Screws Library

v1.0

Generated by Doxygen 1.8.8

Sat May 9 2015 00:09:22

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

Class Documentation

2.1 Adjoint Class Reference

Implements a 6x6 adjoint matrix with basic functionality.

`#include <adjoint.hpp>`

Public Member Functions

- Adjoint ()
  Create a zero 6x6 matrix for the Adjoint.
- Adjoint (const Rotation &R, const Translation &T)
  Create an adjoint matrix from a Rotation and a Translation.
- Adjoint (const HomogeneousTransform &H)
  Create an adjoint for the given homogeneous transformation matrix.
- ∼Adjoint ()
  Default destructor.
- double at (const int &i, const int &j)
  Accessor function for ease of use.
- Adjoint inv () const
  Calculates the inverse of the adjoint in a computationally efficient way.
- arma::colvec::fixed< 6 > operator∗ (const arma::vec::fixed< 6 > &ksi)
  Handle multiplication with a twist coordinates vector (6x1).
- arma::mat operator∗ (const arma::mat &jacobian)
  Handle multiplication with a jacobian.
- bool **operator==** (const Adjoint &A)
- void **print** () const

2.1.1 Detailed Description

Implements a 6x6 adjoint matrix with basic functionality.

Note

Author: Christos Bergeles

Date

2013-07-02

2.1.2 Constructor & Destructor Documentation

2.1.2.1 Adjoint::Adjoint (const Rotation & *R*, const Translation & *T*) [explicit]

Create an adjoint matrix from a Rotation and a Translation.

Parameters

R	the rotational component.
T	the translational component.

2.1.2.2 Adjoint::Adjoint (const HomogeneousTransform & *H*) [explicit]

Create an adjoint for the given homogeneous transformation matrix.

Parameters

H	the homogeneous transformation matrix.

2.1.3 Member Function Documentation

2.1.3.1 double Adjoint::at (const int & *i*, const int & *j*)

Accessor function for ease of use.

Parameters

i	the i-th row of the matrix.
j	the j-th row of the matrix.

Note

No boundary check is performed.

2.1.3.2 arma::vec::fixed< 6 > Adjoint::operator* (const arma::vec::fixed< 6 > & *ksi*)

Handle multiplication with a twist coordinates vector (6x1).

Parameters

ksi	the 6x1 coordinates vector.

Note

No dimension check for ksi is performed.

2.1.3.3 arma::mat Adjoint::operator* (const arma::mat & *jacobian*)

Handle multiplication with a jacobian.

Parameters

jacobian	the 6xN jacobian matrix.

Note

No dimension check for jacobian is performed.

The documentation for this class was generated from the following files:

- /Users/cbergeles/Dropbox/prj/mevislab/CustomLibraries/General/Sources/Shared/Screws/adjoint.hpp
- /Users/cbergeles/Dropbox/prj/mevislab/CustomLibraries/General/Sources/Shared/Screws/adjoint.cpp

2.2 screws::HomogeneousTransform< NumType > Class Template Reference

Implements a 4x4 homogeneous transformation matrix along with basic operators.

`#include <homogeneousTransform.hpp>`

Public Member Functions

- HomogeneousTransform ()
  Create a default homogeneous transformation unit matrix.
- HomogeneousTransform (const Rotation< NumType > &R, const Translation< NumType > &T)
  Create a homogeneous transformation matrix from a rotation and a translation.
- Translation< NumType > translation () const
  Return the 3x1 translation vector.
- void setTranslation (const Translation< NumType > &T)
  Change the translation part of the homogeneous transformation matrix.
- Rotation< NumType > rotation () const
  Return the rotational part of the homogeneous transformation matrix.
- void setRotation (const Rotation< NumType > &R)
  Set the rotational part of the homegeneous transformation matrix.
- Twist< NumType > log () const
  Calculate the twist (log) of the homogeneous transformation matrix.
- Twist< NumType > twist () const
  Calculate the twist (log) of the homogeneous transformation matrix.
- HomogeneousTransform< NumType > inv () const
  Perform fast inversion of the homogeneous transformation matrix.
- HomogeneousTransform< NumType > operator* (const HomogeneousTransform< NumType > &H)
  Matrix multiplication.
- const HomogeneousTransform
  < NumType > & operator*= (const HomogeneousTransform< NumType > &H)
  In-place matrix multiplication.
- Translation< NumType > operator* (const Translation< NumType > &T)
  Multiplication with a translation.
- bool operator== (const HomogeneousTransform &H)
  Element-by-element exact equality operator.
- bool operator!= (const HomogeneousTransform &H)
  Element-by-element inequality operator.
- const NumType & operator() (const unsigned int &i, const unsigned int &j) const
  Return the requested element (read).
- bool approxEq (const HomogeneousTransform< NumType > &H, double eps=FLT_EPSILON)
  Approximal equality operator, within a given epsilon or system precision.
- bool isValid () const
  Perform verification that matrix is indeed in approriate format by checking the rotational component for validity.

Friends

- template<class NumTypeTrans >
  class **Translation**
- template<class NumTypeRot >
  class **Rotation**

2.2.1 Detailed Description

template<class NumType>class screws::HomogeneousTransform< NumType >

Implements a 4x4 homogeneous transformation matrix along with basic operators.

Note

Author: Christos Bergeles

Date

5th May 2015

2.2.2 Constructor & Destructor Documentation

2.2.2.1 template<class NumType> screws::HomogeneousTransform< NumType >::HomogeneousTransform (const Rotation< NumType > & *R*, const Translation< NumType > & *T*) [inline]

Create a homogeneous transformation matrix from a rotation and a translation.

Parameters

R	the Rotation matrix.
T	the Translation vector.

2.2.3 Member Function Documentation

2.2.3.1 template<class NumType> bool screws::HomogeneousTransform< NumType >::approxEq (const HomogeneousTransform< NumType > & *H*, double *eps* = FLT_EPSILON) [inline]

Approximal equality operator, within a given epsilon or system precision.

Parameters

T	compared translation.
eps	desired precision [default: machine precision].

2.2.3.2 template<class NumType> HomogeneousTransform<NumType> screws::HomogeneousTransform< NumType >::inv ( ) const [inline]

Perform fast inversion of the homogeneous transformation matrix.

Returns

the inverted homogeneous transformation matrix.

2.2.3.3 template<class NumType> Twist<NumType> screws::HomogeneousTransform< NumType >::log ( ) const [inline]

Calculate the twist (log) of the homogeneous transformation matrix.

Returns

the 4x4 twist skew symmetric matrix.

2.2.3.4 template<class NumType> bool screws::HomogeneousTransform< NumType >::operator!= (const HomogeneousTransform< NumType > & *H*) [inline]

Element-by-element inequality operator.

Returns

true if any of the elements are not exactly the same.

2.2.3.5 template<class NumType> const NumType& screws::HomogeneousTransform< NumType >::operator() (const unsigned int & *i*, const unsigned int & *j*) const [inline]

Return the requested element (read).

Parameters

i	the index of the row.
j	the index of the column.

Note

No boundary check is performed.

2.2.3.6 template<class NumType> HomogeneousTransform<NumType> screws::HomogeneousTransform< NumType >::operator* (const HomogeneousTransform< NumType > & *H*) [inline]

Matrix multiplication.

Returns

the resulting homogeneous transform.

2.2.3.7 template<class NumType> Translation<NumType> screws::HomogeneousTransform< NumType >::operator* (const Translation< NumType > & *T*) [inline]

Multiplication with a translation.

Returns

the resulting translation.

2.2.3.8 template<class NumType> const HomogeneousTransform<NumType>& screws::HomogeneousTransform< NumType >::operator*= (const HomogeneousTransform< NumType > & *H*) [inline]

In-place matrix multiplication.

Returns

the resulting homogeneous transform.

2.2.3.9 template<class NumType> bool screws::HomogeneousTransform< NumType >::operator== (const HomogeneousTransform< NumType > & *H*) [inline]

Element-by-element exact equality operator.

Returns

true if all elements are exactly the same.

2.2.3.10 template<class NumType> Rotation<NumType> screws::HomogeneousTransform< NumType >::rotation ( ) const [inline]

Return the rotational part of the homogeneous transformation matrix.

Returns

the 3x3 rotational component.

2.2.3.11 template<class NumType> void screws::HomogeneousTransform< NumType >::setRotation (const Rotation< NumType > & *R*) [inline]

Set the rotational part of the homegeneous transformation matrix.

Parameters

R	the new rotational component.

2.2.3.12 template<class NumType> void screws::HomogeneousTransform< NumType >::setTranslation (const Translation< NumType > & *T*) [inline]

Change the translation part of the homogeneous transformation matrix.

Parameters

T	the new translation component.

2.2.3.13 template<class NumType> Translation<NumType> screws::HomogeneousTransform< NumType >::translation ( ) const [inline]

Return the 3x1 translation vector.

Returns

the 3x1 translation vector.

2.2.3.14 template<class NumType> Twist<NumType> screws::HomogeneousTransform< NumType >::twist ( ) const [inline]

Calculate the twist (log) of the homogeneous transformation matrix.

Returns

the 4x4 twist skew symmetric matrix.

The documentation for this class was generated from the following file:

- /Users/cbergeles/Dropbox/prj/mevislab/CustomLibraries/General/Sources/Shared/Screws/homogeneous↩ Transform.hpp

2.3 screws::Rotation< NumType > Class Template Reference

Implements a 3x3 rotation along with basic operators.

`#include <rotation.hpp>`

Public Member Functions

- Rotation ()

  Construct a 3x3 identity rotation matrix.
- Rotation (const Translation< NumType > &c0, const Translation< NumType > &c1, const Translation< NumType > &c2)

  Construct the 3x3 matrix from vectors.
- Rotation (const char &axis, const NumType &theta)

  Construct a 3x3 rotation given common axes.
- Rotation (const Vector3< NumType > &axisVector, const NumType &theta)

  Construct a 3x3 rotation given axis and angle.
- Rotation (const Vector3< NumType > &zAxis)

  Create a rotation instance given a Translation that acts as the z axis.
- ~Rotation ()

  Default destructor.
- Vector3< NumType > axis () const

  Return the axis of rotation.
- NumType angle () const

  Return the magnitude of the rotation. It calculates the axis internally.
- Vector3< NumType > rpy () const

  Return the roll-pitch-yaw angles.
- Rotation< NumType > inv () const

  Invert by taking the transpose.
- Skew< NumType > log () const

  Return the skew (log) symmetric matrix corresponding to this rotation.
- Skew< NumType > skew () const

  Return the skew (log) symmetric matrix corresponding to this rotation.
- const NumType & operator() (const unsigned int &i, const unsigned int &j) const

  Return the requested element (read).
- Rotation< NumType > operator* (const Rotation< NumType > &R) const

  Multiplication operator.
- const Rotation< NumType > & operator*= (const Rotation< NumType > &R)

In-place multiplication operator.

- bool operator== (const Rotation< NumType > &R) const

  Equality operator.
- bool operator!= (const Rotation< NumType > &R) const

  Inequality operator.
- bool approxEq (const Rotation< NumType > &R, double eps=FLT_EPSILON) const

  Approximal equality operator, within a given epsilon or system precision.
- Vector3< NumType > operator* (const Vector3< NumType > &T) const

  Multiplication with a 3x1 Vector acting as a point.
- bool isValid () const

  Perform verification that matrix is indeed in approriate format, by checking if the determinant is zero, and if the columns are othogonal to each other and of magnitude one.

Protected Member Functions

- void **construct** (const NumType &ux, const NumType &uy, const NumType &uz, const NumType &theta)
- void **setData** (const NumType &r00, const NumType &r01, const NumType &r02, const NumType &r10, const NumType &r11, const NumType &r12, const NumType &r20, const NumType &r21, const NumType &r22)
- void **resetData** ()
- void **calculateAxisAndAngle** (Vector3< NumType > &axisVec, NumType &angleVal) const

Protected Attributes

- Eigen::Matrix< NumType, 3, 3 > **_data**

Friends

- template<class NumTypeTrans >
  class **Translation**
- template<class NumTypeSkew >
  class **Skew**
- template<class NumTypeTwist >
  class **Twist**

2.3.1 Detailed Description

template<class NumType>class screws::Rotation< NumType >

Implements a 3x3 rotation along with basic operators.

Note

Author: Christos Bergeles

Date

27th April 2015

2.3.2 Constructor & Destructor Documentation

2.3.2.1 template<class NumType> screws::Rotation< NumType >::Rotation (const Translation< NumType > & *c0,* const Translation< NumType > & *c1,* const Translation< NumType > & *c2*) [inline], [explicit]

Construct the 3x3 matrix from vectors.

Parameters

c0	the vector that will become the first column of the rotation matrix.
c1	the vector that will become the second column of the rotation matrix.
c2	the vector that will become the third column of the rotation matrix.

Note

The constructor checks if the resulting matrix has orthonormal columns.

2.3.2.2 template<class NumType> screws::Rotation< NumType >::Rotation (const char & *axis*, const NumType & *theta*) [inline], [explicit]

Construct a 3x3 rotation given common axes.

Parameters

axis	"x", "y", or "z" for common rotations.
theta	the rotation angle.

2.3.2.3 template<class NumType> screws::Rotation< NumType >::Rotation (const Vector3< NumType > & *axisVector*, const NumType & *theta*) [inline], [explicit]

Construct a 3x3 rotation given axis and angle.

Parameters

axisVector	the axis of the rotation.
theta	the rotation angle.

Note

If axisVector is not normal, it is normalised.

2.3.2.4 template<class NumType> screws::Rotation< NumType >::Rotation (const Vector3< NumType > & *zAxis*) [inline], [explicit]

Create a rotation instance given a Translation that acts as the z axis.

Note

x and y are arbitrarily defined to generate a right-handed system.

Parameters

v	the 3x1 orientation vector.

2.3.3 Member Function Documentation

2.3.3.1 template<class NumType> NumType screws::Rotation< NumType >::angle ( ) const [inline]

Return the magnitude of the rotation. It calculates the axis internally.

Returns

the magnitude of rotation around the axis.

2.3.3.2 template<class NumType> bool screws::Rotation< NumType >::approxEq (const Rotation< NumType > & *R*, double *eps* = FLT_EPSILON) const [inline]

Approximal equality operator, within a given epsilon or system precision.

Parameters

R	compared rotation.
eps	desired precision [default: machine precision].

2.3.3.3 template<class NumType> Vector3<NumType> screws::Rotation< NumType >::axis ( ) const [inline]

Return the axis of rotation.

Returns

the axis of rotation as a 3x1 vector.

2.3.3.4 template<class NumType> Rotation<NumType> screws::Rotation< NumType >::inv ( ) const [inline]

Invert by taking the transpose.

Returns

the inverted rotation matrix.

2.3.3.5 template<class NumType> Skew<NumType> screws::Rotation< NumType >::log ( ) const [inline]

Return the skew (log) symmetric matrix corresponding to this rotation.

Returns

: the skew symmetric matrix.

2.3.3.6 template<class NumType> bool screws::Rotation< NumType >::operator!= (const Rotation< NumType > & *R*) const [inline]

Inequality operator.

Returns

true if one of the element-by-element comparisons return true. Otherwise, false.

2.3.3.7 template<class NumType> const NumType& screws::Rotation< NumType >::operator() (const unsigned int & *i*, const unsigned int & *j*) const [inline]

Return the requested element (read).

Parameters

i	the index of the row.
j	the index of the column.

Note

No boundary check is performed.

2.3.3.8 template<class NumType> bool screws::Rotation< NumType >::operator== (const Rotation< NumType > & *R*) const [inline]

Equality operator.

Returns

true if all element-by-element comparisons return true. Otherwise, false.

2.3.3.9 template<class NumType> Vector3<NumType> screws::Rotation< NumType >::rpy ( ) const [inline]

Return the roll-pitch-yaw angles.

Returns

a 3x1 vector of the rpy angles.

Note

Angles are returned in the [0, 2] domain.

2.3.3.10 template<class NumType> Skew<NumType> screws::Rotation< NumType >::skew ( ) const [inline]

Return the skew (log) symmetric matrix corresponding to this rotation.

Returns

: the skew symmetric matrix.

The documentation for this class was generated from the following files:

- /Users/cbergeles/Dropbox/prj/mevislab/CustomLibraries/General/Sources/Shared/Screws/homogeneous↩ Transform.hpp
- /Users/cbergeles/Dropbox/prj/mevislab/CustomLibraries/General/Sources/Shared/Screws/rotation.hpp

2.4 screws::ScrewException Class Reference

This class handles different exception types for the screws library.

`#include <screwException.hpp>`

Public Member Functions

- ScrewException (std::string msg, std::string filename, std::string function, int line)

  Exception class for libScrews.
- virtual const char * what () const

2.4.1 Detailed Description

This class handles different exception types for the screws library.

Date

27th April 2013

2.4.2 Constructor & Destructor Documentation

2.4.2.1 ScrewException::ScrewException (std::string *msg,* std::string *filename,* std::string *function,* int *line*)

Exception class for libScrews.

Parameters

msg	an std::string with the message to be displayed.
filename	an std::string containing the filename where the error originated.
function	an std::string containing the name of the function where the error originated from.
line	an integer specifying the line of code in the file.

2.4.3 Member Function Documentation

2.4.3.1 const char * ScrewException::what ( ) const [virtual]

Returns

the reported error message.

The documentation for this class was generated from the following files:

- /Users/cbergeles/Dropbox/prj/mevislab/CustomLibraries/General/Sources/Shared/Screws/screwException.hpp
- /Users/cbergeles/Dropbox/prj/mevislab/CustomLibraries/General/Sources/Shared/Screws/screwException.cpp

2.5 screws::Skew< NumType > Class Template Reference

Implenets a class for the 3x3 skew symmetric rotation generators and their operations.

`#include <skew.hpp>`

Public Member Functions

- Skew (const Translation< NumType > &v)

  Create a skew symmetric matrix out of a 3x1 vector.
- Skew (const Rotation< NumType > &R)

  Create a skew symmetric matrix by taking the log of a rotation matrix.
- Skew ()

  Default constructor with zeros. This correponds to the identity rotation.
- ~Skew ()

  Default destructor.
- Skew< NumType > transpose () const

  Transposes the skew.
- Rotation< NumType > exp (const NumType &theta=(NumType) 1) const

  Calculate the exponential of the skew symmetric matrix. This corresponds to a rotation.
- Skew< NumType > normalised () const

  Normalise and remove magnitude from skew symmetric matrix. Only rotation axis information remains.
- NumType norm () const

  Calculate and return the norm of the skew.
- NumType angle () const

  Extract the rotation magnitude without normalising.
- Vector3< NumType > axis () const

  Extract the axis of rotation.
- const NumType & operator() (const unsigned int &i, const unsigned int &j) const

  Return the requested element (read).

- Skew< NumType > operator+ (const Skew< NumType > &S) const

  Element-by-element addition of skew matrices.
- const Skew< NumType > & operator+= (const Skew< NumType > &S)

  Element-by-element in-place addition of skew matrices.
- Skew< NumType > operator* (const NumType &value) const

  Element-by-element multiplication with a value.
- const Skew< NumType > & operator*= (const NumType &value)

  Element-by-element in-place multiplication with a value.
- bool operator== (const Skew< NumType > &S) const

  Element-by-element exact equality operator.
- bool operator!= (const Skew< NumType > &S) const

  Inequality operator.
- bool approxEq (const Skew< NumType > &S, double eps=FLT_EPSILON) const

  Approximal equality operator, within a given epsilon or system precision.
- Vector3< NumType > coordinates () const

  Generates a 3x1 vector given a skew symmetric matrix.
- bool isValid () const

  Perform verification that the matrix corresponds to a rotation.

Protected Member Functions

- void **resetData** ()

Protected Attributes

- Eigen::Matrix< NumType, 3, 3 > **_data**

Friends

- template<class NumTypeRot >
  class **Rotation**
- template<class NumTypeTrans >
  class **Translation**
- template<class NumTypeTwist >
  class **Twist**

2.5.1 Detailed Description

template<class NumType>class screws::Skew< NumType >

Implenets a class for the 3x3 skew symmetric rotation generators and their operations.

Note

Author: Christos Bergeles

Date

5th May 2015

2.5.2 Constructor & Destructor Documentation

2.5.2.1 template<class NumType> screws::Skew< NumType >::Skew (const Translation< NumType > & *v*) [inline], [explicit]

Create a skew symmetric matrix out of a 3x1 vector.

Parameters

v	the 3x1 vector, which also contains the angle information.

2.5.2.2 template<class NumType> screws::Skew< NumType >::Skew (const Rotation< NumType > & *R*) [inline], [explicit]

Create a skew symmetric matrix by taking the log of a rotation matrix.

Parameters

R	the 3x3 rotation matrix.
theta	the extracted rotation angle.

2.5.3 Member Function Documentation

2.5.3.1 template<class NumType> NumType screws::Skew< NumType >::angle ( ) const [inline]

Extract the rotation magnitude without normalising.

Returns

the magnitude of the rotation.

2.5.3.2 template<class NumType> bool screws::Skew< NumType >::approxEq (const Skew< NumType > & *S*, double *eps* = FLT_EPSILON) const [inline]

Approximal equality operator, within a given epsilon or system precision.

Parameters

S	compared skew matrix.
eps	desired precision [default: machine precision].

2.5.3.3 template<class NumType> Vector3<NumType> screws::Skew< NumType >::axis ( ) const [inline]

Extract the axis of rotation.

Returns

the axis of rotation as a 3x1 vector.

2.5.3.4 template<class NumType> Vector3<NumType> screws::Skew< NumType >::coordinates ( ) const [inline]

Generates a 3x1 vector given a skew symmetric matrix.

Returns

the 3x1 vector.

2.5.3.5 template<class NumType> Rotation<NumType> screws::Skew< NumType >::exp (const NumType & *theta* = (NumType)1) const [inline]

Calculate the exponential of the skew symmetric matrix. This corresponds to a rotation.

Parameters

theta	the rotation magnitude.

2.5.3.6 template<class NumType> bool screws::Skew< NumType >::isValid ( ) const [inline]

Perform verification that the matrix corresponds to a rotation.

Returns

true if the matrix corresponds to a rotation, false otherwise.

2.5.3.7 template<class NumType> NumType screws::Skew< NumType >::norm ( ) const [inline]

Calculate and return the norm of the skew.

Returns

the norm of the skew.

2.5.3.8 template<class NumType> Skew<NumType> screws::Skew< NumType >::normalised ( ) const [inline]

Normalise and remove magnitude from skew symmetric matrix. Only rotation axis information remains.

Returns

the normalised skew.

2.5.3.9 template<class NumType> bool screws::Skew< NumType >::operator!= (const Skew< NumType > & *S*) const [inline]

Inequality operator.

Returns

true if one of the element-by-element comparisons return true. Otherwise, false.

2.5.3.10 template<class NumType> const NumType& screws::Skew< NumType >::operator() (const unsigned int & *i*, const unsigned int & *j*) const [inline]

Return the requested element (read).

Parameters

i	the index of the row.
j	the index of the column.

Note

No boundary check is performed.

2.5.3.11 template<class NumType> Skew<NumType> screws::Skew< NumType >::operator* (const NumType & *value*) const [inline]

Element-by-element multiplication with a value.

Note

This operation corresponds to scaling the rotation angle.

2.5.3.12 template<class NumType> const Skew<NumType>& screws::Skew< NumType >::operator*= (const NumType & *value*) [inline]

Element-by-element in-place multiplication with a value.

Note

This operation corresponds to scaling the rotation angle.

2.5.3.13 template<class NumType> Skew<NumType> screws::Skew< NumType >::operator+ (const Skew< NumType > & *S*) const [inline]

Element-by-element addition of skew matrices.

Note

This operation corresponds to multiplication of the respective rotations.

2.5.3.14 template<class NumType> const Skew<NumType>& screws::Skew< NumType >::operator+= (const Skew< NumType > & *S*) [inline]

Element-by-element in-place addition of skew matrices.

Note

This operation corresponds to multiplication of the respective rotations.

2.5.3.15 template<class NumType> bool screws::Skew< NumType >::operator== (const Skew< NumType > & *S*) const [inline]

Element-by-element exact equality operator.

Returns

true if all elements are exactly equal

2.5.3.16 template<class NumType> Skew<NumType> screws::Skew< NumType >::transpose ( ) const [inline]

Transposes the skew.

Returns

the transposed skew.

The documentation for this class was generated from the following files:

- /Users/cbergeles/Dropbox/prj/mevislab/CustomLibraries/General/Sources/Shared/Screws/rotation.hpp
- /Users/cbergeles/Dropbox/prj/mevislab/CustomLibraries/General/Sources/Shared/Screws/skew.hpp

2.6 screws::Translation< NumType > Class Template Reference

Implements a 3x1 translation along with necessary operators.

`#include <translation.hpp>`

Public Member Functions

- Translation ()

  Default constructor with zeros.
- ~Translation ()

  Default destructor.
- Translation (const NumType &x, const NumType &y, const NumType &z)

  Value based constructor.
- Translation< NumType > operator+ (const Translation< NumType > &T) const

  Element-by-element addition.
- const Translation< NumType > & operator+= (const Translation< NumType > &T)

  In-place element-by-element addition.
- Translation operator+ (const NumType &value) const

  Addition of a number to every element.
- const Translation & operator+= (const NumType &value)

  In-place addition of a number to every element.
- Translation< NumType > operator- (const Translation< NumType > &T) const

  Element-by-element subtraction.
- const Translation< NumType > & operator-= (const Translation< NumType > &T)

  In-place element-by-element subtraction.
- Translation< NumType > operator- (const NumType &value) const

  Subtraction of a number from every element.
- const Translation< NumType > & operator-= (const NumType &value)

  In-place subtraction of a number from every element.
- Translation< NumType > operator* (const Translation< NumType > &T) const

  Element-by-element multiplication.
- const Translation< NumType > & operator*= (const Translation< NumType > &T)

  In-place Element-by-element multiplication.
- Translation< NumType > operator* (const NumType &scale) const

  Multiplication of each element by a factor.
- const Translation< NumType > & operator*= (const NumType &scale)

  In-place multiplication of each element by a factor.
- Translation< NumType > operator/ (const Translation< NumType > &T) const

  Element-by-element division.
- const Translation< NumType > & operator/= (const Translation< NumType > &T)

  Element-by-element division.
- Translation< NumType > operator/ (const NumType &scale) const

  Division of each element by a factor.
- const Translation< NumType > & operator/= (const NumType &scale)

  Division of each element by a factor.
- NumType & operator() (const unsigned int &index)

  Accessor operator (write).
- const NumType & operator() (const unsigned int &index) const

  Accessor operator (read).
- Translation< NumType > & operator= (Translation< NumType > prototype)

Assignment operator.

- bool operator== (const Translation< NumType > &T) const

  Equality operator.

- bool operator!= (const Translation< NumType > &T) const

  Inequality operator.

- bool approxEq (const Translation< NumType > &T, double eps=FLT_EPSILON) const

  Approximal equality operator, within a given epsilon or system precision.

- NumType norm () const

  Calculate the norm2 of the translation.

- Translation< NumType > normalised () const

  Normalise the translation as if it were a vector.

- Translation< NumType > cross (const Translation< NumType > &T)

  Find the cross product with a translation vector.

- NumType dot (const Translation< NumType > &T)

  Find the dot product with a translation vector.

Protected Member Functions

- void **setData** (const NumType &x, const NumType &y, const NumType &z)

Protected Attributes

- Eigen::Matrix< NumType, 3, 1, 0, 3, 1 > **_data**

Friends

- template<class NumTypeRot >
  class **Rotation**
- template<class NumTypeHomo >
  class **HomogeneousTransform**
- template<class NumTypeVec >
  class **Vector6**
- template<class NumTypeTw >
  class **Twist**

2.6.1 Detailed Description

template<class NumType>class screws::Translation< NumType >

Implements a 3x1 translation along with necessary operators.

Note

Author: Christos Bergeles

Date

25th April 2015

2.6.2 Constructor & Destructor Documentation

2.6.2.1 template<class NumType> screws::Translation< NumType >::Translation (const NumType & *x*, const NumType & *y*, const NumType & *z*) [inline], [explicit]

Value based constructor.

Parameters

x	the x-coordinate of the translation.
y	the y-coordinate of the translation.
z	the z-coordinate of the translation.

2.6.3 Member Function Documentation

2.6.3.1 template<class NumType> bool screws::Translation< NumType >::approxEq (const Translation< NumType > & *T*, double *eps* = FLT_EPSILON) const [inline]

Approximal equality operator, within a given epsilon or system precision.

Parameters

T	compared translation.
eps	desired precision [default: machine precision].

2.6.3.2 template<class NumType> Translation<NumType> screws::Translation< NumType >::cross (const Translation< NumType > & *T*) [inline]

Find the cross product with a translation vector.

Returns

the cross product with the given translation vector.

2.6.3.3 template<class NumType> NumType screws::Translation< NumType >::dot (const Translation< NumType > & *T*) [inline]

Find the dot product with a translation vector.

Returns

the dot product with the given translation vector.

2.6.3.4 template<class NumType> NumType screws::Translation< NumType >::norm ( ) const [inline]

Calculate the norm2 of the translation.

Returns

the norm of the translation.

2.6.3.5 template<class NumType> Translation<NumType> screws::Translation< NumType >::normalised ( ) const [inline]

Normalise the translation as if it were a vector.

Returns

the normalised vector.

2.6.3.6 template<class NumType> bool screws::Translation< NumType >::operator!= (const Translation< NumType > & *T*) const [inline]

Inequality operator.

Returns

true if one of the element-by-element comparisons return true. Otherwise, false.

2.6.3.7 template<class NumType> NumType& screws::Translation< NumType >::operator() (const unsigned int & *index*) [inline]

Accessor operator (write).

Returns

the index-the element of the translation.

2.6.3.8 template<class NumType> Translation<NumType>& screws::Translation< NumType >::operator= (Translation< NumType > *prototype*) [inline]

Assignment operator.

Returns

A copy of the right hand side argument.

2.6.3.9 template<class NumType> bool screws::Translation< NumType >::operator== (const Translation< NumType > & *T*) const [inline]

Equality operator.

Returns

true if all element-by-element comparisons return true. Otherwise, false.

The documentation for this class was generated from the following files:

- /Users/cbergeles/Dropbox/prj/mevislab/CustomLibraries/General/Sources/Shared/Screws/homogeneous↩ Transform.hpp
- /Users/cbergeles/Dropbox/prj/mevislab/CustomLibraries/General/Sources/Shared/Screws/translation.hpp

2.7 screws::Twist< NumType > Class Template Reference

Implements a 4x4 twist matrix with basic functionality such as exponents, logs etc.

`#include <twist.hpp>`

Public Member Functions

- Twist ()

  Initialise with zeros. This corresponds to the identity homogeneous transformation matrix.
- Twist (const HomogeneousTransform< NumType > &HT)

Create a twist by taking the logarithm of a homogeneous transformation matrix.

- Twist (const TwistCoordinates< NumType > &V)

  Create a twist using a twistcoordinates vector.
- Twist (const NumType &t0, const NumType &t1, const NumType &t2, const NumType &t3, const NumType &t4, const NumType &t5)

  Create a twist using the 6 twist coordinates.
- ~Twist ()

  Default destructor.
- NumType pitch () const

  Calculate and return the pitch of the twist.
- TwistCoordinates< NumType > axis () const

  Calculate and return the axis of the twist.
- TwistCoordinates< NumType > coordinates () const

  Calculate and return the 6x1 twist coordinates.
- NumType norm () const

  Calculate and return the norm of the twist.
- HomogeneousTransform< NumType > exp (const NumType &theta=(NumType) 1) const

  Create a homogeneous transformation matrix by taking the exponential of the twist.
- bool isValid () const

  Perform verification that matrix is indeed in approriate format by checking the Skew component.
- Skew< NumType > skew () const

  Returns the skew symmetric (rotation) part of the twist.
- Translation< NumType > velocity () const

  Returns the velocity part of the twist.
- Twist< NumType > operator+ (const Twist< NumType > &T) const

  Element-by-element addition of twist matrices.
- const Twist< NumType > & operator+= (const Twist< NumType > &T)

  Element-by-element in-place addition of skew matrices.
- bool operator== (const Twist< NumType > &T) const

  Element-by-element exact equality operator.
- bool operator!= (const Twist< NumType > &T) const

  Inequality operator.
- bool approxEq (const Twist< NumType > &T, double eps=FLT_EPSILON) const

  Approximal equality operator, within a given epsilon or system precision.
- const NumType & operator() (const unsigned int &i, const unsigned int &j) const

  Return the requested element (read).

Protected Member Functions

- void **resetData** ()

Protected Attributes

- Skew< NumType > **_skew**
- Translation< NumType > **_velocity**
- NumType **_zeroVal**

Friends

- template<class NumTypeHomo >
  class **HomogeneousTransform**

2.7.1 Detailed Description

template<class NumType>class screws::Twist< NumType >

Implements a 4x4 twist matrix with basic functionality such as exponents, logs etc.

Note

Author: Christos Bergeles

Date

8th May 2015

2.7.2 Constructor & Destructor Documentation

2.7.2.1 template<class NumType> screws::Twist< NumType >::Twist (const HomogeneousTransform< NumType > & *HT*) [inline]

Create a twist by taking the logarithm of a homogeneous transformation matrix.

Parameters

HT	a homogeneous transformation matrix.

2.7.2.2 template<class NumType> screws::Twist< NumType >::Twist (const TwistCoordinates< NumType > & *V*) [inline]

Create a twist using a twistcoordinates vector.

Parameters

V	the twist coordinates.

Note

V(0)-V(2) correspond to the velocity, and V(3)-V(5) to the rotation.

2.7.2.3 template<class NumType> screws::Twist< NumType >::Twist (const NumType & *t0,* const NumType & *t1,* const NumType & *t2,* const NumType & *t3,* const NumType & *t4,* const NumType & *t5*) [inline]

Create a twist using the 6 twist coordinates.

Parameters

t0-t6	the twist coordinates.

2.7.3 Member Function Documentation

2.7.3.1 template<class NumType> bool screws::Twist< NumType >::approxEq (const Twist< NumType > & *T,* double *eps* = FLT_EPSILON) const [inline]

Approximal equality operator, within a given epsilon or system precision.

Parameters

T	compared twist matrix.
eps	desired precision [default: machine precision].

2.7.3.2 template<class NumType> TwistCoordinates<NumType> screws::Twist< NumType >::axis ( ) const [inline]

Calculate and return the axis of the twist.

Returns

a TwistCoordinates vector w containing the axis of the twist, with w(0:2) + ∗w(3:5), {R}.

2.7.3.3 template<class NumType> TwistCoordinates<NumType> screws::Twist< NumType >::coordinates ( ) const [inline]

Calculate and return the 6x1 twist coordinates.

Returns

the TwistCoordinates vector.

Note

V(0)-V(2) contain the velocity of the twist, and V(3)-V(5) the rotation.

2.7.3.4 template<class NumType> HomogeneousTransform<NumType> screws::Twist< NumType >::exp (const NumType & *theta* = (NumType)1) const [inline]

Create a homogeneous transformation matrix by taking the exponential of the twist.

Parameters

theta	the magnitude of the rotation.

Returns

the homogeneous transformation matrix.

2.7.3.5 template<class NumType> NumType screws::Twist< NumType >::norm ( ) const [inline]

Calculate and return the norm of the twist.

Returns

the norm of the twist.

2.7.3.6 template<class NumType> bool screws::Twist< NumType >::operator!= (const Twist< NumType > & *T*) const [inline]

Inequality operator.

Returns

true if one of the element-by-element comparisons return true. Otherwise, false.

2.7.3.7 template<class NumType> const NumType& screws::Twist< NumType >::operator() (const unsigned int & *i*, const unsigned int & *j*) const [inline]

Return the requested element (read).

Parameters

i	the index of the row.
j	the index of the column.

Note

No boundary check is performed.

2.7.3.8 template<class NumType> Twist<NumType> screws::Twist< NumType >::operator+ (const Twist< NumType > & *T*) const [inline]

Element-by-element addition of twist matrices.

Note

This operation corresponds to multiplication of the respective homogeneous transforms.

2.7.3.9 template<class NumType> const Twist<NumType>& screws::Twist< NumType >::operator+= (const Twist< NumType > & *T*) [inline]

Element-by-element in-place addition of skew matrices.

Note

This operation corresponds to multiplication of the respective rotations.

2.7.3.10 template<class NumType> bool screws::Twist< NumType >::operator== (const Twist< NumType > & *T*) const [inline]

Element-by-element exact equality operator.

Returns

true if all elements are exactly equal

2.7.3.11 template<class NumType> NumType screws::Twist< NumType >::pitch ( ) const [inline]

Calculate and return the pitch of the twist.

Returns

the pitch of the twist.

2.7.3.12 template<class NumType> Skew<NumType> screws::Twist< NumType >::skew ( ) const [inline]

Returns the skew symmetric (rotation) part of the twist.

Returns

the skew symmetric part of the twist.

2.7.3.13 template<class NumType> Translation<NumType> screws::Twist< NumType >::velocity ( ) const [inline]

Returns the velocity part of the twist.

Returns

the velocity (3x1 vector) part of the twist.

The documentation for this class was generated from the following files:

- /Users/cbergeles/Dropbox/prj/mevislab/CustomLibraries/General/Sources/Shared/Screws/homogeneous↩ Transform.hpp
- /Users/cbergeles/Dropbox/prj/mevislab/CustomLibraries/General/Sources/Shared/Screws/twist.hpp

2.8 screws::Vector6< NumType > Class Template Reference

Implements a 6x1 vector along with necessary operators.

`#include <vector6.hpp>`

Public Member Functions

- Vector6 ()

  Default constructor with zeros.
- ~Vector6 ()

  Default destructor.
- Vector6 (const Translation< NumType > &v0, const Translation< NumType > &v1)

  Vector based constructor.
- Vector6 (const NumType &v0, const NumType &v1, const NumType &v2, const NumType &v3, const Num↩ Type &v4, const NumType &v5)

  Value based constructor.
- Vector6< NumType > operator+ (const Vector6< NumType > &V) const

  Element-by-element addition.
- const Vector6< NumType > & operator+= (const Vector6< NumType > &V)

  In-place element-by-element addition.
- Vector6 operator+ (const NumType &value) const

  Addition of a number to every element.
- const Vector6 & operator+= (const NumType &value)

  In-place addition of a number to every element.
- Vector6< NumType > operator- (const Vector6< NumType > &V) const

  Element-by-element subtraction.
- const Vector6< NumType > & operator-= (const Vector6< NumType > &V)

  In-place element-by-element subtraction.
- Vector6< NumType > operator- (const NumType &value) const

  Subtraction of a number from every element.
- const Vector6< NumType > & operator-= (const NumType &value)

  In-place subtraction of a number from every element.
- Vector6< NumType > operator∗ (const Vector6< NumType > &V) const

  Element-by-element multiplication.
- const Vector6< NumType > & operator∗= (const Vector6< NumType > &V)

  In-place Element-by-element multiplication.

- Vector6< NumType > operator* (const NumType &scale) const
  Multiplication of each element by a factor.
- const Vector6< NumType > & operator*= (const NumType &scale)
  In-place multiplication of each element by a scalar factor.
- Vector6< NumType > operator/ (const Vector6< NumType > &V) const
  Element-by-element division.
- const Vector6< NumType > & operator/= (const Vector6< NumType > &V)
  Element-by-element division.
- Vector6< NumType > operator/ (const NumType &scale) const
  Division of each element by a factor.
- const Vector6< NumType > & operator/= (const NumType &scale)
  Division of each element by a factor.
- NumType & operator() (const unsigned int &index)
  Accessor operator (write).
- const NumType & operator() (const unsigned int &index) const
  Accessor operator (read).
- Vector6< NumType > & operator= (Vector6< NumType > prototype)
  Assignment operator.
- bool operator== (const Vector6< NumType > &V) const
  Equality operator.
- bool operator!= (const Vector6< NumType > &V) const
  Inequality operator.
- bool approxEq (const Vector6< NumType > &V, double eps=FLT_EPSILON) const
  Approximal equality operator, within a given epsilon or system precision.
- NumType norm () const
  Calculate the norm2 of the vector.
- Vector6< NumType > normalised () const
  Normalise the vector.
- NumType dot (const Vector6< NumType > &V)
  Find the dot product with another Vector6.

Protected Attributes

- Vector3< NumType > **_v0**
- Vector3< NumType > **_v1**

Friends

- template<class NumTypeTwist >
  class **Twist**

2.8.1 Detailed Description

template<class NumType>class screws::Vector6< NumType >

Implements a 6x1 vector along with necessary operators.

Note

Author: Christos Bergeles

Date

08th May 2015

2.8.2 Constructor & Destructor Documentation

2.8.2.1 template<class NumType> screws::Vector6< NumType >::Vector6 (const Translation< NumType > & *v0*, const Translation< NumType > & *v1*) [inline], [explicit]

Vector based constructor.

Parameters

v0	the first 3x1 values.
v1	the final 3x1 values.

2.8.2.2 template<class NumType> screws::Vector6< NumType >::Vector6 (const NumType & *v0*, const NumType & *v1*, const NumType & *v2*, const NumType & *v3*, const NumType & *v4*, const NumType & *v5*) [inline], [explicit]

Value based constructor.

Parameters

v0-v5,the	elements of the vector.

2.8.3 Member Function Documentation

2.8.3.1 template<class NumType> bool screws::Vector6< NumType >::approxEq (const Vector6< NumType > & *V*, double *eps* = FLT_EPSILON) const [inline]

Approximal equality operator, within a given epsilon or system precision.

Parameters

V	compared vector.
eps	desired precision [default: machine precision].

2.8.3.2 template<class NumType> NumType screws::Vector6< NumType >::dot (const Vector6< NumType > & *V*) [inline]

Find the dot product with another Vector6.

Returns

the dot product with the given Vector6.

2.8.3.3 template<class NumType> NumType screws::Vector6< NumType >::norm ( ) const [inline]

Calculate the norm2 of the vector.

Returns

the norm of the vector.

2.8.3.4 template<class NumType> Vector6<NumType> screws::Vector6< NumType >::normalised ( ) const [inline]

Normalise the vector.

Returns

the normalised vector.

2.8.3.5 template<class NumType> bool screws::Vector6< NumType >::operator!= (const Vector6< NumType > & *V*) const [inline]

Inequality operator.

Returns

true if one of the element-by-element comparisons return true. Otherwise, false.

2.8.3.6 template<class NumType> NumType& screws::Vector6< NumType >::operator() (const unsigned int & *index*) [inline]

Accessor operator (write).

Parameters

the	index in the vector.

Returns

the index-the element of the vector.

2.8.3.7 template<class NumType> Vector6<NumType>& screws::Vector6< NumType >::operator= (Vector6< NumType > *prototype*) [inline]

Assignment operator.

Returns

A copy of the right hand side argument.

2.8.3.8 template<class NumType> bool screws::Vector6< NumType >::operator== (const Vector6< NumType > & *V*) const [inline]

Equality operator.

Returns

true if all element-by-element comparisons return true. Otherwise, false.

The documentation for this class was generated from the following files:

- /Users/cbergeles/Dropbox/prj/mevislab/CustomLibraries/General/Sources/Shared/Screws/twist.hpp
- /Users/cbergeles/Dropbox/prj/mevislab/CustomLibraries/General/Sources/Shared/Screws/vector6.hpp

Index