Quaternion Algebra and Rotation Quaternions

Technical Note

Document XXX

Version: Draft

Authors: Mark Pedley and Michael Stanley

Date: xx Aug 2014

Table of Contents

Glossary

$a = a_0 + a_1\boldsymbol{i} + a_2\boldsymbol{j} + a_3\boldsymbol{k}$	Components of quaternion a
$a^* = a_0 - a_1\boldsymbol{i} - a_2\boldsymbol{j} - a_3\boldsymbol{k}$	Quaternion conjugate of a
$a = \{a_0, \boldsymbol{a}\}$	Scalar and vector representation of quaternion a
$a^{-1} = \frac{a^*}{N(a)^2}$	Inverse or reciprocal of quaternion a
a_0	Scalar component of quaternion a
$\boldsymbol{a} = a_1\boldsymbol{i} + a_2\boldsymbol{j} + a_3\boldsymbol{k}$	Vector component of quaternion a
$\boldsymbol{a} \cdot \boldsymbol{b}$	Scalar product of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$
$\boldsymbol{a} \times \boldsymbol{b}$	Vector product of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$
$\hat{\boldsymbol{n}}$	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
$\boldsymbol{R}_x, \boldsymbol{R}_y, \boldsymbol{R}_z$	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
ρ	Compass heading angle

1 Introduction

1.1 Summary

This document contains an introduction to quaternion algebra and documents the mathematics behind the quaternion functions in the file orientation.c.

1.2 Functions

void fQuaternionFromRotationVectorDeg(struct fquaternion *pq, const float rvecdeg[], float fscaling); Function computes a rotation quaternion from a scaled rotation vector.
void fQuaternionFromRotationMatrix(float R[][3], struct fquaternion *pq); Function computes a rotation quaternion from a rotation matrix.
void fRotationMatrixFromQuaternion(float R[][3], const struct fquaternion *pq); Function computes a rotation matrix from a rotation quaternion.
void fRotationVectorDegFromQuaternion(struct fquaternion *pq, float rvecdeg[]); Function computes a rotation vector from a rotation quaternion.
void fLPFOrientationQuaternion(struct fquaternion *pfq, struct fquaternion *pfLPq, float flpf, float fdeltat, float fOmega[], int32 loopcounter); Function low pass filters a sequence of quaternions.
void qAeqBxC(struct fquaternion *pqA, const struct fquaternion *pqB, const struct fquaternion *pqC); Function sets the quaternion A to the quaternion product BC. See section 2.4.
void qAeqAxB(struct fquaternion *pqA, const struct fquaternion *pqB); Function sets the quaternion A to the quaternion product AB. See section 2.4.
struct fquaternion qconjgAxB(const struct fquaternion *pqA, const struct fquaternion *pqB); Function returns the quaternion product A*B where A* is the conjugate of A. See section 2.7.
void fqAeqNormqA(struct fquaternion *pqA); Function normalizes the quaternion A. See section 2.8.
void fqAeq1(struct fquaternion *pqA); Function sets the quaternion A to the unit or identity quaternion. See section 2.1.
void fQuaternionFromRotationVectorDeg(struct fquaternion *pq, const float rvecdeg[], float fscaling);

Computes the rotation quaternion from a rotation vector. See section 3.1.
void fRotationVectorDegFromQuaternion(struct fquaternion *pq, float rvecdeg[]); Computes the rotation vector from a rotation quaternion. See section 4.
void fLPFOrientationQuaternion(struct fquaternion *pfq, struct fquaternion *pfLPq, float flpf, float fdeltat, float fOmega[], int32 loopcounter); See section 5.

2 Quaternion Algebra

2.1 Introduction

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

Whereas a complex number z has two (real and imaginary) components:

$$z = a + bi \qquad \text{Eq 2.1.1}$$

a quaternion a has four components:

$$a = a_0 + a_1\boldsymbol{i} + a_2\boldsymbol{j} + a_3\boldsymbol{k} = \{a_0, a_1, a_2, a_3\} \qquad \text{Eq 2.1.2}$$

where a_0, a_1, a_2 and a_3 are real numbers.

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

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

$$a = a_0 + \boldsymbol{a} = \{a_0, \boldsymbol{a}\} = \left\{a_0, \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}\right\} \qquad \text{Eq 2.1.3}$$

If the scalar component a_0 is zero, the quaternion is termed a pure quaternion or vector. If the vector component $\boldsymbol{a}$ is zero, then the quaternion is simply a real number. The quaternion with scalar component equal to 1 and vector component equal to zero is termed the unit or identity quaternion.

2.2 Equality of Two Quaternions

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

$$a = b \;\Rightarrow\; a_0 = b_0, a_1 = b_1, a_2 = b_2 \; and \; a_3 = b_3 \qquad \text{Eq 2.2.1}$$

2.3 Addition of Two Quaternions

Addition of quaternions is defined as the addition of the four components:

$$a + b = (a_0 + b_0) + (a_1 + b_1)\boldsymbol{i} + (a_2 + b_2)\boldsymbol{j} + (a_3 + b_3)\boldsymbol{k} \qquad \text{Eq 2.3.1}$$

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

$$a + b = b + a \qquad \text{Eq 2.3.2}$$

$$(a + b) + c = a + (b + c) \qquad \text{Eq 2.3.3}$$

2.4 Product of two Quaternions

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

$$c = ab = (a_0 + a_1\boldsymbol{i} + a_2\boldsymbol{j} + a_3\boldsymbol{k})(b_0 + b_1\boldsymbol{i} + b_2\boldsymbol{j} + b_3\boldsymbol{k}) \qquad \text{Eq 2.4.1}$$

$$= a_0(b_0 + b_1\boldsymbol{i} + b_2\boldsymbol{j} + b_3\boldsymbol{k}) + a_1\boldsymbol{i}(b_0 + b_1\boldsymbol{i} + b_2\boldsymbol{j} + b_3\boldsymbol{k})$$

$$+a_2\boldsymbol{j}(b_0 + b_1\boldsymbol{i} + b_2\boldsymbol{j} + b_3\boldsymbol{k}) + a_3\boldsymbol{k}(b_0 + b_1\boldsymbol{i} + b_2\boldsymbol{j} + b_3\boldsymbol{k}) \qquad \text{Eq 2.4.2}$$

$$= (a_0b_0 + a_0b_1\boldsymbol{i} + a_0b_2\boldsymbol{j} + a_0b_3\boldsymbol{k}) + (a_1b_0\boldsymbol{i} + a_1b_1\boldsymbol{ii} + a_1b_2\boldsymbol{ij} + a_1b_3\boldsymbol{ik})$$

$$+(a_2b_0\boldsymbol{j} + a_2b_1\boldsymbol{ji} + a_2b_2\boldsymbol{jj} + a_2b_3\boldsymbol{jk}) + (a_3b_0\boldsymbol{k} + a_3b_1\boldsymbol{ki} + a_3b_2\boldsymbol{kj} + a_3b_3\boldsymbol{kk}) \qquad \text{Eq 2.4.3}$$

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

$$r\boldsymbol{i} = \boldsymbol{i}r \qquad \text{Eq 2.4.4}$$

$r\boldsymbol{j} = \boldsymbol{j}r$ Eq 2.4.5

$r\boldsymbol{k} = \boldsymbol{k}r$ Eq 2.4.6

$\boldsymbol{ii} = \boldsymbol{jj} = \boldsymbol{kk} = -1$ Eq 2.4.7

$\boldsymbol{ij} = -\boldsymbol{ji} = \boldsymbol{k}$ Eq 2.4.8

$\boldsymbol{jk} = -\boldsymbol{kj} = \boldsymbol{i}$ Eq 2.4.9

$\boldsymbol{ki} = -\boldsymbol{ik} = \boldsymbol{j}$ Eq 2.4.10

A consequence of equations 2.4.7 to 2.4.10 is that:

$(\boldsymbol{ij})\boldsymbol{k} = \boldsymbol{i}(\boldsymbol{jk}) = -1$ Eq 2.4.11

Substitution of equations 2.4.4 to 2.4.11 into equation 2.4.3 simplifies the quaternion product to:

$$c = ab = (a_0b_0 - a_1b_1 - a_2b_2 - a_3b_3) + (a_0b_1 + a_1b_0 + a_2b_3 - a_3b_2)\boldsymbol{i}$$
$$+(a_0b_2 - a_1b_3 + a_2b_0 + a_3b_1)\boldsymbol{j} + (a_0b_3 + a_1b_2 - a_2b_1 + a_3b_0)\boldsymbol{k}$$ Eq 2.4.12

The four components of the product quaternion c are therefore:

$c_0 = a_0b_0 - a_1b_1 - a_2b_2 - a_3b_3$ Eq 2.4.13

$c_1 = a_0b_1 + a_1b_0 + a_2b_3 - a_3b_2$ Eq 2.4.14

$c_2 = a_0b_2 - a_1b_3 + a_2b_0 + a_3b_1$ Eq 2.4.15

$c_3 = a_0b_3 + a_1b_2 - a_2b_1 + a_3b_0$ Eq 2.4.16

Examination of equations 2.4.13 to 2.4.16 shows that quaternion multiplication does not commute:

$ab \neq ba$ Eq 2.4.17

Brute force evaluation proves that quaternion multiplication is associative.

$$(ab)c - a(bc) = \{(a_0 + a_1\boldsymbol{i} + a_2\boldsymbol{j} + a_3\boldsymbol{k})(b_0 + b_1\boldsymbol{i} + b_2\boldsymbol{j} + b_3\boldsymbol{k})\}(c_0 + c_1\boldsymbol{i} + c_2\boldsymbol{j} + c_3\boldsymbol{k})$$
$$- (a_0 + a_1\boldsymbol{i} + a_2\boldsymbol{j} + a_3\boldsymbol{k})\{(b_0 + b_1\boldsymbol{i} + b_2\boldsymbol{j} + b_3\boldsymbol{k})(c_0 + c_1\boldsymbol{i} + c_2\boldsymbol{j} + c_3\boldsymbol{k})\} = 0$$

Eq 2.4.18

$\Rightarrow (ab)c = a(bc)$ Eq 2.4.19

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.

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

$ab = a_0b_0 + a_0b_1\boldsymbol{i} + a_0b_2\boldsymbol{j} + a_0b_3\boldsymbol{k}$ Eq 2.5.1

$ba = b_0a_0 + b_1a_0\boldsymbol{i} + b_2a_0\boldsymbol{j} + b_3a_0\boldsymbol{k} = a_0b_0 + a_0b_1\boldsymbol{i} + a_0b_2\boldsymbol{j} + a_0b_3\boldsymbol{k}$ Eq 2.5.2

$\Rightarrow ab = ba\ iff\ a\ is\ a\ scalar$ Eq 2.5.3

The product of a quaternion with a scalar quaternion does commute.

2.6 Product of a Quaternion with a Vector

The product of a quaternion a with a vector or pure quaternion $\boldsymbol{b}$ results in a general quaternion and not another vector since the scalar component of the product is non-zero:

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

Eq 2.6.1

$$\boldsymbol{b}a = (-b_1a_1 - b_2a_2 - b_3a_3) + (b_1a_0 + b_2a_3 - b_3a_2)\boldsymbol{i} + (-b_1a_3 + b_2a_0 + b_3a_1)\boldsymbol{j} + (b_1a_2 - b_2a_1 + b_3a_0)\boldsymbol{k}$$ Eq 2.6.2

Inspection of equations 2.6.1 and 2.6.2 shows that the product of the quaternion a and vector $\boldsymbol{b}$ does not commute:

$$a\boldsymbol{b} \neq \boldsymbol{b}a$$ Eq 2.6.3

2.7 Quaternion Conjugate

The quaternion conjugate a^* is defined as:

$$a^* = a_0 - a_1\boldsymbol{i} - a_2\boldsymbol{j} - a_3\boldsymbol{k}$$ Eq 2.7.1

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

$$(ab)^* = (a_0b_0 - a_1b_1 - a_2b_2 - a_3b_3) - (a_0b_1 + a_1b_0 + a_2b_3 - a_3b_2)\boldsymbol{i}$$
$$-(a_0b_2 - a_1b_3 + a_2b_0 + a_3b_1)\boldsymbol{j} - (a_0b_3 + a_1b_2 - a_2b_1 + a_3b_0)\boldsymbol{k}$$ Eq 2.7.2

The product b^*a^* is:

$$b^*a^* = (b_0 - b_1\boldsymbol{i} - b_2j - b_3\boldsymbol{k})(a_0 - a_1\boldsymbol{i} - a_2\boldsymbol{j} - a_3\boldsymbol{k})$$ Eq 2.7.3

$$= b_0a_0 - b_0a_1\boldsymbol{i} - b_0a_2\boldsymbol{j} - b_0a_3\boldsymbol{k} - b_1\boldsymbol{i}a_0 + b_1\boldsymbol{i}a_1\boldsymbol{i} + b_1\boldsymbol{i}a_2\boldsymbol{j} + b_1\boldsymbol{i}a_3\boldsymbol{k}$$
$$-b_2\boldsymbol{j}a_0 + b_2\boldsymbol{j}a_1\boldsymbol{i} + b_2\boldsymbol{j}a_2\boldsymbol{j} + b_2\boldsymbol{j}a_3\boldsymbol{k} - b_3\boldsymbol{k}a_0 + b_3\boldsymbol{k}a_1\boldsymbol{i} + b_3\boldsymbol{k}a_2\boldsymbol{j} + b_3\boldsymbol{k}a_3\boldsymbol{k}$$ Eq 2.7.4

$$= b_0a_0 - b_0a_1\boldsymbol{i} - b_0a_2\boldsymbol{j} - b_0a_3\boldsymbol{k} - b_1a_0\boldsymbol{i} - b_1a_1 + b_1a_2\boldsymbol{k} - b_1a_3\boldsymbol{j}$$
$$-b_2a_0\boldsymbol{j} - b_2a_1\boldsymbol{k} - b_2a_2 + b_2a_3\boldsymbol{i} - b_3a_0\boldsymbol{k} + b_3a_1j - b_3a_2\boldsymbol{i} - b_3a_3$$ Eq 2.7.5

$$= (b_0a_0 - b_1a_1 - b_2a_2 - b_3a_3) - (b_1a_0 + b_0a_1 + b_3a_2 - b_2a_3)\boldsymbol{i}$$
$$-(b_2a_0 - b_3a_1 + b_0a_2 + b_1a_3)\boldsymbol{j} - (b_3a_0 + b_2a_1 - b_1a_2 + b_0a_3)\boldsymbol{k}$$ Eq 2.7.6

$$\Rightarrow (ab)^* = b^*a^*$$ Eq 2.7.7

Simple extension to higher order products gives:

$$(abc \ldots z)^* = z^* \ldots c^*b^*a^*$$ Eq 2.7.8

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

$$a + a^* = (a_0 + a_1\boldsymbol{i} + a_2\boldsymbol{j} + a_3\boldsymbol{k}) + (a_0 - a_1\boldsymbol{i} - a_2\boldsymbol{j} - a_3\boldsymbol{k}) = 2a_0$$ Eq 2.7.9

2.8 Quaternion Norm

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

$$N(a) = \sqrt{{a_0}^2 + {a_1}^2 + {a_2}^2 + {a_3}^2}$$ Eq 2.8.1

The products of a quaternion a with its conjugate a^* evaluate to:

$$a^*a = (a_0 - a_1\boldsymbol{i} - a_2\boldsymbol{j} - a_3\boldsymbol{k})(a_0 + a_1\boldsymbol{i} + a_2\boldsymbol{j} + a_3\boldsymbol{k})$$ Eq 2.8.2

$$= a_0a_0 + a_0a_1\boldsymbol{i} + a_0a_2\boldsymbol{j} + a_0a_3\boldsymbol{k} - a_1a_0\boldsymbol{i} + a_1a_1 - a_1a_2\boldsymbol{k} + a_1a_3\boldsymbol{j}$$

$-a_2a_0\boldsymbol{j} + a_2a_1\boldsymbol{k} + a_2a_2 - a_2a_3\boldsymbol{i} - a_3a_0\boldsymbol{k} - a_3a_1\boldsymbol{j} + a_3a_2\boldsymbol{i} + a_3a_3$ Eq 2.8.3

$= {a_0}^2 + {a_1}^2 + {a_2}^2 + {a_3}^2 = N(a)^2$ Eq 2.8.4

$aa^* = (a_0 + a_1\boldsymbol{i} + a_2\boldsymbol{j} + a_3\boldsymbol{k})(a_0 - a_1\boldsymbol{i} - a_2\boldsymbol{j} - a_3\boldsymbol{k})$ Eq 2.8.5

$= a_0a_0 - a_0a_1\boldsymbol{i} - a_0a_2\boldsymbol{j} - a_0a_3\boldsymbol{k} + a_1a_0\boldsymbol{i} + a_1a_1 - a_1a_2\boldsymbol{k} + a_1a_3\boldsymbol{j}$
$+a_2a_0\boldsymbol{j} + a_2a_1\boldsymbol{k} + a_2a_2 - a_2a_3\boldsymbol{i} + a_3a_0\boldsymbol{k} - a_3a_1\boldsymbol{j} + a_3a_2\boldsymbol{i} + a_3a_3$ Eq 2.8.6

$= {a_0}^2 + {a_1}^2 + {a_2}^2 + {a_3}^2 = N(a)^2$ Eq 2.8.7

$\Rightarrow N(a) = \sqrt{{a_0}^2 + {a_1}^2 + {a_2}^2 + {a_3}^2} = \sqrt{a^*a} = \sqrt{aa^*}$ Eq 2.8.8

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

$N(a^*) = \sqrt{{a_0}^2 + (-a_1)^2 + (-a_2)^2 + (-a_3)^2} = \sqrt{{a_0}^2 + {a_1}^2 + {a_2}^2 + {a_3}^2} = N(a)$ Eq 2.8.9

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

$$N(ab) = \sqrt{\begin{matrix}(a_0b_0 - a_1b_1 - a_2b_2 - a_3b_3)^2 + (a_0b_1 + a_1b_0 + a_2b_3 - a_3b_2)^2 \\ +(a_0b_2 - a_1b_3 + a_2b_0 + a_3b_1)^2 + (a_0b_3 + a_1b_2 - a_2b_1 + a_3b_0)^2\end{matrix}}$$ Eq 2.8.10

$= \sqrt{({a_0}^2 + {a_1}^2 + {a_2}^2 + {a_3}^2)({b_0}^2 + {b_1}^2 + {b_2}^2 + {b_3}^2)} = N(a)N(b)$ Eq 2.8.11

2.9 Quaternion Inverse and Division

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

$aa^{-1} = a^{-1}a = 1$ Eq 2.9.1

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

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

Eq 2.9.2

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

Eq 2.9.3

The quaternion inverse a^{-1} is therefore identified for all quaternions with non-zero norm as:

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

Eq 2.9.4

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

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

Eq 2.9.5

2.10 Vector Representation of Quaternion Product

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

$$\boldsymbol{a}.\boldsymbol{b} = a_1 b_1 + a_2 b_2 + a_3 b_3 \qquad \text{Eq 2.10.1}$$

$$\boldsymbol{a} \times \boldsymbol{b} = \begin{pmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{pmatrix} \qquad \text{Eq 2.10.2}$$

The product between two quaternions $a = (a_0, \boldsymbol{a})$ and $b = (b_0, \boldsymbol{b})$ can be written in an alternative form involving scalar and vector products on their vector components. Direction expansion of the scalar and vector expression below shows it equals the quaternion product ab:

$$\{a_0 b_0 - \boldsymbol{a}.\boldsymbol{b}, a_0 \boldsymbol{b} + b_0 \boldsymbol{a} + \boldsymbol{a} \times \boldsymbol{b}\} = \left\{ a_0 b_0 - a_1 b_1 + a_2 b_2 + a_3 b_3, a_0 \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} + b_0 \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} + \begin{pmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{pmatrix} \right\} \qquad \text{Eq 2.10.3}$$

$$= \left\{ a_0 b_0 - a_1 b_1 + a_2 b_2 + a_3 b_3, \begin{pmatrix} a_0 b_1 + a_1 b_0 + a_2 b_3 - a_3 b_2 \\ a_0 b_2 - a_1 b_3 + a_2 b_0 + a_3 b_1 \\ a_0 b_3 + a_1 b_2 - a_2 b_1 + a_3 b_0 \end{pmatrix} \right\} \qquad \text{Eq 2.10.4}$$

Comparison with equation 2.4.12 gives the identity:

$$ab = a_0 b_0 - \boldsymbol{a}.\boldsymbol{b}, a_0 \boldsymbol{b} + b_0 \boldsymbol{a} + \boldsymbol{a} \times \boldsymbol{b} \qquad \text{Eq 2.10.5}$$

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 $\hat{\boldsymbol{n}}$ by angle η is defined to be:

$$q = cos\left(\frac{\eta}{2}\right) + \hat{\boldsymbol{n}} sin\left(\frac{\eta}{2}\right) \qquad \text{Eq 3.1.1}$$

Equation 3.1.1 explicitly defines the rotation quaternion in terms of the rotation vector.

q is obviously a unit quaternion since its norm equals 1:

$$N(q) = cos^2\left(\frac{\eta}{2}\right) + sin^2\left(\frac{\eta}{2}\right)\left(\hat{n}_x{}^2 + \hat{n}_y{}^2 + \hat{n}_z{}^2\right) = 1 \qquad \text{Eq 3.1.2}$$

The next section proves that the rotation quaternion is related to the general rotation matrix $\boldsymbol{R}$ operating on a vector $\boldsymbol{v}$ by:

$$\boldsymbol{Rv} = q^*\boldsymbol{v}q \qquad \text{Eq 3.1.3}$$

3.2 Equivalence of Rotation Quaternion and Rotation Matrix

The left hand side of equation 3.1.3 evaluates to:

$$q^*\boldsymbol{v}q = \left(cos\left(\frac{\eta}{2}\right) - \hat{\boldsymbol{n}} sin\left(\frac{\eta}{2}\right)\right)\boldsymbol{v}\left(cos\left(\frac{\eta}{2}\right) + \hat{\boldsymbol{n}} sin\left(\frac{\eta}{2}\right)\right) \qquad \text{Eq 3.2.1}$$

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

Eq 3.2.2

$$= \begin{pmatrix} \{\hat{n}_x{}^2 + (1-\hat{n}_x{}^2)cos\eta\}v_x + \{\hat{n}_x\hat{n}_y(1-cos\eta) + \hat{n}_z sin\eta\}v_y + \{\hat{n}_x\hat{n}_z(1-cos\eta) - \hat{n}_y sin\eta\}v_z \\ \{\hat{n}_x\hat{n}_y(1-cos\eta) - \hat{n}_z sin\eta\}v_x + \{\hat{n}_y{}^2 + (1-\hat{n}_y{}^2)cos\eta\}v_y + \{\hat{n}_y\hat{n}_z(1-cos\eta) + \hat{n}_x sin\eta\}v_z \\ \{\hat{n}_x\hat{n}_z(1-cos\eta) + \hat{n}_y sin\eta\}v_x + \{\hat{n}_y\hat{n}_z(1-cos\eta) - \hat{n}_x sin\eta\}v_y + \{\hat{n}_z{}^2 + (1-\hat{n}_z{}^2)cos\eta\}v_z \end{pmatrix} \qquad \text{Eq 3.2.3}$$

The general rotation matrix $\boldsymbol{R}$ which transforms a vector as a result of a rotation of the coordinate system around the axis $\hat{\boldsymbol{n}}$ by angle η is:

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

Eq 3.2.4

The right hand side of equation 3.1.3 evaluates to:

$$\boldsymbol{Rv} = \begin{pmatrix} \hat{n}_x{}^2 + (1-\hat{n}_x{}^2)cos\eta & \hat{n}_x\hat{n}_y(1-cos\eta) + \hat{n}_z sin\eta & \hat{n}_x\hat{n}_z(1-cos\eta) - \hat{n}_y sin\eta \\ \hat{n}_x\hat{n}_y(1-cos\eta) - \hat{n}_z sin\eta & \hat{n}_y{}^2 + (1-\hat{n}_y{}^2)cos\eta & \hat{n}_y\hat{n}_z(1-cos\eta) + \hat{n}_x sin\eta \\ \hat{n}_x\hat{n}_z(1-cos\eta) + \hat{n}_y sin\eta & \hat{n}_y\hat{n}_z(1-cos\eta) - \hat{n}_x sin\eta & \hat{n}_z{}^2 + (1-\hat{n}_z{}^2)cos\eta \end{pmatrix}\begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \qquad \text{Eq 3.2.5}$$

$$= \begin{pmatrix} \{\hat{n}_x{}^2 + (1-\hat{n}_x{}^2)cos\eta\}v_x + \{\hat{n}_x\hat{n}_y(1-cos\eta) + \hat{n}_z sin\eta\}v_y + \{\hat{n}_x\hat{n}_z(1-cos\eta) - \hat{n}_y sin\eta\}v_z \\ \{\hat{n}_x\hat{n}_y(1-cos\eta) - \hat{n}_z sin\eta\}v_x + \{\hat{n}_y{}^2 + (1-\hat{n}_y{}^2)cos\eta\}v_y + \{\hat{n}_y\hat{n}_z(1-cos\eta) + \hat{n}_x sin\eta\}v_z \\ \{\hat{n}_x\hat{n}_z(1-cos\eta) + \hat{n}_y sin\eta\}v_x + \{\hat{n}_y\hat{n}_z(1-cos\eta) - \hat{n}_x sin\eta\}v_y + \{\hat{n}_z{}^2 + (1-\hat{n}_z{}^2)cos\eta\}v_z \end{pmatrix} \qquad \text{Eq 3.2.6}$$

Equations 3.2.3 and 3.2.6 match proving the identity in equation 3.1.3.

3.3 Inverse Rotation Quaternion

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

$$q^* = cos\left(\frac{\eta}{2}\right) - \hat{\boldsymbol{n}}sin\left(\frac{\eta}{2}\right) = cos\left(\frac{-\eta}{2}\right) + \hat{\boldsymbol{n}}sin\left(\frac{-\eta}{2}\right) \qquad \text{Eq 3.3.1}$$

The inverse nature of the operators q and q^* can also be shown by computing the effects of the product rotations i) q followed by q^* and ii) q^* followed by q applied to the vector $\boldsymbol{v}$ using the associative property of quaternion multiplication:

$$q(q^*\boldsymbol{v}q)q^* = (qq^*)\boldsymbol{v}(qq^*) = \boldsymbol{v} \qquad \text{Eq 3.3.2}$$

$$q^*(q\boldsymbol{v}q^*)q = (q^*q)\boldsymbol{v}(q^*q) = \boldsymbol{v} \qquad \text{Eq 3.3.3}$$

3.4 Product of Rotation Quaternions

The result of successively applying rotation quaternions q_1 followed by q_2 through q_N to vector $\boldsymbol{v}$ is:

$$q_N{}^* \ldots (q_2{}^*(q_1{}^*\boldsymbol{v}q_1)q_2) \ldots q_N = (q_N{}^* \ldots q_2{}^*q_1{}^*)\boldsymbol{v}(q_1q_2 \ldots \ldots q_N) \qquad \text{Eq 3.4.1}$$

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

$$q = q_1q_1q_1 \ldots q_N \qquad \text{Eq 3.4.2}$$

3.5 Negation of Rotation Axis and Angle

Both rotation matrices and quaternions are unchanged if both the rotation angle and axis are simultaneously inverted:

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

$$= \boldsymbol{R}(\hat{\boldsymbol{n}},\eta) \qquad \text{Eq 3.5.1}$$

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

Eq 3.5.2

Equations 3.5.1 and 3.5.2 simply state mathematically the obvious 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 of a rotation quaternion to be non-negative. If a negative scalar component is detected then the entire quaternion (both scalar and vector components) can be safely negated. An example of this can be found in the function fqAeqNormqA.

3.6 Coordinate Frame Rotation Standard

Some texts define the quaternion rotation operator on vector $\boldsymbol{v}$ to be $q\boldsymbol{v}q^*$ instead of $q^*\boldsymbol{v}q$. The explanation is that the operator $q^*\boldsymbol{v}q$ transforms the vector $\boldsymbol{v}$ as a result of rotation of the coordinate system by angle η whereas the operator $q\boldsymbol{v}q^*$ rotates the vector $\boldsymbol{v}$ by angle η in a fixed coordinate system. The standard used in this document and all Freescale sensor fusion software is that it is the coordinate system that is rotating (normally as a result of the smartphone or tablet orientation changing) and not the vector $\boldsymbol{v}$ (which is typically the earth's gravitational or geomagnetic field and therefore fixed in the earth reference frame).

4 Quaternion Derivative

4.1 Definition

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

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

Eq 4.1.1

A consequence of equation 4.1.1 is that in the limit:

$$q(t+dt) = q(t) + dt\dot{q}(t)$$ Eq 4.1.2

4.2 Derivation

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

$$q(t+\delta t) = q(t)\delta q(t)$$ Eq 4.2.1

Equation 4.2.1 is exact by definition. With the assumption that incremental change in orientation results from a constant angular velocity $\boldsymbol{\omega}$ then:

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

Eq 4.2.2

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

In the limit of the integration time δt becoming the infinitesimal dt:

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

Eq 4.2.3

and:

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

Eq 4.2.4

Combining equations 4.1.2 and 4.2.4 gives:

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

Eq 4.2.5

Since quaternion multiplication is distributive:

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

Eq 4.2.6

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

Eq 4.2.7

5 Computing Rotation Vector from Quaternion

The definition of the rotation quaternion in equation 3.1.1 shows that it is closely linked to the equivalent rotation vector. Inverting the process to recover the rotation vector from the quaternion is consequently straightforward.

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

Eq 5.1

Equating the scalar components gives:

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

Eq 5.2

Since q_0 varies between 0 and 1, the rotation angle η in equation 5.1.2 has the correct range 0^o to 180^o.

Equating the remaining three components of the quaternion gives:

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

Eq 5.3

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

Eq 5.4

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

Eq 5.5

In the case where $\sin\left(\frac{\eta}{2}\right)$ is zero, the rotation angle is zero and the rotation axis is irrelevant. The rotation vector can then be safely set to zero.

6 Low Pass Filtering Orientation Quaternion

6.1 Introduction

The Kalman filter algorithms compute an optimal Kalman filter estimate of the orientation. The simpler accelerometer and magnetometer eCompass algorithms require the explicit low pass filtering of the stream of noisy orientation estimates whether in quaternion or rotation matrix forms. This is performed in function fLPFOrientationQuaternion.

Low pass filtering orientation estimates is less intuitive than it might appear. Low pass filtering the components of the rotation quaternion or rotation matrix gives poor results since the result is not, in general, a valid rotation quaternion or rotation matrix. Explicitly re-normalizing the quaternion or orthornormalizing the rotation matrix gives a poor low pass filtered trajectory.

The quaternion low pass filter is developed by analogy with the simple single pole low pass filter in the time domain:

$y[n] = (1-\alpha)y[n-1] + \alpha x[n]$ Eq 6.1.1

where:

$0 < \alpha \leq 1$ Eq 6.1.2

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

In C code, equation 5.1.1 would be written as:

```
yn += alpha * (xn - yn);
```
Eq 6.1.3

Equation 6.1.3 makes it clear that the low pass estimate yn is updated by α times the current input xn minus the previous low pass filtered estimate. The low pass estimate is therefore exponentially steered towards the (time varying) input sequence.

6.2 Quaternion Low Pass Filter

The analogous low pass filter in orientation space would exponentially rotate the current low pass filtered orientation quaternion towards the instantaneous (and noisy) orientation estimate. The rotation axis should be the rotation axis between the low pass and current orientation quaternions and the low pass filtered quaternion should be rotated by the fraction alpha times the angle between the two orientation quaternions.

The incremental rotation quaternion $\Delta q[n]$ at iteration n between the previous estimate of the low pass filtered quaternion estimate $q_{LP}[n-1]$ and the current quaternion $q[n]$ are related by:

$q[n] = q_{LP}[n-1]\Delta q[n]$ Eq 6.2.1

$\Rightarrow \Delta q[n] = \Delta q_0[n] + \Delta q_1[n]\boldsymbol{i} + \Delta q_2[n]\boldsymbol{j} + \Delta q_3[n]\boldsymbol{k} = {q_{LP}}^*[n-1]q[n]$ Eq 6.2.2

If $\Delta q[n]$ has negative scalar component $\Delta q_0[n]$ then the entire quaternion is negated.

The degree to which the low pass and current orientation estimates have diverged is given by the scalar component of $\Delta q[n]$ which equals the cosine of half the rotation angle encoded in $\Delta q[n]$. The smaller the scalar component, the larger the degree to which the low pass filter estimate is lagging the current instantaneous orientation estimate. This allows the use of a variable α which is small when the orientation is not changing rapidly (giving a long time constant and a low noise filtered estimate) and a larger alpha when the orientation is changing rapidly (giving low latency in the filter at the expense of noise which is not visible anyway when the orientation is changing rapidly).

With the terminology that Δq_0 is the scalar component of $\Delta q[n]$, the low pass filter constant α is set to the experimentally determined equation:

$\alpha = \alpha_0 + \frac{3}{4}(1 - \Delta q_0)$ Eq 6.2.3

α is not permitted to exceed the all pass value of $\alpha = 1$. The term α_0 is the nominal filter coefficient and typically has value of about 0.5secs.

The low pass filter update equation is then:

$$q_{LP}[n] = q_{LP}[n-1]\left\{\sqrt{1 - \alpha^2\Delta q_1[n]^2 - \alpha^2\Delta q_2[n]^2 - \alpha^2\Delta q_3[n]^2} + \alpha\Delta q_1[n]\boldsymbol{i} + \alpha\Delta q_2[n]\boldsymbol{j} + \alpha\Delta q_3[n]\boldsymbol{k}\right\} \quad \text{Eq 6.2.4}$$

In plain English, for $\alpha = 0.1$, equation 6.2.3 states "to obtain the new low pass filtered quaternion, rotate the old low pass filtered quaternion by some 10% of the angle between it and the new (noisy) orientation quaternion".