Autonomous Vehicle Simulation (AVS) Laboratory, University of Colorado

Basilisk Technical Memorandum

Document ID: Basilisk-imu

INERTIAL MEASUREMENT UNIT C++ MODEL

Prepared by	S. Carnahan

Status: Tested
Scope/Contents
The Basilisk Inetial Measurement Unit (IMU) module is responsible for producing sensed body rates and acceleration from calculated values. It also provides a change in velocity and change in attitude value for the time between IMU calls. The IMU module applies Gauss-Markov process noise to the true body rates and acceleration. A unit test has been written which validates MRP switching, static bias, process noise, discretization, saturation, spacecraft center of mass (CoM) offset, sensor misalignment, and bias walk bounds for both the gyroscope and accelerometer.

Rev	Change Description	By	Date
1.0	First draft	J. Alcorn	20170713
1.1	Added Math Documentation and User Guide. Implemented AutoTeX	S. Carnahan	20170713

Contents

---

1 Model Description

The Basilisk IMU module imu_sensor.cpp is responsible for producing sensed body rates and acceleration from simulation truth values. It also provides a change in velocity and change in attitude value for the time between IMU calls. Each check within test_imu_sensor.py sets initial attitude MRP, body rates, and accumulated Delta V and validates output for a range of time.

There is a large variation throughout the industry as to what constitutes and IMU. Some manufacturers offer IMUs which output only acceleration and angular rate while others include accumulated change in velocity in attitude. For Basilisk, the IMU is defined as a device which outputs all four values.

1.1 Mathematical Model

1.1.1 Angular Rates

The angular rate of the sensor is output as:

$$^{P}\boldsymbol{\omega}_{S/N} = [PB]\,^{B}\boldsymbol{\omega}_{B/N} \tag{1}$$

Where P is the sensor platform frame, B is the vehicle body frame, and N is the inertial frame. $[PB]$ is the direction cosine matrix from B to P. This allows for an arbitrary angular offset between B and P but does not allow for that offset to be time-varying.

1.1.2 Angular Displacement

The IMU also outputs the angular displacement accumulated between IMU calls. In order to avoid complexities having to do with the relative timestep between the dynamics process and the IMU calls, this is not calculated in the same way as an IMU works physically. In this way, also, the dynamics do not have to be run at a fast enough rate for a physical IMU angular accumulation to be simulated. The modified Rodriguez parameter (MRP) is recorded for the last time (1) the IMU was called. It is then differenced with the current (2) MRP:

$$\boldsymbol{\sigma}_{2/1} = \boldsymbol{\sigma}_{B/N,2} \ominus \boldsymbol{\sigma}_{B/N,1} \tag{2}$$

This is interpreted as the MRP from the body frame at time 1 to the body frame at time 2.$\ominus$ is the MRP subtraction function. Eq. 2 says that the body frame rotation from time 1 to time 2 is equivalent to rotating from N to B at time 2 and then rotating the reverse of the rotation from N to B at time 1. $\boldsymbol{\sigma}_{2/1}$ is then converted to a principle rotation vector (PRV) in the current body frame and that vector is rotated into the sensor platform frame:

$${}^{B}q_i = \frac{\sigma_{2/1,i}}{|\boldsymbol{\sigma}_{2/1}|4\tan^{-1}(|\boldsymbol{\sigma}_{2/1}|)} \tag{3}$$

$${}^{P}\boldsymbol{q} = [PB]^{B}\boldsymbol{q} \tag{4}$$

Above, q is the PRV and i indicates the i^{th} component of $\boldsymbol{q}$ and $\boldsymbol{\sigma}$. $\boldsymbol{q}$ is then made of the elements q_1, q_2, and q_3. q is the module output for angular displacement.

1.1.3 Linear Acceleration

The sensor is assumed to have an arbitrary offset from the center of mass of the spacecraft. However, because of the completely coupled nature of the Basilisks dynamics framework, the center of mass does not need to be present explicitly in equations of motion for the sensor. It is implicit in the motion of the body frame. With that in mind, the equation for the acceleration of the sensor is derived below:

$$\boldsymbol{r}_{S/N} = \boldsymbol{r}_{B/N} + \boldsymbol{r}_{S/B} \tag{5}$$

Using the transport theorem for $\dot{\boldsymbol{r}}_{S/B}$:

$$\dot{\boldsymbol{r}}_{S/N} = \dot{\boldsymbol{r}}_{B/N} + \boldsymbol{r}'_{S/B} + \boldsymbol{\omega}_{B/N} \times \boldsymbol{r}_{S/B} \tag{6}$$

But $\boldsymbol{r}'_{S/B}$ is 0 because the sensor is assumed to be fixed relative to the body frame. Then,

$$\ddot{\boldsymbol{r}}_{S/N} = \ddot{\boldsymbol{r}}_{B/N} + \dot{\boldsymbol{\omega}}_{B/N} \times \boldsymbol{r}_{S/B} + \boldsymbol{\omega}_{B/N} \times (\boldsymbol{\omega}_{B/N} \times \boldsymbol{r}_{S/B}) \tag{7}$$

The equation above is indeed the equation for the inertial acceleration of the sensor, but the sensor will only measure the non-conservative accelerations. To account for this, the equation is modified to be:

$$\ddot{\boldsymbol{r}}_{S/N,\text{sensed}} = (\ddot{\boldsymbol{r}}_{B/N} - \boldsymbol{a}_{\text{g}}) + \dot{\boldsymbol{\omega}}_{B/N} \times \boldsymbol{r}_{S/B} + \boldsymbol{\omega}_{B/N} \times (\boldsymbol{\omega}_{B/N} \times \boldsymbol{r}_{S/B}) \tag{8}$$

where $\boldsymbol{a}_{\text{g}}$ is the instantaneous acceleration due to gravity. Conveniently, $\ddot{\boldsymbol{r}}_{B/N}$ is available directly from the spacecraft, but in the body frame. $\boldsymbol{r}_{S/B}$ is also available in the body frame. $\dot{\boldsymbol{\omega}}_{B/N}$ is given by the spacecraft in body frame coordinates as well. As such, the above equation is first evaluated in the body frame and then converted to the sensor platform frame, giving the acceleration sensed by the IMU in the IMU platform frame coordinates:

$${}^{P}\ddot{\boldsymbol{r}}_{S/N,\text{sensed}} = [PB]^{B}\ddot{\boldsymbol{r}}_{S/N,\text{sensed}} \tag{9}$$

1.1.4 Change In Velocity

The IMU also outputs the velocity accumulated between IMU calls. In order to avoid complexities having to do with the relative time step between the dynamics process and the IMU calls, this is not calculated in the same way as an IMU works physically. In this way, also, the dynamics do not have to be run at a fast enough rate for a physical IMU velocity accumulation to be simulated.

Differencing Eq. 6 with itself from time 1 to time 2 gives the equation:

$$\Delta_{2/1}\dot{\boldsymbol{r}}_{S/N} = \Delta_{2/1}\dot{\boldsymbol{r}}_{B/N} + \Delta_{2/1}(\boldsymbol{\omega}_{B/N} \times \boldsymbol{r}_{S/B}) \tag{10}$$

$\Delta_{2/1}\dot{\boldsymbol{r}}_{B/N}$ is calculated as the difference between the total change in velocity accumulated by the spacecraft body frame at time 2 minus the total change in velocity accumulated by the spacecraft body frame at time 1:

$$\Delta_{2/1}\dot{\boldsymbol{r}}_{B/N} = DV_{\text{body\_total},2} - DV_{\text{body\_total},1} \tag{11}$$

The above DV values are given by the spacecraft module in body frame coordinates. Then,

$$\Delta_{2/1}(\boldsymbol{\omega}_{B/N} \times \boldsymbol{r}_{S/B}) = \boldsymbol{\omega}_{B/N_2} \times \boldsymbol{r}_{S/B_2} - \boldsymbol{\omega}_{B/N_1} \times \boldsymbol{r}_{S/B_1} \tag{12}$$

It is again convenient to evaluate this expression in the body frame because $\boldsymbol{\omega}_{B/N}$ is given by the spacecraft in body frame coordinates, $\boldsymbol{r}_{S/B}$ is given in body frame coordinates, and $\boldsymbol{r}_{S/B}$ is time-invariant in the body frame. i.e.:

$$\boldsymbol{r}_{S/B_1} = \boldsymbol{r}_{S/B_2} = \boldsymbol{r}_{S/B} \tag{13}$$

At this point, Eq. 10 is evaluated in the body frame and converted to the sensor platform frame:

$${}^{P}\Delta_{2/1}\dot{\boldsymbol{r}}_{S/N} = [PB]^{B}\Delta_{2/1}\dot{\boldsymbol{r}}_{S/N} \tag{14}$$

This, the change in velocity sensed by the IMU between IMU calls in platform frame coordinates, is the change in velocity output from the model.

Finally, it should be noted that the values for body frame acceleration and angular acceleration are linear estimates made by the spacecraftPlus module. This is done by dividing the values of change in velocity and change in angular velocity (respectively) by the time step over which spacecraftPlus is integrating. The before and after values of velocity and angular velocity are before and after the most recent spacecraftPlus integration. While not precisely the current spacecraft acceleration (or angular acceleration), this is the most accurate data available without running the dynamics integration an extra time, which would be computationally costly.

1.1.5 Error Modeling

The state which the simulation records for the spacecraft prior to sending that state to the IMU module is considered to be "truth". So, to simulate the errors found in real instrumentation, errors are added to the "truth" values for any measurement, $\mathbf{m}_{\text{truth}}$, that the IMU deals with.

$$\mathbf{m}_{\text{measured}} = \mathbf{m}_{\text{truth}} + \mathbf{e}_{\text{noise}} + \mathbf{e}_{\text{bias}} \tag{15}$$

1.1.6 Data Discretization

Because sensors record data digitally, that data can only be recorded in discrete chunks, rather than the (relatively) continuous values that the computer calculates at each time steps. In order to simulation real IMU behavior in this way, a least significant bit (LSB) value is accepted for both the gyro and the accelerometer. This LSB is applied as follows to any 3-dimensional measurement $\mathbf{m}$:

$$\mathbf{m}_{\text{discretized}} = (\text{LSB})\left\lfloor \left| \frac{\mathbf{m}_{\text{measured}}}{(\text{LSB})} \right| \right\rfloor \tag{16}$$

Where $\lfloor\ \rfloor$ indicate the **floor()** function and LSB can be either the accelerometer or gyro least significant bit.

1.1.7 Saturation

Real sensors can also become saturated. This is modeled simply by comparing $\mathbf{m}_{\text{discretized}}$ to the high and low saturation values for the sensor:

$$\mathbf{m}_{\text{saturated}} = max\Big(\mathbf{m}_{\text{sat,min}}, \big(min(\mathbf{m}_{\text{discretized}}, \mathbf{m}_{\text{sat,max}})\big)\Big) \tag{17}$$

Note that this is actually only computed if $\mathbf{m}_{\text{discretized}}$ proves to be outside of its saturation bounds.

2 Model Functions

The mathematical description of the IMU are implemented in imu_sensor.cpp. This code performs the following primary functions

- **Spacecraft State Measurement**: The code provides measurements of the spacecraft state (angular and linear).
- **Bias Modeling**: The code adds instrument bias and bias random walk to the signals.
- **Noise Modeling**: The code calculates noise according to the Gauss Markov model if the user asks for it and provides a perturbation matrix.
- **Discretization**: The code discretizes the signal to emulate real digital instrumentation. The least significant bit (LSB) can be set by the user.
- **Saturation**: The code bounds the output signal according to user-specified maximum and minimum saturation values.
- **Accelerometer Center of Mass Offset**: The code can handle accelerometer placement other than the center of mass of the spacecraft.
- **IMU Misalignment**: The code can handle IMU placement in a frame with constant rotational offset from assumed IMU orientation.
- **Bias Random Walk Bounds**: The code bounds bias random walk per user-specified bounds.
- **Interface: Spacecraft States**: The code sends and receives spacecraft state information via the Basilisk messaging system.
- **Interface: Spacecraft Mass**: The code receives spacecraft mass information via the Basilisk messaging system.

3 Model Assumptions and Limitations

This code makes assumptions which are common to IMU modeling.

- **Error Inputs**: Because the error models rely on user inputs, these inputs are the most likely source of error in IMU output. Instrument bias would have to be measured experimentally or an educated guess would have to be made. The Guass-Markov noise model has a well-known assumptions and is generally accepted to be a good model for this application.

4 Test Description and Success Criteria

This test is located at `SimCode/sensors/imu_sensor/_UnitTest/test_imu_sensor.py`. In order to get good coverage of all the aspects of the module, the test is broken up into several parts:

1. Gyro/Accelerometer I/O The check verifies basic I/O of body rates and acceleration. Initial attitude MRP, body rates, and Delta V are propagated and corresponding body rates, acceleration, DR, and DV are compared to module output.

2. MRP Switch The check validates that the module accounts for attitude MRP switching in calculation of body rates. Initial attitude MRP and body rates are propagated for a sufficient amount of time for the MRP to switch to the shadow set. The test verifies that the module sets the MRP switch flag to TRUE.

3. Static Bias The check validates static bias in gyro/accel measurements. Gyro and accelerometer static bias are set to nonzero values. Initial MRP, body rates, and DV are propagated. Module output is verified to contain data with static bias.

4. Process Noise The check verifies that the Gauss-Markov model applies noise of appropriate mean and standard deviation to the attitude coordinate output. This check does not consider bias random walk. Accelerometer and gyro noise standard deviations are set to nonzero values for each axis. Module output is verified by taking the standard deviation of output data and comparing to specified values.

5. Discretization The check verifies that the module correctly discretizes the gyro/accel data according to the specified least significant bit (LSB). LSB of gyro and accelerometer are set to nonzero values. Output is verifed to round input to nearest multiple of LSB.

6. Saturation The check verifies that the module saturates the output according to specified values. Gyro and accelerometer maximum output are set to nonzero values. Output is verified to not exceed specified saturation values for both negative and positive cases.

7. Accelerometer Center of Mass Offset The check validates that the accelerometer will give appropriate output based on an offset in center of mass from accelerometer.

8. IMU Misalignment The check validates measurements taken when the IMU is not correctly aligned (i.e. the IMU measurements are taken in a frame with constant rotational offset from assumed IMU orientation).

9. Bias Random Walk Bounds The check verifies that the Gauss-Markov model correctly applies bias random walk to the gyro and accelerometer output. Specified walk bounds are validated.

5 Test Parameters

This section summarizes the specific error tolerances for each test. Error tolerances are determined based on whether the test results comparison should be exact or approximate due to integration or other reasons. Error tolerances for each test are summarized in table 4.

Table 2: Error tolerance for each test.

Test	Tolerated Error
Gyro/Accelerometer I/O	1.0e-06
MRP Switching	-
Static Bias	1.0e-03
Process Noise	1.0e-02
Discretization	1.0e-05
Saturation	1.0e-03
Accelerometer Center of Mass Offset	1.0e-05
IMU Misalignment	1.0e-04
Bias Walk Bounds	-

6 Test Results

All checks within test_imu_sensor.py passed as expected. Table 3 shows the test results. Figures **??** and **??** show the module output for the process noise and walk bounds checks, respectively.

Table 3: Test results

Test	Pass/Fail	Notes
Gyro/Accelerometer I/O	Failed	FAILED: DVFramePlatform
MRP Switching	Failed	FAILED: DVFramePlatform
Static Bias	Failed	FAILED: DVFramePlatform
Process Noise	Passed	
Discretization	Failed	FAILED: DVFramePlatform
Saturation	Failed	FAILED: DVFramePlatform
Accelerometer Center of Mass Offset	Failed	FAILED: DVFramePlatform
IMU Misalignment	Failed	FAILED: DVFramePlatform
Bias Walk Bounds	Failed	FAILED: DVFramePlatform

Fig. 1 and Fig. 2 show that the random walk remains within bounds for the Bias Walk Bounds tests. The bounds are shown by solid, horizontal blue and green lines.

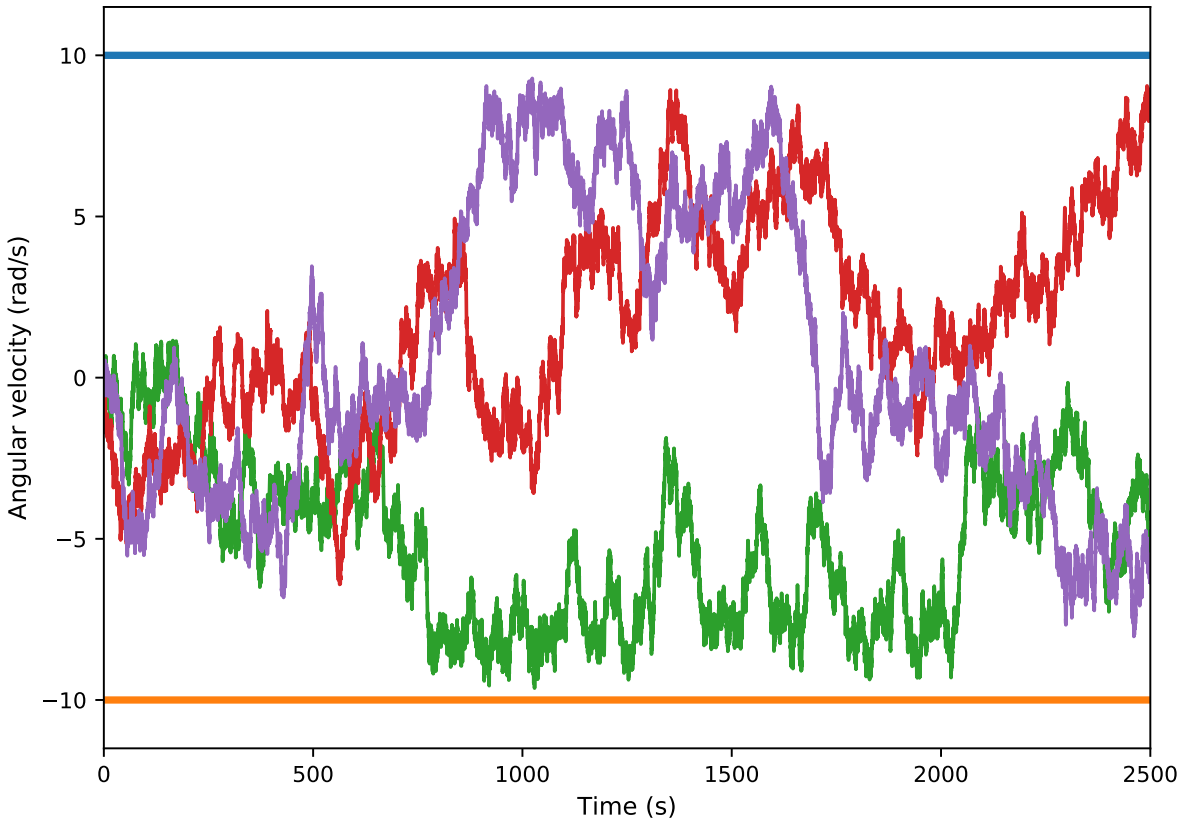

Fig. 1: Module gyro output for random walk bounds check

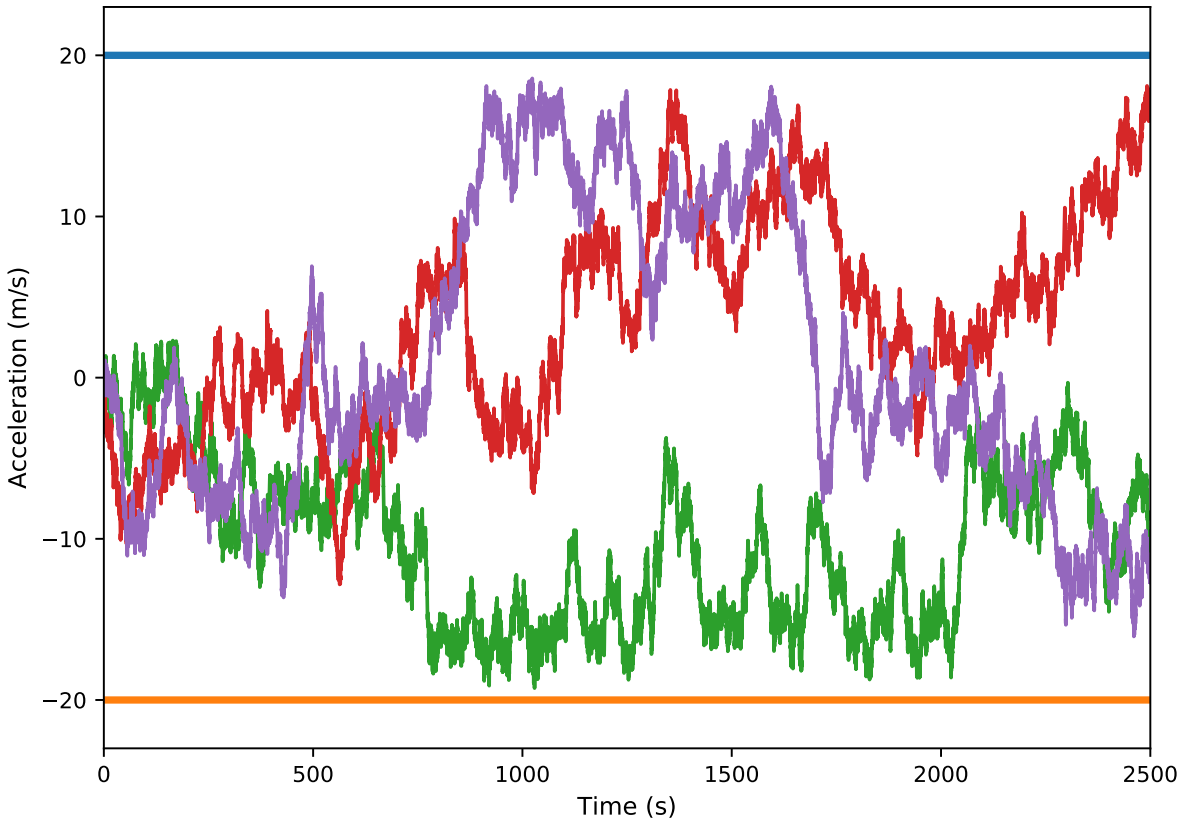

Fig. 2: Module accelerometer output for random walk bounds check

7 User Guide

This section contains conceptual overviews of the code and clear examples for the prospective user.

7.1 Code Diagram

The diagram in Fig. 3 demonstrates the basic logic of the IMU module. There is additional code that deals with auxiliary functions. An example of IMU use is given in test_imu_sensor.py in the imu_sensor_UnitTest folder. Application of each IMU function follows a simple, linear progression until realistic IMU outputs are achieved and sent out via the messaging system.

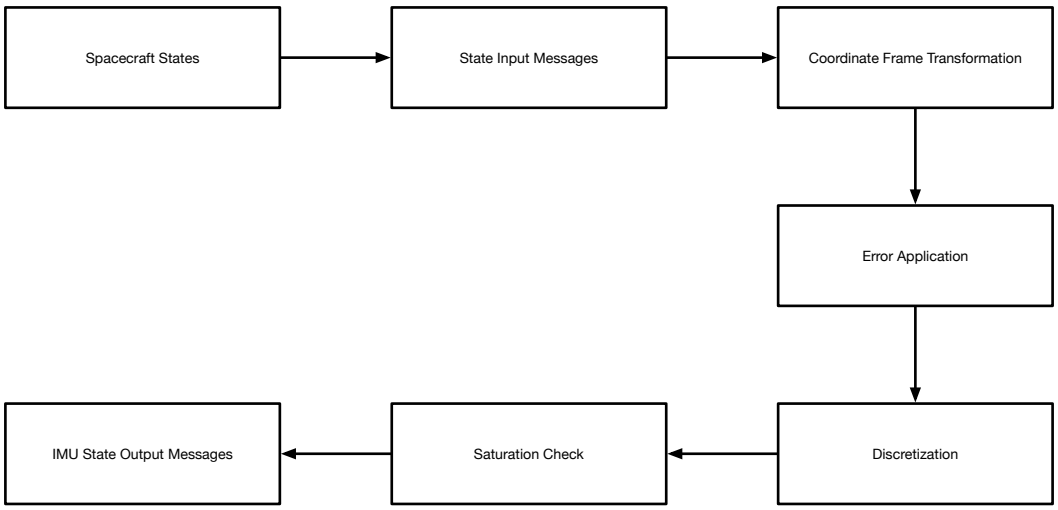

Fig. 3: A pseudo-code diagram demonstrating the flow of inputs and outputs in the IMU module.

7.2 Variable Definitions

The variables in Table 4 are available for user input. Variables used by the module but not available to the user are not mentioned here. Variables with default settings do not necessarily need to be changed by the user, but may be.

Table 4: Definition and Explanation of Variables Used.

Variable	LaTeX Equivalent	Variable Type	Notes
InputStateMsg	N/A	string	Default setting: "inertial_state_output". This is the message from which the IMU receives spacecraft inertial data.
OutputDataMsg	N/A	string	Default setting: "imu_meas_data". This message contains the information output by the IMU.
InputMassMsg	N/A	string	Default setting: "spacecraft_mass_props". This is the message from which the IMU received spacecraft mass data.
SensorPos_B	N/A	double [3]	[m] Required input - no default. This is the sensor position in the body frame relative to the body frame.
roll, pitch, yaw	N/A	double, double, double	Default setting: (0,0,0). To set non-zero initial angles between imu and spacecraft body, call setBodyToPlatformDCM(roll, pitch, yaw)
dcm_PB	N/A	double [3][3]	Default setting: Identity. Setting dcm_PB is equivalent to calling setBodyToPlatformDCM(roll, pitch yaw) above. Use one method or the other.
senRotBias	$\mathbf{e}_{\text{bias}}$	double [3]	[r/s] Default setting: zeros. This is the rotational sensor bias value for each axis.
senTransBias	$\mathbf{e}_{\text{bias}}$	double [3]	[m/s2] Default setting: zeros. This is the translational sensor bias value for each axis
senRotMax	$\mathbf{m}_{\text{sat,max}}$	double	[r/s] Required input - no default. This is the gyro saturation value.
senTransMax	$\mathbf{m}_{\text{sat,max}}$	double	[m/s2] Required input - no default. This is the accelerometer saturation value.
PMatrixAccel	N/A yet	double [3][3]	Default: zeros. This is the covariance matrix used to perturb the state.
PMatrixGyro	N/A yet	double [3][3]	Default: zeros. This is the covariance matrix used to perturb the state.
walkBoundsGyro	N/A yet	double [3]	Default: zeros. This is the "3-sigma" errors to permit for gyro states
walkBoundsAccel	N/A yet	double [3]	Default: zeros. This is the "3-sigma errors ot permit for acceleration states.
accelLSB	(LSB)	double	Default: 0.0. This is the discretization value (least significant bit) for acceleration. Zero indicates no discretization.
gyroLSB	(LSB)	double	Default: 0.0. This is the discretization value (least significant bit) for acceleration. Zero indicates no discretization.