

TKJ Electronics



Development with ease

- [Home](#)
- [Reviews](#)
- [Arduino](#)
- [ARM](#)
- [PIC](#)
- [FPGA](#)
- [TKJ Electronics](#)
- [Webshop](#)
-

Type text to search here...

[Home](#) > [Guides](#), [TKJ Electronics](#) > A practical approach to Kalman filter and how to implement it

A practical approach to Kalman filter and how to implement it

September 10th, 2012 [Kristian Sloth Lauszus](#) [Leave a comment](#) [Go to comments](#)

I have for a long time been interested in Kalman filers and how they work, I also used a Kalman filter for my [Balancing robot](#), but I never explained how it actually was implemented. Actually I had never taken the time to sit down with a pen and a piece of paper and try to do the math by myself, so I actually did not know how it was implemented.

It turned out to be a good thing, as I actually discovered a mistake in the original code, but I will get back to that later.

I actually wrote about the Kalman filter as my master assignment in high school back in December 2011. But I only used the Kalman filter to calculate the true voltage of a DC signal modulated by known Gaussian white noise. My assignment can be found in the following zip file:

http://www.tkjelectronics.dk/uploads/Kalman_SRP.zip. It is in danish, but you can properly use google translate to translate some of it. If you got any specific questions regarding the assignment, then ask in the comments below.

Okay, but back to the subject. As I sad I had never taken the time to sit down and do the math regarding the Kalman filter based on an accelerometer and a gyroscope. It was not as hard as I expected, but I must confess that I still have not studied the deeper theory behind, on why it actually works. But for me, and most people out there, I am more interested in implementing the filter, than in the deeper theory behind and why the equations works.

Before we begin you must have some basic knowledge about matrices like multiplication of matrices and transposing of matrices. If not then please take a look at the following websites:

- http://en.wikipedia.org/wiki/Matrix_multiplication#Matrix_product_.28two_matrices.29
- <http://www.mathwarehouse.com/algebra/matrix/multiply-matrix.php>
- <http://en.wikipedia.org/wiki/Transpose>
- http://en.wikipedia.org/wiki/Covariance_matrix

For those of you who do not know what a Kalman filter is, it is an algorithm which uses a series of measurements observed over time, in this context an accelerometer and a gyroscope. These measurements will contain noise that will contribute to the error of the measurement. The Kalman filter will then try to estimate the state of the system, based on the current and previous states, that tend to be more precise than the measurements alone.

In this context the problem is that the accelerometer is in general very noise when it is used to measure the gravitational acceleration since the robot is moving back and forth. The problem with the gyro is that it drifts over time - just like a spinning wheel-gyro will start to fall down when it is losing speed.

In short you can say that you can only trust the gyroscope on a short term while you can only trust the accelerometer on a long term.

There is actually a very easy way to deal with this by using a complimentary filter, which basicly just

consist of a digital low-pass filter on the accelerometer and digital high-pass filter on the gyroscope readings. But it is not as accurate as the Kalman filter, but other people have successfully build balancing robots using a fine-tuned complimentary filter.

More information about gyroscopes, accelerometer and complimentary filters can be found in this [pdf](#). A comparison between a complimentary filter and a Kalman filter can be found in the following [blog post](#).

The Kalman filter operates by producing a statistically optimal estimate of the system state based upon the measurement(s). To do this it will need to know the noise of the input to the filter called the measurement noise, but also the noise of the system itself called the process noise. To do this the noise has to be [Gaussian distributed](#) and have a mean of zero, luckily for us most random noise have this characteristic.

For more information about the theory behind the filter take a look at the following pages:

- http://en.wikipedia.org/wiki/Kalman_filter
- http://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf
- <http://academic.csuohio.edu/simond/courses/eec644/kalman.pdf>

The system state \mathbf{x}_k

The next of this article might seem very confusing for some, but I promise you if you grab a pen and a piece of paper and try to follow along it is not that hard if you are reasonable at math.

If you, like me, do not have a calculator or computer program that can work with matrices, then I recommend the free online calculator [Wolfram Alpha](#). I used it for all the calculations in this article.

I will use the same notation as the [wikipedia article](#), but I will like to note that when the matrixes are constants and does not depend on the current time you do not have to write the k after them. So for instance F_k can be simplified to F .

Also I would like to write a small explanation of the other aspects of the notations. First I will make a note about whats called the previous state:

$$\hat{\mathbf{x}}_{k-1|k-1}$$

Which is the previous estimated state based on the previous state and the estimates of the states before it.

The next is the a priori state:

$$\hat{\mathbf{x}}_{k|k-1}$$

A priori means the estimate of the state matrix at the current time k based on the previous state of the system and the estimates of the states before it.

The last one is called a posteriori state:

$$\hat{\mathbf{x}}_{k|k}$$

Is the estimated of the state at time k given observations up to and including at time k.

The problem is that the system state itself is hidden and can only be observed through observation z_k . This is also called a [Hidden Markov model](#).

This means that the state will be based upon the state at time k and all the previous states. That also means that you can not trust the estimate of the state before the Kalman filter has stabilized - take a look at the graph at the front page of [my assignment](#).

The hat over the $\hat{\mathbf{x}}$ means that is the estimate of the state. Unlike just a single \mathbf{x} which means the true state - the one we are trying to estimate. So the notation for the state at time k is:

$$\mathbf{x}_k$$

The state of the system at time k if given by:

$$\mathbf{x}_k = F\mathbf{x}_{k-1} + Bu_k + w_k$$

Where \mathbf{x}_k is the state matrix which is given by:

$$\mathbf{x}_k = \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_k$$

As you can see the output of the filter will be the angle θ but also the bias $\dot{\theta}_b$ based upon the measurements from the accelerometer and gyroscope. The bias is the amount the gyro has drifted. This means that one can get the true rate by subtracting the bias from the gyro measurement.

The next is the F matrix, which is the state transition model which is applied to the previous state \mathbf{x}_{k-1} .

In this case F is defined as:

$$\mathbf{F} = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix}$$

I know that the $-\Delta t$ might seem confusing, but it will make sense later (take a look at my [comment](#)).

The next is the control input u_k , in this case it is the gyroscope measurement in degrees per second ($^\circ$ /s) at time k , this is also called the rate $\dot{\theta}$. We will actually rewrite the state equation as:

$$\mathbf{x}_k = \mathbf{F}\mathbf{x}_{k-1} + \mathbf{B}\dot{\theta}_k + w_k$$

The next thing is the B matrix. Which is called the control-input model, which is defined as:

$$\mathbf{B} = \begin{bmatrix} \Delta t \\ 0 \end{bmatrix}$$

This makes perfectly sense as you will get the angle θ when you multiply the rate $\dot{\theta}$ by the delta time Δt and since we can not calculate the bias directly based on the rate we will set the bottom of the matrix to 0.

w_k is process noise which is Gaussian distributed with a zero mean and with covariance Q to the time k :

$$w_k \sim N(0, Q_k)$$

Q_k is the process noise covariance matrix and in this case the covariance matrix of the state estimate of the accelerometer and bias. In this case we will consider the estimate of the bias and the accelerometer to be independent, so it's actually just equal to the variance of the estimate of the accelerometer and bias.

The final matrix is defined as so:

$$Q_k = \begin{bmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{bmatrix} \Delta t$$

As you can see the Q_k covariance matrix depends on the current time k , so the accelerometer variance Q_θ and the variance of the bias $Q_{\dot{\theta}_b}$ is multiplied by the delta time Δt .

This makes sense as the process noise will be larger as longer time it is since the last update of the state. For instance the gyro could have drifted.

You will have to know these constants for the Kalman filter to work.

Note if you set a larger value, the more noise in the estimation of the state. So for instance if the estimated angle starts to drift you have to increase the value of $Q_{\dot{\theta}_b}$. Otherwise if the estimate tends to be slow you are trusting the estimate of the angle too much and should try to decrease the value of Q_θ to make it more responsive.

The measurement z_k

Now we will take a look at the observation or measurement z_k of the true state \mathbf{x}_k . The observation z_k is given by:

$$z_k = \mathbf{H}\mathbf{x}_k + v_k$$

As you can see the measurement z_k is given by the current state \mathbf{x}_k multiplied by the H matrix plus the measurement noise v_k .

H is called the observation model and is used to map the true state space into the observed space. The true state can not be observed. Since the measurement is just the measurement from the accelerometer, H is given by:

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

The noise of the measurement have to be Gaussian distributed as well with a zero mean and R as the covariance:

$$\mathbf{v}_k \sim N(0, \mathbf{R})$$

But as R is not a matrix the measurement noise is just equal to the variance of the measurement, since the covariance of the same variable is equal to the variance. See this [page](#) for more information. Now we can define R as so:

$$\mathbf{R} = E \begin{bmatrix} v_k & v_k^T \end{bmatrix} = \text{var}(v_k)$$

More information about covariance can be found on [Wikipedia](#) and in [my assignment](#).

We will assume that the measurement noise is the same and does not depend on the time k :

$$\text{var}(v_k) = \text{var}(v)$$

Note that if you set the measurement noise variance $\text{var}(v)$ too high the filter will respond really slowly as it is trusting new measurements less, but if it is too small the value might overshoot and be noisy since we trust the accelerometer measurements too much.

So to round up you have to find the the process noise variances Q_θ and $Q_{\dot{\theta}}$ and the measurement variance of the measurement noise $\text{var}(v)$. There are multiple ways to find them, but it is out of the aspect of this article.

The Kalman filter equations

Okay now to the equations we will use to estimate the true state of the system at time k \hat{x}_k . Some clever guys came up with equations found below to estimate the state of the system.

The equations can be written more compact, but I prefer to have them stretched out, so it is easier to implement and understand the different steps.

Predict

In the first two equations we will try to predict the current state and the error covariance matrix at time k . First the filter will try to estimate the current state based on all the previous states and the gyro measurement:

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}\hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}\dot{\theta}_k$$

That is also why it is called a control input, since we use it as an extra input to estimate the state at the current time k called the a priori state $\hat{\mathbf{x}}_{k|k-1}$ as described in the beginning of the article.

The next thing is that we will try to estimate the a priori error covariance matrix $\mathbf{P}_{k|k-1}$ based on the previous error covariance matrix $\mathbf{P}_{k-1|k-1}$, which is defined as:

$$\mathbf{P}_{k|k-1} = \mathbf{F}\mathbf{P}_{k-1|k-1}\mathbf{F}^T + \mathbf{Q}_k$$

This matrix is used to estimate how much we trust the current values of the estimated state. The smaller the more we trust the current estimated state. The principle of the equation above is actually pretty easy to understand, as it is pretty obvious that the error covariance will increase since we last updated the estimate of the state, therefore we multiplied the error covariance matrix by the state transition model \mathbf{F} and the transpose of that \mathbf{F}^T and add the current process noise \mathbf{Q}_k at time k .

The error covariance matrix \mathbf{P} in our case is a 2×2 matrix:

$$\mathbf{P} = \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}$$

Update

The first thing we will do is to compute the difference between the measurement z_k and the a priori state $\hat{\mathbf{x}}_{k|k-1}$, this is also called the innovation:

$$\tilde{\mathbf{y}}_k = z_k - \mathbf{H}\hat{\mathbf{x}}_{k|k-1}$$

The observation model \mathbf{H} is used to map the a priori state $\hat{\mathbf{x}}_{k|k-1}$ into the observed space which is the measurement from the accelerometer, therefore the innovation is not a matrix

$$\tilde{\mathbf{y}}_k = [\tilde{y}]_k$$

The next thing we will do is calculate what's called the innovation covariance:

$$\mathbf{S}_k = \mathbf{H}\mathbf{P}_{k|k-1}\mathbf{H}^T + \mathbf{R}$$

What it does is that it tries to predict how much we should trust the measurement based on the a priori error covariance matrix $\mathbf{P}_{k|k-1}$ and the measurement covariance matrix \mathbf{R} . The observation model \mathbf{H} is used to map the a priori error covariance matrix $\mathbf{P}_{k|k-1}$ into observed space.

The bigger the value of the measurement noise the larger the value of \mathbf{S} , this means that we do not trust the incoming measurement that much.

In this case \mathbf{S} is not a matrix and is just written as:

$$\mathbf{S}_k = [\mathbf{S}]_k$$

The next step is to calculate the Kalman gain. The Kalman gain is used to indicate how much we trust the innovation and is defined as:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}^T\mathbf{S}_k^{-1}$$

You can see that if we do not trust the innovation that much the innovation covariance \mathbf{S} will be high and if we trust the estimate of the state then the error covariance matrix \mathbf{P} will be small the Kalman gain will therefore be small and opposite if we trust the innovation but does not trust the estimation of the current state.

If you take a deeper look you can see that the transpose of the observation model \mathbf{H} is used to map the state of the error covariance matrix \mathbf{P} into observed space. We then compare the error covariance matrix by multiplying with the inverse of the innovation covariance \mathbf{S} .

This makes sense as we will use the observation model \mathbf{H} to extract data from the state error covariance and compare that with the current estimate of the innovation covariance.

Note that if you do not know the state at startup you can set the error covariance matrix like so:

$$\mathbf{P} = \begin{bmatrix} L & 0 \\ 0 & L \end{bmatrix}$$

Where L represent a large number.

For my balancing robot I know the starting angle and I find the bias of the gyro at startup by calibrating, so I assume that the state will be known at startup, so I initialize the error covariance matrix like so:

$$\mathbf{P} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Take a look at [my calibration routine](#) for more information.

In this case the Kalman gain is a 2×1 matrix:

$$\mathbf{K} = \begin{bmatrix} K_0 \\ K_1 \end{bmatrix}$$

Now we can update the a posteriori estimate of the current state:

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k$$

This is done by adding the a priori state $\hat{\mathbf{x}}_{k|k-1}$ with the Kalman gain multiplied by the innovation $\tilde{\mathbf{y}}_k$. Remember that the innovation $\tilde{\mathbf{y}}_k$ is the difference between the measurement z_k and the estimated priori state $\hat{\mathbf{x}}_{k|k-1}$, so the innovation can both be positive and negative.

A little simplified the equation can be understood as we simply correct the estimate of the a priori state $\hat{\mathbf{x}}_{k|k-1}$, that was calculated using the previous state and the gyro measurement, with the measurement - in this case the accelerometer.

The last thing we will do is update the a posteriori error covariance matrix:

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H})\mathbf{P}_{k|k-1}$$

Where I is called the identity matrix and is defined as:

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

What the filter is doing is that it is basically self-correcting the error covariance matrix based on how much we corrected the estimate. This make sense as we corrected the state based the a priori error covariance matrix $P_{k|k-1}$, but also the innovation covariance S_k .

Implementing the filter

In this section I will use the equation from above to implement the filter into a simple c++ code that can be used for [balancing robots](#), [quadcopters](#) and other applications where you need to compute the angle, bias or rate.

In case you want the code next to you, it can be found at github:

<https://github.com/TKJElectronics/KalmanFilter>.

I will simply write the equations at the top of each step and then simplify them after that I will write how it is can be done i C and finally I will link to calculations done in [Wolfram Alpha](#) in the bottom of each step, as I used them to do the calculation.

Step 1:

$$\begin{aligned}\hat{\mathbf{x}}_{k|k-1} &= \mathbf{F}\hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}\dot{\theta}_k \\ \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k-1} &= \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k-1|k-1} + \begin{bmatrix} \Delta t \\ 0 \end{bmatrix} \dot{\theta}_k \\ &= \begin{bmatrix} \theta - \dot{\theta}_b \Delta t \\ \dot{\theta}_b \end{bmatrix}_{k-1|k-1} + \begin{bmatrix} \Delta t \\ 0 \end{bmatrix} \dot{\theta}_k \\ &= \begin{bmatrix} \theta - \dot{\theta}_b \Delta t + \dot{\theta} \Delta t \\ \dot{\theta}_b \end{bmatrix} \\ &= \begin{bmatrix} \theta + \Delta t(\dot{\theta} - \dot{\theta}_b) \\ \dot{\theta}_b \end{bmatrix}\end{aligned}$$

As you can see the a priori estimate of the angle is $\hat{\theta}_{k|k-1}$ is equal to the estimate of the previous state $\hat{\theta}_{k-1|k-1}$ plus the unbiased rate times the delta time Δt .

Since we can not directly measure the bias the estimate of the a priori bias is just equal to the previous one.

This can be written in C like so:

```
rate = newRate - bias;
angle += dt * rate;
```

Note that I calculate the unbiased rate, so it can be be used by the user as well.

Wolfram Alpha links:

- [Eq. 1.1](#)

Step 2:

$$\begin{aligned}\mathbf{P}_{k|k-1} &= \mathbf{F}\mathbf{P}_{k-1|k-1}\mathbf{F}^T + \mathbf{Q}_k \\ \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} &= \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k-1|k-1} \begin{bmatrix} 1 & 0 \\ -\Delta t & 1 \end{bmatrix} + \begin{bmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{bmatrix} \Delta t \\ &= \begin{bmatrix} P_{00} - \Delta t P_{10} & P_{01} - \Delta t P_{11} \\ P_{10} & P_{11} \end{bmatrix}_{k-1|k-1} \begin{bmatrix} 1 & 0 \\ -\Delta t & 1 \end{bmatrix} + \begin{bmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{bmatrix} \Delta t \\ &= \begin{bmatrix} P_{00} - \Delta t P_{10} - \Delta t(P_{01} - \Delta t P_{11}) & P_{01} - \Delta t P_{11} \\ P_{10} - \Delta t P_{11} & P_{11} \end{bmatrix}_{k-1|k-1} + \begin{bmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{bmatrix} \Delta t\end{aligned}$$

$$\begin{aligned}
&= \begin{bmatrix} P_{00} - \Delta t P_{10} - \Delta t (P_{01} - \Delta t P_{11}) + Q_{\theta} \Delta t & P_{01} - \Delta t P_{11} \\ P_{10} - \Delta t P_{11} & P_{11} + Q_{\dot{\theta}_b} \Delta t \end{bmatrix} \\
&= \begin{bmatrix} P_{00} + \Delta t (\Delta t P_{11} - P_{01} - P_{10} + Q_{\theta}) & P_{01} - \Delta t P_{11} \\ P_{10} - \Delta t P_{11} & P_{11} + Q_{\dot{\theta}_b} \Delta t \end{bmatrix}
\end{aligned}$$

The equations above can be written in C like so:

```

P[0][0] += dt * (dt*P[1][1] - P[0][1] - P[1][0] + Q_angle);
P[0][1] -= dt * P[1][1];
P[1][0] -= dt * P[1][1];
P[1][1] += Q_gyroBias * dt;

```

Note that this is the part of the code that there was an error in in the original code that I used.

Wolfram Alpha links:

- [Eq. 2.1](#)
- [Eq. 2.2](#)
- [Eq. 2.3](#)
- [Eq. 2.4](#)

Step 3:

$$\begin{aligned}
\tilde{\mathbf{y}}_k &= \mathbf{z}_k - \mathbf{H} \hat{\mathbf{x}}_{k|k-1} \\
&= \mathbf{z}_k - \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k-1} \\
&= \mathbf{z}_k - \theta_{k|k-1}
\end{aligned}$$

The innovation can be calculated in C like so:

```
y = newAngle - angle;
```

Wolfram Alpha links:

- [Eq. 3.1](#)

Step 4:

$$\begin{aligned}
\mathbf{S}_k &= \mathbf{H} \mathbf{P}_{k|k-1} \mathbf{H}^T + \mathbf{R} \\
&= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \mathbf{R} \\
&= P_{00k|k-1} + \mathbf{R} \\
&= P_{00k|k-1} + \text{var}(v)
\end{aligned}$$

Again the C code is pretty simple:

```
S = P[0][0] + R_measure;
```

Wolfram Alpha links:

- [Eq. 4.1](#)

Step 5:

$$\begin{aligned}
\mathbf{K}_k &= \mathbf{P}_{k|k-1} \mathbf{H}^T \mathbf{S}_k^{-1} \\
\begin{bmatrix} K_0 \\ K_1 \end{bmatrix}_k &= \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \mathbf{S}_k^{-1} \\
&= \begin{bmatrix} P_{00} \\ P_{10} \end{bmatrix}_{k|k-1} \mathbf{S}_k^{-1}
\end{aligned}$$

$$= \frac{\begin{bmatrix} P_{00} \\ P_{10} \end{bmatrix}_{k|k-1}}{S_k}$$

Note that in other cases S can be a matrix and you can not just simply divide P by S . Instead you have to calculate the inverse of the matrix. See the following [page](#) for more information on how to do so.

The C implementation looks like this:

```
K[0] = P[0][0] / S;
K[1] = P[1][0] / S;
```

Wolfram Alpha links:

- [Eq. 5.1](#)

Step 6:

$$\begin{aligned}\hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k \\ \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k} &= \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k-1} + \begin{bmatrix} K_0 \\ K_1 \end{bmatrix}_k \tilde{\mathbf{y}}_k \\ &= \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k-1} + \begin{bmatrix} K_0 \tilde{\mathbf{y}} \\ K_1 \tilde{\mathbf{y}} \end{bmatrix}_k\end{aligned}$$

Yet again the equation end up pretty short, and can be written as so in C:

```
angle += K[0] * y;
bias += K[1] * y;
```

Step 7:

$$\begin{aligned}\mathbf{P}_{k|k} &= (\mathbf{I} - \mathbf{K}_k \mathbf{H}) \mathbf{P}_{k|k-1} \\ \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k} &= \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} K_0 \\ K_1 \end{bmatrix}_k \begin{bmatrix} 1 & 0 \end{bmatrix} \right) \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} \\ &= \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} K_0 & 0 \\ K_1 & 0 \end{bmatrix}_k \right) \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} \\ &= \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} - \begin{bmatrix} K_0 P_{00} & K_0 P_{01} \\ K_1 P_{00} & K_1 P_{01} \end{bmatrix}\end{aligned}$$

Remember that we decrease the error covariance matrix again, since the error of the estimate of the state has been decreased.

The C code looks like this:

```
float P00_temp = P[0][0];
float P01_temp = P[0][1];

P[0][0] -= K[0] * P00_temp;
P[0][1] -= K[0] * P01_temp;
P[1][0] -= K[1] * P00_temp;
P[1][1] -= K[1] * P01_temp;
```

Wolfram Alpha links:

- [Eq. 7.1](#)
- [Eq. 7.2](#)
- [Eq. 7.3](#)

Note that I have found that the following variances works perfectly for most IMUs:

```
float Q_angle = 0.001;
float Q_gyroBias = 0.003;
float R_measure = 0.03;
```


Remember that it's very important to set the target angle at startup if you need to use the output at startup. For more information, see the [calibration routine for my balancing robot](#).

In case you missed it here is the library I wrote a library that can be used by any microcontroller that supports floating math. The source code can be found at github:

<https://github.com/TKJElectronics/KalmanFilter>.

If you prefer a video explanation about the Kalman filter, I recommend the following video series:

http://www.youtube.com/watch?v=FkCT_LV9Syk.

Note that you can not use the library if you need to represent something in a full 3D orientations, as euler angles suffer from what is called [Gimbal lock](#) you will need to use Quaternions to do that, but that is a whole nother story. For now take a look at the following [page](#).

This is all for know, I hope that you will find it helpfull, if you do or have any questions fell free to post a comment below - it supports [LaTeX syntax](#) as well, if you need to write equations.

If you spot any errors please let me know as well.

Categories: [Guides](#), [TKJ Electronics](#) Tags:

[Comments \(33\)](#) [Trackbacks \(0\)](#) [Leave a comment](#) [Trackback](#)



1.

ZiM

April 28th, 2015 at 17:21 | [#1](#)

[Reply](#) | [Quote](#)

Hi Lauszus,

Thank you so much for putting all this together! It helped me a lot.

I have a question. Sorry if other people have asked this before. My system has a periodic jerk input(the acceleration measurement spikes due to that) from knocking into the wall, which mess up the orientation estimation. If the system doesn't have that jerk it works fine.

From reading your document above I feel like maybe I should adjust my process noise covariance matrix accordingly. Am I on the right track? Also I am using a sampling frequency of 500Hz.

thank you very much for your time



2.

Tony

May 8th, 2015 at 06:45 | [#2](#)

[Reply](#) | [Quote](#)

Hi Lasuzus,

Thank you for the filter and all the explanation, my kalman filter is working very well with positive values of my angles from the gyro and the acc but when i try to use the negative values my angle just stays at 0 and after that's done nothing works until a reset my board, i already checked the angles one by one from the gyro and acc and y get the right negative values, i dont know whats going on, i hope you can help thanks



3.

chfakht

May 10th, 2015 at 20:17 | [#3](#)

[Reply](#) | [Quote](#)

Hi , i didn't understand very well this phrase :” Which is the previous estimated state based on the previous state and the estimates of the states before it.” ?? can you simplify plz thanks



4.

chfakht

May 10th, 2015 at 20:36 | [#4](#)

[Reply](#) | [Quote](#)

please how can i specifie the process noise covariances , i' m using a MPU-6050 and kalman filter

to read value from the gyroscope and accelerometer
thnaks



5. chfakht
May 11th, 2015 at 01:27 | [#5](#)
[Reply](#) | [Quote](#)

[@klaus](#)

hi please can you send me your matlab code to chfakht@gmail.com



6. Mike Veal
June 9th, 2015 at 13:02 | [#6](#)
[Reply](#) | [Quote](#)

This article is excellent, easily the best explanation of Kalman I' ve found and I' ve done quite a bit of reading.

I am still struggling to adapt this to my use though. I' m going wrong somewhere and I' d really appreciate some pointers. I am working on a altimeter.

I have a atmospheric pressure sensor, a BMP180. I want to implement a kalman filter to reduce the noise on readings from this part. Height error (noise) on the BMP180 measurement is 0.25m rms. There is only one input to the filter, height. Rate (velocity) may be calculated by looking at the previous height (filter output) and the current height (measurement including noise) and dividing by dt.

The input to the filter is height in meters, I' d like the output to be height in meters.

The output of the sensor (after conversion from hPa) is in meters. The sensor has internal compensation so bias can be considered to be zero.

Based on this article I have the following:

Step 0.

P matrix initialised to be 0,0,0,0 as I' m assuming that I know the start height.

Step 1.

Rate = Measured height-Previous height / dt

Height = Measured height

Step 2.

$P[0][0] += dt * (dt * P[1][1] - P[0][1] - P[1][0] + Q_height);$

$P[0][1] -= dt * P[1][1];$

$P[1][0] -= dt * P[1][1];$

$P[1][1] += Q_gyroBias * dt;$

I don' t have a gyro, I' ve only got one input parameter, so Q-gyroBias is zero.

Step 3.

$y = \text{Current measurement} - \text{Previous filter output};$

Step 4.

$S = P[0][0] + R_measure$

I' ve set R_measure to be 0.25 to match the rms noise in meters.

Step 5.

$K[0] = P[0][0] / S;$

$K[1] = P[1][0] / S;$

Step 6.

$\text{Height} = \text{Height} + K[0]*y$

$\text{Bias} = \text{Bias} + K[1]*y$

I think Bias will always be zero, but hey, I implemented it.

Step 7.

$\text{float } P00_temp = P[0][0];$

$\text{float } P01_temp = P[0][1];$

```
P[0][0] -= K[0] * P00_temp;
P[0][1] -= K[0] * P01_temp;
P[1][0] -= K[1] * P00_temp;
P[1][1] -= K[1] * P01_temp;
```

I've modelled this with an Excel spreadsheet and the kalman filter doesn't appear to be working. Any assistance greatly appreciated. It appears that by removing one of the inputs from the filter I've confused the heck out of myself!

Many thanks,
Mike Veal.



7.

[Kristian Lauszus](#)

June 9th, 2015 at 13:16 | [#7](#)

[Reply](#) | [Quote](#)

[@Mike Veal](#)

Sorry, but I don't have a lot of time at the moment, as I am writing my bachelor's thesis.

But have you considered using an accelerometer as well? All the code I have seen use the accelerometer z-axis in the combination with a barometer in order to estimate the height.

Note that I might be working on this next semester for my flight controller. I don't know what your timeframe is, but you can follow my Github repository:

<https://github.com/Lauszus/LaunchPadFlightController> where the code will be posted.



8.

Mike Veal

June 9th, 2015 at 13:28 | [#8](#)

[Reply](#) | [Quote](#)

@ Kristian

Sadly cost is a driver as are size and power. The device is to be used on a paraglider and is battery powered.

I need to have a filter running in the next month -- then it's off to Solvenia for a week of intensive altimeter testing!

Best of luck with the thesis.

If any one else is able to help I'd greatly appreciate it.



9.

[Kristian Lauszus](#)

June 9th, 2015 at 13:31 | [#9](#)

[Reply](#) | [Quote](#)

[@Mike Veal](#)

But an MPU-6050 is less than \$3 on eBay, so I can't see no reason for not using it?

Also a Kalman filter makes no sense if you are only using one sensor! That is why it is called a sensor fusion algorithm.



10.

Mike Veal

June 10th, 2015 at 12:09 | [#10](#)

[Reply](#) | [Quote](#)

Perhaps that's what I'm struggling to understand. I can find no simplification examples with just one input.

Adding an accelerometer /gyro would also make things much more complicated.

What I really need is two outputs, height and rate of change of height.

A 3 axis accelerometer / gyro could give me that information, but the glider will always have some forward speed as well as vertical speed.

Due to pitch, roll, yaw and repeatability of mounting axis, glider forwards motion will never be exactly on the X axis and vertical motion will never be exactly in the Z axis.

I'd have to try and separate a glider forward motion vector from the sensor output so that I could work out what constitutes vertical movement.

It sounds like a simple low pass filter would be easiest.
Many thanks for your help.



11. [Kristian Lauszus](#)
June 10th, 2015 at 17:20 | [#11](#)
[Reply](#) | [Quote](#)

[@Mike Veal](#)

That can be compensated for by rotating the coordinate system, as I do for my flight controller as well: <https://github.com/Lauszus/LaunchPadFlightController/blob/master/src/IMU.c>. Then you can just use the filtered (in this case 'accFiltered') z-axis and integrate that.

Yes if you only want to use the barometer, then a low pass filter seems like the way to go.



12. [Fabian](#)
August 1st, 2015 at 05:38 | [#12](#)
[Reply](#) | [Quote](#)

Hi, nice article,

have you compared the results from the Kalman filter and a complementary filter?, y did see the link to a web where they do that but the graphics dont look good.

I have made a comparison and with the parameters that I have used the Kalman filter and the complementary filters are practically the same, however I made a test, where I moved the IMU linearly in each axis and with the Kalman filter there was a variation of about +/-10 degrees but with the complementary filter the variation was a lot less (about +/- 2 deg).

Despite that I got similar results using both filters I dont know if I can improve the Kalman filter.



13. [Kristian Lauszus](#)
August 3rd, 2015 at 00:20 | [#13](#)
[Reply](#) | [Quote](#)

[@Fabian](#)

No not really, but to be honest I am starting to use a complimentary based filter more and more, as it is just much easier to tune, as takes less time to compute. For instance I have been using this lately with my flight controller:

<https://github.com/Lauszus/LaunchPadFlightController/blob/master/src/IMU.c>.



14. [Konstantin](#)
August 10th, 2015 at 16:14 | [#14](#)
[Reply](#) | [Quote](#)







Hi! I have a question about the implementation step 1. In this step you said that the apriori estimate of the angle was equal to the estimate of the previous state plus the unbiased rate times the delta time. The issue I am dwelling on is the the unbiased rate itself. This rate seems to be a difference between the rate at time k and the bias at time k-1|k-1. Is is permissible to subtract two values obtained from different time levels?

Thank you in advance!
Konstantin

P.S.: The expression in the brakets in the last row, step 1, would be

unbiased rate = $\theta_{dot,k} - \theta_{dot_bias,k-1|k-1}$

if you didn't omit the indeces.

15.  Erick Medina
September 3rd, 2015 at 00:50 | [#15](#)
[Reply](#) | [Quote](#)
- Hi, I' ve been checking your code about kalman filter, and I' m wondering why you divide by 131.0:
- ```
double gyroZrate = gyroZ / 131.0; // Convert to deg/s
```
- how can I get my own value?
16.  [Konstantin](#)  
September 7th, 2015 at 12:23 | [#16](#)  
[Reply](#) | [Quote](#)
- [@Erick Medina](#)  
It has to do with the so-called Sensitivity Scale Factor. Check the following document, page 12:  
MPU-6000 and MPU-6050 Product Specification Revision 3.3  
The factor is 131 LSB/(deg/s) which means that you must divide the raw data to 131 in order to get real angular velocity.
17.  Abhiroop  
September 9th, 2015 at 08:11 | [#17](#)  
[Reply](#) | [Quote](#)
- How do you compare the HX (which is the angle estimate) with z (which is the accel measurement)?
- Is this an approximation of some kind, and if so, what is the actual equation from which it is derived?
- In the case of this being an approximation, it will only work for balancing type scenarios (small deflections), right?
18.  Abhiroop  
September 9th, 2015 at 08:45 | [#18](#)  
[Reply](#) | [Quote](#)
- [@Sander](#)  
Hi,
- I have the exact doubt. Did you manage to find an explanation for it?
19.  Hasan Askari  
September 9th, 2015 at 15:08 | [#19](#)  
[Reply](#) | [Quote](#)
- Hi,
- First of all I must say that you wrote an excellent atricle. Now let me brief you my problem , I want to estimate heading for a vechile with using IMU . I am using Murata SCC1300-D02 sensor , it has 3axis accerolmeter and 1-axis gyrometer. I am not using any magnetometer so my question is that is it possible to estimate the heading(orientation) of the Vechile with the above mentioned sensor and using Kalman filter which you have written?
20.  Law  
September 27th, 2015 at 05:32 | [#20](#)  
[Reply](#) | [Quote](#)
- Nicely done. Found one typo: "control-input model" should be "control-input matrix".



21.

[Kristian Sloth Lauszus](#)September 30th, 2015 at 14:09 | [#21](#)[Reply](#) | [Quote](#)[@Hasan Askari](#)

No that will be very difficult. You didn't say what axis the gyro were measuring, but I guess it is the z-axis. Anyway I will not recommend this, as the integration will integrate noise as well.

You might want to take a look on how I did it for my flight controller:

<http://blog.tkjelectronics.dk/2015/08/bachelors-thesis-launchpad-flight-controller>. The relevant code is in here: <https://github.com/Lauszus/LaunchPadFlightController/blob/master/src/IMU.c>.

You might also want to read the chapter "Estimation of the orientation" in my report:

<https://github.com/Lauszus/LaunchPadFlightController/blob/master/docs/Kristian%20Sloth%20Lauszus%20-%20Flight%20Controller%20for%20Quad%20Rotor%20Helicopter%20in%20X-configuration.pdf>.

[@Law](#)

Not according to Wiki: [https://en.wikipedia.org/wiki/Kalman\\_filter#Underlying\\_dynamic\\_system\\_model](https://en.wikipedia.org/wiki/Kalman_filter#Underlying_dynamic_system_model), but thanks for your feedback anyway!

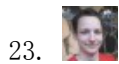


22.

[spookymelonhead](#)January 6th, 2016 at 08:24 | [#22](#)[Reply](#) | [Quote](#)

Hi..excellent article..thanks a ton.

I've been trying to estimate altitude from barometer(measured) and accelerometer(predicts and also gravity removed somehow)..so how do I find the covariance matrix considering if I know the error in accelerometer..how do I find error in position and velocity.?



23.

[Kristian Sloth Lauszus](#)January 6th, 2016 at 16:03 | [#23](#)[Reply](#) | [Quote](#)[@spookymelonhead](#)

I did this for my flight controller: <http://blog.tkjelectronics.dk/2015/08/bachelors-thesis-launchpad-flight-controller>. You can read more about it in the report:

<https://github.com/Lauszus/LaunchPadFlightController/raw/master/docs/Kristian%20Sloth%20Lauszus%20-%20Flight%20Controller%20for%20Quad%20Rotor%20Helicopter%20in%20X-configuration.pdf>.

The relevant code is here:

<https://github.com/Lauszus/LaunchPadFlightController/blob/master/src/AltitudeHold.c>.

Note that I did not use a Kalman filter, but uses a simple complimentary filter. The result can be seen here: <https://www.youtube.com/watch?v=HXX-2L1hKgI>



24.

[spookymelonhead](#)January 15th, 2016 at 09:22 | [#24](#)[Reply](#) | [Quote](#)[@Kristian Sloth Lauszus](#)

Hi..may I ask..How did you came up with the idea of using (theta dot b) as second state variable?



25.

[Muhammad Umair Masood](#)January 18th, 2016 at 08:46 | [#25](#)[Reply](#) | [Quote](#)[@Kristian Sloth Lauszus](#)

Hi, I have one question regarding the angle. In your code you have mentioned an argument as new angle. Can you tell me what is new angle representing here?

Best Regards,  
Umair



26. [Kristian Sloth Lauszus](#)  
January 18th, 2016 at 14:50 | [#26](#)  
[Reply](#) | [Quote](#)

[@spookymelonhead](#)

I saw some other code where they did that as well.

[@Muhammad Umair Masood](#)

newAngle is the angle measured using the accelerometer.



27. [Jim Remington](#)  
January 21st, 2016 at 23:58 | [#27](#)  
[Reply](#) | [Quote](#)

The derivation is missing a state variable (theta dot) and the rate gyro measurement should be used as a measurement, not as a “control variable”.

A more complete discussion of the problem is given in  
<http://home.earthlink.net/~schultdw/bbot/bbot.html>



28. [Kristian Sloth Lauszus](#)  
February 1st, 2016 at 21:06 | [#28](#)  
[Reply](#) | [Quote](#)

[@Jim Remington](#)

I know that the article has some shortcomings, but I don't have time to write a new one at the moment. But remember that I wrote this article back in High School before I was actually formally taught any of this stuff.



29. [spookymelonhead](#)  
February 25th, 2016 at 09:53 | [#29](#)  
[Reply](#) | [Quote](#)

and here's another good source for learning Kalman

<https://www.youtube.com/playlist?list=PLX2gX-ftPVXU3oUFNATxGXY90AULiqnWT>



30. [Kristian Sloth Lauszus](#)  
February 25th, 2016 at 10:06 | [#30](#)  
[Reply](#) | [Quote](#)

[@spookymelonhead](#)

Thanks for the link 😊



31. [Goldy](#)  
March 1st, 2016 at 09:52 | [#31](#)  
[Reply](#) | [Quote](#)

Hi Kristian,

Nice article. I am interested to use kalman filter to my following application can u suggest which code i use.

1. My application is to measure Tilt angle in X & Y direction +/- 35 degrees. I use analog device ADIS16209 (Inclinometer & accelerometer) with ADIS16265 (digital gyroscope). I interested to measure both X & Y direction accurate tilt angle in dynamic condition also.

2. Presently I measure accurate tilt angle in static condition without gyroscope.

Please suggest best algorithm



32.

[spookymelonhead](#)

March 12th, 2016 at 09:09 | [#32](#)

[Reply](#) | [Quote](#)

Hey..quick question..When we write

$$Y_k = Z_k - (H * X_k)$$

which is basically subtraction between measured value and predicted value.

we set H matrix equals as 1 for angle and 0 for bias. That implies no prediction for the Gyro bias via the model and No measurement for Gyro bias too.

But when it comes to updating the predicted states i.e

$$\text{New\_angle} = \text{Predicted\_angle} + \text{Kalman\_gain} * (\text{Innovation})$$

we are updating Gyro Bias

$$\text{GyroBias\_est} = \text{GyroBias\_est} + \text{Kalman\_gain} * \text{Innovation}$$

so same innovation is used for getting new estimates for angle and bias.

I know its correct..since with the help of measured value, it will try to correct Gyro\_bias..but it still seems odd..we updated bias with the innovation calculated for angle..!!



33.

lasdfdsa

June 10th, 2016 at 01:49 | [#33](#)

[Reply](#) | [Quote](#)

In the process noise covariance matrix,  $Q_k$ , I can understand why  $Q_{\text{angle}}$  gets multiplied by  $dt$  since the state you're estimating is the angle, but I do not understand why  $Q_{\text{gyroBias}}$  gets multiplied by  $dt$  when the state being estimated is the gyro bias, which if I understand correctly, is an angular velocity. Can you clarify this? Can you also clarify if  $Q_{\text{angle}}$  is the covariance matrix of the gyroscope measurement and not the accelerometer measurement?

Thanks

Comment pages

[« Previous](#) [1](#) ... [4](#) [5](#) [6](#) [2868](#)

1. No trackbacks yet.

|                      |                                           |
|----------------------|-------------------------------------------|
| <input type="text"/> | Name (required)                           |
| <input type="text"/> | E-Mail (will not be published) (required) |
| <input type="text"/> | Website                                   |
| <input type="text"/> |                                           |

[Subscribe to comments feed](#)

I'm not a robot

reCAPTCHA

[Privacy](#) - [Terms](#)

[Cortex-M4 and 9DOF for \\$10 Raspberry Pi GPIO control](#)  
[RSS](#)

[Donate](#)



## Recent Posts

- [Software Defined Radio with USRP N200 and LabVIEW \(GUIDE\)](#)
- [Bachelor's Thesis: LaunchPad Flight Controller](#)
- [IoT Arduino Vending Machine](#)
- [LaunchPad Flight Controller](#)
- [Handheld XV-11 LIDAR with STM32F429 and MATLAB](#)
- [IoT featured soccer table](#)
- [Full size DIY Balancing Robot](#)
- [USB Host Shield Mini in webshop](#)
- [STM32 Nucleo - a new mbed platform](#)
- [New contributor at TKJ Electronics - Diego](#)

## Categories

- [Arduino](#)
- [ARM](#)
- [Balanduino](#)
- [Bluetooth](#)
- [Development boards](#)
- [Embedded Linux](#)
- [Flight Controller](#)
- [FPGA](#)
- [Guides](#)
- [LaunchPad](#)
- [mini2440](#)
- [Multirotors](#)
- [News](#)
- [Other Microprocessors](#)
- [PIC](#)
- [Quadcopter](#)
- [Raspberry Pi](#)
- [Reviews](#)
- [STM32](#)
- [TKJ Electronics](#)
- [Tools](#)
- [USB](#)

## Blogroll

- [Circuits@Home](#)
- [EEVblog](#)
- [EEWeb](#)
- [Github](#)
- [Order PCB](#)
- [PCB Calculators](#)
- [PCBWeb](#)
- [SeeedStudio](#)
- [Twitter](#)
- [Webshop](#)
- [Youtube](#)



## Meta

- [Register](#)
- [Log in](#)
- [Entries RSS](#)
- [Comments RSS](#)
- [WordPress.org](#)

[Top](#) [WordPress](#)

Copyright © 2016 TKJ Electronics

Theme by [NeoEase](#). Valid [XHTML 1.1](#) and [CSS 3](#).