# Quaternion-Based Extended Kalman Filter for Fixed-Wing UAV Attitude Estimation

## Derivation and Implementation

Philip M. Salmony, October 2019

## 1 Introduction

An essential part in controlling an Unmanned Air Vehicle (UAV) is having accurate and reliable state estimates available for feedback, which are then used in the governing control systems. Unfortunately, estimating these states - such as roll, pitch, and yaw angles - is no trivial task in such a dynamic environment and when using relatively inexpensive, noisy sensors.

A tried-and-tested method of state estimation of dealing with this problem is via the use of an Extended Kalman Filter (EKF), which can also handle non-linear system models via linearisation. The EKF is the industry-standard in most systems these days, such as commercial aircraft and figher jets.

To circumvent problems with computationally expensive operations, numerical stability, and gimbal lock, a quaternion-based EKF is developed, as opposed to representing the aircraft's attitude via Euler angles. The EKF in this document is specifically tailored to a (small) UAV and common sensors aboard such a system.

A final implementation in both Matlab and C code is also given.

# 2 Coordinate System

The aircraft body-frame coordinate system referred to in this document is shown in Figure 1.

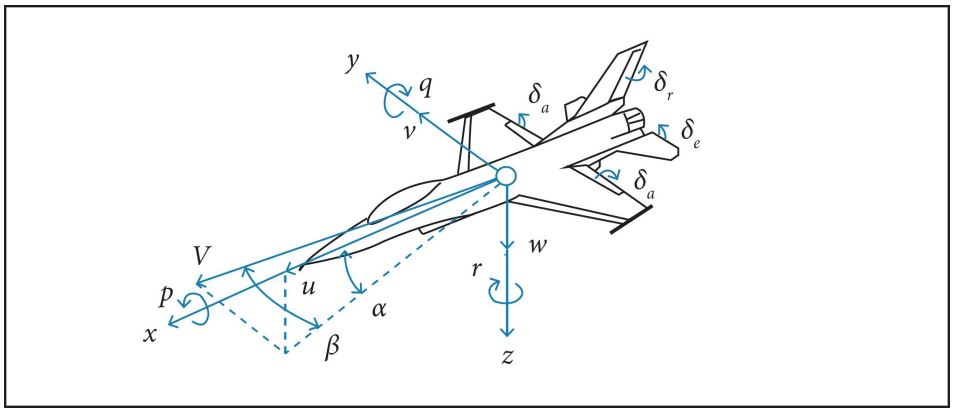


Figure 1: Fixed-Wing Aircraft Body-Frame System

# 3 Sensors

## 3.1 Gyroscope

A three-axis gyroscope is required, giving measurements for rate of rotation around each axis in the body-frame in $\mathrm{rad\,s^{-1}}$. The resulting measurements have the symbols $p$ (rotation about x-axis), $q$ (rotation about y-axis), and $r$ (rotation about z-axis).

## 3.2 Accelerometer

A three-axis accelerometer is required, giving measurements of total acceleration. The measured acceleration in the body-frame is as follows,

$$\underline{a}^b = \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} + \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} u \\ v \\ w \end{bmatrix} - \underline{g}^b. \tag{1}$$

Where u, v, and w are the velocities aligned along the axes in the body-frame and

$\underline{g}^b$ is the gravitational vector rotated to the body-frame. The resulting units are $\mathrm{m\,s^{-2}}$.

### 3.3 Magnetometer

A three-axis magnetometer is used to give a unit-vector measurement of the local magnetic field. Additionally, the magnetic declination at the UAV's position must be known (in rad). This can be acquired via a look-up table or in GPS NMEA data.

### 3.4 Differential Pressure Sensor and Pitot Tube

A differential pressure sensor in combination with a pitot static tube can be used to estimate the UAV's airspeed. Given a measurement of the differential pressure $\Delta P$ and knowledge of the ambient air density $\rho$, the airspeed can be calculated as,

$$V_a = \sqrt{\frac{2 \cdot \Delta P}{\rho}}. \tag{2}$$

## 4 State Vector

The states to be estimated are contained in the state vector $\underline{x}$,

$$\underline{x} = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 & b_p & b_q & b_r \end{bmatrix}^T. \tag{3}$$

Where $q_n$ is the n-th quaternion, and $b_p$, $b_p$, $b_r$ are the respective gyro biases associated with the x, y, and z-axes.

When needed - for instance for use in a flight control loop - the quaternion attitude estimates can be converted to Euler angles via the following relationships,

$$\phi = \arctan \frac{2 \cdot (q_0 q_1 + q_2 q_3)}{1 - 2 \cdot (q_1^2 + q_2^2)}, \tag{4}$$

$$\theta = \arcsin 2 \cdot (q_0 q_2 - q_3 q_1), \tag{5}$$

$$\psi = \arctan \frac{2 \cdot (q_0 q_3 + q_1 q_2)}{1 - 2 \cdot (q_2^2 + q_3^2)}. \tag{6}$$

# 5 Filtering Measurements

Inevitably, all sensors measurements will have high-frequency noise superimposed on them. Thus, before passing the measurements on to the EKF algorithm, it is important to low-pass filter the sensor readings.

A simple, critically-damped, second-order low-pass filter with -40dB per decade roll-off after the cut-off frequency $w_c = 2\pi f_c$ has the following transfer function,

$$G(s) = \frac{w_c^2}{s^2 + 2w_c s + w_c^2}. \tag{7}$$

For a small sampling time $T$, we may use a backward Euler scheme to convert this into a discrete IIR filter. The final difference equation, useful for implementation, is then found to be,

$$y_n = \frac{(w_c T)^2 x_n + 2(1 + w_c T) y_{n-1} - y_{n-2}}{1 + 2w_c T + (w_c T)^2}. \tag{8}$$

Where $x_n$ is the current, raw sensor reading and $y_n$ is the latest filtered measurement.

# 6 EKF Algorithm

The algorithm for the discrete-time extended Kalman filter is given below. The algorithm assumes it is being run with a sample time $T$.

1. **Initialise**
   - State estimate $\underline{x}$,
   - Error covariance matrix $\mathbf{P}$,
   - Process noise matrix $\mathbf{Q}$,
   - Measurement noise matrix $\mathbf{R}$.
2. **Predict**
   - $\underline{x}_{n+1} = \underline{x}_n = T \cdot f(\underline{x}, \underline{u})$,

- $\mathbf{A} = \frac{\partial f}{\partial \underline{x}}(\underline{x}, \underline{u})$,
- $\mathbf{P}_{n+1} = \mathbf{P}_n + T \cdot (\mathbf{A}\mathbf{P}_n + \mathbf{P}_n\mathbf{A}^T + \mathbf{Q})$.

3. **Update**
   - $\mathbf{C} = \frac{\partial z}{\partial \underline{x}}(\underline{x}, \underline{u})$,
   - $\mathbf{K} = \mathbf{P}_{n+1}\mathbf{C}^T \cdot (\mathbf{C}\mathbf{P}_{n+1}\mathbf{C}^T + \mathbf{R})$,
   - $\mathbf{P}_{n+1} = (\mathbf{I} - \mathbf{K}\mathbf{C})\mathbf{P}_{n+1}$,
   - $\underline{x}_{n+1} = \underline{x}_{n+1} + \mathbf{K}(\underline{y} - z(\underline{x}, \underline{u}))$.

# 7 Derivation of EKF Parameters

## 7.1 State Transition Function and Jacobian

The state transition function $f(\underline{x}, \underline{u})$, relates the current states and inputs to the state derivatives such that $\dot{\underline{x}} = f(\underline{x}, \underline{u})$. Via simple Euler integration, the state estimate can be updated at each timestep with sample time $T$,

$$\underline{x}_{n+1} = \underline{x}_n + T \cdot f(\underline{x}_n, \underline{u}_n). \tag{9}$$

The quaternion derivatives are related to the body-frame rotation terms, $\underline{\omega} = \begin{bmatrix} p & q & r \end{bmatrix}^T$, and gyro biases, $\underline{b} = \begin{bmatrix} b_p & b_q & b_r \end{bmatrix}^T$, via the following expression,

$$\dot{\underline{q}} = \frac{1}{2} \cdot \mathbf{Q} \cdot (\underline{\omega} - \underline{b}) \tag{10}$$

Where,

$$\mathbf{Q} = \frac{1}{2} \begin{bmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{bmatrix}. \tag{11}$$

The state transition equations for the gyro biases are simply zero (i.e. $\dot{\underline{b}} = 0$), as we do not have an adequate way of modelling the change in bias over time.

Now, by differentiating the state transition function w.r.t the state vector $\underline{x}$, we arrive at the Jacobian $\mathbf{A}(\underline{x}, \underline{u}) = \frac{\partial f}{\partial \underline{x}}$,

$$\mathbf{A} = \frac{1}{2}\begin{bmatrix} 0 & -(p-b_p) & -(q-b_q) & -(r-b_r) & q_1 & q_2 & q_3 \\ (p-b_p) & 0 & (r-b_r) & -(q-b_q) & -q_0 & q_3 & -q_2 \\ (q-b_q) & -(r-b_r) & 0 & (p-b_p) & -q_3 & -q_0 & q_1 \\ (r-b_r) & (q-b_q) & -(p-b_p) & 0 & q_2 & -q_1 & q_0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{12}$$

## 7.2 Output Function and Jacobian

The output function $z(\underline{x}, \underline{u})$ relates the internal, estimated states to the measurements. Thus, the output function is a model of the relevant sensors.

The output function in this case will model the accelerometers and magnetometer. Firstly, from equation 1 we are unable to directly measure $u$, $v$, and $w$ and the respective time derivatives. Thus, we assume the linear accelerations are small and therefore $\dot{u} = \dot{v} = \dot{w} \approx 0$.

Furthermore, we assume that the sideways velocity ($v$) is negligible. Finally, assuming the angle of attack $\alpha$ is approximately equal to the pitch angle $\theta$ and that the side-slip angle $\beta$ and pitch angle are small, we arrive at,

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} V_a \\ 0 \\ 0 \end{bmatrix}.$$

Thus, the accelerometer model becomes,

$$\underline{a}^b = \begin{bmatrix} 0 \\ V_a(r-b_r) \\ -V_a(q-b_q) \end{bmatrix} - g \begin{bmatrix} 2(q_1q_3 - q_2q_0) \\ 2(q_2q_3 + q_1q_0) \\ 1 - 2(q_1^2 + q_2^2) \end{bmatrix}. \tag{13}$$

Then, we use the quaternion estimates to compute a magnetic field unit vector estimate in the body-frame. For this, we need to know the magnetic field declination $\delta_m$ (in rad). The magnetometer output equations then become,

$$\underline{m}^b = \begin{bmatrix} 2\sin(\delta_m)(q_0q_3 + q_1q_2) - \cos(\delta_m)(2q_2^2 + 2q_3^2 - 1) \\ -2\cos(\delta_m)(q_0q_3 + q_1q_2) - \sin(\delta_m)(2q_1^2 + 2q_3^2 - 1) \\ 2\cos(\delta_m)(q_0q_2 + q_1q_3) - 2\sin(\delta_m)(q_0q_1 - q_2q_3) \end{bmatrix}. \tag{14}$$

The output function $z(\underline{x}, \underline{u})$ is then simply the vertical concatenation of the vectors $\underline{a}^b$ and $\underline{m}^b$.

Again, to compute the Jacobian $\mathbf{C}(\underline{x}, \underline{u})$ we take the derivative of $z(\underline{x}, \underline{u})$ w.r.t to the state vector $\underline{x}$.

$$\mathbf{C} = \begin{bmatrix} 2gq_2 & -2gq_3 & 2gq_0 & -2gq_1 & 0 & 0 & 0 \\ -2gq_1 & -2gq_0 & -2gq_3 & -2gq_2 & 0 & 0 & -V_a \\ 0 & 4gq_1 & 4gq_2 & 0 & 0 & V_a & 0 \\ 2q_3\sin(\delta_m) & 2q_2\sin(\delta_m) & 2(q_1\sin(\delta_m) - 2q_2\cos(\delta_m)) & 2(q_0\sin(\delta_m) - 2q_3\cos(\delta_m)) & 0 & 0 & 0 \\ -2q_3\cos(\delta_m) & 2(q_2\cos(\delta_m) - 2q_1\sin(\delta_m)) & 2q_1\cos(\delta_m) & -2(q_0\cos(\delta_m) + 2q_3\sin(\delta_m)) & 0 & 0 & 0 \\ 2(q_2\cos(\delta_m) - q_1\sin(\delta_m)) & 2(q_3\cos(\delta_m) - q_0\sin(\delta_m)) & 2(q_0\cos(\delta_m) + q_3\sin(\delta_m)) & 2(q_1\cos(\delta_m) + q_2\sin(\delta_m)) & 0 & 0 & 0 \end{bmatrix}$$

# 8 Implementation

## 8.1 Initialisation

The initial quaternion should be set to $\underline{q} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T$, unless a better estimate of the initial orientation is known. Then, using the Euler angle to quaternion formulae, the initial quaternion should be set. Additionally, by averaging gyro measurements when it is stationary, an initial estimate of the gyro bias vector $\underline{b}$ can be set.

Depending on how well the initial states are known, the error covariance matrix $\mathbf{P}$ is to be set along its diagonal elements, corresponding to the respective states in the state vector. A larger value indicates uncertainty in the initial state estimate.

The process noise matrix $\mathbf{Q}$ describes the uncertainty in the model, as well as the influence of sensor noise - in this case arising due to the gyroscope measurements. The sensor noise parameter can be read off from the sensor's datasheet, then converted to the correct units, and squared before being entered into the correct diagonal entry

of the **Q** matrix.

The measurement noise matrix **R** describes the influence of sensor noise in the output equations - in this case arising due to the accelerometer and magnetometer measurements. As before, the sensor noise parameters can be retrieved from the relevant datasheets, converted to the correct units, squared, and finally entered into the correct diagonal element of the **R** matrix.

## 8.2 Quaternion Normalisation

Floating-point operations during run-time will cause the quaternion estimate to deviate from being unit length. Thus, after both the predict and update steps of the EKF, the quaternion in the state vector needs to be re-normalised such that,

$$q_n = \frac{q_n}{\sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^3}}. \tag{15}$$

## 8.3 Unavailable Sensors or Sensor Failure

Should sensors be unavailable or fail while in-flight, for instance, the magnetometer or the differential pressure sensor, another form of the EKF must be available that takes this into account. This mainly means changing the output equation and its Jacobian. The state transition function stays the same.

## 8.4 C Code

Hand-coding the EKF algorithm is tedious, difficult (mainly due to the fact the we need to invert a larger matrix), and prone to containing errors. Therefore, it is simpler and more efficient to implement the algorithm in a Matlab function, test it on simulated sensor data and compare it to ground truth data, and finally use the *Matlab C Coder* to convert the Matlab function to C or C++ code, which can then, for example, be included in the flight control firmware running on a microcontroller.

**Example Matlab scripts and C code can be found at:** `www.philsal.co.uk` **and** `www.github.com/pms67`**.**

# 9 References


- *Small Unmanned Aircraft: Theory and Practice*, R. W. Beard, T. W. McLAIN (Princeton University Press, 2012)
- *Fundamentals of Small Unmanned Aircraft Flight*, J.D. Barton (John Hopkins APL Technical Digest, Volume 31, Number 2, 2012)
- Figure 1: *A Novel Integrated Guidance and Control System Design in Formation Flight*, M. Sadeghi A. Abaspour, S. H. Sadati, (J. Aerosp. Technol. Manag. [online]. 2015, vol.7, n.4 [cited 2019-10-02])