User Manual 4.14 Rotation, AngularCoordinates, Tranform and Attitude : how to use them

Introduction

In PATRIUS, several objects allow the user to represent rotations and perform some frame transformations, from simplest to most complex:

- `Rotation`: this class defines a rotation. No frame and date is associated.
- `AngularCoordinates`: this class defines a rotation, a rotation rate and optionally a rotation acceleration. No frame and date is associated.
- `Transform`: this class defines angular coordinates and PV coordinates at a given date.
- `Attitude`: this class defines angular coordinates at a given date with respect to a reference frame.

Rotation

Definition

A rotation is represented by the object `Rotation` (see Javadoc).

A rotation is a transformation that transforms a reference frame (in black) into a frame of interest (in blue). The result is expressed in the reference frame.

The formula giving the transformation are the following:

[math]\vec{X_{int / ref}} = R(\vec{X_{ref / ref}})[/math] [math]\vec{Y_{int / ref}} = R(\vec{Y_{ref / ref}})[/math] [math]\vec{Z_{int / ref}} = R(\vec{Z_{ref / ref}})[/math]

The rotation object is not linked to any frame, so remember, the rotation is can be applied to any set of coordinates as long as it is consistent with how you defined your rotation.

Features

The `Rotation` objects can be defined by several manners:

- Using quaternions (used as internal representation), see [MAT_ROT_Home Quaternions]
- Using a 3x3 rotation matrix

- Using an axis and an angle
- Using Euler angles
- etc.

The `Rotation` offers several features:

- Rotate a vector expressed in the reference frame to the frame of interest (= express this vector in the frame of interest): `Rotation.applyInverseTo(Vector3D)`
- Rotate a vector expressed in the frame of interest to the reference frame (= express this vector in the reference frame): `Rotation.applyTo(Vector3D)`
- Compose two rotation: `Rotation.applyInverseTo(Rotation)`. R2.applyInverseTo(R1) returns R2^^-1^^ o R1
- Compose two rotation: `Rotation.applyTo(Rotation)`. R2.applyTo(R1) returns R2 o R1
- Get Euler angles (given an Euler sequence): `Rotation.getAngles()`
- Get rotation matrix associated to rotation: `Rotation.getMatrix()`
- etc.

Code examples

Let's consider:

- A frame F1 [x1, y1, z1]
- A frame F2 [x2, y2, z2] with x2 = y1, y2 =-x1, z2 = z1

Then F2 is obtained by a rotation of 90° around z1 of F1

The following code defines the frames F1 and F2 and performs some transformations:

```
// Define the rotation transforming F1 into F2: the axis is [0, 0, 1] and the
angle is (Pi / 2)
final Rotation rotation = new Rotation(Vector3D.PLUS_K, FastMath.PI / 2.);
// Defines a vector v1 in F1
final Vector3D v1 = new Vector3D(1., 1., 1.);
// Express v1 in F2 = v2 = [1,-1, 1]
final Vector3D v2 = rotation.applyInverseTo(v1);
// Express v2 in F1 = v3. v3 is then equal to v1
final Vector3D v3 = rotation.applyTo(v2);
// Retrieve axis of rotation [0, 0, 1] and angle
final Vector3D axis = rotation.getAxis();
final double angle = rotation.getAngle();
```

Other methods are used in a similar fashion.

Angular coordinates

Definition

Angular coordinates are represented by the object `AngularCoordinates` (see Javadoc). Angular coordinates contain:

- A rotation defined with a `Rotation` object (see above)

- A rotation rate defined with a `Vector3D` object.
- A rotation rate derivative (or rotation acceleration) defined with a `Vector3D` object.

The convention used to describe the orientation of the frame of interest in `AngularCoordinates` is the same used in `Rotation`: the rotation represents the transformation from a reference frame to a frame of interest, expressed in reference frame. However `AngularCoordinates.applyTo()` returns the opposite of `Rotation.applyTo()`.

Orientation
The orientation is described by a rotation (see above).

Rotation Rate
The Rotation Rate is a 3D vector expressed**in the frame of interest**. Its norm is the angular velocity of the frame of interest. Its direction is the instant axis of spin. Here is the case of a spin around the Z axis :

Rotation Acceleration
The Rotation Acceleration is a 3D vector expressed**in the frame of interest**. Its norm is the angular acceleration of the frame of interest. As the rotation rate, its direction is the instant axis of spin (see image above).

This computation is optional. All methods are doubled:

- One provides computation without rotation rate derivative.
- One provides computation with or without rotation rate derivative (a boolean is provided):

```
AngularCoordinates(final PVCoordinates u1, final PVCoordinates u2,
                   final PVCoordinates v1, final PVCoordinates v2,
                   final double tolerance, final boolean
spinDerivativesComputation)
```

Don't compute spin derivative if you don't need to since it is time consuming.

Features

The `AngularCoordinates` class offers several features:

- Rotate a vector expressed in the reference frame to the frame of interest (= express this vector in the frame of interest): `AngularCoordinates.applyTo()`. Warning: this is opposite from `Rotation.applyTo()`!
- Given a time shift, shift internal rotation using rotation rate and derivative if provided: `AngularCoordinates.shiftedBy()`
- Internal rotation as a `Rotation` object can be retrieved and methods shown in the Rotation section are available. Retrieved rotation has the same convention as Rotation class.
- etc.

Code examples

Here is an examples of how to time-shift a rotation using the rotation rate. This example involves different frames and helps understand them.

```
// Get the content of angular coordinates
// Rotation represent a transformation from a reference frame F1 to a frame
of interest F2, expressed in F1
// Rotation acceleration and rotation rate are expressed in frame of interest
F2
Vector3D rotation_acceleration = angularCoordinates.getRotationAcceleration()
Vector3D rotation_rate = angularCoordinates.getRotationRate();
Rotation orientation = angularCoordinates.getRotation();

// To compose two rotations, they must be expressed in the same frame (here
F1).
// The orientation is expressed in the reference frame F1, so the shift
(evolution of the rotation) has to be expressed in the reference frame F1
too.
// To create it, the rotation rate has itself to be expressed in the
reference frame F1.

// Get rotation rate which is expressed in F2 in reference frame F1
Vector3D rotation_rate_in_ref_frame = orientation.applyTo(rotation_rate);

// The rotation shift can then be created ("dt" is the time duration of the
shift). This rotation shift is expressed in F1
Rotation shift = new Rotation(rotation_rate_in_ref_frame,
rotation_rate_in_ref_frame.getNorm()* dt);

// The time-shifted orientation can finally be computed: this rotation
represents the transformation from the reference frame F1 to a frame of
interest F3 which is shifted from F2.
Rotation finalRotation = shift.applyTo(orientation);
```

For information, the shifted rotation can be directly retrieved using the method `shyftedBy()` of `AngularCoordinates`.

```
AngularCoordinates shifted = angularCoordinates.shiftedBy(dt);
Rotation shiftedRotation = shifted.getRotation();
```

Transform

Definition

Transform are represented by the object `Transform` (see Javadoc). A Transform object contains:

- A date of the transformation
- A position, velocity, acceleration (optional)
- A rotation, rotation rate, rotation rate derivative (optional)

of a "destination" frame (or frame of interest) in an "origin" frame (or reference frame).

The convention used to describe the orientation and rotation rate of the "destination" frame in Transform is the same as in Rotation and AngularCoordinates : the rotation returned by `Transform.getRotation()` is the one that transforms the basis vectors of the "origin" frame into the ones of the "destination" frame.

The position and velocity of the "destination" are expressed in the "origin" frame.

Features

The `Transform` class offers several features:

- Transform a vector expressed in origin frame in destination frame: `Transform.transformVector()`
- Transform a PVCoordinates expressed in origin frame in destination frame: `Transform.transformPVCoordinates()`
- Retrieve jacobian of transformation: `Transform.getJacobian()`
- Internal rotation as a `Rotation` object (as well as rotation rate, etc.) can be retrieved and methods shown in the Rotation section are available. Retrieved rotation has the same convention as Rotation class. Other attributes (rotation rate, etc.) have the same convention as AngularCoordinates class.
- etc.

Code examples

In the following example, the "origin" is a TNW local orbital frame and the "destination" is the GCRF inertial frame.

```
// Build transform from TNW local orbital frame to GCRF
final Frame lof = new LocalOrbitalFrame(FramesFactory.getGCRF(), LOFType.TNW,
orbit, "TNW");
final Frame gcrf = FramesFactory.getGCRF();
final Transform transform = lof.getTransformTo(gcrf,
AbsoluteDate.J2000_EPOCH);
// Define a vector v = [1, 0, 0] in TNW frame
final Vector3D v_TNW = Vector3D.PLUS_I;
// Get v in GCRF frame
final Vector3D v_GCRF = transform.transformVector(v_TNW);
// Get v back in TNW
final Vector3D v_TNW2 = transform.getInverse().transformVector(v_GCRF);
```

Attitude

Definition

An attitude object contains all information about the satellite's orientation at a date.

Attitudes are represented by the object `Attitude` (see Javadoc). Attitude contain:

- Angular coordinates.
- A date.
- A reference frame.

Attitude object contains information to rotate the reference frame to a frame of interest (often the satellite frame), expressed in the reference frame. For example, attitude laws via the method getAttitude(Frame, etc.) will return the Attitude in provided frame, the attitude being the transformation from provided frame to satellite platform frame).

The convention used to describe the orientation of the frame of interest in `Attitude` is the same used in `AngularCoordinates`: the rotation represents the transformation from a reference frame to a frame of interest, expressed in reference frame.

Features

The `Attitude` class offers several features:

- Change the attitude reference frame: `Attitude.withReferenceFrame()`
- Internal rotation as a `Rotation` object (as well as rotation rate, etc.) can be retrieved and methods shown in the Rotation section are available. Retrieved rotation has the same convention as Rotation class. Other attributes (rotation rate, etc.) have the same convention as AngularCoordinates class.
- Perform interpolation: `Attitude.interpolate()`
- etc.

Code examples

This example creates an attitude from a reference frame GCRF to the satellite frame, change the reference frame to CIRF and then rotate a vector.

```
        // Define attitude from BodyCenterPointing attitude law, expressed in
GCRF frame
        final AttitudeLaw attitudeLaw = new BodyCenterPointing();
        final Frame gcrf = FramesFactory.getGCRF();
        final Attitude attitude_GCRF = attitudeLaw.getAttitude(orbit,
AbsoluteDate.J2000_EPOCH, gcrf);
        // Change attitude reference frame to CIRF
        final Frame cirf = FramesFactory.getCIRF();
        final Attitude attitude_CIRF =
attitude_GCRF.withReferenceFrame(cirf);

        // Define a vector v = [1, 0, 0] in satellite frame
        final Vector3D v_Sat = Vector3D.PLUS_I;
```

```
        // Get attitude rotation (from reference frame = CIRF to frame of
interest = satellite frame)
        final Rotation rotation = attitude_CIRF.getRotation();
        // Get v in CIRF frame
        final Vector3D v_CIRF = rotation.applyInverseTo(v_Sat);
```

More code examples

Here is an example of code that may help understanding the use of rotations in attitudes and frames transformations.

```
// GCRF, reference frame
final Frame gcrf = FramesFactory.getGCRF();

// Axis of the GCRF frame, expressed in GCRF
final Vector3D xGCRF_inGCRF = Vector3D.PLUS_I;
final Vector3D yGCRF_inGCRF = Vector3D.PLUS_J;
final Vector3D zGCRF_inGCRF = Vector3D.PLUS_K;

// Directions associated to those axis
final IDirection xGCRF = new ConstantVectorDirection(xGCRF_inGCRF, gcrf);
final IDirection yGCRF = new ConstantVectorDirection(yGCRF_inGCRF, gcrf);
final IDirection zGCRF = new ConstantVectorDirection(zGCRF_inGCRF, gcrf);

// Creation of a "zero" PV provider
final PVCoordinatesProvider pvProv = new PVCoordinatesProvider() {
public PVCoordinates getPVCoordinates(AbsoluteDate date, Frame frame)
throws PatriusException {
return new PVCoordinates(Vector3D.ZERO, Vector3D.ZERO);
}
};

// a date...
final AbsoluteDate date = new AbsoluteDate(2014, 10, 2, 11, 46, 00,
TimeScalesFactory.getTAI());

// Axis of the satellite frame, expressed in the satellite frame
final Vector3D xSat_inRSat = Vector3D.PLUS_I;
final Vector3D ySat_inRSat = Vector3D.PLUS_J;
final Vector3D zSat_inRSat = Vector3D.PLUS_K;

// The attitude law :
//- axe Xsat on Y_GCRF,
//- axe Ysat "as close as possible" to Z_GCRF.
// We are implicitly defining the orientation of the satellite frame.
final TwoDirectionAttitudeLaw attLaw = new TwoDirectionAttitudeLaw(yGCRF,
zGCRF, xSat_inRSat, ySat_inRSat);

// Attitude computation,
```

```
// and getting of the associated rotation.
Attitude att = attLaw.getAttitude(pvProv, date, gcrf);
Rotation rot = att.getRotation ();

// Computation of the axis of the satellite frame in the GCRF
Vector3D xSat_inGCRF = rot.applyTo(xGCRF_inGCRF);
Vector3D ySat_inGCRF = rot.applyTo(yGCRF_inGCRF);
Vector3D zSat_inGCRF = rot.applyTo(zGCRF_inGCRF);

// Print
System.out.println("xSat_inGCRF: " + xSat_inGCRF);
System.out.println("ySat_inGCRF: " + ySat_inGCRF);
System.out.println("zSat_inGCRF: " + zSat_inGCRF);

// getting the attitude quaternion
Quaternion q = rot.getQuaternion ();
System.out.println("quaternion: " + q);

// Printing the axis and angle of the rotation,
// to visualize its "right" definition.
System.out.println(" axis: " + rot.getAxis());
System.out.println(" angle: " + rot.getAngle());

// Creation of the satellite frame
AttitudeFrame attFrame = new AttitudeFrame(pvProv, attLaw, gcrf);

// getting the transformation from the satellite frame to the GCRF
Transform transform = attFrame.getTransformTo(gcrf, date);

// Changing the expression frame of Xsat from Rsat into GCRF
// to match the previous result :
System.out.println("xSat_inGCRF: " + xSat_inGCRF);
System.out.println("transform.transformVector(xSat_inRSat): " +
transform.transformVector(xSat_inRSat));
```


Menu de navigation

Outils personnels

- Se connecter

Espaces de noms

- Page
- Discussion

Variantes

Affichages

- Lire
- Voir le texte source
- Historique
- Exporter en PDF

Plus

Rechercher

Rechercher Lire

PATRIUS

- Welcome

Evolutions

- Main differences between V4.14 and V4.13
- Main differences between V4.13 and V4.12
- Main differences between V4.12 and V4.11
- Main differences between V4.11 and V4.10
- Main differences between V4.10 and V4.9
- Main differences between V4.9 and V4.8
- Main differences between V4.8 and V4.7
- Main differences between V4.7 and V4.6.1
- Main differences between V4.6.1 and V4.5.1
- Main differences between V4.5.1 and V4.4
- Main differences between V4.4 and V4.3
- Main differences between V4.3 and V4.2
- Main differences between V4.2 and V4.1.1
- Main differences between V4.1.1 and V4.1

- Main differences between V4.1 and V4.0
- Main differences between V4.0 and V3.4.1

User Manual

- User Manual 4.14
- User Manual 4.13
- User Manual 4.12
- User Manual 4.11
- User Manual 4.10
- User Manual 4.9
- User Manual 4.8
- User Manual 4.7
- User Manual 4.6.1
- User Manual 4.5.1
- User Manual 4.4
- User Manual 4.3
- User Manual 4.2
- User Manual 4.1
- User Manual 4.0
- User Manual 3.4.1
- User Manual 3.3

Tutorials

- Tutorials 4.14
- Tutorials 4.13.5
- Tutorials 4.12.1
- Tutorials 4.8.1
- Tutorials 4.5.1
- Tutorials 4.4
- Tutorials 4.1
- Tutorials 4.0

Links

- CNES freeware server

Navigation

Outils

- Pages liées
- Suivi des pages liées
- Pages spéciales
- Adresse de cette version
- Information sur la page
- Citer cette page

- Dernière modification de cette page le 5 septembre 2024 à 14:45.

- Politique de confidentialité
- À propos de Wiki
- Avertissements

-