Kalman on Cortex-M with CMSIS-DSP

Christophe Favergeon

October 2020

Contents

1 Introduction

This document is explaining how to implement an Extended Kalman filter for orientation estimation on a Cortex-M using the CMSIS-DSP library.

1.1 Kalman filter in a nutshell

The Kalman filter is the application of a Bayesian filter to a linear model with normal random variables.

1.1.1 Linear evolution model

Let's assume our knowledge of the system is described by a state $x_{k-1|k-1}$ at instant k and that time is discrete. Let's assume the control is c_k.

We assume that the new state should be:

$$x_{k|k-1} = Fx_{k-1|k-1} + G(c_k + n_k) \tag{1.1.1}$$

$x_{k|k-1}$ is the prediction of the new state at time k based on our knowledge at time $k-1$ which is $x_{k-1|k-1}$.

It is a prediction because the control is not perfectly known and because a model is always an approximation.

This uncertainty is modeled with a noise n_k which is a centered normal random variable.

As consequence, the state variable must also be a normal variable.

The linear transform of a normal variable is also a normal variable.

So, we just need to compute how the mean and variance are transforming.

The mean is transformed as described with equation 1.1.1.

If P is the covariance matrix of the normal variable $x_{k-1|k-1}$ and Q the covariance matrix of the noise n_k then:

$$P_{k|k-1} = F.P_{k-1|k-1}.F^t + G.Q.G^t \tag{1.1.2}$$

Note that the noise component was written:

$$Gn_k \tag{1.1.3}$$

Because n_k is the noise of a sensor in local coordinates and we can measure it. If Q is its covariance matrix then the noise on the state variables is $G.Q.G^t$.

1.1.2 Observation

The covariance matrix of the state variable has been changed by the evolution. We are less certain about the state.

We want to update our knowledge about the state from the observation.

We want to compute:

$$p(x_{k|k}|z_k) \tag{1.1.4}$$

where z_k is the observation and $x_{k|k}$ is our knowledge of the state for time k based on the evolution model ($x_{k|k-1}$) and the observation.

Using Bayes theorem, we can compute this as:

$$p(x_{k|k}|z_k) = \frac{p(z_k|x_{k|k-1})p(x_{k|k-1})}{p(z_k)} \tag{1.1.5}$$

$p(z_k|x_{k|k-1})$ is given by the observation model which is assumed to be linear.

$$z_k = H.x_{k|k-1} + r_k \tag{1.1.6}$$

where r_k is the observation noise with covariance matrice R (and assumed to have a zero mean).

$p(x_{k|k}|z_k)$ is a normal variable so we need to compute its mean and covariance matrix. The most likely value will be the mean.

After some computations which are a bit tricky, we finally get the update equations:

$$y_k = z_k - H.x_{k|k-1} \tag{1.1.7}$$

$$S_k = H.P_{k|k-1}.H^t + R \tag{1.1.8}$$

$$K = P_{k|k-1}.H^t.S_k^{-1} \tag{1.1.9}$$

$$x_{k|k} = x_{k|k-1} + K.y_k \tag{1.1.10}$$

$$P_{k|k} = (I - K.H)P_{k|k-1} \tag{1.1.11}$$

And this is the Kalman filter.

1.2 Quaternion

Orientation will be described using a normalized quaternion : it is our state x.

A normalized quaternion (q_0, q_1, q_2, q_3) is corresponding to the following rotation matrix:

$$R = \begin{pmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_0q_3) & 2(q_0q_2 + q_1q_3) \\ 2(q_0q_3 + q_1q_2) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_0q_1 + q_2q_3) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{pmatrix} \tag{1.2.1}$$

This matrix R is describing the transformation to apply to the body to get its position in space.

As a consequence, a vector $\overrightarrow{v}$ measured in the world coordinate is corresponding to a vector $R^t.\overrightarrow{v}$ in body coordinates since R^t is the inverse of R.

The body is rotating. To know how the quaternion is transformed due to this rotation in an infinitesimal duration, we need to use the angular speed $(\omega_x, \omega_y, \omega_z)$ and compute the following matrix:

$$S(\omega) = \begin{pmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{pmatrix} \tag{1.2.2}$$

The derivative of the quaternion q is given by :

$$q' = \frac{1}{2} S(\omega) q \tag{1.2.3}$$

So, we can write the update equation of the quaternion as:

$$q_0(t + dt) = q_0 + \frac{1}{2}\mathrm{dt}(-q_1\omega_x - q_2\omega_y - q_3\omega_z) \tag{1.2.4}$$

$$q_1(t + dt) = q_1 + \frac{1}{2}\mathrm{dt}(q_0\omega_x + q_2\omega_z - q_3\omega_y) \tag{1.2.5}$$

$$q_2(t + dt) = q_2 + \frac{1}{2}\mathrm{dt}(q_0\omega_y - q_1\omega_z + q_3\omega_x) \tag{1.2.6}$$

$$q_3(t + dt) = q_3 + \frac{1}{2}\mathrm{dt}(q_0\omega_z + q_1\omega_y - q_2\omega_x) \tag{1.2.7}$$

This update equation is corresponding to the Euler integration scheme. It is not the most accurate and a Runge–Kutta method may be preferred.

Note that the speed ω is expressed in local coordinates. The rotation of the body is in world coordinates. The equation 1.2.3 is taking this into account.

2 Prediction

With quaternions, the update equations are no more linear. So, to be able to apply the Kalman update formula, we linearise the equations around the current point : it is the Extended Kalman filter.

The evolution model is a non linear function f.

The matrixes to be applied to the state covariance and process covariances are F_k and G_k and must be recomputed at each time step. They are extracted from the Jacobian matrix of f. F_k is the part related to state variables and G_k is the part related to control variables. In the code G is also named FQ.

The new update equation is very similar to the ones in 1.1.1 and 1.1.2.

For the update of the mean, f is used instead of the F matrix.

For the update of the covariance, we use the new matrixes F_k and G_k.

$$x_{k|k-1} = f(x_{k-1|k-1}, c_k) \tag{2.0.1}$$
$$P_{k|k-1} = F_k.P_{k-1|k-1}.F_k^t + G_k.Q.G_k^t \tag{2.0.2}$$

Q is the covariance matrix of the control variables.

The formula for f is just giving the quaternion update for a given duration dt.

It is important to note that the covariance noise due to control has to be updated each time since it depends on the matrix G_k.

For the linear filter, G_k being constant, the noise convariance is fixed. It is not the case here.

c_k is ω_k since the control is the angular speed measured by the gyro. Starting from now, we will note ω with w to be closer to the final C code.

$$f = \begin{pmatrix} \frac{1}{2}\mathrm{dt}(-q_1 w_x - q_2 w_y - q_3 w_z) + q_0 \\ \frac{1}{2}\mathrm{dt}(q_0 w_x + q_2 w_z - q_3 w_y) + q_1 \\ \frac{1}{2}\mathrm{dt}(q_0 w_y - q_1 w_z + q_3 w_x) + q_2 \\ \frac{1}{2}\mathrm{dt}(q_0 w_z + q_1 w_y - q_2 w_x) + q_3 \end{pmatrix} \tag{2.0.3}$$

The derivative according to the state and control variables are giving by the matrixes:

$$F_k = \begin{pmatrix} 1 & -\frac{\mathrm{dt}w_x}{2} & -\frac{\mathrm{dt}w_y}{2} & -\frac{\mathrm{dt}w_z}{2} \\ \frac{\mathrm{dt}w_x}{2} & 1 & \frac{\mathrm{dt}w_z}{2} & -\frac{\mathrm{dt}w_y}{2} \\ \frac{\mathrm{dt}w_y}{2} & -\frac{\mathrm{dt}w_z}{2} & 1 & \frac{\mathrm{dt}w_x}{2} \\ \frac{\mathrm{dt}w_z}{2} & \frac{\mathrm{dt}w_y}{2} & -\frac{\mathrm{dt}w_x}{2} & 1 \end{pmatrix} \tag{2.0.4}$$

$$G_k = FQ_k = \begin{pmatrix} -\frac{\mathrm{dt}q_1}{2} & -\frac{\mathrm{dt}q_2}{2} & -\frac{\mathrm{dt}q_3}{2} \\ \frac{\mathrm{dt}q_0}{2} & -\frac{\mathrm{dt}q_3}{2} & \frac{\mathrm{dt}q_2}{2} \\ \frac{\mathrm{dt}q_3}{2} & \frac{\mathrm{dt}q_0}{2} & -\frac{\mathrm{dt}q_1}{2} \\ -\frac{\mathrm{dt}q_2}{2} & \frac{\mathrm{dt}q_1}{2} & \frac{\mathrm{dt}q_0}{2} \end{pmatrix} \tag{2.0.5}$$

3 Observation

The observation update equations are nearly the same but since the observation is not linear, we need to use a non linear function h to update the mean and the Jacobian matrix H_k to update the covariance.

The formula is thus:

$$y_k = z_k - h(x_{k|k-1}) \tag{3.0.1}$$
$$S_k = H_k.P_{k|k-1}.H_k^t + R_k \tag{3.0.2}$$
$$Ka_k = P_{k|k-1}.H_k^t.S_k^{-1} \tag{3.0.3}$$
$$x_{k|k} = x_{k|k-1} + Ka_k.y_k \tag{3.0.4}$$
$$P_{k|k} = (I - Ka_k.H_k).P_{k|k-1} \tag{3.0.5}$$

The rotation matrix R to apply to the body to get its orientation is allowing us to convert a vector from the world frame to the body frame using R^t. (In the equation 3.0.1, R_k is not the rotation but the observation noise covariance. This naming is to be coherent with the source code of the implementation.)

So, when we express the body acceleration $\overrightarrow{acc_{body}}$ in the body frame of reference, we should have (when the body is not moving):

$$\overrightarrow{acc_{body}} = R^t. \begin{pmatrix} 0 \\ 0 \\ -9.81 \end{pmatrix} \tag{3.0.6}$$

Similarly, when we express the magnetic field in the body frame of reference we should get:

$$\overrightarrow{mag_{body}} = R^t. \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix} \tag{3.0.7}$$

So our x axis in the world frame of reference is from north to south.

And we get the observation function h :

$$h = \begin{pmatrix} -19.62(q_1 q_3 - q_0 q_2) \\ -19.62(q_0 q_1 + q_2 q_3) \\ -9.81\,(q_0^2 - q_1^2 - q_2^2 + q_3^2) \\ -1.\,(q_0^2 + q_1^2 - q_2^2 - q_3^2) \\ -2.(q_1 q_2 - q_0 q_3) \\ -2.(q_0 q_2 + q_1 q_3) \end{pmatrix} \tag{3.0.8}$$

And the linearized H_k is given by:

$$H_k = \begin{pmatrix} 19.62 q_2 & -19.62 q_3 & 19.62 q_0 & -19.62 q_1 \\ -19.62 q_1 & -19.62 q_0 & -19.62 q_3 & -19.62 q_2 \\ -19.62 q_0 & 19.62 q_1 & 19.62 q_2 & -19.62 q_3 \\ -2.q_0 & -2.q_1 & 2.q_2 & 2.q_3 \\ 2.q_3 & -2.q_2 & -2.q_1 & 2.q_0 \\ -2.q_2 & -2.q_3 & -2.q_0 & -2.q_1 \end{pmatrix} \tag{3.0.9}$$

If the orientation estimation is wrong then the observation will be different from the expected gravity and magnetic field.

Note that during a movement, the board will measure the gravity and acceleration of the board.

So the gravity reference will be lost and this update equation will no more be useful. It is only when the board is kept still (or not moving too quicky) that this will allow the Kalman filter to converge and estimate some orientation.

4 Sensors and calibration

Observation equations are requiring the noise covariance R and the evolution equations are requiring the control noise Q.

It has to be measured. A normal distribution should be fitted to the noise.

4.1 Magnetometer

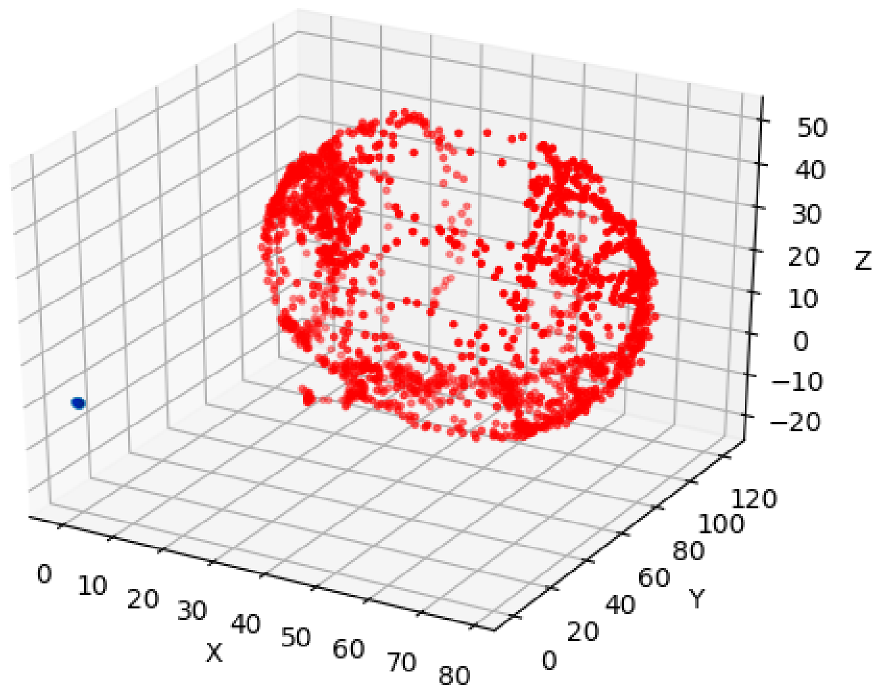
The magnetometer needs to be calibrated.

We expect the measurements to be on a sphere. They generally will be on an ellipse.

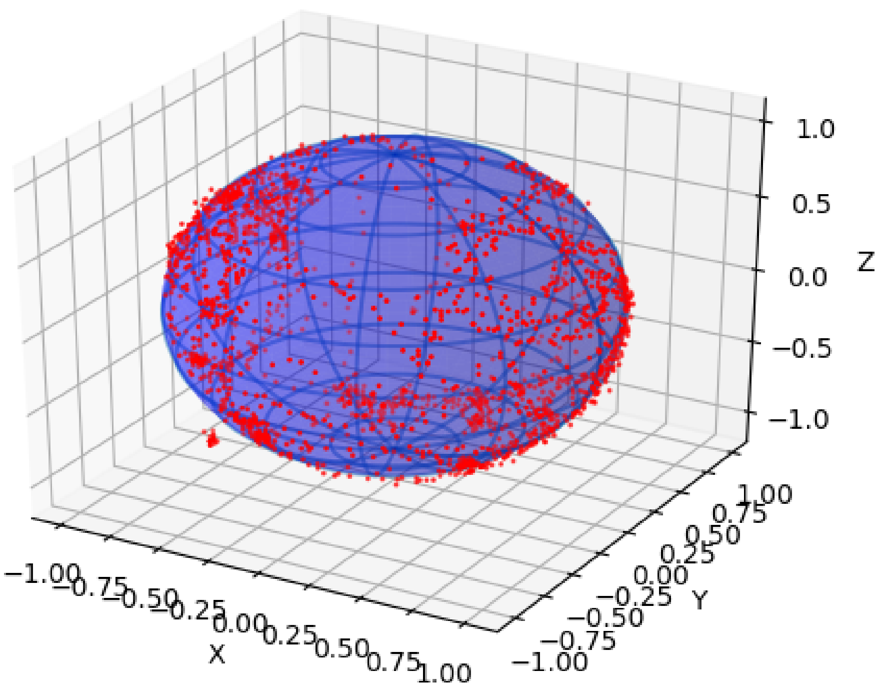
We expect the sphere to be centered in $(0, 0, 0)$ and it won't be the case.

Finally, since we are only interested in the direction of the magnetic field, we expect the sphere to have a radius of 1.

Here in blue we have the target sphere and in red the measured one.



After calibration, the red dots are on the blue sphere.



To do this calibration, we need to apply an offset and a matrix.

$$\overrightarrow{CM_{body}} = M.(\overrightarrow{M_{body}} - \overrightarrow{offset}) \tag{4.1.1}$$

$\overrightarrow{CM_{body}}$ is the calibrated measurement.

To fit an ellipse to a cloud of point, the following article was used: https://ieeexplore.ieee.org/document/1290055

To map the ellipse onto a unit sphere, the following article was used: https://ieeexplore.ieee.org/document/6289882

4.2 Gyroscope

According to the litteratue, it looks like the biases in the gyroscope noise are real and not just a measurement error.

Since the angular speeds are integrated, those biases will result in a drift. So the orientation estimation will diverge.

Those biases should be removed from the measurement before integrating.

But we can expect those biases to evolve in time (dependence on temperature etc ...). There is no automatic way (from this Kalman) to estimate those biases again. There are not enough observations and too many degree of freedom if the biases are included in the state vector.

Those biases would have to be measured again from time to time.

5 Implementation

5.1 Prediction

A macro is used to access the matrix elements using a (row,column) specification.

```
#define M(A,r,c) (A.pData[r * A.numCols + c])
```

We update the quaternion using the equation 2.0.3. Then we normalize the quaternion because only a normalized quaternion is representing a rotation and we want to prevent rounding errors from breaking this property.

We do this normalization each time although in a final implementation it may not ne needed to do it as often. This normalization is involving a square root so is costly.

```
f_update();

normalizeQuat(&NQ0,&NQ1,&NQ2,&NQ3);
```

Then, the F_k and G_k matrixes are computed using equations 2.0.4 and 2.0.5

The code is generated from a computer algebra system since it is the part of the implementation to be changed each time the model is changed.

As consequence, this code is not optimized. A better generation procedure could take into account the symmetry of the matrixes and the possibility of vectorizing to generate a better code.

```
F_update();

FQ_update();
```

We update covariance using equation 2.0.1.

$OmegaErrors$ is the covariance matrix for the gyro errors (in the equation it is named Q). Here, in the code, Q is the result after applying the G_k matrix.

We symmetrize the covariance matrix NP by copying the upper triangle onto the lower one. It is to ensure that, in spite of rounding errors, the matrix is still symetric.

It is not enough to ensure that it remains definite positive. More complex update scheme are required for this (for instance by using Cholesky).

NP is $P_{k|k-1}$.

```
// Update covariance
arm_mat_mult_f32(&F,&P,&Temp1_NS_NS);
arm_mat_trans_f32(&F,&Temp2_NS_NS);
arm_mat_mult_f32(&Temp1_NS_NS,&Temp2_NS_NS,&Temp3_NS_NS);

arm_mat_mult_f32(&G,&OmegaErrors,&Temp1_NS_NC);
arm_mat_trans_f32(&G,&Temp2_NC_NS);
arm_mat_mult_f32(&Temp1_NS_NC,&Temp2_NC_NS,&Q);

arm_mat_add_f32(&Temp3_NS_NS,&Q,&NP);
symmetrize(&NP);
```

5.2 Observation

The observation is computed following 3.0.8

We predict the acceleration and magnetic field in the body frame.

```
h_update();
```

The H matrix is computed following equation 3.0.9.

The code is generated from a computer algebra system.

```
H_update();
```

Then we apply the observation step of the Kalman filter following equations 3.0.1

First, the Kalman gain is computed.

```
arm_sub_f32(obs,predictedobs,y,NBOBS);
```

```
// Compute S
// (obs , state) x (state,state) -> (obs, state)
// H . P{k|k-1}
arm_mat_mult_f32(&H,&NP,&Temp4_NO_NS);
// (obs,state) -> (state,obs)
arm_mat_trans_f32(&H,&Temp5_NS_NO);
// (obs,state) x (state,obs) -> (obs,obs)
// H . P{k|k-1} . H^t
arm_mat_mult_f32(&Temp4_NO_NS,&Temp5_NS_NO,&Temp6_NO_NO);
// H . P{k|k-1} . H^t + R
arm_mat_add_f32(&Temp6_NO_NO,&R,&S);


// Compute K
// (state,state) x (state,obs) -> (state,obs)
// P{k|k-1} . H^t
arm_mat_mult_f32(&NP,&Temp5_NS_NO,&Temp7_NS_NO);

arm_mat_inverse_f32(&S,&Temp8_NO_NO);

// (state,obs) x (obs,obs) -> (state,obs)
// P{k|k-1} . H^t . S^-1
arm_mat_mult_f32(&Temp7_NS_NO,&Temp8_NO_NO,&K);
```

The matrix inversion should use the Cholesky algorithm. But it is not (yet) available in CMSIS-DSP. So, the standard Gauss-Jordan algorithm is used. It is not the right way but it is enough for this demo where the simulation is running for a limited number of samples.

The Kalman gain is used to update the state and covariance matrix of the state.

```
// Equation 3.0.4

arm_mat_vec_mult_f32(&K,y,oldstate);
// x{k|k-1}  + K y
arm_add_f32(oldstate,newstate,oldstate,NBSTATE);
```

```
// Update covariance
// Equation 3.0.5
arm_mat_mult_f32(&K, &H,&Temp1_NS_NS);
arm_mat_sub_f32(&Identity_NS_NS,&Temp1_NS_NS,&Temp2_NS_NS);
arm_mat_mult_f32(&Temp2_NS_NS,&NP,&P);

symmetrize(&P);
```

The version 1.9.0 of the CMSIS-DSP must be used otherwise the function *arm_mat_vec_mult_f32* won't be available.

5.3 Limitations

5.3.1 Matrix inversion

It is a critical part of the algorithm. Current solution (Gauss-Jordan) is not the right way. As soon as Cholesky is added to the CMSIS-DSP, this demo will be updated to use it.

5.3.2 Update

We update the Kalman filter as quickly as possible to ensure that the duration dt used for the integration (in the prediction step) is as low as possible.

But the sensors are not refreshing as often. So, we reuse the last measured value when no new value is available.

Different strategies need to be tested : updating only when a new measurement is available and thus accepting to use bigger dt when integrating the angular speed.

Using a low pass filtered version of the sensor values instead of reusing the last measurements (interpolation of the values).

Or with playing with the covariance matrices of the noise : increasing the coefficient related to a specific sensor when no new data is available for this sensor. By doing this, the filter will give less importance to the reused sample.

6 Conclusions

Tuning an extended Kalman filter is difficult. There are several parameters (covariance matrix of noises) which can be modified. The right trade-off is not easy to find.

The sensors need to be calibrated which is not easy.

Some sensors may have defect (biases) which are evolving in time and it should be taken into account.

In summary : This is a proof of concept. More work is required to get a real implementation from this code.