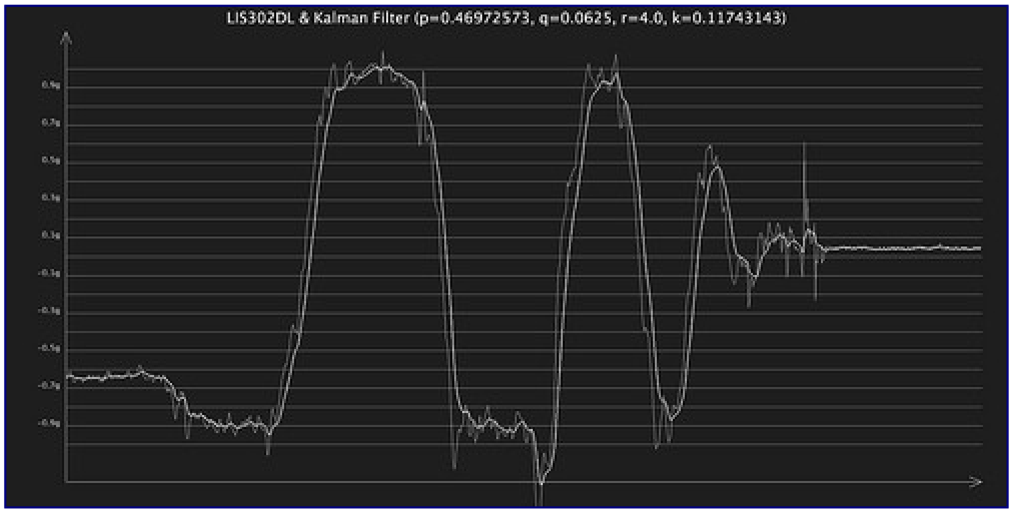
http://interactive-matter.eu/blog/2009/12/18/filtering-sensor-data-with-a-kalman-filter/

Filtering Sensor Data with a Kalman Filter

December 18, 2009

in Tinkering

Some days ago I wrote about noise with a LIS302DL accelerometer. There is obviously something wrong with my hardware. But before I get a new version out I need to implement some software filtering.



After some unsuccessful results with low pass filters I choose the Kalman Filter. The Kalman Filter involves an awful lot of complicated mathematics. But fortunately I just needed a simple one dimensional Kalman Filter without any driving information, which makes the filter much easier.

Implementing the Kalman Filter

If you check the Kalman Filter formulas, you will encounter a lot of complicated matrix and vector math. But after some research on the application domain for an LIS302DL accelerometer I found out that a simple one dimensional filter will do the job. The accelerometer puts out three dimensional acceleration data. But if you just want to get clean acceleration data from it all formulas simply deal with one dimension. So it was easier to use three different one dimensional Kalman filters. It all ended in a small set of formulas:

x = x
p = p + q;

k = p / (p + r);
*x = x + k * (measurement – x);*
*p = (1 – k) * p;*

The first two formulas represent the prediction of the Kalman Filter. And since there is no information about the driving forces it is very simple. The second three formulas calculate the measurement update. The variables are *x* for the filtered value, *q* for the process noise, *r* for the sensor noise, *p* for the estimated error and *k* for the Kalman Gain. The state of the filter is defined by the values of these variables.

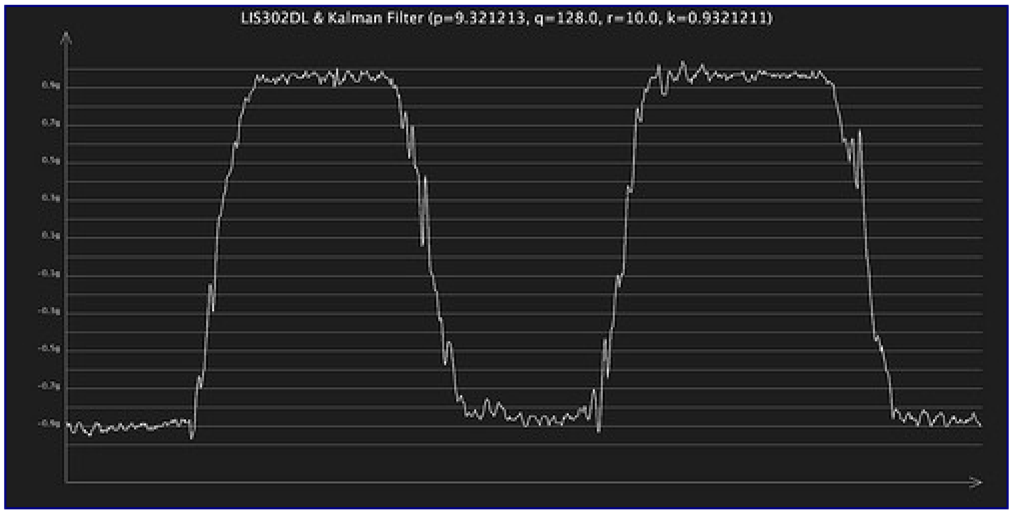
The filter is applied with each measurement and initialized with the process noise *q*, the sensor noise *r*, the initial estimated error *p* and the initial value *x*. The initial values for *p* is not very important since it is adjusted during the process. It must be just high enough to narrow down. The initial value for the readout is also not very important, since it is updated during the process.

But tweaking the values for the process noise and sensor noise is essential to get clear readouts.

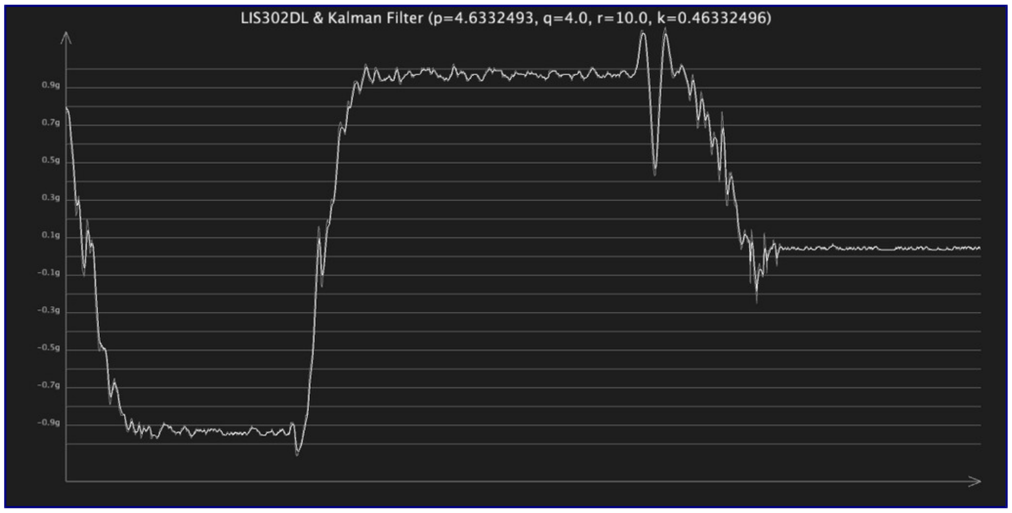
Tweaking the Kalman Filter

The first results were less than ideal. So I decided to use processing and my good old LIS302DL breakout board to get some insight of the optimal values of the different parameters:

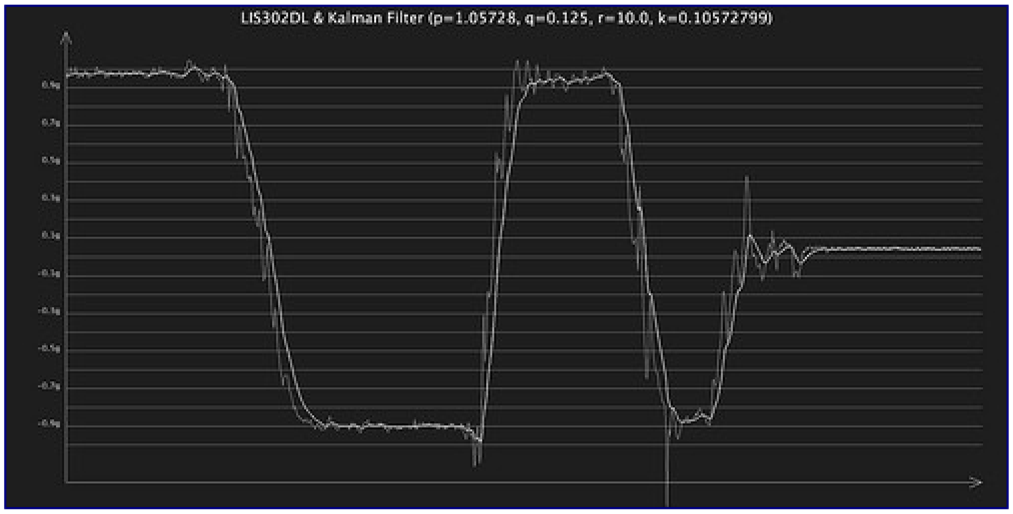
First I looked into the importance of the process noise *q*, starting by a very high value of 128 (which is the maximum output of the accelerometer):



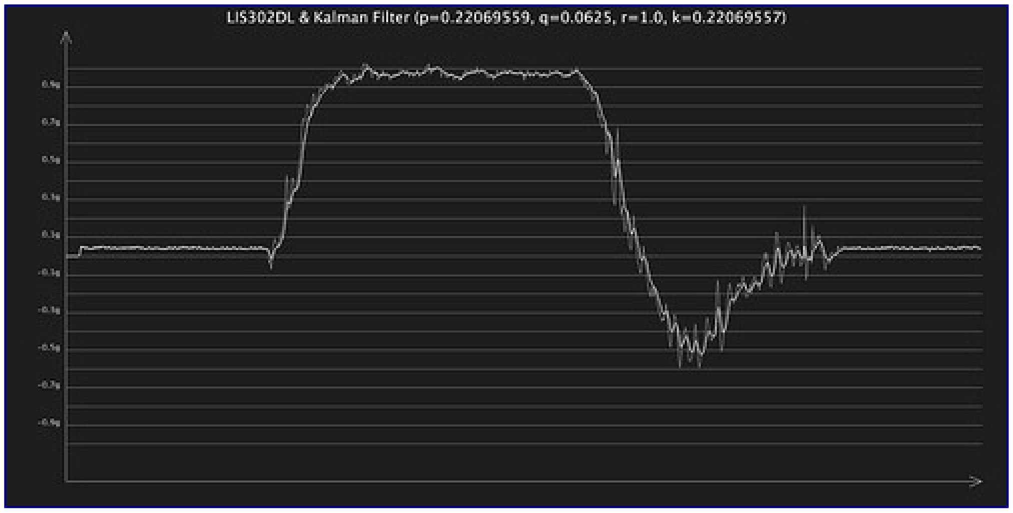
You see there is nearly no difference between the filtered data (white) and the original data (grey) – since you just see the white line.
If you turn down the process noise, e.g. to a value of 4 things start to change:



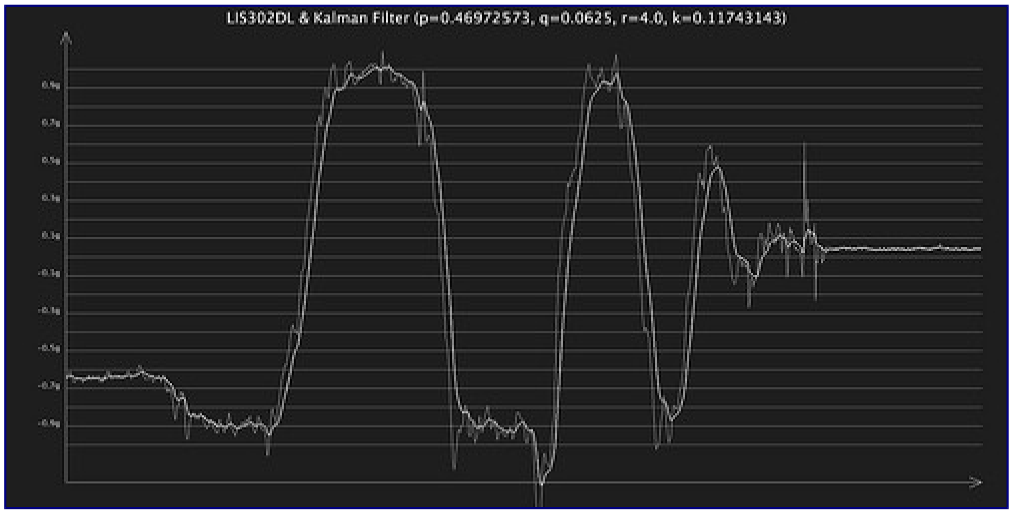
If you look closely you see that the sensor data often overshoots the filtered value. The filtered value is pretty close to the real value, but is missing a lot of noise. For my application I needed a more static output, smoothing out nearly all of the noise, giving a clean an steady signal. So lets turn down the process noise value a bit more, e.g. something like 0.125:



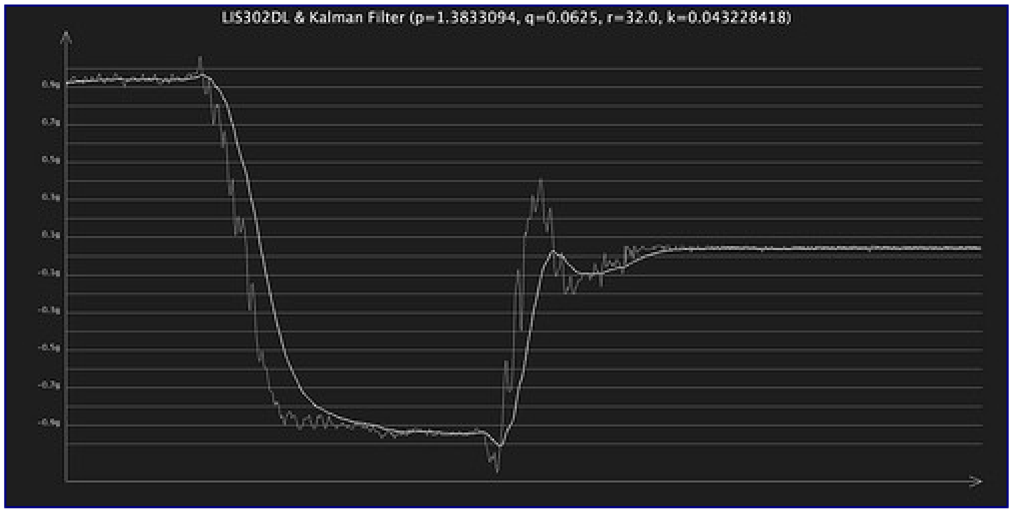
Now we got a quite a clean signal from quite a noisy source. It lags a bit behind the real data, but that is not critical for my application. Compared to the amount of noise reduction it is still quite fast.
After this is set let’s play a bit around with the sensor noise *r*:
We start back at *r*=1:



As expected if we turn down the assumed noise of the sensor the Kalman filter ‘relies’ more on the sensor data and gives more noisy results. If we increase the noise factor of the sensor to 4 we get a more stable result again:



If we go up extreme and assume a noise level of 32 we get the expected steady result:



As expected the filtered value lags significantly behind the real measurement. But it is a much cleaner signal. But it is questionable if this is still useful.

Implementing Kalman in C

Implementing this filter in C code (e.g. for an Arduino) is quite simple. First of all we define a state for a kalman filter:

```
typedef struct {
  double q; //process noise covariance
  double r; //measurement noise covariance
  double x; //value
  double p; //estimation error covariance
  double k; //kalman gain
} kalman_state;
```

Next we need a function to initialize the kalman filter:

```
kalman_state
kalman_init(double q, double r, double p, double intial_value)
{
  kalman_state result;
  result.q = q;
  result.r = r;
  result.p = p;
  result.x = intial_value;

  return result;
}
```

And a function to update the kalman state, by calculating a prediction and verifying that against the real measurement:

```
void
kalman_update(kalman_state* state, double measurement)
{
  //prediction update
  //omit x = x
  state->p = state->p + state->q;

  //measurement update
  state->k = state->p / (state->p + state->r);
  state->x = state->x + state->k * (measurement - state->x);
  state->p = (1 - state->k) * state->p;
}
```

What did we learn from this?

First of all the Kalman filter can be much easier to implement if the underlying model is simple. We had not several dimensions, nor different sensor data to combine. The results are pretty good, but it is a challenge to find the right values for the process and sensor noise. Instead of just coming up with some guesses for the values, they can of course be estimated, but than we have to live with the result of the estimation.

In the end it is a good, robust, simple filter (in one dimension).

I think that in lack of several sensors we throw out the most advantage of the filter to combine several sensor data to a combined result. I think going a bit more in this direction can lead to real interesting results.

Or is there a much easier filter to use for my problem, leading to better results?

Related posts:

- BMP085 Barometric Pressure Sensor Breakout Boards arrived!
- Arduino & Barometric Pressure Sensor BMP085
- Decoupling LIS302DL – There I fixed it!