libdq

Generated by Doxygen 1.6.3

Tue Apr 24 22:38:37 2012

Contents

Chapter 1

libdq doxygen documentation

Author

Edgar Simo-Serra <bobbens@gmail.com>

Version

2.0

Date

February 2011

1.1 License

1.2 Overview

This is a library for using and manipulating unit dual quaternions. Unit dual quaternions are useful for describing rigid body movements using screw theory.

If you use this library please reference it.

1.3 Notation

The naming scheme used is more or less consistant and the following:

- Capital letters are for dual quaternions, they shall be denoted with a "hat" in the documentation as such: $\widehat{Q}$.
- Lowercase will be used for vectors or simple quaternions. In the case of simple quaternions they shall have a "hat" as such: $\widehat{q}$.
- Individual members of quaternions or dual quaternions are lowercase and will be indicated with subscripted index as such: a_0.

For functions:

- Input dual quaternions should be called Q if there is one input or P and Q if there are two.
- Output dual quaternions should be called O or a combination of the input quaternions in the case it is multiplying or transforming like for example PQ.

For notation and quaternion definition please refer to the documentation for the dual quaternion type dq_t .

1.4 Usage

To use dual quaternion library you need to include it as <dq.h>. When linking you should pass -ldq. A simple example would be:

```
#include <dq/dq.h>

int main( int argc, char *argv[] )
{
   dq_t Q;
   double t[3] = { 1.0, 2.0, 3.0 };

   dq_cr_point( Q, t );
   dq_print_vert( Q );

   return 0;
}
```

The example would just create a point dual quaternion and display it. To compile you would have to use:

```
$ gcc -ldq dq_test.c -o dq_test
```

Auxiliary functions are also provided to help manipulate common data structures when working with dual quaternions.

1.5 Changelog

- Version 2.1, April 2012
  - Made it more clear license is LGPL
  - Fixed bug in Lua handling of matrices
  - Added defines for getting version
  - Added dq_version, dq_ch_plane_point, dq_cr_plane
  - Minor correctness fixes
- Version 2.0, April 2011
  - Lua bindings
- Version 1.5, April 2011
  - Install headers into /usr/include/dq by default instead of prefixing with dq_
  - Updated documentation
  - Make docs now places final documentation in /
- Version 1.4, February 2011
  - Fixed major issue in dual quaternion conjugation
  - Added dq_op_extract
  - Added mat3_solve
  - Added vec3_sign
  - Added vec3_distance
  - Minor doxygen improvements
- Version 1.3, December 2010
  - Cleaned up and documented the auxiliary function files so they can be used.
- Version 1.2, December 2010
  - Added dq_op_sign to change sign of a dual quaternion.
  - dq_ch_cmp and dq_ch_cmpV now take into account the fact it could be with a different sign.
- Version 1.1, November 2010
  - Fixed dual quaternion multiplication
  - Implemented more strict unit tests
  - Misc fixes
- Version 1.0, November 2010
  - Initial Revision

1.6 References

- E. Simo-Serra, Chapter 4 of Kinematic Model of the Hand using Computer Vision (Degree Thesis). BarcelonaTECH (UPC), April 2011.
- J. M. Selig. Geometric Fundamentals of Robotics (Monographs in Computer Science). Springer, 2nd edition, November 2004.
- J. M. McCarthy, Introduction to theoretical kinematics, MIT Press, Cambridge, MA, 1990
- A. Perez, Kinematics of Robots (unpublished as of this writing)

1.7 Citation

There is no real convention for citing software, the following is a proposal.

```
@MISC{esimolibdq,
   author = {Edgar Simo-serra},
   title = {libdq: {D}ual {Q}uaternion {L}ibrary},
   year = {2011},
   howpublished = {\url{https://github.com/bobbens/libdq}},
}
```

1.8 Acknowledgements

A big thanks to Alba Perez for having the patience to deal with my repetitive boring dual quaternion questions and for lending me her notes.

See also

dq_t
Dual Quaternion Creation Functions
Dual Quaternion Operations
Dual Quaternion Check Functions
Dual Quaternion Miscellaneous Functions

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Module Documentation

4.1 Dual Quaternion Creation Functions

Set of functions to create dual quaternions.

Functions

- void dq_cr_rotation (dq_t O, double theta, const double s[3], const double c[3])

  Creates a pure rotation dual quaternion.

- void dq_cr_rotation_plucker (dq_t O, double theta, const double s[3], const double s0[3])

  Creates a pure rotation dual quaternion using plucker coordinates.

- void dq_cr_rotation_matrix (dq_t O, double R[3][3])

  Creates a pure rotation dual quaternion from a rotation matrix.

- void dq_cr_translation (dq_t O, double t, const double s[3])

  Creates a pure translation dual quaternion.

- void dq_cr_translation_vector (dq_t O, const double t[3])

  Creates a pure translation dual quaternion from a traslation vector.

- void dq_cr_point (dq_t O, const double pos[3])

  Creates a dual quaternion representing a point.

- void dq_cr_line (dq_t O, const double s[3], const double c[3])

  Creates a dual quaternion representing a line.

- void dq_cr_line_plucker (dq_t O, const double s[3], const double s0[3])

  Creates a dual quaternion representing a line from plucker coordinates.

- void dq_cr_plane (dq_t O, const double n[3], const double d)

  Creates a unit dual quaternion representing a plane.

- void dq_cr_homo (dq_t O, double R[3][3], const double d[3])

  Creates a dual quaternion from a homogeneous transformation matrix.

- void dq_cr_copy (dq_t O, const dq_t Q)

  Copies a dual quaternion.

- void dq_cr_conj (dq_t O, const dq_t Q)

  Conjugates a dual quaternion.

- void dq_cr_inv (dq_t O, const dq_t Q)

  Inverts a dual quaternion.

4.1.1 Detailed Description

Set of functions to create dual quaternions.

4.1.2 Function Documentation

4.1.2.1 void dq_cr_conj (dq_t *O*, const dq_t *Q*)

Conjugates a dual quaternion.

$$\widehat{O} = \widehat{Q}^* = \widehat{q}^* + \epsilon\widehat{q}^*$$

Parameters

→ ***O*** Dual quaternion created (conjugated).

← ***Q*** Dual quaternion to conjugate.

4.1.2.2 void dq_cr_copy (dq_t *O*, const dq_t *Q*)

Copies a dual quaternion.

Parameters

→ ***O*** Dual quaternion created.

← ***Q*** Dual quaternion to copy.

4.1.2.3 void dq_cr_homo (dq_t *O*, double *R*[3][3], const double *d*[3])

Creates a dual quaternion from a homogeneous transformation matrix.

Parameters

→ ***O*** Dual quaternion created.

← ***R*** Rotation matrix.

← ***d*** Translation vector.

4.1.2.4 void dq_cr_inv (dq_t *O*, const dq_t *Q*)

Inverts a dual quaternion.

$$\widehat{O} = \widehat{Q}^{-1} = \frac{\widehat{Q}^*}{\|\widehat{Q}\|^2}$$

First we multiply the dual quaternion by it's conjugate:

$$\widehat{Q}\widehat{Q}^* = \|\widehat{Q}\| = (q_0q_0 + q_1q_1 + q_2q_2 + q_3q_3) + \epsilon 2(q_0q_7 + q_1q_4 + q_2q_5 + q_3q_6)$$

To get rid of the ϵ term we multiply by $(q_0q_0 + q_1q_1 + q_2q_2 + q_3q_3) - \epsilon 2(q_0q_7 + q_1q_4 + q_2q_5 + q_3q_6)$.

$$\begin{aligned}((q_0q_0 + q_1q_1 + q_2q_2 + q_3q_3) + \epsilon 2(q_0q_7 + q_1q_4 + q_2q_5 + q_3q_6))\\ ((q_0q_0 + q_1q_1 + q_2q_2 + q_3q_3) - \epsilon 2(q_0q_7 + q_1q_4 + q_2q_5 + q_3q_6))\\ = (q_0q_0 + q_1q_1 + q_2q_2 + q_3q_3)^2\end{aligned}$$

Due to the fact that the first quaternion represents the rotation, it can be proven that it's equal to the identity:

$$\begin{aligned}\|\widehat{q}\| &= \|(\cos\left(\frac{\theta}{2}\right) + \sin\left(\frac{\theta}{2}\right) s)\| \\ &= (\sin^2\left(\frac{\theta}{2}\right) + \cos^2\left(\frac{\theta}{2}\right)) = 1 = q_0q_0 + q_1q_1 + q_2q_2 + q_3q_3\end{aligned}$$

Therefore with the multiplications we've done we have gotten what we wanted. If we analyze the multiplication we did we see it was:

$$\begin{aligned}&\widehat{Q}^*((q_0q_0 + q_1q_1 + q_2q_2 + q_3q_3) - \epsilon 2(q_0q_7 + q_1q_4 + q_2q_5 + q_3q_6)) = \\ &(q_0q_0 + q_1q_1 + q_2q_2 + q_3q_3)((q_0 - q_1i - q_2j - q_3k) + \epsilon(q_7 - q_4i - q_5j - q_7k)) \\ &- 2(q_0q_7 + q_1q_4 + q_2q_5 + q_3q_6)(q_7 + q_4i - q_5j - q_6k)\end{aligned}$$

Parameters

→ ***O*** Dual quaternion created (inverted).

← ***Q*** Dual quaternion to invert.

4.1.2.5 void dq_cr_line (dq_t *O*, const double *s*[3], const double *c*[3])

Creates a dual quaternion representing a line.

Parameters

→ ***O*** Dual quaternion created.

← ***s*** Direction vector of the line.

← ***c*** A point of the line.

See also

dq_cr_line_plucker
dq_op_f2g

4.1.2.6 void dq_cr_line_plucker (dq_t *O*, const double *s*[3], const double *s0*[3])

Creates a dual quaternion representing a line from plucker coordinates.

Parameters

→ ***O*** Dual quaternion created.

← ***s*** Direction vector of the line.

← ***s0*** The momento f the line.

See also

dq_cr_line
dq_op_f2g

4.1.2.7 void dq_cr_plane (dq_t *O*, const double *n*[3], const double *d*)

Creates a unit dual quaternion representing a plane.

Parameters

→ ***O*** Dual quaternion created.

← ***n*** Normal of the plane.

← ***d*** Distancefrom the origin to the plane.

4.1.2.8 void dq_cr_point (dq_t *O*, const double *pos*[3])

Creates a dual quaternion representing a point.

Parameters

→ ***O*** Dual quaternion created.

← ***pos*** Position of the point.

See also

dq_op_f4g

4.1.2.9 void dq_cr_rotation (dq_t *O*, double *theta*, const double *s*[3], const double *c*[3])

Creates a pure rotation dual quaternion.

Parameters

→ ***O*** Dual quaternion created.

← ***theta*** Angle to rotate.

← ***s*** Vector to rotate around (normalized).

← ***c*** Any point of the vector (to create plucker coordinates).

See also

dq_cr_rotation_plucker
dq_cr_rotation_matrix

4.1.2.10 void dq_cr_rotation_matrix (dq_t *O*, double *R*[3][3])

Creates a pure rotation dual quaternion from a rotation matrix.

Parameters

→ ***O*** Dual quaternion created.

← ***R*** 3x3 Rotation matrix.

See also

dq_cr_rotation
dq_cr_rotation_plucker

4.1.2.11 void dq_cr_rotation_plucker (dq_t *O*, double *theta*, const double *s*[3], const double *s0*[3])

Creates a pure rotation dual quaternion using plucker coordinates.

Parameters

→ ***O*** Dual quaternion created.

← ***theta*** Angle to rotate.

← ***s*** Vector to rotate around (normalized).

← ***s0*** Moment of the vector.

See also

dq_cr_rotation
dq_cr_rotation_matrix

4.1.2.12 void dq_cr_translation (dq_t *O*, double *t*, const double *s*[3])

Creates a pure translation dual quaternion.

Parameters

→ ***O*** Dual quaternion created.

← ***t*** Translation amount.

← ***s*** Translation vector (normalized).

See also

dq_cr_translation_vector

4.1.2.13 void dq_cr_translation_vector (dq_t *O*, const double *t*[3])

Creates a pure translation dual quaternion from a traslation vector.

Parameters

→ ***O*** Dual quaternion created.

← ***t*** Traslation vector.

See also

dq_cr_translation

4.2 Dual Quaternion Operations

Functions for operation on dual quaternions.

Functions

- void dq_op_norm2 (double *real, double *dual, const dq_t Q)

  Gets the square of the norm of a dual quaternion.

- void dq_op_add (dq_t O, const dq_t P, const dq_t Q)

  Adds two dual quaternions.

- void dq_op_sub (dq_t O, const dq_t P, const dq_t Q)

  Subtracts two dual quaternions.

- void dq_op_mul (dq_t PQ, const dq_t P, const dq_t Q)

  Multiplies to dual quaternions.

- void dq_op_sign (dq_t P, const dq_t Q)

  Swaps the sign of all the elements in a dual quaternion.

- void dq_op_f1g (dq_t ABA, const dq_t A, const dq_t B)

  Clifford conjugation transformation of type f_{1g} (Alba Perez notation).

- void dq_op_f2g (dq_t ABA, const dq_t A, const dq_t B)

  Clifford conjugation transformation of type f_{2g} (Alba Perez notation).

- void dq_op_f3g (dq_t ABA, const dq_t A, const dq_t B)

  Clifford conjugation transformation of type f_{3g} (Alba Perez notation).

- void dq_op_f4g (dq_t ABA, const dq_t A, const dq_t B)

  Clifford conjugation transformation of type f_{4g} (Alba Perez notation).

- void dq_op_extract (double R[3][3], double d[3], const dq_t Q)

  Extracts the rotation matrix and translation vector assosciated to a dual quaternion.

4.2.1 Detailed Description

Functions for operation on dual quaternions.

4.2.2 Function Documentation

4.2.2.1 void dq_op_add (dq_t *O*, const dq_t *P*, const dq_t *Q*)

Adds two dual quaternions.

$$\widehat{O} = \widehat{P} - \widehat{Q}$$

Parameters

→ ***O*** The result of the addition.
← ***P*** First quaternion to add.
← ***Q*** Second quaternion to add.

See also

dq_op_sub

4.2.2.2 void dq_op_extract (double *R*[3][3], double *d*[3], const dq_t *Q*)

Extracts the rotation matrix and translation vector assosciated to a dual quaternion.

Parameters

→ ***R*** Rotation matrix.
→ ***d*** Translation vector.
← ***Q*** Dual quaternion to extract R and d from.

4.2.2.3 void dq_op_f1g (dq_t *ABA*, const dq_t *A*, const dq_t *B*)

Clifford conjugation transformation of type f_{1g} (Alba Perez notation).

$$\begin{array}{rcl} f_{1G} : C(V,<,>) & \longrightarrow & C(V,<,>) \\ A : B & \longmapsto & ABA \end{array}$$

Parameters

→ ***ABA*** Result of the transformation.
← ***A*** Dual quaternion representing the transformation.
← ***B*** Dual quaternion being transformated.

See also

dq_op_f2g
dq_op_f3g
dq_op_f4g

4.2.2.4 void dq_op_f2g (dq_t *ABA*, const dq_t *A*, const dq_t *B*)

Clifford conjugation transformation of type f_{2g} (Alba Perez notation).

$$\begin{aligned} f_{2G} : C(V,<,>) &\longrightarrow C(V,<,>) \\ A : B &\longmapsto ABA^* \end{aligned}$$

This transformation is useful for lines.

Parameters

→ ***ABA*** Result of the transformation.

← ***A*** Dual quaternion representing the transformation.

← ***B*** Dual quaternion being transformated.

See also

dq_op_f1g
dq_op_f3g
dq_op_f4g

4.2.2.5 void dq_op_f3g (dq_t *ABA*, const dq_t *A*, const dq_t *B*)

Clifford conjugation transformation of type f_{3g} (Alba Perez notation).

$$\begin{aligned} f_{3G} : C(V,<,>) &\longrightarrow C(V,<,>) \\ A : B &\longmapsto AB(a_0 + a - \epsilon(a^0 + a_7)) \end{aligned}$$

Parameters

→ ***ABA*** Result of the transformation.

← ***A*** Dual quaternion representing the transformation.

← ***B*** Dual quaternion being transformated.

See also

dq_op_f1g
dq_op_f2g
dq_op_f4g

4.2.2.6 void dq_op_f4g (dq_t *ABA*, const dq_t *A*, const dq_t *B*)

Clifford conjugation transformation of type f_{4g} (Alba Perez notation).

$$\begin{aligned} f_{4G} : C(V,<,>) &\longrightarrow C(V,<,>) \\ A : B &\longmapsto AB(a_0 - a + \epsilon(a^0 - a_7)) \end{aligned}$$

This transformation is useful for points.

Parameters

→ ***ABA*** Result of the transformation.

← ***A*** Dual quaternion representing the transformation.

← ***B*** Dual quaternion being transformated.

See also

dq_op_f1g
dq_op_f2g
dq_op_f3g

4.2.2.7 void dq_op_mul (dq_t *PQ*, const dq_t *P*, const dq_t *Q*)

Multiplies to dual quaternions.

$$\widehat{PQ} = \widehat{P}\widehat{Q}$$

The multiplication table used is:

Q1*Q2	Q2.1	Q2.i	Q2.j	Q2.k	Q2.ei	Q2.ej	Q2.ek	Q2.e
Q1.1	1	i	j	k	ei	ej	ek	e
Q1.i	i	-1	k	-j	-e	ek	-ej	ei
Q1.j	j	-k	-1	i	-ek	-e	ei	ej
Q1.k	k	j	-i	-1	ej	-ei	-e	ek
Q1.ei	ei	-e	ek	-ej	0	0	0	0
Q1.ej	ej	-ek	-e	ei	0	0	0	0
Q1.ek	ek	ej	-ei	-e	0	0	0	0
Q1.e	e	ei	ej	ek	0	0	0	0

Parameters

→ ***PQ*** Result of the multiplication.

← ***P*** First dual quaternion to multiply.

← ***Q*** Second dual quaternion to multiply.

4.2.2.8 void dq_op_norm2 (double * *real*, double * *dual*, const dq_t *Q*)

Gets the square of the norm of a dual quaternion.

$$\|\widehat{Q}\|^2 = \widehat{Q}\widehat{Q}^*$$

The square of the norm is a dual number. If we denote Q as the vector part of the dual quaternion (all except q_0 and q_7):

$$\widehat{Q}\widehat{Q}^* = (\widehat{q_0} + Q)(\widehat{q_0} - Q) = \widehat{q_0}^2 + Q \cdot Q$$

If we denote the dual quaternion as $\widehat{Q} = \widehat{q} + \epsilon\widehat{q'}$ with $\widehat{q}$ being the pure real quaternion and $\widehat{q'}$ being the pure dual quaternion we can use the following notation to describe the product:

$$\widehat{Q}\widehat{Q}^* = \widehat{q}\widehat{q}^* + \epsilon(\widehat{q}\widehat{q'}^* + \widehat{q'}\widehat{q}^*) = (q_0q_0 + q_1q_1 + q_2q_2 + q_3q_3) + \epsilon 2(q_0q_7 + q_1q_4 + q_2q_5 + q_3q_6)$$

Note

A norm of 1 indicates that the dual quaternion is a unit dual quaternion.

Parameters

→ ***real*** The real part of the norm of the dual quaternion.

→ ***dual*** The dual port of the norm of the dual quaternion.

← ***Q*** Dual quaternion to get square of norm of.

See also

dq_cr_conj

4.2.2.9 void dq_op_sign (dq_t *P*, const dq_t *Q*)

Swaps the sign of all the elements in a dual quaternion.

Parameters

→ ***P*** Result of swapping all values of the elements.

← ***Q*** Dual quaternion to swap sign of all elements.

4.2.2.10 void dq_op_sub (dq_t *O*, const dq_t *P*, const dq_t *Q*)

Subtracts two dual quaternions.

$$\widehat{O} = \widehat{P} - \widehat{Q}$$

Parameters

→ ***O*** The result of the subtraction.

← ***P*** Dual quaternion to subtract from.

← ***Q*** Dual quaternion to subtract.

See also

dq_op_add

4.3 Dual Quaternion Check Functions

Assorted functions related to dual quaternions properties that can be checked.

Functions

- int dq_ch_unit (const dq_t Q)

  Checks to see if a dual quaternion is a unit quaternion.

- int dq_ch_point_plane (const dq_t P, const dq_t Q)

  Checks to see if a point Q is on the plane P.

- int dq_ch_cmp (const dq_t P, const dq_t Q)

  Compares two dual quaternions.

- int dq_ch_cmpV (const dq_t P, const dq_t Q, double precision)

  Compares two dual quaternions with variable precision.

4.3.1 Detailed Description

Assorted functions related to dual quaternions properties that can be checked.

4.3.2 Function Documentation

4.3.2.1 int dq_ch_cmp (const dq_t *P*, const dq_t *Q*)

Compares two dual quaternions.

Parameters

← ***P*** First dual quaternion to compare.

← ***Q*** Second dual quaternion to compare.

Returns

0 if they are equal.

See also

dq_ch_cmpV

4.3.2.2 int dq_ch_cmpV (const dq_t *P*, const dq_t *Q*, double *precision*)

Compares two dual quaternions with variable precision.

Parameters

← ***P*** First dual quaternion to compare.

← ***Q*** Second dual quaternion to compare.

← ***precision*** Precision to use when comparing members of each dual quaternion.

Returns

0 if they are equal.

See also

dq_ch_cmp

4.3.2.3 int dq_ch_point_plane (const dq_t *P*, const dq_t *Q*)

Checks to see if a point Q is on the plane P.

Parameters

← ***P*** Plane to check if point is on it.

← ***Q*** Point to check if is on plane P.

Returns

1 if point Q is on plane P.

4.3.2.4 int dq_ch_unit (const dq_t *Q*)

Checks to see if a dual quaternion is a unit quaternion.

Parameters

← ***Q*** Dual quaternion to check if is a unit quaternion.

Returns

1 if is a unit dual quaternion or 0 otherwise.

4.4 Dual Quaternion Miscellaneous Functions

Assorted functions related to dual quaternions that don't fit elsewhere.

Functions

- void dq_print (const dq_t Q)
  Prints a quaternion on a single line.
- void dq_print_vert (const dq_t Q)
  Prints a dual quaternion vertically.
- void dq_version (int *major, int *minor)
  Gets the version of the library during runtime.

4.4.1 Detailed Description

Assorted functions related to dual quaternions that don't fit elsewhere.

4.4.2 Function Documentation

4.4.2.1 void dq_print (const dq_t *Q*)

Prints a quaternion on a single line.

Parameters

← ***Q*** Dual quaternion to print.

See also

dq_printVert

4.4.2.2 void dq_print_vert (const dq_t *Q*)

Prints a dual quaternion vertically.

Parameters

← ***Q*** Dual quaternion to print.

See also

dq_print

4.4.2.3 void dq_version (int * *major*, int * *minor*)

Gets the version of the library during runtime.

This returns two values major and minor which can be used to form the version in the form of major.minor.

Parameters

→ ***major*** Major version of the library.

→ ***minor*** Minor version of the library.

4.5 Auxiliary Homogeneous Matrix Functions

Set of auxiliary functions to manipulate homogeneous matrix.

Functions

- void homo_cr_join (double H[3][4], double R[3][3], double d[3])

  Creates a homogeneous matrix by joining a 3x3 rotation matrix and a 3d translation vector.

- void homo_op_mul (double O[3][4], double A[3][4], double B[3][4])

  Multiplies two homogeneous matrix.

- void homo_op_split (double R[3][3], double d[3], double H[3][4])

  Splits a homogeneous matrix into a 3x3 rotation matrix and a 3d translation vector.

- void homo_op_mul_vec (double o[4], double H[3][4], const double v[4])

  Multiplies a homogeneous matrix by a 4d vector.

- int homo_ch_cmpV (double A[3][4], double B[3][4], double precision)

  Compares two homogeneous matrix with variable precision.

- int homo_ch_cmp (double A[3][4], double B[3][4])

  Compares two homogeneous matrix.

- void homo_print (double H[3][4])

  Prints a homogeneous matrix on screen.

4.5.1 Detailed Description

Set of auxiliary functions to manipulate homogeneous matrix. Due to the fact that the bottom row of each homogeneous matrix is always the same, we can simplify the amount of data needed to be stored.

The full homogeneous matrix is:

$$H = \begin{pmatrix} R & d \\ 0 & 1 \end{pmatrix}$$

Where,

$$R \in \mathbb{R}^{3x3} \tag{4.1}$$

$$d \in \mathbb{R}^{3} \tag{4.2}$$

Where R is a 3x3 rotation matrix and d is a 3d translation vector. The bottom row is always $(0, 0, 0, 1)$.

We simplify this to a 3x4 matrix by eliminating the bottom row and thus we end up with:

$$H = \begin{pmatrix} R & t \end{pmatrix} \in \mathbb{R}^{3x4}$$

4.5.2 Function Documentation

4.5.2.1 int homo_ch_cmp (double *A*[3][4], double *B*[3][4])

Compares two homogeneous matrix.

Parameters

← ***A*** First homogeneous matrix to compare.

← ***B*** Second homogeneous matrix to compare.

Returns

0 if they are the same, 1 otherwise.

See also

homo_ch_cmpV

4.5.2.2 int homo_ch_cmpV (double *A*[3][4], double *B*[3][4], double *precision*)

Compares two homogeneous matrix with variable precision.

Parameters

← ***A*** First homogeneous matrix to compare.

← ***B*** Second homogeneous matrix to compare.

← ***precision*** Precision to use when comparing.

Returns

0 if they are the same, 1 otherwise.

See also

homo_ch_cmp

4.5.2.3 void homo_cr_join (double *H*[3][4], double *R*[3][3], double *d*[3])

Creates a homogeneous matrix by joining a 3x3 rotation matrix and a 3d translation vector.

Parameters

→ ***H*** Homogeneous matrix formed by R and d.

← ***R*** 3x3 Rotation matrix.

← ***d*** 3d translation vector.

See also

homo_op_split

4.5.2.4 void homo_op_mul (double *O*[3][4], double *A*[3][4], double *B*[3][4])

Multiplies two homogeneous matrix.

$$O = A * B$$

Parameters

→ ***O*** Resulting homogeneous matrix of the multiplication.

← ***A*** First homogeneous matrix to operate on.

← ***B*** Second homogeneous matrix to operate on.

See also

homo_op_mul_vec

4.5.2.5 void homo_op_mul_vec (double *o*[4], double *H*[3][4], const double *v*[4])

Multiplies a homogeneous matrix by a 4d vector.

To multiply a normal 3d vector, add 1 as it's 4th component.

Parameters

→ ***o*** Resulting 4d vector of the multiplication.

← ***H*** Homogeneous matrix to multiply against.

← ***v*** 4d vector to multiply.

See also

homo_op_mul

4.5.2.6 void homo_op_split (double *R*[3][3], double *d*[3], double *H*[3][4])

Splits a homogeneous matrix into a 3x3 rotation matrix and a 3d translation vector.

Parameters

→ ***R*** 3x3 rotation matrix extracted from the homogeneous matrix.

→ ***d*** 3d translation vector extracted from the homogeneous matrix.

← ***H*** Homogeneous matrix to split.

4.5.2.7 void homo_print (double *H*[3][4])

Prints a homogeneous matrix on screen.

Parameters

← ***H*** Homogeneous matrix to print on screen.

4.6 Auxiliary 3x3 Matrix Functions

Set of auxiliary functions to manipulate 3x3 matrix.

Functions

- void mat3_eye (double M[3][3])

  Creates an identity 3x3 matrix.

- double mat3_det (double M[3][3])

  Gets the determinant of a 3x3 3x3 3x3 matrix.

- void mat3_add (double out[3][3], double A[3][3], double B[3][3])

  Adds two 3x3 matrix.

- void mat3_sub (double out[3][3], double A[3][3], double B[3][3])

  Subtracts two 3x3 matrix.

- void mat3_inv (double out[3][3], double in[3][3])

  Inverts a 3x3 matrix.

- void mat3_mul (double AB[3][3], double A[3][3], double B[3][3])

  Multiplies two 3x3 matrix.

- void mat3_mul_vec (double out[3], double M[3][3], const double v[3])

  Multiplies a 3x3 matrix by a vector.

- void mat3_solve (double x[3], double A[3][3], const double b[3])

  Solves a 3x3 equation system.

- int mat3_cmp (double A[3][3], double B[3][3])

  Compares two 3x3 matrix.

- int mat3_cmpV (double A[3][3], double B[3][3], double precision)

  Compares two 3x3 matrix with custom precision.

- void mat3_print (double M[3][3])

  Prints the value of a matrix on screen.

4.6.1 Detailed Description

Set of auxiliary functions to manipulate 3x3 matrix.

4.6.2 Function Documentation

4.6.2.1 void mat3_add (double *out*[3][3], double *A*[3][3], double *B*[3][3])

Adds two 3x3 matrix.

$$out = A + B$$

Parameters

→ ***out*** Result of the 3x3 matrix addition.

← ***A*** First matrix to operate on.

← ***B*** Second matrix to operate on.

4.6.2.2 int mat3_cmp (double *A*[3][3], double *B*[3][3])

Compares two 3x3 matrix.

Parameters

← ***A*** First matrix to compare.

← ***B*** Second matrix to compare.

Returns

0 if they are the same.

4.6.2.3 int mat3_cmpV (double *A*[3][3], double *B*[3][3], double *precision*)

Compares two 3x3 matrix with custom precision.

Parameters

← ***A*** First matrix to compare.

← ***B*** Second matrix to compare.

← ***precision*** Precision to use when comparing.

Returns

0 if they are the same.

4.6.2.4 double mat3_det (double *M*[3][3])

Gets the determinant of a 3x3 3x3 3x3 matrix.

$$o = \|M\|$$

Parameters

← ***M*** 3x3 Matrix to get the determinant of.

Returns

The determinant of the 3x3 matrix.

4.6.2.5 void mat3_eye (double *M*[3][3])

Creates an identity 3x3 matrix.

$$M = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Parameters

→ ***M*** An identiy matrix.

4.6.2.6 void mat3_inv (double *out*[3][3], double *in*[3][3])

Inverts a 3x3 matrix.

$$out = in^{-1}$$

Parameters

→ ***out*** Inversion of the matrix.

← ***in*** Matrix to invert.

4.6.2.7 void mat3_mul (double *AB*[3][3], double *A*[3][3], double *B*[3][3])

Multiplies two 3x3 matrix.

$$AB = A * B$$

Parameters

→ ***AB*** Result of the matrix multiplication.

← ***A*** First matrix to operate on.

← ***B*** Second matrix to operate on.

4.6.2.8 void mat3_mul_vec (double *out*[3], double *M*[3][3], const double *v*[3])

Multiplies a 3x3 matrix by a vector.

$$out = M * v$$

Parameters

→ ***out*** Result of the multiplication.

← ***M*** Matrix to operate on.

← ***v*** Vector to operate on.

4.6.2.9 void mat3_print (double *M*[3][3])

Prints the value of a matrix on screen.

Parameters

← ***M*** Matrix to print.

4.6.2.10 void mat3_solve (double *x*[3], double *A*[3][3], const double *b*[3])

Solves a 3x3 equation system.

Equation system should be in the form of:

$$Ax = b$$

Uses Cramer's rule to solve the system.

The system should be solveable ($det(A) \neq 0$).

Parameters

→ ***x*** Vector of variables to solve.

← ***A*** Matrix of coefficients.

← ***b*** Independent variable matrix.

4.6.2.11 void mat3_sub (double *out*[3][3], double *A*[3][3], double *B*[3][3])

Subtracts two 3x3 matrix.

$$out = A - B$$

Parameters

→ ***out*** Result of the 3x3 matrix subtraction.

← ***A*** First matrix to operate on.

← ***B*** Second matrix to operate on.

4.7 Auxiliary 3d Vector Functions

Set of auxiliary functions to manipulate 3d vectors.

Functions

- double vec3_dot (const double u[3], const double v[3])

  Does the dot product of two 3d vectors.

- void vec3_cross (double o[3], const double u[3], const double v[3])

  Does the cross product of two 3d vectors.

- void vec3_add (double o[3], const double u[3], const double v[3])

  Adds two 3d vectors.

- void vec3_sub (double o[3], const double u[3], const double v[3])

  Subtracts two 3d vectors.

- void vec3_sign (double v[3])

  Changes the sign of a vector.

- double vec3_norm (const double v[3])

  Gets the norm of a 3d vector.

- void vec3_normalize (double v[3])

  Normalizes a 3d vector.

- double vec3_distance (const double u[3], const double v[3])

  Gets the distance between two vectors.

- int vec3_cmp (const double u[3], const double v[3])

  Compares two 3d vectors.

- int vec3_cmpV (const double u[3], const double v[3], double precision)

  Compares two 3d vectors with variable precision.

- void vec3_print (const double v[3])

  Prints a 3d vector on screen.

4.7.1 Detailed Description

Set of auxiliary functions to manipulate 3d vectors.

4.7.2 Function Documentation

4.7.2.1 void vec3_add (double *o*[3], const double *u*[3], const double *v*[3])

Adds two 3d vectors.

$$o = u + v$$

Parameters

→ ***o*** Result of the addition.

← ***u*** First 3d vector to operate on.

← ***v*** Second 3d vector to operate on.

4.7.2.2 int vec3_cmp (const double *u*[3], const double *v*[3])

Compares two 3d vectors.

Parameters

← ***u*** First 3d vector to compare.

← ***v*** Second 3d vector to compare.

Returns

0 if they are the same.

4.7.2.3 int vec3_cmpV (const double *u*[3], const double *v*[3], double *precision*)

Compares two 3d vectors with variable precision.

Parameters

← ***u*** First 3d vector to compare.

← ***v*** Second 3d vector to compare.

← ***precision*** Precision to use.

Returns

0 if they are the same.

4.7.2.4 void vec3_cross (double *o*[3], const double *u*[3], const double *v*[3])

Does the cross product of two 3d vectors.

$$o = u \times v$$

Parameters

→ ***o*** The cross product of u and v.

← ***u*** First 3d vector to operate on.

← ***v*** Second 3d vector to operate on.

4.7.2.5 double vec3_distance (const double *u*[3], const double *v*[3])

Gets the distance between two vectors.

$$out = \|u - v\|$$

Parameters

← ***u*** Base vector.

← ***v*** Vector to get distance from u.

Returns

The distance between the two vectors.

4.7.2.6 double vec3_dot (const double *u*[3], const double *v*[3])

Does the dot product of two 3d vectors.

$$o = u \cdot v$$

Parameters

← ***u*** First 3d vector to operate on.

← ***v*** Second 3d vector to operate on.

Returns

The dot product of u.v.

4.7.2.7 double vec3_norm (const double *v*[3])

Gets the norm of a 3d vector.

$$o = \|v\|$$

Parameters

← ***v*** Vector to get norm of.

Returns

The norm of the 3d vector.

4.7.2.8 void vec3_normalize (double *v*[3])

Normalizes a 3d vector.

$$v_{out} = \frac{v}{\|v\|}$$

Parameters

v Vector to normalize.

4.7.2.9 void vec3_print (const double *v*[3])

Prints a 3d vector on screen.

Parameters

← ***v*** Vector to print.

4.7.2.10 void vec3_sign (double *v*[3])

Changes the sign of a vector.

$$o = -v$$

Parameters

v Vector to change sign of.

4.7.2.11 void vec3_sub (double *o*[3], const double *u*[3], const double *v*[3])

Subtracts two 3d vectors.

$$o = u - v$$

Parameters

→ ***o*** Result of the subtraction.

← ***u*** First 3d vector to operate on.

← ***v*** Second 3d vector to operate on.

Chapter 5

File Documentation

5.1 dq.h File Reference

The include for the libdq dual quaternion library.

Defines

- #define DQ_VERSION_MAJOR 2
- #define DQ_VERSION_MINOR 0
- #define DQ_PRECISION 1e-10

Typedefs

- typedef double dq_t [8]

  A representation of a dual quaternion.

Functions

- void dq_cr_rotation (dq_t O, double theta, const double s[3], const double c[3])

  Creates a pure rotation dual quaternion.

- void dq_cr_rotation_plucker (dq_t O, double theta, const double s[3], const double s0[3])

  Creates a pure rotation dual quaternion using plucker coordinates.

- void dq_cr_rotation_matrix (dq_t O, double R[3][3])

  Creates a pure rotation dual quaternion from a rotation matrix.

- void dq_cr_translation (dq_t O, double t, const double s[3])

Creates a pure translation dual quaternion.

- void dq_cr_translation_vector (dq_t O, const double t[3])
  Creates a pure translation dual quaternion from a traslation vector.
- void dq_cr_point (dq_t O, const double pos[3])
  Creates a dual quaternion representing a point.
- void dq_cr_line (dq_t O, const double s[3], const double c[3])
  Creates a dual quaternion representing a line.
- void dq_cr_line_plucker (dq_t O, const double s[3], const double s0[3])
  Creates a dual quaternion representing a line from plucker coordinates.
- void dq_cr_plane (dq_t O, const double n[3], const double d)
  Creates a unit dual quaternion representing a plane.
- void dq_cr_homo (dq_t O, double R[3][3], const double d[3])
  Creates a dual quaternion from a homogeneous transformation matrix.
- void dq_cr_copy (dq_t O, const dq_t Q)
  Copies a dual quaternion.
- void dq_cr_conj (dq_t O, const dq_t Q)
  Conjugates a dual quaternion.
- void dq_cr_inv (dq_t O, const dq_t Q)
  Inverts a dual quaternion.
- void dq_op_norm2 (double *real, double *dual, const dq_t Q)
  Gets the square of the norm of a dual quaternion.
- void dq_op_add (dq_t O, const dq_t P, const dq_t Q)
  Adds two dual quaternions.
- void dq_op_sub (dq_t O, const dq_t P, const dq_t Q)
  Subtracts two dual quaternions.
- void dq_op_mul (dq_t PQ, const dq_t P, const dq_t Q)
  Multiplies to dual quaternions.
- void dq_op_sign (dq_t P, const dq_t Q)
  Swaps the sign of all the elements in a dual quaternion.
- void dq_op_f1g (dq_t ABA, const dq_t A, const dq_t B)
  Clifford conjugation transformation of type f_{1g} (Alba Perez notation).

- void dq_op_f2g (dq_t ABA, const dq_t A, const dq_t B)
  Clifford conjugation transformation of type f_{2g} (Alba Perez notation).
- void dq_op_f3g (dq_t ABA, const dq_t A, const dq_t B)
  Clifford conjugation transformation of type f_{3g} (Alba Perez notation).
- void dq_op_f4g (dq_t ABA, const dq_t A, const dq_t B)
  Clifford conjugation transformation of type f_{4g} (Alba Perez notation).
- void dq_op_extract (double R[3][3], double d[3], const dq_t Q)
  Extracts the rotation matrix and translation vector assosciated to a dual quaternion.
- int dq_ch_unit (const dq_t Q)
  Checks to see if a dual quaternion is a unit quaternion.
- int dq_ch_point_plane (const dq_t P, const dq_t Q)
  Checks to see if a point Q is on the plane P.
- int dq_ch_cmp (const dq_t P, const dq_t Q)
  Compares two dual quaternions.
- int dq_ch_cmpV (const dq_t P, const dq_t Q, double precision)
  Compares two dual quaternions with variable precision.
- void dq_print (const dq_t Q)
  Prints a quaternion on a single line.
- void dq_print_vert (const dq_t Q)
  Prints a dual quaternion vertically.
- void dq_version (int *major, int *minor)
  Gets the version of the library during runtime.

5.1.1 Detailed Description

The include for the libdq dual quaternion library.

5.1.2 Define Documentation

5.1.2.1 #define DQ_PRECISION 1e-10

Precision to use when comparing doubles.

5.1.2.2 #define DQ_VERSION_MAJOR 2

Major version of the libdq library.

5.1.2.3 #define DQ_VERSION_MINOR 0

Minor version of the libdq library.

5.1.3 Typedef Documentation

5.1.3.1 typedef double dq_t[8]

A representation of a dual quaternion.

Dual quaternions are elements of the Clifford even subalgebra $C^+_{0,3,1}$. There are many notations for dual quaternions. This library uses the basis used by McArthy which is the same as Selig with minor rearrangements,

$$\{1, e_{23}, e_{31}, e_{12}, e_{41}, e_{42}, e_{43}, e_{1234}\} = \{1, i, j, k, i\epsilon, j\epsilon, k\epsilon, \epsilon\}$$

This allows us to write a dual quaternion as,

$$\widehat{Q} = (q_0 + q_1 i + q_2 j + q_3 k) + \epsilon(q_7 + q_4 i + q_5 j + q_6 k) = \widehat{q} + \epsilon\widehat{q}^0$$

Using vertical notation we would have the following,

$$\widehat{Q} = \begin{Bmatrix} e_{23} \\ e_{31} \\ e_{12} \\ 1 \end{Bmatrix} + \begin{Bmatrix} e_{41} \\ e_{42} \\ e_{43} \\ e_{1234} \end{Bmatrix} = \begin{Bmatrix} i \\ j \\ k \\ 1 \end{Bmatrix} + \epsilon \begin{Bmatrix} i \\ j \\ k \\ 1 \end{Bmatrix}$$

In order for the dual quaternion to be able to represent spatial displacements it must be a unit dual quaternion and thus comply with the following restrictions,

$$\widehat{q}\widehat{q}^0 = 1$$
$$\widehat{q} \cdot \widehat{q}^0 = 0$$

It is important to note that unit dual quaternions double cover the special euclidean group $SE(3)$. This means that $\widehat{Q}$ and $-\widehat{Q}$ represent the same spatial displacement.

5.2 dq_homo.h File Reference

File containing functions related to 4 by 4 homogeneous matrix.

Functions

- void homo_cr_join (double H[3][4], double R[3][3], double d[3])

  Creates a homogeneous matrix by joining a 3x3 rotation matrix and a 3d translation vector.

- void homo_op_mul (double O[3][4], double A[3][4], double B[3][4])

  Multiplies two homogeneous matrix.

- void homo_op_split (double R[3][3], double d[3], double H[3][4])

  Splits a homogeneous matrix into a 3x3 rotation matrix and a 3d translation vector.

- void homo_op_mul_vec (double o[4], double H[3][4], const double v[4])

  Multiplies a homogeneous matrix by a 4d vector.

- int homo_ch_cmpV (double A[3][4], double B[3][4], double precision)

  Compares two homogeneous matrix with variable precision.

- int homo_ch_cmp (double A[3][4], double B[3][4])

  Compares two homogeneous matrix.

- void homo_print (double H[3][4])

  Prints a homogeneous matrix on screen.

5.2.1 Detailed Description

File containing functions related to 4 by 4 homogeneous matrix.

5.3 dq_mat3.h File Reference

File containing functions related to 3 by 3 matrix.

Functions

- void mat3_eye (double M[3][3])
  Creates an identity 3x3 matrix.
- double mat3_det (double M[3][3])
  Gets the determinant of a 3x3 3x3 3x3 matrix.
- void mat3_add (double out[3][3], double A[3][3], double B[3][3])
  Adds two 3x3 matrix.
- void mat3_sub (double out[3][3], double A[3][3], double B[3][3])
  Subtracts two 3x3 matrix.
- void mat3_inv (double out[3][3], double in[3][3])
  Inverts a 3x3 matrix.
- void mat3_mul (double AB[3][3], double A[3][3], double B[3][3])
  Multiplies two 3x3 matrix.
- void mat3_mul_vec (double out[3], double M[3][3], const double v[3])
  Multiplies a 3x3 matrix by a vector.
- void mat3_solve (double x[3], double A[3][3], const double b[3])
  Solves a 3x3 equation system.
- int mat3_cmp (double A[3][3], double B[3][3])
  Compares two 3x3 matrix.
- int mat3_cmpV (double A[3][3], double B[3][3], double precision)
  Compares two 3x3 matrix with custom precision.
- void mat3_print (double M[3][3])
  Prints the value of a matrix on screen.

5.3.1 Detailed Description

File containing functions related to 3 by 3 matrix.

5.4 dq_vec3.h File Reference

File containing functions related to 3d vectors.

Functions

- double vec3_dot (const double u[3], const double v[3])
  Does the dot product of two 3d vectors.
- void vec3_cross (double o[3], const double u[3], const double v[3])
  Does the cross product of two 3d vectors.
- void vec3_add (double o[3], const double u[3], const double v[3])
  Adds two 3d vectors.
- void vec3_sub (double o[3], const double u[3], const double v[3])
  Subtracts two 3d vectors.
- void vec3_sign (double v[3])
  Changes the sign of a vector.
- double vec3_norm (const double v[3])
  Gets the norm of a 3d vector.
- void vec3_normalize (double v[3])
  Normalizes a 3d vector.
- double vec3_distance (const double u[3], const double v[3])
  Gets the distance between two vectors.
- int vec3_cmp (const double u[3], const double v[3])
  Compares two 3d vectors.
- int vec3_cmpV (const double u[3], const double v[3], double precision)
  Compares two 3d vectors with variable precision.
- void vec3_print (const double v[3])
  Prints a 3d vector on screen.

5.4.1 Detailed Description

File containing functions related to 3d vectors.

Index