APPLICATION NOTE

ABSTRACT
This application note describes the use of the Kalman filter for sensory fusion. It covers the theory of operation as well as an implementation example based on the XA microcontroller.

AN715
Kalman filter implemented in the XA eases sensory fusion

Author: Zhimin Ding

1999 May 03

Philips Semiconductors

PHILIPS

Kalman filter implemented in the XA eases sensory fusion

AN715

Introduction

A microcontroller is often used to interface with sensors and generate control actions based on information gathered from sensors. For reliability and completeness, more than one sensor is generally used. If different sensors are employed to provide measurement of the same physical quantity, a better estimation can often be obtained by calculating weighted average of the individual measurements, assuming the noises they carry are not statistically correlated.

Sometimes it is useful to have sensors that measure different physical variables and the combined information is used for control decision making. The operation to combine information from such multi-modal sensors is called sensory fusion.

Kalman filters can be used to derive the best estimation by combining sensory input from different sources. It has been used most extensively in navigation since it was invented in 1960. Recently, it has been used in high-end GPS navigation systems to improve performance.

The concept of Kalman filtering can be illustrated in Figure 1. Suppose a human is traveling in one dimension. He has a positioning device that gives him estimations of where he is at time t1, t2, and t3, with some variance. For example, if the measurement says the person is at position t2, the real position could be anywhere around t2 with a certain distribution function as shown in the figure.

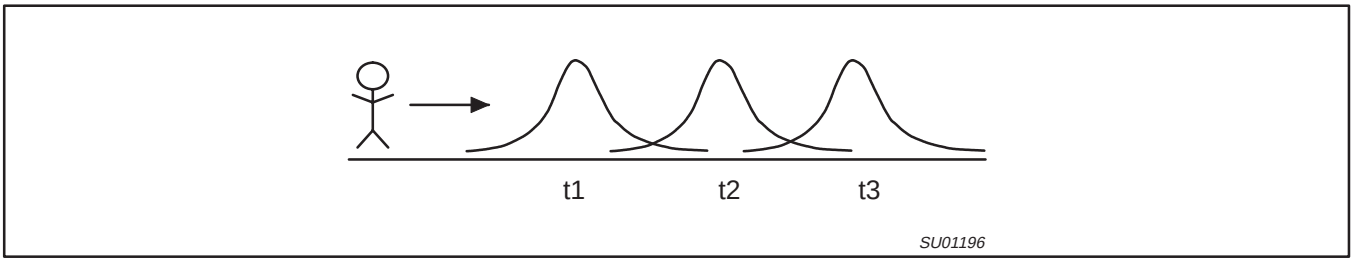

Figure 1. A traveler takes position measurements at time t1, t2, and t3.

Suppose this person also knows how fast he is traveling from a speed sensor. He can estimate the distance by integrating speed over time. Notice the variance increases with time—a phenomenon called drift. Since he has to integrate speed over time to come up with the estimate, the variances that characterize the quality of this kind of estimation will grow linearly over time. This is a typical problem for all inertia navigation systems.

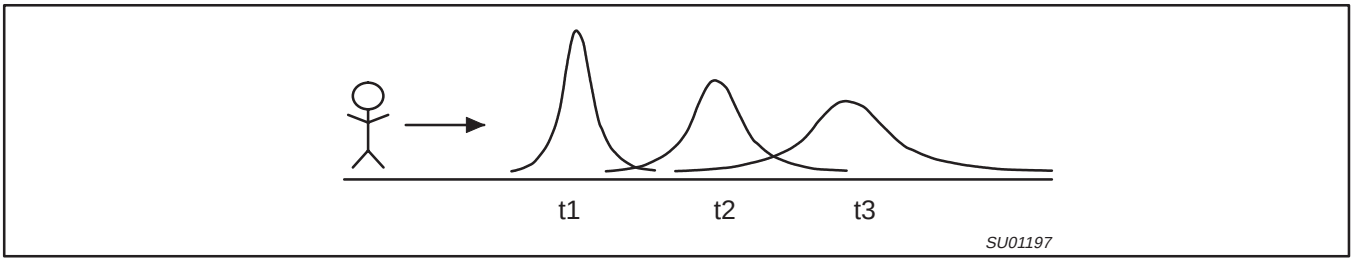

Figure 2. The traveler can also estimate the position by integrating speed measurements.

With Kalman filtering, the two measurements can be combined to produce a better measurement. As shown in Figure 3, the error variance from the combined estimation is both bounded and much reduced.

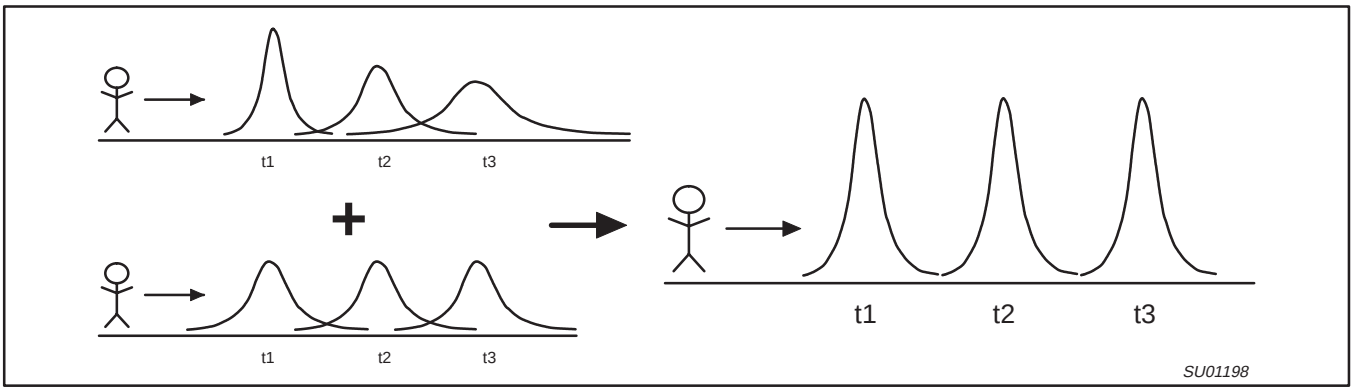

Figure 3. With Kalman filtering, speed and position measurements can be combined to produce better position estimates.

The condition for obtaining better results from Kalman filtering is the following:

- The relationship between different variables, such as speed and distance in this example, is known, i.e., distance += speed * dt.
- There is NO statistical correlation between the errors carried by the speed sensor and those carried by positioning device.
- All noises are white with zero mean.

When different sensing mechanisms are used to provide different measurement, there is usually no reason for the errors to be correlated.

Theory of operation

Let me start with some math equations. For those who do not like to read equations, the numerical example in the next section will make things appear simpler.

Suppose the dynamics of a system can be described as

$$x_{K+1} = \phi_k \, x_k + w_k \tag{1}$$

Assume that the system dynamics can be described by n system variables. x_k is an “n by 1” vector containing all the state variables. Φ_k is an “n by n” matrix that dictates the transition from x_k to $x_{k+1} \cdot w_k$ is called the plant noise. We assume it is white with known covariance Q_k.

Now, suppose part of all the state variables (e.g., m out of n) can be observed also. We use vector z_k to denote all the observable variables, we have:

$$z_k = H_k \, x_k + v_k \tag{2}$$

where z_k is an “m by 1” vector consists of all the observable state parameters. H_k is an “m by n” matrix that extracts observable variables from the state variables. There is some measurement noise denoted by v_k, with known covariance R_k.

Assuming our initial estimation of the state variables is $\hat{x}_k^-$, the error of such estimation can be defined as:

$$e_k^- = x_k - \hat{x}_k^- \tag{3}$$

and the covariance of the error is defined as P_k.

With Kalman filtering, the measurement of x can be improved by incorporating measurable variable z_k.

$$\hat{x}_k = \hat{x}_k^- + K_k\,(z_k - H_k\,\hat{x}_k^-) \tag{4}$$

K_k is an “n by m” matrix yet to be determined. This matrix is called the Kalman gain. It can be calculated as the following:

$$K_k = P_k^- H_k^T\,(H_k\,P_k^- H_k^T + R_k)^{-1} \tag{5}$$

The calculation of Kalman gain not only depends on the known measurement error covariance R_k, but also the state estimation covariance P_k^-. The superscript “–” means it is a prior estimate from the last step.

After calculation of Kalman gain, a new and improved estimation covariance can be obtained from:

$$P_k = (I - K_k\,H_k)P_k^- \tag{6}$$

The error covariance projected to the next step can be calculated as:

$$P_{k+1}^- = \phi_k\,P_k\,\phi_k^T + Q_k \tag{7}$$

Figure 4 explains the implementation of a Kalman filter.

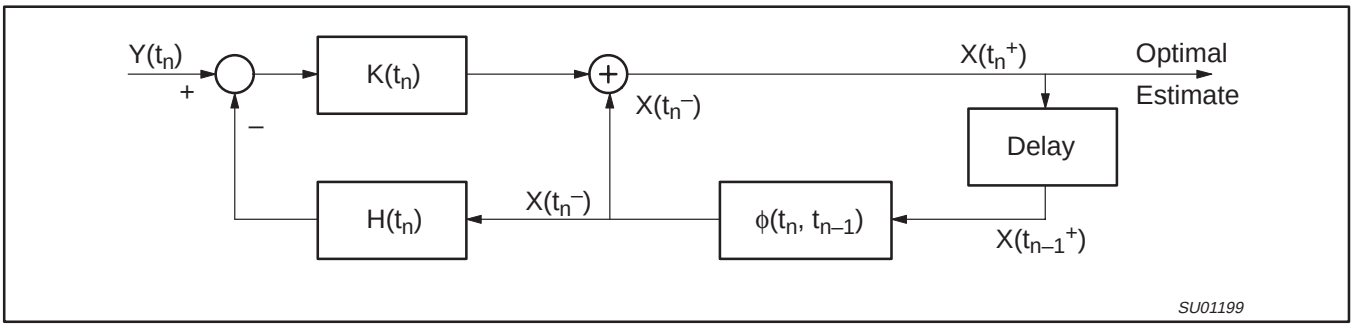

Figure 4. Kalman filter implementation.

Implementation example with the XA

Here we introduce a simple example for Kalman filter implementation. The implementation is realized in C and compiled to run on the XA. Part of the source code is included in the application note; for full source code, please contact the author via e-mail at zhimin.ding@sv.sc.philips.com .

Suppose an object is traveling in a two dimensional plane, its state can be described by vector $x = (px, py, vx, vy)$, where px, py are position coordinates, vx, vy are velocity in x and y directions. Since we know p(k+1) = p(k) + v(k)*dt, assuming dt = 0.1, we have:

$$x_k = \begin{bmatrix} 1 & 0.1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.1 \\ 0 & 0 & 0 & 1 \end{bmatrix} x_k + plant\_noise \tag{8}$$

Now we are interested in knowing the position based on position measurement, $z_k = (px, py)$:

$$z_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} x_k + measurement\_noise \tag{9}$$

Let us assume that the measurement error is about 10m rms (the covariance will be 100), we have:

$$R = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix} \quad (10)$$

Let us assume that the speed measurement is relatively accurate, with 2m/s/s rms:

$$Q = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix} \quad (11)$$

And let us assume that initially we know where the object is with uncertainty of 10 m. With Kalman filter, the average measurement error can be reduced from 10 m to 2.4 m.

The following is the code that calculate Kalman gain and error covariance. These lines of code should be included in a loop and be repeatedly executed:

```
/*
 *
 *.....Compute Kalman gain
 *
 */
     mult_mat_mat(tmp_nm,PMinus,HTranspose);
     mult_mat_mat(tmp_mm,H,tmp_nm);
     add_mat_mat(tmp_mm,tmp_mm,R);
     gaussj(tmp_mm);
     mult_mat_mat(tmp_nm,HTranspose,tmp_mm);
     mult_mat_mat(Gain,PMinus,tmp_nm);
/*
 *
 *.....Compute state estimation x^
 *
 */

     mult_mat_mat(tmp_m_1,H,x_hat_k);
     sub_mat_mat(tmp_m_1,z,tmp_m_1);
     mult_mat_mat(tmp_n_1,Gain,tmp_m_1);
     add_mat_mat(x_hat,x_hat,tmp_n_1);
/*
 *
 *   Compute P – the error covariance
 *
 */
     mult_mat_mat(tmp_nn,Gain,H);
     sub_mat_mat(tmp_nn,Ident,tmp_nn);
     mult_mat_mat(PPlus, tmp_nn, PMinus);
```

```
/*
 *
 *   Project ahead
 *
 */

     mult_mat_mat(tmp_nn, PPlus, PhiTranspose);
     mult_mat_mat(tmp2_nn, Phi, tmp_nn);
     add_mat_mat(PMinus, tmp2_nn, Q);
```

Notice that all the calculations are done between matrices. A matrix is defined as:

```
typedef struct matrix{
     int dim1;
     int dim2;
     float *m;
}matrix;
```

Examples of matrix operations are listed below:

```
void mult_mat_mat(matrix *out,matrix *in1,matrix *in2)
/*
 *
 *.....Multiply two matrices and return answer in a third matrix
 *
 */
{
     int i,j,k;

     for(i=0; i < out->dim1; i++)
          for(j=0; j < out->dim2; j++){
               out->m[(i * out->dim2) + j] = 0.0;
               for(k=0; k < in1->dim2; k++) {
                    out->m[(i * out->dim2) + j] =
                         out->m[(i * out->dim2) + j] +
                         in1->m[(i * in1->dim2) + k] *
                         in2->m[(k * in2->dim2) + j];

               }
          }
}
```

```
void add_mat_mat(matrix *out,matrix *in1,matrix *in2)
/*
 *
 *.....Add two matrices and return answer in a third matrix
 *
 */
{
     int i,j;

     for(i=0; i < out->dim1; i++)
          for(j=0; j < out->dim2; j++)
               out->m[(i * out->dim2) + j] =
                    in1->m[(i * out->dim2) + j] +
                    in2->m[(i * out->dim2) + j];
}
void sub_mat_mat(matrix *out,matrix *in1,matrix *in2)
/*
 *
 *.....Subtract two matrices and return answer in a third matrix
 *
 */
{
     int i,j;

     for(i=0; i < out->dim1; i++)
          for(j=0; j < out->dim2; j++)
               out->m[(i * out->dim2) + j] =
                    in1->m[(i * out->dim2) + j] -
                    in2->m[(i * out->dim2) + j];
}
```

At one place, a matrix needs to be inverted. The algorithm used is the Gauss-Jordan elimination algorithm. It can be found from “Numerical Recipes in C”.

XA performance

The example shown above was tested on the XA. The example takes only 24K of code and about 8K of data space. The calculations can be done within 0.1 second, thus allowing a navigation resolution of 0.1 second. The code is entirely written in C and compiled by TASKING C compiler V2.0.

Figure 5 is the result of a Kalman filter implemented according to the example shown above. The top graph shows the object movement trace compared with the estimations without Kalman filtering. The bottom graph shows position estimation with Kalman filtering. The estimation follows the real path much more closely with Kalman filtering.

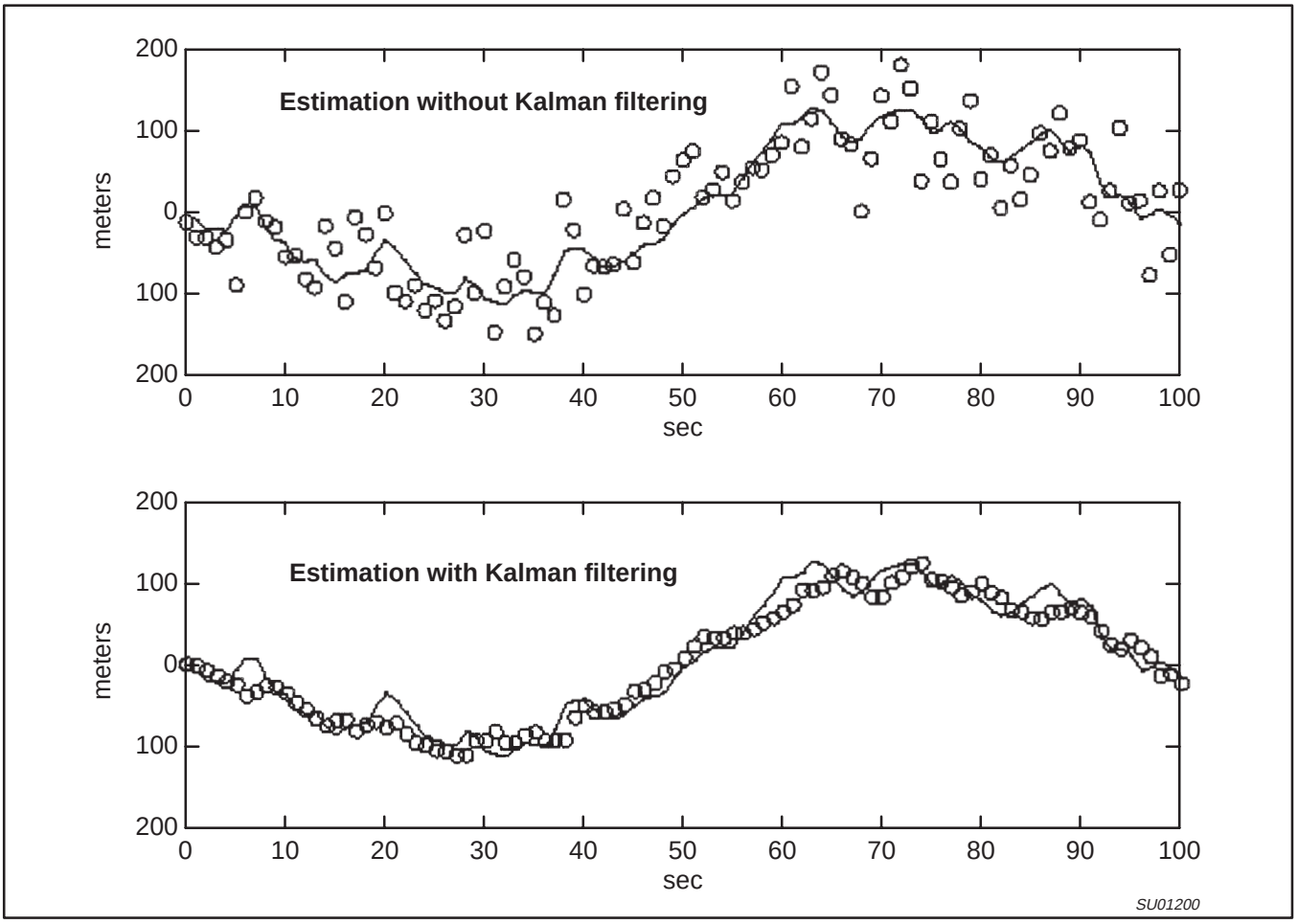

Figure 5. Kalman filter simulation result.

NOTES

Kalman filter implemented in the XA eases sensory fusion

AN715

Definitions

Short-form specification — The data in a short-form specification is extracted from a full data sheet with the same type number and title. For detailed information see the relevant data sheet or data handbook.

Limiting values definition — Limiting values given are in accordance with the Absolute Maximum Rating System (IEC 134). Stress above one or more of the limiting values may cause permanent damage to the device. These are stress ratings only and operation of the device at these or at any other conditions above those given in the Characteristics sections of the specification is not implied. Exposure to limiting values for extended periods may affect device reliability.

Application information — Applications that are described herein for any of these products are for illustrative purposes only. Philips Semiconductors make no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Disclaimers

Life support — These products are not designed for use in life support appliances, devices or systems where malfunction of these products can reasonably be expected to result in personal injury. Philips Semiconductors customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Philips Semiconductors for any damages resulting from such application.

Right to make changes — Philips Semiconductors reserves the right to make changes, without notice, in the products, including circuits, standard cells, and/or software, described or contained herein in order to improve design and/or performance. Philips Semiconductors assumes no responsibility or liability for the use of any of these products, conveys no license or title under any patent, copyright, or mask work right to these products, and makes no representations or warranties that these products are free from patent, copyright, or mask work right infringement, unless otherwise specified.

Philips Semiconductors
811 East Arques Avenue
P.O. Box 3409
Sunnyvale, California 94088–3409
Telephone 800-234-7381

© Copyright Philips Electronics North America Corporation 1999
All rights reserved. Printed in U.S.A.

Date of release: 05-99

Document order number: 9397 750 05746

Let's make things better.

Philips Semiconductors

PHILIPS