

Quaternion Algebra and Rotations

Mark Pedley and Michael Stanley

Contents

1	Introduction	3
1.1	Summary	3
1.2	Terminology	3
1.3	Software Functions	4
2	Quaternion Algebra	5
2.1	Introduction	5
2.2	Equality of Two Quaternions	5
2.3	Addition of Two Quaternions	6
2.4	Product of Two Quaternions	6
2.5	Product of Quaternion and Scalar	7
2.6	Product of a Quaternion with a Vector	7
2.7	Quaternion Conjugate	8
2.8	Quaternion Norm	9
2.9	Quaternion Inverse and Division	10
2.10	Vector Representation of Quaternion Product	10
3	Rotation Quaternion	11
3.1	Definition in Terms of Rotation Vector	11
3.2	Equivalence of Rotation Quaternion and Rotation Matrix	11
3.3	Inverse Rotation Quaternion	12
3.4	Product of Rotation Quaternions	12
3.5	Negation of Rotation Axis and Angle	12
3.6	Coordinate Frame Rotation Standard	13
4	Converting between Quaternion and Rotation Matrix	13
4.1	Rotation Matrix from Quaternion	13
4.2	Quaternion from Rotation Matrix	14
5	Converting between Quaternion and Rotation Vector	16
5.1	Rotation Vector From Quaternion	16
5.2	Quaternion From Rotation Vector	16
6	Low-pass Filtering Orientation Quaternions	17
6.1	Introduction	17
6.2	Exponential Time Domain Low Pass Filter	17
6.3	Exponential Quaternion Low-pass Filter	18
7	Quaternion Derivative	19
7.1	Definition	19
7.2	Derivation for Rotation Quaternions	19

Document Number: AN5022
Revision 1.0, 9/2015



1 Introduction

1.1 Summary

Quaternions are hyper-complex numbers of rank 4 that were invented by William Rowan Hamilton in 1843. They are commonly used to simplify computation of 3-dimensional rotations.

This application note contains an introduction to quaternion algebra and its use to represent rotations. It documents the specific quaternion functions used in the [Freescale Sensor Fusion Library](#) within the file *orientation.c*.

The four components of a quaternion provide a more efficient means, both in storage and computation, to represent orientations as compared to the nine components of a rotation matrix. Quaternions are therefore used extensively in computer gaming and have, in consequence, also been adopted as a standard for representing orientation in sensor fusion applications.

Rotation matrices and rotation quaternions share the very important characteristic of being well behaved mathematically compared to the Euler angles (roll, pitch and yaw). The phenomenon of gimbal lock instability in strapdown sensor systems occurs in the Euler angle representation of orientation, typically at $\pm 90^\circ$ pitch, but is entirely absent in rotation matrix or rotation quaternion representations which are stable and continuous. Euler angles are therefore only used in the [Freescale Sensor Fusion Library](#) as an output representation of calculation performed using rotation matrices and rotation quaternions.

1.2 Terminology

Symbol	Definition
$a = a_0 + a_1\mathbf{i} + a_2\mathbf{j} + a_3\mathbf{k}$	Components of quaternion a
$a^* = a_0 - a_1\mathbf{i} - a_2\mathbf{j} - a_3\mathbf{k}$	Quaternion conjugate of a
$a = \{a_0, \mathbf{a}\}$	Representation of quaternion a in scalar a_0 and vector \mathbf{a} components
$a^{-1} = \frac{a^*}{N(a)^2}$	Inverse or reciprocal of quaternion a
a_0	Scalar component of quaternion a
$\mathbf{a} = a_1\mathbf{i} + a_2\mathbf{j} + a_3\mathbf{k}$	Vector component of quaternion a
$\mathbf{a} \cdot \mathbf{b}$	Scalar product of vectors \mathbf{a} and \mathbf{b}
$\mathbf{a} \times \mathbf{b}$	Vector product of vectors \mathbf{a} and \mathbf{b}
$\hat{\mathbf{n}}$	Unit vector representing rotation axis
$N(a) = \sqrt{a_0^2 + a_1^2 + a_2^2 + a_3^2}$	Norm or magnitude of quaternion a
q_x, q_y, q_z	Rotation quaternions about the x, y and z axes
R_x, R_y, R_z	Rotation matrices around x, y and z axes
r	Real number
z	Complex number
η	General rotation angle
ϕ	Roll angle
θ	Pitch angle
ψ	Yaw angle

Symbol	Definition
ρ	Compass heading angle

A unit quaternion is one in which the norm of the quaternion is of magnitude 1. Unit quaternions are commonly used to model rotations.

1.3 Software Functions

Table 1. Sensor Fusion software functions

Function	Description	Section
<code>void fqAeq1(struct fquaternion *pqA);</code>	Function sets the quaternion A to the unit or identity quaternion.	2.1
<code>void qAeqBxC (struct fquaternion *pqA, const struct fquaternion *pqB, const struct fquaternion *pqC);</code>	Function sets the quaternion A to the quaternion product BC.	2.4
<code>void qAeqAxB(struct fquaternion *pqA, const struct fquaternion *pqB);</code>	Function sets the quaternion A to the quaternion product AB.	2.4
<code>struct fquaternion qconjgAxB (const struct fquaternion *pqA, const struct fquaternion *pqB);</code>	Function returns the quaternion product A*B where A* is the conjugate of A.	2.7
<code>void fqAeqNormqA (struct fquaternion *pqA);</code>	Function normalizes the quaternion A.	2.8
<code>void fRotationMatrixFromQuaternion (float R[][3], const struct fquaternion *pq);</code>	Function computes a rotation matrix from a rotation quaternion.	4.1
<code>void fQuaternionFromRotationMatrix (float R[][3], struct fquaternion *pq);</code>	Function computes a rotation quaternion from a rotation matrix.	4.2
<code>void fRotationVectorDegFromQuaternion (struct fquaternion *pq, float rvecdeg[]);</code>	Computes the rotation vector from a rotation quaternion.	5.1

Function	Description	Section
<pre>void fQuaternionFromRotationVectorDeg (struct fquaternion *pq, const float rvecdeg[], float fscaling);</pre>	Computes the rotation quaternion from a rotation vector with scaling.	5.2
<pre>void fLPFOrientationQuaternion (struct fquaternion *pq, struct fquaternion *pLPq, float flpf, float fdeltat, float fOmega[])</pre>	Function low-pass filters the orientations represented by a sequence of quaternions.	6.2

2 Quaternion Algebra

2.1 Introduction

Quaternions form a class of four-component hyper-complex numbers.

Whereas a complex number z has two components, one real and the other imaginary:

$$z = a + bi \quad (1)$$

a quaternion a has four components:

$$a = a_0 + a_1 \mathbf{i} + a_2 \mathbf{j} + a_3 \mathbf{k} = \{a_0, a_1, a_2, a_3\} \quad (2)$$

where a_0, a_1, a_2 and a_3 are real numbers. The values \mathbf{i}, \mathbf{j} and \mathbf{k} are simply tags identifying the three vector components of the quaternion.

a_0 is termed the scalar component and $\mathbf{a} = a_1 \mathbf{i} + a_2 \mathbf{j} + a_3 \mathbf{k}$ is termed the vector component of the quaternion.

Equivalent representations of the quaternion in its scalar and vector components are:

$$a = a_0 + \mathbf{a} = \{a_0, \mathbf{a}\} = \left\{a_0, \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}\right\} = \left\{a_0, \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix}\right\} \quad (3)$$

If the scalar component a_0 is zero, the quaternion is termed a pure quaternion or vector. If the vector component \mathbf{a} is zero, then the quaternion is a real number. The quaternion with scalar component equal to one and vector component equal to zero is termed the identity quaternion.

2.2 Equality of Two Quaternions

Two quaternions a and b are equal if and only if all their components are equal:

$$a = b \Rightarrow a_0 = b_0 \text{ and } a_1 = b_1 \text{ and } a_2 = b_2 \text{ and } a_3 = b_3 \quad (4)$$

2.3 Addition of Two Quaternions

The addition of quaternions is defined as the addition of their four components:

$$a + b = (a_0 + b_0) + (a_1 + b_1)\mathbf{i} + (a_2 + b_2)\mathbf{j} + (a_3 + b_3)\mathbf{k} \quad (5)$$

A consequence of this definition is that quaternion addition is commutative and associative because the addition of real numbers is commutative and associative:

$$a + b = b + a \quad (6)$$

$$(a + b) + c = a + (b + c) \quad (7)$$

2.4 Product of Two Quaternions

The product of two quaternions is defined to be the distributive product of the components:

$$c = ab = (a_0 + a_1\mathbf{i} + a_2\mathbf{j} + a_3\mathbf{k})(b_0 + b_1\mathbf{i} + b_2\mathbf{j} + b_3\mathbf{k}) \quad (8)$$

$$\begin{aligned} &= a_0(b_0 + b_1\mathbf{i} + b_2\mathbf{j} + b_3\mathbf{k}) + a_1\mathbf{i}(b_0 + b_1\mathbf{i} + b_2\mathbf{j} + b_3\mathbf{k}) + a_2\mathbf{j}(b_0 + b_1\mathbf{i} + b_2\mathbf{j} + b_3\mathbf{k}) \\ &\quad + a_3\mathbf{k}(b_0 + b_1\mathbf{i} + b_2\mathbf{j} + b_3\mathbf{k}) \end{aligned} \quad (9)$$

$$\begin{aligned} &= (a_0b_0 + a_0b_1\mathbf{i} + a_0b_2\mathbf{j} + a_0b_3\mathbf{k}) + (a_1b_0\mathbf{i} + a_1b_1\mathbf{ii} + a_1b_2\mathbf{ij} + a_1b_3\mathbf{ik}) \\ &\quad + (a_2b_0\mathbf{j} + a_2b_1\mathbf{ji} + a_2b_2\mathbf{jj} + a_2b_3\mathbf{jk}) + (a_3b_0\mathbf{k} + a_3b_1\mathbf{ki} + a_3b_2\mathbf{kj} + a_3b_3\mathbf{kk}) \end{aligned} \quad (9)$$

The products of components of a quaternion are defined to satisfy, where r is any real number:

$$r\mathbf{i} = \mathbf{i}r \quad (10)$$

$$r\mathbf{j} = \mathbf{j}r \quad (11)$$

$$r\mathbf{k} = \mathbf{k}r \quad (12)$$

$$\mathbf{ii} = \mathbf{jj} = \mathbf{kk} = -1 \quad (13)$$

$$\mathbf{ij} = -\mathbf{ji} = \mathbf{k} \quad (14)$$

$$\mathbf{jk} = -\mathbf{kj} = \mathbf{i} \quad (15)$$

$$\mathbf{ki} = -\mathbf{ik} = \mathbf{j} \quad (16)$$

A consequence of equations (13) to (16) is that:

$$(ij)k = i(jk) = -1 \quad (17)$$

Substitution of equations (10) to (17) into equation (9) simplifies the quaternion product to:

$$\begin{aligned} c = ab = & (a_0b_0 - a_1b_1 - a_2b_2 - a_3b_3) + (a_0b_1 + a_1b_0 + a_2b_3 - a_3b_2)i \\ & + (a_0b_2 - a_1b_3 + a_2b_0 + a_3b_1)j + (a_0b_3 + a_1b_2 - a_2b_1 + a_3b_0)k \end{aligned} \quad (19)$$

The four components of the product quaternion c are, therefore:

$$c_0 = a_0b_0 - a_1b_1 - a_2b_2 - a_3b_3 \quad (18)$$

$$c_1 = a_0b_1 + a_1b_0 + a_2b_3 - a_3b_2 \quad (19)$$

$$c_2 = a_0b_2 - a_1b_3 + a_2b_0 + a_3b_1 \quad (20)$$

$$c_3 = a_0b_3 + a_1b_2 - a_2b_1 + a_3b_0 \quad (21)$$

Examination of equations (18) to (21) shows that quaternion multiplication does not commute:

$$ab \neq ba \quad (22)$$

Brute force evaluation proves that quaternion multiplication is associative:

$$(ab)c - a(bc) = \{(a_0 + a_1i + a_2j + a_3k)(b_0 + b_1i + b_2j + b_3k)\}(c_0 + c_1i + c_2j + c_3k) \quad (26)$$

$$- (a_0 + a_1i + a_2j + a_3k)\{(b_0 + b_1i + b_2j + b_3k)(c_0 + c_1i + c_2j + c_3k)\} = 0$$

$$\Rightarrow (ab)c = a(bc) \quad (23)$$

2.5 Product of Quaternion and Scalar

A special case of the quaternion product occurs when one of the quaternions has zero vector components and is a scalar. In this case the multiplication does commute.

If a is a scalar so that $a = a_0$ then:

$$ab = a_0b_0 + a_0b_1i + a_0b_2j + a_0b_3k \quad (24)$$

$$ba = b_0a_0 + b_1a_0i + b_2a_0j + b_3a_0k = a_0b_0 + a_0b_1i + a_0b_2j + a_0b_3k = ab \quad (25)$$

$$\Rightarrow ab = ba \text{ iff } a \text{ is a scalar} \quad (26)$$

2.6 Product of a Quaternion with a Vector

The product of a quaternion a with a vector quaternion b has non-zero scalar component and is therefore a general quaternion and not another vector quaternion:

$$a\mathbf{b} = (-a_1b_1 - a_2b_2 - a_3b_3) + (a_0b_1 + a_2b_3 - a_3b_2)\mathbf{i} + (a_0b_2 - a_1b_3 + a_3b_1)\mathbf{j} + (a_0b_3 + a_1b_2 - a_2b_1)\mathbf{k} \quad (27)$$

$$b\mathbf{a} = (-b_1a_1 - b_2a_2 - b_3a_3) + (b_1a_0 + b_2a_3 - b_3a_2)\mathbf{i} + (-b_1a_3 + b_2a_0 + b_3a_1)\mathbf{j} + (b_1a_2 - b_2a_1 + b_3a_0)\mathbf{k} \quad (28)$$

Inspection of equations (27) and (28) shows that the product of the quaternion a and vector \mathbf{b} does not commute:

$$a\mathbf{b} \neq b\mathbf{a} \quad (29)$$

2.7 Quaternion Conjugate

The quaternion conjugate a^* is defined as:

$$a^* = a_0 - a_1\mathbf{i} - a_2\mathbf{j} - a_3\mathbf{k} \quad (30)$$

From the definition of the quaternion product ab , it can be shown that $(ab)^* = b^*a^*$:

$$\begin{aligned} (ab)^* &= (a_0b_0 - a_1b_1 - a_2b_2 - a_3b_3) - (a_0b_1 + a_1b_0 + a_2b_3 - a_3b_2)\mathbf{i} \\ &\quad - (a_0b_2 - a_1b_3 + a_2b_0 + a_3b_1)\mathbf{j} - (a_0b_3 + a_1b_2 - a_2b_1 + a_3b_0)\mathbf{k} \end{aligned} \quad (31)$$

The product b^*a^* evaluates to:

$$b^*a^* = (b_0 - b_1\mathbf{i} - b_2\mathbf{j} - b_3\mathbf{k})(a_0 - a_1\mathbf{i} - a_2\mathbf{j} - a_3\mathbf{k}) \quad (32)$$

$$\begin{aligned} &= b_0a_0 - b_0a_1\mathbf{i} - b_0a_2\mathbf{j} - b_0a_3\mathbf{k} - b_1a_0 + b_1a_1\mathbf{i} + b_1a_2\mathbf{j} + b_1a_3\mathbf{k} - b_2a_0 + b_2a_1\mathbf{i} + b_2a_2\mathbf{j} \\ &\quad + b_2a_3\mathbf{k} - b_3a_0 + b_3a_1\mathbf{i} + b_3a_2\mathbf{j} + b_3a_3\mathbf{k} \end{aligned} \quad (33)$$

$$\begin{aligned} &= b_0a_0 - b_0a_1\mathbf{i} - b_0a_2\mathbf{j} - b_0a_3\mathbf{k} - b_1a_0 + b_1a_1\mathbf{i} + b_1a_2\mathbf{j} + b_1a_3\mathbf{k} - b_2a_0 + b_2a_1\mathbf{i} + b_2a_2\mathbf{j} \\ &\quad + b_2a_3\mathbf{k} - b_3a_0 + b_3a_1\mathbf{i} + b_3a_2\mathbf{j} + b_3a_3\mathbf{k} \end{aligned} \quad (34)$$

$$\begin{aligned} &= (b_0a_0 - b_1a_1 - b_2a_2 - b_3a_3) - (b_1a_0 + b_0a_1 + b_3a_2 - b_2a_3)\mathbf{i} - (b_2a_0 - b_3a_1 + b_0a_2 + b_1a_3)\mathbf{j} \\ &\quad - (b_3a_0 + b_2a_1 - b_1a_2 + b_0a_3)\mathbf{k} \end{aligned} \quad (35)$$

$$\Rightarrow (ab)^* = b^*a^* \quad (36)$$

Simple extension to higher order products results in:

$$(abc \dots z)^* = z^*(abc \dots)^* = z^* \dots c^*b^*a^* \quad (37)$$

The sum of the quaternion a and its conjugate a^* is the scalar $2a_0$:

$$a + a^* = (a_0 + a_1\mathbf{i} + a_2\mathbf{j} + a_3\mathbf{k}) + (a_0 - a_1\mathbf{i} - a_2\mathbf{j} - a_3\mathbf{k}) = 2a_0 \quad (38)$$

2.8 Quaternion Norm

The quaternion norm or magnitude $N(a)$ is defined as:

$$N(a) = \sqrt{a_0^2 + a_1^2 + a_2^2 + a_3^2} \quad (39)$$

The two products a^*a and aa^* of a quaternion a with its conjugate a^* evaluate to the square of the norm:

$$a^*a = (a_0 - a_1\mathbf{i} - a_2\mathbf{j} - a_3\mathbf{k})(a_0 + a_1\mathbf{i} + a_2\mathbf{j} + a_3\mathbf{k}) \quad (42)$$

$$\begin{aligned} &= a_0a_0 + a_0a_1\mathbf{i} + a_0a_2\mathbf{j} + a_0a_3\mathbf{k} - a_1a_0\mathbf{i} + a_1a_1 - a_1a_2\mathbf{k} + a_1a_3\mathbf{j} - a_2a_0\mathbf{j} + a_2a_1\mathbf{k} + a_2a_2 \\ &\quad - a_2a_3\mathbf{i} - a_3a_0\mathbf{k} - a_3a_1\mathbf{j} + a_3a_2\mathbf{i} + a_3a_3 \end{aligned} \quad (40)$$

$$= a_0^2 + a_1^2 + a_2^2 + a_3^2 = N(a)^2 \quad (41)$$

$$aa^* = (a_0 + a_1\mathbf{i} + a_2\mathbf{j} + a_3\mathbf{k})(a_0 - a_1\mathbf{i} - a_2\mathbf{j} - a_3\mathbf{k}) \quad (42)$$

$$\begin{aligned} &= a_0a_0 - a_0a_1\mathbf{i} - a_0a_2\mathbf{j} - a_0a_3\mathbf{k} + a_1a_0\mathbf{i} + a_1a_1 - a_1a_2\mathbf{k} + a_1a_3\mathbf{j} + a_2a_0\mathbf{j} + a_2a_1\mathbf{k} + a_2a_2 \\ &\quad - a_2a_3\mathbf{i} + a_3a_0\mathbf{k} - a_3a_1\mathbf{j} + a_3a_2\mathbf{i} + a_3a_3 \end{aligned} \quad (43)$$

$$= a_0^2 + a_1^2 + a_2^2 + a_3^2 = N(a)^2 \quad (44)$$

$$\Rightarrow N(a) = \sqrt{a_0^2 + a_1^2 + a_2^2 + a_3^2} = \sqrt{a^*a} = \sqrt{aa^*} \quad (45)$$

The norm of a quaternion conjugate equals the norm of the quaternion:

$$N(a^*) = \sqrt{a_0^2 + (-a_1)^2 + (-a_2)^2 + (-a_3)^2} = \sqrt{a_0^2 + a_1^2 + a_2^2 + a_3^2} = N(a) \quad (46)$$

The norm of the product of two quaternions is the product of the individual quaternion norms:

$$N(ab) = \sqrt{(a_0b_0 - a_1b_1 - a_2b_2 - a_3b_3)^2 + (a_0b_1 + a_1b_0 + a_2b_3 - a_3b_2)^2 + (a_0b_2 - a_1b_3 + a_2b_0 + a_3b_1)^2 + (a_0b_3 + a_1b_2 - a_2b_1 + a_3b_0)^2} \quad (47)$$

$$= \sqrt{(a_0^2 + a_1^2 + a_2^2 + a_3^2)(b_0^2 + b_1^2 + b_2^2 + b_3^2)} = N(a)N(b) \quad (48)$$

2.9 Quaternion Inverse and Division

The quaternion inverse a^{-1} is defined to be the quaternion which satisfies:

$$aa^{-1} = a^{-1}a = 1 \quad (49)$$

Pre- and post-multiplication of the quaternion a by $\frac{a^*}{N(a)^2}$ evaluates to:

$$\frac{1}{N(a)^2} (a_0 - a_1\mathbf{i} - a_2\mathbf{j} - a_3\mathbf{k})(a_0 + a_1\mathbf{i} + a_2\mathbf{j} + a_3\mathbf{k}) = \frac{(a_0^2 + a_1^2 + a_2^2 + a_3^2)}{N(a)^2} = 1 \quad (50)$$

$$(a_0 + a_1\mathbf{i} + a_2\mathbf{j} + a_3\mathbf{k}) \frac{1}{N(a)^2} (a_0 - a_1\mathbf{i} - a_2\mathbf{j} - a_3\mathbf{k}) = \frac{(a_0^2 + a_1^2 + a_2^2 + a_3^2)}{N(a)^2} = 1 \quad (51)$$

The quaternion inverse a^{-1} is, therefore, for all quaternions with nonzero norm:

$$a^{-1} = \frac{a^*}{N(a)^2} = \left(\frac{a_0}{N(a)^2} \right) - \left(\frac{a_1}{N(a)^2} \right) \mathbf{i} - \left(\frac{a_2}{N(a)^2} \right) \mathbf{j} - \left(\frac{a_3}{N(a)^2} \right) \mathbf{k} \quad (52)$$

The norm of a quaternion inverse equals the reciprocal of the quaternion norm. The norm and reciprocation operations, therefore, commute:

$$N(a^{-1}) = N\left(\frac{a^*}{N(a)^2}\right) = \left(\frac{1}{N(a)^2}\right) N(a^*) = \left(\frac{N(a)}{N(a)^2}\right) = \left(\frac{1}{N(a)}\right) \quad (53)$$

2.10 Vector Representation of Quaternion Product

The scalar and vector products between the two vectors $\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}$ and $\mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$ are defined as:

$$\mathbf{a} \cdot \mathbf{b} = a_1b_1 + a_2b_2 + a_3b_3 \quad (54)$$

$$\mathbf{a} \times \mathbf{b} = \begin{pmatrix} a_2b_3 - a_3b_2 \\ a_3b_1 - a_1b_3 \\ a_1b_2 - a_2b_1 \end{pmatrix} \quad (55)$$

The product between two quaternions $a = (a_0, \mathbf{a})$ and $b = (b_0, \mathbf{b})$ can be written in terms of the scalar and vector products on their vector components. Direction expansion of the scalar and vector expression below and comparison with equation (9) shows it equals the quaternion product ab :

$$\{a_0 b_0 - \mathbf{a} \cdot \mathbf{b}, a_0 \mathbf{b} + b_0 \mathbf{a} + \mathbf{a} \times \mathbf{b}\} = \left\{ a_0 b_0 - a_1 b_1 + a_2 b_2 + a_3 b_3, a_0 \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} + b_0 \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} + \begin{pmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{pmatrix} \right\} \quad (56)$$

$$= \left\{ a_0 b_0 - a_1 b_1 + a_2 b_2 + a_3 b_3, \begin{pmatrix} a_0 b_1 + a_1 b_0 + a_2 b_3 - a_3 b_2 \\ a_0 b_2 - a_1 b_3 + a_2 b_0 + a_3 b_1 \\ a_0 b_3 + a_1 b_2 - a_2 b_1 + a_3 b_0 \end{pmatrix} \right\} = ab \quad (57)$$

3 Rotation Quaternion

3.1 Definition in Terms of Rotation Vector

The rotation quaternion q for a rotation of the coordinate system about normalized rotation axis $\hat{\mathbf{n}}$ and by angle η is defined to be:

$$q = \cos\left(\frac{\eta}{2}\right) + \hat{\mathbf{n}} \sin\left(\frac{\eta}{2}\right) \quad (58)$$

q is a unit quaternion because its norm equals 1:

$$N(q) = \cos^2\left(\frac{\eta}{2}\right) + \sin^2\left(\frac{\eta}{2}\right) (\hat{n}_x^2 + \hat{n}_y^2 + \hat{n}_z^2) = 1 \quad (59)$$

The value of this definition of the rotation quaternion is shown in the next section which proves that post- and pre-multiplication of a vector \mathbf{v} by the quaternion q and its conjugate q^* is equivalent to multiplication of the vector by the corresponding rotation matrix \mathbf{R} :

$$\mathbf{R}\mathbf{v} = q^* \mathbf{v} q \quad (60)$$

3.2 Equivalence of Rotation Quaternion and Rotation Matrix

The general rotation matrix \mathbf{R} , which transforms a vector as a result of a rotation of the coordinate system around the axis $\hat{\mathbf{n}}$ by angle η , is termed the Rodrigues rotation matrix and has form:

$$\mathbf{R} = \begin{pmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{pmatrix} = \begin{pmatrix} \hat{n}_x^2 + (1 - \hat{n}_x^2) \cos \eta & \hat{n}_x \hat{n}_y (1 - \cos \eta) + \hat{n}_z \sin \eta & \hat{n}_x \hat{n}_z (1 - \cos \eta) - \hat{n}_y \sin \eta \\ \hat{n}_x \hat{n}_y (1 - \cos \eta) - \hat{n}_z \sin \eta & \hat{n}_y^2 + (1 - \hat{n}_y^2) \cos \eta & \hat{n}_y \hat{n}_z (1 - \cos \eta) + \hat{n}_x \sin \eta \\ \hat{n}_x \hat{n}_z (1 - \cos \eta) + \hat{n}_y \sin \eta & \hat{n}_y \hat{n}_z (1 - \cos \eta) - \hat{n}_x \sin \eta & \hat{n}_z^2 + (1 - \hat{n}_z^2) \cos \eta \end{pmatrix} \quad (66)$$

For any vector \mathbf{v} , the left side of equation (60) evaluates to:

$$\mathbf{R}\mathbf{v} = \begin{pmatrix} \hat{n}_x^2 + (1 - \hat{n}_x^2) \cos \eta & \hat{n}_x \hat{n}_y (1 - \cos \eta) + \hat{n}_z \sin \eta & \hat{n}_x \hat{n}_z (1 - \cos \eta) - \hat{n}_y \sin \eta \\ \hat{n}_x \hat{n}_y (1 - \cos \eta) - \hat{n}_z \sin \eta & \hat{n}_y^2 + (1 - \hat{n}_y^2) \cos \eta & \hat{n}_y \hat{n}_z (1 - \cos \eta) + \hat{n}_x \sin \eta \\ \hat{n}_x \hat{n}_z (1 - \cos \eta) + \hat{n}_y \sin \eta & \hat{n}_y \hat{n}_z (1 - \cos \eta) - \hat{n}_x \sin \eta & \hat{n}_z^2 + (1 - \hat{n}_z^2) \cos \eta \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \quad (67)$$

$$= \begin{pmatrix} \{\hat{n}_x^2 + (1 - \hat{n}_x^2) \cos \eta\} v_x + \{\hat{n}_x \hat{n}_y (1 - \cos \eta) + \hat{n}_z \sin \eta\} v_y + \{\hat{n}_x \hat{n}_z (1 - \cos \eta) - \hat{n}_y \sin \eta\} v_z \\ \{\hat{n}_x \hat{n}_y (1 - \cos \eta) - \hat{n}_z \sin \eta\} v_x + \{\hat{n}_y^2 + (1 - \hat{n}_y^2) \cos \eta\} v_y + \{\hat{n}_y \hat{n}_z (1 - \cos \eta) + \hat{n}_x \sin \eta\} v_z \\ \{\hat{n}_x \hat{n}_z (1 - \cos \eta) + \hat{n}_y \sin \eta\} v_x + \{\hat{n}_y \hat{n}_z (1 - \cos \eta) - \hat{n}_x \sin \eta\} v_y + \{\hat{n}_z^2 + (1 - \hat{n}_z^2) \cos \eta\} v_z \end{pmatrix} \quad (68)$$

The right side of equation (60) evaluates to:

$$q^* v q = \left(\cos\left(\frac{\eta}{2}\right) - \hat{n} \sin\left(\frac{\eta}{2}\right) \right) v \left(\cos\left(\frac{\eta}{2}\right) + \hat{n} \sin\left(\frac{\eta}{2}\right) \right) \quad (69)$$

$$= \left(\cos\left(\frac{\eta}{2}\right) - \hat{n}_x \sin\left(\frac{\eta}{2}\right) \mathbf{i} - \hat{n}_y \sin\left(\frac{\eta}{2}\right) \mathbf{j} - \hat{n}_z \sin\left(\frac{\eta}{2}\right) \mathbf{k} \right) (v_x \mathbf{i} + v_y \mathbf{j} + v_z \mathbf{k}) \left(\cos\left(\frac{\eta}{2}\right) + \hat{n}_x \sin\left(\frac{\eta}{2}\right) \mathbf{i} + \hat{n}_y \sin\left(\frac{\eta}{2}\right) \mathbf{j} + \hat{n}_z \sin\left(\frac{\eta}{2}\right) \mathbf{k} \right) \quad (70)$$

$$= \begin{pmatrix} \{\hat{n}_x^2 + (1 - \hat{n}_x^2) \cos \eta\} v_x + \{\hat{n}_x \hat{n}_y (1 - \cos \eta) + \hat{n}_z \sin \eta\} v_y + \{\hat{n}_x \hat{n}_z (1 - \cos \eta) - \hat{n}_y \sin \eta\} v_z \\ \{\hat{n}_x \hat{n}_y (1 - \cos \eta) - \hat{n}_z \sin \eta\} v_x + \{\hat{n}_y^2 + (1 - \hat{n}_y^2) \cos \eta\} v_y + \{\hat{n}_y \hat{n}_z (1 - \cos \eta) + \hat{n}_x \sin \eta\} v_z \\ \{\hat{n}_x \hat{n}_z (1 - \cos \eta) + \hat{n}_y \sin \eta\} v_x + \{\hat{n}_y \hat{n}_z (1 - \cos \eta) - \hat{n}_x \sin \eta\} v_y + \{\hat{n}_z^2 + (1 - \hat{n}_z^2) \cos \eta\} v_z \end{pmatrix} \quad (71)$$

Equations (68) and (71) match, proving the identity in equation (60).

3.3 Inverse Rotation Quaternion

From the definition of q^* , it follows that q^* is the rotation operator about the same axis \hat{n} but by angle $-\eta$ and is, therefore, the inverse of the rotation quaternion q :

$$q^* = \cos\left(\frac{\eta}{2}\right) - \hat{n} \sin\left(\frac{\eta}{2}\right) = \cos\left(\frac{-\eta}{2}\right) + \hat{n} \sin\left(\frac{-\eta}{2}\right) \quad (72)$$

The inverse nature of the operators q and q^* can also be shown by direct evaluation:

$$q(q^* v q) q^* = (q q^*) v (q q^*) = v \quad (73)$$

$$q^*(q v q^*) q = (q^* q) v (q^* q) = v \quad (74)$$

3.4 Product of Rotation Quaternions

The result of successively applying rotation quaternions q_1 followed by q_2 through q_N to vector v is:

$$q_N^* \dots (q_2^* (q_1^* v q_1) q_2) \dots q_N = (q_N^* \dots q_2^* q_1^*) v (q_1 q_2 \dots q_N) \quad (75)$$

The rotation quaternion q , equivalent to the N successive rotations represented by quaternions q_1 to q_N , is therefore:

$$q = q_1 q_1 q_1 \dots q_N \quad (76)$$

3.5 Negation of Rotation Axis and Angle

Both a rotation matrix R and quaternion q are unchanged if both the rotation angle and axis are

simultaneously inverted:

$$\mathbf{R}(-\hat{\mathbf{n}}, -\eta) = \begin{pmatrix} \hat{n}_x^2 + (1 - \hat{n}_x^2)\cos(-\eta) & \hat{n}_x\hat{n}_y(1 - \cos(-\eta)) - \hat{n}_z\sin(-\eta) & \hat{n}_x\hat{n}_z(1 - \cos(-\eta)) + \hat{n}_y\sin(-\eta) \\ \hat{n}_x\hat{n}_y(1 - \cos(-\eta)) + \hat{n}_z\sin(-\eta) & \hat{n}_y^2 + (1 - \hat{n}_y^2)\cos(-\eta) & \hat{n}_y\hat{n}_z(1 - \cos(-\eta)) - \hat{n}_x\sin(-\eta) \\ \hat{n}_x\hat{n}_z(1 - \cos(-\eta)) - \hat{n}_y\sin(-\eta) & \hat{n}_y\hat{n}_z(1 - \cos(-\eta)) + \hat{n}_x\sin(-\eta) & \hat{n}_z^2 + (1 - \hat{n}_z^2)\cos(-\eta) \end{pmatrix} \quad (61)$$

$$= \begin{pmatrix} \hat{n}_x^2 + (1 - \hat{n}_x^2)\cos\eta & \hat{n}_x\hat{n}_y(1 - \cos\eta) + \hat{n}_z\sin\eta & \hat{n}_x\hat{n}_z(1 - \cos\eta) - \hat{n}_y\sin\eta \\ \hat{n}_x\hat{n}_y(1 - \cos\eta) - \hat{n}_z\sin\eta & \hat{n}_y^2 + (1 - \hat{n}_y^2)\cos\eta & \hat{n}_y\hat{n}_z(1 - \cos\eta) + \hat{n}_x\sin\eta \\ \hat{n}_x\hat{n}_z(1 - \cos\eta) + \hat{n}_y\sin\eta & \hat{n}_y\hat{n}_z(1 - \cos\eta) - \hat{n}_x\sin\eta & \hat{n}_z^2 + (1 - \hat{n}_z^2)\cos\eta \end{pmatrix} = \mathbf{R}(\hat{\mathbf{n}}, \eta) \quad (62)$$

$$q(-\hat{\mathbf{n}}, -\eta) = \cos\left(\frac{-\eta}{2}\right) - \hat{\mathbf{n}}\sin\left(\frac{-\eta}{2}\right) = \cos\left(\frac{\eta}{2}\right) + \hat{\mathbf{n}}\sin\left(\frac{\eta}{2}\right) = q(\hat{\mathbf{n}}, \eta) \quad (63)$$

Equations (61) and (63) state mathematically the result that a rotation about a given axis is equivalent to the negative rotation about the negated axis.

It is, therefore, conventional to constrain the scalar component q_0 of a rotation quaternion q to be non-negative. If a negative scalar component is detected then the entire quaternion, both scalar and vector components, can be safely negated.

3.6 Coordinate Frame Rotation Standard

Some texts define the quaternion rotation operator on vector \mathbf{v} to be $q\mathbf{v}q^*$ instead of $q^*\mathbf{v}q$. The explanation is that the operator $q^*\mathbf{v}q$ transforms the vector \mathbf{v} as a result of rotation of the coordinate system by angle η , whereas the operator $q\mathbf{v}q^*$ rotates the vector \mathbf{v} , by angle η in a fixed coordinate system. The standard used in this document and all Freescale Sensor Fusion Library software is that it is the coordinate system that is rotating, normally as a result of the device orientation changing, while the vector \mathbf{v} , which is typically the earth's gravitational or geomagnetic field, remains fixed in its frame.

4 Converting between Quaternion and Rotation Matrix

4.1 Rotation Matrix from Quaternion

Expanding equation (60) into its components gives the identity:

$$\begin{pmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} = (q_0 - q_1\mathbf{i} - q_2\mathbf{j} - q_3\mathbf{k})(v_x\mathbf{i} + v_y\mathbf{j} + v_z\mathbf{k})(q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}) \quad (72)$$

Expanding the right hand side and re-arranging gives:

$$\begin{pmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} = \begin{pmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2q_1q_2 + 2q_0q_3 & 2q_1q_3 - 2q_0q_2 \\ 2q_1q_2 - 2q_0q_3 & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2q_2q_3 + 2q_0q_1 \\ 2q_1q_3 + 2q_0q_2 & 2q_2q_3 - 2q_0q_1 & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \quad (73)$$

Because equation (73) holds for all vectors \mathbf{v} , it follows that:

$$\begin{pmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{pmatrix} = \begin{pmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2q_1q_2 + 2q_0q_3 & 2q_1q_3 - 2q_0q_2 \\ 2q_1q_2 - 2q_0q_3 & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2q_2q_3 + 2q_0q_1 \\ 2q_1q_3 + 2q_0q_2 & 2q_2q_3 - 2q_0q_1 & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{pmatrix} \quad (64)$$

The normalization constraint for a rotation quaternion is:

$$q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1 \quad (75)$$

Substituting equation (75) into equation (64) gives a slightly simpler expression for converting a rotation quaternion to a rotation matrix:

$$\begin{pmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{pmatrix} = \begin{pmatrix} 2(q_0^2 + q_1^2) - 1 & 2(q_1q_2 + q_0q_3) & 2(q_1q_3 - q_0q_2) \\ 2(q_1q_2 - q_0q_3) & 2(q_0^2 + q_2^2) - 1 & 2(q_2q_3 + q_0q_1) \\ 2(q_1q_3 + q_0q_2) & 2(q_2q_3 - q_0q_1) & 2(q_0^2 + q_3^2) - 1 \end{pmatrix} \quad (65)$$

4.2 Quaternion from Rotation Matrix

Equation (65) can also be used to determine the rotation quaternion from the rotation matrix. The procedure is straightforward except for rotations close to 180° where a fallback algorithm is needed to avoid numerical rounding errors.

The sum of 1 plus the trace of the rotation matrix evaluates to:

$$1 + \text{tr}(\mathbf{R}) = 1 + 2(q_0^2 + q_1^2) - 1 + 2(q_0^2 + q_2^2) - 1 + 2(q_0^2 + q_3^2) - 1 \quad (77)$$

$$1 + \text{tr}(\mathbf{R}) = 2q_0^2 + 2q_1^2 + 2q_0^2 + 2q_2^2 - 1 + 2q_0^2 + 2q_3^2 - 1 \quad (66)$$

$$1 + \text{tr}(\mathbf{R}) = (2q_0^2 + 2q_1^2 + 2q_2^2 + 2q_3^2) + 2q_0^2 - 1 + 2q_0^2 - 1 \quad (67)$$

Using the result that the quaternion has unit norm gives:

$$1 + \text{tr}(\mathbf{R}) = 4q_0^2 \quad (80)$$

q_0 is always nonnegative because negative q_0 in a rotation quaternion corresponds to a rotation angle greater than 180° which is equivalent to a negated rotation of less than 180° about the negated rotation axis.

The positive square root of equation (80) can, therefore, always be taken, giving:

$$q_0 = \frac{\sqrt{1 + \text{tr}(\mathbf{R})}}{2} = \frac{\sqrt{1 + R_{xx} + R_{yy} + R_{zz}}}{2} \quad (81)$$

Differencing elements across the diagonal gives the valid solution for all remaining elements of the

rotation quaternion except when $q_0 = 0$ at a 180° rotation:

$$R_{yz} - R_{zy} = 4q_0q_1 \Rightarrow q_1 = \frac{(R_{yz} - R_{zy})}{4q_0} \quad (68)$$

$$R_{zx} - R_{xz} = 4q_0q_2 \Rightarrow q_2 = \frac{(R_{zx} - R_{xz})}{4q_0} \quad (83)$$

$$R_{xy} - R_{yx} = 4q_0q_3 \Rightarrow q_3 = \frac{(R_{xy} - R_{yx})}{4q_0} \quad (69)$$

Equations (68) to (69) fail near 180° rotation about any axis because the rotation matrix becomes symmetric (giving a near-zero numerator) and the scalar quaternion component q_0 approaches zero (giving a near-zero denominator). The fallback algorithm in this case uses the elements on the leading diagonal to give:

$$R_{xx} = 2(q_0^2 + q_1^2) - 1 \Rightarrow q_1 = \pm \sqrt{\frac{1 + R_{xx}}{2} - q_0^2} \quad (70)$$

$$R_{yy} = 2(q_0^2 + q_2^2) - 1 \Rightarrow q_2 = \pm \sqrt{\frac{1 + R_{yy}}{2} - q_0^2} \quad (86)$$

$$R_{zz} = 2(q_0^2 + q_3^2) - 1 \Rightarrow q_3 = \pm \sqrt{\frac{1 + R_{zz}}{2} - q_0^2} \quad (71)$$

The unknown signs in equations (70) to (71) can be resolved by taking the signs of equations (68) to Error! Reference source not found. and using the fact that q_0 is always nonnegative:

$$\text{sign}(q_1) = \text{sign}(R_{yz} - R_{zy}) \quad (88)$$

$$\text{sign}(q_2) = \text{sign}(R_{zx} - R_{xz}) \quad (89)$$

$$\text{sign}(q_3) = \text{sign}(R_{xy} - R_{yx}) \quad (90)$$

5 Converting between Quaternion and Rotation Vector

5.1 Rotation Vector From Quaternion

The definition of the rotation quaternion q in equation (58) shows that it is closely linked to the equivalent rotation vector $\hat{n}\eta$ defined as the product of normalized rotation axis \hat{n} and the rotation angle η . Inverting the process to recover the rotation vector from the quaternion is straightforward.

$$q = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k} = \cos\left(\frac{\eta}{2}\right) + \hat{n}\sin\left(\frac{\eta}{2}\right) \quad (91)$$

Equating the scalar components results in:

$$q_0 = \cos\left(\frac{\eta}{2}\right) \Rightarrow \eta = 2\cos^{-1}(q_0) \quad (92)$$

Because q_0 varies between 0 and 1, the rotation angle η in equation (92) **Error! Reference source not found.** has the required range 0° to 180° .

Equating the remaining three components of the quaternion gives for the general case where $\sin\left(\frac{\eta}{2}\right)$ is non-zero:

$$q_1 = \sin\left(\frac{\eta}{2}\right)n_x \Rightarrow n_x = \frac{q_1}{\sin\left(\frac{\eta}{2}\right)} \quad (93)$$

$$q_2 = \sin\left(\frac{\eta}{2}\right)n_y \Rightarrow n_y = \frac{q_2}{\sin\left(\frac{\eta}{2}\right)} \quad (94)$$

$$q_3 = \sin\left(\frac{\eta}{2}\right)n_z \Rightarrow n_z = \frac{q_3}{\sin\left(\frac{\eta}{2}\right)} \quad (95)$$

For the case where $\sin\left(\frac{\eta}{2}\right) = 0$, the rotation angle η must be zero since η is in the range 0° to 180° . The rotation vector $\hat{n}\eta$ is then zero even though the rotation axis \hat{n} is undefined.

5.2 Quaternion From Rotation Vector

Equation (58) defines the rotation quaternion explicitly in terms of the rotation vector axis \hat{n} and rotation angle η .

$$q = \cos\left(\frac{\eta}{2}\right) + \hat{n}\sin\left(\frac{\eta}{2}\right) \quad (72)$$

6 Low-pass Filtering Orientation Quaternions

6.1 Introduction

The Kalman filter algorithms directly compute an optimal Kalman filter estimate of the orientation. The simpler accelerometer and magnetometer eCompass algorithms, however, require the explicit low-pass filtering of the stream of noisy orientation estimates whether in quaternion or rotation matrix forms. Low pass filtering of orientation quaternions is performed in the function `fLPFOrientationQuaternion` in file *orientation.c*.

The individual elements of orientation matrices and orientation quaternions should not be low pass filtered since the resulting low pass filtered matrix or quaternion is no longer a valid rotation matrix (with orthonormal row and column vectors) or valid rotation quaternion (with unit norm). Any low pass filtering on individual elements must therefore be followed by explicit re-normalizing of the rotation matrix or rotation quaternion. The results are never terribly satisfactory and the low pass filtered trajectories can be counter-intuitive. The preferred method is to use exponential filtering of quaternions as shown in the following subsections.

6.2 Exponential Time Domain Low Pass Filter

The difference equation for the single-pole low-pass filter in the time domain is:

$$y[n] = (1 - \alpha)y[n - 1] + \alpha x[n] \quad (96)$$

where:

$$0 < \alpha \leq 1 \quad (97)$$

The transfer function $H(z)$ is:

$$H(z) = \frac{\alpha}{1 - (1 - \alpha)z^{-1}} \quad (98)$$

with a single pole at $z = (1 - \alpha)$.

In the general case, an impulse at time zero $x[0] = 1$ gives the exponentially decaying output:

$$y[n] = (1 - \alpha)^n \quad (99)$$

The $\frac{1}{e}$ time constant N is then given by:

$$\frac{1}{e} = (1 - \alpha)^N \Rightarrow N = \frac{-1}{\ln(1 - \alpha)} \quad (100)$$

For small α :

$$N = \frac{1}{\alpha} \quad (101)$$

The time constant in samples is therefore approximately equal to the reciprocal of the filter coefficient α and the case $\alpha = 1$ corresponds to an all pass filter.

In C code, equation (96) is written as:

$$y_n += \alpha * (x_n - y_n); \quad (102)$$

An example of this filter is the line below taken from the function `fRun_6DOF_GB_BASIC` and used to filter the geomagnetic inclination angle:

```
// low pass filter the geomagnetic inclination angle with a simple exponential filter
pthisSV->fLPDelta += pthisSV->flpf * (pthisSV->fDelta - pthisSV->fLPDelta);
```

Equation (102) makes it clear that the low-pass estimate y_n is updated by α times the difference between the current input x_n and the previous low-pass filtered estimate y_n . The low-pass estimate y_n is therefore exponentially steered towards the input sequence x_n . The next section describes how the principle of exponential convergence to a noisy time varying input orientation can be applied to orientation quaternions.

6.3 Exponential Quaternion Low-pass Filter

The orientation space quaternion low pass filter used in the function `fLPFOrientationQuaternion` is analogous to the time domain exponential filter of the previous section in that it exponentially rotates the current low-pass filtered orientation quaternion towards the instantaneous, and therefore noisy, orientation quaternion.

The incremental rotation quaternion $\Delta q[n]$ required at iteration n to completely rotate the previous low pass filtered quaternion $q_{LP}[n-1]$ onto the instantaneous noisy quaternion $q[n]$ is given by:

$$q[n] = q_{LP}[n-1]\Delta q[n] \Rightarrow \Delta q[n] = q_{LP}^*[n-1]q[n] \quad (103)$$

The scalar and vector components of $\Delta q[n]$ are related to the angle η and axis \hat{n} between the low pass and instantaneous orientation estimates at time $[n]$ by:

$$\Delta q[n] = \cos\left(\frac{\eta}{2}\right) + \hat{n}\sin\left(\frac{\eta}{2}\right) \quad (104)$$

Applying the incremental quaternion unchanged results in the special case of the all-pass filter where $q_{LP}[n] = q[n]$:

$$q_{LP}[n] = q_{LP}[n-1]\Delta q[n] = q_{LP}[n-1]q_{LP}^*[n-1]q[n] = q[n] \quad (105)$$

Scaling the vector component by a constant factor α results in an incremental quaternion correction $\Delta q'[n]$ which exponentially steers the low pass filtered orientation quaternion onto the instantaneous quaternion with a time constant approximately equal to the reciprocal of the filter coefficient α :

$$\Delta q'[n] = \Delta q_0'[n] + \alpha \hat{n} \sin\left(\frac{\eta}{2}\right) \quad (106)$$

The scalar component $\Delta q_0'[n]$ is determined by the constraint that the norm of $\Delta q'[n]$ equals 1.

This low pass filter can be improved by making the filter coefficient α dependent on the relative angle η . α should be small when η is small in order to give a long time constant and a high degree of noise rejection but α should increase as η increases in order to give faster tracking of aggressive motion at the expense of less averaging.

The expression used in the function `fLPFOrientationQuaternion` is:

$$\alpha' = \alpha + (1 - \alpha)(1 - \Delta q_0[n]) = \alpha + (1 - \alpha) \left(1 - \cos\left(\frac{\eta}{2}\right)\right) \quad (107)$$

where α is the nominal filter coefficient and α' the actual filter coefficient used .

Using a second order Taylor expansion equation gives the variation of filter coefficient α' for small relative angles η and small α as:

$$\alpha' \approx \alpha + \frac{(1 - \alpha)\eta^2}{8} \approx \alpha + \frac{\eta^2}{8} \text{ for } \eta \text{ in radians} \quad (108)$$

α' therefore equals α in the limit of small angles η but increases quadratically as the divergence between the low pass and instantaneous orientation estimates increases resulting in a decreasing time constant and more rapid convergence as the platform dynamics become more aggressive.

As the discrepancy between the low pass and instantaneous orientation estimates reaches the maximum of 180° , $\Delta q_0[n]$ tends to zero and the filter coefficient α' asymptotes to the all pass case bringing the low pass filtered estimate back into immediate synchronization with the instantaneous orientation estimate.

$$\alpha' = 1 \quad (109)$$

7 Quaternion Derivative

7.1 Definition

The quaternion derivative is defined in the conventional manner as the limit:

$$\frac{dq(t)}{dt} = \dot{q}(t) = \lim_{\delta t \rightarrow 0} \left\{ \frac{q(t + \delta t) - q(t)}{\delta t} \right\} \quad (110)$$

$$\Rightarrow q(t + dt) = q(t) + dt\dot{q}(t) \quad (111)$$

7.2 Derivation for Rotation Quaternions

An orientation quaternion can also be propagated forward in time by computing the product of the current rotation quaternion $q(t)$ and the incremental rotation quaternion $\delta q(t)$:

$$q(t + \delta t) = q(t)\delta q(t) \quad (112)$$

With the assumption that incremental change in orientation results from a constant angular velocity ω over the time interval δt then from the definition of the rotation quaternion in terms of rotation angle:

$$\delta q = \{q_0, \delta q_1, \delta q_2, \delta q_3\} = \left\{q_0, \sin\left(\frac{\omega_x \delta t}{2}\right), \sin\left(\frac{\omega_y \delta t}{2}\right), \sin\left(\frac{\omega_z \delta t}{2}\right)\right\} \quad (113)$$

q_0 is determined by the requirement that the rotation quaternion δq be normalized.

In the limit of the interval δt becoming the infinitesimal dt :

$$dq = \{q_0, dq_1, dq_2, dq_3\} = \left\{1, \left(\frac{\omega_x dt}{2}\right), \left(\frac{\omega_y dt}{2}\right), \left(\frac{\omega_z dt}{2}\right)\right\} \quad (114)$$

and:

$$q(t + dt) = q(t)dq = q(t) \left\{1, \left(\frac{\omega_x dt}{2}\right), \left(\frac{\omega_y dt}{2}\right), \left(\frac{\omega_z dt}{2}\right)\right\} \quad (115)$$

Combining equations (111) and (115) gives:

$$q(t) + dt\dot{q}(t) = q(t) \left\{1, \left(\frac{\omega_x dt}{2}\right), \left(\frac{\omega_y dt}{2}\right), \left(\frac{\omega_z dt}{2}\right)\right\} \quad (116)$$

Using the result that quaternion multiplication is distributive gives the expression for the quaternion derivative in terms of angular velocity as:

$$q(t) + dt\dot{q}(t) = q(t) + \left(\frac{dt}{2}\right) q(t) \{0, \omega_x, \omega_y, \omega_z\} \quad (117)$$

$$\Rightarrow \dot{q}(t) = \left(\frac{1}{2}\right) q(t) \omega(t) \quad (118)$$

where $\omega(t)$ is the vector quaternion $\{0, \omega_x(t), \omega_y(t), \omega_z(t)\}$.

How to Reach Us:**Home Page:**freescale.com**Web Support:**freescale.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. Freescale reserves the right to make changes without further notice to any products herein.

Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages.

“Typical” parameters that may be provided in Freescale data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address:

freescale.com/SalesTermsandConditions.

Freescale and the Freescale logo, are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. All other product or service names are the property of their respective owners.

© 2015 Freescale Semiconductor, Inc.

ⁱ Some texts use the first three components of the quaternion (a_0 , a_1 and a_2) as the vector component, and the last as the scalar component. Make sure you know which notation is being used when comparing texts.