

# SpatialAlgebra

Generated by Doxygen 1.12.0



# Chapter 1

## Hierarchical Index

### 1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

SpatialAlgebra::ArticulatedBodyInertia . . . . .	??
Eigen::Matrix3d	
Rotation . . . . .	??
SpatialAlgebra::PluckerTransform . . . . .	??
SpatialAlgebra::RigidBodyInertia . . . . .	??
SpatialAlgebra::SpatialOperations . . . . .	??
SpatialAlgebra::SpatialVector . . . . .	??
SpatialAlgebra::ForceVector . . . . .	??
SpatialAlgebra::MotionVector . . . . .	??



## Chapter 2

# Class Index

### 2.1 Class List

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

<a href="#">SpatialAlgebra::ArticulatedBodyInertia</a>	
Class representing inertial properties of an articulated body . . . . .	??
<a href="#">SpatialAlgebra::ForceVector</a>	
Class representing spatial force vectors . . . . .	??
<a href="#">SpatialAlgebra::MotionVector</a>	
Class representing spatial motion vectors . . . . .	??
<a href="#">SpatialAlgebra::PluckerTransform</a>	
Class representing spatial coordinate transformations . . . . .	??
<a href="#">SpatialAlgebra::RigidBodyInertia</a>	
Class representing the inertial properties of a rigid body . . . . .	??
<a href="#">Rotation</a>	
Class for handling 3D rotations with multiple representations . . . . .	??
<a href="#">SpatialAlgebra::SpatialOperations</a>	
Static class providing utility operations for spatial vectors and transforms . . . . .	??
<a href="#">SpatialAlgebra::SpatialVector</a>	
. . . . .	??



# Chapter 3

## File Index

### 3.1 File List

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

include/ <a href="#">ArticulatedBodyInertia.h</a>	
Class representing articulated body inertia properties . . . . .	??
include/ <a href="#">AxialScrewTransform.h</a> . . . . .	??
include/ <a href="#">ForceVector.h</a>	
Class representing spatial force vectors . . . . .	??
include/ <a href="#">MotionVector.h</a>	
Class representing spatial motion vectors . . . . .	??
include/ <a href="#">PluckerTransform.h</a>	
Class representing coordinate transformations in Plucker coordinates . . . . .	??
include/ <a href="#">RigidBodyInertia.h</a>	
Class representing rigid body inertia properties . . . . .	??
include/ <a href="#">Rotation.h</a>	
Class representing 3D rotations with various representations . . . . .	??
include/ <a href="#">SpatialOperations.h</a>	
Static utility class for spatial algebra operations . . . . .	??
include/ <a href="#">SpatialUtils.h</a>	
Utility functions for spatial algebra operations . . . . .	??
include/ <a href="#">SpatialVector.h</a> . . . . .	??





## Chapter 4

# Class Documentation

### 4.1 SpatialAlgebra::ArticulatedBodyInertia Class Reference

Class representing inertial properties of an articulated body.

```
#include <ArticulatedBodyInertia.h>
```

#### Public Member Functions

- [ArticulatedBodyInertia](#) (const Vector6d &inertia, const Eigen::Matrix3d &h, const Vector6d &M)  
*Construct articulated body inertia.*
- **ArticulatedBodyInertia** ()  
*Default constructor creating zero inertia.*
- const Vector6d & **getInertia** () const
- const Eigen::Matrix3d & **getH** () const
- const Vector6d & **getM** () const
- [ArticulatedBodyInertia operator+](#) (const [ArticulatedBodyInertia](#) &other) const  
*Add two articulated body inertias.*
- [ArticulatedBodyInertia operator+](#) (const [RigidBodyInertia](#) &other) const  
*Add rigid body inertia to articulated body inertia.*
- [ArticulatedBodyInertia operator\\*](#) (double scalar) const
- [fv apply](#) (const [mv](#) &[mv](#)) const  
*Apply articulated body inertia to motion vector.*
- void **print** () const  
*Print inertia properties.*

#### 4.1.1 Detailed Description

Class representing inertial properties of an articulated body.

Stores and manages generalized inertia matrix components for articulated body dynamics calculations.

#### 4.1.2 Constructor & Destructor Documentation

##### 4.1.2.1 ArticulatedBodyInertia()

```
SpatialAlgebra::ArticulatedBodyInertia::ArticulatedBodyInertia (  
    const Vector6d & inertia,  
    const Eigen::Matrix3d & h,  
    const Vector6d & M)
```

Construct articulated body inertia.

## Parameters

<i>inertia</i>	Generalized inertia vector
<i>h</i>	Coupling matrix
<i>M</i>	Mass matrix components

### 4.1.3 Member Function Documentation

#### 4.1.3.1 `apply()`

```
fv SpatialAlgebra::ArticulatedBodyInertia::apply (
    const mv & mv) const [inline]
```

Apply articulated body inertia to motion vector.

## Parameters

<i>mv</i>	Motion vector to apply
-----------	------------------------

## Returns

Resulting force vector

#### 4.1.3.2 `operator+()` [1/2]

```
ArticulatedBodyInertia SpatialAlgebra::ArticulatedBodyInertia::operator+ (
    const ArticulatedBodyInertia & other) const [inline]
```

Add two articulated body inertias.

## Parameters

<i>other</i>	<a href="#">ArticulatedBodyInertia</a> to add
--------------	---

## Returns

Combined [ArticulatedBodyInertia](#)

#### 4.1.3.3 `operator+()` [2/2]

```
ArticulatedBodyInertia SpatialAlgebra::ArticulatedBodyInertia::operator+ (
    const RigidBodyInertia & other) const [inline]
```

Add rigid body inertia to articulated body inertia.

## Parameters

other	<a href="#">RigidBodyInertia</a> to add
-------	---

## Returns

Combined [ArticulatedBodyInertia](#)

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

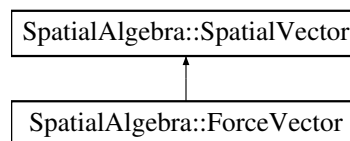
- include/[ArticulatedBodyInertia.h](#)

## 4.2 SpatialAlgebra::ForceVector Class Reference

Class representing spatial force vectors.

```
#include <ForceVector.h>
```

Inheritance diagram for SpatialAlgebra::ForceVector:



### Public Member Functions

- **ForceVector** ()  
*Default constructor creating zero force.*
- [ForceVector](#) (const Vector3d &angular, const Vector3d &linear)  
*Construct from angular and linear components.*
- [ForceVector](#) (const [SpatialVector](#) &other)  
*Construct from spatial vector.*
- Vector3d **getAngular** () const
- Vector3d **getLinear** () const
- [ForceVector](#) **operator+** (const [ForceVector](#) &other) const
- [ForceVector](#) **operator-** (const [ForceVector](#) &other) const
- [ForceVector](#) **operator\*** (double scalar) const
- [ForceVector](#) **crossMotion** (const [ForceVector](#) &other) const
- [ForceVector](#) **crossForce** (const [ForceVector](#) &other) const
- double **dot** (const [ForceVector](#) &other) const
- void **print** () const

## Public Member Functions inherited from [SpatialAlgebra::SpatialVector](#)

- **SpatialVector** (const Vector3d &angular, const Vector3d &linear)
- **SpatialVector** (const [SpatialVector](#) &other)
- Vector3d **getAngular** () const
- Vector3d **getLinear** () const
- [SpatialVector](#) **operator+** (const [SpatialVector](#) &other) const
- [SpatialVector](#) **operator-** (const [SpatialVector](#) &other) const
- [SpatialVector](#) **operator\*** (double scalar) const
- [SpatialVector](#) **crossMotion** (const [SpatialVector](#) &other) const
- [SpatialVector](#) **crossForce** (const [SpatialVector](#) &other) const
- double **dot** (const [SpatialVector](#) &other) const
- void **print** () const

## Additional Inherited Members

## Protected Attributes inherited from [SpatialAlgebra::SpatialVector](#)

- Vector3d **angular**
- Vector3d **linear**

### 4.2.1 Detailed Description

Class representing spatial force vectors.

Specialized spatial vector for representing force quantities like torques and linear forces.

### 4.2.2 Constructor & Destructor Documentation

#### 4.2.2.1 ForceVector() [1/2]

```
SpatialAlgebra::ForceVector::ForceVector (
    const Vector3d & angular,
    const Vector3d & linear)
```

Construct from angular and linear components.

#### Parameters

<i>angular</i>	Torque component
<i>linear</i>	Force component

#### 4.2.2.2 ForceVector() [2/2]

```
SpatialAlgebra::ForceVector::ForceVector (
    const SpatialVector & other)
```

Construct from spatial vector.

## Parameters

<i>other</i>	Spatial vector to convert
--------------	---------------------------

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

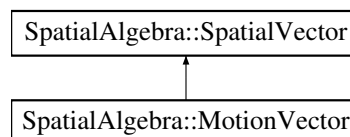
- include/ForceVector.h
- src/ForceVector.cpp

## 4.3 SpatialAlgebra::MotionVector Class Reference

Class representing spatial motion vectors.

```
#include <MotionVector.h>
```

Inheritance diagram for SpatialAlgebra::MotionVector:



### Public Member Functions

- **MotionVector** ()  
*Default constructor creating zero motion.*
- **MotionVector** (const Vector3d &angular, const Vector3d &linear)  
*Construct from angular and linear components.*
- **MotionVector** (const SpatialVector &other)  
*Construct from spatial vector.*
- Vector3d **getAngular** () const
- Vector3d **getLinear** () const
- **MotionVector operator+** (const MotionVector &other) const
- **MotionVector operator-** (const MotionVector &other) const
- **MotionVector operator\*** (double scalar) const
- **MotionVector crossMotion** (const MotionVector &other) const
- **MotionVector crossForce** (const MotionVector &other) const
- double **dot** (const MotionVector &other) const
- void **print** () const

### Public Member Functions inherited from SpatialAlgebra::SpatialVector

- **SpatialVector** (const Vector3d &angular, const Vector3d &linear)
- **SpatialVector** (const SpatialVector &other)
- Vector3d **getAngular** () const
- Vector3d **getLinear** () const
- **SpatialVector operator+** (const SpatialVector &other) const
- **SpatialVector operator-** (const SpatialVector &other) const
- **SpatialVector operator\*** (double scalar) const
- **SpatialVector crossMotion** (const SpatialVector &other) const
- **SpatialVector crossForce** (const SpatialVector &other) const
- double **dot** (const SpatialVector &other) const
- void **print** () const

## Additional Inherited Members

## Protected Attributes inherited from [SpatialAlgebra::SpatialVector](#)

- Vector3d **angular**
- Vector3d **linear**

### 4.3.1 Detailed Description

Class representing spatial motion vectors.

Specialized spatial vector for representing motion quantities like angular and linear velocities.

### 4.3.2 Constructor & Destructor Documentation

#### 4.3.2.1 MotionVector() [1/2]

```
SpatialAlgebra::MotionVector::MotionVector (
    const Vector3d & angular,
    const Vector3d & linear)
```

Construct from angular and linear components.

##### Parameters

<i>angular</i>	Angular velocity component
<i>linear</i>	Linear velocity component

#### 4.3.2.2 MotionVector() [2/2]

```
SpatialAlgebra::MotionVector::MotionVector (
    const SpatialVector & other)
```

Construct from spatial vector.

##### Parameters

<i>other</i>	Spatial vector to convert
--------------	---------------------------

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

- include/[MotionVector.h](#)
- src/MotionVector.cpp

## 4.4 SpatialAlgebra::PluckerTransform Class Reference

Class representing spatial coordinate transformations.

```
#include <PluckerTransform.h>
```

### Public Member Functions

- [PluckerTransform](#) (const [Rotation](#) &rotation, const [Vector3d](#) &translation)  
*Construct a Plucker transform.*
- [SpatialVector](#) transformMotion (const [SpatialVector](#) &vec) const  
*Transform a motion vector.*
- [SpatialVector](#) transformForce (const [SpatialVector](#) &vec) const  
*Transform a force vector.*
- [SpatialVector](#) inverseTransformMotion (const [SpatialVector](#) &vec) const  
*Inverse transform a motion vector.*
- [SpatialVector](#) inverseTransformForce (const [SpatialVector](#) &vec) const  
*Inverse transform a force vector.*
- [RigidBodyInertia](#) tformRBI (const [RigidBodyInertia](#) &lhat) const  
*Transform rigid body inertia.*
- [RigidBodyInertia](#) invtformRBI (const [RigidBodyInertia](#) &lhat) const  
*Inverse transform rigid body inertia.*
- [ArticulatedBodyInertia](#) tformABI (const [ArticulatedBodyInertia](#) &la) const
- [ArticulatedBodyInertia](#) invtformABI (const [ArticulatedBodyInertia](#) &la) const
- [PluckerTransform](#) **inverse** () const  
*Compute inverse transform.*
- auto **multiply** (const [PluckerTransform](#) &X) const
- auto **apply** (const [fv](#) &v) const
- auto **apply** (const [mv](#) &v) const
- [PluckerTransform](#) **apply** (const [PluckerTransform](#) &X) const
- void **print** () const  
*Print transform components.*
- auto **get** (int &a)
- auto **get** (double &b)

### 4.4.1 Detailed Description

Class representing spatial coordinate transformations.

Implements rigid body transformations in Plucker coordinates, consisting of rotation and translation components.

### 4.4.2 Constructor & Destructor Documentation

#### 4.4.2.1 PluckerTransform()

```
PluckerTransform::PluckerTransform (
    const Rotation & rotation,
    const Vector3d & translation)
```

Construct a Plucker transform.

## Parameters

<i>rotation</i>	<a href="#">Rotation</a> component
<i>translation</i>	Translation component

### 4.4.3 Member Function Documentation

#### 4.4.3.1 `inverseTransformForce()`

```
SpatialVector SpatialAlgebra::PluckerTransform::inverseTransformForce (
    const SpatialVector & vec) const
```

Inverse transform a force vector.

## Parameters

<i>vec</i>	Input force vector
------------	--------------------

## Returns

Inverse transformed force vector

#### 4.4.3.2 `inverseTransformMotion()`

```
SpatialVector SpatialAlgebra::PluckerTransform::inverseTransformMotion (
    const SpatialVector & vec) const
```

Inverse transform a motion vector.

## Parameters

<i>vec</i>	Input motion vector
------------	---------------------

## Returns

Inverse transformed motion vector

#### 4.4.3.3 `invtfomRBI()`

```
RigidBodyInertia SpatialAlgebra::PluckerTransform::invtfomRBI (
    const RigidBodyInertia & Ihat) const
```

Inverse transform rigid body inertia.



## Parameters

<i>Ihat</i>	Input rigid body inertia
-------------	--------------------------

## Returns

Inverse transformed inertia

**4.4.3.4 tformRBI()**

```
RigidBodyInertia SpatialAlgebra::PluckerTransform::tformRBI (  
    const RigidBodyInertia & Ihat) const
```

Transform rigid body inertia.

## Parameters

<i>Ihat</i>	Input rigid body inertia
-------------	--------------------------

## Returns

Transformed inertia

**4.4.3.5 transformForce()**

```
SpatialVector PluckerTransform::transformForce (  
    const SpatialVector & vec) const
```

Transform a force vector.

## Parameters

<i>vec</i>	Input force vector
------------	--------------------

## Returns

Transformed force vector

**4.4.3.6 transformMotion()**

```
SpatialVector PluckerTransform::transformMotion (  
    const SpatialVector & vec) const
```

Transform a motion vector.

## Parameters

vec	Input motion vector
-----	---------------------

## Returns

Transformed motion vector

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

- include/[PluckerTransform.h](#)
- src/[PluckerTransform.cpp](#)

## 4.5 SpatialAlgebra::RigidBodyInertia Class Reference

Class representing the inertial properties of a rigid body.

```
#include <RigidBodyInertia.h>
```

### Public Member Functions

- [RigidBodyInertia](#) (double mass, const Vector3d &com, const Vector6d &inertiaMatrixLT)  
*Construct a rigid body inertia.*
- [RigidBodyInertia](#) ()  
*Default constructor creating zero inertia.*
- double **getMass** () const
- const Vector3d & **getCom** () const
- const Vector6d & **getInertiaMatrixLT** () const
- [RigidBodyInertia operator+](#) (const [RigidBodyInertia](#) &other) const  
*Add two rigid body inertias.*
- [RigidBodyInertia operator\\*](#) (double scalar) const  
*Scale rigid body inertia.*
- [ForceVector apply](#) (const [MotionVector](#) &mv) const  
*Apply rigid body inertia to motion vector.*
- void **print** () const  
*Print inertia properties.*

### 4.5.1 Detailed Description

Class representing the inertial properties of a rigid body.

Stores and manages mass, center of mass, and rotational inertia matrix for rigid body dynamics calculations.

### 4.5.2 Constructor & Destructor Documentation

#### 4.5.2.1 RigidBodyInertia()

```
SpatialAlgebra::RigidBodyInertia::RigidBodyInertia (
    double mass,
    const Vector3d & com,
    const Vector6d & inertiaMatrixLT) [inline]
```

Construct a rigid body inertia.

## Parameters

<i>mass</i>	Mass of the body
<i>com</i>	Center of mass position
<i>inertiaMatrixLT</i>	Lower triangular part of inertia matrix

### 4.5.3 Member Function Documentation

#### 4.5.3.1 apply()

```
ForceVector SpatialAlgebra::RigidBodyInertia::apply (
    const MotionVector & mv) const [inline]
```

Apply rigid body inertia to motion vector.

## Parameters

<i>mv</i>	Motion vector to apply
-----------	------------------------

## Returns

Resulting force vector

Implements:  $[I + com \times v; m*v - com \times ]$

#### 4.5.3.2 operator\*()

```
RigidBodyInertia SpatialAlgebra::RigidBodyInertia::operator* (
    double scalar) const [inline]
```

Scale rigid body inertia.

## Parameters

<i>scalar</i>	Scaling factor
---------------	----------------

## Returns

Scaled [RigidBodyInertia](#)

#### 4.5.3.3 operator+()

```
RigidBodyInertia SpatialAlgebra::RigidBodyInertia::operator+ (
    const RigidBodyInertia & other) const [inline]
```

Add two rigid body inertias.

## Parameters

other	<a href="#">RigidBodyInertia</a> to add
-------	---

## Returns

Combined [RigidBodyInertia](#)

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

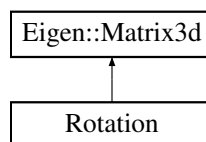
- include/[RigidBodyInertia.h](#)

## 4.6 Rotation Class Reference

Class for handling 3D rotations with multiple representations.

```
#include <Rotation.h>
```

Inheritance diagram for Rotation:



### Public Member Functions

- **Rotation** ()  
*Default constructor initializing to identity rotation.*
- **Rotation** (const Eigen::Matrix3d &matrix)  
*Construct from 3x3 rotation matrix.*
- **Rotation** (const Eigen::AngleAxisd &angleAxis)  
*Construct from angle-axis representation.*
- **Rotation** (const Eigen::Quaterniond &quaternion)  
*Construct from quaternion representation.*
- void **setFromAngleAxis** (const Eigen::AngleAxisd &angleAxis)  
*Set rotation from angle-axis representation.*
- void **setFromQuaternion** (const Eigen::Quaterniond &quaternion)  
*Set rotation from quaternion representation.*
- Eigen::AngleAxisd **toAngleAxis** () const  
*Convert to angle-axis representation.*
- Eigen::Quaterniond **toQuaternion** () const  
*Convert to quaternion representation.*
- **Rotation inverse** () const  
*Compute inverse rotation.*
- **Rotation transpose** () const  
*Compute matrix transpose.*
- **Rotation operator\*** (const **Rotation** &other) const  
*Multiply with another rotation.*
- Eigen::Vector3d **operator\*** (const Eigen::Vector3d &vector) const  
*Apply rotation to a vector.*

### 4.6.1 Detailed Description

Class for handling 3D rotations with multiple representations.

Extends Eigen::Matrix3d to provide additional functionality for rotation operations including conversions between different rotation representations.

### 4.6.2 Constructor & Destructor Documentation

#### 4.6.2.1 Rotation() [1/3]

```
Rotation::Rotation (
    const Eigen::Matrix3d & matrix)
```

Construct from 3x3 rotation matrix.

Parameters

<i>matrix</i>	Input rotation matrix
---------------	-----------------------

#### 4.6.2.2 Rotation() [2/3]

```
Rotation::Rotation (
    const Eigen::AngleAxisd & angleAxis)
```

Construct from angle-axis representation.

Parameters

<i>angleAxis</i>	Angle-axis rotation
------------------	---------------------

#### 4.6.2.3 Rotation() [3/3]

```
Rotation::Rotation (
    const Eigen::Quaterniond & quaternion)
```

Construct from quaternion representation.

Parameters

<i>quaternion</i>	Unit quaternion representing rotation
-------------------	---------------------------------------

### 4.6.3 Member Function Documentation

#### 4.6.3.1 operator\*() [1/2]

```
Eigen::Vector3d Rotation::operator* (
    const Eigen::Vector3d & vector) const
```

Apply rotation to a vector.

**Parameters**

<i>vector</i>	3D vector to rotate
---------------	---------------------

**Returns**

Rotated vector

**4.6.3.2 operator\*() [2/2]**

```
Rotation Rotation::operator* (  
    const Rotation & other) const
```

Multiply with another rotation.

**Parameters**

<i>other</i>	Right-hand side rotation
--------------	--------------------------

**Returns**

Combined rotation

**4.6.3.3 setFromAngleAxis()**

```
void Rotation::setFromAngleAxis (  
    const Eigen::AngleAxisd & angleAxis)
```

Set rotation from angle-axis representation.

**Parameters**

<i>angleAxis</i>	Angle-axis rotation
------------------	---------------------

**4.6.3.4 setFromQuaternion()**

```
void Rotation::setFromQuaternion (  
    const Eigen::Quaterniond & quaternion)
```

Set rotation from quaternion representation.

**Parameters**

<i>quaternion</i>	Unit quaternion
-------------------	-----------------

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

- include/Rotation.h
- src/Rotation.cpp

## 4.7 SpatialAlgebra::SpatialOperations Class Reference

Static class providing utility operations for spatial vectors and transforms.

```
#include <SpatialOperations.h>
```

### Static Public Member Functions

- static [SpatialVector](#) [crossProductMotion](#) (const [SpatialVector](#) &v1, const [SpatialVector](#) &v2)  
*Computes the motion cross product between two spatial vectors.*
- static [SpatialVector](#) [crossProductForce](#) (const [SpatialVector](#) &v, const [SpatialVector](#) &f)  
*Computes the force cross product between two spatial vectors.*
- static [RigidBodyInertia](#) [transformInertia](#) (const [RigidBodyInertia](#) &inertia, const [PluckerTransform](#) &transform)  
*Transforms rigid body inertia using a Plucker transform.*

### 4.7.1 Detailed Description

Static class providing utility operations for spatial vectors and transforms.

### 4.7.2 Member Function Documentation

#### 4.7.2.1 crossProductForce()

```
static SpatialVector SpatialAlgebra::SpatialOperations::crossProductForce (
    const SpatialVector & v,
    const SpatialVector & f) [static]
```

Computes the force cross product between two spatial vectors.

#### Parameters

<i>v</i>	Force spatial vector
<i>f</i>	Motion spatial vector

#### Returns

Result of force cross product

#### 4.7.2.2 crossProductMotion()

```
static SpatialVector SpatialAlgebra::SpatialOperations::crossProductMotion (
    const SpatialVector & v1,
    const SpatialVector & v2) [static]
```

Computes the motion cross product between two spatial vectors.

## Parameters

<i>v1</i>	First spatial vector
<i>v2</i>	Second spatial vector

## Returns

Result of motion cross product

## 4.7.2.3 transformInertia()

```
static RigidBodyInertia SpatialAlgebra::SpatialOperations::transformInertia (
    const RigidBodyInertia & inertia,
    const PluckerTransform & transform) [static]
```

Transforms rigid body inertia using a Plucker transform.

## Parameters

<i>inertia</i>	Input rigid body inertia
<i>transform</i>	Plucker transform to apply

## Returns

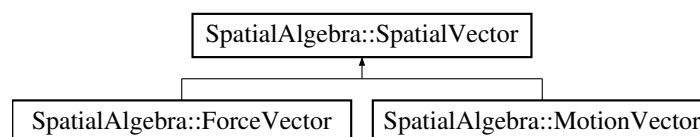
Transformed rigid body inertia

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

- include/[SpatialOperations.h](#)

## 4.8 SpatialAlgebra::SpatialVector Class Reference

Inheritance diagram for SpatialAlgebra::SpatialVector:



## Public Member Functions

- **SpatialVector** (const Vector3d &angular, const Vector3d &linear)
- **SpatialVector** (const [SpatialVector](#) &other)
- Vector3d **getAngular** () const
- Vector3d **getLinear** () const
- [SpatialVector](#) **operator+** (const [SpatialVector](#) &other) const
- [SpatialVector](#) **operator-** (const [SpatialVector](#) &other) const
- [SpatialVector](#) **operator\*** (double scalar) const
- [SpatialVector](#) **crossMotion** (const [SpatialVector](#) &other) const
- [SpatialVector](#) **crossForce** (const [SpatialVector](#) &other) const
- double **dot** (const [SpatialVector](#) &other) const
- void **print** () const



### Protected Attributes

- Vector3d **angular**
- Vector3d **linear**

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

- include/SpatialVector.h
- src/SpatialVector.cpp



# Chapter 5

## File Documentation

### 5.1 include/ArticulatedBodyInertia.h File Reference

Class representing articulated body inertia properties.

```
#include <array>
#include <iostream>
#include "RigidBodyInertia.h"
#include "SpatialUtils.h"
#include <Eigen/Geometry>
```

#### Classes

- class [SpatialAlgebra::ArticulatedBodyInertia](#)  
*Class representing inertial properties of an articulated body.*

#### Typedefs

- using **Vector6d** = Eigen::Matrix<double, 6, 1>
- using **SpatialAlgebra::abi** = [ArticulatedBodyInertia](#)

#### 5.1.1 Detailed Description

Class representing articulated body inertia properties.

## 5.2 ArticulatedBodyInertia.h

[Go to the documentation of this file.](#)

```
00001 #ifndef ARTIC_BODY_INERTIA_H
00002 #define ARTIC_BODY_INERTIA_H
00003
00009 #include <array>
00010 #include <iostream>
00011 #include "RigidBodyInertia.h"
00012 #include "SpatialUtils.h"
00013 #include <Eigen/Geometry> // Include for RotationMatrix
00014
00015 using Vector6d = Eigen::Matrix<double, 6, 1>;
00016
00017 namespace SpatialAlgebra
00018 {
00025     class ArticulatedBodyInertia
00026     {
00027     private:
00028         Vector6d Inertia;
00029         Eigen::Matrix3d H;
00030         Vector6d M;
00031
00032     public:
00039         ArticulatedBodyInertia(const Vector6d &inertia, const Eigen::Matrix3d &h, const Vector6d &M);
00040
00042         ArticulatedBodyInertia();
00043
00044         // getters
00045         inline const Vector6d &getInertia() const { return Inertia; }
00046         inline const Eigen::Matrix3d &getH() const { return H; }
00047         inline const Vector6d &getM() const { return M; }
00048
00054         inline ArticulatedBodyInertia operator+(const ArticulatedBodyInertia &other) const;
00055
00061         inline ArticulatedBodyInertia operator+(const RigidBodyInertia &other) const;
00062
00063         inline ArticulatedBodyInertia operator*(double scalar) const
00064         {
00065             return ArticulatedBodyInertia(Inertia * scalar, H * scalar, M * scalar);
00066         }
00067
00073         inline fv apply(const mv &mv) const;
00074
00076         inline void print() const;
00077     };
00078
00079     using abi = ArticulatedBodyInertia;
00080
00081 } // namespace SpatialAlgebra
00082
00083 #endif
```

## 5.3 AxialScrewTransform.h

```
00001
```

## 5.4 include/ForceVector.h File Reference

Class representing spatial force vectors.

```
#include <iostream>
#include "SpatialVector.h"
```

### Classes

- class [SpatialAlgebra::ForceVector](#)  
*Class representing spatial force vectors.*

## Typedefs

- using `SpatialAlgebra::fv` = `ForceVector`

### 5.4.1 Detailed Description

Class representing spatial force vectors.

## 5.5 ForceVector.h

[Go to the documentation of this file.](#)

```
00001 #ifndef FORCE_VECTOR_H
00002 #define FORCE_VECTOR_H
00003
00009 #include <iostream>
00010
00011 #include "SpatialVector.h"
00012
00013 // Force vector class inherits from SpatialVector
00014 namespace SpatialAlgebra
00015 {
00022     class ForceVector : public SpatialVector
00023     {
00024     public:
00026         ForceVector();
00027
00033         ForceVector(const Vector3d &angular, const Vector3d &linear);
00034
00039         ForceVector(const SpatialVector &other);
00040
00041         // Accessors
00042         Vector3d getAngular() const;
00043         Vector3d getLinear() const;
00044
00045         // Basic operations
00046         ForceVector operator+(const ForceVector &other) const;
00047         ForceVector operator-(const ForceVector &other) const;
00048         ForceVector operator*(double scalar) const;
00049
00050         // Cross-product operations
00051         ForceVector crossMotion(const ForceVector &other) const; // crm equivalent
00052         ForceVector crossForce(const ForceVector &other) const; // crf equivalent
00053
00054         // dot product
00055         double dot(const ForceVector &other) const;
00056
00057         // Printing
00058         void print() const;
00059     };
00060
00061     using fv = ForceVector;
00062
00063 } // namespace SpatialAlgebra
00064
00065 #endif
```

## 5.6 include/MotionVector.h File Reference

Class representing spatial motion vectors.

```
#include <array>
#include <iostream>
#include "SpatialVector.h"
```

## Classes

- class [SpatialAlgebra::MotionVector](#)  
*Class representing spatial motion vectors.*

## Typedefs

- using [SpatialAlgebra::mv](#) = [MotionVector](#)

### 5.6.1 Detailed Description

Class representing spatial motion vectors.

## 5.7 MotionVector.h

[Go to the documentation of this file.](#)

```

00001 #ifndef MOTION_VECTOR_H
00002 #define MOTION_VECTOR_H
00003
00009 #include <array>
00010 #include <iostream>
00011
00012 #include "SpatialVector.h"
00013
00014 // Motion vector class inherits from SpatialVector
00015
00016 namespace SpatialAlgebra
00017 {
00024     class MotionVector : public SpatialVector
00025     {
00026     public:
00028         MotionVector();
00029
00035         MotionVector(const Vector3d &angular, const Vector3d &linear);
00036
00041         MotionVector(const SpatialVector &other);
00042
00043         // Accessors
00044         Vector3d getAngular() const;
00045         Vector3d getLinear() const;
00046
00047         // Basic operations
00048         MotionVector operator+(const MotionVector &other) const;
00049         MotionVector operator-(const MotionVector &other) const;
00050         MotionVector operator*(double scalar) const;
00051
00052         // Cross-product operations
00053         MotionVector crossMotion(const MotionVector &other) const; // crm equivalent
00054         MotionVector crossForce(const MotionVector &other) const; // crf equivalent
00055
00056         // dot product
00057         double dot(const MotionVector &other) const;
00058
00059         // Printing
00060         void print() const;
00061     };
00062
00063     using mv = MotionVector;
00064
00065 } // namespace SpatialAlgebra
00066
00067 #endif

```

## 5.8 include/PluckerTransform.h File Reference

Class representing coordinate transformations in Plucker coordinates.

```
#include "SpatialVector.h"
#include "Rotation.h"
#include "ForceVector.h"
#include "MotionVector.h"
#include "RigidBodyInertia.h"
#include "ArticulatedBodyInertia.h"
#include <Eigen/Geometry>
```

### Classes

- class [SpatialAlgebra::PluckerTransform](#)  
*Class representing spatial coordinate transformations.*

### Typedefs

- using **Vector3d** = Eigen::Matrix<double, 3, 1>
- using **SpatialAlgebra::plux** = [PluckerTransform](#)

### 5.8.1 Detailed Description

Class representing coordinate transformations in Plucker coordinates.

## 5.9 PluckerTransform.h

[Go to the documentation of this file.](#)

```
00001 #ifndef PLUCKER_TRANSFORM_H
00002 #define PLUCKER_TRANSFORM_H
00003
00009 #include "SpatialVector.h"
00010 #include "Rotation.h"
00011 #include "ForceVector.h"
00012 #include "MotionVector.h"
00013 #include "RigidBodyInertia.h"
00014 #include "ArticulatedBodyInertia.h"
00015 #include <Eigen/Geometry> // Include for RotationMatrix
00016
00017 using Vector3d = Eigen::Matrix<double, 3, 1>;
00018
00019 namespace SpatialAlgebra
00020 {
00027     class PluckerTransform
00028     {
00029     private:
00030         Rotation rotation;
00031         Vector3d translation;
00032
00033     public:
00039         PluckerTransform(const Rotation &rotation, const Vector3d &translation);
00040
00046         SpatialVector transformMotion(const SpatialVector &vec) const;
00047
00053         SpatialVector transformForce(const SpatialVector &vec) const;
00054
00060         SpatialVector inverseTransformMotion(const SpatialVector &vec) const;
00061
00067         SpatialVector inverseTransformForce(const SpatialVector &vec) const;
```

```

00068
00074     RigidBodyInertia tformRBI(const RigidBodyInertia &Ihat) const;
00075
00081     RigidBodyInertia invtformRBI(const RigidBodyInertia &Ihat) const;
00082
00083     // transform abi
00084     ArticulatedBodyInertia tformABI(const ArticulatedBodyInertia &Ia) const;
00085     ArticulatedBodyInertia invtformABI(const ArticulatedBodyInertia &Ia) const;
00086
00088     PluckerTransform inverse() const;
00089
00090     // multiply by X
00091     // PluckerTransform multiply(const PluckerTransform &X) const;
00092     auto multiply(const PluckerTransform &X) const;
00093
00094     auto apply(const fv& v) const{
00095         return transformMotion(v);
00096     }
00097     auto apply(const mv& v) const{
00098         return transformForce(v);
00099     }
00100
00101     PluckerTransform apply(const PluckerTransform &X) const;
00102
00103     // template <typename T>
00104     // T apply(const T&) const{
00105
00106     // }
00107
00109     void print() const;
00110
00111     // testing
00112     auto get(int &a) { return a; }
00113     auto get(double &b)
00114     {
00115         char c{'c'};
00116         return c;
00117     }
00118 };
00119
00120     using plux = PluckerTransform;
00121
00122 } // namespace SpatialAlgebra
00123
00124 #endif

```

## 5.10 include/RigidBodyInertia.h File Reference

Class representing rigid body inertia properties.

```

#include <array>
#include <iostream>
#include "SpatialVector.h"
#include "ForceVector.h"
#include "MotionVector.h"
#include <Eigen/Geometry>

```

### Classes

- class [SpatialAlgebra::RigidBodyInertia](#)  
Class representing the inertial properties of a rigid body.

### Typedefs

- using **Vector3d** = Eigen::Matrix<double, 3, 1>
- using **Vector6d** = Eigen::Matrix<double, 6, 1>
- using **SpatialAlgebra::rbi** = [RigidBodyInertia](#)



### 5.10.1 Detailed Description

Class representing rigid body inertia properties.

## 5.11 RigidBodyInertia.h

[Go to the documentation of this file.](#)

```

00001 #ifndef RIGID_BODY_INERTIA_H
00002 #define RIGID_BODY_INERTIA_H
00003
00009 #include <array>
00010 #include <iostream>
00011 #include "SpatialVector.h"
00012 #include "ForceVector.h"
00013 #include "MotionVector.h"
00014 #include <Eigen/Geometry> // Include for RotationMatrix
00015
00016 using Vector3d = Eigen::Matrix<double, 3, 1>;
00017 using Vector6d = Eigen::Matrix<double, 6, 1>;
00018
00019 namespace SpatialAlgebra
00020 {
00021     class RigidBodyInertia
00022     {
00023     private:
00024         double mass;
00025         Vector3d com;
00026         Vector6d inertiaMatrixLT;
00027
00028     public:
00029         inline RigidBodyInertia(double mass, const Vector3d &com, const Vector6d &inertiaMatrixLT)
00030             : mass(mass), com(com), inertiaMatrixLT(inertiaMatrixLT) {}
00031
00032         inline RigidBodyInertia()
00033         {
00034             RigidBodyInertia(0.0, {0.0, 0.0, 0.0}, {0.0, 0.0, 0.0, 0.0, 0.0, 0.0});
00035         }
00036
00037         // Getters
00038         inline double getMass() const { return mass; }
00039         inline const Vector3d &getCom() const { return com; }
00040         inline const Vector6d &getInertiaMatrixLT() const { return inertiaMatrixLT; }
00041
00042         inline RigidBodyInertia operator+(const RigidBodyInertia &other) const
00043         {
00044             // const double sumMass = mass + other.mass;
00045             // const Vector3d sumCom = com + other.com;
00046             return RigidBodyInertia(mass + other.mass, com + other.com, inertiaMatrixLT +
00047 other.inertiaMatrixLT);
00048         }
00049
00050         inline RigidBodyInertia operator*(double scalar) const // Changed to pass by value
00051         {
00052             return RigidBodyInertia(mass * scalar, com * scalar, inertiaMatrixLT * scalar);
00053         }
00054
00055         inline ForceVector apply(const MotionVector &mv) const
00056         {
00057             // rbi = [m, com, I_LT], mv = [v, w]
00058             // rbi.apply(mv) = [I + com x v; m*v - com x w]
00059             Vector3d omega = mv.getAngular();
00060             Vector3d v = mv.getLinear();
00061             Vector3d comCrossV = com.cross(v);
00062             Vector3d comCrossOmega = com.cross(omega);
00063             Vector3d Iomega = inertiaMatrixLT.head<3>().cwiseProduct(omega) +
00064 inertiaMatrixLT.tail<3>().cwiseProduct(omega);
00065             Vector3d force = mass * v - comCrossOmega;
00066             Vector3d torque = Iomega + comCrossV;
00067             return ForceVector(torque, force);
00068         }
00069
00070         inline void print() const
00071         {
00072             std::cout << "Mass: " << mass << '\n'
00073 << "Center of mass: " << com.transpose() << '\n'
00074 << "Inertia matrix (LT): " << inertiaMatrixLT.transpose() << std::endl;
00075         }
00076     };
00077 }
00078
00079 #endif

```

```

00106     using rbi = RigidBodyInertia;
00107
00108 } // namespace SpatialAlgebra
00109
00110 #endif

```

## 5.12 include/Rotation.h File Reference

Class representing 3D rotations with various representations.

```
#include <Eigen/Dense>
```

### Classes

- class [Rotation](#)  
*Class for handling 3D rotations with multiple representations.*

### 5.12.1 Detailed Description

Class representing 3D rotations with various representations.

## 5.13 Rotation.h

[Go to the documentation of this file.](#)

```

00001 #ifndef ROTATION_H
00002 #define ROTATION_H
00003
00009 #include <Eigen/Dense>
00010
00017 class Rotation : public Eigen::Matrix3d
00018 {
00019 public:
00021     Rotation();
00022
00027     Rotation(const Eigen::Matrix3d &matrix);
00028
00033     Rotation(const Eigen::AngleAxisd &angleAxis);
00034
00039     Rotation(const Eigen::Quaterniond &quaternion);
00040
00045     void setFromAngleAxis(const Eigen::AngleAxisd &angleAxis);
00046
00051     void setFromQuaternion(const Eigen::Quaterniond &quaternion);
00052
00054     Eigen::AngleAxisd toAngleAxis() const;
00055
00057     Eigen::Quaterniond toQuaternion() const;
00058
00060     Rotation inverse() const;
00061
00063     Rotation transpose() const;
00064
00070     Rotation operator*(const Rotation &other) const;
00071
00077     Eigen::Vector3d operator*(const Eigen::Vector3d &vector) const;
00078 };
00079
00080 #endif // ROTATION_H

```

## 5.14 include/SpatialOperations.h File Reference

Static utility class for spatial algebra operations.

```
#include "SpatialVector.h"
#include "PluckerTransform.h"
#include "RigidBodyInertia.h"
#include <Eigen/Geometry>
```

### Classes

- class [SpatialAlgebra::SpatialOperations](#)  
*Static class providing utility operations for spatial vectors and transforms.*

### 5.14.1 Detailed Description

Static utility class for spatial algebra operations.

## 5.15 SpatialOperations.h

[Go to the documentation of this file.](#)

```
00001 #ifndef SPATIAL_OPERATIONS_H
00002 #define SPATIAL_OPERATIONS_H
00003
00009 #include "SpatialVector.h"
00010 #include "PluckerTransform.h"
00011 #include "RigidBodyInertia.h"
00012 #include <Eigen/Geometry> // Include for RotationMatrix
00013
00014 namespace SpatialAlgebra {
00015
00019 class SpatialOperations {
00020 public:
00027     static SpatialVector crossProductMotion(const SpatialVector& v1, const SpatialVector& v2);
00028
00035     static SpatialVector crossProductForce(const SpatialVector& v, const SpatialVector& f);
00036
00043     static RigidBodyInertia transformInertia(const RigidBodyInertia& inertia,
00044                                             const PluckerTransform& transform);
00045 };
00046
00047 } // namespace SpatialAlgebra
00048
00049 #endif
```

## 5.16 include/SpatialUtils.h File Reference

Utility functions for spatial algebra operations.

```
#include "SpatialVector.h"
#include "MotionVector.h"
#include "ForceVector.h"
```

## Functions

- Eigen::Matrix3d [SpatialAlgebra::skew](#) (const Eigen::Vector3d &v) noexcept  
*Creates a 3x3 skew-symmetric matrix from a 3D vector.*
- double [SpatialAlgebra::dot](#) (const [SpatialVector](#) &v1, const [SpatialVector](#) &v2) noexcept  
*Computes the spatial dot product between two spatial vectors.*
- double [SpatialAlgebra::dot](#) (const [MotionVector](#) &v1, const [MotionVector](#) &v2) noexcept  
*Computes the spatial dot product between two motion vectors.*
- double [SpatialAlgebra::dot](#) (const [ForceVector](#) &v1, const [ForceVector](#) &v2) noexcept  
*Computes the spatial dot product between two force vectors.*
- double **SpatialAlgebra::dot** (const [MotionVector](#) &v1, const [ForceVector](#) &v2) noexcept
- [ForceVector](#) [SpatialAlgebra::cross](#) (const [MotionVector](#) &v1, const [ForceVector](#) &v2) noexcept  
*Computes the spatial cross product between a motion vector and a force vector.*

### 5.16.1 Detailed Description

Utility functions for spatial algebra operations.

This file contains various utility functions used in spatial algebra computations, including cross products, dot products, and matrix operations.

### 5.16.2 Function Documentation

#### 5.16.2.1 `cross()`

```
ForceVector SpatialAlgebra::cross (
    const MotionVector & v1,
    const ForceVector & v2) [inline], [noexcept]
```

Computes the spatial cross product between a motion vector and a force vector.

#### Parameters

<code>v1</code>	Motion vector
<code>v2</code>	Force vector

#### Returns

Resulting force vector

Implements:  $\text{crf}(v1, v2) = [w1 \times f1 + v1 \times f2, w1 \times f1]$

#### 5.16.2.2 `dot()` [1/3]

```
double SpatialAlgebra::dot (
    const ForceVector & v1,
    const ForceVector & v2) [inline], [noexcept]
```

Computes the spatial dot product between two force vectors.

**Parameters**

<i>v1</i>	First force vector
<i>v2</i>	Second force vector

**Returns**

Scalar dot product result

**5.16.2.3 dot() [2/3]**

```
double SpatialAlgebra::dot (
    const MotionVector & v1,
    const MotionVector & v2) [inline], [noexcept]
```

Computes the spatial dot product between two motion vectors.

**Parameters**

<i>v1</i>	First motion vector
<i>v2</i>	Second motion vector

**Returns**

Scalar dot product result

**5.16.2.4 dot() [3/3]**

```
double SpatialAlgebra::dot (
    const SpatialVector & v1,
    const SpatialVector & v2) [inline], [noexcept]
```

Computes the spatial dot product between two spatial vectors.

**Parameters**

<i>v1</i>	First spatial vector
<i>v2</i>	Second spatial vector

**Returns**

Scalar dot product result

**5.16.2.5 skew()**

```
Eigen::Matrix3d SpatialAlgebra::skew (
    const Eigen::Vector3d & v) [inline], [noexcept]
```

Creates a 3x3 skew-symmetric matrix from a 3D vector.

## Parameters

$v$	Input 3D vector
-----	-----------------

## Returns

Skew-symmetric matrix representation

The resulting matrix  $S$  satisfies  $S \cdot x = v \times x$  for any vector  $x$

## 5.17 SpatialUtils.h

[Go to the documentation of this file.](#)

```

00001 #ifndef SPATIAL_UTILS_H
00002 #define SPATIAL_UTILS_H
00003
00012 #include "SpatialVector.h"
00013 #include "MotionVector.h"
00014 #include "ForceVector.h"
00015
00016 namespace SpatialAlgebra {
00017
00024 inline Eigen::Matrix3d skew(const Eigen::Vector3d& v) noexcept {
00025     Eigen::Matrix3d m;
00026     m << 0, -v(2), v(1),
00027         v(2), 0, -v(0),
00028         -v(1), v(0), 0;
00029     return m;
00030 }
00031
00038 inline double dot(const SpatialVector& v1, const SpatialVector& v2) noexcept {
00039     return v1.getAngular().dot(v2.getAngular()) +
00040         v1.getLinear().dot(v2.getLinear());
00041 }
00042
00049 inline double dot(const MotionVector& v1, const MotionVector& v2) noexcept {
00050     return dot(static_cast<const SpatialVector&>(v1),
00051         static_cast<const SpatialVector&>(v2));
00052 }
00053
00060 inline double dot(const ForceVector& v1, const ForceVector& v2) noexcept {
00061     return dot(static_cast<const SpatialVector&>(v1),
00062         static_cast<const SpatialVector&>(v2));
00063 }
00064
00065 inline double dot(const MotionVector& v1, const ForceVector& v2) noexcept {
00066     return dot(static_cast<const SpatialVector&>(v1),
00067         static_cast<const SpatialVector&>(v2));
00068 }
00069
00077 inline ForceVector cross(const MotionVector& v1, const ForceVector& v2) noexcept {
00078     const Vector3d& w1 = v1.getAngular();
00079     const Vector3d& v1_lin = v1.getLinear();
00080     const Vector3d& f1 = v2.getAngular();
00081     const Vector3d& f2 = v2.getLinear();
00082
00083     return ForceVector(
00084         w1.cross(f1),
00085         w1.cross(f2) + v1_lin.cross(f1)
00086     );
00087 };
00088
00089 } // namespace SpatialAlgebra
00090 #endif // SPATIAL_UTILS_H

```

## 5.18 SpatialVector.h

```

00001 #ifndef SPATIAL_VECTOR_H
00002 #define SPATIAL_VECTOR_H
00003

```

```

00004 // #include <array>
00005 // #include <iostream>
00006 #include <Eigen/Dense>
00007
00008 using Vector3d = Eigen::Matrix<double, 3, 1>;
00009
00010 namespace SpatialAlgebra
00011 {
00012
00013     class SpatialVector
00014     {
00015     protected:
00016         Vector3d angular; // : Angular velocity/force
00017         Vector3d linear; // v or f: Linear velocity/force
00018
00019     public:
00020         // Constructors
00021         SpatialVector();
00022         SpatialVector(const Vector3d &angular, const Vector3d &linear);
00023
00024         SpatialVector(const SpatialVector &other);
00025
00026         // Accessors
00027         Vector3d getAngular() const;
00028         Vector3d getLinear() const;
00029
00030         // Basic operations
00031         SpatialVector operator+(const SpatialVector &other) const;
00032         SpatialVector operator-(const SpatialVector &other) const;
00033         SpatialVector operator*(double scalar) const;
00034
00035         // Cross-product operations
00036         SpatialVector crossMotion(const SpatialVector &other) const; // crm equivalent
00037         SpatialVector crossForce(const SpatialVector &other) const; // crf equivalent
00038
00039         // dot product
00040         double dot(const SpatialVector &other) const;
00041
00042         // Printing
00043         void print() const;
00044     };
00045
00046 } // namespace SpatialAlgebra
00047 #endif

```

