# DATA FUSION III: Estimation Theory

**Date:** March 16, 2006 **Time:** 5:00 – 7:30 PM
**Location:** B-300-2-3 (AAR-400) (Main Building, 2nd floor, near freight elevators)
**Instructor:** Dr. James K Beard
**Credits:** 1 **Course Code:** 0901-501-05 **Registration Number:** 13460

## Today's topics:

# 1 Care and Feeding of Radar Trackers

## 1.1 The Measurement Covariance Matrix R

The measurement covariance matrix nearly always consists of variances of the measurements – that is, the errors in the measurements are nearly always uncorrelated. Thus the problem of finding the measurement covariance matrix *R* usually consists of finding the mean square errors in each measurement. There are three principal ways to do this:

- An estimate based on an analysis that includes measurement of the receiver noise.
- A prediction based on known radar characteristics and known or estimated target strength.
- An *ad hoc* assignment of values that is observed to make simulated or real tracker performance meet requirements.

The first is preferred, and the radar signal processor itself provides measurement error variances based on observed noise floors and measurement characteristics in many radars. However, in many radars, including most FAA radars, there is no provision for reporting or estimating measurement noise in the signal processor. In some radars the noise floor may be available from CFAR or other processing, but again in FAA radars this is not the case.

The second alternative provides good estimates of measurement variances, particularly for radars that have periodic built-in test (BIT) or calibration parameters available to the data processor. The missing parameter is the target strength, which may be available as part of the radar contact data package. If the target strength is available, the radar noise floor can be accurately estimated from radar parameters, so the signal to noise ratio (SNR) can be determined. When SNR is available, measurement variances can be estimated by

$$\sigma_y^2 = c_0 + \frac{c_1}{1 + c_2 \cdot SNR} \tag{1.1}$$

In the case of monopulse measurements, we include an additional multiplicative factor on the second term that is quadratic in the angle off beam boresight. The parameters $c_0$, $c_1$ and $c_2$ can usually be given in terms of radar parameters but may be assigned by trial and error as part of the tracker tuning process.

## 1.2 The Covariance of the Process Noise Covariance Q

The process noise covariance matrix *Q* is usually not well-defined in practical applications. Usually, for a particular aircraft, one can estimate it from first principles of aerodynamics and an understanding of the aircraft. However, this requires identification of the aircraft by type, an availability of the aerodynamics of that aircraft. In all, the process noise matrix will depend on

- The aircraft type and trim.
- The aircraft speed.
- The aircraft altitude.

- The weather.
- Whether or not the autopilot is being used and the autopilot settings.
- Noisy engine thrust variations.

In addition, the process noise matrix will be diagonal only if it is posed in aircraft coordinates – axes of fore, starboard, and down, centered at the aircraft center of gravity. There are several ways to pose the aircraft process noise matrix:

- An intelligent assignment based on knowledge of all significant influences on the aircraft, with influences fore-and-aft, port-and-starboard, and up-and-down addressed separately.
- An intelligent assignment based on aircraft type and position – descending at low speed and altitude will indicate that the flaps are down, for instance.
- An *ad hoc* assignment based on getting acceptable tracker performance.

As a practical matter, usually it is best to define the process noise matrix for an aircraft as follows:

- For constant velocity or constant turn rate aircraft motion models, assign process noise only to the velocity states. In general use nonzero process noise in the states corresponding to the highest derivative with respect to time.
- Pose the aircraft process noise in aircraft coordinates. Use the process noise mapping matrix $G$ to rotate the coordinates to whatever is used in the state vector. Aircraft coordinates can be found using the velocity vector and the trajectory acceleration; fore is in the direction of the negative of the velocity vector, down is the sum of the trajectory acceleration and the gravity vector, and starboard is the cross product of the down and fore vectors. This will allow the process noise matrix to be accurately described as diagonal – the process noises in these axes will usually be uncorrelated.
- Define the magnitudes of the process noises according to simple rules about aircraft type and state – cruise, approach, circling, landing, takeoff, etc. Make the process noise adaptive as described below.

Adaptive process noise can be formulated by making it proportional to the squared tracker error. For example the squared error of the tracker error in range, divided by its variance as estimated by the Kalman filter, will have a mean of zero and a variance of one; deviation of conditions as modeled by the Kalman filter will result in differences that in general make this statistic larger. This can be used to adjust the process noise by making the process noise proportional to this stastic, or a smoothed estimate of this stastic. A specific example of this type of adaptive process noise is described below in Section 2.4 below.

## *1.3 Initialzing the Kalman filter*

Some elementary works on Kalman filtering beg the question of good initialization, and state that the filter can be initialized on the first measurement by defining the state estimates the measurements, or functions of the measurements, except when the states are unobservable from single measurements, and defining very high variances for the states that are not observable on the first measurement. This approach can be explained in

terms of first principles but the fact is that whatever you provide in Kalman filter initialization is processed as information, just as if it had come in as measurement information. If this information is incorrect, then it will contaminate subsequent estimates updated with measurement data. You should always initialize a Kalman filter from measurement data. If more than one measurement is required to provide observability of all the states, use the first few hits in a small batch estimator to provide a state vector and covariance matrix as initialization of the Kalman filter.

# 2 The Snake Oil Two-State Tracker

Here we present a very simple two state adaptive Kalman filter that is suitable for use as a replacement for an adaptive alpha-beta tracker. It also illustrates simple principles of adaptive Kalman filtering, and this example shows how a tracker can be tuned without rebuilding software modules using designated parameters that are kept as part of an editable configuration file.

## *2.1 Formulation*

We begin with the Kalman filter equations for two states defined in general as a position state and a position rate state, and a constant velocity model that gives us the simplest of system transition matrices:

$$\underline{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \text{Position} \\ \dfrac{d\text{Position}}{dt} \end{bmatrix}$$
$$\Phi = \begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix} \tag{2.1}$$

The measurements are scalars *y* and the measurement variances $\sigma_y^2$ are likewise scalars.

## *2.2 Initialization*

Initialization is done on the first two "hits" or radar returns. We do a simple placeholder on the first hit and a full initialization on the second hit.

### 2.2.1 Setting the States and Covariance on the First Radar Return

On the first return, we simply set the position state to the measurement and the covariance to the measurement variance, but no position rate information is available. We simply set the position rate state to zero and the variance to an arbitrary high value. The cross terms in the covariance matrix are set to zero:

$$\hat{x}_1(t_1) = y(t_1)$$
$$\hat{x}_2(t_1) = 0 \tag{2.2}$$
$$P(t_1) = \begin{bmatrix} \sigma_y^2(t_1) & 0 \\ 0 & \langle \text{Arbitrary large value} \rangle \end{bmatrix}$$

The number of hits in the track file or other track file quality indicator can be used to interpret the state vector and its covariance if needed.

### 2.2.2 Full Initialization on the Second Radar Return

On the second radar return, we set the position state to the most recent return and the position rate state to the observed position rate:

$$\begin{aligned} \hat{x}_1(t_2) &= y(t_2) \\ \hat{x}_2(t_2) &= \frac{y(t_2) - y(t\ )}{t_2 - t_1} \end{aligned} \tag{2.3}$$

The covariance matrix is properly set as that of these two values:

$$\begin{aligned} P(t_2) &= \mathrm{Cov}\left\{ \begin{bmatrix} y(t_2) \\ \dfrac{y(t_2) - y(t_1)}{dt} \end{bmatrix} \right\} \\ &= \begin{bmatrix} \sigma_y^2(t_2) & \dfrac{\sigma_y^2(t_2)}{dt} \\ \dfrac{\sigma_y^2(t_2)}{dt} & \dfrac{\sigma_y^2(t_1) + \sigma_y^2(t_2)}{dt^2} \end{bmatrix} \end{aligned} \tag{2.4}$$

At this point the track file is fully initialized and the state vector estimate and its covariance can be used and interpreted normally.

## 2.3 Extrapolation and Update

The state vector extrapolation is trivial,

$$\begin{aligned} dt &= t_i - t_{i-1} \\ \tilde{x}_1 &= \hat{x}_1(t_{i-1}) + \hat{x}_2(t_{i-1}) \cdot dt \\ \tilde{x}_2 &= \hat{x}_2(t_{i-1}) \end{aligned} \tag{2.5}$$

The covariance extrapolation is

$$\tilde{P} = \Phi \cdot P_{\_} \cdot \Phi^T + Q \cdot dt \tag{2.6}$$

We take the process noise covariance matrix $Q$ as diagonal. Then, we can write the covariance extrapolation explicitly as

$$\tilde{P} = \begin{bmatrix} p_{11}(t_{i-1}) + 2 \cdot p_{12}(t_{i-1}) \cdot dt + p_{22}(t_{i-1}) \cdot dt^2 + q_{11} \cdot dt & p_{12}(t_{i-1}) + p_{22}(t_{i-1}) \cdot dt \\ p_{12}(t_{i-1}) + p_{22}(t_{i-1}) \cdot dt & p_{22}(t_{i-1}) + q_{22} \cdot dt \end{bmatrix} \tag{2.7}$$

The measurement sensitivity matrix $H$ is very simple for this tracker:

$$H = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{2.8}$$

The Kalman gain is also very simple

$$\begin{aligned} K &= \tilde{P} \cdot H^T \cdot \left( H \cdot \tilde{P} \cdot H^T + R \right)^{-1} \\ &= \frac{1}{\tilde{p}_{11} + \sigma_y^2} \cdot \begin{bmatrix} \tilde{p}_{11} \\ \tilde{p}_{12} \end{bmatrix} \end{aligned} \tag{2.9}$$

The covariance update is the key, because this is the critical numerical step. Since the Kalman gain is unmodified from the minimum variance solution, we can use the short form as the simplest algebraic equation:

$$
\begin{aligned}
P &= \left(I - K \cdot H\right) \cdot \tilde{P} \\
&= \left( \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \frac{1}{\tilde{p}_{11} + \sigma_y^2} \cdot \begin{bmatrix} \tilde{p}_{11} \\ \tilde{p}_{12} \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \end{bmatrix} \right) \cdot \tilde{P} \\
&= \frac{1}{\tilde{p}_{11} + \sigma_y^2} \cdot \begin{bmatrix} \tilde{p}_{11} \cdot \sigma_y^2 & \tilde{p}_{12} \cdot \sigma_y^2 \\ \tilde{p}_{12} \cdot \sigma_y^2 & \tilde{p}_{11} \cdot \tilde{p}_{22} - \tilde{p}_{12}^2 + \tilde{p}_{22} \cdot \sigma_y^2 \end{bmatrix} \\
&= \frac{1}{1 + \tilde{p}_{11} / \sigma_y^2} \cdot \begin{bmatrix} \tilde{p}_{11} & \tilde{p}_{12} \\ \tilde{p}_{12} & \tilde{p}_{22} + \left|\tilde{P}\right| / \sigma_y^2 \end{bmatrix}
\end{aligned}
\tag{2.10}
$$

The form given in the last line of (2.10) is very simple and robust. Because it contains no subtractions, it is not subject to numerical problems. The implicit subtraction in the determinant can be accompanied by a check, and the determinant set to zero when a negative result is detected, but this should never happen when the covariance matrix is properly initialized. It is simple to show that

$$\left|P\right| = \frac{1}{1 + \tilde{p}_{11} / \sigma_y^2} \cdot \left|\tilde{P}\right| \tag{2.11}$$

and an examination of (2.6) and (2.7) shows that

$$\left|\tilde{P}\right| \geq \left|P\left(t_{i-1}\right)\right| \tag{2.12}$$

with equality only when the process noise matrix *Q* is zero. Thus the covariance matrix will always be well-conditioned, and no numerical problems will ever be observed with this tracker when properly initialized.

## *2.4 Making the Tracker Adaptive*

The tracker error is a scalar,

$$x_e = y\left(t_i\right) - \tilde{x}_1\left(t_i\right) \tag{2.13}$$

This quantity can be compared to its variance to provide a statistic,

$$z = \frac{y\left(t_i\right) - \tilde{x}_1}{\sqrt{\sigma_y^2 + \tilde{p}_{11}}} \tag{2.14}$$

This statistic is Gaussian with zero mean and variance one, so it can be compared with a set threshold for determination of proper association of radar returns to the track files. This value, squared, has an exponential distribution with a mean of one. We can use this fact, with the knowledge that the tracker error will increase when the Kalman gain is too low, and make the process noise matrix entries proportional to $z^2$. The process noise matrix becomes

$$Q = \begin{bmatrix} qc_{11} & 0 \\ 0 & qc_{22} \end{bmatrix} \cdot \left( \frac{\left(y\left(t_i\right) - \tilde{x}_1\left(t_i\right)\right)^2}{\sigma_y^2 + \tilde{p}_{11}\left(t_i \middle| Q = 0\right)} \right) \tag{2.15}$$

where $\tilde{p}_{11}\left(t_i \middle| Q = 0\right)$ is the extrapolated position state variance with the process noise taken as zero.

The process noise coefficients $qc_{11}$ and $qc_{22}$ are selected to give the proper process noise for $z=1$ and adjusted to optimize tracker performance as part of the tracker tuning process. This device will make the tracker adaptive to changes such as aircraft turning or acceleration, and will make this very simple tracker perform well as a range-only or bearing-only tracker with aircraft passing close by the radar.

This adaptive device is derived from the classical approach of estimating the process noise covariance from the measurements and making the elements of the process noise elements of the state vector. The results are similar to this approach for a Kalman filter except that

- Adjustment of the process noise is effective in the next measurement when elements of the process noise covariance are states, but are effective for the current measurement, making the tracker adaptation more quickly.
- The proportionality constant is determined empirically in our approach as presented here, whereas the quantity is given by the Kalman update with no user constants when the elements of the process noise matrix are states.

# 3 The Serious Options

## *3.1 Batch Estimators*

### 3.1.1 When You Need a Batch Estimator

There are three situations in which a batch estimator is truly required:

- A small batch estimator should be used to initialize a Kalman filter from the first few measurements, but not all the states are observable from a single measurement, and *ad hoc* initialization is unsatisfactory.
- When an unbiased, efficient estimator is required for a problem that does not have process noise, such as in some problems in offline data analysis, or real-time problems such as estimation of time of start of turn or descent of an aircraft from the radar hit history data in the track file.

There are other times when a batch estimator should be considered along with a sequential estimator (Kalman filter):

- When a radar contact or other data has behavior that can be modeled without the use of process noise.
- When the problem can be posed algebraically so that the parameters to be estimated are constant over the data base available, such as orbital parameters of a contact with a ballistic trajectory – such as a satellite, missile, or artillery shell.

### 3.1.2 How to formulate a batch estimator

There are several types of batch estimators that are commonly used:

- Maximum Likelihood estimators (MLEs).
- Maximum *a priori* (MAP) estimators.
- Least squares estimators.
- Bayesian mean estimators.

Most often the least squares or and MLE is the method of choice. Most least squares methods can be formulated by looking at an MLE and simplifying it by omitting unknown variances or other simple and obvious modifications; exceptions include least-squares formulations that assume approximate values for unknowns to produce linear or quadratic functions of the unknowns in terms of the measurements. Here we will focus on the MLE for nonlinear relationships between the measurements and the states.

In the lectures of Feburary 23, we showed that the MLE for a set of measurements $\underline{y}$ that were related to a set of states $\underline{x}$ through a nonlinear relationship could be found through the relationship

$$\hat{\underline{x}} = \tilde{\underline{x}} + P \cdot H^T \cdot R^{-1} \cdot \left( \underline{y} - \underline{h}\left( \tilde{\underline{x}} \right) \right) \tag{3.1}$$

where $P$ is the covariance of the estimate,

$$P^{-1} = \tilde{P}^{-1} + H^T \cdot R^{-1} \cdot H \tag{3.2}$$

the vector function $\underline{h}(\underline{x})$ is an algebraic description of the measurements in terms of the states, $R$ is the covariance of Gaussian measurement errors, and the measurement sensitivity matrix $H$ is the gradient of $\underline{h}(\underline{x})$ with respect to the states $\underline{x}$:

$$H = \frac{\partial \underline{h}\left( \underline{x} \right)}{\partial \underline{x}} \tag{3.3}$$

When we begin with no *a priori* information, as is usually the case, we take the inverse of the covariance of the initialization vector $\tilde{P}^{-1}$ as zero. When the states are nonlinear functions of the measurements, the matrix $H$ is a function of the states. Thus both (3.1) and (3.2) will be functions of the states. The way this is handled is to iterate (3.1), recomputing $H$ and $P$ each iteration, and driving the estimation error $\left( \underline{y} - \underline{h}\left( \tilde{\underline{x}} \right) \right)$ to zero.

In many practical cases, the iteration of (3.1) can provide challenges:

- The nonlinear behavior of the recursion, which can be viewed as Newton's method in N-space, may provide corrections to the state vector that are not directed toward the general solution. This can cause implementation to fail to converge, or even to diverge with computer issues such as overflow.
- The iteration minimizes the cost function $J = \left( \underline{y} - \underline{h}\left( \tilde{\underline{x}} \right) \right)^T \cdot R^{-1} \cdot \left( \underline{y} - \underline{h}\left( \tilde{\underline{x}} \right) \right)$ by finding the optimum value of the state vector. The behavior of the gradient of cost function near the solution may be sufficiently nonlinear so that the existence or location of a solution may not be at all evident, particularly with large amounts of data.

The solution to obtaining convergence is a staged approach. The sequence of events is

- Begin with as few data points as possible that will provide complete observability of the state vector. Select them as approximately evenly spaced through the database (first and last, first middle and last, etc.). Execute the batch estimator for this simplest problem. Do whatever is required, including methods such as a search over the parameter space for an appropriate initialization of the state vector for a properly converging iteration, as necessary.

- Add data points approximately centered between the data points that you used to obtain the first solution. Use the solution from the first few data points to initialize the recursion. One or two iterations should suffice to update the iteration to account for the new values.
- Again add data points approximately centered between the data points already being used and update the solution, and repeat until all data points have been used.

Other methods are important in making nonlinear iterations work:

- Select the coordinate system in which the states are posed so that the arithmetic signs of the elements of *H* are not functions of the variables. If possible, minimize variation of the magnitudes of the elements of *H* as a function of the state variables.
- Limit the amount that the magnitude of the correction of the state vector may take in any given iteration. This can be important in the first estimation using two or three data points.
- Edit the input data – examine $\left(y_i - h_i\left(\underline{\hat{x}}\right)\right) \Big/ \sqrt{r_{ii} + \left(H \cdot P \cdot H^T\right)_{ii}}$ and threshold it, and throw out data that clearly won't provide good information to the algorithm.
- Put limits on excursion of each state variable based on reasonable bounds expected in the scenario.

Use is similar for a batch estimator with input data that is provided sequentially, as for a tracker that estimates a trajectory with zero process noise, or a maneuver detector. However, early in the run, an additional data point may provide a non-viable data base for estimation – a solution may not exist that is consistent with reasonable expectations. When this occurs, pitch the solution and re-initialize when the next measurement comes in, using all of the tricks above. This may need to be done two or three times when the data is noisy. After a reasonable data base has accrued, state vector updates will become routine single applications of the recursion relationship with each additional data point.

## *3.2 Square Root Filters*

There are several square root filters in use:

- The Potter square root filter.
- The upper triangularization filter.
- The UDUT factorization filter.
- The square root information filter (SRIF).

All of them operate by using a square root of the covariance matrix (or it's inverse, in the case of the SRIF). Conceptually, variances are replaced by RMS errors in the formulations, so that dynamic ranges of the variables are the square root of those seen in operations that use the covariance matrix directly.

The Potter sqaure root filter was the first. It begins with a triangular factorization of the initial covariance matrix, and the square root covariance matrix *S* is carried through its implementation. The upper triangularization filter uses a square root of the covariance

matrix $P$ such that $P = U \cdot U^T$ and $U$ is upper triangular throughout the implementation. The UDUT is a simplification of this filter that carries the diagonal terms as a diagonal matrix $D$ and an upper triangular matris $U$ that has diagonal elements that are all ones.

The SRIF is a different animal entirely. It is based on formulating extrapolations and update of the Kalman filter as normalized data equations, or equations of the form

$$A \cdot \underline{x} = \underline{z} \tag{3.4}$$

where the matrix $A$, in general, has more rows than columns, and the vector $\underline{z}$ is made up of uncorrelated Gaussian variables with mean zero and unity variance. The covariance of both sides of this equation is the identity matrix. The equation is left-multiplied by Householder transformations to triangularize $A$, providing a solution for $\underline{x}$.

The square root filters most often used are the UDUT and the SRIF. They are compared with the Kalman filter without factorization in the table below.

**Table 1 Comparison of Kalman Filter Implementations**

| Property | SRIF | UDUT | Kalman |
|---|---|---|---|
| Numerical Conditioning | Excellent | Excellent | Fair |
| Handling of unobservable or poorly observable states | Excellent | Fair to Good | Poor |
| Numerical efficiency (operation count) | Excellent | Excellent | Good |
| Drop-in for Kalman | NO | YES | NA |

# 4 Operation Counts with Various Kalman Filters

Operation counts for implementation of various Kalman filters is taken from *Factorization Methods for Discrete Sequential Estimation*, by Gerald J. Bierman, Academic Press (1977), ISBN 0-12-097350-2, pp. 111. Only the covariance extrapolation and the state and covariance update are included, because the state extrapolation is the same for all of them. The operation counts reported there are from implementations given as design language (simplified symbolic FORTRAN) source code in that book. They are summarized in the table below.

**Table 2 Operation Counts for *n* States and *m* Measurements**

| Filter Type | Adds | Multiples | Divisions | Square Roots |
|---|---|---|---|---|
| Short Form | $\left(1.5\cdot n^2+3.5\cdot n\right)\cdot m$ | $\left(1.5\cdot n^2+4.5\cdot n\right)\cdot m$ | $m$ | 0 |
| Joseph Stabilized Form | $\left(4.5\cdot n^2+5.5\cdot n\right)\cdot m$ | $\left(4.5\cdot n^2+7.5\cdot n\right)\cdot m$ | $m$ | 0 |
| Potter Square Root | $\left(3\cdot n^2+3\cdot n\right)\cdot m$ | $\left(3\cdot n^2+4\cdot n\right)\cdot m$ | $2\cdot m$ | $m$ |
| UDUT Factorization | $\left(1.5\cdot n^2+1.5\cdot n\right)\cdot m$ | $\left(1.5\cdot n^2+5.5\cdot n\right)\cdot m$ | $n\cdot m$ | 0 |
| Upper triangular factorization | $\left(1.5\cdot n^2+3.5\cdot n\right)\cdot m$ | $\left(2\cdot n^2+5\cdot n\right)\cdot m$ | $2\cdot n\cdot m$ | $n\cdot m$ |
| Normalized data equation | $\frac{1}{3}\cdot n^3+\frac{1}{2}\cdot n^2+\frac{7}{6}\cdot n$ | $\frac{1}{3}\cdot n^3+2\cdot n^2-\frac{1}{3}\cdot n$ | $2\cdot n$ | $n$ |
| SRIF | $n^2+3\cdot n$ $+\left(n^2+2\cdot n\right)\cdot m$ | $1.5\cdot n^2+3.5\cdot n$ $+\left(n^2+3\cdot n\right)\cdot m$ | $n$ | $n$ |

A modern microprocessor with a hyperscalar architecture will execute most arithmetic operations, including division and even the square root, in a single machine cycle because execution will pass on to other functions such as subscript setup or deferred memory operations and come back to use the output. Standard operations such as floating point arithmetic operations and the square root, and the elementary transcendental functions of logarithm, exponential, and the trigonometric functions and their inverses are done in a FIFO stack by an auxiliary processor that is essentially asynchronous with the main ALU. As such, we can reasonably give equal weight to addition, multiplication, and division. We will give a weight of four to the square root operation, although that is probably conservative. Using these weights, we have the operation counts for various numbers of states and measurements of the following table.

In the table, we include only the Joseph stabilized form, the UDUT factorization and the Square Root Information Filter (SRIF) as the most-often used algorithm types. The figure in parenthesis is the time in microseconds for a 1 GHz clock, assuming that four master clocks are used for one microprocessor cycle.

**Table 3 Kalman Update Times for Three Functions**

| Function | N | M | Joseph | UDUT | SRIF |
|---|---|---|---|---|---|
| ASR Tracker | 4 | 2 | 394 (1.576 $\mu s$) | 160 (0.64 $\mu s$) | 190 (0.76 $\mu s$) |

| | | | | | |
|---|---|---|---|---|---|
| 3-D Radar | 6 | 4 | 1612 (6.448 $\mu s$) | 624 (2.496 $\mu s$) | 567 (2.268 $\mu s$) |
| 2-D Fusion | 4 | 4 | 788 (3.152 $\mu s$) | 320 (1.28 $\mu s$) | 294 (1.176 $\mu s$) |

The functions represented in the table are an ASR tracker that does not track altitude and has range and bearing for measurements, a three dimensional radar with three-channel Monopulse and Doppler measurements, and two-dimensional data fusion using position and velocity states without a bias state. With an added bias state for the fusion operation, operation counts and times are 40% to 50% larger.

The times compare to about 80 milliseconds, not microseconds to correlate 100 radar hits with 400 track files, for an ASR tracker. Filtering of candidate track files by angle sector and general magnitude of range, as well as other data can reduce this estimate dramatically, but association remains the largest use of computational resources when a large numbers of targets are present.

The bottom line is that the computer time required to perform the operations are actually less than the Joseph stabilized form when the square root filters are used. None of the Kalman filter methodologies use significant computer resources compared to those required for the data association algorithms, which are the same for any Kalman filter method.

# 5 Assignment

Examine your problems and decide which should be approached with a batch estimator, which would benefit from addition of a bias state in the estimator, and write a short paragraph on what you believe would be best and why. Please reply by e-mail by noon Wednesday.