**Quaternion Algebra and Rotation Quaternions**

**Technical Note**

Document XXX

Version: Draft

Authors: Mark Pedley and Michael Stanley

Date: xx Aug 2014



Table of Contents

[1 Introduction 4](#_Toc395529254)

[1.1 Summary 4](#_Toc395529255)

[1.2 Functions 4](#_Toc395529256)

[2 Quaternion Algebra 6](#_Toc395529257)

[2.1 Introduction 6](#_Toc395529258)

[2.2 Equality of Two Quaternions 6](#_Toc395529259)

[2.3 Addition of Two Quaternions 6](#_Toc395529260)

[2.4 Product of two Quaternions 6](#_Toc395529261)

[2.5 Product of Quaternion and Scalar 7](#_Toc395529262)

[2.6 Product of a Quaternion with a Vector 7](#_Toc395529263)

[2.7 Quaternion Conjugate 8](#_Toc395529264)

[2.8 Quaternion Norm 8](#_Toc395529265)

[2.9 Quaternion Inverse and Division 9](#_Toc395529266)

[2.10 Vector Representation of Quaternion Product 9](#_Toc395529267)

[3 Rotation Quaternion 11](#_Toc395529268)

[3.1 Definition in Terms of Rotation Vector 11](#_Toc395529269)

[3.2 Equivalence of Rotation Quaternion and Rotation Matrix 11](#_Toc395529270)

[3.3 Inverse Rotation Quaternion 11](#_Toc395529271)

[3.4 Product of Rotation Quaternions 12](#_Toc395529272)

[3.5 Negation of Rotation Axis and Angle 12](#_Toc395529273)

[3.6 Coordinate Frame Rotation Standard 12](#_Toc395529274)

[4 Computing Rotation Vector from Quaternion 13](#_Toc395529275)

[5 Low Pass Filtering Orientation Quaternion 14](#_Toc395529276)

[5.1 Introduction 14](#_Toc395529277)

[5.2 Quaternion Low Pass Filter 14](#_Toc395529278)

[6 Low Pass Filtering Orientation Quaternion 16](#_Toc395529279)

**Glossary**

Components of quaternion

Quaternion conjugate of

Scalar and vector representation of quaternion

Inverse or reciprocal of quaternion

Scalar component of quaternion

Vector component of quaternion

Scalar product of vectors and

Vector product of vectors and

Unit vector representing rotation axis

Norm or magnitude of quaternion

Rotation quaternions about the *x*, *y* and *z* axes

Rotation matrices around *x*, *y* and *z* axes

Real number

Complex number

General rotation angle

Roll angle

Pitch angle

Yaw angle

Compass heading angle

## Introduction

### Summary

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

### 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. |

## Quaternion Algebra

### Introduction

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

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

Eq 2.1.1

a quaternion has four components:

Eq 2.1.2

where , , and are real numbers.

is termed the scalar component and the vector component of the quaternion.

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

Eq 2.1.3

If the scalar component is zero, the quaternion is termed a pure quaternion or vector. If the vector component 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.

### Equality of Two Quaternions

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

Eq 2.2.1

### Addition of Two Quaternions

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

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:

Eq 2.3.2

Eq 2.3.3

### Product of two Quaternions

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

Eq 2.4.1

Eq 2.4.2

Eq 2.4.3

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

Eq 2.4.4

Eq 2.4.5

Eq 2.4.6

Eq 2.4.7

Eq 2.4.8

Eq 2.4.9

Eq 2.4.10

A consequence of equations 2.4.7 to 2.4.10 is that:

Eq 2.4.11

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

Eq 2.4.12

The four components of the product quaternion are therefore:

Eq 2.4.13

Eq 2.4.14

Eq 2.4.15

Eq 2.4.16

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

Eq 2.4.17

Brute force evaluation proves that quaternion multiplication is associative.

Eq 2.4.18

Eq 2.4.19

### 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 is a scalar so that then:

Eq 2.5.1

Eq 2.5.2

Eq 2.5.3

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

### Product of a Quaternion with a Vector

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

Eq 2.6.1

Eq 2.6.2

Inspection of equations 2.6.1 and 2.6.2 shows that the product of the quaternion and vector does not commute:

Eq 2.6.3

### Quaternion Conjugate

The quaternion conjugate is defined as:

Eq 2.7.1

From the definition of the quaternion product it can be shown that :

Eq 2.7.2

The product is:

Eq 2.7.3

Eq 2.7.4

Eq 2.7.5

Eq 2.7.6

Eq 2.7.7

Simple extension to higher order products gives:

Eq 2.7.8

The sum of the quaternion and its conjugate is the scalar :

Eq 2.7.9

### Quaternion Norm

The quaternion norm or magnitude is defined as:

Eq 2.8.1

The products of a quaternion with its conjugate evaluate to:

Eq 2.8.2

Eq 2.8.3

Eq 2.8.4

Eq 2.8.5

Eq 2.8.6

Eq 2.8.7

Eq 2.8.8

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

Eq 2.8.9

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

Eq 2.8.10

Eq 2.8.11

### Quaternion Inverse and Division

The quaternion inverse is defined to be the quaternion which satisfies:

Eq 2.9.1

Pre- and post-multiplication of by evaluates to:

Eq 2.9.2

Eq 2.9.3

The quaternion inverse is therefore identified for all quaternions with non-zero norm as:

Eq 2.9.4

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

Eq 2.9.5

### Vector Representation of Quaternion Product

The standard scalar and vector products between the two vectors and are defined as:

Eq 2.10.1

Eq 2.10.2

The product between two quaternions and 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 :

Eq 2.10.3

Eq 2.10.4

Comparison with equation 2.4.12 gives the identity:

Eq 2.10.5

## Rotation Quaternion

### Definition in Terms of Rotation Vector

The rotation quaternion for a rotation of the coordinate system about normalized rotation axis by angle is defined to be:

Eq 3.1.1

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

is obviously a unit quaternion since its norm equals 1:

Eq 3.1.2

The next section proves that the rotation quaternion is related to the general rotation matrix operating on a vector by:

Eq 3.1.3

### Equivalence of Rotation Quaternion and Rotation Matrix

The left hand side of equation 3.1.3 evaluates to:

Eq 3.2.1

Eq 3.2.2

Eq 3.2.3

The general rotation matrix which transforms a vector as a result of a rotation of the coordinate system around the axis by angle is:

Eq 3.2.4

The right hand side of equation 3.1.3 evaluates to:

Eq 3.2.5

Eq 3.2.6

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

### Inverse Rotation Quaternion

From the definition of , it follows that is the rotation operator about the same axis but by angle and is therefore the inverse of the rotation quaternion :

Eq 3.3.1

The inverse nature of the operators and can also be shown by computing the effects of the product rotations i) followed by and ii) followed by applied to the vector using the associative property of quaternion multiplication:

Eq 3.3.2

Eq 3.3.3

### Product of Rotation Quaternions

The result of successively applying rotation quaternions followed by through to vector is:

Eq 3.4.1

The rotation quaternion equivalent to the successive rotations represented by quaternions to is then:

Eq 3.4.2

### Negation of Rotation Axis and Angle

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

Eq 3.5.1

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.

### Coordinate Frame Rotation Standard

Some texts define the quaternion rotation operator on vector to be instead of . The explanation is that the operator transforms the vector as a result of rotation of the coordinate system by angle whereas the operator rotates the vector by angle 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 (which is typically the earth's gravitational or geomagnetic field and therefore fixed in the earth reference frame).

## Quaternion Derivative

### Definition

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

Eq 4.1.1

A consequence of equation 4.1.1 is that in the limit:

Eq 4.1.2

### Derivation

The orientation can also be propagated forward in time by computing the product of the current orientation quaternion and the incremental quaternion .

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 then:

Eq 4.2.2

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

In the limit of the integration time becoming the infinitesimal :

Eq 4.2.3

and:

Eq 4.2.4

Combining equations 4.1.2 and 4.2.4 gives:

Eq 4.2.5

Since quaternion multiplication is distributive:

Eq 4.2.6

Eq 4.2.7

## 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.

Eq 5.1

Equating the scalar components gives:

Eq 5.2

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

Equating the remaining three components of the quaternion gives:

Eq 5.3

Eq 5.4

Eq 5.5

In the case where is zero, the rotation angle is zero and the rotation axis is irrelevant. The rotation vector can then be safely set to zero.

## Low Pass Filtering Orientation Quaternion

### 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:

Eq 6.1.1

where:

Eq 6.1.2

The time constant in samples is approximately equal to the reciprocal of . The case 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 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.

### 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 at iteration n between the previous estimate of the low pass filtered quaternion estimate and the current quaternion are related by:

Eq 6.2.1

Eq 6.2.2

If has negative scalar component 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 which equals the cosine of half the rotation angle encoded in 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 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 is the scalar component of , the low pass filter constant is set to the experimentally determined equation:

Eq 6.2.3

is not permitted to exceed the all pass value of . The term is the nominal filter coefficient and typically has value of about 0.5secs.

The low pass filter update equation is then:

Eq 6.2.4

In plain English, for , 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".