AN5022

Quaternion Algebra and Rotations

Rev. 2.0 — 21 June 2016 **Application note**

Document information

Info	Content
Abstract	This application note contains an introduction to quaternion algebra and its use to represent rotations. It documents the specific quaternion functions used in the NXP Sensor Fusion Library contained in the file *orientation.c*.

Revision history

Document ID	Release date	Supercedes
AN5022 v2.0	20160621	AN5022 v1.0
Modifications:	• Minor changes • The format of this document has been redesigned to comply with the new identity guidelines of NXP Semiconductors. Legal texts have been adapted to the new company name where appropriate.	
AN5022 v1.0	2015 September	—

Contact information

For more information, please visit: http://www.nxp.com

1. Introduction

1.1 Summary

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

Quaternions are four dimensional hyper-complex numbers that were invented by Professor William Rowan Hamilton in 1843. 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. The Euler angles (roll, pitch and yaw) are, in contrast, mathematically unsuited for use in sensor fusion software. The phenomenon of gimbal lock instability in strapdown sensor systems occurs only in the Euler angle representation of orientation and is entirely absent in rotation matrix or rotation quaternion representations. Euler angles are therefore only used in the NXP Sensor Fusion Library as an alternative final representation of an orientation which has been computed using rotation matrices or rotation quaternions.

1.2 Terminology

Symbol	Definition
$a = a_0 + a_1\boldsymbol{i} + a_2\boldsymbol{j} + a_3\boldsymbol{k}$	Components of quaternion a
$a^* = a_0 - a_1\boldsymbol{i} - a_2\boldsymbol{j} - a_3\boldsymbol{k}$	Conjugate of quaternion a
$a = \{a_0, \boldsymbol{a}\}$	Representation of quaternion a in terms of scalar a_0 and vector $\boldsymbol{a}$ components
$a^{-1} = \frac{a^*}{N(a)^2}$	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

Symbol	Definition
r	Real number
z	Complex number
η	General rotation angle
ϕ	Roll angle
θ	Pitch angle
ψ	Yaw angle
ρ	Compass heading angle

1.3 Software Functions

Table 1. Sensor Fusion software functions

Functions	Description	Reference Section
`void fqAeq1(struct fquaternion *pqA);`	Function sets the quaternion A to the unit or identity quaternion.	2.1
`void qAeqBxC (struct fquaternion *pqA, const struct fquaternion *pqB, const struct fquaternion *pqC);`	Function sets the quaternion A to the quaternion product BC.	2.4
`void qAeqAxB(struct fquaternion *pqA, const struct fquaternion *pqB);`	Function sets the quaternion A to the quaternion product AB.	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.	2.7
`void fqAeqNormqA (struct fquaternion *pqA);`	Function normalizes the quaternion A.	2.8
`void fRotationMatrixFromQuaternion (float R[][3], const struct fquaternion *pq);`	Function computes a rotation matrix from a rotation quaternion.	4.1
`void fQuaternionFromRotationMatrix (float R[][3], struct fquaternion *pq);`	Function computes a rotation quaternion from a rotation matrix.	4.2

Functions	Description	Reference Section
`void fRotationVectorDegFromQuaternion (struct fquaternion *pq, float rvecdeg[]);`	Computes the rotation vector from a rotation quaternion.	5.1
`void fQuaternionFromRotationVectorDeg (struct fquaternion *pq, const float rvecdeg[], float fscaling);`	Computes the rotation quaternion from a rotation vector with scaling.	5.2
`void fLPFOrientationQuaternion (struct fquaternion *pq, struct fquaternion *pLPq, float flpf, float fdeltat, float fOmega[])`	Function low-pass filters the orientations represented by a sequence of quaternions.	6.3

2. Quaternion Algebra

2.1 Introduction

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

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

$$z = a + bi \tag{1}$$

the quaternion a has four components:

$$a = a_0 + a_1\boldsymbol{i} + a_2\boldsymbol{j} + a_3\boldsymbol{k} = \{a_0, a_1, a_2, a_3\} \tag{2}$$

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

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

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

$$a = a_0 + \boldsymbol{a} = \{a_0, \boldsymbol{a}\} = \left\{a_0, \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}\right\} = \left\{a_0, \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix}\right\} \tag{3}$$

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 a real number. The quaternion $a = \{1, 0, 0, 0\}$ 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:

$$a = b \Rightarrow a_0 = b_0, a_1 = b_1, a_2 = b_2, a_3 = b_3 \tag{4}$$

2.3 Addition of Two Quaternions

The sum of two quaternions is defined to be the quaternion with summed components:

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

Because the addition of real numbers is commutative and associative, quaternion addition is also commutative and associative:

$$a + b = b + a \tag{6}$$

$$(a + b) + c = a + (b + c) \tag{7}$$

2.4 Product of Two Quaternions

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

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

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

$$\begin{aligned} = (a_0b_0 + a_0b_1\boldsymbol{i} + a_0b_2\boldsymbol{j} + a_0b_3\boldsymbol{k}) + (a_1b_0\boldsymbol{i} + a_1b_1\boldsymbol{ii} + a_1b_2\boldsymbol{ij} + a_1b_3\boldsymbol{ik}) \\ + (a_2b_0\boldsymbol{j} + a_2b_1\boldsymbol{ji} + a_2b_2\boldsymbol{jj} + a_2b_3\boldsymbol{jk}) \\ + (a_3b_0\boldsymbol{k} + a_3b_1\boldsymbol{ki} + a_3b_2\boldsymbol{kj} + a_3b_3\boldsymbol{kk}) \end{aligned} \tag{10}$$

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

$$r\boldsymbol{i} = \boldsymbol{i}r \tag{11}$$

$$r\boldsymbol{j} = \boldsymbol{j}r \tag{12}$$

$$r\boldsymbol{k} = \boldsymbol{k}r \tag{13}$$

$$\boldsymbol{ii} = \boldsymbol{jj} = \boldsymbol{kk} = -1 \tag{14}$$

$$\boldsymbol{ij} = -\boldsymbol{ji} = \boldsymbol{k} \tag{15}$$

$$\boldsymbol{jk} = -\boldsymbol{kj} = \boldsymbol{i} \tag{16}$$

$$\boldsymbol{ki} = -\boldsymbol{ik} = \boldsymbol{j} \tag{17}$$

A consequence of equations (14) to (17) is that:

$$(\boldsymbol{ij})\boldsymbol{k} = \boldsymbol{i}(\boldsymbol{jk}) = -1 \tag{18}$$

Substitution of equations (14) to (18) into equation (10) simplifies the quaternion product to:

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

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

$$c_0 = a_0b_0 - a_1b_1 - a_2b_2 - a_3b_3 \tag{20}$$

$$c_1 = a_0b_1 + a_1b_0 + a_2b_3 - a_3b_2 \tag{21}$$

$$c_2 = a_0b_2 - a_1b_3 + a_2b_0 + a_3b_1 \tag{22}$$

$$c_3 = a_0b_3 + a_1b_2 - a_2b_1 + a_3b_0 \tag{23}$$

Examination of equations (20) to (23) shows that quaternion multiplication does not commute:

$$ab \neq ba \tag{24}$$

Brute force evaluation proves that quaternion multiplication is associative:

$$\begin{aligned} (ab)c - a(bc) = \{(a_0 + a_1\boldsymbol{i} + a_2\boldsymbol{j} + a_3\boldsymbol{k})(b_0 + b_1\boldsymbol{i} + b_2\boldsymbol{j} + b_3\boldsymbol{k})\}(c_0 + c_1\boldsymbol{i} + c_2\boldsymbol{j} + c_3\boldsymbol{k}) \\ - (a_0 + a_1\boldsymbol{i} + a_2\boldsymbol{j} + a_3\boldsymbol{k})\{(b_0 + b_1\boldsymbol{i} + b_2\boldsymbol{j} + b_3\boldsymbol{k})(c_0 + c_1\boldsymbol{i} + c_2\boldsymbol{j} + c_3\boldsymbol{k})\} = 0 \end{aligned} \tag{25}$$

$$\Rightarrow (ab)c = a(bc) \tag{26}$$

For the special case $a = b$, the product $c = aa$ evaluates to:

$$c = aa = (a_0a_0 - a_1a_1 - a_2a_2 - a_3a_3) + 2a_0a_1\boldsymbol{i} + 2a_0a_2\boldsymbol{j} + 2a_0a_3\boldsymbol{k} \tag{27}$$

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:

$$ab = a_0b_0 + a_0b_1\boldsymbol{i} + a_0b_2\boldsymbol{j} + a_0b_3\boldsymbol{k} \tag{28}$$

$$ba = b_0a_0 + b_1a_0\boldsymbol{i} + b_2a_0\boldsymbol{j} + b_3a_0\boldsymbol{k} = a_0b_0 + a_0b_1\boldsymbol{i} + a_0b_2\boldsymbol{j} + a_0b_3\boldsymbol{k} = ab \tag{29}$$

$$\Rightarrow ab = ba \; if \; a \; is \; a \; scalar \tag{30}$$

2.6 Product of a Quaternion with a Vector

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

$$\begin{aligned} a\boldsymbol{b} = {} & (-a_1b_1 - a_2b_2 - a_3b_3) + (a_0b_1 + a_2b_3 - a_3b_2)\boldsymbol{i} + \\ & (a_0b_2 - a_1b_3 + a_3b_1)\boldsymbol{j} + (a_0b_3 + a_1b_2 - a_2b_1)\boldsymbol{k} \end{aligned} \tag{31}$$

$$\begin{aligned} \boldsymbol{b}a = {} & (-b_1a_1 - b_2a_2 - b_3a_3) + (b_1a_0 + b_2a_3 - b_3a_2)\boldsymbol{i} + \\ & (-b_1a_3 + b_2a_0 + b_3a_1)\boldsymbol{j} + (b_1a_2 - b_2a_1 + b_3a_0)\boldsymbol{k} \end{aligned} \tag{32}$$

Inspection of equations (31) and (32) shows that the product of the quaternion a and vector $\boldsymbol{b}$ does not commute:

$$a\boldsymbol{b} \neq \boldsymbol{b}a \tag{33}$$

2.7 Quaternion Conjugate

The quaternion conjugate a^* is defined as:

$$a^* = a_0 - a_1\boldsymbol{i} - a_2\boldsymbol{j} - a_3\boldsymbol{k} \tag{34}$$

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

$$\begin{aligned} (ab)^* = {} & (a_0b_0 - a_1b_1 - a_2b_2 - a_3b_3) - (a_0b_1 + a_1b_0 + a_2b_3 - a_3b_2)\boldsymbol{i} \\ & - (a_0b_2 - a_1b_3 + a_2b_0 + a_3b_1)\boldsymbol{j} - (a_0b_3 + a_1b_2 - a_2b_1 + a_3b_0)\boldsymbol{k} \end{aligned} \tag{35}$$

The product b^*a^* evaluates to:

$$b^*a^* = (b_0 - b_1\boldsymbol{i} - b_2 j - b_3\boldsymbol{k})(a_0 - a_1\boldsymbol{i} - a_2\boldsymbol{j} - a_3\boldsymbol{k}) \tag{36}$$

$$= b_0a_0 - b_0a_1\boldsymbol{i} - b_0a_2\boldsymbol{j} - b_0a_3\boldsymbol{k} - b_1\boldsymbol{i}a_0 + b_1\boldsymbol{i}a_1\boldsymbol{i} + b_1\boldsymbol{i}a_2\boldsymbol{j} + b_1\boldsymbol{i}a_3\boldsymbol{k} - b_2\boldsymbol{j}a_0 + b_2\boldsymbol{j}a_1\boldsymbol{i} + b_2\boldsymbol{j}a_2\boldsymbol{j} + b_2\boldsymbol{j}a_3\boldsymbol{k} - b_3\boldsymbol{k}a_0 + b_3\boldsymbol{k}a_1\boldsymbol{i} + b_3\boldsymbol{k}a_2\boldsymbol{j} + b_3\boldsymbol{k}a_3\boldsymbol{k} \tag{37}$$

$$= b_0a_0 - b_0a_1\boldsymbol{i} - b_0a_2\boldsymbol{j} - b_0a_3\boldsymbol{k} - b_1a_0\boldsymbol{i} - b_1a_1 + b_1a_2\boldsymbol{k} - b_1a_3\boldsymbol{j} - b_2a_0\boldsymbol{j} - b_2a_1\boldsymbol{k} - b_2a_2 + b_2a_3\boldsymbol{i} - b_3a_0\boldsymbol{k} + b_3a_1 j - b_3a_2\boldsymbol{i} - b_3a_3 \tag{38}$$

$$= (b_0a_0 - b_1a_1 - b_2a_2 - b_3a_3) - (b_1a_0 + b_0a_1 + b_3a_2 - b_2a_3)\boldsymbol{i} - (b_2a_0 - b_3a_1 + b_0a_2 + b_1a_3)\boldsymbol{j} - (b_3a_0 + b_2a_1 - b_1a_2 + b_0a_3)\boldsymbol{k} \tag{39}$$

$$\Rightarrow (ab)^* = b^*a^* \tag{40}$$

Simple extension to higher order products gives:

$$(abc \ldots z)^* = z^*(abc \ldots)^* = z^* \ldots c^*b^*a^* \tag{41}$$

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

$$a + a^* = (a_0 + a_1\boldsymbol{i} + a_2\boldsymbol{j} + a_3\boldsymbol{k}) + (a_0 - a_1\boldsymbol{i} - a_2\boldsymbol{j} - a_3\boldsymbol{k}) = 2a_0 \tag{42}$$

2.8 Quaternion Norm

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

$$N(a) = \sqrt{{a_0}^2 + {a_1}^2 + {a_2}^2 + {a_3}^2} \tag{43}$$

The two products a^*a and aa^* of a quaternion a with its conjugate a^* evaluate to the norm squared $N(a)^2$:

$$a^*a = (a_0 - a_1\boldsymbol{i} - a_2\boldsymbol{j} - a_3\boldsymbol{k})(a_0 + a_1\boldsymbol{i} + a_2\boldsymbol{j} + a_3\boldsymbol{k}) \tag{44}$$

$$= a_0a_0 + a_0a_1\boldsymbol{i} + a_0a_2\boldsymbol{j} + a_0a_3\boldsymbol{k} - a_1a_0\boldsymbol{i} + a_1a_1 - a_1a_2\boldsymbol{k} + a_1a_3\boldsymbol{j} - a_2a_0\boldsymbol{j} + a_2a_1\boldsymbol{k} + a_2a_2 - a_2a_3\boldsymbol{i} - a_3a_0\boldsymbol{k} - a_3a_1\boldsymbol{j} + a_3a_2\boldsymbol{i} + a_3a_3 \tag{45}$$

$$= {a_0}^2 + {a_1}^2 + {a_2}^2 + {a_3}^2 = N(a)^2 \tag{46}$$

$$aa^* = (a_0 + a_1\boldsymbol{i} + a_2\boldsymbol{j} + a_3\boldsymbol{k})(a_0 - a_1\boldsymbol{i} - a_2\boldsymbol{j} - a_3\boldsymbol{k}) \tag{47}$$

$$= a_0a_0 - a_0a_1\boldsymbol{i} - a_0a_2\boldsymbol{j} - a_0a_3\boldsymbol{k} + a_1a_0\boldsymbol{i} + a_1a_1 - a_1a_2\boldsymbol{k} + a_1a_3\boldsymbol{j} + a_2a_0\boldsymbol{j} + a_2a_1\boldsymbol{k} + a_2a_2 - a_2a_3\boldsymbol{i} + a_3a_0\boldsymbol{k} - a_3a_1\boldsymbol{j} + a_3a_2\boldsymbol{i} + a_3a_3 \quad (48)$$

$$= {a_0}^2 + {a_1}^2 + {a_2}^2 + {a_3}^2 = N(a)^2 \quad (49)$$

$$\Rightarrow N(a) = \sqrt{{a_0}^2 + {a_1}^2 + {a_2}^2 + {a_3}^2} = \sqrt{a^*a} = \sqrt{aa^*} \quad (50)$$

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

$$N(a^*) = \sqrt{{a_0}^2 + (-a_1)^2 + (-a_2)^2 + (-a_3)^2} = \sqrt{{a_0}^2 + {a_1}^2 + {a_2}^2 + {a_3}^2} = N(a) \quad (51)$$

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

$$N(ab) = \sqrt{\begin{matrix}(a_0b_0 - a_1b_1 - a_2b_2 - a_3b_3)^2 + (a_0b_1 + a_1b_0 + a_2b_3 - a_3b_2)^2 \\ +(a_0b_2 - a_1b_3 + a_2b_0 + a_3b_1)^2 + (a_0b_3 + a_1b_2 - a_2b_1 + a_3b_0)^2\end{matrix}} \quad (52)$$

$$= \sqrt{({a_0}^2 + {a_1}^2 + {a_2}^2 + {a_3}^2)\left({b_0}^2 + {b_1}^2 + {b_2}^2 + {b_3}^2\right)} = N(a)N(b) \quad (53)$$

2.9 Quaternion Inverse and Division

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

$$aa^{-1} = a^{-1}a = 1 \quad (54)$$

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

$$\frac{1}{N(a)^2}(a_0 - a_1\boldsymbol{i} - a_2\boldsymbol{j} - a_3\boldsymbol{k})(a_0 + a_1\boldsymbol{i} + a_2\boldsymbol{j} + a_3\boldsymbol{k}) = \frac{({a_0}^2 + {a_1}^2 + {a_2}^2 + {a_3}^2)}{N(a)^2} = 1 \quad (55)$$

$$(a_0 + a_1\boldsymbol{i} + a_2\boldsymbol{j} + a_3\boldsymbol{k})\frac{1}{N(a)^2}(a_0 - a_1\boldsymbol{i} - a_2\boldsymbol{j} - a_3\boldsymbol{k}) = \frac{({a_0}^2 + {a_1}^2 + {a_2}^2 + {a_3}^2)}{N(a)^2} = 1 \quad (56)$$

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

$$a^{-1} = \frac{a^*}{N(a)^2} = \left(\frac{a_0}{N(a)^2}\right) - \left(\frac{a_1}{N(a)^2}\right)\boldsymbol{i} - \left(\frac{a_2}{N(a)^2}\right)\boldsymbol{j} - \left(\frac{a_3}{N(a)^2}\right)\boldsymbol{k} \quad (57)$$

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

$$N(a^{-1}) = N\left(\frac{a^*}{N(a)^2}\right) = \left(\frac{1}{N(a)^2}\right)N(a^*) = \left(\frac{N(a)}{N(a)^2}\right) = \left(\frac{1}{N(a)}\right) = \{N(a)\}^{-1} \quad (58)$$

2.10 Vector Representation of Quaternion Product

The 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:

$$\boldsymbol{a}.\boldsymbol{b} = a_1b_1 + a_2b_2 + a_3b_3 \quad (59)$$

$$\boldsymbol{a} \times \boldsymbol{b} = \begin{pmatrix} a_2b_3 - a_3b_2 \\ a_3b_1 - a_1b_3 \\ a_1b_2 - a_2b_1 \end{pmatrix} \quad (60)$$

The product between two quaternions $a = (a_0, \boldsymbol{a})$ and $b = (b_0, \boldsymbol{b})$ can be written in terms of the scalar and vector products on their vector components. Direct expansion of the scalar and vector expression below and comparison with equation (10) shows it equals the quaternion product ab:

$$\{a_0b_0 - \boldsymbol{a}.\boldsymbol{b},\ a_0\boldsymbol{b} + b_0\boldsymbol{a} + \boldsymbol{a} \times \boldsymbol{b}\}$$
$$= \left\{a_0b_0 - a_1b_1 + a_2b_2 + a_3b_3, a_0\begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} + b_0\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} + \begin{pmatrix} a_2b_3 - a_3b_2 \\ a_3b_1 - a_1b_3 \\ a_1b_2 - a_2b_1 \end{pmatrix}\right\} \quad (61)$$

$$= \left\{a_0b_0 - a_1b_1 + a_2b_2 + a_3b_3, \begin{pmatrix} a_0b_1 + a_1b_0 + a_2b_3 - a_3b_2 \\ a_0b_2 - a_1b_3 + a_2b_0 + a_3b_1 \\ a_0b_3 + a_1b_2 - a_2b_1 + a_3b_0 \end{pmatrix}\right\} = ab \quad (62)$$

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

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

q is a unit quaternion because its norm equals 1:

$$N(q) = cos^2\left(\frac{\eta}{2}\right) + sin^2\left(\frac{\eta}{2}\right)\left(\hat{n}_x{}^2 + \hat{n}_y{}^2 + \hat{n}_z{}^2\right) = 1 \quad (64)$$

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 $\boldsymbol{v}$ by the quaternion q and its conjugate q^* is equivalent to multiplication of the vector by the corresponding rotation matrix $\boldsymbol{R}$:

$$\boldsymbol{Rv} = q^*\boldsymbol{v}q \tag{65}$$

3.2 Equivalence of Rotation Quaternion and Rotation Matrix

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 termed the Rodrigues rotation matrix and has form:

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

For any vector $\boldsymbol{v}$, the left side of equation (65) evaluates to:

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

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

The right side of equation (65) evaluates to:

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

$$\begin{aligned} = \left(cos\left(\frac{\eta}{2}\right) - \hat{n}_x sin\left(\frac{\eta}{2}\right)\boldsymbol{i} - \hat{n}_y sin\left(\frac{\eta}{2}\right)\boldsymbol{j} - \hat{n}_z sin\left(\frac{\eta}{2}\right)\boldsymbol{k}\right)(v_x\boldsymbol{i} + v_y\boldsymbol{j} + v_z\boldsymbol{k})\left(cos\left(\frac{\eta}{2}\right) + \hat{n}_x sin\left(\frac{\eta}{2}\right)\boldsymbol{i}\right. \\ \left. + \hat{n}_y sin\left(\frac{\eta}{2}\right)\boldsymbol{j} + \hat{n}_z sin\left(\frac{\eta}{2}\right)\boldsymbol{k}\right) \end{aligned} \tag{70}$$

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

Equations (68) and (71) match, proving the identity in equation (65).

Rotating the coordinate system by angle η plus 360° about the normalized rotation axis $\hat{\boldsymbol{n}}$ is obviously equivalent to rotating by angle η about the same normalized rotation axis $\hat{\boldsymbol{n}}$. However, from the definition of the rotation quaternion in equation (63), the extra 360° rotation results in the negation of the entire quaternion. In practice, this is more annoying than problematic since the negation of the quaternion appears twice on the right hand side of equation (65) and therefore has no effect when rotating vectors. It is, however,

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.

3.3 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:

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

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

$$q(q^*\boldsymbol{v}q)q^* = (qq^*)\boldsymbol{v}(qq^*) = \boldsymbol{v} \tag{73}$$

$$q^*(q\boldsymbol{v}q^*)q = (q^*q)\boldsymbol{v}(q^*q) = \boldsymbol{v} \tag{74}$$

3.4 Product of Rotation Quaternions

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

$$q_N{}^* \ldots (q_2{}^*(q_1{}^*\boldsymbol{v}q_1)q_2) \ldots q_N = (q_N{}^* \ldots q_2{}^*q_1{}^*)\boldsymbol{v}(q_1q_2 \ldots \ldots q_N) \tag{75}$$

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

$$q = q_1q_2q_3 \ldots q_N \tag{76}$$

3.5 Negation of Rotation Axis and Angle

Both a rotation matrix $\boldsymbol{R}$ and quaternion q are unchanged if both the rotation angle and axis are simultaneously inverted:

$$\begin{aligned}&\boldsymbol{R}(-\hat{\boldsymbol{n}}, -\eta)\\ &= \begin{pmatrix} \hat{n}_x{}^2 + (1-\hat{n}_x{}^2)cos(-\eta) & \hat{n}_x\hat{n}_y(1-cos(-\eta)) - \hat{n}_zsin(-\eta) & \hat{n}_x\hat{n}_z(1-cos(-\eta)) + \hat{n}_ysin(-\eta) \\ \hat{n}_x\hat{n}_y(1-cos(-\eta)) + \hat{n}_zsin(-\eta) & \hat{n}_y{}^2 + (1-\hat{n}_y{}^2)cos(-\eta) & \hat{n}_y\hat{n}_z(1-cos(-\eta)) - \hat{n}_xsin(-\eta) \\ \hat{n}_x\hat{n}_z(1-cos(-\eta)) - \hat{n}_ysin(-\eta) & \hat{n}_y\hat{n}_z(1-cos(-\eta)) + \hat{n}_xsin(-\eta) & \hat{n}_z{}^2 + (1-\hat{n}_z{}^2)cos(-\eta) \end{pmatrix}\end{aligned} \tag{77}$$

$$= \begin{pmatrix} \hat{n}_x{}^2 + (1-\hat{n}_x{}^2)cos\eta & \hat{n}_x\hat{n}_y(1-cos\eta) + \hat{n}_zsin\eta & \hat{n}_x\hat{n}_z(1-cos\eta) - \hat{n}_ysin\eta \\ \hat{n}_x\hat{n}_y(1-cos\eta) - \hat{n}_zsin\eta & \hat{n}_y{}^2 + (1-\hat{n}_y{}^2)cos\eta & \hat{n}_y\hat{n}_z(1-cos\eta) + \hat{n}_xsin\eta \\ \hat{n}_x\hat{n}_z(1-cos\eta) + \hat{n}_ysin\eta & \hat{n}_y\hat{n}_z(1-cos\eta) - \hat{n}_xsin\eta & \hat{n}_z{}^2 + (1-\hat{n}_z{}^2)cos\eta \end{pmatrix} = \boldsymbol{R}(\hat{\boldsymbol{n}}, \eta) \tag{78}$$

$$q(-\hat{\boldsymbol{n}}, -\eta) = cos\left(\frac{-\eta}{2}\right) - \hat{\boldsymbol{n}}sin\left(\frac{-\eta}{2}\right) = cos\left(\frac{\eta}{2}\right) + \hat{\boldsymbol{n}}sin\left(\frac{\eta}{2}\right) = q(\hat{\boldsymbol{n}}, \eta) \tag{79}$$

3.6 Square Root of Rotation Quaternion

The square root $\sqrt{q}$ of the rotation quaternion q for rotation by angle η about axis $\hat{\boldsymbol{n}}$ is, by inspection, the quaternion for rotation by angle $\left(\frac{\eta}{2}\right)$ about the same axis $\hat{\boldsymbol{n}}$:

$$q(\hat{\boldsymbol{n}}\eta) = cos\left(\frac{\eta}{2}\right) + \hat{\boldsymbol{n}}sin\left(\frac{\eta}{2}\right) = \left\{cos\left(\frac{\eta}{4}\right) + \hat{\boldsymbol{n}}sin\left(\frac{\eta}{4}\right)\right\}\left\{cos\left(\frac{\eta}{4}\right) + \hat{\boldsymbol{n}}sin\left(\frac{\eta}{4}\right)\right\} = q\left(\frac{\hat{\boldsymbol{n}}\eta}{2}\right)q\left(\frac{\hat{\boldsymbol{n}}\eta}{2}\right) \quad \textbf{(80)}$$

Standard double angle trigonometric identities can be used to simplify the calculation of the components of the square root quaternion:

$$\left|cos\left(\frac{\eta}{4}\right)\right| = \sqrt{\frac{1 + cos\left(\frac{\eta}{2}\right)}{2}} = \sqrt{\frac{1 + q_0}{2}} \quad \textbf{(81)}$$

$$\left|sin\left(\frac{\eta}{4}\right)\right| = \sqrt{\frac{1 - cos\left(\frac{\eta}{2}\right)}{2}} = \sqrt{\frac{1 - q_0}{2}} \quad \textbf{(82)}$$

The scaling factor for the vector component of the square root is the ratio:

$$\left|\frac{sin\left(\frac{\eta}{4}\right)}{sin\left(\frac{\eta}{2}\right)}\right| = \frac{\sqrt{1 - cos\left(\frac{\eta}{2}\right)}}{\sqrt{2}\sqrt{1 - cos^2\left(\frac{\eta}{2}\right)}} = \frac{\sqrt{1 - q_0}}{\sqrt{2}\sqrt{1 - {q_0}^2)}} = \frac{\sqrt{1 - q_0}}{\sqrt{2}\sqrt{1 - q_0}\sqrt{1 + q_0}} = \frac{1}{\sqrt{2 + 2q_0}} \quad \textbf{(83)}$$

The square root $\sqrt{q}$ of rotation quaternion q then equals:

$$\sqrt{q} = \sqrt{\{q_0, q_1, q_2, q_3\}} = \left\{\sqrt{\frac{1 + q_0}{2}}, \frac{q_1}{\sqrt{2 + 2q_0}}, \frac{q_2}{\sqrt{2 + 2q_0}}, \frac{q_3}{\sqrt{2 + 2q_0}}\right\} \quad \textbf{(84)}$$

Direct expansion of the product $\sqrt{q}\sqrt{q}$ equals q as expected. Substituting into equation (27) gives:

$$\sqrt{q}\sqrt{q} = \left(\frac{1 + q_0}{2}\right) - \left(\frac{{q_1}^2 + {q_2}^2 + {q_3}^2}{2 + 2q_0}\right) + 2\sqrt{\frac{1 + q_0}{2}}\left(\frac{q_1}{\sqrt{2 + 2q_0}}\boldsymbol{i} + \frac{q_2}{\sqrt{2 + 2q_0}}\boldsymbol{j} + \frac{q_3}{\sqrt{2 + 2q_0}}\boldsymbol{k}\right) \quad \textbf{(85)}$$

$$= \left(\frac{1 + q_0}{2}\right) - \frac{(1 - {q_0}^2)}{2(1 + q_0)} + q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k} = q_0 + q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k} = q \quad \textbf{(86)}$$

Mathematically, a second square root exists with opposite sign:

$$\sqrt{q} = -\left\{\sqrt{\frac{1+q_0}{2}}, \frac{q_1}{\sqrt{2+2q_0}}, \frac{q_2}{\sqrt{2+2q_0}}, \frac{q_3}{\sqrt{2+2q_0}}\right\} \tag{87}$$

Since this is simply the negated quaternion, which corresponds to a meaningless additional 360° rotation, it can be ignored.

3.7 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 the NXP 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 $\boldsymbol{v}$, 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 (65) into its components gives the identity:

$$\begin{pmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} = (q_0 - q_1\boldsymbol{i} - q_2\boldsymbol{j} - q_3\boldsymbol{k})(v_x\boldsymbol{i} + v_y\boldsymbol{j} + v_z\boldsymbol{k})(q_0 + q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k}) \tag{88}$$

Expanding the right hand side and re-arranging gives:

$$\begin{pmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} = \begin{pmatrix} {q_0}^2 + {q_1}^2 - {q_2}^2 - {q_3}^2 & 2q_1q_2 + 2q_0q_3 & 2q_1q_3 - 2q_0q_2 \\ 2q_1q_2 - 2q_0q_3 & {q_0}^2 - {q_1}^2 + {q_2}^2 - {q_3}^2 & 2q_2q_3 + 2q_0q_1 \\ 2q_1q_3 + 2q_0q_2 & 2q_2q_3 - 2q_0q_1 & {q_0}^2 - {q_1}^2 - {q_2}^2 + {q_3}^2 \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} \tag{89}$$

Because equation (89) holds for all vectors $\boldsymbol{v}$, it follows that:

$$\begin{pmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{pmatrix} = \begin{pmatrix} {q_0}^2 + {q_1}^2 - {q_2}^2 - {q_3}^2 & 2q_1q_2 + 2q_0q_3 & 2q_1q_3 - 2q_0q_2 \\ 2q_1q_2 - 2q_0q_3 & {q_0}^2 - {q_1}^2 + {q_2}^2 - {q_3}^2 & 2q_2q_3 + 2q_0q_1 \\ 2q_1q_3 + 2q_0q_2 & 2q_2q_3 - 2q_0q_1 & {q_0}^2 - {q_1}^2 - {q_2}^2 + {q_3}^2 \end{pmatrix} \tag{90}$$

The normalization constraint for a rotation quaternion is:

$${q_0}^2 + {q_1}^2 + {q_2}^2 + {q_3}^2 = 1 \tag{91}$$

Substituting equation (91) into equation (90) gives a slightly simpler expression for converting a rotation quaternion to a rotation matrix:

$$\begin{pmatrix} R_{xx} & R_{xy} & R_{xz} \\ R_{yx} & R_{yy} & R_{yz} \\ R_{zx} & R_{zy} & R_{zz} \end{pmatrix} = \begin{pmatrix} 2({q_0}^2 + {q_1}^2) - 1 & 2(q_1 q_2 + q_0 q_3) & 2(q_1 q_3 - q_0 q_2) \\ 2(q_1 q_2 - q_0 q_3) & 2({q_0}^2 + {q_2}^2) - 1 & 2(q_2 q_3 + q_0 q_1) \\ 2(q_1 q_3 + q_0 q_2) & 2(q_2 q_3 - q_0 q_1) & 2({q_0}^2 + {q_3}^2) - 1 \end{pmatrix} \quad (92)$$

4.2 Quaternion from Rotation Matrix

Equation (92) 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:

$$1 + tr(\boldsymbol{R}) = 1 + 2({q_0}^2 + {q_1}^2) - 1 + 2({q_0}^2 + {q_2}^2) - 1 + 2({q_0}^2 + {q_3}^2) - 1 \quad (93)$$

$$= 2{q_0}^2 + 2{q_1}^2 + 2{q_0}^2 + 2{q_2}^2 - 1 + 2{q_0}^2 + 2{q_3}^2 - 1 \quad (94)$$

$$= \left(2{q_0}^2 + 2{q_1}^2 + 2{q_2}^2 + 2{q_3}^2\right) + 2{q_0}^2 - 1 + 2{q_0}^2 - 1 \quad (95)$$

Using the result that the quaternion has unit norm gives:

$$1 + tr(\boldsymbol{R}) = 4{q_0}^2 \quad (96)$$

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 (96) can, therefore, always be taken, giving:

$$q_0 = \frac{\sqrt{1 + tr(\boldsymbol{R})}}{2} = \frac{\sqrt{1 + R_{xx} + R_{yy} + R_{zz}}}{2} \quad (97)$$

Differencing elements across the diagonal gives the solution for the vector components of the rotation quaternion:

$$R_{yz} - R_{zy} = 4q_0 q_1 \Rightarrow q_1 = \frac{\left(R_{yz} - R_{zy}\right)}{4q_0} \quad (98)$$

$$R_{zx} - R_{xz} = 4q_0 q_2 \Rightarrow q_2 = \frac{(R_{zx} - R_{xz})}{4q_0} \quad (99)$$

$$R_{xy} - R_{yx} = 4q_0 q_3 \Rightarrow q_3 = \frac{\left(R_{xy} - R_{yx}\right)}{4q_0} \quad (100)$$

Equations (98) to (100) 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). A fallback algorithm is needed which uses the elements on the leading diagonal to give:

$$R_{xx} = 2({q_0}^2 + {q_1}^2) - 1 \Rightarrow q_1 = \pm\sqrt{\frac{1 + R_{xx}}{2} - {q_0}^2} \quad \textbf{(101)}$$

$$R_{yy} = 2({q_0}^2 + {q_2}^2) - 1 \Rightarrow q_2 = \pm\sqrt{\frac{1 + R_{yy}}{2} - {q_0}^2} \quad \textbf{(102)}$$

$$R_{zz} = 2({q_0}^2 + {q_3}^2) - 1 \Rightarrow q_3 = \pm\sqrt{\frac{1 + R_{zz}}{2} - {q_0}^2} \quad \textbf{(103)}$$

The unknown signs in equations (101) to (103) can be resolved by taking the signs of equations (98) to (100) and using the fact that q_0 is always nonnegative:

$$sign(q_1) = sign\left(R_{yz} - R_{zy}\right) \quad \textbf{(104)}$$

$$sign(q_2) = sign(R_{zx} - R_{xz}) \quad \textbf{(105)}$$

$$sign(q_3) = sign\left(R_{xy} - R_{yx}\right) \quad \textbf{(106)}$$

5. Converting between Quaternion and Rotation Vector

5.1 Rotation Vector From Quaternion

The definition of the rotation quaternion q in equation (63) shows that it is closely linked to the equivalent rotation vector $\hat{\boldsymbol{n}}\eta$ defined as the product of normalized rotation axis $\hat{\boldsymbol{n}}$ and the rotation angle η. Inverting the process to recover the rotation vector from the quaternion is straightforward.

$$q = q_0 + q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k} = cos\left(\frac{\eta}{2}\right) + \hat{\boldsymbol{n}}sin\left(\frac{\eta}{2}\right) \quad \textbf{(107)}$$

Equating the scalar components gives:

$$q_0 = cos\left(\frac{\eta}{2}\right) \Rightarrow \eta = 2cos^{-1}(q_0) \quad \textbf{(108)}$$

Because q_0 varies between 0 and 1, the rotation angle η in equation (108) has the required range 0° to 180°.

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

$$q_1 = sin\left(\frac{\eta}{2}\right) n_x \Rightarrow n_x = \frac{q_1}{sin\left(\frac{\eta}{2}\right)} \quad \textbf{(109)}$$

$$q_2 = sin\left(\frac{\eta}{2}\right) n_y \Rightarrow n_y = \frac{q_2}{sin\left(\frac{\eta}{2}\right)} \quad \textbf{(110)}$$

$$q_3 = sin\left(\frac{\eta}{2}\right) n_z \Rightarrow n_z = \frac{q_3}{sin\left(\frac{\eta}{2}\right)} \quad \textbf{(111)}$$

For the case where $sin\left(\frac{\eta}{2}\right) = 0$, the rotation angle η is also zero since η is in the range 0^o to 180^o. The rotation axis $\hat{\boldsymbol{n}}$ is then undefined which makes physical sense for the case of zero rotation angle η.

5.2 Quaternion From Rotation Vector

Equation (63) defines the rotation quaternion explicitly in terms of the rotation vector axis $\hat{\boldsymbol{n}}$ and rotation angle η.

$$q = cos\left(\frac{\eta}{2}\right) + \hat{\boldsymbol{n}} sin\left(\frac{\eta}{2}\right) \quad \textbf{(112)}$$

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 separately 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:

$$y[n] = (1 - \alpha)y[n-1] + \alpha x[n] \tag{113}$$

where:

$$0 < \alpha \leq 1 \tag{114}$$

The transfer function $H(z)$ is:

$$H(z) = \frac{\alpha}{1 - (1 - \alpha)z^{-1}} \tag{115}$$

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:

$$y[n] = (1 - \alpha)^n \tag{116}$$

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

$$\frac{1}{e} = (1 - \alpha)^N \Rightarrow N = \frac{-1}{ln(1 - \alpha)} \tag{117}$$

For small α:

$$N = \frac{1}{\alpha} \tag{118}$$

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

In C code, equation (113) can be efficiently written as:

$$\text{yn} \mathrel{+}= \text{alpha} * (\text{xn} - \text{yn}); \tag{119}$$

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 (119) makes it clear that the low-pass estimate yn is updated by α times the difference between the current input xn and the previous low-pass filtered estimate yn.

The low-pass estimate yn is therefore exponentially steered towards the input sequence xn. 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:

$$q[n] = q_{LP}[n-1]\Delta q[n] \Rightarrow \Delta q[n] = {q_{LP}}^*[n-1]q[n] \quad \textbf{(120)}$$

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

$$\Delta q[n] = cos\left(\frac{\eta}{2}\right) + \hat{\boldsymbol{n}} sin\left(\frac{\eta}{2}\right) \quad \textbf{(121)}$$

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

$$q_{LP}[n] = q_{LP}[n-1]\Delta q[n] = q_{LP}[n-1]{q_{LP}}^*[n-1]q[n] = q[n] \quad \textbf{(122)}$$

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:

$$\Delta q'[n] = \Delta {q_0}'[n] + \alpha \hat{\boldsymbol{n}} sin\left(\frac{\eta}{2}\right) \quad \textbf{(123)}$$

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:

$$\alpha' = \alpha + (1-\alpha)\sqrt{(1-\Delta q_0[n]^2)} = \alpha + (1-\alpha)\sqrt{1-cos^2\left(\frac{\eta}{2}\right)} = \alpha + (1-\alpha)\left|sin\left(\frac{\eta}{2}\right)\right| \quad (124)$$

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

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

$$\alpha' \approx \alpha + \frac{(1-\alpha)|\eta|}{2} \approx \alpha + \frac{|\eta|}{2} \; for\; \eta\; in\; radians \quad (125)$$

α' therefore equals α in the limit of small angles η but increases linearly 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 180º, $\Delta q_0[n]$ 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.

$$\alpha' = 1 \quad (126)$$

7. Quaternion Derivative

7.1 Definition

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

$$\frac{dq(t)}{dt} = \dot{q}(t) = \lim_{\delta t \to 0}\left\{\frac{q(t+\delta t) - q(t)}{\delta t}\right\} \quad (127)$$

$$\Rightarrow q(t+dt) = q(t) + dt\dot{q}(t) \quad (128)$$

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 $\delta q(t)$:

$$q(t+\delta t) = q(t)\delta q(t) \quad (129)$$

With the assumption that incremental change in orientation results from a constant angular velocity $\boldsymbol{\omega}$ over the time interval δt then using the definition of the rotation quaternion in terms of rotation angle gives:

$$\delta q = \{q_0, \delta q_1, \delta q_2, \delta q_3\} = \left\{q_0, sin\left(\frac{\omega_x \delta t}{2}\right), sin\left(\frac{\omega_y \delta t}{2}\right), sin\left(\frac{\omega_z \delta t}{2}\right)\right\} \quad (130)$$

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:

$$dq = \{q_0, dq_1, dq_2, dq_3\} = \left\{1, \left(\frac{\omega_x dt}{2}\right), \left(\frac{\omega_y dt}{2}\right), \left(\frac{\omega_z dt}{2}\right)\right\} \quad (131)$$

and:

$$q(t + dt) = q(t)dq = q(t)\left\{1, \left(\frac{\omega_x dt}{2}\right), \left(\frac{\omega_y dt}{2}\right), \left(\frac{\omega_z dt}{2}\right)\right\} \quad (132)$$

Combining equations (128) and (132) gives:

$$q(t) + dt\dot{q}(t) = q(t)\left\{1, \left(\frac{\omega_x dt}{2}\right), \left(\frac{\omega_y dt}{2}\right), \left(\frac{\omega_z dt}{2}\right)\right\} \quad (133)$$

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

$$q(t) + dt\dot{q}(t) = q(t) + \left(\frac{dt}{2}\right) q(t)\{0, \omega_x, \omega_y, \omega_z\} \quad (134)$$

$$\Rightarrow \dot{q}(t) = \left(\frac{1}{2}\right) q(t)\boldsymbol{\omega}(t) \quad (135)$$

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

8. Legal information

8.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

8.2 Disclaimers

Information in this document is provided solely to enable system and software implementers to use NXP 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. NXP reserves the right to make changes without further notice to any products herein. NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP 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 NXP 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. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/salestermsandconditions.

8.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

NXP, the NXP logo, Freescale, and the Freescale logo are trademarks of NXP B.V. ARM and Cortex are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere.

9. List of tables

10. Contents

Date of release: 21 June 2016
Document identifier: AN5022