TKJ Electronics

Development with ease

A practical approach to Kalman filter and how to implement it

September 10th, 2012 Lauszus Leave a comment Go to comments

I have for a long time been interrested 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 interrested 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 that 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 succesfully 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 $\boldsymbol{x}_k$
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*:

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

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*:

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

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*:

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

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

$$\boldsymbol{x}_k$$

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

$$\boldsymbol{x}_k = \boldsymbol{F}x_{k-1} + \boldsymbol{B}u_k + w_k$$

Where x_k is the state matrix which is given by:

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

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 prevouis state x_{k-1}.

In this case F is defined as:

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

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 (°/s) at time k, this is also called the rate $\dot{\theta}$. We will actually rewrite the state equation as:

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

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

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

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:

$$\boldsymbol{w}_k \sim N\left(0, \boldsymbol{Q}_k\right)$$

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:

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

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

$$\boldsymbol{z}_k = \boldsymbol{H}x_k + v_k$$

As you can see the measurement z_k is given by the current state 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:

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

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

$$\boldsymbol{v}_k \sim N\left(0, \boldsymbol{R}\right)$$

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:

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

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:

$$var(v_k) = var(v)$$

Note that if you set the measurement noise variance $var(v)$ 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 $Q_{\dot{\theta}_b}$ and the measurement variance of the measurement noise $var(v)$. 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:

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

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 $\hat{x}_{k|k-1}$ as described in the beginning of the article.

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

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

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 F and the transpose of that F^T and add the current process noise Q_k at time k.

The error covariance matrix P in our case is a 2×2 matrix:

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

Update
The fist thing we will do is to compute the difference between the measurement z_k and the a priori state $x_{k|k-1}$, this is also called the innovation:

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

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

$$\tilde{\boldsymbol{y}}_k = \left[\tilde{\boldsymbol{y}}\right]_k$$

The next thing we will do is calculate whats called the innovation covariance:

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

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

The bigger the value of the measurement noise the larger the value of S, this means that we do not trust the incoming measurement that much.
In this case S is not a matrix and is just written as:

$$\boldsymbol{S}_k = \left[\boldsymbol{S}\right]_k$$

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

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

You can see that if we do not trust the innovation that much the innovation covariance S will be high and if we trust the estimate of the state then the error covariance matrix P will be small the Kalman gain will therefore be small and oppesite 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 H is used to map the state of the error covariance matrix P into observed space. We then compare the error covariance matrix by multiplying with the inverse of the innovation covariance S.

This make sense as we will use the observation model 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:

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

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:

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

Take a look at my calibration routine for more information.

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

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

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

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

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

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

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

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

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:

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

$$\begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k-1} = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k-1|k-1} + \begin{bmatrix} \Delta t \\ 0 \end{bmatrix} \dot{\theta}_k$$

$$= \begin{bmatrix} \theta - \dot{\theta}_b\Delta t \\ \dot{\theta}_b \end{bmatrix}_{k-1|k-1} + \begin{bmatrix} \Delta t \\ 0 \end{bmatrix} \dot{\theta}_k$$

$$= \begin{bmatrix} \theta - \dot{\theta}_b\Delta t + \dot{\theta}\Delta t \\ \dot{\theta}_b \end{bmatrix}$$

$$= \begin{bmatrix} \theta + \Delta t(\dot{\theta} - \dot{\theta}_b) \\ \dot{\theta}_b \end{bmatrix}$$

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:

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

$$\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k-1|k-1} \begin{bmatrix} 1 & 0 \\ -\Delta t & 1 \end{bmatrix} + \begin{bmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{bmatrix} \Delta t$$

$$= \begin{bmatrix} P_{00} - \Delta t P_{10} & P_{01} - \Delta t P_{11} \\ P_{10} & P_{11} \end{bmatrix}_{k-1|k-1} \begin{bmatrix} 1 & 0 \\ -\Delta t & 1 \end{bmatrix} + \begin{bmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{bmatrix} \Delta t$$

$$= \begin{bmatrix} P_{00} - \Delta t P_{10} - \Delta t(P_{01} - \Delta t P_{11}) & P_{01} - \Delta t P_{11} \\ P_{10} - \Delta t P_{11} & P_{11} \end{bmatrix}_{k-1|k-1} + \begin{bmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{bmatrix} \Delta t$$

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

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

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:

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

$$= \boldsymbol{z}_k - \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k-1}$$

$$= \boldsymbol{z}_k - \theta_{k|k-1}$$

The innovation can be calculated in C like so:

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

Wolfram Alpha links:

- Eq. 3.1

Step 4:

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

$$= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \boldsymbol{R}$$

$$= P_{00_{k|k-1}} + \boldsymbol{R}$$

$$= P_{00_{k|k-1}} + var(v)$$

Again the C code is pretty simple:

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

Wolfram Alpha links:

- Eq. 4.1

Step 5:

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

$$\begin{bmatrix} K_0 \\ K_1 \end{bmatrix}_k = \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \boldsymbol{S}_k^{-1}$$

$$= \begin{bmatrix} P_{00} \\ P_{10} \end{bmatrix}_{k|k-1} \boldsymbol{S}_k^{-1}$$

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

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:

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

$$\begin{bmatrix}\theta \\ \dot{\theta}_b\end{bmatrix}_{k|k} = \begin{bmatrix}\theta \\ \dot{\theta}_b\end{bmatrix}_{k|k-1} + \begin{bmatrix}K_0 \\ K_1\end{bmatrix}_k \tilde{\boldsymbol{y}}_k$$

$$= \begin{bmatrix}\theta \\ \dot{\theta}_b\end{bmatrix}_{k|k-1} + \begin{bmatrix}K_0\,\tilde{\boldsymbol{y}} \\ K_1\,\tilde{\boldsymbol{y}}\end{bmatrix}_k$$

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:

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

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

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

$$= \begin{bmatrix}P_{00} & P_{01} \\ P_{10} & P_{11}\end{bmatrix}_{k|k-1} - \begin{bmatrix}K_0\,P_{00} & K_0\,P_{01} \\ K_1\,P_{00} & K_1\,P_{01}\end{bmatrix}$$

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.

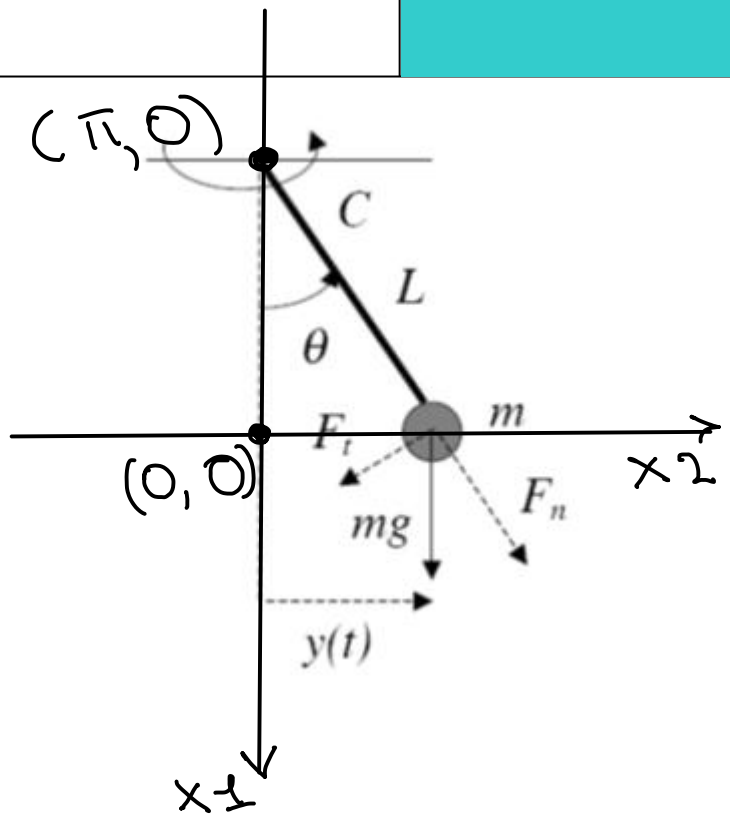
Note that you can not use the library if you need to represent something in a full 3D orientations, as euler angles suffer from what is called Gimbal lock you will need to use Quaternions to do that, but that is a whole nother story. For now take a look at the following page.

This is all for know, I hope that you will find i 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 (343) Trackbacks (7) Leave a comment Trackback

Example – Simple pendulum

Simple pendulum



Input: $u(t) = C(t)$

State: $x_{1(t)} = \theta(t), \quad x_{2(t)} = \dot{\theta}(t),$

Output: $y(t) = L \sin(\theta)$

$$\bar{x} = \begin{bmatrix} 2k\pi \\ 2k\pi \end{bmatrix}, \quad \bar{x} = \begin{bmatrix} k\pi \\ 2k\pi \end{bmatrix}$$

Nonlinear model:

$$\begin{cases} \dot{x}_1(t) = x_2(t) \\ \dot{x}_2(t) = -\dfrac{g}{L} \sin x_1(t) - \dfrac{b}{mL} x_2(t) + \dfrac{1}{mL^2} u(t) \\ y(t) = L \sin x_1(t). \end{cases}$$

$$F = \frac{\partial f}{\partial x}\Big|_{(\bar{x}, \bar{u})} = \begin{bmatrix} 0 & 1 \\ -\frac{g}{L}\cos\bar{x}_1 & -\frac{b}{mL} \end{bmatrix}$$

Example – Simple pendulum

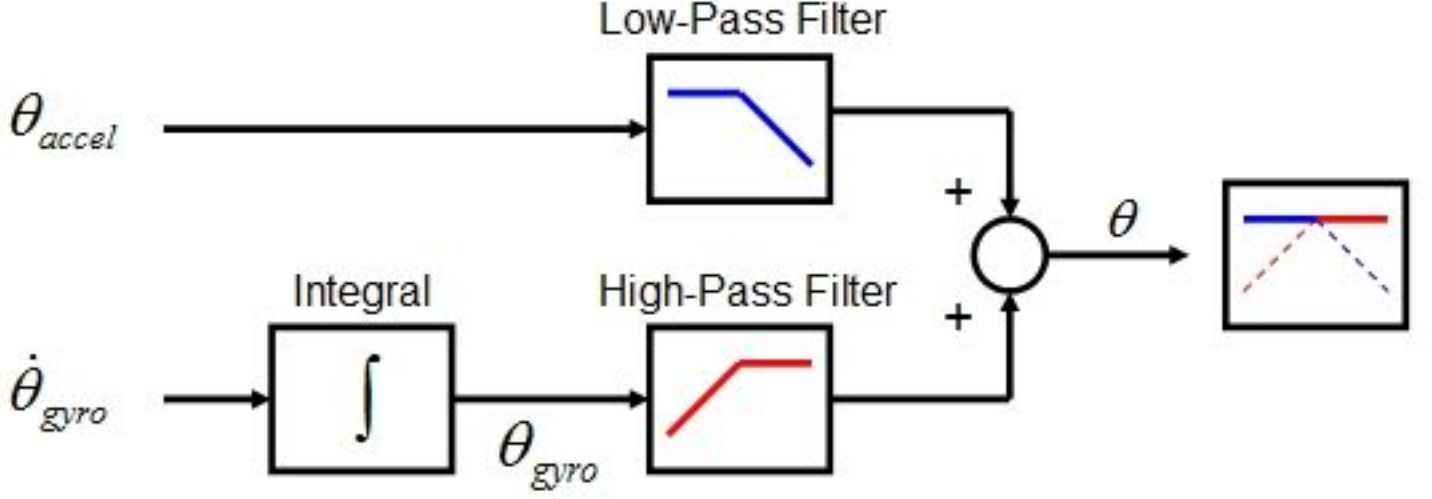
Linear model in the neighbourhood of (0,0)

$$\mathbf{A} = \left.\frac{\partial f}{\partial x}\right|_{\mathbf{x}=\mathbf{x_0};u=0} = \begin{bmatrix} 0 & 1 \\ -\frac{g}{L}\cos x_{10} & -\frac{b}{mL} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{g}{L} & -\frac{b}{mL} \end{bmatrix}$$

$$\mathbf{B} = \left.\frac{\partial f}{\partial u}\right|_{\mathbf{x}=\mathbf{x_0};u=0} = \begin{bmatrix} 0 \\ \frac{1}{mL^2} \end{bmatrix}; \mathbf{C} = \left.\frac{\partial g}{\partial x}\right|_{\mathbf{x}=\mathbf{x_0};u=0} = \begin{bmatrix} L\cos x_{10} & 0 \end{bmatrix} = \begin{bmatrix} L & 0 \end{bmatrix}$$

$$\mathbf{D} = \left.\frac{\partial g}{\partial u}\right|_{\mathbf{x}=\mathbf{x_0};u=0} = [0]$$

$$\begin{cases} \dot{x}_1(t) = x_2(t) \\ \dot{x}_2(t) = -\frac{g}{L}\, x_1(t) - \frac{b}{mL}\, x_2(t) + \frac{1}{mL^2}\, u(t) \\ y(t) = L\, x_1(t). \end{cases}$$


Low-Pass Filter
θ_{accel}
+
θ
Integral
High-Pass Filter
+
$\dot{\theta}_{gyro}$
$\int$
θ_{gyro}