OrientationModel

5.0

Generated by Doxygen 1.8.14

Contents

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Module Documentation

5.1 Models

Modules

- Utils

5.1.1 Detailed Description

5.2 Utils

Modules

- Orientation

5.2.1 Detailed Description

5.3 Orientation

Files

- file orientation.hh

  Define the Orientation class.
- file orientation_messages.hh

  Define the class OrientationMessages, the class that specifies the message IDs used in the orientation model.
- file eigen_rotation.cc

  Define Orientation methods related to computing single axis rotations.
- file euler_angles.cc

  Define Orientation methods related to computing Euler angles.
- file orientation.cc

  Define methods for the NewOrientation class.
- file orientation_messages.cc

  Implement the class OrientationMessages.

Namespaces

- jeod

  Namespace jeod.

5.3.1 Detailed Description

Chapter 6

Namespace Documentation

6.1 jeod Namespace Reference

Namespace jeod.

Data Structures

- struct EulerInfo

  Contains data needed to construct a transformation matrix given a sequence of Euler angles and to extract a sequence of Euler angles from a matrix.
- class Orientation

  Specifies the orientiation of one reference frame with respect to another.
- class OrientationMessages

  Declares messages associated with the orientation model.

Variables

- static const EulerInfo Euler_info [12]

  Contains twelve EulerInfo objects, one per each of the JEOD Euler sequences.

6.1.1 Detailed Description

Namespace jeod.

6.1.2 Variable Documentation

6.1.2.1 Euler_info

```
const EulerInfo jeod::Euler_info[12]  [static]
```

Initial value:

```
= {

   { {0, 1, 2},   0,    2,    true,   true },
   { {0, 2, 1},   0,    1,   false,   true },
   { {1, 2, 0},   1,    0,    true,   true },
   { {1, 0, 2},   1,    2,   false,   true },
   { {2, 0, 1},   2,    1,    true,   true },
   { {2, 1, 0},   2,    0,   false,   true },
   { {0, 1, 0},   2,    2,    true,  false },
   { {0, 2, 0},   1,    1,   false,  false },
   { {1, 2, 1},   0,    0,    true,  false },
   { {1, 0, 1},   2,    2,   false,  false },
   { {2, 0, 2},   1,    1,    true,  false },
   { {2, 1, 2},   0,    0,   false,  false }
}
```

Contains twelve EulerInfo objects, one per each of the JEOD Euler sequences.

The elements are arranged per the values of the Orientation::EulerSequence enumeration items.

Definition at line 96 of file euler_angles.cc.

Referenced by jeod::Orientation::compute_euler_angles_from_matrix(), jeod::Orientation::compute_matrix_from_euler_angles(), and jeod::Orientation::compute_quaternion_from_euler_angles().

Chapter 7

Data Structure Documentation

7.1 jeod::EulerInfo Struct Reference

Contains data needed to construct a transformation matrix given a sequence of Euler angles and to extract a sequence of Euler angles from a matrix.

Data Fields

- unsigned int indices [3]

  The axes about which the rotations are performed in the order in which the rotations are performed, with X=0, Y=1, Z=2.
- unsigned int alternate_x

  The initial element of the sequence for aerodynamics sequences, but the index of the omitted axis for astronomical sequences.
- unsigned int alternate_z

  The final element of the sequence for aerodynamics sequences, but the index of the omitted axis for astronomical sequences.
- bool is_even_permutation

  Indicates whether the 3-axis rotation sequence generated by replacing the final element of the sequence with the one axis not specified by the first two elements of the sequence is an even (true) permutation or odd permutation (false) of XYZ.
- bool is_aerodynamics_sequence

  True if the sequence is an aerodynamics sequence such as XYZ; false for an astronomical sequence such as ZXZ.

7.1.1 Detailed Description

Contains data needed to construct a transformation matrix given a sequence of Euler angles and to extract a sequence of Euler angles from a matrix.

See Orientation::compute_euler_angles_from_matrix for details.

Definition at line 49 of file euler_angles.cc.

7.1.2 Field Documentation

7.1.2.1 alternate_x

```
unsigned int jeod::EulerInfo::alternate_x
```

The initial element of the sequence for aerodynamics sequences, but the index of the omitted axis for astronomical sequences.

For example, the omitted axis in a ZXZ sequence is Y=1.trick_units(–)

Definition at line 63 of file euler_angles.cc.

Referenced by jeod::Orientation::compute_euler_angles_from_matrix().

7.1.2.2 alternate_z

```
unsigned int jeod::EulerInfo::alternate_z
```

The final element of the sequence for aerodynamics sequences, but the index of the omitted axis for astronomical sequences.

trick_units(–)

Definition at line 69 of file euler_angles.cc.

Referenced by jeod::Orientation::compute_euler_angles_from_matrix().

7.1.2.3 indices

```
unsigned int jeod::EulerInfo::indices[3]
```

The axes about which the rotations are performed in the order in which the rotations are performed, with X=0, Y=1, Z=2.

For example, an XYZ or roll pitch yaw sequence is {0,1,2} while a ZXZ sequence is {2,0,2}.trick_units(–)

Definition at line 56 of file euler_angles.cc.

Referenced by jeod::Orientation::compute_euler_angles_from_matrix(), jeod::Orientation::compute_matrix_from↩ _euler_angles(), and jeod::Orientation::compute_quaternion_from_euler_angles().

7.1.2.4 is_aerodynamics_sequence

```
bool jeod::EulerInfo::is_aerodynamics_sequence
```

True if the sequence is an aerodynamics sequence such as XYZ; false for an astronomical sequence such as ZXZ.

trick_units(–)

Definition at line 87 of file euler_angles.cc.

Referenced by jeod::Orientation::compute_euler_angles_from_matrix().

7.1.2.5 is_even_permutation

```
bool jeod::EulerInfo::is_even_permutation
```

Indicates whether the 3-axis rotation sequence generated by replacing the final element of the sequence with the one axis not specified by the first two elements of the sequence is an even (true) permutation or odd permutation (false) of XYZ.

The alternative 3-axis sequence is identical to the original sequence in the case of aerodynamics sequences. The astronomical ZXZ sequence becomes ZXY via this replacement rule. Since ZXY is an even permutation of XYZ, the is_even_permutation member for a ZXZ sequence is true.trick_units(–)

Definition at line 81 of file euler_angles.cc.

Referenced by jeod::Orientation::compute_euler_angles_from_matrix().

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

- euler_angles.cc

7.2 jeod::Orientation Class Reference

Specifies the orientiation of one reference frame with respect to another.

```
#include <orientation.hh>
```

Public Types

- enum DataSource {
  InputNone = -1, InputMatrix = 0, InputQuaternion = 1, InputEigenRotation = 2,
  InputEulerRotation = 3 }

  Specifies which representation has been input by the user.
- enum EulerSequence {
  NoSequence = -1, EulerXYZ = 0, EulerXZY = 1, EulerYZX = 2,
  EulerYXZ = 3, EulerZXY = 4, EulerZYX = 5, EulerXYX = 6,
  EulerXZX = 7, EulerYZY = 8, EulerYXY = 9, EulerZXZ = 10,
  EulerZYZ = 11, Roll_Pitch_Yaw = 0, Roll_Yaw_Pitch = 1, Pitch_Yaw_Roll = 2,
  Pitch_Roll_Yaw = 3, Yaw_Roll_Pitch = 4, Yaw_Pitch_Roll = 5, RollPitchYaw = 0,
  RollYawPitch = 1, PitchYawRoll = 2, PitchRollYaw = 3, YawRollPitch = 4,
  YawPitchRoll = 5 }

  Identifies which type of Euler sequence has been specified.

Public Member Functions

- Orientation (void)

    Construct an Orientation instance.
- Orientation (const double trans_in[3][3])

    Construct an Orientation instance from a transformation matrix.
- Orientation (const Quaternion &quat_in)

    Construct an Orientation instance from a Quaternion.
- Orientation (double eigen_angle_in, const double eigen_axis[3])

    Construct an Orientation instance from an eigen rotation.
- Orientation (EulerSequence sequence_in, const double angles_in[3])

    Construct an Orientation instance from an Euler rotation.
- virtual ∼Orientation (void)

    Destruct an Orientation instance.
- virtual void reset (void)

    Forget that we have any data.
- virtual void compute_transform (void)

    Compute the transformation matrix from the source.
- virtual void compute_quaternion (void)

    Compute the left transformation quaternion from the source.
- virtual void compute_eigen_rotation (void)

    Compute the eigen rotation from the source.
- virtual void compute_euler_angles (void)

    Compute the eigen rotation from the source.
- virtual void compute_all_products (void)

    Compute all represented charts on SO3 from the specified source.
- virtual void compute_transformation_and_quaternion (void)

    Compute the transformation matrix and quaternion.
- void set_quaternion (const Quaternion &quat)

    Reset the instance with a new quaternion.
- void get_quaternion (Quaternion &quat)

    Accessor for the left transformation quaternion.
- void set_transform (const double trans[3][3])

    Reset the instance with a new matrix.
- void get_transform (double trans[3][3])

    Accessor for the transformation matrix.
- void set_eigen_rotation (double eigen_angle, const double eigen_axis[3])

    Reset the instance with a new eigen rotation.
- void get_eigen_rotation (double ∗eigen_angle, double eigen_axis[3])

    Accessor for the eigen rotation.
- void set_euler_angles (EulerSequence sequence, const double angles[3])

    Reset the instance with a new Euler rotation.
- void set_euler_angles (const double angles[3])

    Reset the instance with a new Euler rotation.
- void get_euler_angles (EulerSequence ∗sequence, double angles[3])

    Accessor for the Euler angles.
- void get_euler_angles (double angles[3])

    Accessor for the Euler angles.
- EulerSequence get_euler_sequence (void)

    Accessor for the Euler sequence data member.
- void set_euler_sequence (EulerSequence sequence)

    Set the euler_sequence data member.
- void clear_euler_sequence (void)

    Reset the euler_sequence data member.

Static Public Member Functions

- static void compute_quaternion_from_euler_angles (EulerSequence sequence, const double angles[3], Quaternion &quat)

  Compute the left transformation quaternion from the Euler sequence.
- static void compute_matrix_from_euler_angles (EulerSequence sequence, const double angles[3], double trans[3][3])

  Compute the transformation matrix from the Euler sequence.
- static void compute_euler_angles_from_matrix (const double trans[3][3], EulerSequence sequence, double angles[3])

  Extract an Euler sequence from the transformation matrix.
- static void compute_matrix_from_eigen_rotation (double eigen_angle, const double eigen_axis[3], double trans[3][3])

  Compute the transformation matrix from the eigen rotation.
- static void compute_eigen_rotation_from_matrix (const double trans[3][3], double ∗eigen_angle, double eigen_axis[3])

  Compute the eigen rotation from the transformation matrix.

Data Fields

- DataSource data_source

  Orientation data source – specifies whether the user has provided as input an Euler rotation, a transformation matrix, or a left transformation quaternion.
- EulerSequence euler_sequence

  The Euler rotation sequence corresponding to euler_angles.
- double euler_angles [3]

  Euler angles corresponding to rotation sequence euler_sequence.
- double trans [3][3]

  Transformation matrix.
- Quaternion quat

  Left transformation unit quaternion.
- double eigen_angle

  Single axis rotation angle.
- double eigen_axis [3]

  Single axis rotation axis unit vector.

Protected Member Functions

- void mark_input_as_available ()

  Mark the item specified by the data_source as available.
- void compute_quaternion_from_euler_angles (void)

  Compute the left transformation quaternion that corresponds to the provided Euler rotation sequence.
- void compute_matrix_from_euler_angles (void)

  Compute the transformation matrix that corresponds to the provided Euler rotation sequence.
- void compute_euler_angles_from_matrix (void)

  Compute an Euler rotation sequence that corresponds to the provided transformation matrix.
- void compute_matrix_from_eigen_rotation (void)

  Compute the transformation matrix that corresponds to the provided eigen rotation.
- void compute_eigen_rotation_from_matrix (void)

  Compute a eigen rotation that corresponds to the provided transformation matrix.

Protected Attributes

- bool have_transformation_

    True if transformation matrix has been set/computed.
- bool have_quaternion_

    True if quaternion has been set/computed.
- bool have_eigen_rotation_

    True if eigen rotation has been set/computed.
- bool have_euler_angles_

    True if an Euler rotation has been set/computed.

Static Protected Attributes

- static double gimbal_lock_threshold = 1e-13

    Threshold for detecting gimbal lock in compute_euler_angles_from_matrix.

Friends

- class InputProcessor
- void init_attrjeod__Orientation ()

7.2.1 Detailed Description

Specifies the orientiation of one reference frame with respect to another.

There are many competing charts on the rotation group. This class provides means for representing rotations as Euler rotations, transformation matrices, left transformation quaternions, and eigen rotations. The class also provides mechanisms for converting these representations into the alternative representations.

Definition at line 81 of file orientation.hh.

7.2.2 Member Enumeration Documentation

7.2.2.1 DataSource

`enum jeod::Orientation::DataSource`

Specifies which representation has been input by the user.

Enumerator

InputNone	No source specified.
InputMatrix	Transformation matrices supplied by user.
InputQuaternion	Quaternions supplied by user.
InputEigenRotation	Single axis rotation supplied by user.
InputEulerRotation	Euler sequence and angles supplied by user.

Definition at line 91 of file orientation.hh.

7.2.2.2 EulerSequence

`enum jeod::Orientation::EulerSequence`

Identifies which type of Euler sequence has been specified.

Enumerator

NoSequence	No sequence specified.
EulerXYZ	XYX sequence (roll pitch yaw)
EulerXZY	XZY sequence (roll yaw pitch)
EulerYZX	YZX sequence (pitch yaw roll)
EulerYXZ	YXZ sequence (pitch roll yaw)
EulerZXY	ZXY sequence (yaw roll pitch)
EulerZYX	ZYX sequence (yaw pitch roll)
EulerXYX	XYX sequence.
EulerXZX	XZX sequence.
EulerYZY	YZY sequence.
EulerYXY	YXY sequence.
EulerZXZ	The canonical ZXZ Euler sequence.
EulerZYZ	ZYZ sequence.
Roll_Pitch_Yaw	XYX sequence (roll pitch yaw)
Roll_Yaw_Pitch	XZY sequence (roll yaw pitch)
Pitch_Yaw_Roll	YZX sequence (pitch yaw roll)
Pitch_Roll_Yaw	YXZ sequence (pitch roll yaw)
Yaw_Roll_Pitch	ZXY sequence (yaw roll pitch)
Yaw_Pitch_Roll	ZYX sequence (yaw pitch roll)
RollPitchYaw	XYX sequence (roll pitch yaw)
RollYawPitch	XZY sequence (roll yaw pitch)
PitchYawRoll	YZX sequence (pitch yaw roll)
PitchRollYaw	YXZ sequence (pitch roll yaw)
YawRollPitch	ZXY sequence (yaw roll pitch)
YawPitchRoll	ZYX sequence (yaw pitch roll)

Definition at line 103 of file orientation.hh.

7.2.3 Constructor & Destructor Documentation

7.2.3.1 Orientation() [1/5]

```
jeod::Orientation::Orientation (
            void )
```

Construct an Orientation instance.

All data products are marked as unavailable, the two enum values are set to invalid values, and the composite elements are set to their default values.

Definition at line 69 of file orientation.cc.

References eigen_axis, euler_angles, and trans.

7.2.3.2 Orientation() [2/5]

```
jeod::Orientation::Orientation (
            const double trans_in[3][3] )  [explicit]
```

Construct an Orientation instance from a transformation matrix.

Parameters

in	*trans_in*	Transformation matrix

Definition at line 94 of file orientation.cc.

References eigen_axis, euler_angles, and trans.

7.2.3.3 Orientation() [3/5]

```
jeod::Orientation::Orientation (
            const Quaternion & quat_in )  [explicit]
```

Construct an Orientation instance from a Quaternion.

Parameters

in	*quat_in*	Quaternion

Definition at line 120 of file orientation.cc.

References eigen_axis, euler_angles, and trans.

7.2.3.4 Orientation() [4/5]

```
jeod::Orientation::Orientation (
            double eigen_angle_in,
            const double eigen_axis_in[3] )  [explicit]
```

Construct an Orientation instance from an eigen rotation.

Parameters

in	*eigen_angle↩_in*	Rotation angle Units: r
in	*eigen_axis_in*	Rotation axis, unit vector

Definition at line 146 of file orientation.cc.

References eigen_axis, euler_angles, and trans.

7.2.3.5 Orientation() [5/5]

```
jeod::Orientation::Orientation (
            EulerSequence sequence_in,
            const double angles_in[3] )  [explicit]
```

Construct an Orientation instance from an Euler rotation.

Parameters

in	*sequence↩_in*	Euler sequence
in	*angles_in*	Euler angles Units: r

Definition at line 174 of file orientation.cc.

References eigen_axis, euler_angles, and trans.

7.2.3.6 ∼Orientation()

```
jeod::Orientation::~Orientation (
            void  )  [virtual]
```

Destruct an Orientation instance.

This is intentionally null; this class doesn't allocate resources.

Definition at line 201 of file orientation.cc.

7.2.4 Member Function Documentation

7.2.4.1 clear_euler_sequence()

```
void jeod::Orientation::clear_euler_sequence (
            void )
```

Reset the euler_sequence data member.

Issues arise if the data source is the Euler rotation sequence. The resolution is to preserve the existing input elsewhere.

Definition at line 906 of file orientation.cc.

References compute_matrix_from_euler_angles(), compute_quaternion_from_euler_angles(), data_source, euler_sequence, have_euler_angles_, have_quaternion_, have_transformation_, InputEulerRotation, InputMatrix, and NoSequence.

Referenced by set_euler_sequence().

7.2.4.2 compute_all_products()

```
void jeod::Orientation::compute_all_products (
            void ) [virtual]
```

Compute all represented charts on SO3 from the specified source.

Definition at line 552 of file orientation.cc.

References compute_eigen_rotation(), compute_euler_angles(), compute_quaternion(), and compute_transform().

7.2.4.3 compute_eigen_rotation()

```
void jeod::Orientation::compute_eigen_rotation (
            void ) [virtual]
```

Compute the eigen rotation from the source.

Definition at line 425 of file orientation.cc.

References compute_eigen_rotation_from_matrix(), compute_matrix_from_euler_angles(), data_source, eigen_angle, eigen_axis, have_eigen_rotation_, have_transformation_, InputEigenRotation, InputEulerRotation, InputMatrix, InputNone, InputQuaternion, mark_input_as_available(), and quat.

Referenced by compute_all_products(), and get_eigen_rotation().

7.2.4.4 compute_eigen_rotation_from_matrix() [1/2]

```
void jeod::Orientation::compute_eigen_rotation_from_matrix (
            const double trans[3][3],
            double * eigen_angle,
            double eigen_axis[3] )  [static]
```

Compute the eigen rotation from the transformation matrix.

There are several alternate expressions for computing the eigen rotation from a matrix, all of which are equivalent in infinite precision arithmetic. The use of finite precision arithmetic means that care must be taken in choosing the algorithm to be used. The starting point is the generic expression

$$T_{ij} = \cos\phi\,\delta_{ij} + (1-\cos\phi)\,\hat{u}_i\hat{u}_j + \epsilon_{ijk}\sin\phi\,\hat{u}_k$$

From this, the trace of the matrix and the difference between and sum of pairs of off-diagonal elements are

$$\begin{aligned} \mathrm{tr}(T) &= 2\cos\phi + 1 \\ T_{ij} - T_{ji} &= 2\epsilon_{ijk}\sin\phi\,\hat{u}_k \\ T_{ij} + T_{ji} &= 2(1-\cos\phi)\,\hat{u}_i\hat{u}_j \end{aligned}$$

Method 1
One approach to determining the eigen rotation involves the construction of a vector of differences between pairs of off-diagonal elements of the transformation matrix,

$$d_k = T_{ij} - T_{ji} = 2\sin\phi\,\hat{u}_k$$

where (i,j,k) is an even permutation of (0,1,2). With this,

$$\begin{aligned} \phi &= \arcsin\left(\frac{||\mathbf{d}||}{2}\right) \\ \hat{\mathbf{u}} &= \frac{\mathbf{d}}{||\mathbf{d}||} \end{aligned}$$

Note that the above of the inverse sine will restrict the rotation angle to be between 0 and 90 degrees. Special processing is needed when the rotation angle is between 90 and 180 degrees. Note also that the symmetric difference vector will be identically zero if the rotation angle is 0 or 180 degrees and will be very small for rotation angles close to 0 or 180 degrees. The precision loss for rotation angles near 0 and 180 degrees means the individual components of the eigen axis will not be as precise with this approach compared to alternatives.

Method 2
The diagonal elements of the matrix yields another method for determining the single axis rotation angle and the rotation axis:

$$\begin{aligned} \phi &= \arccos\left(\frac{\mathrm{tr}(T)-1}{2}\right) \\ |\hat{u}_i| &= \sqrt{\frac{T_{ii}-\cos\phi}{1-\cos\phi}} \end{aligned}$$

Note that this approach determines the magnitudes but not the signs of the components of the eigen axis vector. Because this method is based on the inverse cosine, the calculated phi angle will be less precise than that obtained by method 1 for angles near 0 or 180 degrees. The unit vector however will be more accurate than that obtained from method 1 for small rotation angles.

Method 3
Yet another alternative for computing components of the eigen axis is to use the sum of pairs of off-diagonal elements of the transformation metrix,

$$\begin{aligned} T_{ij} + T_{ji} &= 2(1-\cos\phi)\,\hat{u}_i\hat{u}_j \\ T_{ik} + T_{ki} &= 2(1-\cos\phi)\,\hat{u}_i\hat{u}_k \end{aligned}$$

This enables the calculation of two components of the unit vector. One component needs to be computed by one of the two previous methods.

Assumptions and Limitations

- The matrix is a proper transformation matrix.

Parameters

in	*trans*	Transformation matrix
out	*eigen_angle*	Resultant rotation angle Units: r
out	*eigen_axis*	Resultant rotation axis

Definition at line 203 of file eigen_rotation.cc.

References eigen_angle, eigen_axis, and trans.

7.2.4.5 compute_eigen_rotation_from_matrix() [2/2]

```
void jeod::Orientation::compute_eigen_rotation_from_matrix (
            void ) [inline], [protected]
```

Compute a eigen rotation that corresponds to the provided transformation matrix.

Definition at line 358 of file orientation.hh.

References eigen_angle, eigen_axis, and trans.

Referenced by compute_eigen_rotation().

7.2.4.6 compute_euler_angles()

```
void jeod::Orientation::compute_euler_angles (
            void ) [virtual]
```

Compute the eigen rotation from the source.

Definition at line 481 of file orientation.cc.

References compute_euler_angles_from_matrix(), compute_matrix_from_eigen_rotation(), data_source, euler_sequence, EulerXYZ, EulerZYZ, have_euler_angles_, have_transformation_, InputEigenRotation, InputEulerRotation, InputMatrix, InputNone, InputQuaternion, jeod::OrientationMessages::invalid_enum, mark_input_as_available(), quat, and trans.

Referenced by compute_all_products(), and get_euler_angles().

7.2.4.7 compute_euler_angles_from_matrix() [1/2]

```
void jeod::Orientation::compute_euler_angles_from_matrix (
            const double trans[3][3],
            EulerSequence euler_sequence,
            double euler_angles[3] )  [static]
```

Extract an Euler sequence from the transformation matrix.

A transformation matrix constructed from an XYZ Euler sequence is of the form

$$\left[\begin{array}{ccc} \cos\psi\cos\theta & \cdots & \cdots \\ -\sin\psi\cos\theta & \cdots & \cdots \\ \sin\theta & -\cos\theta\sin\phi & \cos\theta\cos\phi \end{array}\right]$$

Note that the [2][0] element of the matrix depends on theta only. The other two elements of the leftmost column are simple terms that depend on theta and psi only, and the other two elements of the bottommost row are simple terms that depend on theta and phi only. Those five elements are the key to extracting an XYZ Euler sequence from a transformation matrix. The same principle applies to all twelve of the Euler sequences: Five key elements contain all of the information needed to extract the desired sequence. The location and form of those key elements of course depends on the sequence.

A problem arises in the above when cos(theta) is zero, or nearly so. This siutation is called 'gimbal lock'. Those four elements used to determine phi and psi are zero or nearly so. Fortunately That ugly stuff isn't so ugly in the case of gimbal lock. Once again looking at the matrix generated from an XYZ Euler sequence, when theta=pi/2 the matrix becomes

$$\left[\begin{array}{ccc} 0 & \sin(\phi+\psi) & -cos(\phi+\psi) \\ 0 & \cos(\phi+\psi) & \sin(\phi+\psi) \\ 1 & 0 & 0 \end{array}\right]$$

In this case there no way to determine both phi and psi; all that can be determined is their sum. One way to overcome this problem is to arbitrarily set one of those angles to an arbitrary value such as zero. That is the approach used in this method. This arbitrary setting enables an XYZ Euler sequence to be extracted from the matrix even in the case of gimbal lock. The same principle once again applies to all twelve sequences.

In summary, for a transformation matrix corresponding to an XYZ sequence,

- The [2][0] element of the matrix specifies theta.
- The [1][0] and [0][0] elements of the matrix specify psi.
- The [2][1] and [2][2] elements of the matrix specify phi. These psi and phi values are valid only when gimbal lock is not present.
- The [1][2] and [1][1] elements of the matrix specify phi in the case of gimbal lock.

Extending this analysis to the remaining eleven sequences provides the essential information needed to extract the desired Euler angles from a transformation matrix. This information is captured in the EulerInfo array Euler_info defined at the head of this file. With a reference `info` to the appropriate element of this array,

- The [info.indices[2]][info.indices[0]] element of the matrix specifies the angle theta.
- The [info.indices[1]][info.indices[0]] and [info.alternate_x][info.indices[0]] elements of the matrix specify the angle psi when gimbal lock is not present.
- The [info.indices[2]][info.indices[1]] and [info.indices[2]][info.alternate_z] elements of the matrix specify the angle phi when gimbal lock is not present.
- The [info.indices[1]][info.alternate_z] and [info.indices[1]][info.indices[1]] elements of the matrix specify angle phi when gimbal lock is present.

Assumptions and Limitations

- To within numerical accuracy, the transformation matrix in the Orientation object *is* a proper transformation matrix:
  - The magnitude of each row and column vector is nearly one.
  - The inner product of any two different rows / two different columns of the matrix nearly zero.
  - The determinant of the matrix is nearly one.
  - An element whose value is outside the range [-1,1] is only slightly outside that range and the deviation is numerical.

Parameters

`in`	*trans*	Transformation matrix
`in`	*euler_sequence*	Euler sequence
`out`	*euler_angles*	Resultant Euler angles Units: r

Definition at line 283 of file euler_angles.cc.

References jeod::EulerInfo::alternate_x, jeod::EulerInfo::alternate_z, euler_angles, jeod::Euler_info, euler_sequence, gimbal_lock_threshold, jeod::EulerInfo::indices, jeod::EulerInfo::is_aerodynamics_sequence, jeod::EulerInfo::is_even_permutation, and trans.

7.2.4.8 compute_euler_angles_from_matrix() `[2/2]`

```
void jeod::Orientation::compute_euler_angles_from_matrix (
            void ) [inline], [protected]
```

Compute an Euler rotation sequence that corresponds to the provided transformation matrix.

Definition at line 330 of file orientation.hh.

References euler_angles, euler_sequence, and trans.

Referenced by compute_euler_angles().

7.2.4.9 compute_matrix_from_eigen_rotation() `[1/2]`

```
void jeod::Orientation::compute_matrix_from_eigen_rotation (
            double eigen_angle,
            const double eigen_axis[3],
            double trans[3][3] ) [static]
```

Compute the transformation matrix from the eigen rotation.

Given a rotation by an angle ϕ about an axis $\hat{\mathbf{u}}$, the [i][j] element of the transformation matrix is given by

$$T_{ij} = \cos\phi\,\delta_{ij} + (1-\cos\phi)\,\hat{u}_i\hat{u}_j + \epsilon_{ijk}\sin\phi\,\hat{u}_k$$

where

- δ_{ij} is the Kronecker delta,
- *k* is $(i+j) \bmod 3$, and
- ϵ_{ijk} is the Levi-Civita symbol taken with respect to (0,1,2).

Assumptions and Limitations

- The eigen axis is a unit vector.

Parameters

in	*eigen_angle*	Rotation angle Units: r
in	*eigen_axis*	Rotation axis, unit vector
out	*trans*	Resultant transformation matrix

Definition at line 82 of file eigen_rotation.cc.

References eigen_angle, eigen_axis, and trans.

7.2.4.10 compute_matrix_from_eigen_rotation() [2/2]

```
void jeod::Orientation::compute_matrix_from_eigen_rotation (
            void ) [inline], [protected]
```

Compute the transformation matrix that corresponds to the provided eigen rotation.

Definition at line 344 of file orientation.hh.

References eigen_angle, eigen_axis, and trans.

Referenced by compute_euler_angles(), and compute_transform().

7.2.4.11 compute_matrix_from_euler_angles() [1/2]

```
void jeod::Orientation::compute_matrix_from_euler_angles (
            EulerSequence euler_sequence,
            const double euler_angles[3],
            double trans[3][3] ) [static]
```

Compute the transformation matrix from the Euler sequence.

The matrix is formed by generating a sequence of three simple transformation matrices corresponding to the three rotations. The composite transformation matrix is the reverse-order product of these three simple matrices.

Parameters

in	*euler_sequence*	Euler sequence
in	*euler_angles*	Euler angles Units: r
out	*trans*	Resultant transformation matrix

Definition at line 158 of file euler_angles.cc.

References euler_angles, jeod::Euler_info, euler_sequence, jeod::EulerInfo::indices, and trans.

7.2.4.12 compute_matrix_from_euler_angles() [2/2]

```
void jeod::Orientation::compute_matrix_from_euler_angles (
            void ) [inline], [protected]
```

Compute the transformation matrix that corresponds to the provided Euler rotation sequence.

Definition at line 316 of file orientation.hh.

References euler_angles, euler_sequence, and trans.

Referenced by clear_euler_sequence(), compute_eigen_rotation(), and compute_transform().

7.2.4.13 compute_quaternion()

```
void jeod::Orientation::compute_quaternion (
            void ) [virtual]
```

Compute the left transformation quaternion from the source.

Definition at line 373 of file orientation.cc.

References compute_quaternion_from_euler_angles(), data_source, eigen_angle, eigen_axis, have_quaternion↩
_, InputEigenRotation, InputEulerRotation, InputMatrix, InputNone, InputQuaternion, mark_input_as_available(), quat, and trans.

Referenced by compute_all_products(), compute_transformation_and_quaternion(), and get_quaternion().

7.2.4.14 compute_quaternion_from_euler_angles() [1/2]

```
void jeod::Orientation::compute_quaternion_from_euler_angles (
            EulerSequence euler_sequence,
            const double euler_angles[3],
            Quaternion & quat ) [static]
```

Compute the left transformation quaternion from the Euler sequence.

The quaternion is formed by generating a sequence of three simple quaternions corresponding to the three rotations. The composite quaternion is the reverse-order product of these three simple quaternions.

Parameters

in	*euler_sequence*	Euler sequence
in	*euler_angles*	Euler angles Units: r
out	*quat*	Resultant quaternion

Definition at line 124 of file euler_angles.cc.

References euler_angles, jeod::Euler_info, euler_sequence, jeod::EulerInfo::indices, and quat.

7.2.4.15 compute_quaternion_from_euler_angles() [2/2]

```
void jeod::Orientation::compute_quaternion_from_euler_angles (
            void ) [inline], [protected]
```

Compute the left transformation quaternion that corresponds to the provided Euler rotation sequence.

Definition at line 302 of file orientation.hh.

References euler_angles, euler_sequence, and quat.

Referenced by clear_euler_sequence(), and compute_quaternion().

7.2.4.16 compute_transform()

```
void jeod::Orientation::compute_transform (
            void ) [virtual]
```

Compute the transformation matrix from the source.

Definition at line 321 of file orientation.cc.

References compute_matrix_from_eigen_rotation(), compute_matrix_from_euler_angles(), data_source, have_↩ transformation_, InputEigenRotation, InputEulerRotation, InputMatrix, InputNone, InputQuaternion, mark_input_↩ as_available(), quat, and trans.

Referenced by compute_all_products(), compute_transformation_and_quaternion(), and get_transform().

7.2.4.17 compute_transformation_and_quaternion()

```
void jeod::Orientation::compute_transformation_and_quaternion (
            void ) [virtual]
```

Compute the transformation matrix and quaternion.

Definition at line 569 of file orientation.cc.

References compute_quaternion(), and compute_transform().

7.2.4.18 get_eigen_rotation()

```
void jeod::Orientation::get_eigen_rotation (
            double * eigen_angle_out,
            double eigen_axis_out[3] )
```

Accessor for the eigen rotation.

Parameters

out	*eigen_angle_out*	Copy of the single axis rotation angle Units: r
out	*eigen_axis_out*	Copy of the single axis rotation axis

Definition at line 647 of file orientation.cc.

References compute_eigen_rotation(), eigen_angle, eigen_axis, and have_eigen_rotation_.

7.2.4.19 get_euler_angles() [1/2]

```
void jeod::Orientation::get_euler_angles (
            EulerSequence * sequence,
            double angles[3] )
```

Accessor for the Euler angles.

Parameters

out	*sequence*	Copy of the Euler sequence
out	*angles*	Copy of the Euler angles Units: r

Definition at line 673 of file orientation.cc.

References compute_euler_angles(), euler_angles, euler_sequence, and have_euler_angles_.

7.2.4.20 get_euler_angles() [2/2]

```
void jeod::Orientation::get_euler_angles (
            double angles[3] )
```

Accessor for the Euler angles.

Parameters

out	*angles*	Copy of the Euler angles Units: r

Definition at line 698 of file orientation.cc.

References compute_euler_angles(), euler_angles, and have_euler_angles_.

7.2.4.21 get_euler_sequence()

```
Orientation::EulerSequence jeod::Orientation::get_euler_sequence (
            void )
```

Accessor for the Euler sequence data member.

Returns

Euler sequence data member

Definition at line 721 of file orientation.cc.

References euler_sequence.

7.2.4.22 get_quaternion()

```
void jeod::Orientation::get_quaternion (
            Quaternion & quat_out )
```

Accessor for the left transformation quaternion.

Parameters

out	*quat_out*	Copy of the quaternion

Definition at line 623 of file orientation.cc.

References compute_quaternion(), have_quaternion_, and quat.

7.2.4.23 get_transform()

```
void jeod::Orientation::get_transform (
            double trans_out[3][3] )
```

Accessor for the transformation matrix.

Parameters

out	*trans_out*	Copy of the transformation matrix

Definition at line 600 of file orientation.cc.

References compute_transform(), have_transformation_, and trans.

7.2.4.24 mark_input_as_available()

```
void jeod::Orientation::mark_input_as_available (
            void ) [protected]
```

Mark the item specified by the data_source as available.

Note that this method doesn't compute any products.

Assumptions and Limitations

- The data_source member datum is valid.

Definition at line 246 of file orientation.cc.

References data_source, have_eigen_rotation_, have_euler_angles_, have_quaternion_, have_transformation_, InputEigenRotation, InputEulerRotation, InputMatrix, InputNone, InputQuaternion, and jeod::OrientationMessages::invalid_enum.

Referenced by compute_eigen_rotation(), compute_euler_angles(), compute_quaternion(), and compute_transform().

7.2.4.25 reset()

```
void jeod::Orientation::reset (
            void ) [virtual]
```

Forget that we have any data.

Note that this method does not reset the euler_sequence member; that is intentional.

Definition at line 227 of file orientation.cc.

References data_source, have_eigen_rotation_, have_euler_angles_, have_quaternion_, have_transformation_, and InputNone.

Referenced by set_eigen_rotation(), set_euler_angles(), set_quaternion(), and set_transform().

7.2.4.26 set_eigen_rotation()

```
void jeod::Orientation::set_eigen_rotation (
            double eigen_angle_in,
            const double eigen_axis_in[3] )
```

Reset the instance with a new eigen rotation.

Parameters

in	*eigen_angle_in*	New single axis rotation angle
in	*eigen_axis_in*	New single axis rotation axis

Definition at line 783 of file orientation.cc.

References data_source, eigen_angle, eigen_axis, have_eigen_rotation_, InputEigenRotation, and reset().

7.2.4.27 set_euler_angles() [1/2]

```
void jeod::Orientation::set_euler_angles (
            EulerSequence sequence,
            const double angles[3] )
```

Reset the instance with a new Euler rotation.

Parameters

in	*sequence*	New Euler sequence
in	*angles*	New Euler angles Units: r

Definition at line 804 of file orientation.cc.

References data_source, euler_angles, euler_sequence, EulerXYZ, EulerZYZ, have_euler_angles_, InputEulerRotation, jeod::OrientationMessages::invalid_enum, and reset().

7.2.4.28 set_euler_angles() [2/2]

```
void jeod::Orientation::set_euler_angles (
            const double angles[3] )
```

Reset the instance with a new Euler rotation.

Assumptions and Limitations

- The euler_sequence data member must have previously been set to a valid value.

Parameters

in	*angles*	New Euler angles Units: r

Definition at line 840 of file orientation.cc.

References data_source, euler_angles, euler_sequence, EulerXYZ, EulerZYZ, have_euler_angles_, InputEuler↩
Rotation, jeod::OrientationMessages::invalid_enum, and reset().

7.2.4.29 set_euler_sequence()

```
void jeod::Orientation::set_euler_sequence (
            EulerSequence sequence )
```

Set the euler_sequence data member.

Parameters

in	*sequence*	New Euler sequence

Definition at line 872 of file orientation.cc.

References clear_euler_sequence(), euler_sequence, EulerXYZ, EulerZYZ, have_euler_angles_, and jeod::↩
OrientationMessages::invalid_enum.

7.2.4.30 set_quaternion()

```
void jeod::Orientation::set_quaternion (
            const Quaternion & quat_in )
```

Reset the instance with a new quaternion.

Parameters

in	*quat↩ _in*	New quaternion

Definition at line 764 of file orientation.cc.

References data_source, have_quaternion_, InputQuaternion, quat, and reset().

7.2.4.31 set_transform()

```
void jeod::Orientation::set_transform (
            const double trans_in[3][3] )
```

Reset the instance with a new matrix.

Parameters

in	*trans_in*	New transformation matrix

Definition at line 746 of file orientation.cc.

References data_source, have_transformation_, InputMatrix, reset(), and trans.

7.2.5 Friends And Related Function Documentation

7.2.5.1 init_attrjeod__Orientation

```
void init_attrjeod__Orientation ( )  [friend]
```

7.2.5.2 InputProcessor

```
friend class InputProcessor  [friend]
```

Definition at line 83 of file orientation.hh.

7.2.6 Field Documentation

7.2.6.1 data_source

```
DataSource jeod::Orientation::data_source
```

Orientation data source – specifies whether the user has provided as input an Euler rotation, a transformation matrix, or a left transformation quaternion.

trick_units(–)

Definition at line 239 of file orientation.hh.

Referenced by clear_euler_sequence(), compute_eigen_rotation(), compute_euler_angles(), compute_quaternion(), compute_transform(), mark_input_as_available(), reset(), set_eigen_rotation(), set_euler_angles(), set_quaternion(), and set_transform().

7.2.6.2 eigen_angle

`double jeod::Orientation::eigen_angle`

Single axis rotation angle.

trick_units(rad)

Definition at line 265 of file orientation.hh.

Referenced by compute_eigen_rotation(), compute_eigen_rotation_from_matrix(), compute_matrix_from_eigen↩_rotation(), compute_quaternion(), get_eigen_rotation(), and set_eigen_rotation().

7.2.6.3 eigen_axis

`double jeod::Orientation::eigen_axis[3]`

Single axis rotation axis unit vector.

trick_units(–)

Definition at line 270 of file orientation.hh.

Referenced by compute_eigen_rotation(), compute_eigen_rotation_from_matrix(), compute_matrix_from_eigen↩_rotation(), compute_quaternion(), get_eigen_rotation(), Orientation(), and set_eigen_rotation().

7.2.6.4 euler_angles

`double jeod::Orientation::euler_angles[3]`

Euler angles corresponding to rotation sequence euler_sequence.

The elements are stored in the order specified by that sequence.trick_units(rad)

Definition at line 250 of file orientation.hh.

Referenced by compute_euler_angles_from_matrix(), compute_matrix_from_euler_angles(), compute_↩quaternion_from_euler_angles(), get_euler_angles(), Orientation(), and set_euler_angles().

7.2.6.5 euler_sequence

`EulerSequence jeod::Orientation::euler_sequence`

The Euler rotation sequence corresponding to euler_angles.

trick_units(–)

Definition at line 244 of file orientation.hh.

Referenced by clear_euler_sequence(), compute_euler_angles(), compute_euler_angles_from_matrix(), compute_matrix_from_euler_angles(), compute_quaternion_from_euler_angles(), get_euler_angles(), get_euler↩_sequence(), set_euler_angles(), and set_euler_sequence().

7.2.6.6 gimbal_lock_threshold

```
double jeod::Orientation::gimbal_lock_threshold = 1e-13 [static], [protected]
```

Threshold for detecting gimbal lock in compute_euler_angles_from_matrix.

The threshold for determining whether a gimbal lock condition exists.

trick_units(–)

Gimbal lock occurs when sin(theta) (aerodynamics Euler sequences) or cos(theta) (astronomical sequences) is very close to -1 or +1. This static variable quantifies the meaning of 'very close'.

Definition at line 144 of file orientation.hh.

Referenced by compute_euler_angles_from_matrix().

7.2.6.7 have_eigen_rotation_

```
bool jeod::Orientation::have_eigen_rotation_ [protected]
```

True if eigen rotation has been set/computed.

trick_units(–)

Definition at line 287 of file orientation.hh.

Referenced by compute_eigen_rotation(), get_eigen_rotation(), mark_input_as_available(), reset(), and set_↩ eigen_rotation().

7.2.6.8 have_euler_angles_

```
bool jeod::Orientation::have_euler_angles_ [protected]
```

True if an Euler rotation has been set/computed.

trick_units(–)

Definition at line 292 of file orientation.hh.

Referenced by clear_euler_sequence(), compute_euler_angles(), get_euler_angles(), mark_input_as_available(), reset(), set_euler_angles(), and set_euler_sequence().

7.2.6.9 have_quaternion_

```
bool jeod::Orientation::have_quaternion_ [protected]
```

True if quaternion has been set/computed.

trick_units(–)

Definition at line 282 of file orientation.hh.

Referenced by clear_euler_sequence(), compute_quaternion(), get_quaternion(), mark_input_as_available(), reset(), and set_quaternion().

7.2.6.10 have_transformation_

```
bool jeod::Orientation::have_transformation_ [protected]
```

True if transformation matrix has been set/computed.

trick_units(–)

Definition at line 277 of file orientation.hh.

Referenced by clear_euler_sequence(), compute_eigen_rotation(), compute_euler_angles(), compute_transform(), get_transform(), mark_input_as_available(), reset(), and set_transform().

7.2.6.11 quat

```
Quaternion jeod::Orientation::quat
```

Left transformation unit quaternion.

trick_units(–)

Definition at line 260 of file orientation.hh.

Referenced by compute_eigen_rotation(), compute_euler_angles(), compute_quaternion(), compute_quaternion_from_euler_angles(), compute_transform(), get_quaternion(), and set_quaternion().

7.2.6.12 trans

```
double jeod::Orientation::trans[3][3]
```

Transformation matrix.

trick_units(–)

Definition at line 255 of file orientation.hh.

Referenced by compute_eigen_rotation_from_matrix(), compute_euler_angles(), compute_euler_angles_from_matrix(), compute_matrix_from_eigen_rotation(), compute_matrix_from_euler_angles(), compute_quaternion(), compute_transform(), get_transform(), Orientation(), and set_transform().

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

- orientation.hh
- eigen_rotation.cc
- euler_angles.cc
- orientation.cc

7.3 jeod::OrientationMessages Class Reference

Declares messages associated with the orientation model.

```
#include <orientation_messages.hh>
```

Static Public Attributes

- static char const ∗ invalid_enum = "utils/orientation/" "invalid_enum"

  Issued when a enum value is not one of the enumerated values.
- static char const ∗ invalid_data = "utils/orientation/" "invalid_data"

  Issued when an orientation specification is invalid.
- static char const ∗ invalid_request = "utils/orientation/" "invalid_request"

  Issued when an requested is invalid.

Private Member Functions

- OrientationMessages (void)
- OrientationMessages (const OrientationMessages &)
- OrientationMessages & operator= (const OrientationMessages &)

Friends

- class InputProcessor
- void init_attrjeod__OrientationMessages ()

7.3.1 Detailed Description

Declares messages associated with the orientation model.

Definition at line 83 of file orientation_messages.hh.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 OrientationMessages() [1/2]

```
jeod::OrientationMessages::OrientationMessages (
            void ) [private]
```

7.3.2.2 OrientationMessages() [2/2]

```
jeod::OrientationMessages::OrientationMessages (
            const OrientationMessages & ) [private]
```

7.3.3 Member Function Documentation

7.3.3.1 operator=()

```
OrientationMessages& jeod::OrientationMessages::operator= (
            const OrientationMessages & ) [private]
```

7.3.4 Friends And Related Function Documentation

7.3.4.1 init_attrjeod__OrientationMessages

```
void init_attrjeod__OrientationMessages ( ) [friend]
```

7.3.4.2 InputProcessor

```
friend class InputProcessor  [friend]
```

Definition at line 86 of file orientation_messages.hh.

7.3.5 Field Documentation

7.3.5.1 invalid_data

```
char const * jeod::OrientationMessages::invalid_data = "utils/orientation/" "invalid_data"
[static]
```

Issued when an orientation specification is invalid.

trick_units(–)

Definition at line 100 of file orientation_messages.hh.

7.3.5.2 invalid_enum

```
char const * jeod::OrientationMessages::invalid_enum = "utils/orientation/" "invalid_enum"
[static]
```

Issued when a enum value is not one of the enumerated values.

trick_units(–)

Definition at line 95 of file orientation_messages.hh.

Referenced by jeod::Orientation::compute_euler_angles(), jeod::Orientation::mark_input_as_available(), jeod::↩
Orientation::set_euler_angles(), and jeod::Orientation::set_euler_sequence().

7.3.5.3 invalid_request

```
char const * jeod::OrientationMessages::invalid_request = "utils/orientation/" "invalid_↩
request"  [static]
```

Issued when an requested is invalid.

trick_units(–)

Definition at line 105 of file orientation_messages.hh.

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

- orientation_messages.hh
- orientation_messages.cc

Chapter 8

File Documentation

8.1 eigen_rotation.cc File Reference

Define Orientation methods related to computing single axis rotations.

```
#include <cmath>
#include "utils/math/include/matrix3x3.hh"
#include "utils/math/include/vector3.hh"
#include "../include/orientation.hh"
```

Namespaces

- jeod

    Namespace jeod.

8.1.1 Detailed Description

Define Orientation methods related to computing single axis rotations.

8.2 euler_angles.cc File Reference

Define Orientation methods related to computing Euler angles.

```
#include <cmath>
#include "utils/math/include/matrix3x3.hh"
#include "../include/orientation.hh"
```

Data Structures

- struct jeod::EulerInfo

    Contains data needed to construct a transformation matrix given a sequence of Euler angles and to extract a sequence of Euler angles from a matrix.

Namespaces

- jeod

  Namespace jeod.

Variables

- static const EulerInfo jeod::Euler_info [12]

  Contains twelve EulerInfo objects, one per each of the JEOD Euler sequences.

8.2.1 Detailed Description

Define Orientation methods related to computing Euler angles.

8.3 orientation.cc File Reference

Define methods for the NewOrientation class.

```
#include <cmath>
#include "utils/math/include/matrix3x3.hh"
#include "utils/math/include/vector3.hh"
#include "utils/message/include/message_handler.hh"
#include "../include/orientation.hh"
#include "../include/orientation_messages.hh"
```

Namespaces

- jeod

  Namespace jeod.

8.3.1 Detailed Description

Define methods for the NewOrientation class.

8.4 orientation.hh File Reference

Define the Orientation class.

```
#include "utils/sim_interface/include/jeod_class.hh"
#include "utils/quaternion/include/quat.hh"
```

Data Structures

- class jeod::Orientation

  Specifies the orientiation of one reference frame with respect to another.

Namespaces

- jeod

  Namespace jeod.

8.4.1 Detailed Description

Define the Orientation class.

8.5 orientation_messages.cc File Reference

Implement the class OrientationMessages.

```
#include "utils/message/include/make_message_code.hh"
#include "../include/orientation_messages.hh"
```

Namespaces

- jeod

  Namespace jeod.

Macros

- #define MAKE_ORIENTATION_MESSAGE_CODE(id) JEOD_MAKE_MESSAGE_CODE(Orientation↩
  Messages, "utils/orientation/", id)

8.5.1 Detailed Description

Implement the class OrientationMessages.

8.5.2 Macro Definition Documentation

8.5.2.1 MAKE_ORIENTATION_MESSAGE_CODE

```
#define MAKE_ORIENTATION_MESSAGE_CODE(
              id ) JEOD_MAKE_MESSAGE_CODE(OrientationMessages, "utils/orientation/", id)
```

Definition at line 38 of file orientation_messages.cc.

8.6 orientation_messages.hh File Reference

Define the class OrientationMessages, the class that specifies the message IDs used in the orientation model.

```
#include "utils/sim_interface/include/jeod_class.hh"
```

Data Structures

- class jeod::OrientationMessages

  Declares messages associated with the orientation model.

Namespaces

- jeod

  Namespace jeod.

8.6.1 Detailed Description

Define the class OrientationMessages, the class that specifies the message IDs used in the orientation model.

Index