Autonomous Vehicle Simulation (AVS) Laboratory, University of Colorado

Basilisk Technical Memorandum

Document ID: Basilisk-coarseSunSensor

MODULE TO SENSE SUN DIRECTION

Prepared by	S. Carnahan

Status: Tested
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. The CSS unit test tests a pure signal as well as each of the corruptions individually and combined. Constellations are set up in multiple ways to ensure proper functionality.

Rev	Change Description	By	Date
1.0	Initial Document	H. Schaub	20170720
1.1	Full Draft with test description	S. Carnahan	20170803

Contents

1 Model Description

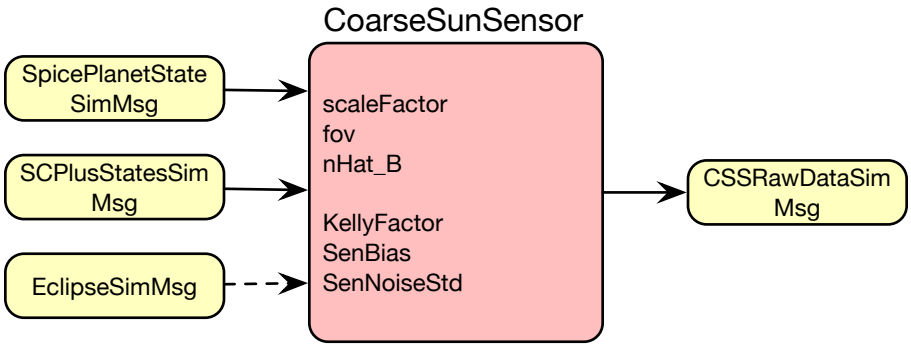

Fig. 1: Illustration of the CoarseSunSensor() module I/O

1.1 Overview

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.

1.2 Single CSS module

1.2.1 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.

1.2.2 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 than 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 level is non-dimensional relative to the unit cosine curve. Therefore, it will also be scaled when a scale factor is applied to the output.

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

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

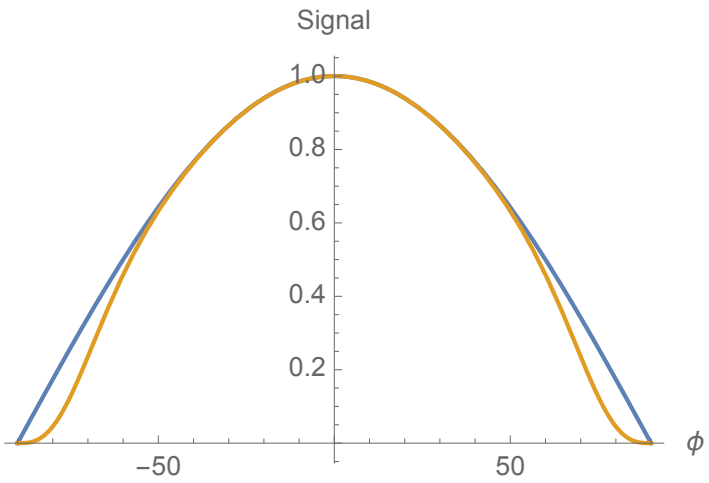

Fig. 2: Kelly distortion illustration with $\kappa = 0.1$

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.

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

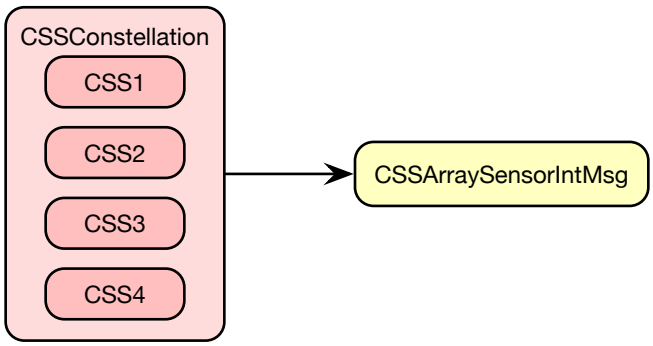

Fig. 3: Illustration of the CSS Constellation class

1.3 Constellation or Array of CSS Modules

The `CSSConstellation` class is defined as a convenience such that the Basilisk update function is called on a group of CSS modules. Here the CSS BSK modules are created and configured as before, but are then provided to the `CSSConstellation` class object as a list of CSS in python. The CSS can be configured individually, and must each be connected with the required input messages. However, the `CSSConstellation` will gather all the CSS outputs and write a single CSS Constellation output message of the type `CSSArraySensorIntMsg` as shown in Figure 3.

2 Model Functions

The coarse sun sensor functions have been mentioned in passing but are described below:

- **Cosine Curve**: Given the sensor vector to the sun, a cosine signal is generated and returned.

- **Field of View**: The module can limit it's outputs to be zero outside of a given half-conic angle field of view.
- **Kelly Curve**: The module can apply a Kelly Factor to the cosine curve to simulation realistic effects at high incidence angles.
- **Noise**: The module can apply Gaussian noise to the signal.
- **Bias**: the module can apply a bias to the signal.
- **Single Sensor**: The module can set up a signal coarse sun sensor.
- **CSS Constellation**: The module can set up a constellation of coarse sun sensors.
- **Interface: Spacecraft State**: The module receives spacecraft state information from the messaging system.
- **Interface: Sun State**: The module receives sun ephemeris information through the messaging system
- **Interface: Eclipse**: The module receives optional eclipse information through the messaging system.
- **Interface: Outputs**: The single CSS modules create individual CSS sensor output messages, while the CSS Constellation creates a single output message with all the CSS sensor signals.

3 Model Assumptions and Limitations

Assumptions made in this coarse sun sensor module and the corresponding limitations are shown below.

1. **Conical Symmetry**: The module assumes that sensor readings and bias are the same in every direction from normal. In reality, there may be different biases or errors depending on which direction the sun vector is from the sensor normal vector. This limits the use of this module to sensors that are symmetric or only slightly asymmetric.
2. **Albedo**: Currently, the model is hard-coded to zero-out any albedo input, so it is limited to cases where albedo is not important.
3. **Cosine Behavior**: The model assumes the sensor to have a nominal cosine behavior. This is distorted by the Kelly factor. To improve behavior closer to reality where housing and spacecraft glint might cause slightly non-conical non-cosine behaviors, a look-up table of sensor outputs compared to sun headings could be implemented in future work.

4 Test Description and Success Criteria

The Coarse Sun Sensor Test, test_coarseSunSensor.py, contains ten tests. The simulation is set up with only the coarse sun sensor(s) and made-up messages to simulation spacecraft, sun, and eclipse information. The spacecraft is in a convenient attitude relative to the sun and rotates all of the sensors in a full circle past the sun.

1. **Basic Functionality**: A single sensor is run with minimal modifications and compared to a cosine.

   Success Criteria: The output curve should match a cosine curve.

2. **Eclipse**: A single sensor is run with an eclipse simulated and compared to an eclipse factored cosine.

   Success Criteria: The output curve should match a cosine curve times the eclipse factor input

3. **Field of View**: A single sensor is run with a smaller field of view and compared to a clipped cosine.

   Success Criteria: The output curve should match a cosine curve truncated to zero beyond the field of view input.

4. **Kelly Factor**: A single sensor is run with a Kelly factor input and compared to a modified cosine.

   Success Criteria: The output curve should match a cosine curve modified by the kelly curve equation seen previously in this report

5. **Scale Factor**: A single sensor is run with a scale factor and compared to a scaled cosine.

   Success Criteria: The output curve should match a cosine curve multiplied by the scaleFactor input.

6. **Bias**: A single sensor is run with a bias and compared to a modified cosine.

   Success Criteria: The output curve should match a cosine curve shifted in magnitude by the bias input.

7. **Noise**: A single sensor is run with noise and the standard deviation of that noise is compared to the input standard deviation.

   Success Criteria: Once a clean cosine curve is subtracted from the output, the standard deviation should match the standard deviation input.

8. **Albedo**: A single sensor is run with an albedo input and shown to be no different than the standard cosine truth value. This is done because there is some albedo functionality programmed into the module but it should be inactive at this time.

   Success Criteria: The output curve should match a cosine curve because the albedo input should have no effect.

9. **Combined**: All of the inputs above are run on a single simulation. The expected result without noise is subtracted from the result. Then, the standard deviation of the noise. is compared to the expected standard deviation.

   Success Criteria: Once a cosine curve modified by eclipse, field of view, kelly factor, scale factor, bias, and albedo are subtracted from the output, the standard deviation should match the given standard deviation.

10. **Constellation**: Two constellations of sensors are set up using various set up methods and simulated with a clean signal. The two constellations are tested to be identical to one another. Constellation P1 is established by directly assigning normal vectors to four sensors. Constellation P2 is established by giving angles that specify the placement of the constellation platform relative to the spacecraft body frame. Then, for constellation 2, the unit direction vector for each sensor is set with an azimuth and an elevation via coarseSunSensor.setUnitDirectionVectorWithPerturbation(). Finally, the fourth sensor in constellation P2 is set up in a different way than the others. It is not assigned platform frame angles but it is given incorrect azimuth and elevation headings which are

corrected with "perturbation" inputs. Through all of these tests, constellation set up is verified, including the default platform DCM (identity).

Success Criteria: The output curve from constellation P1 should match the output curve from constellation P2.

5 Test Parameters

Pytest runs the following cases (numbered as above) when it is called for this test:

Table 2: Error tolerance for each test.

Test	use Constellation	visibility Factor	fov	kelly	scale Factor	bias	noise Std	albedo Value	errTol
1	False	1.00	1.5708	0.00	1.00	0.00	0.00	0	1.0e-12
2	False	0.50	1.5708	0.00	1.00	0.00	0.00	0	1.0e-12
3	False	1.00	1.1781	0.00	1.00	0.00	0.00	0	1.0e-12
4	False	1.00	1.5708	0.15	1.00	0.00	0.00	0	1.0e-12
5	False	1.00	1.5708	0.00	2.00	0.00	0.00	0	1.0e-12
6	False	1.00	1.5708	0.00	1.00	0.50	0.00	0	1.0e-12
7	False	1.00	1.5708	0.00	1.00	0.00	0.25	0	1.0e-03
8	False	1.00	1.5708	0.00	1.00	0.00	0.00	0	1.0e-12
9	False	0.50	1.1781	0.15	2.00	0.50	0.25	0	1.0e-03
10	True	1.00	1.5708	0.00	1.00	0.00	0.00	0	1.0e-12

6 Test Results

The results of each test are shown in the table below. If a test did not pass, an error message is included.

Table 3: Test results.

Test	Pass/Fail	Notes
1	PASSED	
2	PASSED	
3	PASSED	
4	PASSED	
5	PASSED	
6	PASSED	
7	PASSED	
8	PASSED	
9	PASSED	
10	PASSED	

In addition to the tabulated results, test data has been plotted for visual inspection. In Fig. 4, all single coarse sun sensor simulations have been plotted on top of one another. This makes for convenient comparison between the cases. For instance, the scaleFactor case can be seen to peak at 2 rather than 1 whereas the eclipse case peaks at 0.5. Furthermore, the fieldOfView test drops to zero at a value less than $\frac{\pi}{2}$ but it follows the plain case otherwise. The kellyFactor case similarly follows the plain curve, except at the edges. The albedo curve cannot be seen because it lies directly beneath the plain curve. The bias curve is equivalent to the plain curve, but shifted up. Finally, the two curves with noise clearly follow the ordinary curve pattern.

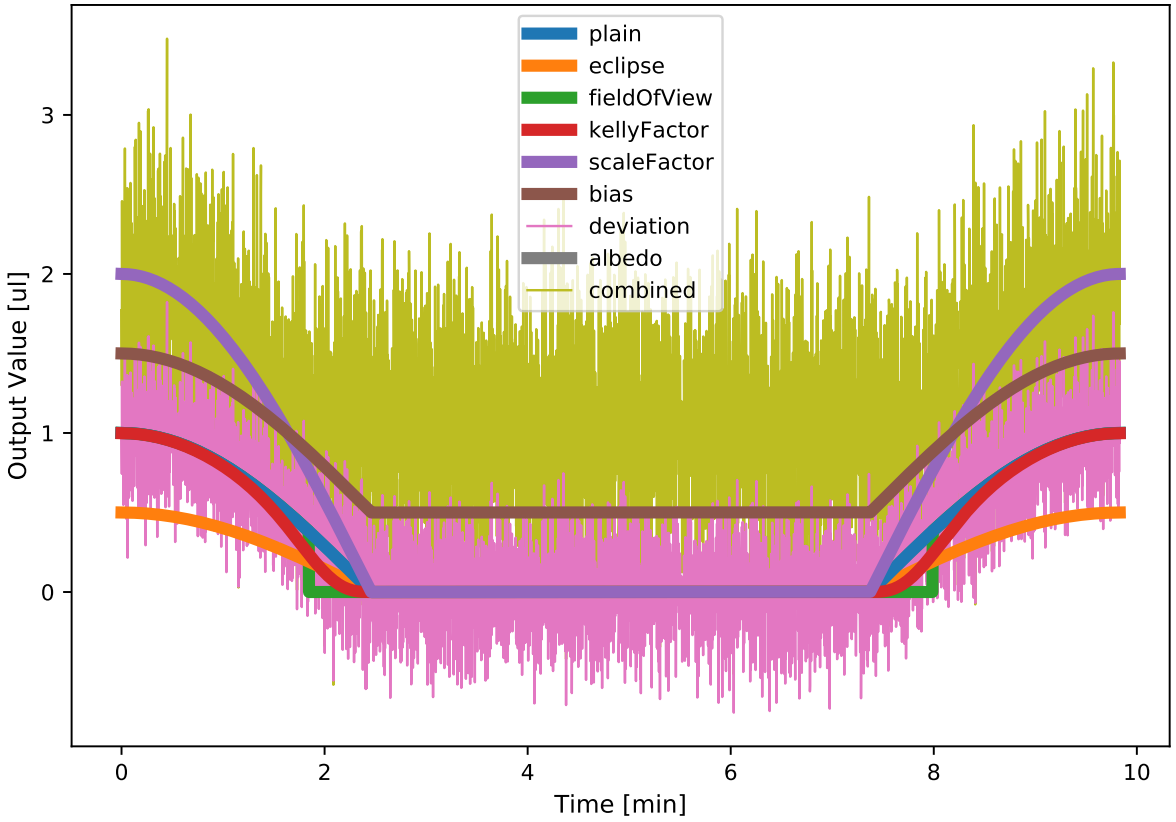

Fig. 4: Plot of all cases of individual coarse sun sensor in comparison to each other

The constellation test results are shown in Fig. 5. The results are identical to each other and the test has been successful.

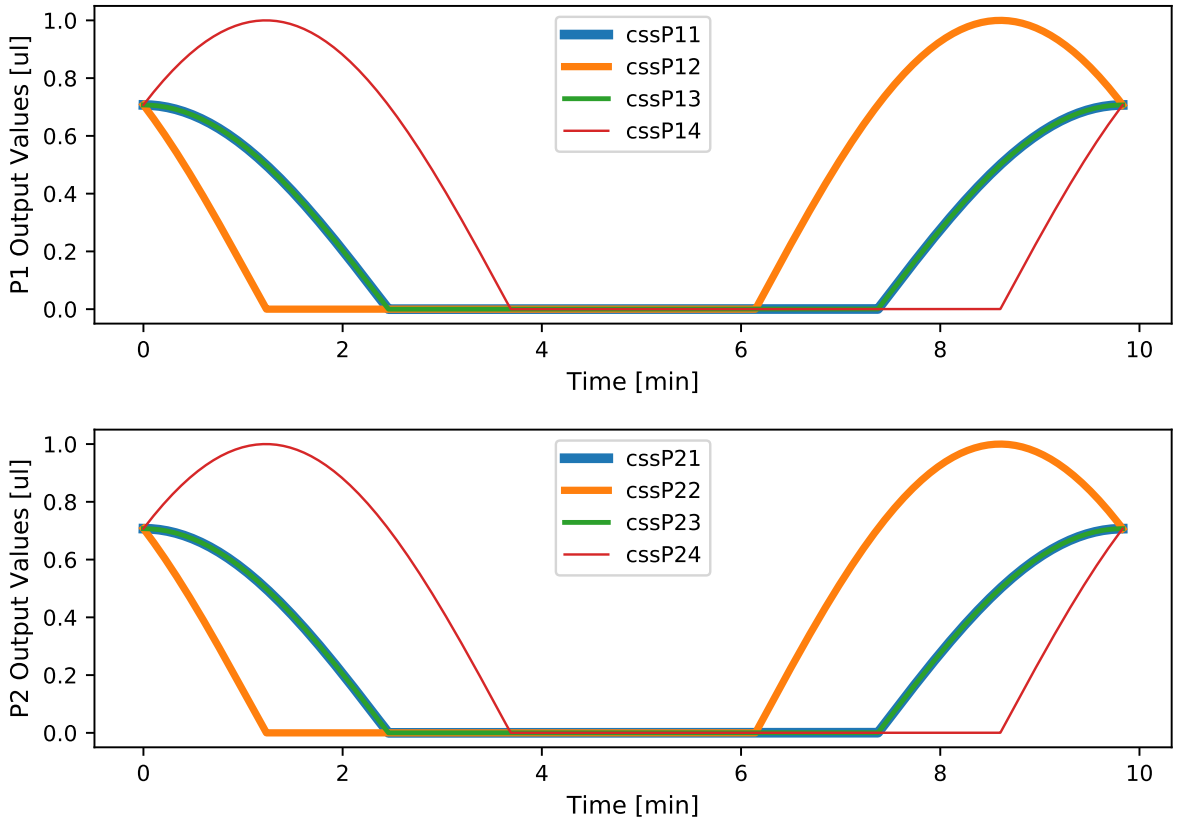

Fig. 5: Plot of first and second constellation outputs for comparision.

7 User Guide

7.1 Setting the CSS Unit Direction Vector

7.1.1 Direct Method

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

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

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

7.1.2 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

$$\phi_{\text{actual}} = \phi_{\text{true}} + \phi_p \tag{9}$$
$$\theta_{\text{actual}} = \theta_{\text{true}} + \theta_p \tag{10}$$

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 $\mathcal{P}$.

7.2 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.

7.3 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.

7.4 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.

7.5 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.

7.6 Setting Up CSS Modules

7.6.1 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.

7.6.2 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

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