User Manual 4.13 Maneuvers

User Manual 4.13 Maneuvers

Sommaire

Introduction

Scope

The scope of this section is to present the maneuver models available through the Patrius libraries.

Javadoc

All the classes related to maneuvers are in the following packages:

Library	Javadoc
Patrius	Package fr.cnes.sirius.patrius.forces.maneuvers

Links

None as of now.

Useful Documents

None as of now.

Package Overview

The maneuver package `fr.cnes.sirius.patrius.forces.maneuvers` contains:

- Continuous maneuvers: both constant and variable)
- Impulsive maneuvers
- Maneuvers sequences
- An analytical model for small maneuvers
- Impulsive maneuvers based on orbital increments (da, de, di).

Features Description

Continuous Thrust Maneuver

Continuous thrust maneuvers are defined using the class `ContinuousThrustManeuver`. With it, you can either define:

- A constant maneuver
- A variable maneuver

A maneuver contains the following parameters:

- A start date / end date: this can be defined either:
  - With a start date and a duration (former method)
  - With event detectors, first detector for starting the maneuver, second detector for stopping the maneuver. **Action.STOP** is required to trigger (start/stop) the thrust. Any other action will be discarded.

When using event detectors, pay attention to events that may occur several times. If you want the maneuver to perform only once, the event detectors have to be included in a nth occurrence detector or its `shouldBeRemoved()` method have to return `true`. Otherwise the maneuver may be performed more than once: if a maneuver starts at the perigee (using a perigee detector) and stops at the apogee (using an apogee detector), then a maneuver will start at every perigee and stop at every apogee.

- A frame in which the thrust direction is provided. This can either be:
  - A generic `frame`
  - A local orbital frame `LOFType`
  - The spacecraft frame (in this case, attitude is mandatory to compute transform from spacecraft frame from propagation frame)
- A `PropulsiveProperty` which contains:
  - Thrust: it can be either constant (using `double`) or variable (using `IDependentVariable<SpacecraftState>`). If defined as constant, thrust is then defined as a `Parameter` and derivative with respect to thrust can be computed
  - ISP: it can be either constant (using `double`) or variable (using `IDependentVariable<SpacecraftState>`)
- A `MassProvider` used to compute total mass of spacecraft
- A `TankProperty` indicating the tank to use for the maneuver

If your propagation starts within a maneuver, two methods are available to manually start/stop a

maneuver :

- `isFiring()`
- `setFiring(final boolean isFiring)`

A user wanting to start a propagation during a maneuver will call `setFiring(true)` before launching the propagation. This is useful since maneuver start and stop criteria are defined through event detectors. This mecanism is available for any non-impulsive maneuver i.e. `ContinuousThrustManeuver` and `ConstantThrustError`.

The construction possibilities are listed in the following array:

Parameter type	Number of possibilities	Detail
**Thrust start/end criteria	2	Date + thrust duration or event detectors for start/end.
**ISP, Thrust	1	Propulsive property (can be constant or variable).
**Thrust direction	2	Constant (Vector3D) or variable (IDependentVectorVariable<SpacecraftState>).
**Thrust direction frame	3	Satellite frame, local orbital frame or Frame object.
**Fuel tank	1	TankProperty.

All these possibilities lead to 12 constructors.

This class also provides a method to retrieve current DV used: `getDeltaVSat`. The total used Delta-V is updated after each integration step.

Below are examples with constant and variable thrust maneuvers.

Constant Thrust Maneuver

In this case, the thrust, the ISP and the acceleration direction are constant values.

Here is an example of a constant maneuver of direction [1, 0, 0] in TNW frame.

```
final double thrust = 420;
final double isp = 318;
final Vector3D direction = Vector3D.PLUS_I;
final PropulsiveProperty propulsiveProperty = new PropulsiveProperty(thrust,
isp);
final TankProperty tankProperty = new TankProperty(1000.);
final ContinuousThrustManeuver maneuver = new ContinuousThrustManeuver(date,
duration, propulsiveProperty, direction, massProvider, tankProperty,
LOFType.TNW);
```

Partial derivatives with respect to thrust can be computed. To do so, you either:

- Define your own thrust as a parameter:

```
final double thrustParam = new Parameter("Thrust", 420);
```

```
final double ispParam = new Parameter("ISP", 318);
final PropulsiveProperty propulsiveProperty = new
PropulsiveProperty(thrustParam, ispParam);
```

And then call `maneuver.addDAccDParam(..., thrustParam, ...)`

- Or retrieve the thrust parameter:

```
final double thrust = 420;
final double isp = 318;
final PropulsiveProperty propulsiveProperty = new PropulsiveProperty(thrust,
isp);
finl Parameter thrustParam = propulsiveProperty.getThrustParam();
```

And then call `maneuver.addDAccDParam(..., thrustParam, ...)`

Variable Thrust Maneuver

In this case, the thrust, the ISP and the acceleration direction can be variable values.

Here is an example of a variable maneuver of variable direction in TNW frame and variable thrust and ISP.

```
final IDependentVariable<SpacecraftState> thrust = ...;
final IDependentVariable<SpacecraftState> isp = ...;
final IDependentVectorVariable<SpacecraftState> direction = ...;
final PropulsiveProperty propulsiveProperty  = new PropulsiveProperty(thrust,
isp);
final TankProperty tankProperty = new TankProperty(1000.);
final ContinuousThrustManeuver maneuver = new ContinuousThrustManeuver(date,
duration, propulsiveProperty, direction, massProvider, tankProperty,
LOFType.TNW);
```

No implementation of `IDependentVariable<SpacecraftState>` and `IDependentVectorVariable<SpacecraftState>` are provided in Patrius. You can however easily create your own.

Remember that partial derivatives with respect to thrust cannot be computed using variable maneuvers.

Impulse Maneuver

This `ImpulseManeuver` class implements the `EventDetector` interface.
This maneuver is provided with an underlying event detector; when an underlying event is triggered, and only if its action is set to STOP, the maneuver is triggered and the current `SpacecraftState` is reset. An increment or a decrement is applied to its velocity and a ratio is applied to the mass. These changes depend on the direction of variation of the integration variable (time) during integration : propagation or retro-propagation case.

This class also provides a method to retrieve current DV used: `getDeltaVSat`. In the impulse

maneuver case, the DV is either 0 or the DV provided by the user once the maneuver has been performed.

List of maneuvers

The class `ManeuversSequence` handles the creation and the operations on a list of maneuvers; the maneuvers that can be added to the list are the following ones :

- Impulse maneuvers;
- Continue constant maneuvers;
- Continue maneuvers with variable thrust and ISP;

The user can do the following operations on the list:

- add a new maneuver; to add it, the following conditions must be met:
  - there must be no superposition between the new maneuver and any maneuver in the list.
    If the new maneuver is an impulse one and its triggering event detector is not a date detector, it always adds the maneuver (there is no way to know in advance its date). Otherwise if the new maneuver is an impulse one and its triggering event detector is a date detector, it gets the date of the maneuver from the triggering detector and uses this information to check the superposition;
  - the time between the end of the previous maneuver and the start of the new maneuver, as well as the time between the end of the new maneuver and the start of the next maneuver, must be bigger than a threshold value. The threshold value depends on the type of maneuver (impulse or continue) and can be chosen by the user.
- remove a maneuver already in the list;
- get the number of maneuvers in the list;
- add all the maneuvers of the list to the propagator;

The following lines show the usage of the class:

```
final ManeuversSequence sequence = new ManeuversSequence(0.0, 0.0);
// add the maneuvers to the sequence:
sequence.add(constantManeuver);
sequence.add(impulseManeuver);
// apply the maneuvers to the propagator:
sequence.applyTo(numericalPropagator);
```

Impulse maneuvers based on orbital increments

Three class are currently available:

- `ImpulseDaManeuver`: semi-major axis change
- `ImpulseDeManeuver`: semi-major axis as well as eccentricity change
- `ImpulseDiManeuver`: semi-major axis as well as inclination change

These classes simply modify the orbital parameters according to user provided data (da, de or di). The DV and used mass are computed accordingly. These maneuvers inherit `ImpulseManeuver`,

hence should be used in the same way with propagators.

Getting Started

Modèle:SpecialInclusion prefix=$theme sub section="GettingStarted"/

Contents

Interfaces

Classes

Class	Summary	Javadoc
ContinuousThrustManeuver	This class implements a continuous thrust maneuver.	...
ImpulseManeuver	This class implements an impulse maneuver model.	...
SmallManeuverAnalyticalModel	This class implements an analytical model for small maneuvers.	...
ManeuversSequence	This class implements a sequence of maneuvers.	...
ImpulseDaManeuver	This class implements an impulse maneuver modifying the orbit semi-major axis.	...
ImpulseDeManeuver	This class implements an impulse maneuver modifying the orbit semi-major axis and eccentricity.	...
ImpulseDiManeuver	This class implements an impulse maneuver modifying the orbit semi-major axis and inclination.	...

Catégorie :

- User Manual 4.13 Mission

Menu de navigation

Outils personnels

- 3.145.105.149
- Discussion avec cette adresse IP
- Créer un compte
- 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 V4.15 and V4.14
- 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 4.15
- 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 4.15
- 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

- Accueil
- Modifications récentes
- Page au hasard
- Aide

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 25 juillet 2024 à 10:00.

- Politique de confidentialité
- À propos de Wiki
- Avertissements
-