Extended Kalman Filter

The Extended Kalman Filter is one of the most used algorithms in the world, and this module will use it to compute the attitude as a quaternion with the observations of tri-axial gyroscopes, accelerometers and magnetometers.

The **state** is the physical state, which can be described by dynamic variables. The **noise** in the measurements means that there is a certain degree of uncertainty in them [Hartikainen2011].

A dynamical system is a system whose state evolves over time, so differential equations are normally used to model them [Labbe2015]. There is also noise in the dynamics of the system, **process noise**, which means we cannot be entirely deterministic, but we can get indirect noisy measurements.

Here, the term **filtering** refers to the process of *filtering out* the noise in the measurements to provide an optimal estimate for the state, given the observed measurements.

The instantaneous state of the system is represented with a vector updated through discrete time increments to generate the next state. The simplest of the state space models are linear models [Hartikainen2011], which can be expressed with equations of the following form:

$$\begin{aligned} \mathbf{x}_t &= \mathbf{F}\mathbf{x}_{t-1} + \mathbf{w}_x \\ \mathbf{z}_t &= \mathbf{H}\mathbf{x}_t + \mathbf{w}_z \end{aligned}$$

where

- $\mathbf{x}_t \in \mathbb{R}^n$ is the **state** of the system describing the condition of n elements at time t.
- $\mathbf{z}_t \in \mathbb{R}^m$ are the **measurements** at time t.
- $\mathbf{w}_x \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_t)$ is the **process noise** at time t.
- $\mathbf{w}_z \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_t)$ is the **measurement noise** at time t.
- $\mathbf{F} \in \mathbb{R}^{n \times n}$ is called either the **State Transition Matrix** or the **Fundamental Matrix**, and sometimes is represented with $\mathbf{\Phi}$. It depends on the literature.
- $\mathbf{H}$ is the measurement model matrix.

Many linear models are also described with continuous-time state equations of the form:

$$\dot{\mathbf{x}}_t = \frac{d\mathbf{x}_t}{dt} = \mathbf{A}\mathbf{x}_t + \mathbf{L}\mathbf{w}_t$$

where $\mathbf{A}$ and $\mathbf{L}$ are **constant** matrices characterizing the behaviour of the model, and $\mathbf{w}_t$ is a white noise with a power spectral density $\sigma^2_{\mathbf{w}}$.

The main difference is that $\mathbf{A}$ models a set of linear differential equations, and is continuous. $\mathbf{F}$ is discrete, and represents a set of linear equations (not differential equations) which transitions $\mathbf{x}_{t-1}$ to $\mathbf{x}_t$ over a discrete time step Δt.

A common way to obtain $\mathbf{F}$ uses the matrix exponential, which can be expanded with a Taylor series [Sola]:

$$\mathbf{F} = e^{\mathbf{A}\Delta t} = \mathbf{I} + \mathbf{A}\Delta t + \frac{(\mathbf{A}\Delta t)^2}{2!} + \frac{(\mathbf{A}\Delta t)^3}{3!} + \cdots = \sum_{k=0}^{\infty} \frac{(\mathbf{A}\Delta t)^k}{k!}$$

The main goal is to find an equation that recursively finds the value of $\mathbf{x}_t$ in terms of $\mathbf{x}_{t-1}$.

Kalman Filter

The solution proposed by [Kalman1960] models a system with a set of n^{th}-order differential equations, converts them into an equivalent set of first-order differential equations, and puts them into the matrix form $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$. Once in this form several techniques are used to convert these linear differential equations into the recursive equation $\mathbf{x}_t = \mathbf{F}\mathbf{x}_{t-1}$.

The Kalman filter has two steps:

1. The **prediction step** estimates the next state, and its covariance, at time t of the system given the previous state at time $t-1$.

$$\begin{aligned}
\hat{\mathbf{x}}_t &= \mathbf{F}\mathbf{x}_{t-1} + \mathbf{B}\mathbf{u}_t \\
\hat{\mathbf{P}}_t &= \mathbf{F}\mathbf{P}_{t-1}\mathbf{F}^T + \mathbf{Q}_t
\end{aligned}$$

2. The **correction step** rectifies the estimation with a set of measurements $\mathbf{z}$ at time t.

$$\begin{aligned}
\mathbf{v}_t &= \mathbf{z}_t - \mathbf{H}\hat{\mathbf{x}}_t \\
\mathbf{S}_t &= \mathbf{H}\hat{\mathbf{P}}_t\mathbf{H}^T + \mathbf{R} \\
\mathbf{K}_t &= \hat{\mathbf{P}}_t\mathbf{H}^T\mathbf{S}_t^{-1} \\
\mathbf{x}_t &= \hat{\mathbf{x}}_t + \mathbf{K}_t\mathbf{v}_t \\
\mathbf{P}_t &= \hat{\mathbf{P}}_t - \mathbf{K}_t\mathbf{S}_t\mathbf{K}_t^T
\end{aligned}$$

where:

- $\hat{\mathbf{P}}_t \in \mathbb{R}^{n \times n}$ is the **Predicted Covariance** of the state before seeing the measurements $\mathbf{z}_t$.
- $\mathbf{P}_t \in \mathbb{R}^{n \times n}$ is the **Estimated Covariance** of the state after seeing the measurements $\mathbf{z}_t$.
- $\mathbf{u}_t \in \mathbb{R}^k$ is a **Control input vector** defining the expected behaviour of the system.
- $\mathbf{B} \in \mathbb{R}^{n \times k}$ is the **Control input model**.
- $\mathbf{H} \in \mathbb{R}^{m \times n}$ is the **Observation model** linking the predicted state and the measurements.
- $\mathbf{v}_t \in \mathbb{R}^m$ is the **Innovation** or **Measurement residual**.
- $\mathbf{S}_t \in \mathbb{R}^{m \times m}$ is the **Measurement Prediction Covariance**.
- $\mathbf{K}_t \in \mathbb{R}^{n \times m}$ is the filter *gain*, a.k.a. the **Kalman Gain**, telling how much the predictions should be corrected.

The *predicted* state $\hat{\mathbf{x}}_t$ is estimated in the first step based on the previously computed state $\mathbf{x}_{t-1}$, and later is *"corrected"* during the second step to obtain the final estimation $\mathbf{x}_t$. A similar computation happens with its covariance $\mathbf{P}$.

> **Note**
>
> The hat notation is (mostly) used to indicate an *estimated value*. It marks here the calculation of the state in the prediction step at time t.

The loop starts with prior mean $\mathbf{x}_0$ and prior covariance $\mathbf{P}_0$, which are defined by the system model.

Extended Kalman Filter

The functions have been Gaussian and linear so far and, thus, the output was always another Gaussian, but Gaussians are not closed under nonlinear functions.

The EKF handles nonlinearity by forming a Gaussian approximation to the joint distribution of state $\mathbf{x}$ and measurements $\mathbf{z}$ using Taylor series based transformations [Hartikainen2011].

Likewise, the EKF is split into two steps:

Prediction

$$
\begin{aligned}
\hat{\mathbf{x}}_t &= \mathbf{f}(\mathbf{x}_{t-1}, \mathbf{u}_t) \\
\hat{\mathbf{P}}_t &= \mathbf{F}(\mathbf{x}_{t-1}, \mathbf{u}_t)\mathbf{P}_{t-1}\mathbf{F}^T(\mathbf{x}_{t-1}, \mathbf{u}_t) + \mathbf{Q}_t
\end{aligned}
$$

Correction

$$
\begin{aligned}
\mathbf{v}_t &= \mathbf{z}_t - \mathbf{h}(\mathbf{x}_t) \\
\mathbf{S}_t &= \mathbf{H}(\mathbf{x}_t)\hat{\mathbf{P}}_t\mathbf{H}^T(\mathbf{x}_t) + \mathbf{R}_t \\
\mathbf{K}_t &= \hat{\mathbf{P}}_t\mathbf{H}^T(\mathbf{x}_t)\mathbf{S}_t^{-1} \\
\mathbf{x}_t &= \hat{\mathbf{x}}_t + \mathbf{K}_t\mathbf{v}_t \\
\mathbf{P}_t &= \big(\mathbf{I}_4 - \mathbf{K}_t\mathbf{H}(\mathbf{x}_t)\big)\hat{\mathbf{P}}_t
\end{aligned}
$$

where $\mathbf{f}$ is the nonlinear dynamic model function, and $\mathbf{h}$ is the nonlinear measurement model function. The matrices $\mathbf{F}$ and $\mathbf{H}$ are the Jacobians of $\mathbf{f}$ and $\mathbf{h}$, respectively:

$$
\begin{aligned}
\mathbf{F}(\mathbf{x}_{t-1}, \mathbf{u}_t) &= \frac{\partial \mathbf{f}(\mathbf{x}_{t-1}, \mathbf{u}_t)}{\partial \mathbf{x}} \\
\mathbf{H}(\mathbf{x}_t) &= \frac{\partial \mathbf{h}(\mathbf{x}_t)}{\partial \mathbf{x}}
\end{aligned}
$$

Notice that the matrices $\mathbf{F}_{t-1}$ and $\mathbf{H}_t$ of the normal KF are replaced with Jacobian matrices $\mathbf{F}(\mathbf{x}_{t-1}, \mathbf{u}_t)$ and $\mathbf{H}(\hat{\mathbf{x}}_t)$ in the EKF, respectively. The predicted state, $\hat{\mathbf{x}}_t$, and the residual of the prediction, $\mathbf{v}_t$, are also calculated differently.

> **Attention**
>
> The state transition and the observation models **must** be differentiable functions.

Quaternion EKF

In this case, we will use the EKF to estimate an orientation represented as a quaternion $\mathbf{q}$. First, we *predict* the new state (newest orientation) using the immediate measurements of the gyroscopes, then we *correct* this state using the measurements of the accelerometers and magnetometers.

All sensors are assumed to have a fixed sampling rate ($f = \frac{1}{\Delta t}$), even though the time step can be computed at any sample n as $\Delta t = t_n - t_{n-1}$. Numerical integration will give a discrete set of n values, that are approximations at discrete times $t_n = t_0 + n\Delta t$.

Gyroscope data are treated as external inputs to the filter rather than as measurements, and their measurement noises enter the filter as *process noise* rather than as measurement noise [Sabatini2011].

For this model, the quaternion $\mathbf{q}$ will be the **state vector**, and the angular velocity $\boldsymbol{\omega}$, in *rad/s*, will be the **control vector**:

$$\mathbf{x} \triangleq \mathbf{q} = \begin{bmatrix} q_w & \mathbf{q}_v \end{bmatrix}^T = \begin{bmatrix} q_w & q_x & q_y & q_z \end{bmatrix}^T$$

$$\mathbf{u} \triangleq \boldsymbol{\omega} = \begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^T$$

Therefore, transition models are described as:

$$\dot{\mathbf{q}}_t = \mathbf{A}\mathbf{q}_{t-1} + \mathbf{B}\boldsymbol{\omega}_t + \mathbf{w}_\mathbf{q}$$

$$\hat{\mathbf{q}}_t = \mathbf{F}\mathbf{q}_{t-1} = e^{\mathbf{A}\Delta t}\mathbf{q}_{t-1}$$

with $\mathbf{w}_\mathbf{q}$ being the **process noise**.

Prediction Step

In the first step, the quaternion at time t is **predicted** by integrating the angular rate $\boldsymbol{\omega}$, and adding it to the formerly computed quaternion $\mathbf{q}_{t-1}$:

$$\hat{\mathbf{q}}_t = \mathbf{q}_{t-1} + \int_{t-1}^{t} \boldsymbol{\omega}\, dt$$

This constant augmentation is sometimes termed the **Attitude Propagation**. Because exact closed-form solutions of the integration are not available, an approximation method is required.

Discretization

Using the Euler-Rodrigues rotation formula to redefine the quaternion [Sabatini2011] we find:

$$\hat{\mathbf{q}}_t = \left[\cos\left(\frac{\|\boldsymbol{\omega}\|\Delta t}{2}\right)\mathbf{I}_4 + \frac{2}{\|\boldsymbol{\omega}\|}\sin\left(\frac{\|\boldsymbol{\omega}\|\Delta t}{2}\right)\boldsymbol{\Omega}_t \right]\mathbf{q}_{t-1}$$

where $\mathbf{I}_4$ is a 4×4 Identity matrix, and:

$$\boldsymbol{\Omega}_t = \begin{bmatrix} 0 & -\boldsymbol{\omega}^T \\ \boldsymbol{\omega} & \lfloor\boldsymbol{\omega}\rfloor_\times \end{bmatrix} = \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix}$$

The expression $\lfloor\mathbf{x}\rfloor_\times$ expands a vector $\mathbf{x} \in \mathbb{R}^3$ into a 3×3 skew-symmetric matrix. The large term inside the brackets, multiplying $\mathbf{q}_{t-1}$, is an orthogonal rotation retaining the normalization of the propagated attitude quaternions, and we might be tempted to consider it equal to $\mathbf{F}$, but it is not yet linear, although it is **already discrete**. It must be linearized to be used in the EKF.

Linearization

n^{th}-order polynomial linearization methods can be built from Taylor series of $\mathbf{q}(t_n + \Delta t)$ around the time $t = t_n$:

$$\mathbf{q}_t = \mathbf{q}_{t-1} + \dot{\mathbf{q}}_{t-1}\Delta t + \frac{1}{2!}\ddot{\mathbf{q}}_{t-1}\Delta t^2 + \frac{1}{3!}\dddot{\mathbf{q}}_{t-1}\Delta t^3 + \cdots$$

where $\dot{\mathbf{q}} = \frac{d\mathbf{q}}{dt}$ is the derivative of the quaternion [1]:

$$\begin{aligned}
\dot{\mathbf{q}} &= \lim_{\Delta t \rightarrow 0} \frac{\mathbf{q}(t+\Delta t) - \mathbf{q}(t)}{\Delta t} \\
&= \frac{1}{2}\mathbf{\Omega}_t \mathbf{q}_{t-1} \\
&= \frac{1}{2}\begin{bmatrix} -\omega_x q_x - \omega_y q_y - \omega_z q_z \\ \omega_x q_w + \omega_z q_y - \omega_y q_z \\ \omega_y q_w - \omega_z q_x + \omega_x q_z \\ \omega_z q_w + \omega_y q_x - \omega_x q_y \end{bmatrix}
\end{aligned}$$

The angular rates $\boldsymbol{\omega}$ are measured by the gyroscopes in the local *sensor frame*. Hence, this term describes the evolution of the orientation with respect to the local frame [Sola].

Using the definition of $\dot{\mathbf{q}}$, the predicted state, $\hat{\mathbf{q}}_t$ is written as [Wertz]:

$$\begin{aligned}
\hat{\mathbf{q}}_t &= \left(\mathbf{I}_4 + \frac{1}{2}\mathbf{\Omega}_t \Delta t + \frac{1}{2!}\left(\frac{1}{2}\mathbf{\Omega}_t \Delta t\right)^2 + \frac{1}{3!}\left(\frac{1}{2}\mathbf{\Omega}_t \Delta t\right)^3 + \cdots\right)\mathbf{q}_{t-1} \\
&\quad + \frac{1}{4}\dot{\mathbf{\Omega}}_t \Delta t^2 \mathbf{q}_{t-1} + \left[\frac{1}{12}\dot{\mathbf{\Omega}}_t \mathbf{\Omega}_t + \frac{1}{24}\mathbf{\Omega}_t \dot{\mathbf{\Omega}}_t + \frac{1}{12}\ddot{\mathbf{\Omega}}_t\right]\Delta t^3 \mathbf{q}_{t-1} + \cdots
\end{aligned}$$

Assuming the angular rate is constant over the period $[t-1, t]$, we have $\dot{\boldsymbol{\omega}} = 0$, and we can prescind from the derivatives of $\mathbf{\Omega}$ reducing the series to:

$$\hat{\mathbf{q}}_t = \left(\mathbf{I}_4 + \frac{1}{2}\mathbf{\Omega}_t \Delta t + \frac{1}{2!}\left(\frac{1}{2}\mathbf{\Omega}_t \Delta t\right)^2 + \frac{1}{3!}\left(\frac{1}{2}\mathbf{\Omega}_t \Delta t\right)^3 + \cdots\right)\mathbf{q}_{t-1}$$

Notice the series has the known form of the matrix exponential:

$$e^{\frac{\Delta t}{2}\mathbf{\Omega}_t} = \sum_{k=0}^{\infty} \frac{1}{k!}\left(\frac{\Delta t}{2}\mathbf{\Omega}_t\right)^k$$

The error of the approximation vanishes rapidly at higher orders, or when the time step $\Delta t \rightarrow 0$. The more terms we have, the better our approximation becomes, with the downside of a big computational demand. For simple architectures (like embedded systems) we can reduce this burden by truncating the series to its second term making it a **First Order EKF** [2], and achieving fairly good results. Thus, our **process model** shortens to:

$$\begin{aligned}
\hat{\mathbf{q}}_t &= \mathbf{f}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t) \\
&= \left(\mathbf{I}_4 + \frac{\Delta t}{2}\mathbf{\Omega}_t\right)\mathbf{q}_{t-1} \\
\begin{bmatrix} \hat{q_w} \\ \hat{q_x} \\ \hat{q_y} \\ \hat{q_z} \end{bmatrix} &= \begin{bmatrix} q_w - \frac{\Delta t}{2}\omega_x q_x - \frac{\Delta t}{2}\omega_y q_y - \frac{\Delta t}{2}\omega_z q_z \\ q_x + \frac{\Delta t}{2}\omega_x q_w - \frac{\Delta t}{2}\omega_y q_z + \frac{\Delta t}{2}\omega_z q_y \\ q_y + \frac{\Delta t}{2}\omega_x q_z + \frac{\Delta t}{2}\omega_y q_w - \frac{\Delta t}{2}\omega_z q_x \\ q_z - \frac{\Delta t}{2}\omega_x q_y + \frac{\Delta t}{2}\omega_y q_x + \frac{\Delta t}{2}\omega_z q_w \end{bmatrix}
\end{aligned}$$

And now, we simply compute $\mathbf{F}$ as [3]:

$$
\begin{aligned}
\mathbf{F}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t) &= \frac{\partial \mathbf{f}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t)}{\partial \mathbf{q}} \\
&= \begin{bmatrix} \frac{\partial \mathbf{f}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t)}{\partial q_w} & \frac{\partial \mathbf{f}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t)}{\partial q_x} & \frac{\partial \mathbf{f}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t)}{\partial q_y} & \frac{\partial \mathbf{f}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t)}{\partial q_z} \end{bmatrix} \\
&= \begin{bmatrix} 1 & -\frac{\Delta t}{2}\omega_x & -\frac{\Delta t}{2}\omega_y & -\frac{\Delta t}{2}\omega_z \\ \frac{\Delta t}{2}\omega_x & 1 & \frac{\Delta t}{2}\omega_z & -\frac{\Delta t}{2}\omega_y \\ \frac{\Delta t}{2}\omega_y & -\frac{\Delta t}{2}\omega_z & 1 & \frac{\Delta t}{2}\omega_x \\ \frac{\Delta t}{2}\omega_z & \frac{\Delta t}{2}\omega_y & -\frac{\Delta t}{2}\omega_x & 1 \end{bmatrix}
\end{aligned}
$$

Process Noise Covariance

Per definition, the predicted error state covariance is calculated as:

$$\hat{\mathbf{P}}_t = \mathbf{F}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t)\mathbf{P}_{t-1}\mathbf{F}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t)^T + \mathbf{Q}_t$$

We already know how to compute $\mathbf{F}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t)$, but we still need to compute the *Process Noise Covariance Matrix* $\mathbf{Q}_t$.

The noise at the prediction step lies mainy in the control input. Consequently, $\mathbf{Q}_t$ is derived mainly from the gyroscope.

We define $\boldsymbol{\sigma}_{\boldsymbol{\omega}}^2 = \begin{bmatrix} \sigma_{\omega x}^2 & \sigma_{\omega y}^2 & \sigma_{\omega z}^2 \end{bmatrix}^T$ as the spectral density of the gyroscopic noises on each axis, whose standard deviation, $\boldsymbol{\sigma}_{\boldsymbol{\omega}}$, is specified as a scalar in *rad/s*.

Without boring too much into DSP analysis, we can frankly say that the smaller the spectral density of a sensor is, the less noisy its signals are, and we would tend to trust its readings a bit more. Normally, these noises can be found already in the datasheets provided by the sensor manufacturers.

Taking for granted that the gyroscope is the same kind on its axes with the manufacturer guaranteeing perfect orthogonality and uncorrelation between them, we build the *spectral noise covariance matrix* as:

$$\boldsymbol{\Sigma}_{\boldsymbol{\omega}} = \begin{bmatrix} \sigma_{\omega x}^2 & 0 & 0 \\ 0 & \sigma_{\omega y}^2 & 0 \\ 0 & 0 & \sigma_{\omega z}^2 \end{bmatrix}$$

The easiest solution is to assume that this noise is consistent, and our process noise covariance would simply be $\mathbf{Q}_t = \boldsymbol{\Sigma}_{\boldsymbol{\omega}}$, but in a real system this covariance changes depending on the angular velocity, so it will need to be recomputed for every prediction.

We need to know how much noise is added to the system over a discrete interval Δt and, therefore, we integrate the process noise over the interval $[0, \Delta t]$

$$\mathbf{Q}_t = \int_0^{\Delta t} e^{\mathbf{A}_t} \boldsymbol{\Sigma}_{\boldsymbol{\omega}} e^{\mathbf{A}_t^T} dt$$

As we have seen, the matrix exponentials tend to be computationally demanding, so, we opt to use the Jacobian of the prediction, but with respect to the angular rate:

$$
\begin{aligned}
\mathbf{W}_t &= \frac{\partial \mathbf{f}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t)}{\partial \boldsymbol{\omega}} \\
&= \begin{bmatrix} \frac{\partial \mathbf{f}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t)}{\partial \omega_x} & \frac{\partial \mathbf{f}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t)}{\partial \omega_y} & \frac{\partial \mathbf{f}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t)}{\partial \omega_z} \end{bmatrix} \\
&= \frac{\Delta t}{2} \begin{bmatrix} -q_x & -q_y & -q_z \\ q_w & -q_z & q_y \\ q_z & q_w & -q_x \\ -q_y & q_x & q_w \end{bmatrix}
\end{aligned}
$$

Notice this Jacobian is very similar to $\mathbf{F}$, but this one has partial derivatives with respect to $\boldsymbol{\omega}$. Having the noise values and the Jacobian $\mathbf{W}_t$, the Process Noise Covariance is computed at each time t with:

$$\mathbf{Q}_t = \mathbf{W}_t \boldsymbol{\Sigma}_{\boldsymbol{\omega}} \mathbf{W}_t^T$$

For convenience, it is assumed that the noises are equal on each axis, and don't influence each other, yielding a white, uncorrelated and isotropic noise [4]:

$$\boldsymbol{\Sigma}_{\boldsymbol{\omega}} = \sigma_{\boldsymbol{\omega}}^2 \mathbf{I}_3$$

further simplifying the computation to:

$$\mathbf{Q}_t = \sigma_{\boldsymbol{\omega}}^2 \mathbf{W}_t \mathbf{W}_t^T$$

Warning

The assumption that the noise variances of the gyroscope axes are all equal ($\sigma_{wx} = \sigma_{wy} = \sigma_{wz}$) is almost never true in reality. It is possible to infer the individual variances through a careful modeling and calibration process [Lam2003]. If these three different values are at hand, it is then recommended to compute the Process Noise Covariance with $\mathbf{W}_t \boldsymbol{\Sigma}_{\boldsymbol{\omega}} \mathbf{W}_t^T$.

Finally, the prediction step of this model would propagate the covariance matrix like:

$$\hat{\mathbf{P}}_t = \mathbf{F}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t) \mathbf{P}_{t-1} \mathbf{F}(\mathbf{q}_{t-1}, \boldsymbol{\omega}_t)^T + \sigma_{\boldsymbol{\omega}}^2 \mathbf{W}_t \mathbf{W}_t^T$$

Correction Step

The gyroscope measurements have been used, so far, to predict the new state of the quaternion, but there are also accelerometer and magnetometer readings available, that can be used to *correct* the estimation.

We know, from the equations above, that the corrected state can be computed as:

$$\mathbf{q}_t = \hat{\mathbf{q}}_t + \mathbf{K}_t \big(\mathbf{z}_t - \mathbf{h}(\mathbf{q}_t) \big)$$

where

- $\mathbf{z}_t \in \mathbb{R}^6$ is the current measurement.
- $\mathbf{h}_t \in \mathbb{R}^6$ is the predicted measurement.
- $\mathbf{K}_t \in \mathbb{R}^{4 \times 6}$ is the Kalman gain.

Tip

The Kalman Gain can be thought of as a *blending value* within the range $[0.0, 1.0]$, that decides how much of the innovation $\mathbf{v}_t = \mathbf{z}_t - \mathbf{h}(\mathbf{q}_t)$ will be considered. You can see, for example, that:

- If $\mathbf{K} = 0$, there is no correction: $\mathbf{q}_t = \hat{\mathbf{q}}_t$.
- If $\mathbf{K} = 1$, the state will be corrected with *all* the innovation: $\mathbf{q}_t = \hat{\mathbf{q}}_t + \mathbf{v}_t$.

We start by defining the measurement vector as:

$$\mathbf{z}_t = \begin{bmatrix} \mathbf{a}_t \\ \mathbf{m}_t \end{bmatrix} = \begin{bmatrix} a_x \\ a_y \\ a_z \\ m_x \\ m_y \\ m_z \end{bmatrix}$$

These are the values obtained from the sensors, where $\mathbf{a} \in \mathbb{R}^3$ is a tri-axial accelerometer sample, in $\frac{m}{s^2}$, and $\mathbf{m} \in \mathbb{R}^3$ is a tri-axial magnetometer sample, in μT.

Their noises $\mathbf{w_a}$ and $\mathbf{w_m}$ are assumed independent white Gaussian with zero mean of covariances $\mathbf{\Sigma_a} = \boldsymbol{\sigma}_\mathbf{a}^2 \mathbf{I}_3$ and $\mathbf{\Sigma_m} = \boldsymbol{\sigma}_\mathbf{m}^2 \mathbf{I}_3$.

However, while the gyroscopes give measurements in the *sensor frame*, the accelerometers and magnetometers deliver measurements of the *global frame*. We must represent the accelerometers and magnetometer readings, if we want to correct the estimated orientation in the *sensor frame*.

The predicted quaternion $\hat{\mathbf{q}}_t$ describes the orientation of the sensor frame with respect to the global frame. For that reason, it will be used to rotate the sensor measurements.

To rotate any vector $\mathbf{x} \in \mathbb{R}^3$ through a quaternion $\mathbf{q} \in \mathbb{H}^4$, we can use its representation as a rotation matrix $\mathbf{C}(\mathbf{q}) \in \mathbb{R}^{3\times3}$. For the estimated quaternion $\hat{\mathbf{q}}_t$ this becomes:

$$\begin{aligned} \mathbf{x}' &= \mathbf{C}(\hat{\mathbf{q}})\mathbf{x} \\ &= \begin{bmatrix} 1 - 2(\hat{q}_y^2 + \hat{q}_z^2) & 2(\hat{q}_x\hat{q}_y - \hat{q}_w\hat{q}_z) & 2(\hat{q}_x\hat{q}_z + \hat{q}_w\hat{q}_y) \\ 2(\hat{q}_x\hat{q}_y + \hat{q}_w\hat{q}_z) & 1 - 2(\hat{q}_x^2 + \hat{q}_z^2) & 2(\hat{q}_y\hat{q}_z - \hat{q}_w\hat{q}_x) \\ 2(\hat{q}_x\hat{q}_z - \hat{q}_w\hat{q}_y) & 2(\hat{q}_w\hat{q}_x + \hat{q}_y\hat{q}_z) & 1 - 2(\hat{q}_x^2 + \hat{q}_y^2) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \end{aligned}$$

Global References

There are two main global reference frames based on the local tangent plane:

- **NED** defines the X-, Y-, and Z-axis colinear to the geographical *North*, *East*, and *Down* directions, respectively.
- **ENU** defines the X-, Y-, and Z-axis colinear to the geographical *East*, *North*, and *Up* directions, respectively.

The gravitational acceleration vector in a global *NED* frame is normally defined as $\mathbf{g}_{\text{NED}} = \begin{bmatrix} 0 & 0 & -9.81 \end{bmatrix}^T$, where the normal acceleration, $g_0 \approx 9.81 \frac{m}{s^2}$, acts solely on the Z-axis [5]. For the *ENU* frame, it simply flips the sign along the Z-axis, $\mathbf{g}_{\text{ENU}} = \begin{bmatrix} 0 & 0 & 9.81 \end{bmatrix}^T$.

Earth's magnetic field is also represented with a 3D vector, $\mathbf{r} = \begin{bmatrix} r_x & r_y & r_z \end{bmatrix}^T$, whose values indicate the direction of the magnetic flow. In an ideal case only the *North* component holds a value, yielding the options $\mathbf{r}_{\text{NED}} = \begin{bmatrix} r_x & 0 & 0 \end{bmatrix}^T$ or $\mathbf{r}_{\text{ENU}} = \begin{bmatrix} 0 & r_y & 0 \end{bmatrix}^T$

The geomagnetic field is, however, not regular on the planet, and it even changes over time (see WMM for more details.) Usually, certain authors prefer to define this referencial vector discarding the eastwardly magnetic information, sometimes projecting its magnitude against the XY plane with the help of the magnetic dip angle, θ, resulting in $\mathbf{r}_{\text{NED}} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \end{bmatrix}^T$ and $\mathbf{r}_{\text{ENU}} = \begin{bmatrix} 0 & \cos\theta & -\sin\theta \end{bmatrix}^T$.

From the acceleration and magnetic field merely their directions are required, not really their magnitudes. We can, therefore, use their normalized values, simplifying their definition to:

$$\mathbf{g} = \begin{cases} \begin{bmatrix} 0 & 0 & -1 \end{bmatrix}^T & \text{if NED} \\ \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T & \text{if ENU} \end{cases}$$

$$\mathbf{r} = \begin{cases} \frac{1}{\sqrt{\cos^2\theta + \sin^2\theta}} \begin{bmatrix} \cos\theta & 0 & \sin\theta \end{bmatrix}^T & \text{if NED} \\ \frac{1}{\sqrt{\cos^2\theta + \sin^2\theta}} \begin{bmatrix} 0 & \cos\theta & -\sin\theta \end{bmatrix}^T & \text{if ENU} \end{cases}$$

To compare these vectors against their corresponding observations, we must **also normalize** the sensors' measurements:

$$\mathbf{a} = \frac{1}{\sqrt{a_x^2 + a_y^2 + a_z^2}} \begin{bmatrix} a_x & a_y & a_z \end{bmatrix}^T$$

$$\mathbf{m} = \frac{1}{\sqrt{m_x^2 + m_y^2 + m_z^2}} \begin{bmatrix} m_x & m_y & m_z \end{bmatrix}^T$$

Measurement Model

The *expected* gravitational acceleration in the sensor frame, $\hat{\mathbf{a}}$, can be estimated from the *estimated orientation*:

$$\hat{\mathbf{a}} = \mathbf{C}(\hat{\mathbf{q}})^T \mathbf{g}$$

Similarly, the *expected magnetic field* in sensor frame, $\hat{\mathbf{m}}$ is:

$$\hat{\mathbf{m}} = \mathbf{C}(\hat{\mathbf{q}})^T \mathbf{r}$$

The measurement model, $\mathbf{h}(\hat{\mathbf{q}}_t)$, and its Jacobian, $\mathbf{H}(\hat{\mathbf{q}}_t)$, can be used to correct the predicted model. The *Measurement model* is directly defined with these transformations as:

$$
\begin{aligned}
\mathbf{h}(\hat{\mathbf{q}}_t) &= \begin{bmatrix} \hat{\mathbf{a}} \\ \hat{\mathbf{m}} \end{bmatrix} = \begin{bmatrix} \mathbf{C}(\hat{\mathbf{q}})^T \mathbf{g} \\ \mathbf{C}(\hat{\mathbf{q}})^T \mathbf{r} \end{bmatrix} \\
&= 2 \begin{bmatrix}
g_x(\frac{1}{2} - q_y^2 - q_z^2) + g_y(q_w q_z + q_x q_y) + g_z(q_x q_z - q_w q_y) \\
g_x(q_x q_y - q_w q_z) + g_y(\frac{1}{2} - q_x^2 - q_z^2) + g_z(q_w q_x + q_y q_z) \\
g_x(q_w q_y + q_x q_z) + g_y(q_y q_z - q_w q_x) + g_z(\frac{1}{2} - q_x^2 - q_y^2) \\
r_x(\frac{1}{2} - q_y^2 - q_z^2) + r_y(q_w q_z + q_x q_y) + r_z(q_x q_z - q_w q_y) \\
r_x(q_x q_y - q_w q_z) + r_y(\frac{1}{2} - q_x^2 - q_z^2) + r_z(q_w q_x + q_y q_z) \\
r_x(q_w q_y + q_x q_z) + r_y(q_y q_z - q_w q_x) + r_z(\frac{1}{2} - q_x^2 - q_y^2)
\end{bmatrix}
\end{aligned}
$$

The measurement equations are nonlinear, which forces to, accordingly, compute their Jacobian matrix as:

$$
\begin{aligned}
\mathbf{H}(\hat{\mathbf{q}}_t) &= \frac{\partial \mathbf{h}(\hat{\mathbf{q}}_t)}{\partial \mathbf{q}} \\
&= \begin{bmatrix} \frac{\partial \mathbf{h}(\hat{\mathbf{q}}_t)}{\partial q_w} & \frac{\partial \mathbf{h}(\hat{\mathbf{q}}_t)}{\partial q_x} & \frac{\partial \mathbf{h}(\hat{\mathbf{q}}_t)}{\partial q_y} & \frac{\partial \mathbf{h}(\hat{\mathbf{q}}_t)}{\partial q_z} \end{bmatrix} \\
&= 2 \begin{bmatrix}
g_y q_z - g_z q_y & g_y q_y + g_z q_z & -2g_x q_y + g_y q_x - g_z q_w & -2g_x q_z + g_y q_w + g_z q_x \\
-g_x q_z + g_z q_x & g_x q_y - 2g_y q_x + g_z q_w & g_x q_x + g_z q_z & -g_x q_w - 2g_y q_z + g_z q_y \\
g_x q_y - g_y q_x & g_x q_z - g_y q_w - 2g_z q_x & g_x q_w + g_y q_z - 2g_z q_y & g_x q_x + g_y q_y \\
r_y q_z - r_z q_y & r_y q_y + r_z q_z & -2r_x q_y + r_y q_x - r_z q_w & -2r_x q_z + r_y q_w + r_z q_x \\
-r_x q_z + r_z q_x & r_x q_y - 2r_y q_x + r_z q_w & r_x q_x + r_z q_z & -r_x q_w - 2r_y q_z + r_z q_y \\
r_x q_y - r_y q_x & r_x q_z - r_y q_w - 2r_z q_x & r_x q_w + r_y q_z - 2r_z q_y & r_x q_x + r_y q_y
\end{bmatrix}
\end{aligned}
$$

This Jacobian can be refactored as:

$$
\mathbf{H}(\hat{\mathbf{q}}_t) = \begin{bmatrix} \frac{\partial \hat{\mathbf{a}}}{\partial \mathbf{q}} \\ \frac{\partial \hat{\mathbf{m}}}{\partial \mathbf{q}} \end{bmatrix} = 2 \begin{bmatrix} \mathbf{u}_g & \lfloor \mathbf{u}_g + \hat{q}_w \mathbf{g} \rfloor_\times + (\hat{\mathbf{q}}_v \cdot \mathbf{g}) \mathbf{I}_3 - \mathbf{g} \hat{\mathbf{q}}_v^T \\ \mathbf{u}_r & \lfloor \mathbf{u}_r + \hat{q}_w \mathbf{r} \rfloor_\times + (\hat{\mathbf{q}}_v \cdot \mathbf{r}) \mathbf{I}_3 - \mathbf{r} \hat{\mathbf{q}}_v^T \end{bmatrix}
$$

with

$$
\begin{aligned}
\mathbf{u}_g &= \lfloor \mathbf{g} \rfloor_\times \hat{\mathbf{q}}_v = \mathbf{g} \times \hat{\mathbf{q}}_v \\
\mathbf{u}_r &= \lfloor \mathbf{r} \rfloor_\times \hat{\mathbf{q}}_v = \mathbf{r} \times \hat{\mathbf{q}}_v
\end{aligned}
$$

The measurement noise covariance matrix, $\mathbf{R} \in \mathbb{R}^{6 \times 6}$, is expressed directly in terms of the statistics of the measurement noise affecting each sensor [Sabatini2011]. The sensor noises are considered as uncorrelated and isotropic, which creates a diagonal matrix:

$$
\mathbf{R} = \begin{bmatrix} \boldsymbol{\sigma}_{\mathbf{a}}^2 \mathbf{I}_3 & \mathbf{0}_3 \\ \mathbf{0}_3 & \boldsymbol{\sigma}_{\mathbf{m}}^2 \mathbf{I}_3 \end{bmatrix}
$$

This definition allows us to obtain simple expressions for $\mathbf{P}_t$. The rest of the EKF elements in the correction step are obtained as defined originally:

$$
\begin{aligned}
\mathbf{v}_t &= \mathbf{z}_t - \mathbf{h}(\hat{\mathbf{q}}_t) \\
\mathbf{S}_t &= \mathbf{H}(\hat{\mathbf{q}}_t) \hat{\mathbf{P}}_t \mathbf{H}^T(\hat{\mathbf{q}}_t) + \mathbf{R} \\
\mathbf{K}_t &= \hat{\mathbf{P}}_t \mathbf{H}^T(\hat{\mathbf{q}}_t) \mathbf{S}_t^{-1}
\end{aligned}
$$

Lastly, the corrected state (quaternion) and its covariance at time t are computed with:

$$
\begin{aligned}
\mathbf{q}_t &= \hat{\mathbf{q}}_t + \mathbf{K}_t \mathbf{v}_t \\
\mathbf{P}_t &= \big(\mathbf{I}_4 - \mathbf{K}_t \mathbf{H}(\hat{\mathbf{q}}_t)\big)\hat{\mathbf{P}}_t
\end{aligned}
$$

The EKF is not an optimal estimator and might diverge quickly with an incorrect model. The origin of the problem lies in the lack of independence of the four components of the quaternion, since they are related by the unit-norm constraint. The easiest solution is to normalize the corrected state.

$$
\mathbf{q}_t \leftarrow \frac{1}{\|\mathbf{q}_t\|}\mathbf{q}_t
$$

Even though it is neither elegant nor optimal, this "brute-force" approach to compute the final quaternion is proven to work generally well [Sabatini2011].

Initial values

The EKF state vector represents the quaternion, and we must provide an appropriate initial state, $\mathbf{q}_0$, that corresponds to a valid quaternion. That means, the initial state must have a norm equal to one ($\|\mathbf{q}_0\| = 1$)

To estimate this initial orientation a simple ecompass method is used, which requires single observations of a tri-axial accelerometer and a tri-axial magnetometer, in a motionless state.

Similar to TRIAD, the orientation can be estimated with only one observation of both sensors. A rotation matrix $\mathbf{C} \in \mathbb{R}^{3\times 3}$ of this estimation is computed as:

$$
\mathbf{C} = \begin{bmatrix} (\mathbf{a}_0 \times \mathbf{m}_0) \times \mathbf{a}_0 & \mathbf{a}_0 \times \mathbf{m}_0 & \mathbf{a}_0 \end{bmatrix}
$$

where $\mathbf{a}_0$ and $\mathbf{m}_0$ are the accelerometer and magnetometer measurements. Each column of this rotation matrix should be normalized. Then, we get the initial quaternion with [Chiaverini]:

$$
\mathbf{q}_0 = \begin{bmatrix}
\frac{1}{2}\sqrt{c_{11} + c_{22} + c_{33} + 1} \\
\frac{1}{2}\mathrm{sgn}(c_{32} - c_{23})\sqrt{c_{11} - c_{22} - c_{33} + 1} \\
\frac{1}{2}\mathrm{sgn}(c_{13} - c_{31})\sqrt{c_{22} - c_{33} - c_{11} + 1} \\
\frac{1}{2}\mathrm{sgn}(c_{21} - c_{12})\sqrt{c_{33} - c_{11} - c_{22} + 1}
\end{bmatrix}
$$

The initial state covariance matrix, $\mathbf{P}_0$ is set equal to a 4×4 identity matrix $\mathbf{I}_4$.

Appropriate values of $\boldsymbol{\sigma}_{\boldsymbol{\omega}}^2$, $\boldsymbol{\sigma}_{\mathbf{a}}^2$ and $\boldsymbol{\sigma}_{\mathbf{m}}^2$ should also be provided. These values can be normally found in the datasheet of each sensor. The accelerometer's and magnetometers measurement noise variances are increased in value, thus giving more weight to the filter predictions. For our case, the default values are:

$$
\begin{aligned}
\boldsymbol{\sigma}_{\boldsymbol{\omega}}^2 &= 0.3^2 \\
\boldsymbol{\sigma}_{\mathbf{a}}^2 &= 0.5^2 \\
\boldsymbol{\sigma}_{\mathbf{m}}^2 &= 0.8^2
\end{aligned}
$$

Final Notes

The model described and implemented here does not consider any biases in the signal, which can be also added to the state vector to be dynamically computed. But this extra analysis is left out to provide a simple EKF. Therefore, the given **sensor signals are considered to be calibrated already**.

No compensation for magnetic disturbances is considered either. It is assumed that the magnetometer has been calibrated in the enviroment, where it is used, so that scaling and bias errors are neglected too.

The modularity of the class allows for scalability in the estimation. It is possible, for example, to expand the state vector (and its covariance) to include the bias terms, if we need to, by simply setting the attribute `q`.

Footnotes

[1] The successive derivatives of $\mathbf{q}_n$ are obtained by repeatedly applying the expression of the quaternion derivative, with $\ddot{\boldsymbol{\omega}} = 0$.

$$\begin{aligned}
\dot{\mathbf{q}}_n &= \frac{1}{2}\mathbf{q}_n\boldsymbol{\omega}_n \\
\ddot{\mathbf{q}}_n &= \frac{1}{4}\mathbf{q}_n\boldsymbol{\omega}_n^2 + \frac{1}{2}\mathbf{q}_n\dot{\boldsymbol{\omega}} \\
\dddot{\mathbf{q}}_n &= \frac{1}{6}\mathbf{q}_n\boldsymbol{\omega}_n^3 + \frac{1}{4}\mathbf{q}_n\dot{\boldsymbol{\omega}}\boldsymbol{\omega}_n + \frac{1}{2}\mathbf{q}\boldsymbol{\omega}_n\dot{\boldsymbol{\omega}} \\
\mathbf{q}_n^{i\geq 4} &= \frac{1}{2^i}\mathbf{q}_n\boldsymbol{\omega}_n^i + \cdots
\end{aligned}$$

[2] Other applications of the EKF truncate the series after the second term, producing a 2nd, 3rd, or nth-order EKF. This slightly increases the accuracy of the model, with the disadvantage of an elevated computational demand.

[3] Numerically, the $\mathbf{F}$ can be *approximated* with the matrix exponential $e^{\mathbf{A}\Delta t}$. This function has been largerly implemented in several packages, including Scipy: `F=scipy.linalg.expm(A*Dt)`; however, the Jacobian of $\mathbf{f}$ is always prefered.

[4] A three-dimensional isotropic covariance describes a sphere centered at the origin, which means that its mean and covariance matrix are invariant upon rotations. In algebraic terms, an isotropic covariance matrix is an identity matrix multiplied by a scalar.

[5] The magnitude of the normal gravity also changes with respect to the location on Earth. Simpler models consider only the latitude and height above sea level to estimate this magnitude. For the purpose of analysis a common value of *9.81* is given.

References

[Kalman1960] Rudolf Kalman. A New Approach to Linear Filtering and Prediction Problems. 1960.

[Hartikainen2011] *(1, 2, 3)* J. Hartikainen, A. Solin and S. Särkkä. Optimal Filtering with Kalman Filters and Smoothers. 2011

[Sabatini2011] *(1, 2, 3, 4)* Sabatini, A.M. Kalman-Filter-Based Orientation Determination Using Inertial/Magnetic Sensors: Observability Analysis and Performance Evaluation. Sensors 2011, 11, 9182-9206. (https://www.mdpi.com/1424-8220/11/10/9182)

[Labbe2015] Roger R. Labbe Jr. Kalman and Bayesian Filters in Python. (https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python)

[Lam2003] Lam, Quang & Stamatakos, Nick & Woodruff, Craig & Ashton, Sandy. Gyro Modeling and Estimation of Its Random Noise Sources. AAIA 2003. DOI: 10.2514/6.2003-5562. (https://www.researchgate.net/publication/268554081)

---

class **ahrs.filters.ekf.EKF**(*gyr: numpy.ndarray = None, acc: numpy.ndarray = None, mag: numpy.ndarray = None, frequency: float = 100.0, frame: str = 'NED', **kwargs*)

Extended Kalman Filter to estimate orientation as Quaternion.

Examples

```
>>> import numpy as np
>>> from ahrs.filters import EKF
>>> from ahrs.common.orientation import acc2q
>>> ekf = EKF()
>>> num_samples = 1000              # Assuming sensors have 1000 samples each
>>> Q = np.zeros((num_samples, 4))  # Allocate array for quaternions
>>> Q[0] = acc2q(acc_data[0])       # First sample of tri-axial accelerometer
>>> for t in range(1, num_samples):
...     Q[t] = ekf.update(Q[t-1], gyr_data[t], acc_data[t])
```

The estimation is simplified by giving the sensor values at the construction of the EKF object. This will perform all steps above and store the estimated orientations, as quaternions, in the attribute `Q`.

```
>>> ekf = EKF(gyr=gyr_data, acc=acc_data)
>>> ekf.Q.shape
(1000, 4)
```

In this case, the measurement vector, set in the attribute `z`, is equal to the measurements of the accelerometer. If extra information from a magnetometer is available, it will also be considered to estimate the attitude.

```
>>> ekf = EKF(gyr=gyr_data, acc=acc_data, mag=mag_data)
>>> ekf.Q.shape
(1000, 4)
```

For this case, the measurement vector contains the accelerometer and magnetometer measurements together: `z = [acc_data, mag_data]` at each time t.

The most common sampling frequency is 100 Hz, which is used in the filter. If that is different in the given sensor data, it can be changed too.

```
>>> ekf = EKF(gyr=gyr_data, acc=acc_data, mag=mag_data, frequency=200.0)
```

Normally, when using the magnetic data, a referencial magnetic field must be given. This filter computes the local magnetic field in Munich, Germany, but it can also be set to a different reference with the parameter `mag_ref`.

```
>>> ekf = EKF(gyr=gyr_data, acc=acc_data, mag=mag_data, magnetic_ref=[17.06, 0.78, 34.39])
```

If the full referencial vector is not available, the magnetic dip angle, in degrees, can be also used.

```
>>> ekf = EKF(gyr=gyr_data, acc=acc_data, mag=mag_data, magnetic_ref=60.0)
```

The initial quaternion is estimated with the first observations of the tri-axial accelerometers and magnetometers, but it can also be given directly in the parameter `q0`.

```
>>> ekf = EKF(gyr=gyr_data, acc=acc_data, mag=mag_data, q0=[0.7071, 0.0, -0.7071, 0.0])
```

Measurement noise variances must be set from each sensor, so that the Process and Measurement Covariance Matrix can be built. They are set in an array equal to `[0.3**2, 0.5**2, 0.8**2]` for the gyroscope, accelerometer and magnetometer, respectively. If a different set of noise variances is used, they can be set with the parameter `noises`:

```
>>> ekf = EKF(gyr=gyr_data, acc=acc_data, mag=mag_data, noises=[0.1**2, 0.3**2, 0.5**2])
```

or the individual variances can be set separately too:

```
>>> ekf = EKF(gyr=gyr_data, acc=acc_data, mag=mag_data, var_acc=0.3**2)
```

This class can also differentiate between NED and ENU frames. By default it estimates the orientations using the NED frame, but ENU is used if set in its parameter:

```
>>> ekf = EKF(gyr=gyr_data, acc=acc_data, mag=mag_data, frame='ENU')
```

Parameters:

- **gyr** (*numpy.ndarray*, *default: None*) – N-by-3 array with measurements of angular velocity in rad/s
- **acc** (*numpy.ndarray*, *default: None*) – N-by-3 array with measurements of acceleration in in m/s^2
- **mag** (*numpy.ndarray*, *default: None*) – N-by-3 array with measurements of magnetic field in mT
- **frequency** (*float*, *default: 100.0*) – Sampling frequency in Herz.
- **frame** (*str*, *default: 'NED'*) – Local tangent plane coordinate frame. Valid options are right-handed `'NED'` for North-East-Down and `'ENU'` for East-North-Up.
- **q0** (*numpy.ndarray*, *default: None*) – Initial orientation, as a versor (normalized quaternion).
- **magnetic_ref** (*float or numpy.ndarray*) – Local magnetic reference.
- **noises** (*numpy.ndarray*) – List of noise variances for each type of sensor. Default values: `[0.3**2, 0.5**2, 0.8**2]`.
- **Dt** (*float*, *default: 0.01*) – Sampling step in seconds. Inverse of sampling frequency. NOT required if `frequency` value is given.

Omega(*x: numpy.ndarray*) → numpy.ndarray

Omega operator.

Given a vector $\mathbf{x} \in \mathbb{R}^3$, return a 4×4 matrix of the form:

$$\boldsymbol{\Omega}(\mathbf{x}) = \begin{bmatrix} 0 & -\mathbf{x}^T \\ \mathbf{x} & \lfloor \mathbf{x} \rfloor_\times \end{bmatrix} = \begin{bmatrix} 0 & -x_1 & -x_2 & -x_3 \\ x_1 & 0 & x_3 & -x_2 \\ x_2 & -x_3 & 0 & x_1 \\ x_3 & x_2 & -x_1 & 0 \end{bmatrix}$$

This operator is constantly used at different steps of the EKF.

Parameters: **x** (*numpy.ndarray*) – Three-dimensional vector.

Returns: **Omega** – Omega matrix.

Return type: numpy.ndarray

dfdq(*omega: numpy.ndarray*, *dt: float*) → numpy.ndarray

Jacobian of linearized predicted state.

$$\mathbf{F} = \frac{\partial \mathbf{f}(\mathbf{q}_{t-1})}{\partial \mathbf{q}} = \begin{bmatrix} 1 & -\frac{\Delta t}{2}\omega_x & -\frac{\Delta t}{2}\omega_y & -\frac{\Delta t}{2}\omega_z \\ \frac{\Delta t}{2}\omega_x & 1 & \frac{\Delta t}{2}\omega_z & -\frac{\Delta t}{2}\omega_y \\ \frac{\Delta t}{2}\omega_y & -\frac{\Delta t}{2}\omega_z & 1 & \frac{\Delta t}{2}\omega_x \\ \frac{\Delta t}{2}\omega_z & \frac{\Delta t}{2}\omega_y & -\frac{\Delta t}{2}\omega_x & 1 \end{bmatrix}$$

Parameters:
- **omega** (*numpy.ndarray*) – Angular velocity in rad/s.
- **dt** (*float*) – Time step, in seconds, between consecutive Quaternions.

Returns: **F** – Jacobian of state.

Return type: numpy.ndarray

dhdq(*q: numpy.ndarray*, *mode: str = 'normal'*) → numpy.ndarray

Linearization of observations with Jacobian.

If only the gravitational acceleration is used to correct the estimation, a 3×4 matrix:

$$\mathbf{H}(\hat{\mathbf{q}}_t) = 2 \begin{bmatrix} g_y q_z - g_z q_y & g_y q_y + g_z q_z & -2g_x q_y + g_y q_x - g_z q_w & -2g_x q_z + g_y q_w + g \\ -g_x q_z + g_z q_x & g_x q_y - 2g_y q_x + g_z q_w & g_x q_x + g_z q_z & -g_x q_w - 2g_y q_z + g \\ g_x q_y - g_y q_x & g_x q_z - g_y q_w - 2g_z q_x & g_x q_w + g_y q_z - 2g_z q_y & g_x q_x + g_y q_y \end{bmatrix}$$

If the gravitational acceleration and the geomagnetic field are used, then a 6×4 matrix is used:

$$\mathbf{H}(\hat{\mathbf{q}}_t) = 2 \begin{bmatrix} g_y q_z - g_z q_y & g_y q_y + g_z q_z & -2g_x q_y + g_y q_x - g_z q_w & -2g_x q_z + g_y q_w + g \\ -g_x q_z + g_z q_x & g_x q_y - 2g_y q_x + g_z q_w & g_x q_x + g_z q_z & -g_x q_w - 2g_y q_z + g \\ g_x q_y - g_y q_x & g_x q_z - g_y q_w - 2g_z q_x & g_x q_w + g_y q_z - 2g_z q_y & g_x q_x + g_y q_y \\ r_y q_z - r_z q_y & r_y q_y + r_z q_z & -2r_x q_y + r_y q_x - r_z q_w & -2r_x q_z + r_y q_w + r \\ -r_x q_z + r_z q_x & r_x q_y - 2r_y q_x + r_z q_w & r_x q_x + r_z q_z & -r_x q_w - 2r_y q_z + r \\ r_x q_y - r_y q_x & r_x q_z - r_y q_w - 2r_z q_x & r_x q_w + r_y q_z - 2r_z q_y & r_x q_x + r_y q_y \end{bmatrix}$$

If `mode` is equal to `'refactored'`, the computation is carried out as:

$$\mathbf{H}(\hat{\mathbf{q}}_t) = 2 \begin{bmatrix} \mathbf{u}_g & \lfloor \mathbf{u}_g + \hat{q}_w \mathbf{g} \rfloor_\times + (\hat{\mathbf{q}}_v \cdot \mathbf{g})\mathbf{I}_3 - \mathbf{g}\hat{\mathbf{q}}_v^T \\ \mathbf{u}_r & \lfloor \mathbf{u}_r + \hat{q}_w \mathbf{r} \rfloor_\times + (\hat{\mathbf{q}}_v \cdot \mathbf{r})\mathbf{I}_3 - \mathbf{r}\hat{\mathbf{q}}_v^T \end{bmatrix}$$

Warning

The refactored mode might lead to slightly different results as it employs more and different operations than the normal mode, created by the nummerical capabilities of the host system.

Parameters:
- **q** (*numpy.ndarray*) – Predicted state estimate.
- **mode** (*str*, *default: 'normal'*) – Computation mode for Observation matrix.

Returns: **H** – Jacobian of observations.

Return type: numpy.ndarray

f(*q: numpy.ndarray*, *omega: numpy.ndarray*, *dt: float*) → **numpy.ndarray**

Linearized function of Process Model (Prediction.)

$$\mathbf{f}(\mathbf{q}_{t-1}) = \left(\mathbf{I}_4 + \frac{\Delta t}{2}\mathbf{\Omega}_t\right)\mathbf{q}_{t-1} = \begin{bmatrix} q_w - \frac{\Delta t}{2}\omega_x q_x - \frac{\Delta t}{2}\omega_y q_y - \frac{\Delta t}{2}\omega_z q_z \\ q_x + \frac{\Delta t}{2}\omega_x q_w - \frac{\Delta t}{2}\omega_y q_z + \frac{\Delta t}{2}\omega_z q_y \\ q_y + \frac{\Delta t}{2}\omega_x q_z + \frac{\Delta t}{2}\omega_y q_w - \frac{\Delta t}{2}\omega_z q_x \\ q_z - \frac{\Delta t}{2}\omega_x q_y + \frac{\Delta t}{2}\omega_y q_x + \frac{\Delta t}{2}\omega_z q_w \end{bmatrix}$$

Parameters:
- **q** (*numpy.ndarray*) – A-priori quaternion.
- **omega** (*numpy.ndarray*) – Angular velocity, in rad/s.
- **dt** (*float*) – Time step, in seconds, between consecutive Quaternions.

Returns: **q** – Linearized estimated quaternion in **Prediction** step.

Return type: numpy.ndarray

h(*q: numpy.ndarray*) → **numpy.ndarray**

Measurement Model

If only the gravitational acceleration is used to correct the estimation, a vector with 3 elements is used:

$$\mathbf{h}(\hat{\mathbf{q}}_t) = 2\begin{bmatrix} g_x(\frac{1}{2} - q_y^2 - q_z^2) + g_y(q_w q_z + q_x q_y) + g_z(q_x q_z - q_w q_y) \\ g_x(q_x q_y - q_w q_z) + g_y(\frac{1}{2} - q_x^2 - q_z^2) + g_z(q_w q_x + q_y q_z) \\ g_x(q_w q_y + q_x q_z) + g_y(q_y q_z - q_w q_x) + g_z(\frac{1}{2} - q_x^2 - q_y^2) \end{bmatrix}$$

If the gravitational acceleration and the geomagnetic field are used, then a vector with 6 elements is used:

$$\mathbf{h}(\hat{\mathbf{q}}_t) = 2\begin{bmatrix} g_x(\frac{1}{2} - q_y^2 - q_z^2) + g_y(q_w q_z + q_x q_y) + g_z(q_x q_z - q_w q_y) \\ g_x(q_x q_y - q_w q_z) + g_y(\frac{1}{2} - q_x^2 - q_z^2) + g_z(q_w q_x + q_y q_z) \\ g_x(q_w q_y + q_x q_z) + g_y(q_y q_z - q_w q_x) + g_z(\frac{1}{2} - q_x^2 - q_y^2) \\ r_x(\frac{1}{2} - q_y^2 - q_z^2) + r_y(q_w q_z + q_x q_y) + r_z(q_x q_z - q_w q_y) \\ r_x(q_x q_y - q_w q_z) + r_y(\frac{1}{2} - q_x^2 - q_z^2) + r_z(q_w q_x + q_y q_z) \\ r_x(q_w q_y + q_x q_z) + r_y(q_y q_z - q_w q_x) + r_z(\frac{1}{2} - q_x^2 - q_y^2) \end{bmatrix}$$

Parameters: **q** (*numpy.ndarray*) – Predicted Quaternion.

Returns: Expected Measurements.

Return type: numpy.ndarray

update(*q: numpy.ndarray*, *gyr: numpy.ndarray*, *acc: numpy.ndarray*, *mag: numpy.ndarray = None*, *dt: float = None*) → **numpy.ndarray**

Perform an update of the state.

Parameters:
- **q** (*numpy.ndarray*) – A-priori orientation as quaternion.
- **gyr** (*numpy.ndarray*) – Sample of tri-axial Gyroscope in rad/s.
- **acc** (*numpy.ndarray*) – Sample of tri-axial Accelerometer in m/s^2.
- **mag** (*numpy.ndarray*) – Sample of tri-axial Magnetometer in uT.
- **dt** (*float*, *default: None*) – Time step, in seconds, between consecutive Quaternions.

Returns: **q** – Estimated a-posteriori orientation as quaternion.

Return type: numpy.ndarray