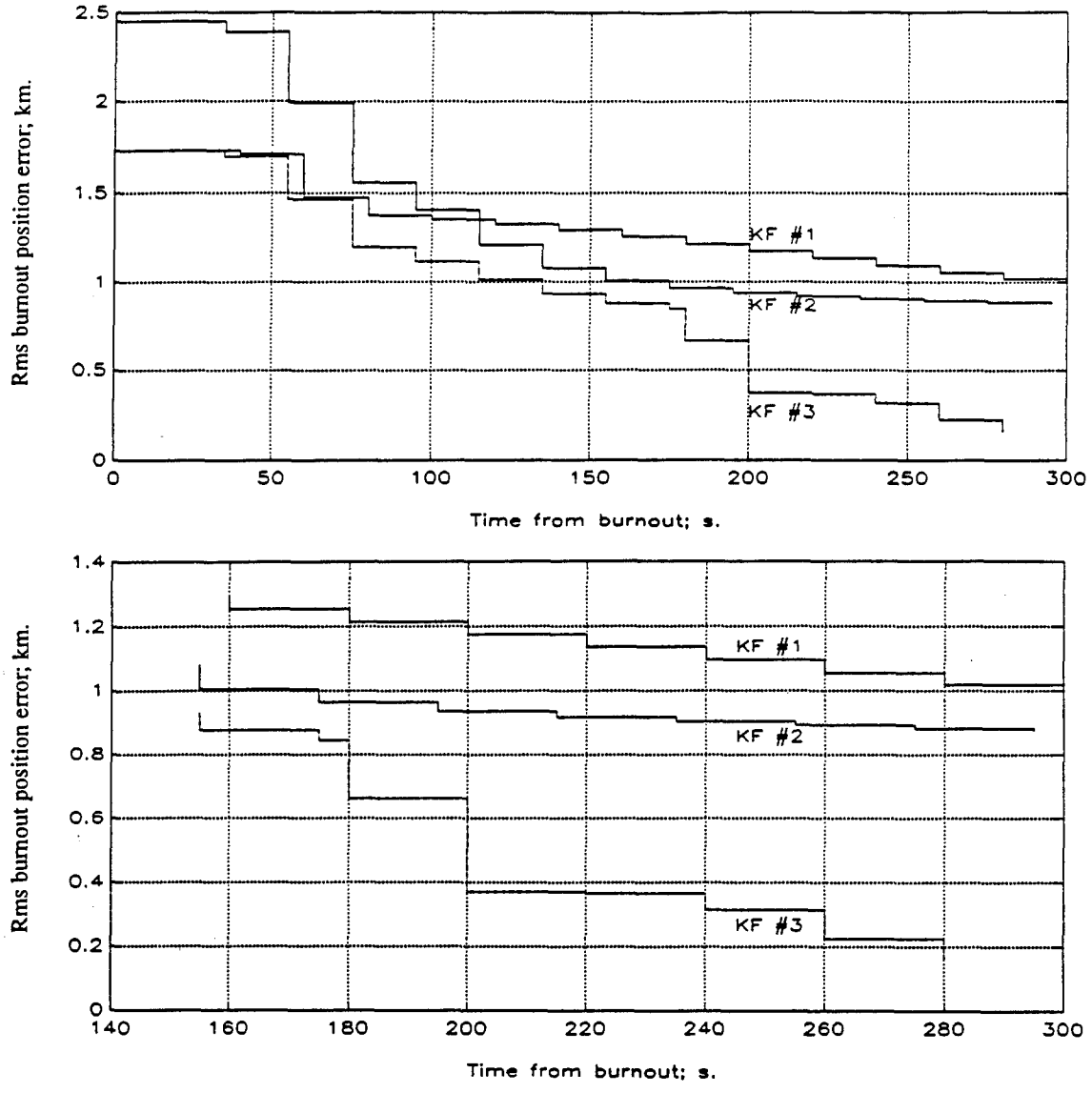



**Figure 2-11. Impact of Geographic Diversity on Burnout Position Error**
***(Inputs: KF1 from S1; KF2 from S2; KF3 first from S2, then from S1 starting at 180 sec)***

2. A reliable track is established. At this stage, high accuracy is not vital. More crucial is the estimation of orbital parameters from which general direction of flight, impact point and launch point can be estimated. The appropriate sensor, tracker, and interceptor suite also can be chosen.

3. The interceptor is selected and launched. At this point, an accurate track is important because it is a major factor in the probability of kill ($P_k$).

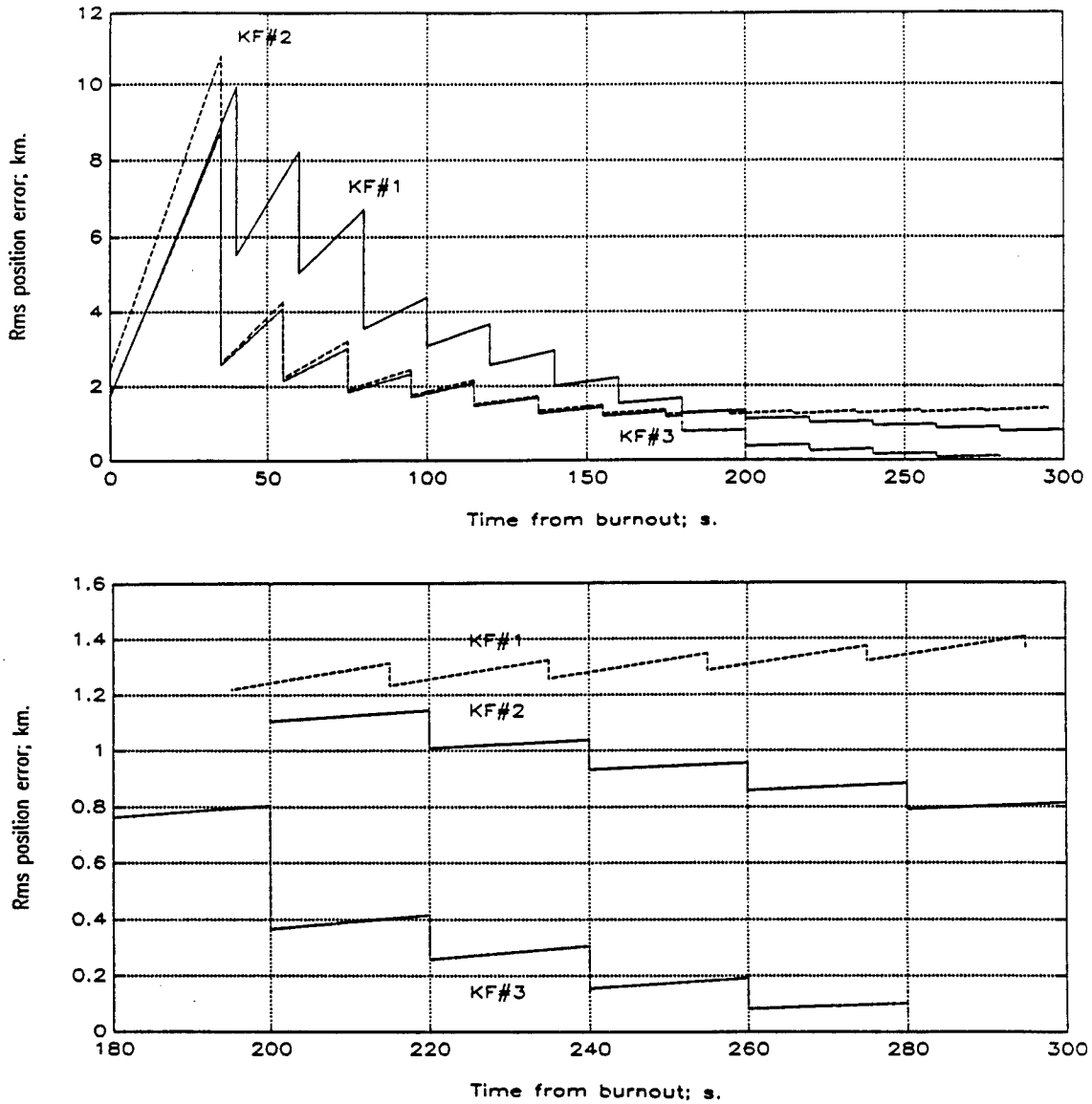



**Figure 2-12. Impact of Geographic Diversity on Position Error**
***(Inputs: KF1 from S1; KF2 from S2; KF3 first from S2, then from S1 starting at 180 sec)***

4. An in-flight update is received. Some interceptors have the capability of accepting track corrections thus greatly improving the $P_k$.
5. A kill assessment is made.
6. If shoot-look-shoot operation and handover to another tracker are considered, steps 3–6 are repeated.

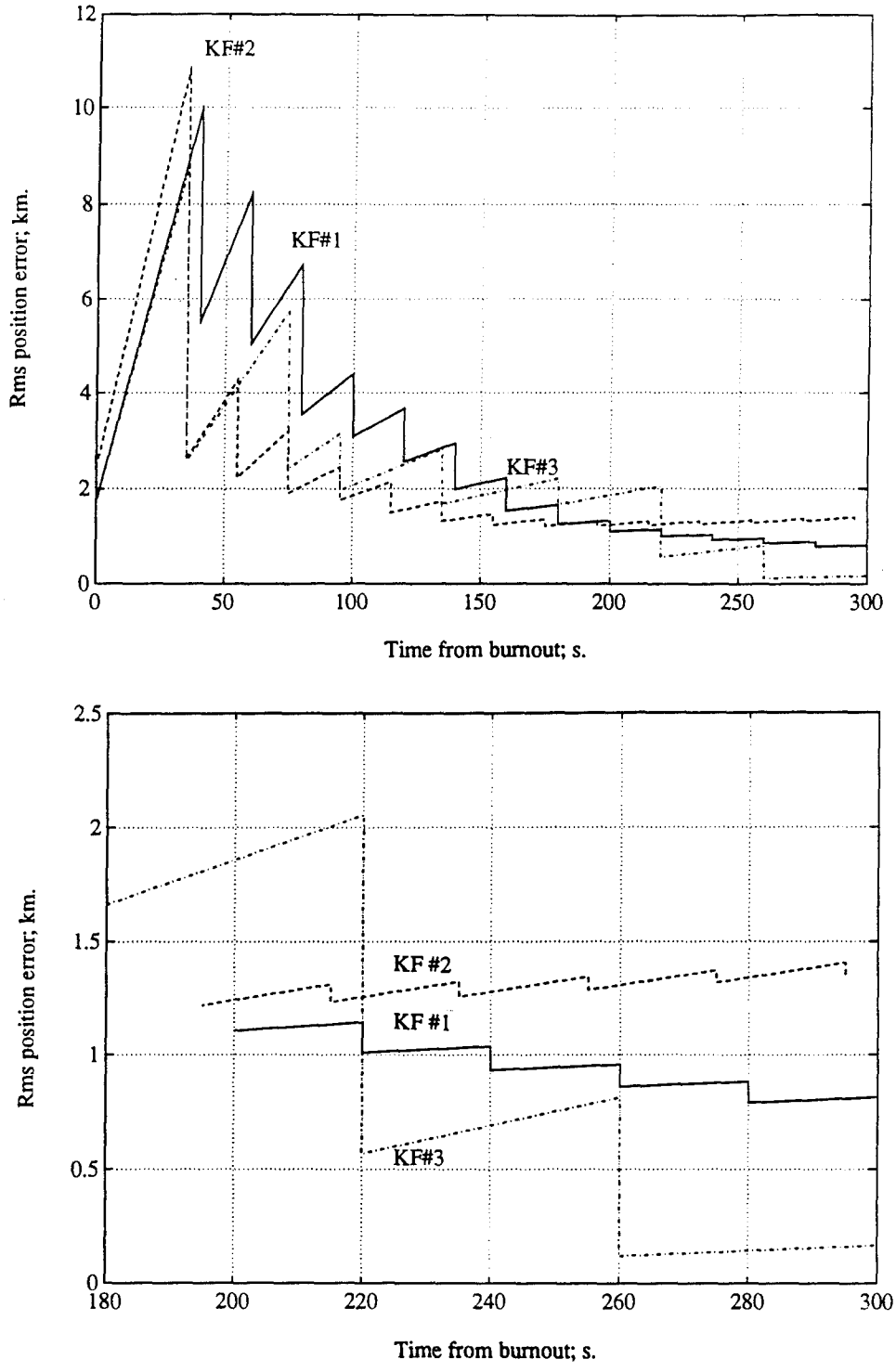



**Figure 2-13. Impact of Reduced Data Rate on Position Error**
***(Inputs: KF1 from S1; KF2 from S2; KF3 from S1 and S2 at reduced rate)***

While tracking is an integral part at each step, sensor fusion is not. In step 1 and step 2, sensor fusion is not necessary because high accuracy is not vital. By contrast, it can be a powerful tool in step 3. This situation is illustrated in Figure 2-14, which was produced in the same run as Figure 2-9. As evidenced in Figure 2-14, receiving a measurement from a second sensor, in this case S2, can improve accuracy dramatically. The obvious strategy is to obtain such a measurement just before interceptor launch. Similarly, the timing of this external help does matter but—at least in this example—is less critical. For example, if an interceptor launch is to occur at 220 sec for an intercept 20 sec later, fusion with KF2 at 95 sec would result in a rms prediction error of 270 m at 240 sec. If the fusion occurs at a time closer to the interceptor launch, at 175 sec, the predicted rms error at intercept equals 180 m.

For the in-flight update of step 4 or the shoot-look-shoot operation of step 6, repeated updates with a track from another EKF offer versatility and improved performance. This is illustrated in Figure 2-5, where KF4 is updated three times with a track from KF2. These three updates can come from different sources and can be used for either shoot-look-shoot or in-flight update. Thus, the whole process of intercept acquires a much more flexible and dynamic quality, in two different ways: (1) updates with another track are performed on command, at critical moments and (2) the availability of additional, but not necessarily foreseen sources of update, become part of the decision process. For example, if a new track suddenly becomes available, this might point to the desirability of a second shot. The element of timing also can be used in a much more flexible manner. As discussed previously and shown in Figure 2-5, each update from a track has the full value of all prior measurements—the longer the wait, the more measurements. However, the rewards of enhanced accuracy come at the expense of time delays.

### 5. Reduced Communication Requirements

Table 2-3 illustrates the tradeoffs available between performance and communication requirements. In all seven cases we assume that an EKF receives data, measurements, or tracks from another EKF. This table shows the communications load incurred by sending the data, as well as the accuracy ranking resulting from fusing the data with the recipient's track. A track message consists of a total of 28 words: 6 for the SV, 21 for the error covariance matrix, and 1 for time. A measurement message consists of four words: the three polar coordinates and time.

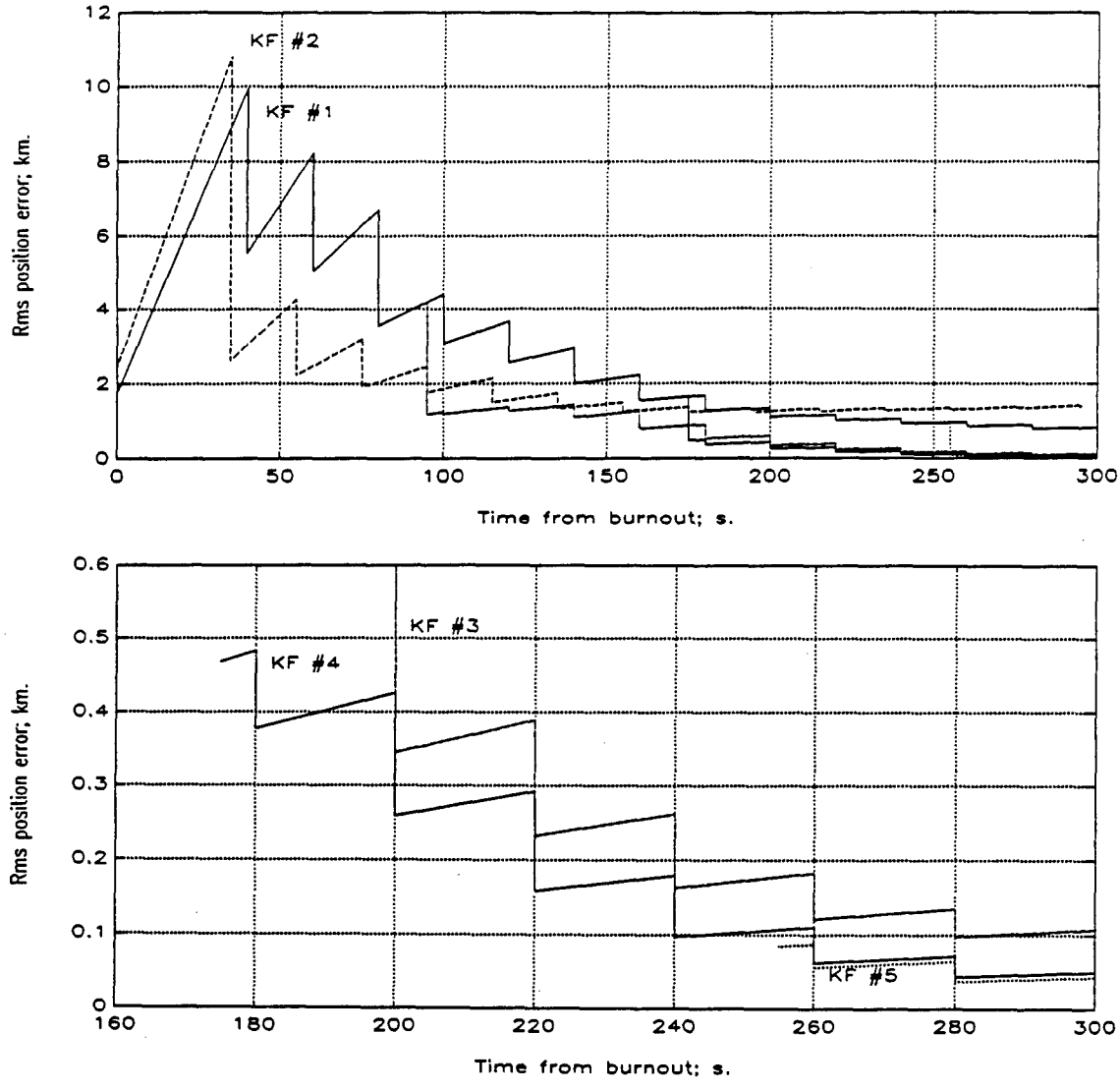



**Figure 2-14. Impact of Single Measurement From a Second Sensor on Position Error**

***(Inputs: KF1 from S1; KF2 from S2; KF3, 4, 5 from S1 and single input from S2)***

The first example in Table 2-3 is the baseline EKF, which appeared in all the runs. It receives 14 measurements (not counting initialization) from either S1 or S2 during the 300-sec scenario. In each set, the measurements are spaced 20 sec apart and are unsynchronized. Each measurement is sent over as a single messages of four words including a time tag.

Table 2-3. Communication Load of Various Fusion Architectures

| Example No. | Description | Words Per Run | Messages Per Run | Accuracy (Ranking) |
|---|---|---|---|---|
| 1 | 14 measurements from S1 or S2; Figure 2-10 KF1 or KF2 | 56 | 14 | 3 |
| 2 | 7 switched measurements from S1 and S2; Figure 2-15 KF3 | 28 | 7 | 3 |
| 3 | 14 interleaved measurements from S1 and S2; Figure 2-16 KF3 | 56 | 14 | 2 |
| 4 | 28 interleaved measurements from S1 and S2; Figure 2-5 KF3 | 112 | 28 | 1 |
| 5 | 14 measurements from S1 and 3 tracks based on S2; Figure 2-10 KF4 | 140 | 17 | 1 |
| 6 | 14 measurements from S1 and 2 tracks based on S2; Figure-2-10 KF4 | 112 | 16 | 1 |
| 7 | 14 measurements from S1 and 1 track based on S2; Figure 2-10 KF4 | 84 | 15 | 1 |

Example 2 (Figure 2-15) shows a strategy that preserves the same performance but halves the communication load. This strategy is achieved by switching the data from one sensor to another at 180 sec. It exploits the enhanced accuracy inherent in geographic diversity but reduces the data rate to the level that preserves the original accuracy.

Example 3 (Figure 2-16) preserves the communication load of the baseline system but uses sensor diversity. This strategy improves accuracy without increasing the communication load.

Example 4 uses all measurements available from both sensors. This results in the highest possible accuracy; however, this accuracy is achieved at the cost of maximum communication load. In examples 5 through 7, at the time of an update from a track based on data from the second sensor, the highest possible accuracy is achieved. At all other times, the performance is poorer than that in example 4. In terms of total number of messages, the communication load is always better than that in example 4. In terms of total number of words, the communication load could be better or worse depending on how

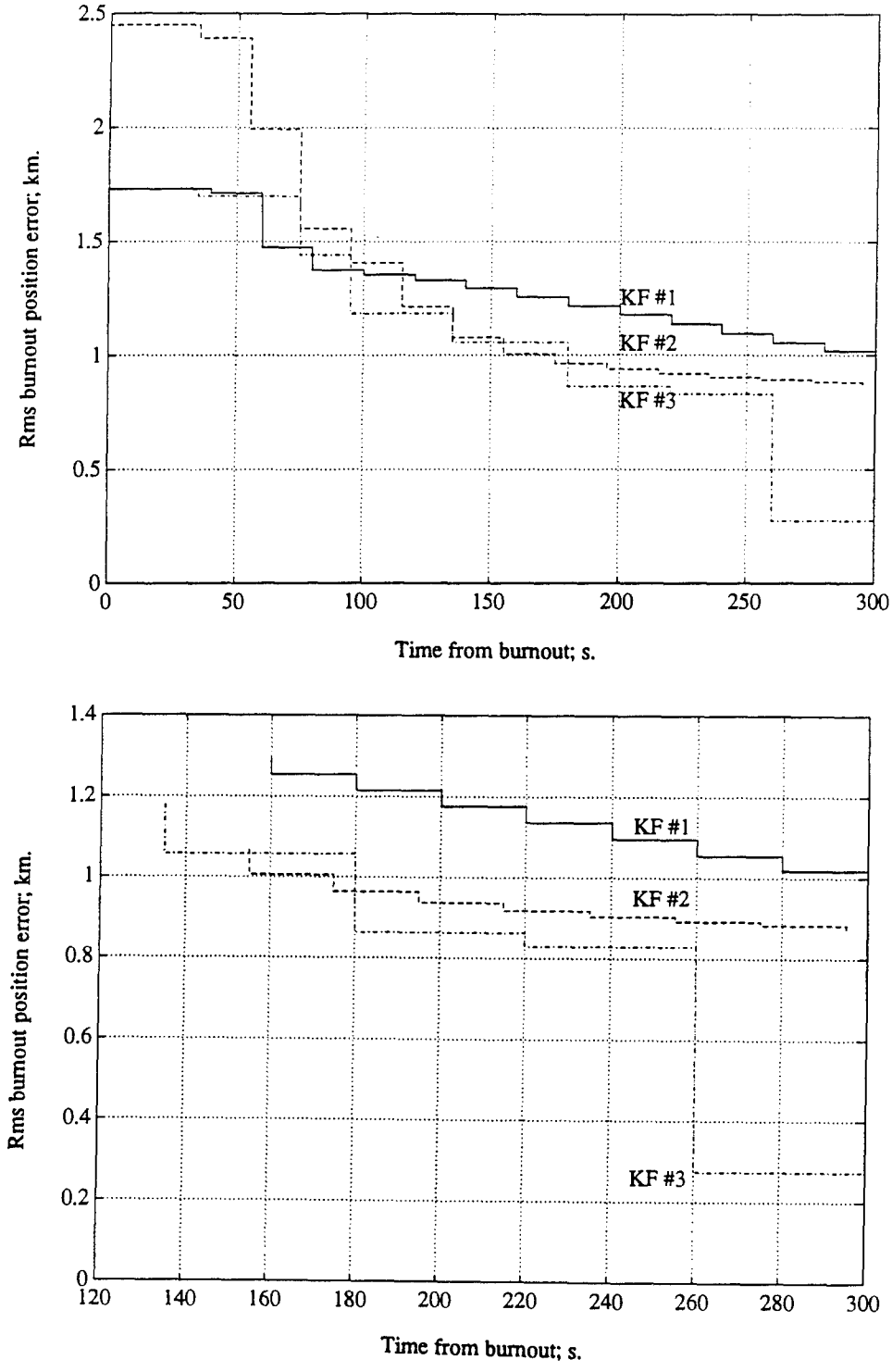



**Figure 2-15. Tradeoff Between Geographic Diversity and Reduced Data Rate** ***(Inputs: KF1 from S1; KF2 from S2; KF3 at reduced data rate switched at 180 sec from S2 to S1)***

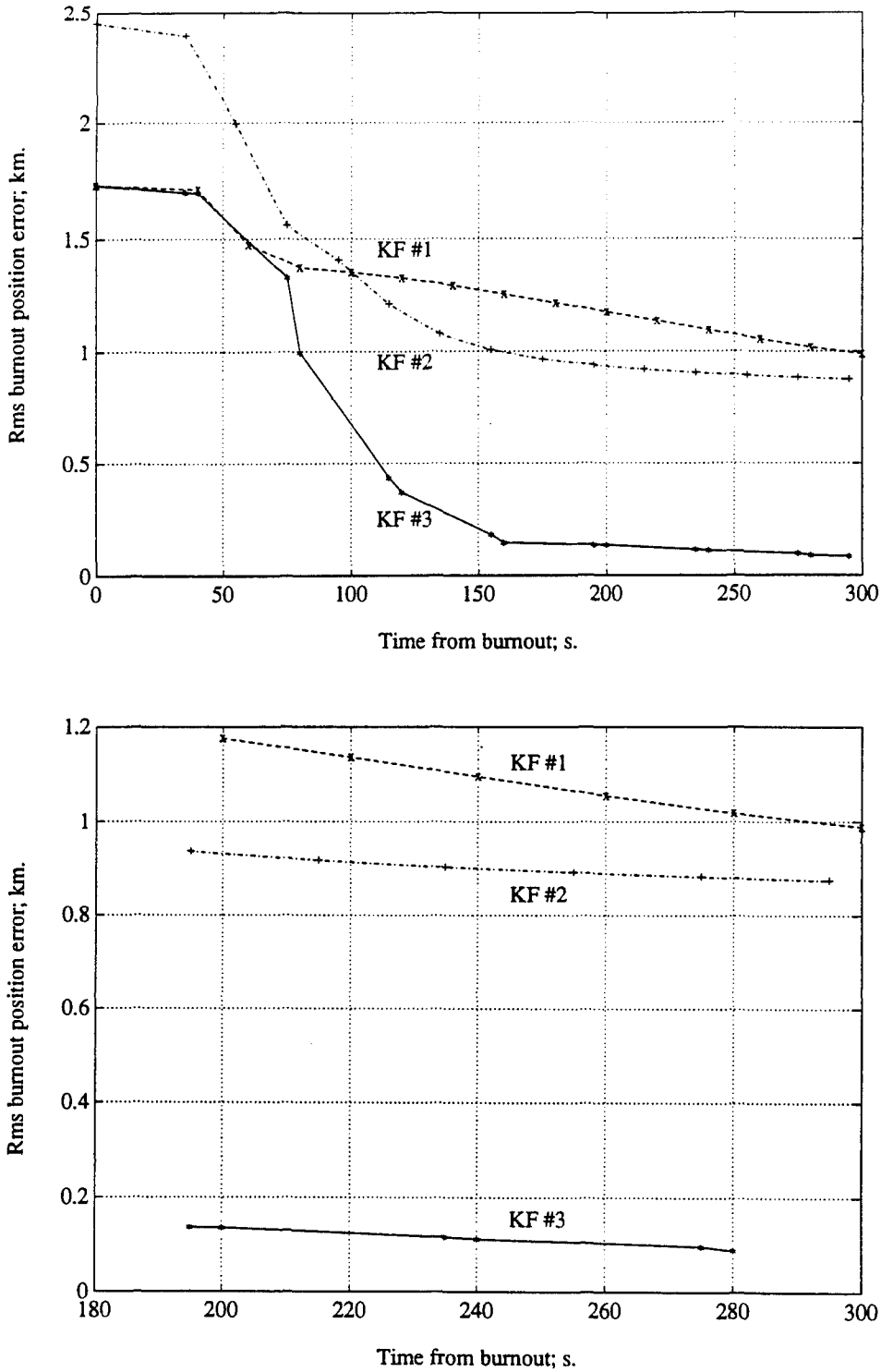



**Figure 2-16. Impact of Geographic Diversity**

***(Inputs: All three KFs at the same rate: KF1 from S1; KF2 from S2; KF3 from S1 and S2 interleaved)***

often an update by a track is desired. Thus, if the track update is used only once (e.g., just before an interceptor launch), example 7 shows that at that instant the best possible accuracy is achieved at great economy in the total number of words that need be transmitted. However, if three fusion operations are needed, such as in repeated shoot-look-shoot (example 5), the communication load in terms of the total number of words is greater.

The examples in Table 2-3 show that the performance/communication tradeoff is not a simple matter. It depends, among other factors, on the particular application. On the other hand, the sensor fusion alternatives described in this paper offer a great deal of flexibility in the optimization of the architecture.

# III. SUMMARY AND CONCLUSIONS

## A. BASIC PRINCIPLE

To reap the full benefits of sensor fusion, each tracking filter should be capable of being updated at any given time (asynchronously) by either a measurement from any sensor or a track from any other tracking filter. We refer to this mode of operation as Flexible Asynchronous Fusion (FAF).

## B. ALGORITHMS FOR FAF

When applied to EKFs that operate from different sensors, the algorithms needed for achieving FAF are straightforward and do not present appreciable computational problems. For updating by a sensor measurement (Figure 2-2), the two major operations are (1) raising the process transition matrix to a power, which is a computation inherent in any asynchronous operation, and (2) computing the Jacobian of the transformation from sensor coordinates to the common coordinate system, which is inherent in any sensor fusion implementation. When updating by an SV from another sensor and associated EKF, the seemingly formidable problems presented by correlation are overcome by the simple algorithm shown in Figure 2-4.

## C. THE SENSOR FUSION ARCHITECTURE MODEL (SFAM)

The software model, developed in MATLAB®, is a tool for evaluating the algorithms, scenarios, and various strategies for sensor fusion. It is easy to use and new features are easy to add.

## D. STRATEGIES FOR REALIZING THE BENEFITS OF FAF

Table 3-1 shows means of improving performance and reducing the communication load in five specific areas through FAF. Section II D provides additional insight into potential high-payoff strategies.

**Table 3-1. Strategies for Realizing the Benefits of FAF**

| Benefit | Strategy | Remarks |
|---|---|---|
| Improving accuracy | Measurements from two sensors | Geographic diversity yields major improvement if geometry is favorable even if sensor data are noisy |
| | Updates by track from other EKF | After update same accuracy as if all measurements from updating EKF were used |
| | Reporting rules | Reporting rule based on total rms position error foregoes benefits of geographic diversity; hence, this is inadvisable |
| Flexible attack on launchers | Flexible tradeoff between accuracy and fast response | Many mixed strategies become available; updates on request from other sources, shoot-look-shoot decisions as data becomes available |
| Efficient sensor management | Mixed strategies that assign on demand | Many potential payoffs: enhancing performance, freeing resources for other tasks, improving reliability |
| Dynamic intercept support | One measurement | Ask for it before interceptor launch; exact timing is not critical |
| | Mixture of measurements and tracks | This flexible string of updates from tracks and measurements can be used for shoot-look-shoot or in-flight update |
| Reduce communication load | Update only when needed | Reducing number of updates may reduce load |
| | Select update | Option of using track or measurement leads to economy |
| | Combine sources | Using geographic diversity and reducing data rate preserves accuracy at reduced communication load |

# IV. SOME REMAINING CHALLENGES

The scenario investigated in this paper is of the utmost simplicity. It consists of a single target in a ballistic trajectory without false signals or clutter. Although many important aspects of a real-life situation need to be examined, the most important are as follows:

- **The impact of the geographic configuration.** All of the examples in this paper are based on the configuration in Figure 2-7. A more critical examination, encompassing a number of different target trajectories relative to the sensors' baseline, must be conducted. The most important question concerns the improvement in performance from geographic diversity for various scenarios.
- **Algorithm for updating with a track in the presence of plant noise.** The classical Kalman equations presuppose plant noise. In our case, this would mean perturbations (drag or gravitational anomalies) and powered flight. The algorithm must be used for these cases not only because they occur in the theater scenario but also to extend this approach to other applications of urgent interest to the Department of Defense (DoD).
- **Multiple targets, false signals and clutter.** In the past, a substantial part of the tracking investigations sponsored by the SDIO concerned the following correlations: observation-to-observation, observation-to-track, and track-to-track. The approaches in this paper might lead to new algorithms and improved performance and to additional conclusions in these areas .
- **Latency.** We can easily show that latency is not a problem for ballistic trajectories, although we did not do so in this paper. In fact, with minor modifications, the algorithms of Figure 2-2 and Figure 2-4 can be used when the input for an update appears out of time sequence. However, this is not the case in the presence of plant noise, and extending these algorithms to data that are out of time sequence and pertaining to nonballistic missiles is important.
- **IR sensors**. Only radar sensors were addressed in this paper. IR sensors are an important component of the theater scenario and should be included in subsequent analyses.

- **Eliminating bias.** A source of error or concern is bias or systematic error, which to some extent is always present. Its importance depends on its magnitude and on the particular applications. Sensor fusion seems to lend itself to the evaluation and elimination of such systematic errors when they are, as is often the case, uncorrelated from sensor to sensor.
- **Reporting rules and sensor management.** These two topics are interrelated. We alluded to them in the body of this report. Algorithms for the optimum utilization of sensor resources and for the dissemination of tracking data are crucial to effective operations, such as sensor pointing, selection of search or track mode, and schedule for revisiting each target.

# APPENDIX

# SFAM CODE

# APPENDIX

# SFAM CODE

This appendix contains the complete code of the SFAM, including input data for a typical run. As described in this paper, most of the subroutines are in MATLAB® code.

```
%FUSESH27.M This is the shell of the sensor fusion evaluator.
%The files  filenameX -where X is the serial number of the run-  are
% run specific. They are filed as filenameX.m
%for each run.  FuseshX, prmtrsX and pltcomX must be edited for each run.
%For the record sverunX, prmtrsX and pltcomX should be saved.
%
%This run (X=27) has three filters, and S1, S2 = two independent radar sensors.
% The filters have the following inputs: KF1: S1 and KF2; KF2: S2;
%KF3:S1 . Independent sources with identical covariances are assumed for
% initialization, and the filters are updated accordingly, however -and this
% is the main purpose of this run- the actual error of KFfocus, namely KF1, is
% evaluated in COREV, when the assumption of independent sources for
%initialization is false and in fact KF1 and KF2 are initialized  from the same
%source. If this feature is of no interest the COREV and STATTRU comands below
%should be eliminated.
clear
%EDIT NEXT LINE
prmtrs27
Tsort
%In prmtrsxx.m Ttable was prepared manualy, listing all filters' update events.
%Table is sorted to arange the entries in chronological order, the first column
%being the time of the update event.
%Next for each entry of Ttable foreman is called, which in turn updates the KF
%whose ID is in the second column. In addition, for this particular run COREV
% is called in order to evaluate the true covariance when the sensors being
% fused are corelated.
[m,n]=size(Ttable)
for i=1:m
foreman
%COREV computes the true covariance and updates the error covariance matrix
%associated with KFfocus and the crosscovariance matrix associated with
% KFfocus and KFsendr. Their ID is defined in Prmtrs27
corev
end
%The results of the runs are put int suitable form by stat and saved in the
% SVERUNxx.MAT
stat
%if corev is used (as above) it's results are processed:
stattru
%! denotes a DOS comand.
%EDIT NEXT LINE
save sverun27
%all variables are saved in sverunXX.mat and may be later recalled with the load
%command
%The results are automatically ploted and displayed on the screen.
%Pressing any key shows the next plot. At any later time booting
%MATLAB and the commands load sverunX and pltcomX displays the plots.
%pltcomX.M also saves the plot in temp.met
% To print them exit MATLAB and call Matprint.bat which translates temp.met
%for the apropriate printer and prints the file. Temp.met is overwritten for
%each run
%EDIT NEXT LINE
pltcom27
```

```
% PRMTRS27.M
%Configuration
nS=2 %nS=number of sensors
nKF=3 %nKF=Number of KFs
dKF=6  %dKF=dimension of state vector (SV) for the KFs
%
%Sensors.
%Coords = Location coordinates for all sensors. One column per sensor.
Coords = [160,15,0;500,15,0]'  %
%Theta = covariances of measurement errors for all types of sensors
%in sensor coordinates. If a sensor type has dimension <6, supefluous
% elements(>measdim) filled with 10. Types separated by",".
Theta= [diag([1.7E-4,1.7E-4,1E-4,10,10,10]),diag([1.7E-4,1.7E-4,1E-4,10,10,10])]
erfrom=[1,2]
%For each sensor in sequence ( erfrom(i) for sensor i) erfrom is the pointer
% to the 6x6 matrices of its measurement errors given in Theta.
%
%Process model
X0 = [350,300,50,0.1,-1.5,1.4]'  %X0 = Initial state vector at t=0
Phi = [eye(3),eye(3);zeros(3),eye(3)]  %Phi = Propagation matrix over one s.
Vx1=zeros(6,6)% Process noise
%
%Initial conditions of KFs
Xh1=[351, 299, 50,0.05, -1.45, 1.42]'
Xh2=Xh1
Xh0=[Xh1,Xh2]
%The columns of Xh0 are i estimates of X0 available from external
% sources; they are used for initialization. Column "i" is for source i.
P01 = [eye(3),zeros(3);zeros(3),0.02*eye(3)]
P02=P01
P0=[P01,P02]
% P0i = error covariances of Xh0i.
%
% First certain variables are initialized:
Sch=zeros(nKF,1) %Sch is the update count for each KF
KFtime=zeros(nKF,1) % KFtime is the time of last update for each KF
Xtruelst=zeros(dKF,nKF)% The true state vector is set to zero
UU=zeros(nKF*dKF,20*dKF)
Pb=UU
Pa=UU
%The following two matrices will contain the Kalman gain and measurement
%mapping matrix at each update.
Kout=zeros(dKF,dKF)
Cc=zeros(dKF,dKF)
Kall=UU
Ccall=UU
%Xlast, Plast and tlast will contain for each KF the SV its covariance and
%time it was sent last from each other KF as an input
Xlast=zeros(nKF*dKF,nKF)
Plast=zeros(nKF*dKF,nKF*dKF)
tlast=zeros(nKF,nKF)
%
%The following variables are initialized only if impact of faulty indepedence
%is being evaluated in COREV
KFfocus=1
KFsendr=2
Pblast=P01
Palast=P01
Pcrlast=P01
```

```
Pbtrue=Pblast
Patrue=Palast
%
%Ttable describes the complete scenario of the run
%The code for the source (third column) is as follows: 0=no input;
% 1-9= sensor ID; 21 to 29=external source; usually used for initialization.
% with second digit:source ID (indexes P0 and Xo above). 31-49=another KF,
% the  second digit being the source KF ID.  If the first digit = 4, the
%destination KF was initialized before, and this is the first appearance of
% that source- destination KF pair
%Note that when KFid (2nd col.) first appears, 3rd col. must be 21-39 i.e a KF
%is initialized either from and external source or another KF
%Later this table will be sorted in chronological sequence (sort in fuseshX)
%For proper sorting it is important that the update of a KF be listed
% before it is shown as an input to another KF.
%For this run all three KF's are first initialized from an external source,
%but next KF3 gets an input from KF1, i.e it gets 2 identical inputs in
%a row, thus itsd output is the same as if KF1 & 2 were merged and independent
%
%Time;s.     KF ID          Source
%
Ttable=[
     0          1          21
     0          2          21
     0          3          21
     40         1          1
     40         2          2
     40         3          1
     60         1          1
     60         2          2
     60         3          1
     60         1          42
     80         1          1
     80         2          2
     80         3          1
     100        1          1
     100        2          2
     100        3          1
     120        1          1
     120        2          2
     120        3          1
     120        1          32
     140        1          1
     140        2          2
     140        3          1
     ]
```

```
%FOREMAN.M takes the next "job" from Ttable, unpacks the relevant input
%variables, updates the apropriate filter, and packs the results
%First we identify the count (row index in Ttable) of the next job:
k=sum(Sch) %k=the row index of the last job processed by "foreman"
tout=Ttable(k+1,1) %time of this job
KFid=Ttable(k+1,2) %KF to be updated
sensid=Ttable(k+1,3) % update input from this source
if Sch(KFid)==0
  inflag=1 %this flag is used further down to skip some operations
%If the KF was not initialed yet, do so:
  initial
  Sch(KFid)=1
  KFtime(KFid,Sch(KFid))=tout %This is the matrix of update times for each KF
else
  tin=KFtime(KFid,Sch(KFid)) %time of previous update for this KF
  s=tout-tin
  Phiup=power(Phi,s) %Transition matrix for updating SV to present
  Xtruein=Xtruelst(:,KFid)
end
if sensid<10
  Ins=sensid
elseif sensid<40
  Ins=sensid-30
else
 Ins=sensid-40
end
%The previous sequence unpacks the ID of the input to this KF
if inflag==0
  %The remaining lines are executed only if this update is
  %not an initialization
update
else
  inflag=0
end
```

```
%UPDATE.m
  %Called by FOREMAN to perform all update and hindsite operations. It does so
  %after determining the input parametes for each operation
  r=dKF*Sch(Ins)
  Xin=Xa(KFid*dKF-dKF+1:KFid*dKF-dKF+6,Sch(KFid))
  u=dKF*Sch(KFid)
  Pin=Pa(KFid*dKF-dKF+1:KFid*dKF-dKF+6,u-dKF+1:u)
  %Xin and Pin are the prior outputs of the KF; they are to be updated
  [Xb(KFid*dKF-dKF+1:KFid*dKF-dKF+6,Sch(KFid)+1),Pb(KFid*dKF-dKF+1...
  :KFid*dKF-dKF+6,u+1:u+dKF),Xtrue(:,k+1)]=...
  timeup(Xin,Pin,Phi,s,Xtruein,Vx1)
  %This was the time update. Its result is now defined as inputs
  % for the measurement update
  Xin=Xb(KFid*dKF-dKF+1:KFid*dKF-dKF+6,Sch(KFid)+1)
  Pin=Pb(KFid*dKF-dKF+1:KFid*dKF-dKF+6,u+1:u+dKF)
  Xtruein=Xtrue(:,k+1)
  %Depending on the source of measurement, IR, Radar, external source
  %or another KF certain parameters have to be derived in "measgen"
  measgen
  %The measgen outputs are used as parameters in the function measurup:
  [Xa(KFid*dKF-dKF+1:KFid*dKF-dKF+6,Sch(KFid)+1),...
  Pa(KFid*dKF-dKF+1:KFid*dKF-dKF+6,u+1:u+dKF),Kout]=...
  measurup(Xin,Pin,Thetain,z,Cc)
  %The next lines are used to save the mapping matrix and Kalman gain; for many
  %runs they may be eliminated
  [kk,ll]=size(Kout)
  sK=[kk;ll]
  [gg,hh]=size(Cc')
  sC=[gg;hh]
  KK=zeros(dKF,dKF)
  Cp=zeros(dKF,dKF)
  KK(1:kk,1:ll)=Kout
  Cp(1:gg,1:hh)=Cc'
  Kall(KFid*dKF-dKF+1:KFid*dKF-dKF+6,u+1:u+dKF)=KK
  Ccall(KFid*dKF-dKF+1:KFid*dKF-dKF+6,u+1:u+dKF)=Cp
  sizeK(KFid*dKF-dKF+1:KFid*dKF-dKF+2,Sch(KFid)+1)=sK
  sizeCc(KFid*dKF-dKF+1:KFid*dKF-dKF+2,Sch(KFid)+1)=sC
  %The burnout position is estimated:
  [Xlaunch(KFid*dKF-dKF+1:KFid*dKF-dKF+6,Sch(KFid)+1),...
  Plaunch(KFid*dKF-dKF+1:KFid*dKF-dKF+6,u+1:u+dKF),siter(KFid,Sch(KFid)+1)]=...
  hindsite(Xa(KFid*dKF-dKF+1:KFid*dKF-dKF+6,Sch(KFid)+1),...
  Pa(KFid*dKF-dKF+1:KFid*dKF-dKF+6,u+1:u+dKF),Phi,tout,X0)
  KFtime(KFid,Sch(KFid)+1)=tout
  Sch(KFid)=Sch(KFid)+1
  Xtruelst(:,KFid)=Xtrue(:,k+1)
```

```
%INITIAL.M
%    INITIAL.M assignes the initial  estimate and its error to a KF.
% The source of these initial estimates is either an external source
% or another KF.  For initialization Pa=Pb
Xtrue(:,k+1)=power(Phi,tout)*X0-gravity(tout)
Xtruelst(:,KFid)=Xtrue(:,k+1)
w=sensid
if w<30
   Ins=w-20
   Xb(KFid*dKF-dKF+1:KFid*dKF,1)=Xh0(:,Ins)
   Pb(KFid*dKF-dKF+1:KFid*dKF,1:dKF)=P0(:,dKF*(Ins-1)+1:dKF*Ins)
   Xa(KFid*dKF-dKF+1:KFid*dKF,1)=Xb(KFid*dKF-dKF+1:KFid*dKF,1)
   Pa(KFid*dKF-dKF+1:KFid*dKF,1:dKF)=Pb(KFid*dKF-dKF+1:KFid*dKF,1:dKF)
  else
   Ins1=w-30 %w is the ID of the source (KF) from which this KF is initialized
   Ins2=Sch(Ins1) %This is the update count of the source KF
   Xb(KFid*dKF-dKF+1:KFid*dKF,1)=Xa(Ins1*dKF-dKF+1:Ins1*dKF,Ins2)
   Pb(KFid*dKF-dKF+1:KFid*dKF,1:dKF)=Pa(Ins1*dKF-dKF+1:Ins1*dKF,...
   Ins2*dKF-dKF+1: Ins2*dKF)
   Xa(KFid*dKF-dKF+1:KFid*dKF,1)= Xb(KFid*dKF-dKF+1:KFid*dKF,1)
   Pa(KFid*dKF-dKF+1:KFid*dKF,1:dKF)= Pb(KFid*dKF-dKF+1:KFid*dKF,1:dKF)
%Having updated the KF from another KF the updated inputs are
%stored; they are used later when another input from the same source KF
%is processed
  Xlast(KFid*dKF-dKF+1:KFid*dKF,Ins1)=Xb(KFid*dKF-dKF+1:KFid*dKF,1)
   Plast(KFid*dKF-dKF+1:KFid*dKF,Ins1*dKF-dKF+1:Ins1*dKF)=...
   Pb(KFid*dKF-dKF+1:KFid*dKF,1:dKF)
   tlast(KFid,Ins1)=tout
end
[Xlaunch(KFid*dKF-dKF+1:KFid*dKF,1),Plaunch(KFid*dKF-dKF+1:KFid*dKF,1:dKF),...
siter(KFid,1)] = hindsite(Xa(KFid*dKF-dKF+1:KFid*dKF,1),...
Pa(KFid*dKF-dKF+1:KFid*dKF,1:dKF),Phi,tout,X0)
```

```
%TIMEUP.M is the function which is the time update segment of the KF. It yields
 %the predicted position and its error covariance and the true position.
 %
 %Inputs
 %s=time increment
 % Xin=SV to be updated
 % Phi = process propagation matrix in unit time
 % Pin = covariance matrix to be updated
 % Xtruein= true value of state vector to be updated
 % Vx1= plant nois3 in 1 s.
% Outputs
%C  Xout = updated SV  estimate
 % Pout =error covariance matrix of Xout
%Xtrueout=updated true SV.
function[Xout,Pout,Xtrueout]=timeup(Xin,Pin,Phi,s,Xtruein, Vx1)
%We compute a normal random vector namely the process noise over s s.
% Over 1 s. its components are independent, with covariance matrix Vx1
rand('normal')
Vwd=sqrt(diag(Vx1))
U=0*Vwd
for i=1:s
U=Phi*U+rand(Vwd).*Vwd
end
% Next we compute CU, the covariance of U
% over s seconds given the process transition matrix Phi over one second
%and the covariance of the process noise PP over one second
[m,n]=size(Phi)
W = eye(m)
CU=0*Phi
for i= 1:s
W =W*Phi
CU = CU+W*Vx1*W'
end
Phiup=Power(Phi,s)
Xout=Phiup*Xin-gravity(s)
Pout=Phiup*Pin*Phiup' + CU
Xtrueout=Phiup*Xtruein-gravity(s) + U
```

```
%MEASGEN.M This subroutine generates a measurement z, its error covariance
% Thetain and the measurement mapping function Cc.
if sensid<6
  %This is for radar
  m=erfrom(sensid)
  Thetain=Theta(1:3,6*(m-1)+1:6*(m-1)+3)
  U=Coords(:,sensid)
  Xp1=Xtruein(1:3)-U
    Jack=Jacob(Xp1) %Jack=the jacobian of the coord.xformation
    %from polar to cart. Jack=delta(Xp)/delta(Xc). X's dimension=6,
  % i.e. pos and vel. The subscripts p and c will denote polar and cartesian.
  %Mp=Cp*Xp+Ep; Xp~=Jack*Xc  Mp=Cp*Jack*Xc+Ep, hence Cc=Cp*Jack
  Cp=[eye(3,3),zeros(3,3)]
  %Thetain is the measurement error covariance in polar coordinates
  % centereed on the sensor
  Cc=Cp*Jack
  z=Cc*Xtruein+sqrt(diag(Thetain))
  %Here the measurement error was taken=rms value.
  % In montecarlo a random vector will be generated
elseif sensid<10
  %This s for IR sensor
  Cp=eye(2,2)
elseif sensid<40
  %This is for another KF or an external input. The following entries
 %will derive a synthetic measurement z, and its
%error covariance Thetain which would have given rise to Pnew and Xnew. These wi
%update the filter. z has the following characteristics: dimension = dKF; if z w
%actual measurement to update the source KF, it sould result in updated
%SV equal to z and the same state vector as actualy obtained namely Pnew
%This procedure enables us to use an SV from another KF as just another measurem
  Cc=eye(6,6)
  %The last input from the KF whose input is processed now is updated
  %for whitening
  tstep=tout-tlast(KFid,Ins)
  %Vxoth=plntn(Px,Phi,tstep)
  Plastin=Plast(KFid*dKF-dKF+1:KFid*dKF,Ins*dKF-dKF+1:Ins*dKF)
  Xlastin=Xlast(KFid*dKF-dKF+1:KFid*dKF,Ins)
  %The present input from the KF with ID Ins:
  Xnew=Xa(Ins*dKF-dKF+1:Ins*dKF,Sch(Ins))
  Pnew=Pa(Ins*dKF-dKF+1:Ins*dKF,r-dKF+1:r)
 % Xpred=Phiup*Xlastin-gravity(tstep)
 % Ppred=Phiup*Plastin*Phiup'+Vxoth
  [Xpred,Ppred,Xdummy]=timeup(Xlastin,Plastin,Phi,tstep,Xtruein,Vx1)
  Thetain=eye(dKF)/((Ppred\(Ppred-Pnew))/Ppred)-Ppred
  z=((Thetain+Ppred)/Ppred)*(Xnew-Xpred) +Xpred
  %The most recent inputs from the source KF are saved to be used later as
%above:
  tlast(KFid,Ins)=tout
  Xlast(KFid*dKF-dKF+1:KFid*dKF,Ins)=Xnew
  Plast(KFid*dKF-dKF+1:KFid*dKF,Ins*dKF-dKF+1:Ins*dKF)=Pnew
 %If the source KFs SV is input for the first time, it is uncorelated,
 %and it may be used directly without the previous computations:
else
  Cc=eye(6,6)
  z=Xa(Ins*dKF-dKF+1:Ins*dKF,Sch(Ins))
  Thetain=Pa(Ins*dKF-dKF+1:Ins*dKF,r-dKF+1:r)
  tlast(KFid,Ins)=tout
  Plast(KFid*dKF-dKF+1:KFid*dKF,Ins*dKF-dKF+1:Ins*dKF)=Thetain
end
```

```
%MEASURUP.M
%This function is the measurement update portion of the KF.
function [Xout,Pout,K]=measurup(Xin,Pin,Thetain,z,Cc)
%The flag f=0 means that there is no measurement at this update and the
% estimates "after" are set to their "before" value
% The usual KF filter equations apply
  %The Kalman gain is given by
  K=Pin*Cc'/(Thetain+Cc*Pin*Cc')
  %The updated covariance matrix is
  Pout=(eye(6,6)-K*Cc)*Pin
  Xout=Xin+K*(z-Cc*Xin)
```

```
function [Xout,Pout,siter] = hindsite(Xin,Pin,Phi,tback,X0)
%HINDSITE.M computes the launch site from the present position as well as it
%rms error and error covariance
%Inputs.
%Xin=present estimated SV
%Pin = covariance of Xin
%Phi=state transition matrix over 1 sec
%tback=present time
%X0=true value of the SV at t=0
%Outputs
%Xout=estimate of SV at t=0
%Pout=covariance of Xout
%siter=rms distance error of Xout
[m,n]=size(Phi)
Phiback=eye(m)/power(Phi,tback)
Xout=Phiback*Xin+gravity(tback)
Pout=Phiback*Pin*Phiback'
siter=(Pout(1,1)+Pout(2,2)+Pout(3,3))^0.5
```

```
%COREV.M evaluates the true covariance of KFfocus and the crosscovariance
%of KFfocus and KFsendr, when their errors are correlated (for example due to
% the common initialization source), and they are fused
%
%
%If the KF being updated is the one being evaluated, & it was already
%initialized before & it is receiving an update for the first time from the KF
% with which it is correlated:
if (KFid==KFfocus)&(Sch(KFid)>1)&(sensid==40+KFsendr)
Checkit=1
elseif (KFid==KFfocus)&(Sch(KFid)>1)&(sensid~=(40+KFsendr))
Checkit=2
elseif (KFid==KFsendr)&(Sch(KFid)>1)
Checkit=3
else
Checkit=4
end
%
if Checkit==1
corev1
%
elseif Checkit==2
corev2
%
elseif Checkit==3
corev3
%
else
Checkit2=4
end
```

```
%COREV1.m
%(KFid==KFfocus)&(Sch(KFid)>1)&(sensid==40+KFsendr)
%this case is the one in which the measurement and the filter error are
%*correlated and the crosscorelation matrix as well as the filter error
%correlation matrix must be updated:
Checkit2=1
KH=Kout*Cc
[uu,vv]=size(KH)
IKH=[eye(uu)-KH]
Pblast =Phiup*Palast*Phiup'
Pbtrue=[Pbtrue,Pblast]
Pcrlast=Phiup*Pcrlast
IPI=IKH*Pblast*IKH'
IKHPK=IKH*Pcrlast*Kout'
KTK=Kout*Thetain*Kout'
Palast=IPI+KTK+IKHPK+IKHPK'
Patrue=[Patrue,Palast]
```

```
%COREV2.m
%if the KF being updated is the one being evaluated and it was initialized
%before, and its input is anything else but the first time input from the
%KF with which it is correlated
% the measurement input is not correlated with the filter error, however the
%crosscorrleation between the errorr in KFfocus and KFsendr as well as the true
%error correlation must be updated
Checkit2=2
KH=Kout*Cc
[uu,vv]=size(KH)
IKH=[eye(uu)-KH]
Pblast =Phiup*Palast*Phiup'
Pbtrue=[Pbtrue,Pblast]
Pcrlast=Phiup*Pcrlast
Pcrlast=IKH*Pcrlast
Palast=IKH*Pblast*IKH'+Kout*Thetain*Kout'
Patrue=[Patrue,Palast]
```

```
%COREV3.M
%in this case the error crosscorrelation between KFfocus and KFsendr
%must be updated
Checkit2=3
KH=Kout*Cc
[uu,vv]=size(KH)
IKH=[eye(uu)-KH]
Pcrlast=(IKH*Phiup*Pcrlast')'
```

```
%STAT.M . This subroutine processes the various tables to derive final
%results.
%ERB and ERA contain the rms position errors of the KFs before and after
%measurements. d=nKFxmaxS
ERB=zeros(nKF,max(Sch))
ERA=ERB
  for i=1:nKF,
    for j=1:Sch(i),
      a=dKF*(i-1)+1
      b=dKF*(j-1)+1
      ERB(i,j)=(Pb(a,b)+Pb(a+1,b+1)+Pb(a+2,b+2))^0.5
      ERA(i,j)=(Pa(a,b)+Pa(a+1,b+1)+Pa(a+2,b+2))^0.5
    end
  end
Erall=[ERB;ERA]
for i=1:nKF,
  Poser(:,3*(i-1)+1)=KFtime(i,:)'
  Poser(:,3*(i-1)+2)=Erall(i,:)'
  Poser(:,3*(i-1)+3)=Erall(nKF+i,:)'
end
for i=1:nKF
PA1=Poser(:,3*(i-1)+1)
PB1=PA1
PC1=[PA1,PB1]'
PA2=Poser(:,3*(i-1)+2)
PB2=Poser(:,3*(i-1)+3)
PC2=[PA2,PB2]'
Poserall(:,2*(i-1)+1)=PC1(:)
Poserall(:,2*(i-1)+2)=PC2(:)
end
Xtruepr=Xtrue'
Xbpr=Xb'
Xwhence=Xlaunch'
Pwhence=Plaunch'
whencer=siter'
Xapr=Xa'
for i=1:nKF
  PA1=KFtime(i,:)'
  PB1=PA1
  PC1=[PA1,PB1]'
  PA2=whencer(:,i)
  PB2=PA2
  PC2=[PA2,PB2]
  PC2=[PA2,PB2]'
  Burnout(:,2*(i-1)+1)=PC1(:)
  BU2=PC2(:)
  k=size(PC2(:))
  Burnout(:,2*(i-1)+2)=[BU2(1,1),BU2(1:k-1)']'
end
Xall1=[KFtime',Xbpr,Xapr]
```

```
%STATTRU.M . This subroutine computes the true rms error of a KF whose
%parameters are incorrect
%
%ERB and ERA contain the rms position errors of the KFs before and after
%measurements. d=nKFxmaxS
ERB=zeros(1,max(Sch))
ERA=ERB
for j=1:Sch(KFfocus)
   b=dKF*(j-1)+1
   ERBT(j)=(Pbtrue(1,b)+Pbtrue(2,b+1)+Pbtrue(3,b+2))^0.5
   ERAT(j)=(Patrue(1,b)+Patrue(2,b+1)+Patrue(3,b+2))^0.5
end
Eralltru=[ERBT;ERAT]
  Posertru(:,1)=KFtime(KFfocus,:)'
  [vv,uu]=size(Eralltru(2,:))
  Posertru(1:uu,2)=Eralltru(1,:)'
  [vv,uu]=size(Eralltru(2,:))
  Posertru(1:uu,3)=Eralltru(2,:)'
PA1=Posertru(:,1)
PB1=PA1
PC1=[PA1,PB1]'
PA2=Posertru(:,2)
PB2=Posertru(:,3)
PC2=[PA2,PB2]'
Ertruall(:,1)=PC1(:)
Ertruall(:,2)=PC2(:)
```

```
% PLTCOM27.M
% It is assumed that all variables from a prior run were saved in SVERUNxx.m
%  This was done by fusesh'x'
% If MATLAB is exited subsequently later LOAD SVERUxx
%may be called up in MATLAB before calling this subroutine.
%This subroutine generates any number of plots and puts them into temp.met.
%Afterexiting MATLAB, matprint.bat is called up in DOS. It translates
%temp.met into temp.jet for the jet printer and subsequently prits all
%plots, two plots to a page.
%This subroutine first deletes temp.m so that only plot generated on this run
%will be printed.
%
!del temp.met
%
%
plot(Poserall(1:2*Sch(1),1),Poserall(1:2*Sch(1),2),'-',...
Poserall(1:2*Sch(2),3),Poserall(1:2*Sch(2),4),'--'...
,Poserall(1:2*Sch(3),5),Poserall(1:2*Sch(3),6),'-.'...
,Ertruall(1:2*Sch(1),1),Ertruall(1:2*Sch(1),2),':')
grid
title('RUN # 27. RMS POSITION ERROR')
xlabel('Time from burnout; s.')
ylabel('Rms position error; km.')
text(100,9,'_    KF #1')
text(100,8,'--  KF #2')
text(100,7,'_ . KF #3')
text(100,6,'... True error')
pause
meta temp
%
%plot(Poserall(16:24,5),Poserall(16:24,6),...
%'-',Poserall(9:15,7),Poserall(9:15,8),'--')
%grid
%title('RUN # 22.RMS POSITION ERROR')
%xlabel('Time from burnout; s.')
%ylabel('Rms position error; km.')
%text(40,11,'KF #2')
%text(75,7,'KF #1')
%text(50,1.3,'KF #3')
%text(180,0.6,'KF #4')
%pause
%meta temp
```

```
%CARPOL.M This function transforms from cartesian to polar coordinates.
%X is the cartesian coordinates, Y is the polar (az,el,R)
function Y=carpol(X)
Y(1)=atan((X(2))/(X(1)))
Y(2)=atan(X(3)/((X(1))^2+(X(2))^2)^0.5)
Y(3) =norm(X)
```

```
%POWER.M. This function raises a matrix to a power q>>1 by factring q.
function U=power(A,q)
a=fix(q/64)
r=q-64*a
b=fix(r/8)
c=r-8*b
B=A^8
C=B^8
D=C^a
E=B^b
F=A^c
U=D*E*F
```

```
%GRAVITY.M
%This function computes the contribution of gravity over time t to the
%z component of a state vector
function Gcontr=gravity(t)
d=0.009807*t^2/2
v=0.009807*t
Gcontr=[0,0,d,0,0,v]'
```

```
%JACOB.M computes the Jacobian of the cartesian to az, el,R transformation
% of the six-component (Z,Zdot) vector
%X is the cartesian coordinates, Y is the polar (az,el,R)
% and Der(i,j)=delta (Y(i))/delta(X(j))
function Derbig=Jacob(X)
Der=zeros(3,3)
%Der is the Jacoban of the position transformation
Y(1)=atan((X(2))/(X(1)))
Y(2)=atan(X(3)/((X(1))^2+(X(2))^2)^0.5)
Y(3) =norm(X)
U(1,1)=X(1)+1
U(2)=X(2)+1
U(3)=X(3)+1
Der(1,1)=atan((X(2))/(U(1)))-Y(1)
Der(2,1)=atan((X(3))/(((((U(1))^2)+((X(2))^2))^0.5)))-Y(2)
Der(3,1) =(X(1))/(Y(3))
Der(1,2)=atan(((U(2))/X(1)))-Y(1)
Der(2,2)=atan(X(3)/((X(1)^2+(U(2))^2))^0.5)-Y(2)
Der(3,2) =(X(2))/(Y(3))
Der(1,3)=atan((X(2))/(X(1)))-Y(1)
Der(2,3)=atan((U(3))/(((((X(1))^2)+((X(2))^2))^0.5)))-Y(2)
Der(3,3) =(X(3))/(Y(3))
Derbig=[Der,zeros(3,3);zeros(3,3),Der]
%Derbig is the Jacoban of the full six-component vector
```

```
%Tsort.m This subroutine sorts the Ttable in chronological order and changes som
%and parameters accordingly
[U,I]=sort(Ttable(:,1))
Tnew(:,1)=U
Tnew(:,2)=Ttable(I,2)
Tnew(:,3)=Ttable(I,3)
Ttable=Tnew
```

```
rem MATPRINT.BAT
 del temp.jet
call C:\matlab\bin\GPP temp /Dps
 copy temp.ps lpt2
```

# GLOSSARY

| | |
|---|---|
| ASCII | American Standard Code for Information Interchange |
| BM | ballistic missile |
| $BM/C^3$ | ballistic missile/command, control, and communications |
| BMDO | Ballistic Missile Defense Office |
| DoD | Department of Defense |
| DOS | disk operating system |
| EKF | Extended Kalman Filter |
| FAF | Flexible Architecture Fusion |
| GDOP | Geographic Dilution of Position |
| ID | identification |
| IR | infrared |
| KF | Kalman Filter |
| km | kilometer |
| m | meter |
| $P_k$ | probability of kill |
| rad | radians |
| rms | root mean squared |
| s | second (used in several figures) |
| SAR | Synthetic Aperture Radar |
| SDIO | Strategic Defense Initiative Organization |
| sec | second |
| SFAM | Sensor Fusion Architecture Model |
| SV | state vector |
| TBMD | Theater Ballistic Missile Defense |

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704-0188*



| 1. AGENCY USE ONLY *(Leave blank)* | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | April 1994 | Final—September 1993–April 1994 |

**4. TITLE AND SUBTITLE**

Flexible Architectures for Sensor Fusion in Theater Missile Defense

**5. FUNDING NUMBERS**

C–MDA 903 89 C 0003

T–T-R2-597.12

**6. AUTHOR(S)**

Dr. Gabriel Frenkel

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**

Institute for Defense Analyses
1801 N. Beauregard St.
Alexandria, VA 22311-1772

**8. PERFORMING ORGANIZATION REPORT NUMBER**

IDA Paper P-2935

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

Ballistic Missile Defense Organization
The Pentagon, Room 1E1062
Washington, DC 20301-7100

**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

**11. SUPPLEMENTARY NOTES**

**12a. DISTRIBUTION/AVAILABILITY STATEMENT**

Approved for public release; distribution unlimited.

**12b. DISTRIBUTION CODE**

**13. ABSTRACT** *(Maximum 180 words)*

This paper presents sensor fusion procedures and algorithms for tracking and intercept support. The effort was undertaken in support of Ballistic Missile Defense Office (BMDO) activities pertaining to the design of a effective Theater Ballistic Missile Defense (TBMD) Command and Control structure and associated Operational Control and Combat Command elements design. The following major issues are addressed: (1) Methods of sensor fusion and associated algorithms (their advantages and disadvantages and their dynamic utilization in the battle scenario). Issues of track fusion vs. data fusion are given particular attention. (2) The communication loads of particular sensor fusion architectures and of the associated operational control activities. (3) Algorithms for the effective utilization of sensor fusion for intercept support and for burnout point estimation. This paper addresses the simplest possible case: a single missile on a ballistic trajectory tracked by two radar sensors in a flat earth model. False signals, clutter, closely spaced objects, tumbling, and gravitational anomalies are not considered. We demonstrate that even this simple case is conducive for providing valuable answers in the broad areas listed above. Numerous graphs are included. These graphs are the results of simulation runs by the Sensor Fusion Architecture Model (SFAM) developed for this effort.

| 14. SUBJECT TERMS | 15. NUMBER OF PAGES |
|---|---|
| algorithms, ballistic missile defense, communications load, data fusion, Kalman filter, sensor fusion, track fusion, tracking | 68 |
| | 16. PRICE CODE |

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| UNCLASSIFIED | UNCLASSIFIED | UNCLASSIFIED | SAR |