Autonomous Vehicle Simulation (AVS) Laboratory, University of Colorado

Basilisk Technical Memorandum

Document ID: Basilisk-coarseSunSensor

MODULE TO APPLY A PRESCRIBED FORCE OR TORQUE ONTO A RIGID BODY

Prepared by	H. Schaub

Status: First Version
Scope/Contents
This module defines both individual Coarse Sun Sensor (CSS) modules, as well as an array or constellation of CSS devices. The CSS modules determine an ideal cosine response behavior, and can be corrupted through a Kelly-curve and gaussian noise. The CSS response can also be reduced due to being partially or fully in a planet's shadow.

Rev:	**Change Description**	**By**
v1.0	Initial document	H. Schaub

Contents

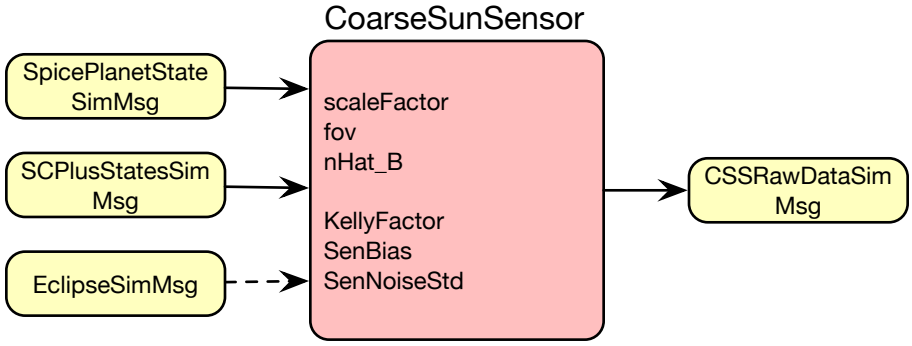

Fig. 1: Illustration of the CoarseSunSensor() modulej

1 Introduction

This document describes how Coarse Sun Sensor (CSS) devices are modeled in the Basilisk software. Each CSS is modeled through a nominal cosine response relative to the sunlight incidence angle. This response can then be corrupted, or reduced due to being in a planet's shadow. It is possible to add individual CSS sensors to the BSK evaluation stack. However, it is typically simpler to use the `CSSConstellation()` class to store a list of CSS units which are updated as a group, along with a unique CSS array output message.

2 Single CSS module

2.1 I/O Messages

First, let us discuss the input and output messages of the individual CSS sensor module. The two required input messages are of the type `SpicePlanetStateSimMsg` and `SCPlusStatesSimMsg`. The first message is used to determine the sun's location relative to the inertial frame ${}^{\mathcal{N}}\boldsymbol{r}_{\odot/\mathcal{N}}$. The second message is used to read in the spacecraft inertial position vector relative to the same inertial frame ${}^{\mathcal{N}}\boldsymbol{r}_{\mathcal{B}/\mathcal{N}}$. Finally, the last message is optional. If it is connected, it provides the sun shadow parameter indicating if the spacecraft is in a planet's shadow.

The output message of an individual CSS unit creates a message containing the simulated CSS sensor.

2.2 CSS Signal Simulation

To evaluate the sun heading vector $\boldsymbol{s}$, the satellite and sun position vectors are used.

$$ {}^{\mathcal{N}}\boldsymbol{s} = {}^{\mathcal{N}}\boldsymbol{r}_{\odot/\mathcal{N}} - {}^{\mathcal{N}}\boldsymbol{r}_{\mathcal{B}/\mathcal{N}} \tag{1}$$

After normalizing this vector to $\hat{\boldsymbol{s}}$ and mapping $\boldsymbol{\sigma}_{\mathcal{B}/\mathcal{N}}$ to $[BN]$, it is mapped into body frame components through

$$ {}^{\mathcal{B}}\hat{\boldsymbol{s}} = [BN]{}^{\mathcal{N}}\hat{\boldsymbol{s}} \tag{2}$$

The CSS sensor unit normal vector is given by ${}^{\mathcal{B}}\hat{\boldsymbol{n}}$ in body frame components. The normalized cosine sensor signal $\hat{\gamma}$ is thus determined through

$$ \hat{\gamma} = \hat{\boldsymbol{n}} \cdot \hat{\boldsymbol{s}} = \cos\phi \tag{3}$$

where ϕ is the CSS sunlight incidence angle. This is the normalized CSS signal where 1 is returned if the sensor is looking straight at the sun. If the sensor axis $\hat{\boldsymbol{n}}$ is more the field of view half-angle (set through `fov`) from the sun axis, then a 0 signal is simulated. This `fov` variable is the angle from $\hat{\boldsymbol{n}}$ beyond which the CSS signal is set to zero.

Let p_s be the local solar shadow parameter. If the spacecraft is outside of a planet's shadow, this value is 1. If it is within the shadow, then it is $0 \leqslant p_s < 1$. The shadow adjusted CSS signal is thus computed as

$$ \hat{\gamma}_s = \hat{\gamma} p_s \tag{4}$$

To simulate CSS signal corruptions, then a Kelly curve corruption, a bias and a gaussian noise can be included. The normalized bias is set through `SenBias`, while the normalized noise is set through `SenNoiseStd`. Let $\boldsymbol{p}_n$ be the normalized sensor noise. Note that this noise levels is non-dimensional relative to the unit cosine curve.

The Kelly corruption parameter allows for the CSS signal to pinch towards zero for larger incidence angles as illustrated in Figure 2. The amount of signal distortion is set through κ, where the Kelly factor p_κ is then computed as

$$ p_\kappa = 1 - e^{-\hat{\gamma}_s^2/\kappa} \tag{5}$$

The corrupted normalized CSS signal is then

$$ \hat{\gamma}_c = (\hat{\gamma}_s + p_a)p_\kappa + p_n \tag{6}$$

At this point the signal shape is correct, but not the scale. The final step is to scale. Using the `scaleFactor` $= \alpha$ parameter, the CSS signal is now scaled to a dimensional value where the peak is not 1, but rather `scaleFactor`.

$$ \gamma = \hat{\gamma}_c \alpha \tag{7}$$

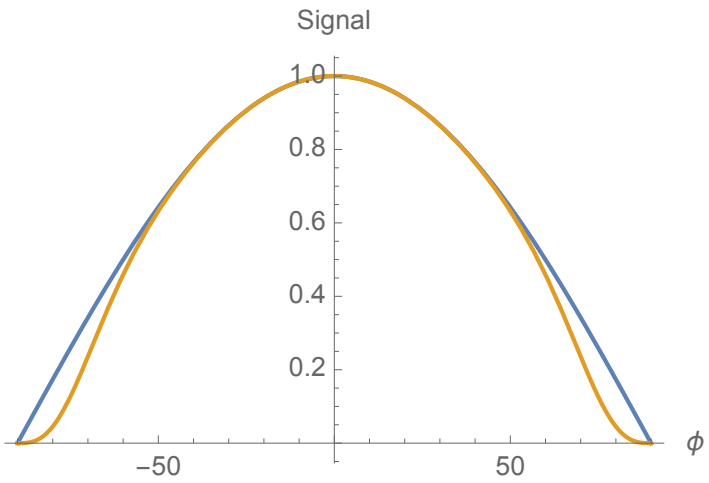

Fig. 2: Kelly distortion illustration with $\kappa = 0.1$

2.3 Setting the CSS Unit Direction Vector

The unit vector of each CSS device is set through the $\mathcal{B}$ frame vector representation

$$\texttt{nHat\_B} \equiv {}^{\mathcal{B}}\hat{\boldsymbol{n}} \tag{8}$$

It is possible to set these vectors directly. However, there are some convenience functions that make this process easier.

Multiple CSS devices are often arranged together on a single CSS platform. The orientation of the body-fixed platform frame $\mathcal{P}$ is given through $[PB]$. In the CSS module, the DCM is specified through the variable `dcm_PB`.

2.4 Module Parameters

This section describes the various module parameters that can be set via Python to configure the CSS unit. The platform frame is

2.4.1 `theta` Parameter

This is the CSS azimuth angle relative to the platform frame.

3 User Guide

3.1 Setting the CSS Unit Direction Vector

3.1.1 Direct Method

The unit normal vector of each CSS sensor is set through the $\mathcal{B}$ frame vector representation

$$\texttt{nHat\_B} \equiv {}^{\mathcal{B}}\hat{\boldsymbol{n}} \tag{9}$$

It is possible to set these vectors directly. However, there are some convenience functions that make this process easier.

3.1.2 Via a Common Sensor Platform

Multiple CSS devices are often arranged together on a single CSS platform. The orientation of the body-fixed platform frame $\mathcal{P} : \{\hat{\boldsymbol{p}}_1, \hat{\boldsymbol{p}}_2, \hat{\boldsymbol{p}}_3\}$ relative to the body frame $\mathcal{B} : \{\hat{\boldsymbol{b}}_1, \hat{\boldsymbol{b}}_2, \hat{\boldsymbol{b}}_3\}$ is given through $[PB]$. In the CSS module, the DCM is specified through the variable `dcm_PB`.

Assume the DCM `dcm_PB` is set directly via Python. Two angles then define the orientation of the sensor normal axis $\hat{\boldsymbol{n}}$. The elevation angle is ϕ and the azimuth angle is θ. These are an Euler angle

sequence with the order 3-(-2). The elevation angle is a positive 3-axis rotation, while the azimuth is a minus 2-axis rotation. The helper function

$$\texttt{setUnitDirectionVectorWithPerturbation}(\theta_p, \phi_p)$$

where θ_p and ϕ_p are CSS heading perturbation can can be specified. The Euler angles implemented are then

$$\phi_{\text{actual}} = \phi_{\text{true}} + \phi_p \tag{10}$$
$$\theta_{\text{actual}} = \theta_{\text{true}} + \theta_p \tag{11}$$

To setup un-perturbed CSS sensor axes simple set these perturbation angles to zero.

Instead of setting the DCM `dcm_PB` variable directly, this can also be set via the helper function

$$\texttt{setBodyToPlatformDCM}(\psi, \theta, \phi)$$

where (ψ, θ, ϕ) are classical $3-2-1$ Euler angles that map from the body frame $\mathcal{B}$ to the platform frame $\mathcal{P}$.

3.2 CSS Field-of-View

The CSS sensor field of view is set by specifying the class variable

$$\texttt{fov}$$

This angle is the angle from the bore-sight axis to the edge of the field of view, and is expressed in terms of radians.

3.3 CSS Output Scale

The general CSS signal computation is performed in a normalized manner yielding an unperturbed output between 0 and 1. The CSS module variable

$$\texttt{scaleFactor}$$

is the scale factor α which scales the output to the desired range of values, as well as the desired units. For example, if the maximum sun signal ($\hat{\boldsymbol{n}}$ points directly at sun) should yield 1 mA, then the scale factor is set to this value.

3.4 Specifying CSS Sensor Noise

Three types of CSS signal corruptions can be simulated. If not specified, all these corruptions are zeroed.

The Kelly corruption parameter κ is set by specifying the variable

$$\texttt{KellyFactor}$$

Second, to add a gaussian noise component to the normalized output the variable, the variable

$$\texttt{SenNoiseStd}$$

is set to a non-zero value. This is the standard deviation of normalized gaussian noise. Note that this noise magnitude is in terms of normalized units as it is added to the 0-1 nominal signal.

Lastly, to simulate a signal bias, the variable

$$\texttt{SenBias}$$

is set to a non-zero value. This constant bias of the normalized gaussian noise.

3.5 Connecting Messages

Of the three possible input messages to the CSS module, the following message inputs are required for the module to properly operate:

`SpicePlanetStateSimMsg` `SCPlusStatesSimMsg`

The first message is used to know the sun heading vector, while the second message provides the spacecraft inertial orientation,

The last message is optional. If the `EclipseSimMsg` is connected, then the solar eclipse information is taking into account. The eclipse info provides 0 if the spacecraft is fully in a planet's shadow, 1 if in free space fully exposed to the, and a value between (0,1) if in the penumbra region. The cosine sensor value $\hat{\gamma}$ is scaled by this eclipse value. If the message is not connected, then this value default to 1, thus simulating a spacecraft that is fully exposed to the sun.

3.6 Setting Up CSS Modules

3.6.1 Individual CSS Units

It is possible to add Individual CSS units to the Basilisk simulation. This is done by invoking instances of the `CoarseSunSensor()` class from python, configuring the required states, and then adding each to the BSK evaluation stack. Each `CoarseSunSensor` class has it's own `UpdateState()` method that get's evaluated each update period.

This setup is convenient if only 1-2 CSS units have to be modeled, but can be cumbersome if a larger cluster of CSS units must be administered. When setup this way, each CSS unit will output an individual CSS output message.

3.6.2 Array or Constellation of CSS Units

An alternate method to setup a series of CSS units is to use the `CSSConstellation()` class. This class is able to store a series of CSS `CoarseSunSensor` objects, and engage the update routine on all of them at once. This way only the `CSSConstellation` module needs to be added to the BSK update stack. In this method the `CSSConstellation` module outputs a single CSS sensor message which contains an array of doubles with the CSS sensor signal. Here the individual CSS units `CSS1`, `CSS2`, etc. are setup and configured first. Next, they are added to the `CSSConstellation` through the python command

$$\texttt{cssConstellation.sensorList} = \texttt{coarse\_sun\_sensor.CSSVector}([\texttt{CSS1}, \texttt{CSS2}, ..., \texttt{CSS8}])$$