Using Nonlinear Kalman Filtering to Estimate Signals

Dan Simon
Revised September 10, 2013

It appears that no particular approximate [nonlinear] filter is consistently better than any other, though ... any nonlinear filter is better than a strictly linear one.[1]

The Kalman filter is a tool that can estimate the variables of a wide range of processes. In mathematical terms we would say that a Kalman filter estimates the states of a linear system. There are two reasons that you might want to know the states of a system:

- First, you might need to estimate states in order to control the system. For example, the electrical engineer needs to estimate the winding currents of a motor in order to control its position. The aerospace engineer needs to estimate the velocity of a satellite in order to control its orbit. The biomedical engineer needs to estimate blood sugar levels in order to regulate insulin injection rates.
- Second, you might need to estimate system states because they are interesting in their own right. For example, the electrical engineer needs to estimate power system parameters in order to predict failure probabilities. The aerospace engineer needs to estimate satellite position in order to intelligently schedule future satellite activities. The biomedical engineer needs to estimate blood protein levels in order to evaluate the health of a patient.

The standard Kalman filter is an effective tool for estimation, but it is limited to linear systems. Most real-world systems are nonlinear, in which case Kalman filters do not directly apply. In the real world, nonlinear filters are used more often than linear filters, because in the real world, systems are nonlinear. In fact, the very first use of Kalman filters involved nonlinear Kalman filters in NASA's space program in the 1960s. This article will tell you the basic concepts that you need to know to design and implement a nonlinear Kalman filter. I'll also illustrate the use of nonlinear Kalman filters by looking at a motor example.

Review of Kalman filters

I wrote an article about Kalman filters in this magazine a few years ago ("Kalman Filtering," June 2001), but I'll review the idea here for those readers who don't have their back issues handy.

If we want to use a standard Kalman filter to estimate a signal, the process that we're measuring has to be able to be described by linear system equations. A linear system is a process that can be described by the following two equations:

$$\text{State equation}: \; x_{k+1} = Ax_k + Bu_k + w_k$$
$$\text{Output Equation}: \; y_k = Cx_k + v_k$$

[1] L. Schwartz and E. Stear, "A computational comparison of several nonlinear filters," *IEEE Transactions on Automatic Control*, vol. 13, no. 1, pp. 83-86 (February 1968).

These equations define a linear system because there are not any exponential functions, trigonometric functions, or any other functions that wouldn't be a straight line when plotted on a graph. The above equations have several variables:

- A, B, and C are matrices
- k is the time index
- x is called the state of the system
- u is a known input to the system (called the control signal)
- y is the measured output
- w and v are the noise – w is called the process noise, and v is called the measurement noise.

Each of these variables are (in general) vectors and therefore contain more than one element.

In state estimation problems, we want to estimate x because it contains all of the information about the system. The problem is that we cannot measure x directly. Instead we measure y, which is a function of x that is corrupted by the noise v. We can use y to help us obtain an estimate of x, but we cannot necessarily take the information from y at its face value because it is corrupted by noise.

As an example, suppose that our system is a tank, a mobile robot, a pizza-delivery car, or some other vehicle moving in a straight line. Then we can say that the state consists of the vehicle position and velocity. The input u is the acceleration, and the output y is the measured position. Let's suppose that we are able to measure the position every T seconds. A previous article in this magazine ("Kalman Filtering," June 2001) showed that this system can be described like this:

$$x_{k+1} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} T^2/2 \\ T \end{bmatrix} u_k + w_k$$

$$y_k = \begin{bmatrix} 1 & 0 \end{bmatrix} x_k + v_k$$

x_k is a vector that contains vehicle position and velocity at time k, u_k is a scalar[2] that is equal to the acceleration, and y_k is a scalar that is equal to the measured position. w_k is a vector that has process noise due to potholes, uncertainties in our knowledge of u_k, and other unmodeled effects. v_k is a scalar that is equal to the measurement noise (that is, instrumentation error).

Now suppose that we want to control the vehicle position to follow a specific path, or we want to estimate the vehicle position for some other reason. We could just use y_k as our position estimate, but y_k is noisy. We could do better by using a Kalman filter. This is because a Kalman filter not only uses the position measurement y_k, but also uses the information that is contained in the state equation. The Kalman filter equations can be written like this:[3]

$$K_k = P_k C^T (C P_k C^T + R)^{-1}$$

$$\hat{x}_{k+1} = (A\hat{x}_k + Bu_k) + K_k (y_k - C\hat{x}_k)$$

$$P_{k+1} = A(I - K_k C) P_k A^T + Q$$

---

[2] A scalar is a regular number – not a vector. Some examples of scalars are 2, −194, and π.

[3] There are many mathematically equivalent ways of writing the Kalman filter equations. This can be a source of confusion to novice and veteran alike.

where the time step $k = 0, 1, 2, \ldots$ This is called a *linear* filter because the $\hat{x}$ equation does not contain any exponential functions, trigonometric functions, or any other functions that would not appear as a straight line on a graph. Here are the meanings of the variables in the Kalman filter equations:

- $\hat{x}_k$ is the estimate of x_k
- K_k is called the Kalman gain (it is a matrix)
- P_k is called the estimation-error covariance (also a matrix)
- Q is the covariance of the process noise w_k, and R is the covariance of the measurement noise v_k (two more matrices).
- The -1 superscript indicates matrix inversion
- The T superscript indicates matrix transposition
- I is the identity matrix.

In order to initialize the Kalman filter, we need to start with an estimate $\hat{x}_0$ of the state at the initial time. We also need to start with an initial estimation error covariance P_0 , which represents our uncertainty in our initial state estimate. If we're very confident in our initial estimate $\hat{x}_0$, then P_0 should be small. If we're very uncertain of our initial estimate $\hat{x}_0$, then P_0 should be large. In the long run, these initialization values will not make much difference in the filter performance.

Linearity limitations

The Kalman filter is a linear filter that can be applied to a linear system. Unfortunately, linear systems do not exist – all systems are ultimately nonlinear. Even the simple $I = V/R$ relationship of Ohm's Law is only an approximation over a limited range of operation. If the voltage across a resistor exceeds a certain value, then Ohm's Law breaks down. Figure 1 shows a typical relationship between the current through a resistor and the voltage across the resistor. At small input voltages the relationship is a straight line, but if the power dissipated by the resistor exceeds some value, then the relationship becomes very nonlinear. Even a device as simple as a resistor is only approximately linear, and even then only in a limited range of operation.

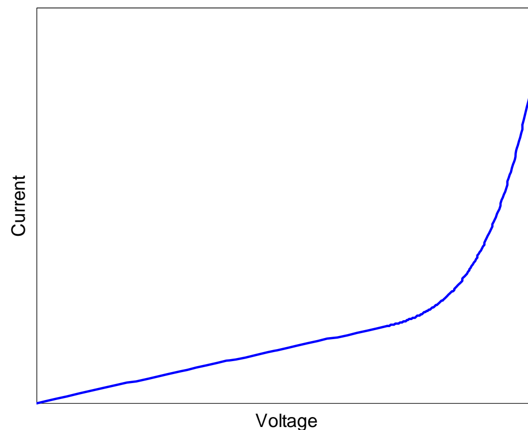

Figure 1 – This shows a typical current/voltage relationship for a resistor. The relationship is linear for a limited range of operation, but becomes highly nonlinear beyond that range. This illustrates the fact that linear systems do not exist in the real world.

So we see that linear systems do not really exist. However, many systems are close enough to linear, so that linear estimation approaches (e.g., standard Kalman filters) give good results. But "close enough" can only be carried so far. Eventually we run across a system that does not behave linearly even over a small range of operation, and the standard Kalman filter no longer gives good results. In this case, we need to explore nonlinear filters.

Nonlinear filtering can be difficult and complex. It is certainly not as well-understood as linear filtering. However, some nonlinear estimation methods have become (or are becoming) widespread. These methods include nonlinear extensions of the Kalman filter, unscented filtering, and particle filtering.

In this article I will talk about the two most basic nonlinear extensions of the Kalman filter. The standard Kalman filter summarized earlier in this article does not directly apply to nonlinear systems. However, if we *linearize* a nonlinear system, then we can use linear estimation methods (such as the Kalman filter) to estimate the states. In order to linearize a nonlinear system, we'll use a mathematical tool called Taylor series expansion, which I discuss next.

Taylor series expansion

The key to nonlinear Kalman filtering is to expand the nonlinear terms of the system equation in a Taylor series expansion[4] around a nominal point $\bar{x}$. A Taylor series expansion of a nonlinear function can be written as

[4] The Taylor series is named after Brook Taylor, an English mathematician who lived from 1685–1731. It wasn't until about 50 years after his death that others saw the importance of the series expansion that he proposed. Other mathematicians, including Isaac Newton, discovered variations of the Taylor series earlier and independently of Taylor. I wonder what Taylor would have thought if he had known that his series would some day be used for spacecraft navigation.

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(\bar{x})\Delta x^n}{n!}$$
$$= f(\bar{x}) + f'(\bar{x})\Delta x + f''(\bar{x})\Delta x^2 / 2 + \cdots$$

In the above equation,

- $\Delta x = x - \bar{x}$
- $f^{(n)}(\bar{x})$ is the nth derivative of $f(x)$, evaluated at $x = \bar{x}$

The above equation looks complicated, but it is really pretty simple. Let's look at an example. Suppose we want to expand $f(x) = \cos(x)$ in a Taylor series around the point $\bar{x} = 0$. Remember that the derivative of $\cos(x)$ is $-\sin(x)$, and the derivative of $\sin(x)$ is $\cos(x)$. That means we can write the Taylor series expansion of $\cos(x)$ as

$$\cos(x) = \cos(\bar{x}) - \sin(\bar{x})\Delta x - \cos(\bar{x})\Delta x^2 / 2 + \cdots$$

Since we're expanding $\cos(x)$ around the nominal point $x = 0$, we see that $\bar{x} = 0$, and $\Delta x = x - \bar{x} = x$. The Taylor series expansion of $\cos(x)$ becomes equal to

$$\cos(x) = \cos(0) - \sin(0)\, x - \cos(0)\, x^2 / 2 + \ldots$$
$$= 1 - x^2 / 2 + \ldots$$

If we use a "second-order" Taylor series expansion of $\cos(x)$, then we can say that $\cos(x)$ is approximately equal to $1 - x^2 / 2$. (It's called "second-order" because it includes terms up to an including the second power of x.) In other words, we can ignore the rest of the terms in the Taylor series. This is because additional terms in the Taylor series involve higher powers of x that are divided by ever-increasing factorials. If x is small, then as we raise x to higher and higher powers, and divide by larger and larger factorials, the additional terms in the Taylor series become insignificant compared to the lower order terms.

Try out the second-order Taylor series expansion of $\cos(x)$ for yourself. Table 1 shows $\cos(x)$ and its second-order Taylor series expansion for various values of x. We see that as x gets smaller (that is, as it gets closer to the nominal point $\bar{x} = 0$), the Taylor series expansion gives a better approximation to the true value of $\cos(x)$.

x	$\cos(x)$	$1 - x^2 / 2$
0	1	1
0.25	0.969	0.969
0.5	0.878	0.875
0.75	0.732	0.719
1.00	0.540	0.500
1.25	0.315	0.219
1.50	0.071	–0.125

Table 1 – The second-order Taylor series expansion of $\cos(x)$ is accurate for small values of x, but it quickly becomes inaccurate as x gets large (i.e., as x gets farther away from the Taylor series expansion point).

Linearizing a function means expanding it in a "first-order" Taylor series around some expansion point. In other words, the first-order Taylor series expansion of a function $f(x)$ is equal to

$$f(x) = f(\bar{x}) + f'(\bar{x})\Delta x$$

Figure 2 shows the function sin(x) along with its first-order Taylor series expansion[5] around the nominal point $\bar{x} = 0$. Note that for small values of x, the two lines in the figure are quite close, which shows that the Taylor series expansion is a good approximation to sin(x). But as x gets larger, the two lines diverge, which means that for large values of x, the Taylor series expansion is a poor approximation.

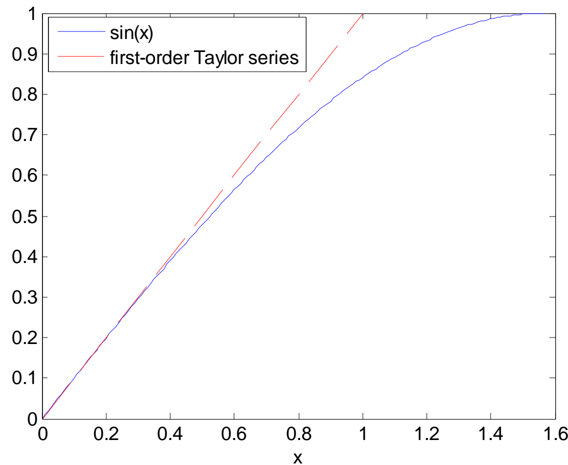

Figure 2 – The first-order Taylor series expansion of sin(x) is accurate for small values of x, but becomes poor for larger values of x.

The linearized Kalman filter

Now that we understand Taylor series, we can derive the linearized Kalman filter. The basic idea of the linearized Kalman filter is to start with a nonlinear system, and then find a linear system whose states represent the *deviations* from a nominal trajectory of the nonlinear system. We can then use the Kalman filter to estimate the deviations from the nominal trajectory. This indirectly gives us an estimate of the states of the nonlinear system.

Here is a general nonlinear system model:

$$\text{State equation: } x_{k+1} = f(x_k, u_k) + w_k$$

$$\text{Output Equation: } y_k = h(x_k) + v_k$$

The state equation $f(\cdot)$ and the measurement equation $h(\cdot)$ are nonlinear functions. As an example, suppose we have a system that looks like this:

$$\text{State equation: } x_{k+1} = x_k^2 + \cos x_k + u_k + w_k$$

$$\text{Output Equation: } y_k = 1/x_k + v_k$$

The state equation has two nonlinear terms (a square term and a cosine term). The output equation is also nonlinear because it contains $1/x_k$. If either the state equation or the output equation has nonlinear terms, then the system is called a nonlinear system.

[5] The first-order Taylor series expansion of sin(x) around the point $x = 0$ can be derived as sin(x) $\approx x$. If you can derive this, then you are well on your way to understanding Taylor series.

In the linearized Kalman filter, we use first-order Taylor series to expand the state equation and output equation around a nominal state. The nominal state is a function of time, so it is sometimes referred to as a *trajectory*. The nominal trajectory is based on a guess of what the system behavior might look like. For example, if the system equations represent the dynamics of an airplane, then the nominal state might be the planned flight trajectory. The *actual* flight trajectory will differ from this nominal trajectory due to mismodeling, disturbances, and other unforeseen effects. But hopefully the actual trajectory will be *close* to the nominal trajectory, in which case the Taylor series linearization should be reasonably accurate. The Taylor series linearizations of the nonlinear system equation and output equation give

$$\begin{aligned} x_{k+1} &= f(x_k, u_k) + w_k \\ &\approx f(\bar{x}_k, u_k) + f'(\bar{x}_k, u_k)\Delta x_k + w_k \\ y_k &= h(x_k) + v_k \\ &\approx h(\bar{x}_k) + h'(\bar{x}_k)\Delta x_k + v_k \end{aligned}$$

Now notice that the deviations from the nominal trajectory can be written as

$$\begin{aligned} \Delta x_{k+1} &= x_{k+1} - \bar{x}_{k+1} \\ &= x_{k+1} - f(\bar{x}_k, u_k) \\ \Delta y_k &= y_k - \bar{y}_k \\ &= y_k - h(\bar{x}_k) \end{aligned}$$

Combining these equations with the previous equations gives

$$\begin{aligned} \Delta x_{k+1} &= f'(\bar{x}_k, u_k)\Delta x_k + w_k \\ \Delta y_k &= h'(\bar{x}_k)\Delta x_k + v_k \end{aligned}$$

Now look at what we've accomplished – we have state and output equations that are linear functions of Δx and Δy. That means that we can use a standard Kalman filter to estimate Δx.

There are two important points to remember when using the linearized Kalman filter:

- After the Kalman filter is used to estimate Δx, we need to add the estimate of Δx to the nominal state $\bar{x}$ in order to get an estimate of the state x. This is because $\Delta x = x - \bar{x}$.
- If the true state x gets too far away from the nominal state $\bar{x}$, then the linearized Kalman filter will not give good results. This is because the Taylor series approximation breaks down if x gets too far away from $\bar{x}$ (see Figure 2).

Here's a summary of the linearized Kalman filter algorithm:

1. The system equations are given as

   State equation : $x_{k+1} = f(x_k, u_k) + w_k$

   Output Equation : $y_k = h(x_k) + v_k$

2. The nominal trajectory is known ahead of time:

   $$\begin{aligned} \bar{x}_{k+1} &= f(\bar{x}_k, u_k) \\ \bar{y}_k &= h(\bar{x}_k) \end{aligned}$$

3. At each time step, compute the following partial derivative matrices, evaluated at the nominal state:

$A_k = f'(\bar{x}_k, u_k)$

$C_k = h'(\bar{x}_k)$

The derivatives in the above equation are taken with respect to x_k.

4. Define Δy_k as the difference between the actual measurement y_k and the nominal measurement $\bar{y}_k$:

   $$\begin{aligned}\Delta y_k &= y_k - \bar{y}_k \\ &= y_k - h(\bar{x}_k)\end{aligned}$$

5. Execute the following Kalman filter equations:

   $$K_k = P_k C_k^T (C_k P_k C_k^T + R)^{-1}$$
   $$\Delta\hat{x}_{k+1} = A_k \Delta\hat{x}_k + K_k (\Delta y_k - C_k \Delta\hat{x}_k)$$
   $$P_{k+1} = A_k (I - K_k C_k) P_k A_k^T + Q$$
   $$\hat{x}_{k+1} = \bar{x}_{k+1} + \Delta\hat{x}_{k+1}$$

The extended Kalman filter

The linerized Kalman filter that we derived above is fine as far as it goes, but there is a problem with it: we need to know the nominal trajectory $\bar{x}$ ahead of time. For some systems we might know a nominal trajectory ahead of time (for example, an aircraft flight with a predetermined flight plan, or a robot arm moving in a manufacturing environment). But for other systems we may have no way of knowing a nominal trajectory.

The idea of the EKF (extended Kalman filter) is to use our estimate of x as the nominal trajectory in the linearized Kalman filter. In other words, we set $\bar{x}$ equal to $\hat{x}$ in the linearized Kalman filter. This is a clever bootstrapping approach to state estimation; we use a nominal trajectory to estimate x, and then we use the estimated value of x as the nominal trajectory. After making these substitutions into the linearized Kalman filter equations and going through some mathematical manipulations, we get the following EKF algorithm:

1. The system equations are given as

   State equation : $x_{k+1} = f(x_k, u_k) + w_k$

   Output Equation : $y_k = h(x_k) + v_k$

2. At each time step, compute the following derivative matrices, evaluated at the current state estimate:

   $A_k = f'(\hat{x}_k, u_k)$

   $C_k = h'(\hat{x}_k)$

   Note that the derivatives are taken with respect to x_k, and then evaluated at $x_k = \hat{x}_k$.

3. Execute the following Kalman filter equations:

$$K_k = P_k C_k^T (C_k P_k C_k^T + R)^{-1}$$
$$\hat{x}_{k+1} = f(\hat{x}_k, u_k) + K_k [y_k - h(\hat{x}_k)]$$
$$P_{k+1} = A_k (I - K_k C_k) P_k A_k^T + Q$$

Motor state estimation

To illustrate the use of the EKF, let's use it to estimate the states of a two-phase permanent magnet synchronous motor. We might want to estimate the states so that we can regulate them with a control algorithm, or we might want to estimate the states because we want to know the position or velocity of the motor for some other reason. Let's suppose that we can measure the motor winding currents, and we want to use the EKF to estimate the rotor position and velocity. The system equations are

$$\dot{I}_a = \frac{-R}{L} I_a + \frac{\omega\lambda}{L}\sin\theta + \frac{u_a + \Delta u_a}{L}$$
$$\dot{I}_b = \frac{-R}{L} I_b - \frac{\omega\lambda}{L}\cos\theta + \frac{u_b + \Delta u_b}{L}$$
$$\dot{\omega} = \frac{-3\lambda}{2J} I_a \sin\theta + \frac{3\lambda}{2J} I_b \cos\theta - \frac{F\omega}{J} + \Delta\alpha$$
$$\dot{\theta} = \omega$$
$$y = \begin{bmatrix} I_a \\ I_b \end{bmatrix} + \begin{bmatrix} v_a \\ v_b \end{bmatrix}$$

The variables in these equations are defined as follows:

- I_a and I_b are the currents in the two motor windings.
- θ and ω are the angular position and velocity of the rotor.
- R and L are the motor winding's resistance and inductance.
- λ is the flux constant of the motor.
- F is the coefficient of viscous friction that acts on the motor shaft and its load.
- J is the moment of inertia of the motor shaft and its load.
- u_a and u_b are the voltages that are applied across the two motor windings.
- Δu_a and Δu_b are noise terms due to errors in u_a and u_b.
- $\Delta\alpha$ is a noise term due to uncertainty in the load torque.
- y is the measurement. We're assuming here that we have measurements of the two winding currents, maybe using sense resistors. The measurements are distorted by measurement noises v_a and v_b, which are due to things like sense resistance uncertainty, electrical noise, and quantization errors in our microcontroller.

If we want to apply an EKF to the motor, we need to define the states of the system. The states can be seen by looking at the system equations and noting wherever a derivative appears. If a variable is differentiated in the system equations, then that quantity is a state. So we see from the above motor equations that our system has four states, and the state vector x can be defined as

$$x = \begin{bmatrix} I_a \\ I_b \\ \omega \\ \theta \end{bmatrix}$$

The system equation is obtained by discretizing the differential equations to obtain

$$\begin{aligned} x_{k+1} &= f(x_k, u_k) + w_k \\ &= x_k + \begin{bmatrix} -Rx_k(1)/L + x_k(3)\lambda \sin x_k(4)/L + u_{ak}/L \\ -Rx_k(2)/L + x_k(3)\lambda \cos x_k(4)/L + u_{bk}/L \\ -3\lambda x_k(1)\sin x_k(4)/2J + 3\lambda x_k(2)\cos x_k(4)/2J - Fx_k(3)/J \\ x_k(3) \end{bmatrix} \Delta t + \begin{bmatrix} \Delta u_{ak}/L \\ \Delta u_{bk}/L \\ \Delta\alpha \\ 0 \end{bmatrix} \Delta t \end{aligned}$$

$$\begin{aligned} y_k &= h(x_k) + v_k \\ &= \begin{bmatrix} x_k(1) \\ x_k(2) \end{bmatrix} + \begin{bmatrix} v_{ak} \\ v_{bk} \end{bmatrix} \end{aligned}$$

where Δt is the step size that we're using for estimation in our microcontroller or DSP.

In order to use an EKF, we need to find the derivatives of $f(x_k, u_k)$ and $h(x_k)$ with respect to x_k. This is a new twist, because $f(x_k, u_k)$ and $h(x_k)$ are both vector functions of the vector x_k. How can we find the derivative of a vector with respect to another vector? If you know how to take derivatives, then it is really not too difficult to find the derivative of a vector. For example, if we write the vectors f and x as

$$f = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} \qquad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

then the derivative of f with respect to x is equal to the 4×4 matrix

$$f'(x) = \begin{bmatrix} df_1/dx_1 & df_1/dx_2 & df_1/dx_3 & df_1/dx_4 \\ df_2/dx_1 & df_2/dx_2 & df_2/dx_3 & df_2/dx_4 \\ df_3/dx_1 & df_3/dx_2 & df_3/dx_3 & df_3/dx_4 \\ df_4/dx_1 & df_4/dx_2 & df_4/dx_3 & df_4/dx_4 \end{bmatrix}$$

This can be generalized for any size vectors f and x. With this background, we can now find the derivative matrices as

$$A_k = f'(\hat{x}_k, u_k)$$

$$= I + \Delta t \begin{bmatrix} -R/L & 0 & \lambda \sin \hat{x}_k(4)/L & \hat{x}_k(3)\lambda \cos \hat{x}_k(4)/L \\ 0 & -R/L & -\lambda \cos \hat{x}_k(4)/L & \hat{x}_k(3)\lambda \sin \hat{x}_k(4)/L \\ -3\lambda \sin \hat{x}_k(4)/2J & 3\lambda \cos \hat{x}_k(4)/2J & -F/J & -3\lambda[\hat{x}_k(1)\cos \hat{x}_k(4) + \hat{x}_k(2)\sin \hat{x}_k(4)]/2J \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$C_k = h'(\hat{x}_k)$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Let’s simulate the EKF to see how well we can estimate rotor position and velocity. We’ll assume that the measurement noise terms, v_{ak} and v_{bk}, are zero-mean random variables with standard deviations equal to 0.1 amps. The control inputs (winding voltages) are equal to

$$u_a(t) = \sin 2\pi t$$

$$u_b(t) = \cos 2\pi t$$

This means that in discrete time, the control inputs are equal to

$$u_{ak} = \sin 2\pi k \Delta t$$

$$u_{bk} = \cos 2\pi k \Delta t$$

The voltages that are applied to the winding currents are equal to these values plus Δu_{ak} and Δu_{bk}, which are zero-mean random variables with standard deviations equal to 0.001 amps. The noise due to load torque disturbances $\Delta \alpha_k$ has a standard deviation of 0.05 rad/sec^2. Even though our measurements consist only of the winding currents, we can use an EKF to estimate the rotor position and velocity. I used Matlab to simulate the motor system and the EKF. The code listing is shown at the end of this article, and the simulation results are shown in Figure 3. We see that the rotor position and velocity are estimated quite well.

Practically speaking, this means that we can figure out the rotor position and velocity without using an encoder. Instead of an encoder to get rotor position, we just need a couple of sense resistors, and a Kalman filter in our microcontroller. This is a good example of a trade-off between instrumentation and mathematics. If you know how to apply the mathematics of a Kalman filter in a microcontroller, then you can get rid of your encoder and maybe save a lot of money in your embedded systems product.

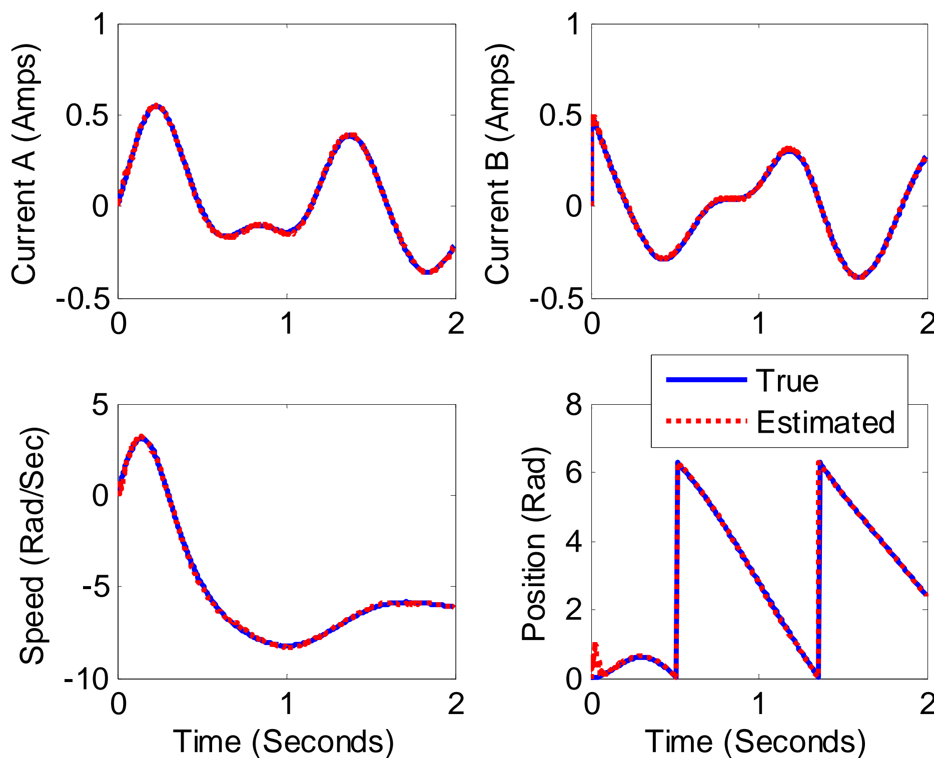

Figure 3 – Extended Kalman filter simulation results for a two-phase permanent magnet synchronous motor. Winding current measurements are obtained once per millisecond.

It's important to know about the limitations of the EKF. For example, Figure 3 was generated using a 1 ms sample time for measurements. If the sample time is increased to 2 ms, then the EKF estimate "blows up"; that is, the EKF becomes unstable. If the measurement noise increases by a factor of 10, then the EKF gives position estimates that are essentially worthless. These are not really limitations of the EKF, but are rather limitations of how much information we can squeeze out of a certain system.

Conclusion

We've seen how the Kalman filter can be modified for state estimation in nonlinear systems. The resulting filter is called the EKF (extended Kalman filter). I think it's interesting that the first applications of the Kalman filter were not for linear systems, but were for nonlinear systems; that is, spacecraft navigation problems in the 1960s. Stanley Schmidt was the driving force behind the use of Kalman filters in NASA's space program. Schmidt was the branch chief of the NASA Ames Dynamic Analysis Branch in the late 1950s and early 1960s when NASA was starting feasibility studies of lunar missions. Kalman and Schmidt happened to be acquaintances during the time that Kalman developed his theory, and Schmidt needed a navigation algorithm. Schmidt played such an important role in nonlinear Kalman filtering that during the early 1960s, the EKF was often referred to as the Kalman–Schmidt filter.

We talked about the "discrete-time EKF" in this article. This is the type of EKF that we use if the system description is discrete-time. If the system equations are continuous-time and the measurements are also continuous-time, then we use the "continuous-time EKF." If the system equations are continuous-time and the measurements are discrete-time, then we use the "hybrid EKF." These topics are discussed in textbooks on Kalman filtering.

The key to being able to use an EKF is to be able to represent the system with a mathematical model. That is, the EKF designer needs to understand the system well enough to be able to describe its behavior with differential equations. In practice, this is often the most difficult part of implementing a Kalman filter.[6] Another challenge in Kalman filtering is to be able to accurately model the noise in the system. In our motor example, we assumed that the measurement noises were zero-mean with a standard deviation of 0.1 amps. The only way to figure this out is if you know the tolerance of your sense resistors, and if you have a good understanding of the electrical noise and discretization errors that are corrupting your winding-current measurements. Again, it all boils down to having a good understanding of the system whose states you're trying to estimate.

When we derived the EKF in this article, we used a first-order Taylor series to approximate the nonlinear system equations. What if we use a second-order Taylor series to approximate the equations? Then we would have a more accurate approximation to our nonlinear equations. This is an example of what is called a "higher-order" approach to nonlinear Kalman filtering. Higher-order approaches may give better results if the system nonlinearities are especially severe. These higher-order approaches include second-order Kalman filtering, iterated Kalman filtering, sum-based Kalman filtering, and grid-based Kalman filtering. These filters provide ways to reduce the linearization errors that are part of the EKF. They usually give estimation performance that is better than the EKF, but they're a lot more complicated, and they required a lot more computer time. Other popular nonlinear state estimators include unscented Kalman filters and particle filters.

Dan Simon worked for 14 years in industry before becoming a professor in the Electrical and Computer Engineering Department at Cleveland State University in Cleveland, Ohio. He is also the owner of Innovatia Software, an independent consulting firm. His teaching and research interests include control and estimation, embedded systems, and artificial intelligence. He's applied optimal filtering algorithms to missile navigation, GPS-based vehicle tracking, neural network training, fuzzy system training, aircraft engine fault detection, and motor position estimation. He is presently trying to use Kalman filtering to optimize ethical decision-making processes. You can contact him at d.j.simon@csuohio.edu.

[6] This is a general principle in engineering. The hardest part of solving a problem is putting it in a form that can be attacked with the tools that are at your disposal. Once the problem is in a familiar form, the application of proven tools is straightforward.

For Further Reading

Simon, D. *Optimal State Estimation: Kalman, H∞, and Nonlinear Approaches*. New York: John Wiley & Sons, 2006.
I wrote down everything I learned in the past 20 years about state estimation and put it in this book. The web site for the book is at http://academic.csuohio.edu/simond/estimation, and includes Matlab source code for all of the examples in the book.

Code Listing – Matlab-based Kalman filter simulation code

```
function Motor1

% Discrete-time extended Kalman filter simulation for two-phase
% step motor. Estimate the stator currents, and the rotor position
% and velocity, on the basis of noisy measurements of the stator
% currents.

Ra = 2; % Winding resistance
L = 0.003; % Winding inductance
lambda = 0.1; % Motor constant
J = 0.002; % Moment of inertia
B = 0.001; % Coefficient of viscous friction

ControlNoise = 0.001; % std dev of uncertainty in control inputs (amps)
MeasNoise = 0.1; % std dev of measurement noise (amps)

R = [MeasNoise^2 0; 0 MeasNoise^2]; % Measurement noise covariance
xdotNoise = [ControlNoise/L; ControlNoise/L; 0.5; 0];
% Define the continuous-time process noise covariance Q
Q = [xdotNoise(1)^2 0 0 0;
    0 xdotNoise(2)^2 0 0;
    0 0 xdotNoise(3)^2 0;
    0 0 0 xdotNoise(4)^2];
P = 1*eye(4); % Initial state estimation covariance

dt = 0.0002; % Simulation step size (seconds)
T = 0.001; % how often measurements are obtained
tf = 2; % Simulation length

x = [0; 0; 0; 0]; % Initial state
xhat = x; % Initial state estimate
w = 2 * pi; % Control input frequency (rad/sec)
Q = Q * T; % discrete-time process noise covariance

% Initialize arrays for plotting at the end of the program
NumTimeSteps = round(tf / T); % number of time steps
xArray = zeros(4, NumTimeSteps); % true state
xhatArray = zeros(4, NumTimeSteps); % estimated state
trPArray = zeros(1, NumTimeSteps); % trace of estimation error covariance
tArray = zeros(1, NumTimeSteps); % time array
i = 0; % loop index

% Begin simulation loop
for t = 0 : T : tf-T+eps
    y = [x(1); x(2)] + MeasNoise * randn(2,1); % noisy measurement

    % Save data for plotting
```

```
    i = i + 1;
    xArray(:, i) = x;
    xhatArray(:, i) = xhat;
    trPArray(i) = trace(P);
    tArray(i) = t;

    % System simulation
    for tau = 0 : dt : T-dt+eps
        time = t + tau;
        ua = sin(w*time);
        ub = cos(w*time);
        xdot = [-Ra/L*x(1) + x(3)*lambda/L*sin(x(4)) + ua/L;
            -Ra/L*x(2) - x(3)*lambda/L*cos(x(4)) + ub/L;
            -3/2*lambda/J*x(1)*sin(x(4)) + ...
            3/2*lambda/J*x(2)*cos(x(4)) - B/J*x(3);
            x(3)];
        xdot = xdot + xdotNoise .* randn(4,1);
        x = x + xdot * dt; % rectangular integration
        x(4) = mod(x(4), 2*pi); % keep the angle between 0 and 2*pi
    end

    % Compute the partial derivative matrices
    A = eye(4) + T * [-Ra/L, 0, lambda/L*sin(xhat(4)), xhat(3)*lambda/L*cos(xhat(4));
        0, -Ra/L, -lambda/L*cos(xhat(4)), xhat(3)*lambda/L*sin(xhat(4));
        -3/2*lambda/J*sin(xhat(4)), 3/2*lambda/J*cos(xhat(4)), -B/J, ...
        -3/2*lambda/J*(xhat(1)*cos(xhat(4))+xhat(2)*sin(xhat(4)));
        0 0 1 0];
    C = [1 0 0 0; 0 1 0 0];
    % Compute the Kalman gain
    K = P * C' / (C * P * C' + R);
    % Update the state estimate
    ua = sin(w*t);
    ub = cos(w*t);
    deltax = [-Ra/L*xhat(1) + xhat(3)*lambda/L*sin(xhat(4)) + ua/L;
            -Ra/L*xhat(2) - xhat(3)*lambda/L*cos(xhat(4)) + ub/L;
            -3/2*lambda/J*xhat(1)*sin(xhat(4)) + ...
            3/2*lambda/J*xhat(2)*cos(xhat(4)) - B/J*xhat(3);
            xhat(3)] * T;
    xhat = xhat + deltax + K * (y - [xhat(1); xhat(2)]);
    % keep the angle estimate between 0 and 2*pi
    xhat(4) = mod(xhat(4), 2*pi);
    % Update the estimation error covariance.
    P = A * ((eye(4) - K * C) * P) * A' + Q;
end

% Plot the results
close all;
figure; set(gcf,'Color','White');

subplot(2,2,1); hold on; box on;
plot(tArray, xArray(1,:), 'b-', 'LineWidth', 2);
plot(tArray, xhatArray(1,:), 'r:', 'LineWidth', 2)
set(gca,'FontSize',12); ylabel('Current A (Amps)');

subplot(2,2,2); hold on; box on;
plot(tArray, xArray(2,:), 'b-', 'LineWidth', 2);
plot(tArray, xhatArray(2,:), 'r:', 'LineWidth', 2)
set(gca,'FontSize',12); ylabel('Current B (Amps)');

subplot(2,2,3); hold on; box on;
plot(tArray, xArray(3,:), 'b-', 'LineWidth', 2);
plot(tArray, xhatArray(3,:), 'r:', 'LineWidth', 2)
set(gca,'FontSize',12);
```

```
xlabel('Time (Seconds)'); ylabel('Speed (Rad/Sec)');

subplot(2,2,4); hold on; box on;
plot(tArray, xArray(4,:), 'b-', 'LineWidth', 2);
plot(tArray,xhatArray(4,:), 'r:', 'LineWidth', 2)
set(gca,'FontSize',12);
xlabel('Time (Seconds)'); ylabel('Position (Rad)');
legend('True', 'Estimated');

figure;
plot(tArray, trPArray); title('Trace(P)', 'FontSize', 12);
set(gca,'FontSize',12); set(gcf,'Color','White');
xlabel('Seconds');

% Compute the std dev of the estimation errors
N = size(xArray, 2);
N2 = round(N / 2);
xArray = xArray(:,N2:N);
xhatArray = xhatArray(:,N2:N);
iaEstErr = sqrt(norm(xArray(1,:)-xhatArray(1,:))^2 / size(xArray,2));
ibEstErr = sqrt(norm(xArray(2,:)-xhatArray(2,:))^2 / size(xArray,2));
wEstErr = sqrt(norm(xArray(3,:)-xhatArray(3,:))^2 / size(xArray,2));
thetaEstErr = sqrt(norm(xArray(4,:)-xhatArray(4,:))^2 / size(xArray,2));
disp(['Std Dev of Estimation Errors = ',num2str(iaEstErr),', ', ...
    num2str(ibEstErr),', ',num2str(wEstErr),', ',num2str(thetaEstErr)]);

% Display the P version of the estimation error standard deviations
disp(['Sqrt(P) = ',num2str(sqrt(P(1,1))),', ',num2str(sqrt(P(2,2))),', ', ...
    num2str(sqrt(P(3,3))),', ',num2str(sqrt(P(4,4)))]);
```