Artikel Auswahl

- STorM32 BGC: 3-Achs STM32 Brushless Gimbal Controller
- Mikro Brushless Gimbal V3.0
- Brushless Gimbal Direct Drive: Theory and Experiment
- IMU Data Fusing: Complementary,Kalman and Mahony Filter
- Fast Computation of Functions on Microcontrollers
- owSilProg, BLHeliTool, BLHeliBox: Tutorials
- STM32: Hello World
- Mini / Micro Gimbal Motors

Schlagwörter

AutoQuad AVR-Projekt boardless brushless CP direct-drive DSM2 EMotor Erfahrung Filter Fun Gimbal Grundlagen Gyro Gyro-Mischer Heli IMU

IMU Data Fusing: Complementary, Kalman, and Mahony Filter

17. Sep. 2013 | Letzte Änderung 16. Jan. 2015 | Thema Grundlagen | 133482 Views | 18 Kommentare

Tag Filter, Grundlagen, Gyro, IMU

An ***inertial measurement unit***, or IMU, measures accelerations and rotation rates, and possibly earth's magnetic field, in order to determine a body's attitude. Anyone who is serious about reading this article is likely familiar with the topic, and the need of data fusing, and I shouldn't spend more words on this. I should however – since this is going to be a long article – spend some words on what this article is about:

The literature on the web (and I should say that the web is my only source of information) on the topic is abundant. It appears however that it is based to a greater or lesser extent on some few works, e.g. by Colton [SC], Premerlani and Bizard [PB], Starlino [St], Lauszus [La], Mahony [RM] and Madgwick [SM], which – so it seems – have become standard references (for hobbiests!). The number of different algorithms and implementation details given there is somewhat confusing, but – even though different buzz words are certainly used – it is not always obvious to what extend they are different. This article presents an analysis and comparison of the data fusing filters described in these works, in order to understand better their behavior, and differences and similarities. As a corollary simplified and/or improved algorithms surface. The article considers 6DOF IMUs only.

Three basic filter approaches are discussed, the ***complementary filter***, the ***Kalman filter*** (with constant matrices), and the ***Mahony&Madgwick filter***. The article starts with some preliminaries, which I find relevant. It then considers the case of a single axis (called one dimensional or 1D). First the most simplest method is discussed, where gyro bias is not estimated (called 1st order). Then gyro bias estimation is included (called 2nd order). Finally, the complete situation of three axes (called 3D) is considered, and some approximations and improvements are evaluated.

Koax Konverter Lipo M4
mCPX Mischer
MultiCopter
Programierung PWM
QuadCopter RC-
Elektronik RC-Sender
Rotor Schaltungstechnik
STM32-Projekt Test
TRex450 Tutorial

Suche

Login

Dashboard

Notation: The discrete time step is denoted as Δt, and n or k is used as time-step index. The estimate of a quantity is indicated by a hat, e.g. $\hat{x}$, which will however often be dropped for simplicity, whenever confusion seems impossible. Bold symbols represent vectors or matrices in $\mathbb{R}^3$ (vectors and matrices in e.g. state space won’t be bold), and quaternions.

A note: Madgwick’s scheme is significantly different in some aspects to Mahony’s but shares his feedback loop idea. Moreover, Madgwick presented C code for Mahony’s filter, which I found very useful. That’s why I denote Mahony’s approach as Mahony&Madgwick filter. Madgwick’s steepest decent scheme will be looked at only very briefly below.

1. Preliminaries

1.1. Notes on Kinematics and IMU Algorithms

The task of attitude estimation corresponds to evaluating (computationally) the kinematic equation for the rotation of a body:

$$\dot{\mathbf{R}} = \mathbf{R}\boldsymbol{\Omega}_\times \qquad \text{with} \qquad \boldsymbol{\Omega}_\times = \boldsymbol{\omega}\mathbf{J} = \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix}, \qquad \text{Eq. (1.1)}$$

where $\boldsymbol{\omega} = (\omega_x, \omega_y, \omega_z)^T$ is the measured rotation rate vector. The rotation $\mathbf{R}$ represents the orientation of the body-fixed reference frame as observed in the earth reference frame. For any vector $\mathbf{v}$, the coordinates $\mathbf{v}_{earth}$ with respect to the earth frame become in the body-fixed frame $\mathbf{v}_{body} = \mathbf{R}^T\mathbf{v}_{earth}$, which evolve as $\dot{\mathbf{v}}_{body} = \boldsymbol{\Omega}_\times^T\mathbf{v}_{body} = -(\boldsymbol{\omega} \times \mathbf{v}_{body})$ (the minus sign comes in here since $\boldsymbol{\omega}$ represents the rotation of the body-fixed coordinate system).

As simple as it may appear, equation Eq. (1.1) presents us with some fundamental issues:

(1) Equation (1.1) is non-linear. This can complicate filter design enormously.

(2) Equation (1.1) is susceptible to numerical errors. Well, numerical errors are present in any calculation performed on a micro processor, but in most cases they are well-behaved in the sense that they do not accumulate. However, for Eq. (1) the errors continuously grow if no counter-measures are taken, and $\mathbf{R}$ eventually ceases to represent a rotation. Importantly, this is related to the global non-commutativity of rotations (in 3 dimensions) and hence is fundamental. It is here where cool buzz words such as *direction cosine matrix* (DCM) or *quaternions* enter the game.

(3) The rotation $\mathbf{R}$ can be represented in several ways [RO], and each representation has its own set of advantages and disadvantages. Most well-known are the representations by a rotation matrix or DCM, Euler angles and related angles (Cardan, Tait-Brian), axis and angle, and quaternions, but some more exist. Obviously, the algorithm will depend a lot on which representation is chosen.

You may note, no words were yet spend on measurement noise and data fusing; I haven't added it to the list since it's not really rooted in Eq. (1.1), although it's of course an important point – and in fact the topic of this article.

Anyhow, since the challenges are alike, all algorithms presented by the above authors exhibit a similar structure:

(T1) Integrate rate of change of the DCM, Euler angles, quaternion, or whatever is used to represent $\mathbf{R}$.
For *(T1)* the DCM, quaternion, and Euler angle representations were used, or – if only the direction was required – the „gravity“ vector $\mathbf{d} = -\mathbf{R}^T\mathbf{e}_z$.

(T2) Take care that the numerical representation of $\mathbf{R}$ ***represents in fact a „real“ physical rotation.***
Several strategies were presented. In [PB] and [St2] the DCM is renormalized by subtracting the errors for the x and y directions to equal parts, in [RM07] the matrix exponential and Rodriguez formulas or alternatively 2nd-order Runge-Kutta is suggested, and in [RM08] and [SM] the common approach of quaternion normalizing is employed. In the MultiWii code (and in [St1]) the drastic approach of neglecting this step altogether is taken; that is, it is left to the data fusing step to ensure the properness of the estimated attitude.

(T3) Improve the attitude estimate by fusing accelerometer and gyroscope data.
This is the crucial (and intelectually most challenging) step, since it decides about the actual quality of the filter in terms of filter performance. The task is to compute from a given attitude $\mathbf{R}$ and measured acceleration vector $\mathbf{a}$ an improved attitude estimate $\mathbf{R}^{improved}$, or formally to identify a function $\mathbf{R}^{improved} = f(\mathbf{R}, \mathbf{a})$. In the above mentioned articles three approaches were taken, which are refereed here to as the complementary filter, Kalman filter, and Mahony&Madgwick filter.

The present article is about task ***T3***; tasks ***T1*** and ***T2*** are not discussed.

1.2. Discretization and Implementation Issues

Well, it’s not big news that for a given system, described by e.g. a continuous-time transfer function $G(s)$, there are many different possible implementations in computer code, and that they do not all show identical performance even though they are derived from one and the same function $G(s)$.

One reason for arriving at different implementations is that there is no universal rule for converting a continuous-time transfer function $G(s)$ to a discrete-time transfer function $H(z)$, since it’s approximative. The conversion from $G(s)$ to $H(z)$ is thus not unique, and typical choices are:

backward difference:	$s = \frac{1}{\Delta t}(1 - z^{-1})$,	Eq. (1.2)

bilinear transformation, expansion of $ln(z)$:	$$s = \frac{2}{\Delta t}\frac{1-z^{-1}}{1+z^{-1}},$$	Eq. (1.3)
impulse invariance transformation:	$$\frac{G(z)}{1-z^{-1}} = Z\left(\frac{G(s)}{s}\right),$$	Eq. (1.4)

Another reason is that a given discrete-time transfer function $H(z)$ can be implemented in different ways, and the different implementations possibly show different behavior in various aspects, such as stability and high-frequency noise. The topic is not simple, and is beyond my competences, but the basic principle is clear.

Let's consider as an example, since it's so familiar, the PID controller $G(s) = K_p + \frac{1}{s}K_i + sK_d$. Using backward difference one finds $H(z) = K_p + \frac{K_i\Delta t}{1-z^{-1}} + \frac{K_d}{\Delta t}(1-z^{-1})$, which can be implemented by first evaluating $I_n = \frac{K_i\Delta t}{1-z^{-1}}x_n$ and then $y_n = K_p x_n + I_n + \frac{K_d}{\Delta t}(1-z^{-1})x_n$, or by directly solving $y_n = H(z)x_n$. In the first case one obtains the ***positional PID algorithm***

$$I_n = I_{n-1} + K_i\Delta t x_n, \qquad \text{Eq. (1.5a)}$$
$$y_n = K_p x_n + I_n + \frac{K_d}{\Delta t}(x_n - x_{n-1}), \qquad \text{Eq. (1.5b)}$$

while the second case leads to the ***velocity PID algorithm***

$$y_n = y_{n-1} + K_p(x_n - x_{n-1}) + K_i\Delta t x_n + \frac{K_d}{\Delta t}(x_n + 2x_{n-1} - x_{n-2}). \qquad \text{Eq. (1.6)}$$

Both derive from the same function $H(z)$, or $G(s)$, but differ e.g. with regards to wind up, overflow of internal variables, number of storage elements and so on.

Corollary #1: *In the positional PID algorithm, Eq. (1.5), the order of the two equations can be reversed,*

$$y_n = K_p x_n + I_{n-1} + \frac{K_d}{\Delta t}(x_n - x_{n-1}), \qquad \textit{Eq. (1.7a)}$$
$$I_n = I_{n-1} + K_i\Delta t x_n, \qquad \textit{Eq. (1.7b)}$$

with an irrelevant change of the parameter $K_p \to K_p + K_i\Delta t$. That is, the order of their execution or implementation in code is irrelevant.

1.3. Kalman Filter with Constant Matrices

The Kalman filter [KA] takes noise into account via covariance matrices, which are updated regularly at each time step using relatively complicated equations. However, if they would be constant with time, then the Kalman filter equations would simplify enormously. I don't know for which conditions exactly these matrices become constant, but intuitively it seems reasonable that they are constant for „usual" systems, e.g., one would not expect the noise spectra of the gyro and accelerometer to vary quickly. Anyhow, in the following they will be assumed to be constant, as well as the system matrices.

The (discrete-time) Kalman filter applies to systems modeled in phase space as

$$x_k = Ax_{k-1} + Bu_k + w, \qquad \text{Eq. (1.8a)}$$
$$z_k = Hx_k + v. \qquad \text{Eq. (1.8b)}$$

Here, x_k ist the state space vector, u_k the control vector, and z_k the measurement vector. I've dropped the time-step index on the other quantities to emphasize their constant spectra. A subtlety occurs here: Sometimes Eq. (1.8a) is written with u_k, but most times with u_{k-1}. In real implementations I would use the latest control vector u_k as only this makes sense to me, but I honestly have no idea whether that's good or bad. The Kalman filter equations are then

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k, \qquad \text{Eq. (1.9a)}$$
$$\hat{x}_k = \hat{x}_k^- + K(z_k - H\hat{x}_k^-), \qquad \text{Eq. (1.9b)}$$

where $\hat{x}_k^-$ is the predicted and $\hat{x}_k$ the updated state estimate.

A general issue with the Kalman filter is that for one and the same system usually several distinct state space models can be set up, and hence distinct Kalman filters derived.

Comment: *In contrast to the situation for the PID controller (Corollary #1) it is not obvious how to reverse the order in Eq. (1.9). In fact, Eq. (1.9) has a clear „philosophy": One first advances the previous estimate $\hat{x}_{k-1}$ according to the systems dynamics, and then applies the correction/improvement due to the latest measurement.*

Comment: *Equation (1.9b) has the looks of feedback with error $z_k - H\hat{x}_k^-$, it can however be reformulated as*

$$\begin{aligned}\hat{x}_k &= (1-KH)\hat{x}_k^- + Kz_k \\ &= (1-KH)A\hat{x}_{k-1} + (1-KH)Bu_k + Kz_k,\end{aligned} \qquad \textit{Eq. (1.10)}$$

and then pretty much looks like a complementary filter. This is an important observation! A Kalman filter with constant matrices and a complementary filter are conceptually similar.

2. 1D IMU Data Fusing – 1st Order (wo Drift Estimation)

In this chapter we will consider the simplest case of IMU data fusing, namely that of fusing the angles for a single axis as determined from the time-integrated rotation rate and accelerometer data, without explicitly estimating the gyro's drift. It has relevance for applications, but here it is of interest mainly as a „warm up", and because it provides some insight which will help us to better understand the more advanced cases below.

In the following, θ denotes the estimated angle (except in Chapter 2.2), a the angle calculated from the accelerometer measurements, and ω the rotation rate measured by the gyro.

2.1. Complementary Filter

The complementary filter fuses the accelerometer and integrated gyro data by passing the former through a 1st-order low pass and the latter through a 1st-order high pass filter and adding the outputs. An excellent discussion of the complementary filter is given in [RM05][RM08], and at a more elementary level in [SC]. The transfer function reads

$$\theta = \frac{1}{1+Ts}a + \frac{Ts}{1+Ts}\frac{1}{s}\omega = \frac{a+T\omega}{1+Ts}, \qquad \text{Eq. (2.1)}$$

where T determines the filter cut-off frequencies. Using backward difference yields $1+Ts = (1+\frac{T}{\Delta t}) - \frac{T}{\Delta t}z^{-1}$. Insertion into Eq. (2.1) and rearrangement leads to our final result

$$\theta_k = \alpha(\theta_{k-1} + \omega_k \Delta t) + (1-\alpha)a_k \qquad \text{Eq. (2.2)}$$

where $\alpha = \frac{T}{\Delta t}/(1+\frac{T}{\Delta t})$. This relation can be implemented in code in several ways; three of them were discussed in [SC] (though only #3 makes sense). For better comparison with the other cases below, the result is reformulated as

$$\theta_k = \alpha\theta_{k-1} + (1-\alpha)a_k + \alpha\omega_k\Delta t. \qquad \text{Eq. (2.3)}$$

2.2. Kalman Filter

In the simple case considered here the state space model of the system is simply

$$\begin{aligned} \theta_k &= \theta_{k-1} + \omega_k\Delta t \\ z_k &= a_k \end{aligned} \qquad \text{Eq. (2.4)}$$

The state vectors become $x = (\theta)$, $u = (\omega)$, and the matrices $A = (1)$, $B = (\Delta t)$, $H = (1)$, $K = (K_0)$. With that the Kalman equations read

$$\begin{aligned} \hat{\theta}_k^- &= \hat{\theta}_{k-1} + \omega_k\Delta t \\ \hat{\theta}_k &= (1-K_0)\hat{\theta}_k^- + K_0 a_k \end{aligned} \qquad \text{Eq. (2.5)}$$

These equations can also be expressed as

$$\hat{\theta}_k = \alpha\hat{\theta}_{k-1} + (1-\alpha)a_k + \alpha\omega_k\Delta t \qquad \text{Eq. (2.6)}$$

with $\alpha = 1 - K_0$.

2.3. Mahony&Madgwick Filter

Here data fusing is done with a P controller and an integrating process, where the „accelerometer“ angle a becomes the setpoint and the rotation rate ω a disturbance (of type d_1). The transfer function is thus

$$\theta = K_p\frac{1}{s}(a-\theta) + \frac{1}{s}\omega, \qquad \text{Eq. (2.7)}$$

where $a - \theta$ is the error input to the P controller. Rearranging this equation for θ yields exactly the transfer function of the complementary filter, Eq. (2.1), which from standard arguments of control theory is however not surprising (see e.g. [RM08]).

The controller Eq. (2.7) is usually implemented by first rearranging it to $\theta = \frac{1}{s}\left[K_p(a-\theta)+\omega\right]$, then separating it into $e = a - \theta$ and $\theta = \frac{1}{s}(K_p e + \omega)$, and finally discretizing it as

$$\begin{aligned} e_k &= a_k - \theta_{k-1} \\ \theta_k &= \theta_{k-1} + (K_p e_k + \omega_k)\Delta t \end{aligned} \qquad \text{Eq. (2.8)}$$

That is, for the Mahony&Madgwick filter one arrives at the update law

$$\theta_k = \alpha\theta_{k-1} + (1-\alpha)a_k + \omega_k \Delta t, \qquad \text{Eq. (2.9)}$$

with $\alpha = 1 - K_p\Delta t$.

It is worthwhile to elaborate a bit further on the controller aspect. For a (simple) controller on finds in general that

$$y = \frac{G_c G_p}{1 + G_c G_p} r + \frac{G_p}{1 + G_c G_p} d_1 + \frac{1}{1 + G_c G_p} d_2, \qquad \text{Eq. (2.10)}$$

where $G_c(s)$ and $G_p(s)$ are the controller and process transfer functions, respectively (the other symbols should be obvious). The conversion of the complementary filter transfer function, Eq. (2.1), into the controller form, Eq. (2.7), is accomplished by multiplying the nominator and denominator in Eq. (2.1) by $1/s$:

$$\theta = \frac{1}{1+Ts} a + \frac{Ts}{1+Ts}\frac{1}{s}\omega = \frac{K_p \frac{1}{s}}{1 + K_p \frac{1}{s}} a + \frac{\frac{1}{s}}{1 + K_p \frac{1}{s}}\omega, \qquad \text{Eq. (2.11)}$$

and identifying $G_c(s) = K_p = 1/T$ and $G_p(s) = 1/s$. Equations (2.1) and (2.7) are identical, but the former is expressed in terms of polynomials in s while the latter is expressed in terms of polynomials in $1/s$.

2.4. Comparison and Conclusions

A couple of observations can be made from the above findings.

(1) The complementary and Kalman filter lead to identical update equations, Eqs. (2.2) and (2.6), and are thus identical, also as regards the transfer functions.

(2) The complementary and Mahony&Madgwick filters are described by identical transfer functions.

(3) From (1) and (2) it follows that all three filters are identical at the level of the transfer function.

(4) Despite (3) the algorithm of the Mahony&Madgwick filter is ***not*** identical to that of the complementary and/or Kalman filter, see Eq. (2.9) and Eqs. (2.2), (2.6).

It is worthwhile to discuss the last point further. For convenience the two update laws are reproduced:

$$\theta_k = \alpha\theta_{k-1} + (1-\alpha)a_k + \alpha\omega_k\Delta t, \qquad \text{Eqs. (2.2),(2.6)}$$
$$\theta_k = \alpha\theta_{k-1} + (1-\alpha)a_k + \omega_k\Delta t. \qquad \text{Eq. (2.9)}$$

One can look at the difference in two ways. Firstly, Eqs. (2.2,6) may be read to say that the angle is first advanced by integrating the rotation rate to give an updated angle and then filtered with a_k to give an improved angle. This may be expressed by the bracketing $\theta_k = \alpha[\theta_{k-1} + \omega_k\Delta t] + (1-\alpha)a_k$. In contrast, Eq. (2.9) may be read to say that the angle is first filtered with a_k and then advanced by integrating the rotation rate, corresponding to the bracketing $\theta_k = [\alpha\theta_{k-1} + (1-\alpha)a_k] + \omega_k\Delta t$. Secondly, the equations can be rearranged into the algorithms

complementary or Kalman filter (1D, 1st order)	Mahony&Madgwick filter (1D, 1st order)
$\theta_k^- = \theta_{k-1} + \omega_k\Delta t$ Eq. (2.12) $e_k^- = a_k - \theta_k^-$ $\theta_k = \theta_k^- + K_0 e_k^-$	$\theta_k^- = \theta_{k-1} + \omega_k\Delta t$ Eq. (2.13) $e_k = a_k - \theta_{k-1}$ $\theta_k = \theta_k^- + K_p\Delta t e_k$

They are essentially identical, except of the important difference that on the left the feedback error uses the updated angle θ_k^- while on the right it uses the previous angle estimate θ_{k-1}! The two algorithms cannot be directly converted into each other, even though they derive from the same transfer function, which should be considered a characteristic feature expressing the different underlying „philosophies“.

3. 1D IMU Data Fusing – 2nd Order (with Drift Estimation)

In this chapter the single-axis filters will be improved by explicitly taking into account the bias/drift of the gyro sensors. To the best of my knowledge, a complementary filter accomplishing this task has not been described before, and hence the order of the discussion of the filters is changed as compared to Chapter 2.

3.1. Kalman Filter

As suggested in [La], the state space model of the system is extended to:

$$\begin{aligned} \theta_k &= \theta_{k-1} + (\omega_k \mp b_{k-1})\Delta t \\ b_k &= b_{k-1} \\ z_k &= a_k \end{aligned} \qquad \text{Eq. (3.1)}$$

where b represents the gyro bias. One would naturally introduce it such that it corrects the measured rotation rate as $\omega_k - b_{k-1}$, i.e. with a „-“ sign in Eq. (3.1). However, we will also allow for a „+“ sign for reasons, which will become clear later on. The state vectors and matrices become

$$x = \begin{pmatrix} \theta \\ b \end{pmatrix}, \quad u = (\omega), \quad A = \begin{pmatrix} 1 & \mp\Delta t \\ 0 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} \Delta t \\ 0 \end{pmatrix}, \quad z = (a), \quad H = \begin{pmatrix} 1 & 0 \end{pmatrix}, \quad K = \begin{pmatrix} K_0 \\ K_1 \end{pmatrix}, \qquad \text{Eq. (3.2)}$$

and the Kalman equations are derived as

$$\begin{aligned} \hat{\theta}_k^- &= \hat{\theta}_{k-1} + (\omega_k \mp \hat{b}_{k-1})\Delta t \\ \hat{\theta}_k &= (1 - K_0)\hat{\theta}_k^- + K_0 a_k \\ \hat{b}_k &= \hat{b}_{k-1} + K_1(a_k - \hat{\theta}_k^-). \end{aligned} \qquad \text{Eq. (3.3)}$$

As before these equations can be reexpressed, yielding the update laws

$$\begin{aligned} \hat{\theta}_k &= \alpha\hat{\theta}_{k-1} + (1-\alpha)a_k + \alpha(\omega_k \mp \hat{b}_{k-1})\Delta t \\ \hat{b}_k &= \hat{b}_{k-1} + K_1(a_k - \hat{\theta}_k^-) \end{aligned} \qquad \text{Eq. (3.4)}$$

with $\alpha = 1 - K_0$. The similarity to the 1st order result, Eq. (2.6), is obvious. The measured rotation rate is corrected

by the estimated bias, which in turn is obtained by integrating the „error“ $a_k - \hat{\theta}_k^-$.

Corollary #2: *The effect of changing the sign in the bias correction to „+“ should be noted. The corrected rotation rate is given as $\omega_k + \hat{b}_{k-1}$, as expected, but the bias update law in Eqs. (3.3) and (3.4) is* ***not*** *altered! The outcome is thus not equivalent to a variable substitution $\hat{b}_k \to -\hat{b}_k$.*

Comment: *One could also start from a state space vector $x = (\theta, \omega)^T$ as suggested by [KA1][KA3][KA4], which however leads to obviously improper filters in our case (thus: it's one thing to derive a Kalman filer but another thing to get one that's working properly ;-)).*

3.2. Mahony&Madgwick Filter

The gyro drift estimation is facilitated by using a PI controller [RM08], and the transfer function is accordingly

$$\theta = \left(K_p + K_i \frac{1}{s}\right) \frac{1}{s}(a - \theta) + \frac{1}{s}\omega, \qquad \text{Eq. (3.5)}$$

Following again the standard implementation (positional PID algorithm) one separates Eq. (3.5) into $e = a - \theta$, $I = K_i \frac{1}{s} e$, and $\theta = \frac{1}{s}(K_p e + I + \omega)$, and discretizes it as

$$\begin{aligned} e_k &= a_k - \theta_{k-1} \\ I_k &= I_{k-1} + K_i \Delta t e_k \\ \theta_k &= \theta_{k-1} + (K_p e_k + I_k + \omega_k)\Delta t \end{aligned} \qquad \text{Eq. (3.6)}$$

This results in the update laws

$$\begin{aligned} I_k &= I_{k-1} + K_i \Delta t (a_k - \theta_{k-1}) \\ \theta_k &= \alpha \theta_{k-1} + (1 - \alpha) a_k + (\omega_k + I_k)\Delta t \end{aligned} \qquad \text{Eq. (3.7)}$$

with $\alpha = 1 - K_p \Delta t$.

Here Corollary #1 is recalled, which tells that the sequence of the two equations in Eq. (3.7) can be reversed (with an insignificant parameter change).

3.3. Comparison

Comparing the update laws for θ_k for the 2nd order Kalman and Mahony&Madgwick filters shows the same aspects discussed in Chapter 2.4. This shall be further emphasized by contrasting again the rearranged algorithms:

Kalman filter (1D, 2nd order)		Mahony&Madgwick filter (1D, 2nd order)	
$\theta_k^- = \theta_{k-1} + (\omega_k + b_{k-1})\Delta t$ $e_k^- = a_k - \theta_k^-$ $\theta_k = \theta_k^- + K_0 e_k^-$ $b_k = b_{k-1} + K_1 e_k^-$	Eq. (3.8)	$\theta_k^- = \theta_{k-1} + (\omega_k + I_{k-1})\Delta t$ $e_k = a_k - \theta_{k-1}$ $\theta_k = \theta_k^- + K_p' \Delta t e_k$ $I_k = I_{k-1} + K_i \Delta t e_k$	Eq. (3.9)

It should be noted that in order to arrive at these equations the sign in the bias estimation in the Kalman filter was changed to „+“ (Corollary #2), and the order of equations in the Mahony&Madgwick filter was reversed (Corollary #1) and $K_p' = K_p + K_i \Delta t$ was introduced.

The two update laws are essentially identical, except of the important difference that the Kalman filter uses the updated angle θ_k^- in the error while the Mahony&Madgwick filter uses the previous angle estimate θ_{k-1}, as observed before for the 1st order filters (Chapter 2.4).

3.4. Complementary Filter

A complementary filter is easily derived by solving the transfer function of the Mahony&Madgwick filter for the angle θ, which yields

$$\theta = \frac{1 + \frac{K_p}{K_i} s}{1 + \frac{K_p}{K_i} s + \frac{1}{K_i} s^2} a + \frac{\frac{1}{K_i} s^2}{1 + \frac{K_p}{K_i} s + \frac{1}{K_i} s^2} \frac{1}{s} \omega. \qquad \text{Eq. (3.10)}$$

Obviously, and not unexpectedly, this complementary filter is build from 2nd order filters. Note that the filter acting on the acceleration data actually consists of a low-pass plus band-pass filter.

This result has interesting consequences. Being 2nd order filters, the frequency response of the acceleration and rotation rate filters are characterized by the resonance frequency and damping factor

$$\omega_0 = \sqrt{K_i}, \qquad \xi = \frac{K_p}{2\sqrt{K_i}}. \qquad \text{Eq. (3.11)}$$

The damping factor determines the overshoot at the resonance frequency. For high-pass (and low-pass) filters the frequency response is flat (and the step response non-oscillatory) for $\xi \geq 1$. This suggests the criterion

$$K_i \leq \frac{1}{4} K_p^2 \qquad \text{Eq. (3.12)}$$

in order to avoid overshoot in the gyro channel. In order to minimize also overshoot in the accelerometer channel, the damping should be somewhat larger than that; $\xi \approx 2$ might be a good compromise between smooth frequency response and fast bias estimation. Accordingly, as a rule of thumb $K_i \approx 0.05 \ldots 0.1 \, K_p^2$.

Note that – unless K_i is set to very small values – the crossover frequency is determined now by K_i (or the inverse square root of it), and not by K_p as in the 1st order case! It could in fact be appropriate to use the parameters ω_0 or $T = 2\pi/\sqrt{K_i}$ and ξ instead of K_p and K_i; the tuning of the filter might be more intuitive to the user.

These considerations obviously also apply to the Mahony&Madgwick filter, and the Kalman filter.

The complementary filter may be implemented as in Eq. (3.6), or with any of the algorithms used with advantage for digital filters. The direct form II would be a typical choice (see e.g. here).

3.5. Summary on 1D Filters

As lengthy as it was, the above detailed discussion in chapters 2 and 3 of different approaches to the data fusing for a single axis result in a very short summary:

The three considered different approaches are in fact not that different, even in the 2nd order case.

As a bonus a criterion for the choice of the bias estimator gain K_i or K_1, respectively, has been obtained, as well as a potentially easier and/or more flexible direct implementation as a complementary filter.

There is a difference between the Kalman and Mahony&Madgwick filters in how the error is calculated. This may be interpreted as conceptually different „philosophies“, but besides that it’s not clear to me if this has also practical consequences, such as different stability properties or high-frequency noise. (Does anyone know?)

4. 3D IMU Data Fusing with Mahony Filter

In this chapter we will discuss data fusing filters for the three-axis problem, which involve the full non-linear kinematic equation, Eq. (1.1). I have seen various Kalman filters which were designed for this case, but because of the non-linearity they are much more complicated, and require much more computational power (which is potentially beyond reach for standard hobby micro controllers). They go beyond the scope of this articles (and my competences), and are thus not further considered. This leaves us with Mahony's *„complementary filter on SO(3)"* [RM05].

When thinking about how to merge gyro and accelerometer data it becomes quickly clear that the problem lies in the different tensorial nature of the rotation rate vector $\boldsymbol{\omega}$ and acceleration vector $\mathbf{a}$, and the fact that $\mathbf{a}$ describes only a direction. Mahony described an elegant way to cope with both.

4.1. „Original" Mahony Filter

Mahony's idea was to correct the input to the kinematic equation Eq. (1.1), which is the rotation rate vector $\boldsymbol{\omega}$, by some correction vector $\delta\boldsymbol{\omega}$, which is provided by a PI controller, where the error vector $\mathbf{e}$ driving the PI controller is determined by some means from the previously estimated attitude and the accelerometer vector $\mathbf{a}$. Mahony suggested to use $\mathbf{e} = \mathbf{a} \times \mathbf{d}$, where $\mathbf{d}$ is the direction of the gravity vector as given by the estimated attitude. Mahony's algorithm [RM08][SM2], reads

$$\begin{aligned} \boldsymbol{\omega}' &= \boldsymbol{\omega} + \left(K_p + K_i \frac{1}{s} \right) \mathbf{a} \times \mathbf{d} \\ \mathbf{R} &= \frac{1}{s} \mathbf{R} \boldsymbol{\Omega}'_{\times} \end{aligned} \quad . \qquad \text{Eq. (4.1)}$$

or, as an algorithmic template:

1. measure $\boldsymbol{\omega}$ and $\mathbf{a}$ (optionally normalize $\mathbf{a}$)
2. determine gravity vector $\mathbf{d}$ from the current attitude estimate (in the DCM representation $\mathbf{d} = -\mathbf{R}^T \mathbf{e}_z$)
3. calculate error vector $\mathbf{e} = \mathbf{a} \times \mathbf{d}$
4. apply PI controller to get rotation rate correction; $\delta\boldsymbol{\omega} = (K_p + K_i \frac{1}{s})\mathbf{e}$

5. calculate corrected rotation rate $\boldsymbol{\omega}' = \boldsymbol{\omega} + \delta\boldsymbol{\omega}$
6. integrate rate of change using $\boldsymbol{\omega}'$ (in the DCM representation $\dot{\mathbf{R}} = \mathbf{R}\boldsymbol{\Omega}'_{\times}$)
7. repeat with step 1

It should be noted that steps 2 and 6 can readily be expressed in terms of quaternions, and hence tasks *T1* and *T2* be elegantly accomplished [RO1]. The algorithm in quaternion form is given in the Appendix.

Step 4 could be implemented in code as $\mathbf{I}_n = \mathbf{I}_{n-1} + K_i \Delta t \mathbf{e}_n$ and $\delta\boldsymbol{\omega}_n = K_p \mathbf{e}_n + \mathbf{I}_n$. Step 6 would be implemented in the DCM representation as $\mathbf{R}_n = \mathbf{R}_{n-1} + \mathbf{R}_{n-1}\boldsymbol{\Omega}'_{\times}\Delta t$.

As regards the proper tuning of the filter gains K_p and K_i not much is said in the original literature. However, although I can't „proof" this, it seems reasonable that the conclusion inferred in Chapter 3.4 for the single-axis case applly also to the three-axis case.

4.2. 2D Mahony Filter and Simplifications

In many practical cases one doesn't need to know the attitude including yaw, but the direction of the gravity vector is sufficient. In this case the calculations can be significantly simplified.

The quantity of interest is then $\mathbf{d} = -\mathbf{R}^T \mathbf{e}_z$; the full rotation matrix $\mathbf{R}$ itself is not needed. The kinematic equation implies for $\mathbf{d}$ the dynamics

$$\dot{\mathbf{d}} = \boldsymbol{\Omega}'^T_{\times}\mathbf{d} = \mathbf{d} \times \boldsymbol{\omega}' \qquad \text{Eq. (4.2)}$$

where in the second step it was used that $\boldsymbol{\Omega}_{\times}$ is a skew-symmetric matrix, and a prime was added to the rotation rate in anticipation of Mahony's algorithm. The rate of change is hence given by $\mathbf{d} \times \boldsymbol{\omega}' = \mathbf{d} \times \boldsymbol{\omega} + \mathbf{d} \times \delta\boldsymbol{\omega}$, where thanks to linearity the correction vector is obtained by applying the controller to the error $\mathbf{E} = \mathbf{d} \times (\mathbf{a} \times \mathbf{d})$. The Mahony filter simplifies to

$$\dot{\mathbf{d}} = \mathbf{d} \times \boldsymbol{\omega} + (K_p + K_i \tfrac{1}{s})\mathbf{E}. \qquad \text{Eq. (4.3)}$$

The Mahony filter reduces to applying the single-axis filter of Chapter 3 to each of the three axes independently. It can further be simplified by noting that $\mathbf{d} \times (\mathbf{a} \times \mathbf{d}) = \mathbf{a}d^2 - \mathbf{d}(\mathbf{a} \cdot \mathbf{d})$ and that both $\mathbf{d}$ and $\mathbf{a}$ can be normalized. In addition one can assume that $\mathbf{d}$ is a good estimate of $\mathbf{a}$. Thus, $\mathbf{a} \cdot \mathbf{d} \approx 1$ and $\mathbf{E} = \mathbf{a} - \mathbf{d}$. The Mahony filter

becomes

$$\mathbf{d} = \frac{1}{s}\mathbf{d} \times \boldsymbol{\omega} + \left(K_p + K_i\frac{1}{s}\right)\frac{1}{s}(\mathbf{a} - \mathbf{d}). \qquad \text{Eq. (4.4)}$$

The striking similarity to the single-axis case, Eqs. (3.5)-(3.7), is obvious.

The result is useful. First, the overall simplification in terms of code size and speed is quite significant. One first calculates $\mathbf{d} \times \boldsymbol{\omega}$ and then applies the 2nd order single-axis filter separately to each coordinate. Task *T2* cannot easily be achieved, and the MultiWii approach suggests itself here. Having settled on the MultiWii approach, this in turn however means, that Mahony's filter can be implemented by simply extending the 1st order filters used there to 2nd order filters.

As a little exercise we briefly calculate the low-angle approximation to Eq. (4.4), which holds when the body is nearly horizontally oriented (within $\pm 30^\circ$ or so). Then one may assume $d_z \approx -1$, and the pitch and roll angles are approximately $\theta_x \approx d_x$ and $\theta_y \approx -d_y$. One obtains

$$\dot{\theta}_x = \omega_y + \theta_y\omega_z + (K_p + K_i\tfrac{1}{s})(a_x - \theta_x), \qquad \dot{\theta}_y = \omega_x - \theta_x\omega_z + (K_p + K_i\tfrac{1}{s})(a_y - \theta_y). \qquad \text{Eq. (4.5)}$$

There is a cross coupling to the yaw rate ω_z. This is exactly the result found by Shane [SC]. Furthermore, the single-axis filter equations apply separately to each angle.

4.3. Premerlani & Bizard's IMU Filter

Premerlani and Bizard described in great detailed an IMU algorithm in [PB], which was later taken up also by Starlino [ST2] (providing also C code).

Premerlani and Bizard followed Mahony's algorithm very closely, with one exception. Mahony presented the algorithm for both the DCM and quaternion representations [RM05] (the discussion in the above uses DCM language, Madgwick's C code which is reproduced below in the Appendix uses quaternions). In the DCM version Mahony suggested e.g. using matrix exponentials and Rodriguez' formula to handle task *T2* [RM07]. Premerlani and Bizard used Mahony's algorithm in the DCM form, but introduced a simpler approach for task *T2* or the reorthogonalization of the DCM. With the DCM written as

$$\mathbf{R}^T = \left(\mathbf{r}_x \, \mathbf{r}_y \, \mathbf{r}_z \right)$$

it goes as follows (where I incorporated an obvious improvement suggested in [ST2])

1. calculate a correction magnitude $\delta r = \frac{1}{2} \mathbf{r}_x \mathbf{r}_y$
2. correct the x and y column vectors in the DCM as
   $\mathbf{r}'_x = \mathbf{r}_x - \delta r \, \mathbf{r}_y$
   $\mathbf{r}'_y = \mathbf{r}_y - \delta r \, \mathbf{r}_x$
3. normalize these column vectors, using a simplified Taylor expansion based method: $\mathbf{r}''_\alpha = \frac{1}{2} \left(3 - |\mathbf{r}'_\alpha|^2 \right) \mathbf{r}'_\alpha$
4. calculate a new z column vector as $\mathbf{r}''_z = \mathbf{r}''_x \times \mathbf{r}''_y$
5. the reorthogonalized DCM is then $\mathbf{R}^T = \left(\mathbf{r}''_x \, \mathbf{r}''_y \, \mathbf{r}''_z \right)$

Comment: *The coordinate vector $\mathbf{r}_i$ (with $i = x, y, z$) represents the coordinates of the i-th basis vector of the earth frame as measured in the body-fixed frame.*

5. Further 3D Filters

Madgwick's IMU Filter

Madgwick has presented an interesting approach, which is based on formulating task *T3* as a minimization problem and solving it with a gradient technique [SM]. I will argue here that this approach is – IMHO – not appropriate for IMUs which are using only gyro and accelerometer data (6DOF IMU).

Madgwick uses a quaternion approach to represent the attitude, which immediately poses the problem of how to convert the measured acceleration vector $\mathbf{a}$ into a quaternion. Madgwick has described the problem in clear detail: A body's attitude (quaternion) cannot be unambiguously represented by a direction (vector) since any rotation of the body around that direction gives the same vector but a different quaternion. The solution manifold is a „line" and not a „point". Or plainly: The body's yaw angle is totally undetermined.

To tackle this problem he suggested to determine that rotation, which brings the gravity vector $\mathbf{g}^{earth} = -\mathbf{e}_z$ in the

earth frame in coincidence with the measured acceleration $\mathbf{g}^{body} = \mathbf{a}$ in the body frame, that is to find the rotation $\mathbf{R}_a$ for which $\mathbf{a} = -\mathbf{R}_a^T \mathbf{e}_z$, or the quaternion $\mathbf{q}_a$ for which $\mathbf{a} = -\mathbf{q}_a^{-1} \mathbf{e}_z \mathbf{q}_a$, respectively. Converting the measured vector to a quaternion is very desirable since then data fusing could be done directly on quaternions, which has favorable mathematical properties [RO2]. In order to determine this rotation computationally, Madgwick suggested to formulate it as minimization problem and to solve it iteratively by the method of steepest decent.

This approach has two problems. The exact solution is not unique but there are infinitely many, and not the exact solution is calculated. One can hence expect that the yaw angle in the computed attitude $\mathbf{q}_a$ is not only arbitrary, but determined by the noise introduced by the incomplete steepest decent. The yaw angle fluctuates.

At this point one could analyze the algorithm (reproduced below) by asking: Let's assume that our gyro and accelerometer data is perfect and exact, what is then the algorithm doing? Clearly, one would expect the algorithm to produce the exact attitude, and the data fusing filter not to introduce any corrections. For the filters described in the above this is obviously fulfilled, e.g. in Mahony's filter the error vector is zero, $\mathbf{e} = 0$. In Madgwick's filter, the correction step $\delta \mathbf{s}$ is however not zero (since $\mathbf{a} = -\mathbf{q}^{-1} \mathbf{e}_z \mathbf{q}$). That is, the filter in fact pushes the estimated attitude away from the correct attitude. In my opinion this is a signature of the noise mentioned before.

Due to these reasons I believe that Madgwick's approach is not appropriate for implementing a 6DOF IMU.

It has to be stressed however that the comments do NOT apply to the case, where accelerometer and magnetometer data are used (9DOF IMU). This case is fundamentally different since the measured data span a two-dimensional sub space and hence allow to determine a unique, unambiguous attitude. Madgwick's algorithm could in fact be an excellent choice for this case (but I can't tell).

Appendix: References

[SC] Fun with the Complementary Filter [link] and The Balance Filter (Jun. '07) [.pdf] – by Shane Colton
[PB] Direction Cosine Matrix IMU: Theory (May '09) [.pdf] – by William Premerlani, Paul Bizard,
see also the google repository gentlenav [link] (it hosts also some of Mahony's papers)
[St1] A Guide To using IMU (Accelerometer and Gyroscope Devices) in Embedded Applications (Dez. '09) [link] – by Starlino

[St2] DCM Tutorial – An Introduction to Orientation Kinematics (May '11) [link] – by Starlino (or as [.pdf])
[La] A practical approach to Kalman filter and how to implement it (Sep. '12) [link] – by Lauszus, TKJ Electronics

Mahony's papers:
[RM05] Complementary filter design on the special orthogonal group SO(3) (Dec. '05) [.pdf] – by Robert Mahony, Tarek Hamel, Jean-Michel Pflimlin
[RM07] Complementary filter design on the Special Euclidean group SO(3) ('07) [.pdf] – by Grant Baldwin, Robert Mahony, Jochen Trumpf, Tarek Hamel, Thibault Cheviron
[RM08] Nonlinear Complementary Filters on the Special Orthogonal Group (Jun. '08) [link] – by Robert Mahony, Tarek Hamel, Jean-Michel Pflimlin (it is also hosted on gentlenav)

Madgwick's report and codes:
[SM1] An efficient orientation filter for inertial and inertial/magnetic sensor arrays (Apr.'10) [.pdf] – by Sebastian Madgwick (internal report on his thesis and MARG)
[SM2] Codes and Resources: Open source IMU and AHRS algorithms [link] (original repository imumargalgorithm30042010sohm)

Kalman filter:
[KA1] Kalman Filtering (June '01) – by Dan Simon
[KA2] An Introduction to the Kalman Filter – by Greg Welch, Gary Bishop (or here)
[KA3] Understanding the Basis of the Kalman Filter Via a Simple and Intuitive Derivation (Sep. '12) – by Ramsey Faragher
[KA4] What is the Kalman Filter and How can it be used for Data Fusion? (Dec. '05) – by Sandra Mau (*Note: This ref should be considered with caution, I added it because the first two presented filters are of pedagogical value, but otherwise the work shouldn't be taken seriously*)

Rotation representations:
[RO1] Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors – by James Diebel *(excellent!)*
[RO2] Rotation Representations and Performance Issues – by David Eberly
[RO3] Rotation formalisms in three dimensions [link] – Wikipedia
[RO4] on Euler, Tait-Bryan and Cardan angles see Euler Angles [link] – Wikipedia
[RO5] Application of Quaternions to Computation with Rotations – by Eugene Salamin

Miscellaneous:
[LTB] Other MARG/AHRS/IMU/INS Open Code Projects – by Lewis De Payne (Lew's Tech Blog)
[SHO] Quaternions – by Ken Shoemake
[WTH] A Comparison of Complementary and Kalman Filtering – by Walter T. Higgins

Appendix: IMU Implementations

MultiWii 2.2 (code: MultiWii_2_2.zip)

1. measure $\boldsymbol{\omega}$ and $\mathbf{a}$
2. normalize $\mathbf{a}$
3. integrate rate of change for estimated gravity vector using $\dot{\mathbf{d}} = \mathbf{d} \times \boldsymbol{\omega}$

   $$\mathbf{d}_n = \mathbf{d}_{n-1} + \mathbf{d}_{n-1} \times \boldsymbol{\omega}\Delta t = \mathbf{d}_{n-1} + \begin{pmatrix} d_y\omega_z - d_z\omega_y \\ d_z\omega_x - d_x\omega_z \\ d_x\omega_y - d_y\omega_x \end{pmatrix} \Delta t$$

   Note: d_x ist defined negative in the code
4. apply complementary filter if $|\mathbf{a}| - 1 < 0.15$
   Note: not sure if there is an error in the implementation, seems so, but the intention is clear
5. calculate angles

   $$\theta_{pitch} = \operatorname{atan2}\left(\frac{d_x}{d_z}\right)$$

   $$\theta_{roll} = \operatorname{atan2}\left(\frac{d_y}{\sqrt{d_x^2 + d_y^2}}\right)$$
6. repeat with step 1

In my opinion the key point in the MultiWii code is that there is no explicit effort to ensure that the attitude estimate is SO(3) compatible (e.g. by reorthogonalizing a DCM or normalizing a quaternion). Hence, task *T2* to correct for errors in the rate integration due to the gyro's imperfections is entirely left to the effectiveness of the complementary filter; there is no explicit effort.

KK

I must admit that I never really looked into the KK code – it's ASM and I always had something better to do with my time… however, there is enough information on the web to piece things together. If it's incorrect what I'm saying, then … well, then it doesn't matter, then „KK" is just my label to denote this possible algorithm:

$$\theta_{pitch,n} = \theta_{pitch,n-1} + (\omega_{pitch} + \theta_{roll,n-1}\omega_{yaw})\,\Delta t + K_p(a_{pitch} - \theta_{pitch,n-1})$$

$$\theta_{roll,n} = \theta_{roll,n-1} + (\omega_{roll} - \theta_{pitch,n-1}\omega_{yaw})\,\Delta t + K_p(a_{roll} - \theta_{roll,n-1})$$

In my opinion the characteristic feature of this code is that the linear approximation, as described in Chapter 4.2, Eq. (4.5), is implemented, together with a simple complementary filter for each axis.

Mahony's Filter in Quaternion Form as Implemented by Madgwick (code: madgwick_algorithm_c.zip)

1. measure $\boldsymbol{\omega}$ and $\mathbf{a}$
2. normalize $\mathbf{a}$
3. get estimated gravity vector $\mathbf{d}$ from quaternion $\mathbf{q}$ (representing $\mathbf{R}$)
   $$\mathbf{d} = \mathrm{Im}\left[\mathbf{q}^{-1}\mathbf{e}_z\mathbf{q}\right] = 2\begin{pmatrix} q_1q_3 - q_0q_2 \\ q_0q_1 + q_2q_3 \\ q_0^2 + q_3^2 - \frac{1}{2} \end{pmatrix}$$
4. calculate error vector $\mathbf{e}$
   $\mathbf{e} = \mathbf{a} \times \mathbf{d}$
5. calculate I term (which is a vector too)
   $\mathbf{I}_n = \mathbf{I}_{n-1} + K_i\Delta t\mathbf{e}$
6. calculate P term and add everything together
   $\boldsymbol{\omega}' = \boldsymbol{\omega} + K_p\mathbf{e} + \mathbf{I}_n$
7. integrate rate of change using $\dot{\mathbf{q}} = \frac{1}{2}\mathbf{q} \otimes \boldsymbol{\omega}'$
   $$\mathbf{q}_n = \mathbf{q}_{n-1} + \tfrac{1}{2}\mathbf{q}_{n-1} \otimes \boldsymbol{\omega}'\Delta t = \mathbf{q}_{n-1} + \frac{1}{2}\begin{pmatrix} -q_1\omega'_x - q_2\omega'_y - q_3\omega'_z \\ q_0\omega'_x + q_2\omega'_z - q_3\omega'_y \\ q_0\omega'_y - q_1\omega'_z + q_3\omega'_x \\ q_0\omega'_z + q_1\omega'_y - q_2\omega'_x \end{pmatrix}\Delta t$$
8. normalize quaternion $\mathbf{q}$
9. repeat with step 1

Madgwick's 6DOF IMU Filter (wo gyro drift correction) (code: madgwick_algorithm_c.zip)

1. measure $\boldsymbol{\omega}$ and $\mathbf{a}$
2. normalize $\mathbf{a}$
3. calculate rate of change $\delta\mathbf{q}$

$$\delta\mathbf{q} = \tfrac{1}{2}\mathbf{q} \otimes \boldsymbol{\omega} = \frac{1}{2}\begin{pmatrix} -q_1\omega_x - q_2\omega_y - q_3\omega_z \\ q_0\omega_x + q_2\omega_z - q_3\omega_y \\ q_0\omega_y - q_1\omega_z + q_3\omega_x \\ q_0\omega_z + q_1\omega_y - q_2\omega_x \end{pmatrix}$$

4. calculate corrective step $\delta\mathbf{s}$

$$\delta\mathbf{s} = \begin{pmatrix} 4q_0q_2^2 + 2q_2a_x + 4q_0q_1^2 - 2q_1a_y \\ 4q_1q_3^2 - 2q_3a_x + 4q_0^2q_1 - 2q_0a_y - 4q_1 + 8q_1^3 + 8q_1q_2^2 + 4q_1a_z \\ 4q_0^2q_2 + 2q_0a_x + 4q_2q_3^2 - 2q_3a_y - 4q_2 + 8q_2q_1^2 + 8q_2^3 + 4q_2a_z \\ 4q_1^2q_3 - 2q_1a_x + 4q_2^2q_3 - 2q_2a_y \end{pmatrix}$$

5. normalize $\delta\mathbf{s}$
6. add P term

   $\delta\mathbf{q}' = \delta\mathbf{q} - \beta\delta\mathbf{s}$
7. integrate rate of change using $\dot{\mathbf{q}} = \delta\mathbf{q}'$

   $\mathbf{q}_n = \mathbf{q}_{n-1} + \delta\mathbf{q}'\Delta t$
8. normalize quaternion $\mathbf{q}$
9. repeat with step 1

Note: Since in step 4 the quaternion is normalized, the calculation can be enormously simplified to:

$$\delta\mathbf{s} = 4(q_1^2 + q_2^2)\mathbf{q} + 2\begin{pmatrix} q_2a_x - q_1a_y \\ -q_3a_x - q_0a_y + 2q_1a_z \\ +q_0a_x - q_3a_y + 2q_2a_z \\ -q_1a_x - q_2a_y \end{pmatrix}$$

---

18 Kommentare

Pingback: *RM10 UAV 22/07/2016 – IMU Design | Gigawolf*

matteo *sagt:*

6. Juni 2016 um 15:27

Hello Olli,

the article is great and it create clarity in a world of papers mathematics.

I would like to trigger a reasoning here. Imagine that with tactical grade IMUs one obtains the necessary performance in the state estimates with just integrating the gyros or at most a linear complementary filter using the current vs accelerometer-derived state estimation to generate the error term to compensate the gyros before integrating them. On the other side, for low cost IMUs, one needs to be more clever with the state estimation if the performance requirements are still stringent and safety is involved i.e. don't want a UAV to crash onto people. To this purpose, assuming one wants to move to cheaper industrial grade IMUs but still maintaining state estimation algorithms which are deterministic, which algorithms in your opinion could give acceptable results? Assume only pitch and roll necessary to be estimated.

Another, separate question is: would you comment on the case whether the Mahony formulation with respect to the formulation of my reasoning above may allow cheaper sensors to be used and still similar performance?

Let me know what you think.

Antworten

Peter *sagt:*

6. April 2016 um 15:05

Hi Olli, thanks for your very good article! I have one question about Eq. 1.1: Why is it non-linear? If I do the matrix multiplication, I get:
r_11′ = r_12 * w_z – r_13 * w_y
r_12′ = – r_11 * w_z + r_13 * w_y

…
r_33′ = r_31 * w_y – r_32 * w_x
For me, these are ordinary linear inhomogenous DEs with non-constant (wrt. time) coefficients. Am I getting something wrong here? Thanks for clarification. Peter

Antworten

OlliW *sagt:*

6. April 2016 um 19:01

the non-constant (wrt. time) coefficients make them non linear
e.g. assume a exp(-iwt) time dependence of the quantities, you will get exp(-iwt) and exp(-i2wt) terms, which do not cancel out, as for linear DEs

Pietro *sagt:*

8. Februar 2016 um 22:06

Hi Olli,
thanks for the great post and explanations. I've downloaded the matlab implementation of Madgwick filter… the problem is that I have a triaxial accelerometer and triaxial gyroscope so no magnetometer…and I have no idea how to exclude (if this is possible) magnetometer data from Madgwick code. Do you have any suggestions? Please consider that unfortunately you are writing to a „just press the button" user 🙂 with basic Matlab programming capabilities.
Thansk for your time,
Cheers,
Pietro

Antworten

le mat *sagt:*

11. Mai 2016 um 15:18

Hi Pietro,

I have just read your post about Madgwick code and I am a bit like you „just press the button“ user 🙂

I am trying to use the code with my data, I have magnetometer data (but I can delete it if needed).

I am not able to use to code, it does not give me a code movement I guess.

Did you finally succeed it ?

If yes, what did you change in the code ? Would you have an example ?

Thank you so much !

Yann

ELAHE ALAVI *sagt:*

4. Juli 2014 um 09:22

HI

can i use cubature kalman filter or particle filter instead of extended kalman filter.

Antworten

Florian Augustin *sagt:*

21. März 2014 um 23:03

It’s clear that this is a delay element of one deltat in this case. But I just cannot get here the theta (angle) comes from, just searching any intagration of the omega(angular velocity). But all I can achieve is getting a (z)/(z-1) which would be 1 in time domain. I thought about integration in Laplace would be 1/s and then taking this in z-domain but I’m really unsure about this.

Antworten

Florian Augustin *sagt:*

22. März 2014 um 01:39

Solved – sorry for the dumbass questions, it was really so easy…

Florian Augustin *sagt:*

21. März 2014 um 21:45

Hello,
I'm trying hard to get from Eq. (2.1) to Eq. (2.2). Where does the angle of the last cycle come from and whats with the z^(-1)? Perhaps someone could spend some words on this, this would really be great!
Thanks!

Antworten

OlliW *sagt:*

21. März 2014 um 21:58

come one, search on the net for the z transformation and get aquainted first with what z^-1 means, sorry to say that, but this is elementary textbook knowledge 🙂

Rud Merriam *sagt:*

4. Februar 2014 um 00:25

Hi Olli,

I really appreciate the work you did for this page. It clarified the sensor fusion techniques considerably for me.

I have been working to combine the data from a sensor without success using the Mahony quaternion code as modified by Madgwick. Maybe you can help me by answering a couple of questions – a least to start?

Is the output quaternion the fusion of the sensors in the global frame? I ask because my output does not look at all like this so felt it best to check.

In which frame is the gravity vector (bold d)? I am trying to understand what it represents for my debugging.

Are the angular x, y, and z and the acceleration x, y, and z values represent the same motion on the axis? I ask because with my sensor the rotation reported around the y-axis corresponds to the acceleration change for the x-axis. I am uncertain whether this is typical and accounted for by the math or if I need to swap the values to get good results.

I appreciate any help,
Rud

Antworten

OlliW *sagt:*

4. Februar 2014 um 00:49

>Is the output quaternion the fusion of the sensors in the global frame?

the quaternion describes the rotation by which the body and global (earth) frames are related, so it's neiter in the global nor the body frame, it links the two

>In which frame is the gravity vector (bold d)?

body frame [in the earth frame the gravity vector would be just -e_z = (0,0,-1)]

>Are the angular x, y, and z and the acceleration x, y, and z values represent the same motion on the axis?

I find this language a bit unclear, so I can only guess in my answer: the angular x represents the rotation rate around the x axis and the acceleration x represents the acceleration along the x axis (note the „around" and „along", they hence don't describe the same motion)
in short, if the x axis of your gyro concides with the x axis of your accelerometer (and similar for

y,z) then you're fine
in case of doubts you find an detailed description for aligning the coordinate axes in [St1] (Part 3)

good luck, Olli

le mat *sagt:*

11. Mai 2016 um 15:22

Hi Rud,

I am also using without any success the Mahony quaternion code as modified by Madgwick.
I undersood pretty well you comment as I am feeling the same thing. The axis of accelerometer data and gyroscope does not match I guess. I am trying to figure out how the motion is running..
Did you finally succeed to use the code ?
Did you change something about the axis ?
I know it's been a long time since your comment, but it's been few days I am working on this code and I don't have the result I am expected.

If you could help me it would be perfect !

Thank you by advance.

Yann

Vincent *sagt:*

21. Januar 2014 um 13:30

Hallo Olli

Sehr schöne Zusammenfassung!

Hast du dir den Matlab/C-Code von der Mahony Implementierung, wo mit den Kreuzproduktern herumgerechnet wird, eventuell auch schonmal angesehn? Da gibts nämlich auch eine Magnetometer-Implementierung

Die folgendermaßen aussieht:

```
% Reference direction of Earth's magnetic field
h = quaternProd(q, quaternProd([0 Magnetometer], quaternConj(q)));
b = [0 norm([h(2) h(3)]) 0 h(4)];

% Estimated direction of magnetic field
w = [2*b(2)*(0.5 – q(3)^2 – q(4)^2) + 2*b(4)*(q(2)*q(4) – q(1)*q(3))
2*b(2)*(q(2)*q(3) – q(1)*q(4)) + 2*b(4)*(q(1)*q(2) + q(3)*q(4))
2*b(2)*(q(1)*q(3) + q(2)*q(4)) + 2*b(4)*(0.5 – q(2)^2 – q(3)^2)];
```

Ich verstehe diesen Teil leider nicht komplett. Könntest du mir vielleicht erklären wieso hier X/Y Komponente des Magnetometers normalisiert wird und was genau w dann ist?
Ich denk mal, dass ähnlich wie bei der Gravitation ein Teil der Rotationsmatrix ausgerechnet wird?
Was bei dem von dir beschriebenen Algorithmus unter Punkt 3.) geschieht, ist ja auch nichts anderes als die letzte Spalte der aktuellen Rotationsmatrix mit invertierter X und Y Komponente, sprich eben der geschätzte Beschleunigungsvektor aus der aktuellen Quaternion-Raumlage…

Aber wozu beim Magnetometer X und Y normalisiert wird… und dann dieses w… Bahnhof =)

Grüße
Vincent

Antworten

OlliW *sagt:*

21. Januar 2014 um 14:28

Hallo Vincent,

ich hatte schon mal einen Blick auf den C-Code geworfen, aber keinen Versuch gemacht ihn und die Grundlagen zu “verdauen”… ich bin noch auf 6DOF :-). Ich denke du hast schon recht dass das so ähnlich wie bei Mahony in Punkt (3) (ich denke das meintest du) ist, allerdings kann man beim Magnetometer nicht einfach den Magnetfeldvektor nehmen (wie bei der Beschleunigung), sondern braucht die Komponenten desselben in der XY Ebene (der Erde), d.h. hier ist noch zusätzlich eine Umrechnung mit der geschätzen Raumlage ins Erdkoordinaten-System nötig (das ist das h und b). In [PB] findest du eine ausführliche Betrachtung, ich könnte mir vorstellen dass das in Quaternionen übersetzt auf dein w führt. Normierungen macht man meist um (1) numerische Fehler zu veringern oder (2) weil es schneller geht als der “geradlinige” Weg (hier könnte es z.B. da sein um die normierte Projektion des Vektor in die XY zu bekommen). Aber wie gesagt, richtig tief beschäftigt habe ich mich mit der Magnetometer-Sache noch nicht…

Cheers, Olli

Vincent *sagt:*

21. Januar 2014 um 17:55

Hallo Olli

Ok, habs eben selbst gelöst…
Is eh selbsterklärend… nur ziemlich bescheuert geschrieben.

Der transformierte (vom Sensor ins Erd-System) und normierte Vektor b muss natürlich wieder zurück ins Sensorsystem transformiert werden, um ihn mit dem gemessenen Magnetfeldvektor vergleichen zu können!

sprich das ganze
w = [2*b(2)*(0.5 – q(3)^2 – q(4)^2) + 2*b(4)*(q(2)*q(4) – q(1)*q(3))

2*b(2)*(q(2)*q(3) – q(1)*q(4)) + 2*b(4)*(q(1)*q(2) + q(3)*q(4))
2*b(2)*(q(1)*q(3) + q(2)*q(4)) + 2*b(4)*(0.5 – q(2)^2 – q(3)^2)];

lässt sich auch beschrieben via
w = [b(2) 0 b(4)]*quat2dcm(q)'
sprich der Vektor * der transponierten Rotationsmatrix

oder
w = quatmultiply(quatconj(q), quatmultiply(b, q));
sprich der Vektor über die Quaternionmultiplikation sowie der Multiplikation mit der Inversen zurücktransformiert

lg
Vincent

OlliW *sagt:*

21. Januar 2014 um 19:32

supi
Danke für die Rückmeldung und Aufklärung!

Hinterlasse einen Kommentar

Deine E-Mail-Adresse wird nicht veröffentlicht. Erforderliche Felder sind markiert *

Kommentar

Name *

E-Mail *

Website

Kommentar abschicken

Impressum

Es sollte klar sein: Das ist eine private Webseite eines Freizeitbastlers, ich übernehme natürlich keinerlei Verantwortung für die Richtigkeit der Inhalte. Für eigene Inhalte beanspruche ich aber das übliche Copyright; eine kommerzielle Nutzung jeglicher Art ist ausdrücklich untersagt.

Diese Website benutzt Google Analytics, Datenschutzerklärung wie hier unter Punkt 4. Die IP ist anonymisiert. Die Daten werden NICHT weitergehend ausgewertet.

Blogroll

- Documentation
- FPV Community
- Helifreak
- Mikrocontroller.Net
- OlliW42 -YouTube
- RC Groups Aircraft/Helis
- RC-Heli
- RCLine Forum

Themen

- 450er CP Heli
- AVR Projekte
- Blog
- Brushless Gimbal
- Grundlagen
- Koax Helikopter
- Micro CP Heli
- RC AVR Projekte
- RC Elektronik
- RC Elektronik Projekte
- STM32 Projekte

Seiten

- Formelsammlung
- Resourcen
- Willkommen bei OlliW's Bastelseiten
- Alle Artikel
- RCLine Artikel
- Kontakt