EKF2 Mathematical Documentation

Complete Mathematical Reference for Quaternion-Based Extended Kalman Filter Implementation

Author: Based on implementation in `EKF2.hpp` and `EKF2.cpp` **Date:** October 2025 **Primary Reference:** Sabatini, A.M. (2006). "Quaternion-Based Extended Kalman Filter for Determining Orientation by Inertial and Magnetic Sensing." IEEE Transactions on Biomedical Engineering, 53(7), 1346-1356.

Table of Contents

Overview

The EKF2 implementation is a **7-state Extended Kalman Filter** for estimating: - **Orientation** (as unit quaternion) - **Gyroscope bias** (3D vector)

Sensors used: - **Gyroscope** - measures angular velocity (prediction) - **Accelerometer** - measures gravity direction (update/correction)

Key features: - Quaternion-based orientation (avoids gimbal lock) - In-line gyroscope bias estimation - First-order linearization for real-time performance

---

State Vector Definition

State Vector (7D)

$$\mathbf{x} = \begin{bmatrix} \mathbf{q} \\ \mathbf{b} \end{bmatrix} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ b_x \\ b_y \\ b_z \end{bmatrix} \in \mathbb{R}^7$$

Components: - **q** = $[q_0, q_1, q_2, q_3]^T$: Unit quaternion representing orientation - q_0 (w): scalar part - q_1, q_2, q_3 (x, y, z): vector part - Constraint: ||**q**|| = 1 (unit norm)

- **b** = $[b_x$, b_y, b_z$]^T$: Gyroscope bias vector (rad/s)

Code Reference:

```
// EKF2.hpp:14-18
Eigen::VectorXd x;  // 7x1 state vector
Eigen::Vector4d getQuaternion() const { return x.head<4>(); }
Eigen::Vector3d getBias() const { return x.tail<3>(); }
```

Quaternion Fundamentals

Quaternion Definition

A quaternion **q** represents a rotation by angle θ about axis **n**:

$$\mathbf{q} = \begin{bmatrix} q_0 \\ \mathbf{q}_v \end{bmatrix} = \begin{bmatrix} \cos(\theta/2) \\ \sin(\theta/2) \cdot \mathbf{n} \end{bmatrix}$$

Where: - $q_0 = \cos(\theta/2)$ - scalar part - **q**_v = $[q_1, q_2, q_3]^T = \sin(\theta/2)\cdot$**n** - vector part - **n** = unit rotation axis

Quaternion to Rotation Matrix

The Direction Cosine Matrix (DCM) **R**(q) ∈ SO(3):

$$\mathbf{R}(\mathbf{q}) = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}$$

Properties: - Rᵀ(q) = R(q)⁻¹ (orthogonal) - det(R) = 1 (special orthogonal) - Transforms vectors from body frame to navigation frame

Code Reference:

```
// EKF2.cpp:152-158
Eigen::Matrix3d EKF2::quaternionToRotationMatrix(const Eigen::Vector4d& q) const
```

Source: Sabatini (2006), Equation (2); Chou (1992) [8]

Quaternion to Euler Angles

Roll (φ):

$$\phi = \text{atan2}(2(q_0 q_1 + q_2 q_3), 1 - 2(q_1^2 + q_2^2))$$

Pitch (θ):

$$\theta = \text{asin}(2(q_0 q_2 - q_3 q_1))$$

Yaw (ψ):

$$\psi = \text{atan2}(2(q_0 q_3 + q_1 q_2), 1 - 2(q_2^2 + q_3^2))$$

Code Reference:

```
// EKF2.cpp:123-144
double EKF2::getRoll() const
double EKF2::getPitch() const
```

Gimbal lock handling: When |sin(pitch)| ≥ 1

Process Model (Prediction Step)

Continuous-Time Quaternion Kinematics

The fundamental differential equation for quaternion under angular velocity **ω**:

$$\dot{\mathbf{q}} = \frac{1}{2}\Omega(\omega)\mathbf{q}$$

Where **Ω**(ω) is the **skew-symmetric matrix**:

$$\Omega(\omega) = \frac{1}{2}\begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix}$$

Source: Sabatini (2006), Equations (3)-(4); Chou (1992) [8]

Discrete-Time State Transition

Gyroscope measurement model:

$$\omega_{measured} = \omega_{true} + \mathbf{b} + \mathbf{v}_g$$

Where: - ω_true: true angular velocity - **b**: gyroscope bias - **v**_g: gyroscope measurement noise

Bias-corrected angular velocity:

$$\tilde{\omega} = \omega_{measured} - \mathbf{b}$$

State transition (first-order Euler integration):

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \omega_k) = \begin{bmatrix} \mathbf{q}_k + \frac{\Delta t}{2}\Omega(\tilde{\omega})\mathbf{q}_k \\ \mathbf{b}_k \end{bmatrix}$$

Quaternion update:

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \frac{\Delta t}{2}\Omega(\tilde{\omega})\mathbf{q}_k$$

Bias update (random walk model):

$$\mathbf{b}_{k+1} = \mathbf{b}_k$$

Normalization (to maintain unit quaternion):

$$\mathbf{q}_{k+1} \leftarrow \frac{\mathbf{q}_{k+1}}{||\mathbf{q}_{k+1}||}$$

Code Reference:

```
// EKF2.cpp:44-72
void EKF2::predict(const Eigen::Vector3d& gyro)
```

Source: Sabatini (2006), Equations (6), (8)-(9)

Measurement Model (Update Step)

Accelerometer Measurement

Physical principle: At rest or constant velocity, accelerometer measures gravity in body frame.

Expected measurement (predicted):

$$\mathbf{h}(\mathbf{x}) = \mathbf{R}^T(\mathbf{q})\mathbf{g}^n$$

Where: - **g**^n = [0, 0, 1]ᵀ : normalized gravity vector in navigation frame (pointing up) - **R**ᵀ(q): rotation from navigation to body frame - **h**: predicted accelerometer reading (normalized)

Actual measurement:

$$\mathbf{z}_k = \frac{\mathbf{a}_{measured}}{||\mathbf{a}_{measured}||} + \mathbf{v}_a$$

Where: - **a**_measured: raw accelerometer reading - Normalization removes magnitude, keeps only direction - **v**_a: measurement noise

Innovation (residual):

$$\mathbf{y}_k = \mathbf{z}_k - \mathbf{h}(\mathbf{x}_k^-)$$

Code Reference:

```
// EKF2.cpp:74-95
void EKF2::update(const Eigen::Vector3d& accel)
```

Source: Sabatini (2006), Equation (11); Gebre-Egziabher et al. (2000) [10]

Why Normalize Accelerometer?

Problem: Accelerometer measures **a** = **g** + **a**_body (gravity + motion)

Solution: During motion, ||**a**|| ≠ g. By normalizing: - We extract only the **direction** information - Measurement becomes: "which way is down?" (relative to body) - Removes magnitude errors from body acceleration

Limitation: Only valid when motion acceleration is small or can be detected and rejected.

State Transition Jacobian (F)

Definition

The Jacobian $\mathbf{F}$ linearizes the nonlinear state transition around the current estimate:

$$\mathbf{F}_k = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\mathbf{x}_k^-}$$

Structure (7×7 block matrix):

$$\mathbf{F} = \begin{bmatrix} \mathbf{F}_{qq} & \mathbf{F}_{qb} \\ \mathbf{F}_{bq} & \mathbf{F}_{bb} \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathbf{q}_{k+1}}{\partial \mathbf{q}_k} & \frac{\partial \mathbf{q}_{k+1}}{\partial \mathbf{b}_k} \\ \frac{\partial \mathbf{b}_{k+1}}{\partial \mathbf{q}_k} & \frac{\partial \mathbf{b}_{k+1}}{\partial \mathbf{b}_k} \end{bmatrix}$$

Block F_qq (4×4): ∂q/∂q

From: q_{k+1} = q_k + (Δt/2)Ω(ω̃)q_k

$$\mathbf{F}_{qq} = \frac{\partial \mathbf{q}_{k+1}}{\partial \mathbf{q}_k} = \mathbf{I}_4 + \frac{\Delta t}{2}\Omega(\tilde{\omega})$$

Code:

```
// EKF2.cpp:175
F.block<4, 4>(0, 0) = Eigen::Matrix4d::Identity() + 0.5 * dt * Omega;
```

Block F_qb (4×3): ∂q/∂b

From chain rule: ω̃ = ω_measured - b, so ∂ω̃/∂b = -I

The derivative of Ω(ω̃)q with respect to ω gives:

$$\frac{\partial(\Omega\mathbf{q})}{\partial\omega} = \begin{bmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{bmatrix}$$

Then using chain rule:

$$\mathbf{F}_{qb} = \frac{\partial \mathbf{q}_{k+1}}{\partial \mathbf{b}_k} = \frac{\Delta t}{2}\begin{bmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{bmatrix}$$

Code:

```
// EKF2.cpp:178-184
Eigen::Matrix<double, 4, 3> F_qb;
F_qb.row(0) = 0.5 * dt * Eigen::Vector3d(-q(1), -q(2), -q(3));
F_qb.row(1) = 0.5 * dt * Eigen::Vector3d( q(0), -q(3),  q(2));
F_qb.row(2) = 0.5 * dt * Eigen::Vector3d( q(3),  q(0), -q(1));
F_qb.row(3) = 0.5 * dt * Eigen::Vector3d(-q(2),  q(1),  q(0));
```

Block F_bq (3×4): ∂b/∂q

Bias is independent of quaternion:

$$\mathbf{F}_{bq} = \frac{\partial \mathbf{b}_{k+1}}{\partial \mathbf{q}_k} = \mathbf{0}_{3\times 4}$$

Block F_bb (3×3): ∂b/∂b

Bias follows identity (constant model):

$$\mathbf{F}_{bb} = \frac{\partial \mathbf{b}_{k+1}}{\partial \mathbf{b}_k} = \mathbf{I}_3$$

Code Reference:

```
// EKF2.cpp:163-187
Eigen::MatrixXd EKF2::computeF(const Eigen::Vector3d& w) const
```

Source: Sabatini (2006), Section II.C; Marins et al. (2001) [7]

Measurement Jacobian (H)

Definition

The Jacobian **H** linearizes the measurement function:

$$\mathbf{H}_k = \left.\frac{\partial \mathbf{h}}{\partial \mathbf{x}}\right|_{\mathbf{x}_k^-}$$

Structure (3×7 block matrix):

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_q & \mathbf{H}_b \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathbf{h}}{\partial \mathbf{q}} & \frac{\partial \mathbf{h}}{\partial \mathbf{b}} \end{bmatrix}$$

Block H_q (3×4): ∂h/∂q

From: h = R^T(q) g^n

We need: ∂(R^T g^n)/∂q_i for i = 0,1,2,3

General formula for any g^n = [g_x, g_y, g_z]ᵀ:

∂h/∂q_0:

$$\frac{\partial \mathbf{h}}{\partial q_0} = 2 \begin{bmatrix} q_0 g_x + q_3 g_y - q_2 g_z \\ -q_3 g_x + q_0 g_y + q_1 g_z \\ q_2 g_x - q_1 g_y + q_0 g_z \end{bmatrix}$$

∂h/∂q_1:

$$\frac{\partial \mathbf{h}}{\partial q_1} = 2 \begin{bmatrix} q_1 g_x + q_2 g_y + q_3 g_z \\ q_2 g_x - q_1 g_y - q_0 g_z \\ q_3 g_x + q_0 g_y - q_1 g_z \end{bmatrix}$$

∂h/∂q_2:

$$\frac{\partial \mathbf{h}}{\partial q_2} = 2 \begin{bmatrix} -q_2 g_x + q_1 g_y - q_0 g_z \\ q_1 g_x + q_2 g_y + q_3 g_z \\ -q_0 g_x + q_3 g_y + q_2 g_z \end{bmatrix}$$

∂h/∂q_3:

$$\frac{\partial \mathbf{h}}{\partial q_3} = 2 \begin{bmatrix} -q_3 g_x + q_0 g_y + q_1 g_z \\ -q_0 g_x - q_3 g_y + q_2 g_z \\ q_1 g_x + q_2 g_y + q_3 g_z \end{bmatrix}$$

For g^n = [0, 0, 1]ᵀ (simplification):

$$\mathbf{H}_q = 2 \begin{bmatrix} -q_2 & q_3 & -q_0 & q_1 \\ q_1 & -q_0 & q_3 & q_2 \\ q_0 & -q_1 & q_2 & q_3 \end{bmatrix}$$

Code:

```cpp
// EKF2.cpp:189-211
Eigen::MatrixXd EKF2::computeH() const
```

Block H_b (3×3): ∂h/∂b

Measurement is independent of gyro bias:

$$\mathbf{H}_b = \frac{\partial \mathbf{h}}{\partial \mathbf{b}} = \mathbf{0}_{3\times 3}$$

Source: Sabatini (2006), Equation (16)-(17); Shuster (1993)

Noise Covariance Matrices

Process Noise Covariance Q (7×7)

Models uncertainty in the process model.

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_q & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_b \end{bmatrix}$$

Quaternion process noise (4×4):

$$\mathbf{Q}_q = \sigma_q^2 \mathbf{I}_4$$

Accounts for: - Gyroscope white noise - Linearization errors - Unmodeled dynamics

Bias process noise (3×3):

$$\mathbf{Q}_b = \sigma_b^2 \mathbf{I}_3$$

Accounts for: - Bias random walk - Temperature drift - Slow bias variations

Code:

```
// EKF2.cpp:35-38
Q = Eigen::MatrixXd::Identity(7, 7);
Q.block<4, 4>(0, 0) *= 0.001;  // σ_q² = 0.001
Q.block<3, 3>(4, 4) *= 0.0001; // σ_b² = 0.0001
```

Typical values: - σ_q ≈ 0.03 rad (quaternion) - σ_b ≈ 0.01 rad/s (bias drift)

Measurement Noise Covariance R (3×3)

Models accelerometer measurement uncertainty.

$$\mathbf{R} = \sigma_a^2 \mathbf{I}_3$$

Accounts for: - Electronic noise - Quantization errors - Vibration - Thermal noise

Code:

```
// EKF2.cpp:41
R = Eigen::MatrixXd::Identity(3, 3) * 0.1; // σ_a² = 0.1
```

Typical value: σ_a ≈ 0.3 m/s² (for MEMS accelerometer)

Adaptive R (not implemented in EKF2, but mentioned in paper): - If ||a|| ≠ g → increase R (body is moving) - If ||a|| ≈ g → use normal R (body at rest)

Source: Sabatini (2006), Equations (10), (12)-(15)

EKF Algorithm

Initialization

State initialization:

$$\mathbf{x}_0 = \begin{bmatrix} \mathbf{q}_0 \\ \mathbf{0} \end{bmatrix}$$

Where q_0 is computed from initial accelerometer reading:

$$\phi_0 = \text{atan2}(a_y, a_z)$$

$$\theta_0 = \text{atan2}(-a_x, \sqrt{a_y^2 + a_z^2})$$

Euler to quaternion (with ψ = 0):

$$\begin{aligned} q_0 &= \cos(\phi_0/2)\cos(\theta_0/2) \\ q_1 &= \sin(\phi_0/2)\cos(\theta_0/2) \\ q_2 &= \cos(\phi_0/2)\sin(\theta_0/2) \\ q_3 &= -\sin(\phi_0/2)\sin(\theta_0/2) \end{aligned}$$

Covariance initialization:

$$\mathbf{P}_0 = \begin{bmatrix} 0.1\mathbf{I}_4 & \mathbf{0} \\ \mathbf{0} & 0.01\mathbf{I}_3 \end{bmatrix}$$

Code:

```cpp
// EKF2.cpp:5-42
EKF2::EKF2(double dt, const Eigen::Vector3d& initial_accel)
```

Source: Gebre-Egziabher et al. (2000) [10]

Prediction Step

1. State prediction (a priori estimate):

$$\mathbf{x}_k^- = \mathbf{f}(\mathbf{x}_{k-1}^+, \omega_k)$$

2. Quaternion normalization:

$$\mathbf{q}_k^- \leftarrow \frac{\mathbf{q}_k^-}{||\mathbf{q}_k^-||}$$

3. Compute Jacobian F_k

4. Covariance prediction:

$$\mathbf{P}_k^- = \mathbf{F}_k \mathbf{P}_{k-1}^+ \mathbf{F}_k^T + \mathbf{Q}$$

Code:

```
// EKF2.cpp:44-72
void EKF2::predict(const Eigen::Vector3d& gyro)
```

Update Step

1. Compute predicted measurement:

$$\mathbf{h}_k = \mathbf{R}^T(\mathbf{q}_k^-)\mathbf{g}^n$$

2. Compute innovation:

$$\mathbf{y}_k = \mathbf{z}_k - \mathbf{h}_k$$

3. Compute Jacobian H_k

4. Innovation covariance:

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{R}$$

5. Kalman gain:

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^T \mathbf{S}_k^{-1}$$

6. State update (a posteriori estimate):

$$\mathbf{x}_k^+ = \mathbf{x}_k^- + \mathbf{K}_k \mathbf{y}_k$$

7. Quaternion normalization:

$$\mathbf{q}_k^+ \leftarrow \frac{\mathbf{q}_k^+}{||\mathbf{q}_k^+||}$$

8. Covariance update:

$$\mathbf{P}_k^+ = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)\mathbf{P}_k^-$$

Code:

```
// EKF2.cpp:74-95
void EKF2::update(const Eigen::Vector3d& accel)
```

Source: Maybeck (1979) [28]; Sabatini (2006), Fig. 1

Symbol Glossary

State Variables

Symbol	Dimension	Description	Units
$\mathbf{x}$	7×1	State vector	-
$\mathbf{q}$	4×1	Orientation quaternion (unit norm)	-
q_0	scalar	Quaternion scalar part (w)	-
q_1, q_2, q_3	scalars	Quaternion vector part (x,y,z)	-
$\mathbf{b}$	3×1	Gyroscope bias vector	rad/s
$\mathbf{P}$	7×7	State error covariance matrix	-

Measurements

Symbol	Dimension	Description	Units
$\boldsymbol{\omega}$	3×1	Angular velocity (gyro measurement)	rad/s
$\mathbf{a}$	3×1	Acceleration (accelerometer reading)	m/s^2
$\mathbf{z}$	3×1	Normalized accelerometer measurement	-
$\mathbf{h}$	3×1	Predicted measurement	-
$\mathbf{y}$	3×1	Innovation (measurement residual)	-

Reference Frames

Symbol	Description
$\mathbf{g}^n$	Gravity vector in navigation frame (inertial)
$\mathbf{g}^b$	Gravity vector in body frame
^{n}	Superscript: navigation (world/inertial) frame
^{b}	Superscript: body (sensor) frame

Matrices

Symbol	Dimension	Description
$\mathbf{R}(q)$	3×3	Rotation matrix (DCM) from quaternion
$\mathbf{\Omega}(\omega)$	4×4	Quaternion multiplication matrix (skew-symmetric)
$\mathbf{F}$	7×7	State transition Jacobian
$\mathbf{H}$	3×7	Measurement Jacobian
$\mathbf{Q}$	7×7	Process noise covariance
$\mathbf{R}$	3×3	Measurement noise covariance
$\mathbf{S}$	3×3	Innovation covariance
$\mathbf{K}$	7×3	Kalman gain

Time Indices

Symbol	Description
k	Discrete time step index
x_k^-	A priori estimate (before measurement)
x_k^+	A posteriori estimate (after measurement)
Δt	Sampling interval

Noise Variables

Symbol	Description	Distribution
$\mathbf{v}_g$	Gyro measurement noise	$\mathcal{N}(0, \Sigma_g)$
$\mathbf{v}_a$	Accelerometer measurement noise	$\mathcal{N}(0, \Sigma_a)$
$\mathbf{w}_q$	Quaternion process noise	$\mathcal{N}(0, Q_q)$
$\mathbf{w}_b$	Bias process noise	$\mathcal{N}(0, Q_b)$
σ_q	Quaternion process noise std dev	-
σ_b	Bias process noise std dev	rad/s
σ_a	Accelerometer noise std dev	m/s^2

Euler Angles

Symbol	Description	Range
φ	Roll angle	$[-\pi, \pi]$
θ	Pitch angle	$[-\pi/2, \pi/2]$
ψ	Yaw angle	$[-\pi, \pi]$

References

Primary References

[Sabatini 2006] Sabatini, A.M. (2006). "Quaternion-Based Extended Kalman Filter for Determining Orientation by Inertial and Magnetic Sensing." *IEEE Transactions on Biomedical Engineering*, 53(7), 1346-1356.

Quaternion Mathematics

[8] Chou, J.C.K. (1992). "Quaternion kinematic and dynamic differential equations." *IEEE Transactions on Robotics and Automation*, 8(1), 53-64.

[9] Kirtley, C. (2001). "Summary: Quaternions vs. Euler angles." BIOMCH-L Discussion.

[27] Choukroun, D. (2003). "Novel methods for attitude determination using vector observations." Ph.D. thesis, Technion, Israel Institute of Technology.

Kalman Filtering

[28] Maybeck, P.S. (1979). *Stochastic Models, Estimation and Control*. Academic Press, New York.

[29] Bortz, J.E. (1971). "A new mathematical formulation for strapdown inertial navigation." *IEEE Transactions on Aerospace and Electronic Systems*, AES-7(1), 61-66.

Related EKF Implementations

[7] Marins, J.L., Yun, X., Bachmann, E.R., McGhee, R.B., & Zyda, M.J. (2001). "An extended Kalman filter for quaternion-based orientation estimation using MARG sensors." *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2003-2011.

[10] Gebre-Egziabher, D., Elkaim, G.H., Powell, J.D., & Parkinson, B.W. (2000). "A gyro-free quaternion-based attitude determination

system suitable for implementation using low cost sensors." *Proc. IEEE Position, Location and Navigation Symp.*, 185-192.

[11] Foxlin, E., Harrington, M., & Altshuler, Y. (1998). "Miniature 6-DOF inertial system for tracking HMDs." *Proc. SPIE*, 3362, 1-15.

Inertial Sensor Calibration

[19] Ferraris, F., Grimaldi, U., & Parvis, M. (1995). "Procedure for effortless in-field calibration of three-axis rate gyros and accelerometers." *Sensors and Materials*, 7, 311-330.

[20] Gebre-Egziabher, D., Elkaim, G.H., Powell, J.D., & Parkinson, B.W. (2001). "A non-linear, two-step estimation algorithm for calibrating solid-state strapdown magnetometers." *Proc. Int. Conf. Integrated Navigation Systems*, 290-297.

Human Motion Applications

[6] Luinge, H.J. & Veltink, P.H. (2004). "Inclination measurement of human movement using a 3-D accelerometer with autocalibration." *IEEE Trans. Neural Syst. Rehab. Eng.*, 12(1), 112-121.

[21] Sabatini, A.M. (2005). "Quaternion based strap-down integration method for applications of inertial sensing to gait analysis." *Med. Biol. Eng. Comput.*, 42, 97-105.

[24] Sabatini, A.M., Martelloni, C., Scapellato, S., & Cavallo, F. (2005). "Assessment of walking features from foot inertial sensing." *IEEE Trans. Biomed. Eng.*, 52(3), 486-494.

Appendix: Code-to-Math Mapping

State Vector Access

```
// Math: x = [q₀, q₁, q₂, q₃, bₓ, b_y, b_z]ᵀ
Eigen::VectorXd x;                    // 7×1
Eigen::Vector4d q = x.head<4>();    // q = [q₀, q₁, q₂, q₃]ᵀ
Eigen::Vector3d b = x.tail<3>();    // b = [bₓ, b_y, b_z]ᵀ
```

Quaternion Kinematics

```
// Math: q̇ = ½ Ω(ω) q
// Discrete: q_{k+1} = q_k + (Δt/2) Ω(ω̃) q_k

Eigen::Vector3d w = gyro - bias;    // ω̃ = ω_measured - b
```

```
Eigen::Matrix4d Omega;              // Build Ω matrix
Eigen::Vector4d q_new = q + 0.5 * dt * Omega * q;
```

Rotation Matrix

```
// Math: R(q) = [formula from equation]
Eigen::Matrix3d R = quaternionToRotationMatrix(q);

// Math: h = Rᵀ g^n
Eigen::Vector3d h = R.transpose() * g_n;
```

Jacobians

```
// Math: F = ∂f/∂x (7×7)
Eigen::MatrixXd F = computeF(w);

// Math: H = ∂h/∂x (3×7)
Eigen::MatrixXd H = computeH();
```

EKF Equations

```
// Prediction
P = F * P * F.transpose() + Q;      // P⁻ = FPFᵀ + Q

// Update
S = H * P * H.transpose() + R;      // S = HPHᵀ + R
K = P * H.transpose() * S.inverse(); // K = PHᵀS⁻¹
x = x + K * y;                       // x⁺ = x⁻ + Ky
P = (I - K * H) * P;                 // P⁺ = (I - KH)P⁻
```

End of Mathematical Documentation

This document provides a complete mathematical reference for the EKF2 quaternion-based Extended Kalman Filter implementation. All equations are traceable to source papers and validated through implementation.