Module: smallBodyWaypointFeedback

Executive Summary

This module is provides a feedback control law for waypoint-to-waypoint control about a small body. The waypoints are defined in the Hill frame of the body.

Message Connection Descriptions

The following table lists all the module input and output messages. The module msg connection is set by the user from python. The msg type contains a link to the message structure definition, while the description provides information on what this message is used for.

Module I/O Messages

Msg Variable Name	Msg Type	Description
navTransInMsg	**NavTransMsgPayload**	translational navigation input message
navAttInMsg	**NavAttMsgPayload**	attitude navigation input message
asteroidEphemerisInMsg	**EphemerisMsgPayload**	asteroid ephemeris input message
sunEphemerisInMsg	**EphemerisMsgPayload**	sun ephemeris input message
forceOutMsg	**CmdForceBodyMsgPayload**	force command output
forceOutMsgC	**CmdForceBodyMsgPayload**	C-wrapped force output message

Detailed Module Description

General Function

The `smallBodyWaypointFeedback()` module provides a solution for waypoint-to-waypoint control about a small body. The feedback control law is similar to the cartesian coordinate continuous feedback control law in Chapter 14 of **Analytical Mechanics of Space**

Systems. A cannonball SRP model, third body perturbations from the sun, and point-mass gravity are utilized. The state inputs are messages written out by **Module: simpleNav** and **Module: planetNav** modules or an estimator that provides the same input messages.

Algorithm

The state vector is defined as follows:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} = \begin{bmatrix} {}^O\mathbf{r}_{B/O} \\ {}^O\dot{\mathbf{r}}_{B/O} \end{bmatrix} \tag{1}$$

The associated frame definitions may be found in the following table.

Frame Definitions

Frame Description	Frame Definition
Small Body Hill Frame	$O : \{\hat{\mathbf{o}}_1, \hat{\mathbf{o}}_2, \hat{\mathbf{o}}_3\}$
Spacecraft Body Frame	$B : \{\hat{\mathbf{b}}_1, \hat{\mathbf{b}}_2, \hat{\mathbf{b}}_3\}$

The derivation of the control law is skipped here for brevity. The thrust, however, is computed as follows:

$$\mathbf{u} = -(f(\mathbf{x}) - f(\mathbf{x}_{ref})) - [K_1]\Delta\mathbf{x}_1 - [K_2]\Delta\mathbf{x}_2 \tag{2}$$

The relative velocity dynamics are described in detail by **Takahashi** and **Scheeres**.

$$\begin{aligned} f(\mathbf{x}) ={}^O \ddot{\mathbf{r}}_{S/O} &= -\ddot{F}[\tilde{\hat{\mathbf{o}}}_3]\mathbf{x}_1 - 2\dot{F}[\tilde{\hat{\mathbf{o}}}_3]\mathbf{x}_2 - \dot{F}^2[\tilde{\hat{\mathbf{o}}}_3][\tilde{\hat{\mathbf{o}}}_3]\mathbf{x}_1 - \frac{\mu_a \mathbf{x}_1}{||\mathbf{x}_1||^3} + \frac{\mu_s(3{}^O\hat{\mathbf{d}}{}^O\hat{\mathbf{d}}^T - [I_{3\times3}])\mathbf{x}_1}{d^3} \\ &+ C_{SRP}\frac{P_0(1+\rho)A_{sc}}{M_{sc}}\frac{(1\mathrm{AU})^2}{d^2}\hat{\mathbf{o}}_1 + \sum_i^I \frac{{}^O\mathbf{F}_i}{M_{sc}} + \sum_j^J \frac{{}^O\mathbf{F}_j}{M_{sc}} \end{aligned} \tag{3}$$

User Guide

A detailed example of the module is provided in **scenarioSmallBodyFeedbackControl**. However, the initialization of the module is also shown here. The module is first initialized as follows:

```
waypointFeedback = smallBodyWaypointFeedback.SmallBodyWaypointFeedback()
```

The asteroid ephemeris input message is then connected. In this example, we use the **Module: planetNav** module.

```
waypointFeedback.asteroidEphemerisInMsg.subscribeTo(planetNavMeas.ephemerisOutMsg)
```

A standalone message is created for the sun ephemeris message.

```
sunEphemerisMsgData = messaging.EphemerisMsgPayload()
sunEphemerisMsg = messaging.EphemerisMsg()
sunEphemerisMsg.write(sunEphemerisMsgData)
waypointFeedback.sunEphemerisInMsg.subscribeTo(sunEphemerisMsg)
```

The navigation attitude and translation messages are then subscribed to

```
waypointFeedback.navAttInMsg.subscribeTo(simpleNavMeas.attOutMsg)
waypointFeedback.navTransInMsg.subscribeTo(simpleNavMeas.transOutMsg)
```

Finally, the area, mass, inertia, and gravitational parameter of the asteroid are initialized

```
waypointFeedback.A_sc = 1.  # Surface area of the spacecraft, m^2
waypointFeedback.M_sc = mass  # Mass of the spacecraft, kg
waypointFeedback.IHubPntC_B = unitTestSupport.np2EigenMatrix3d(I)  # sc inertia
waypointFeedback.mu_ast = mu  # Gravitational constant of the asteroid
```

The reference states are then defined:

```
waypointFeedback.x1_ref = [-2000., 0., 0.]
waypointFeedback.x2_ref = [0.0, 0.0, 0.0]
```

Finally, the feedback gains are set:

```
waypointFeedback.K1 = unitTestSupport.np2EigenMatrix3d([5e-4, 0e-5, 0e-5, 0e-5, 5e-4, 0e-5,
0e-5, 0e-5, 5e-4])
waypointFeedback.K2 = unitTestSupport.np2EigenMatrix3d([1., 0., 0., 0., 1., 0., 0., 0., 1.])
```

***class* SmallBodyWaypointFeedback : *public* SysModel**

#include <smallBodyWaypointFeedback.h>

This module is provides a Lyapunov feedback control law for waypoint to waypoint guidance and control about a small body. The waypoints are defined in the Hill frame of the body.

Public Functions

`SmallBodyWaypointFeedback()`

This is the constructor for the module class. It sets default variable values and initializes the various parts of the model

`~SmallBodyWaypointFeedback()`

Module Destructor

void `SelfInit()`

Self initialization for C-wrapped messages.

Initialize C-wrapped output messages

void `Reset(uint64_t CurrentSimNanos)`

This method is used to reset the module and checks that required input messages are connect.

void `UpdateState(uint64_t CurrentSimNanos)`

This is the main method that gets called every time the module is updated. Provide an appropriate description.

void `readMessages()`

This method reads the input messages each call of updateState

void `computeControl(uint64_t CurrentSimNanos)`

This method computes the control using a Lyapunov feedback law

void `writeMessages(uint64_t CurrentSimNanos)`

This method reads the input messages each call of updateState

Public Members

ReadFunctor<NavTransMsgPayload> `navTransInMsg`

translational navigation input message

ReadFunctor<NavAttMsgPayload> `navAttInMsg` attitude navigation input message

ReadFunctor<EphemerisMsgPayload> `asteroidEphemerisInMsg`

asteroid ephemeris input message

ReadFunctor<EphemerisMsgPayload> `sunEphemerisInMsg`

sun ephemeris input message

Message<CmdForceBodyMsgPayload> `forceOutMsg`

force command output

CmdForceBodyMsg_C `forceOutMsgC` = {}

C-wrapped force output message.

BSKLogger `bskLogger`

— BSK Logging

double `C_SRP`

SRP scaling coefficient.

double `P_0`

SRP at 1 AU.

double `rho`

Surface reflectivity.

double `A_sc`

Surface area of the spacecraft.

double `M_sc`

Mass of the spacecraft.

Eigen::Matrix3d `IHubPntC_B`

sc inertia

double `mu_ast`

Gravitational constant of the asteroid.

Eigen::Vector3d `x1_ref`

Desired Hill-frame position.

Eigen::Vector3d `x2_ref`

Desired Hill-frame velocity.

Eigen::Matrix3d `K1`

Position gain.

Eigen::Matrix3d `K2`

Velocity gain.

Private Members

NavTransMsgPayload `navTransInMsgBuffer`

local copy of message buffer

NavAttMsgPayload `navAttInMsgBuffer`

local copy of message buffer

EphemerisMsgPayload `asteroidEphemerisInMsgBuffer`

local copy of message buffer

EphemerisMsgPayload `sunEphemerisInMsgBuffer`

local copy of message buffer

uint64_t `prevTime`

Previous time, ns.

double `mu_sun`

Gravitational parameter of the sun.

Eigen::Matrix3d `o_hat_3_tilde`

Tilde matrix of the third asteroid orbit frame base vector.

Eigen::Vector3d `o_hat_1`

First asteroid orbit frame base vector.

Eigen::MatrixXd `I`

3 x 3 identity matrix

ClassicElements `oe_ast`

Orbital elements of the asteroid.

double `F_dot`

Time rate of change of true anomaly.

double `F_ddot`

Second time derivative of true anomaly.

Eigen::Vector3d `r_BN_N`

Eigen::Vector3d `v_BN_N`

Eigen::Vector3d `v_ON_N`

Eigen::Vector3d `r_ON_N`

Eigen::Vector3d `r_SN_N`

Eigen::Matrix3d `dcm_ON`

DCM from the inertial frame to the small-body's hill frame.

Eigen::Vector3d `r_SO_O`

Vector from the small body's origin to the inertial frame origin in small-body hill frame components.

Eigen::Vector3d `f_curr`

Eigen::Vector3d `f_ref`

Eigen::Vector3d `x1`

Eigen::Vector3d `x2`

Eigen::Vector3d `dx1`

> **Eigen::Vector3d** `dx2`

> **Eigen::Vector3d** `thrust_O`

> **Eigen::Vector3d** `thrust_B`