

# Design and Implementation of a Generic Kalman Filter in Ada

Leslie Hyll and Larry Gearhart

TRW/A&SG/MEAD  
Dayton Engineering Laboratory

## Abstract

The Ada generic package construct offers an opportunity for the development of reusable modules embodying very high level algorithms. One such example is the Kalman Filter. Kalman Filters come in a number of variants: full-state vs. error-state, continuous vs. discrete, classical vs. square root vs. U-D, etc. One therefore envisions not a single Ada package that satisfies all requirements, but a single package for each variant of Kalman Filter. This paper describes the general classification and design problem and then examines the design trade-offs for a specific flavor, the discrete, error-state, Bierman Filter. This version was implemented by TRW for the Integrated Test Bed (ITB) project, for use in the Sandia Inertial Terrain-Aided Navigation (SITAN) filter.

The paper describes the use of the Kalman Filter in the SITAN application. The SITAN navigation problem is briefly described, along with elements of the navigation solution. The paper defines the approach taken to adapt the generic Kalman Filter package to SITAN, the advantages of the approach for accommodating later upgrades to SITAN, and the inevitable performance trade-offs of a generic design.

## Introduction

The Sandia Inertial Terrain-Aided Navigation (SITAN) system provides a means of autonomous and semi-passive

navigation which integrates a stored terrain model with an inertial reference and terrain sensors. The purpose of this paper is to define the approach taken in the ITB project to implement SITAN in Ada, to highlight the core software - a generic Kalman Filter package - and to show the relevance of the approach to upgrading and adapting the implementation. Several key ideas of software design are emphasized. A design for a generic Ada Kalman Filter is presented which is relevant to general Kalman Filtering applications.

The work described was carried out as part of the Integrated Test Bed (ITB) System Integration project, F33615-87-D-1451/003, under the sponsorship of the Air Force Wright Research and Development Center/Avionics Laboratory with additional sponsorship by the Naval Air Development Center/System Integration Laboratory. A more detailed report on this work may be found in the ITB SITAN technical report [ref. 1].

## SITAN

SITAN begins with an estimate of aircraft state provided by an inertial unit. It then filters that information given predicted vs. actual mapping sensor return data. (See figure 1.) The prediction of mapping sensor data relies upon a detailed map of the sensor environment. (See figure 2.) Filter performance is a function of state model fidelity, measurement noise, mapping data error and efficiency. Measurement noise includes the sensor noise and errors introduced by fitting the current estimate of position to the map. The

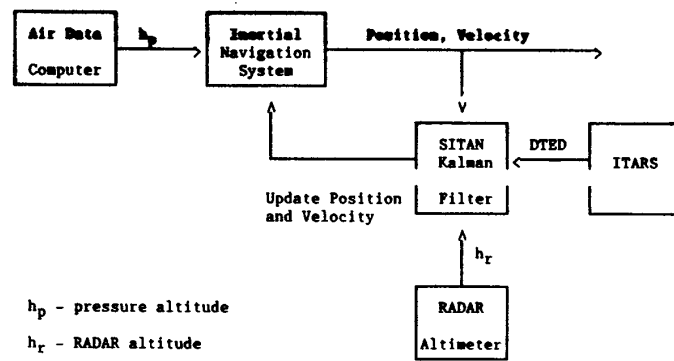


Figure 1 - The SITAN Kalman Filter

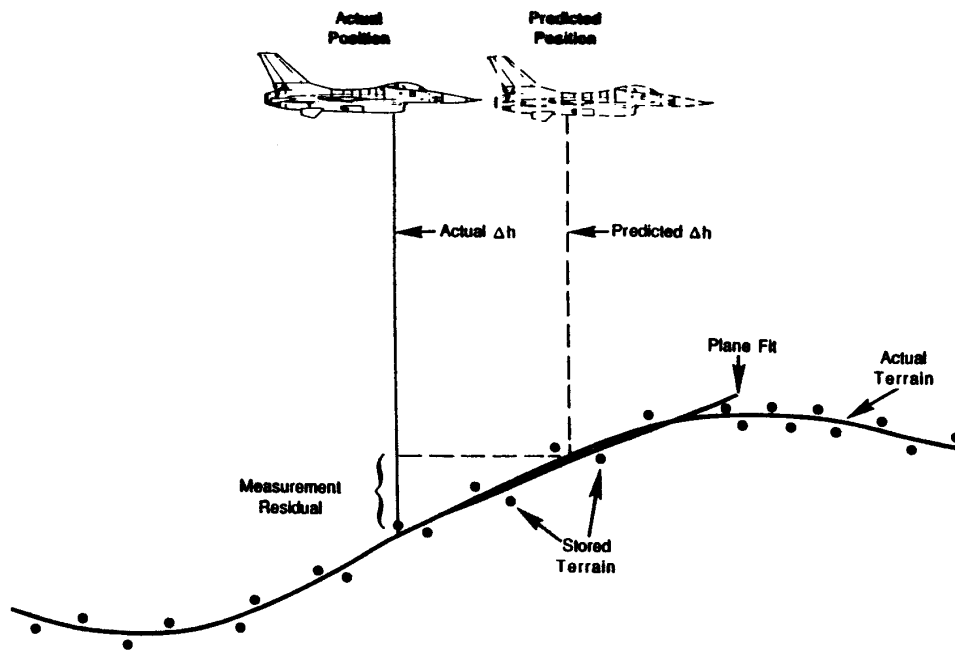


Figure 2. SITAN Measurement Residuals

position fit statistics complicate the process of modelling the measurement noise for state estimation and for the purpose of making critical decisions.

SITAN operates in three execution states: acquisition, track and suspended. When SITAN is first brought up, it enters suspended state and waits for a command to enter acquisition. In acquisition it obtains an initial estimate of position from the previous NAV mode. It then subjects that estimate, and positions within a grid surrounding the estimate, to a fit using an array of simplified filters. (SITAN acquisition uses an array of simplified Kalman filters indexed and initialized by the surrounding grid points. By testing for clustering, and testing the best fit within the closest cluster, SITAN acquisition chooses a "winner".) After enough iterations of the simplified filters have occurred to decide upon a winner, the winning grid point becomes the initial estimate of position supplied to the track filter. SITAN then enters track state and updates the estimate of position and velocity based upon data from a radar altimeter. If the test of mapping fit is insufficient to provide confidence in the acquisition result, SITAN reenters suspended state, requiring intervention by the pilot or other higher level system. If tracking performance is of high quality, SITAN may update the Inertial Navigation System.

### Generic Filter Design

Any generic filter implementation should satisfy the following requirements:

- It should adapt easily to changes in the system or measurement model.
- It should allow for time dependency in the model (the non-linear, or "extended", filter case).
- It should accommodate either a full-state or an error-state model, as required.

- Designs which incorporate this generic filter should be nearly as efficient as their custom versions.

- It should provide normalized measurement residuals, but not be involved in the statistical use of those residuals other than for measurement updates.

Lastly, to accommodate the special case of SITAN Acquisition, the design should make it possible to declare "filter arrays" without a substantial loss of efficiency.

### Application to SITAN

Application of the generic filter package within a SITAN implementation is pictured in figure 3.

In this mechanization, the generic Kalman Filter equations are supplied with details of the state and measurement models as needed. The filter package is instantiated twice: once for the track filter and once for the acquisition filter. With the proper compiler options, this can lead to substantial code sharing without sacrificing data storage efficiency.

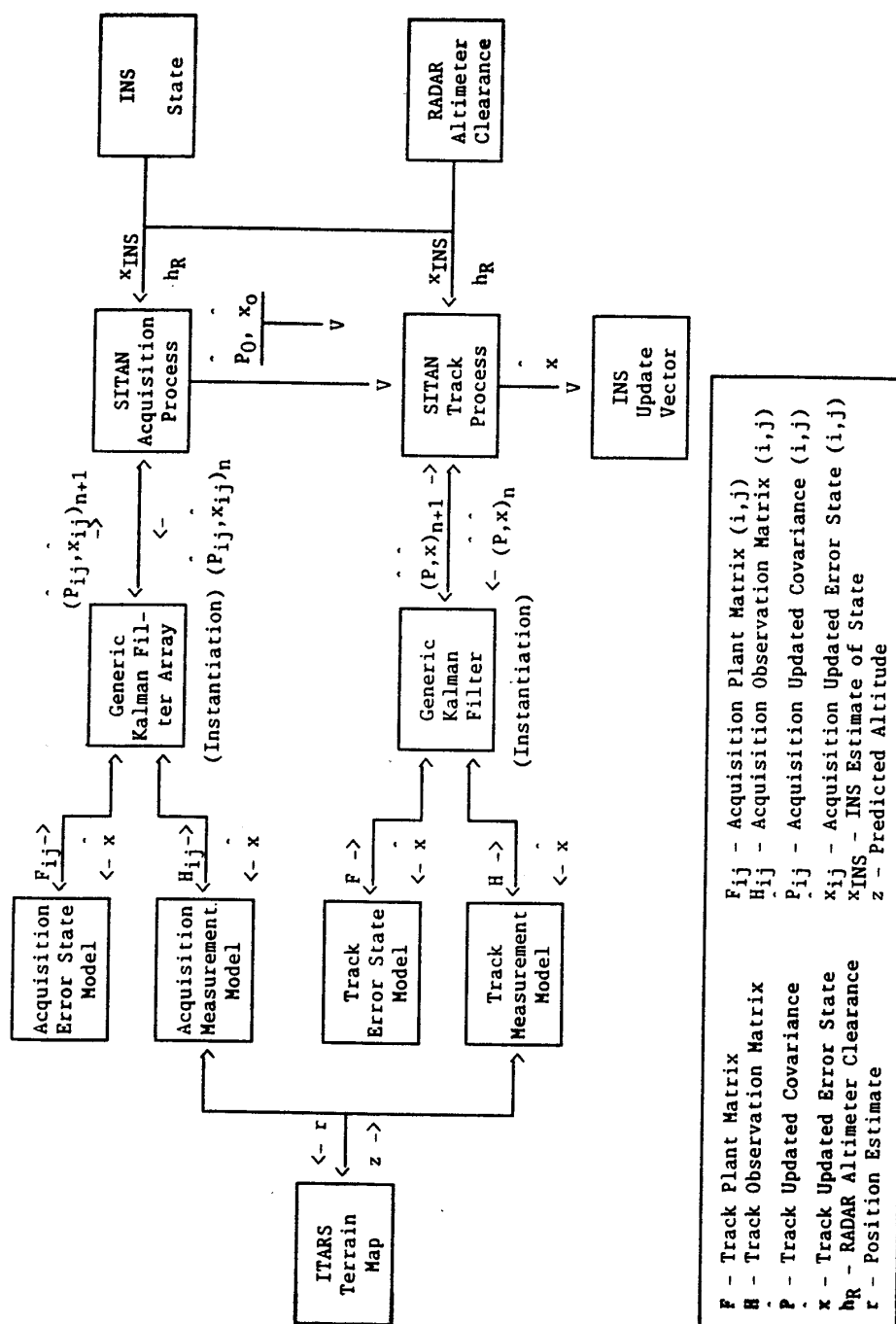


Figure 3 - Application of the Generic Filter Module to SITAN

## The Generic Filter

The generic filter is designed as an Ada generic package. It is designed to be an application-independent implementation of a specific update method. The application instantiates the package with the dimension of the state vector and the precision of the calculations. It then supplies model data in procedure calls, as appropriate, prior to calling Propagate and Update routines.

The following generic package specification encapsulates the desired services of a Kalman Filter:

```
with Vectors, Matrices;
generic
  type Real is digits <>; -- the base floating point type
  N : Positive; -- state dimension
package Kalman_Filter is

  package Real_N_Vectors is new Vectors (Real, N);
  package Real_M_by_N_Matrices is new Matrices (Real, M, N);

  type State is Vector (1..N);
  type State_Matrix is Matrix (1..N, 1..N);
  type Observation_Vector is Vector (1..N);

  procedure Set_State_Matrix (F : in State_Matrix);
  procedure Set_State_Noise (Q : in State_Matrix);
  procedure Set_Observer (h : in Observation_Vector);
  procedure Set_Measurement_Noise (r : in Real);
  procedure Propagate (X : in out State;
    P : in out State_Matrix);

  procedure Update (z : in Real;
    X : in out State;
    P : in out State_Matrix;
    R : out Real); -- normalized residual.

end Kalman_Filter;
```

Figure 4 - Generic Kalman Filter

The above definition gives a simplified version of a more general approach. (See [ref. 1] for an adaptation to the Bierman U-D Implementation.) This interface is practical for the classical Kalman Filter implementation, but is not quite appropriate for a Square-Root or U-D Filter. One can introduce additional types in the specification part for diagonal and triangular/symmetric matrices. Alternatively, these definitions may be hidden to the user by placing them in a private part of the package specification, and by

introducing an abstract "Filter\_State" data type. The specification part might be modified as follows:

```
type Filter_State is private;
procedure Initialise_Filter (X : in State;
  P : in State_Matrix;
  S : out Filter_State);

procedure Propagate (X : in out State;
  S : in out Filter_State);

procedure Update (z : in Real;
  X : in out State;
  S : in out Filter_State;
  R : out Real); -- normalized residual.

procedure Get_Covariance (S : in Filter_State;
  Q : out State_Matrix);
```

A full definition of the Filter State, suitable to the underlying implementation, would then be given in the private part.

In a more conventional Object-Oriented design, one would defer declaration of the filter state to the generic package body. In contrast, the approach defined here can be adapted to the requirements of the SITAN Acquisition Filter. One can define an array of Filter\_States to store factored covariance data for each filter.

## Application to Error State Models

Given a measurement vector,  $H$ , the measurement model is given by:

$$z = Hx + v$$

where  $x$  is the error state and  $v$  is white Gaussian measurement noise. Given this model, the Generic Kalman Filter will compute the residual,  $r$ , using the formula

$$r = z - Hx$$

If we assume that  $x$  is an error state, then we must compute  $z$  as an "error" as well. Considering the SITAN example, suppose that  $x$  is the difference between true and INU state. Then one may compute  $z$  as the difference between radar-measured clearance and the prediction of clearance given by the INU

estimate of position. In detail, the SITAN measurement term,  $Hx$ , is given by.

$$Hx = -x\_slope*(x_t - x_i) - y\_slope*(y_t - y_i) + (h_t - h_i)$$

where  $x_t - x_i$  and  $y_t - y_i$  are the x and y components of error between true and INU estimates. (See the ETMP report [ref. 2] for details.) Define  $f(x, y)$  to be ITARS terrain height at position  $x, y$ . Consider the expression

$$f(x_t, y_t) - x\_slope*(x_t - x_i) - y\_slope*(y_t - y_i)$$

For  $x_t$  close to  $x_i$  and  $y_t$  close to  $y_i$ , this expression is very close to the INU-predicted terrain height,  $f(x_i, y_i)$ , as may be seen from a Taylor Series expansion of  $f$ . Suppose we model radar clearance as true clearance plus white noise, as in

$$radar = h_t - f(x_t, y_t) + v$$

Then we may calculate

$$\begin{aligned} z &= radar - (h_i - f(x_i, y_i)) \\ &= h_t - f(x_t, y_t) + v - h_i + f(x_i, y_i) \end{aligned}$$

Using the Taylor approximation, this expression simplifies to

$$\begin{aligned} z &= -x\_slope*(x_t - x_i) - y\_slope*(y_t - y_i) + h_t - h_i + v \\ &= H(x_t - x_i) + v \end{aligned}$$

which is in the form required for application of the generic filter.

This same methodology is readily applied to other contexts. (See Maybeck [ref. 3] for a similar example.)

### Trade-offs and Limitations

Using the generic approach, one must discard potential optimizations which have appeared in the typical Kalman Filter implementation. For example, many optimizations take advantage of the sparsity of the system plant matrix. The generic approach defined above makes no assumption concerning the character of this matrix. Even its dimension is a "parameter". On the other hand, one may

assume that the covariance matrices are symmetric, and save both space and computation time. One should be careful with these optimizations, however, to avoid paying an added cost in the form of extra matrix procedures for the special case of a symmetric matrix.

The reader may note that the packages Vectors and Matrices are instantiated within the generic filter package. This could produce awkward dependencies within the package structure of an application, if the Vector or Matrix operations are used outside of the filter. Unfortunately, it doesn't work to instantiate these packages outside of the filter, since the filter's formal parameters would be incompatible with the actual parameters of the package instantiations of the packages Vectors and Matrices.

### Conclusions

SITAN achieves a fairly high level of autonomous navigation performance over many hours of flight time. The various implementations of the SITAN algorithms are based upon models of navigational error dynamics and terrain measurements which are conceptually very simple. To date, the implementations have been tailored at every stage to details of these models in order to achieve maximum execution efficiency. Unfortunately, the result has been that improvements in the algorithms design ripple throughout the software, requiring almost a complete rewrite. Furthermore, this is true of nearly all previous Kalman Filter development projects.

It is possible to implement the SITAN filter in Ada in such a way that the generic filter algorithms are separated from the specific design of the state and measurement equations. With additional effort, this genericity can be enhanced to cover a very wide range of Kalman Filter applications. While some sacrifices in efficiency are inevitable here, these have been minimized.

The entire thrust of this approach is to raise the level of expression of the formulae for the purpose of reducing the impact to the code from changes in the system models. One may implement a standard Kalman Filter algorithm in this generic form, test it on one or two examples, and have confidence that it will continue to work in future applications. Furthermore, one may safely alter the application models as the need dictates, e.g. for improved tracking accuracy or greater efficiency. This approach is in line with the migration to more powerful hardware and the need to build a software infrastructure for future development.

#### References

1. ITB SITAN: A Design for Reusable Ada Software, WRDC-TR-90-107, August 1990, Contract Number: F33615-87-D-1451/003.
2. Nordmeyer, R., Enhanced Terrain Masked Penetration - Final Technical Report, AFWAL-TR-86-1079, September, 1986.
3. Maybeck, P.S., Stochastic Models, Estimation and Control, Vol. 1, Academic Press, New York, 1979.