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 Inertial 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 outputs of the IMU.

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
1.2	Rewrote for new IMU implementation and test	S. Carnahan	20170914

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 Platform Frame and Sensor Labels

It will be helpful to note for the following descriptions that the sensor is labeled with a capital S while the sensor platform frame is labeled with a capital P. To be more explicit, There is a coordinate frame, P, the platform frame, in which is the IMU is defined. In all cases so far, the IMU sits at the platform frame origin and its axes are aligned with the platform frame axes. So, any position or velocity vector

describing the sensor also describes the platform frame origin. With that in mind, in this report, it has been attempted to track the kinematics of the sensor, while reporting values in the platform frame, rather than tracking the kinematics of the platform form.

1.1.2 Frame Depedent Derivatives

Inertial time derivatives are marked with a dot (i.e. $\frac{\mathcal{N}d}{dt}x = \dot{x}$). Body frame time derivatives are marked with a prime (i.e. $\frac{\mathcal{B}d}{dt}x = x'$)

1.1.3 Angular Rates

The angular rate of the sensor in platform frame coordinates is output as:

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

Where $\mathcal{P}$ is the sensor platform frame, $\mathcal{B}$ is the vehicle body frame, and $\mathcal{N}$ is the inertial frame. [PB] is the direction cosine matrix from $\mathcal{B}$ to $\mathcal{P}$. This allows for an arbitrary angular offset between $\mathcal{B}$ and $\mathcal{P}$ and allows for that offset to be time-varying. $^{\mathcal{B}}\boldsymbol{\omega}_{B/N}$ is provided by the spacecraftPlus output message from the most recent dynamics integration.

1.1.4 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. Once the new MRP is received, both are converted to DCMs and the step-PRV is computed as follows The current MRP is always provided by the spacecraftPlus message from the most recent dynamics integration.

$$[PN]_2 = [PB][BN]_2 \tag{2}$$

$$[PN]_1 = [PB][BN]_1 \tag{3}$$

$$[NP]_1 = [PN]_1^T \tag{4}$$

$$[P_2P_1] = [PN]_2[NP]_1 \tag{5}$$

$$\boldsymbol{q} = \texttt{C2PRV}([P_2P_1]) \tag{6}$$

where $\boldsymbol{q}$ above is the principal rotation vector for the sensor from timestep 1 to timestep 2. The functions used in conversion from the DCM to PRV are part of the Basilisk Rigid Body Kinematics library. The double conversion is used to avoid singularities.

1.1.5 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:

$$\boldsymbol{r}_{S/N} = \boldsymbol{r}_{B/N} + \boldsymbol{r}_{S/B} \tag{7}$$

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

$$\dot{\boldsymbol{r}}_{S/N} = \dot{\boldsymbol{r}}_{B/N} + \boldsymbol{r}'_{S/B} + \boldsymbol{\omega}_{B/N} \times \boldsymbol{r}_{S/B} \tag{8}$$

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

$$\ddot{\boldsymbol{r}}_{S/N} = \ddot{\boldsymbol{r}}_{B/N} + \dot{\boldsymbol{\omega}}_{B/N} \times \boldsymbol{r}_{S/B} + \boldsymbol{\omega}_{B/N} \times (\boldsymbol{\omega}_{B/N} \times \boldsymbol{r}_{S/B}) \tag{9}$$

The equation above is 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:

$$\ddot{\boldsymbol{r}}_{S/N,\text{sensed}} = (\ddot{\boldsymbol{r}}_{B/N} - \boldsymbol{a}_{\text{g}}) + \dot{\boldsymbol{\omega}}_{B/N} \times \boldsymbol{r}_{S/B} + \boldsymbol{\omega}_{B/N} \times (\boldsymbol{\omega}_{B/N} \times \boldsymbol{r}_{S/B}) \tag{10}$$

where $\boldsymbol{a}_{\text{g}}$ is the instantaneous acceleration due to gravity. Conveniently, $(\ddot{\boldsymbol{r}}_{B/N} - \boldsymbol{a}_{\text{g}})$ is available from the spacecraft, but in the body frame. The acceleration provided, though, is the time-averaged acceleration between the last two dynamics integration calls and not the instantaneous acceleration. $\boldsymbol{r}_{S/B}$ is also available in the body frame. $\dot{\boldsymbol{\omega}}_{B/N}$ is given by the spacecraft in body frame coordinates as well. Again, this is a time-averaged value output by the spacecraft, rather than an instantaneous value. Because all values are given in the body frame, the above equation is calulated in the body frame and then converted as seen below:

$$^{\mathcal{P}}\ddot{\boldsymbol{r}}_{S/N,\text{sensed}} = [PB]\,^{\mathcal{B}}\ddot{\boldsymbol{r}}_{S/N,\text{sensed}} \tag{11}$$

1.1.6 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. 8 with itself from time 1 to time 2 gives the equation:

$$\Delta_{2/1}\dot{\boldsymbol{r}}_{S/N} = \Delta_{2/1}\dot{\boldsymbol{r}}_{B/N} + \Delta_{2/1}(\boldsymbol{\omega}_{B/N} \times \boldsymbol{r}_{S/B}) \tag{12}$$

$\Delta_{2/1}\dot{\boldsymbol{r}}_{B/N}$ 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:

$$\Delta_{2/1}\dot{\boldsymbol{r}}_{B/N} = DV_{\text{body\_non-conservative},2} - DV_{\text{body\_non-conservative},1} \tag{13}$$

The above DV values are given by the spacecraft module in body frame coordinates but used in inertial coordinates. They are computed by accumulating the velocity after each dynamics integration and subtracting out the time-averaged gravitational acceleration multiplied by the dynamics time step. Then,

$$\Delta_{2/1}(\boldsymbol{\omega}_{B/N} \times \boldsymbol{r}_{S/B}) = \boldsymbol{\omega}_{B/N_2} \times \boldsymbol{r}_{S/B_2} - \boldsymbol{\omega}_{B/N_1} \times \boldsymbol{r}_{S/B_1} \tag{14}$$

$\boldsymbol{\omega}_{B/N}$ output by the spacecraft is the angular rate from the most recent dynamics integration. Again, the above equation is calculated in the inertial frame and then converted to platform frame coordinates. this means that the values given by spacecraft plus are first converted into inertial frame coordinates, including the location of the sensor in the body frame. At this point, Eq. 12 is evaluated in the body frame and converted to the sensor platform frame:

$$^{\mathcal{P}}\Delta_{2/1}\dot{\boldsymbol{r}}_{S/N} = [PN]\,^{\mathcal{N}}\Delta_{2/1}\dot{\boldsymbol{r}}_{S/N} \tag{15}$$

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. To be clear, this is the sensed inertial velocity change in platform frame coordinates. This makes the assumption that the IMU is tracking its attitude and performing the calculations internally to return this value correctly. It is not simply the integral of the acceleration value above in the body frame coordinates.

1.1.7 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 acceleration and angular velocity:

$$\mathbf{a}_{\text{measured}} = \mathbf{a}_{\text{truth}} + \mathbf{e}_{\text{a,noise}} + \mathbf{e}_{\text{a,bias}} \tag{16}$$

$$\boldsymbol{\omega}_{\text{measured}} = \boldsymbol{\omega}_{\text{truth}} + \mathbf{e}_{\omega,\text{noise}} + \mathbf{e}_{\omega,\text{bias}} \tag{17}$$

Then, these error values are "integrated" over the IMU timestep and applied to the Δv and PRV values:

$$\mathbf{DV}_{\text{measured}} = \mathbf{DV}_{\text{truth}} + (\mathbf{e}_{\text{a,noise}} + \mathbf{e}_{\text{a,bias}})\Delta t \tag{18}$$

$$\mathbf{q}_{\text{measured}} = \mathbf{q}_{\text{truth}} + (\mathbf{e}_{\omega,\text{noise}} + \mathbf{e}_{\omega,\text{bias}})\Delta t \tag{19}$$

This convenient approximation that $\boldsymbol{q} = \boldsymbol{\omega}\Delta t$ for a given timestep proves useful through the application of IMU errors.

1.1.8 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 simulate 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 in the following way:

$$\mathbf{a}_{\text{discretized}} = \texttt{sign}(\mathbf{a}_{\text{measured}})(\text{LSB})\left\lfloor \left| \frac{\mathbf{a}_{\text{measured}}}{(\text{LSB})} \right| \right\rfloor \tag{20}$$

$$\mathbf{e}_{\text{d,a}} = \mathbf{a}_{\text{measured}} - \mathbf{a}_{\text{discretized}} \tag{21}$$

$$\mathbf{DV}_{\text{discretized}} = \mathbf{DV}_{\text{measured}} - \mathbf{e}_{\text{d,a}}\Delta t \tag{22}$$

$$\boldsymbol{\omega}_{\text{discretized}} = \texttt{sign}(\boldsymbol{\omega}_{\mathbf{measured}})(\text{LSB})\left\lfloor \left| \frac{\boldsymbol{\omega}_{\text{measured}}}{(\text{LSB})} \right| \right\rfloor \tag{23}$$

$$\mathbf{e}_{\text{d},\omega} = \boldsymbol{\omega}_{\text{measured}} - \boldsymbol{\omega}_{\text{discretized}} \tag{24}$$

$$\mathbf{q}_{\text{discretized}} = \mathbf{q}_{\text{measured}} - \mathbf{e}_{\text{d},\omega}\Delta t \tag{25}$$

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

1.1.9 Saturation

Real sensors can also become saturated. Saturation is the last effect implemented on the IMU, *in an elementwise manner*:

$$\boldsymbol{a}_{\text{sat}} = \max\big(a_{\min}, \min\big(\boldsymbol{a}_{\text{discretized}}, a_{\max}\big)\big) \tag{26}$$

$$\boldsymbol{\omega}_{\text{sat}} = \max\big(\omega_{\min}, \min\big(\boldsymbol{\omega}_{\text{discretized}}, \omega_{\max}\big)\big) \tag{27}$$

The above operations are performed element-wise. This is only computed if the values are found to be outside of the max-min range. Now, along each axis that was saturated:

$$DV_{\text{sat},i} = a_{\text{sat},i}\Delta t \tag{28}$$

$$q_{\text{sat},i} = \omega_{\text{sat},i}\Delta t \tag{29}$$

The above is calculated any time that a_i or ω_i are found to be outside of their max-min bounds. Note again the use of the approximation of the PRV as the integral of the angular rates.

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, sensor resolution) 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.
- **Scale Factor**: The code can apply a scale factor, different for each axis of acceleration and angular rate, to the measured value (truth + noise + bias). This is a linear scaling of the output.
- **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.

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 well-known assumptions and is generally accepted to be a good model for this application.
- **Error Integration**: Errors for integrated values (DV and PRV) are calculated as acceleration and angular velocity errors multiplied by the IMU time step. If the IMU timestep matches the dynamics process rate, this is possibly a good assumption. However, if the IMU is run slower than the dynamics process, then the velocity errors may not be related to the instantaneous acceleration errors at the sampling time.
- **Integral Saturation**: Because the DV and PRV output values are calculated only at the IMU time step and not actually by integrating rates multiple times between calls, their saturation values are taken as the time integral of the rate saturation values. This misses some possibilities with varying accelerations between IMU time steps. Furthermore, the PRV is taken to be the integral of the angular rate over the time step. This should be a good approximation if the attitude of the spacecraft doesn't change "too much" over a relevant time step.

- **IMU Rate Limitation**: As with a real IMU, this model will only be run at a finite speed. It is limited in that it cannot correctly capture dynamics that are happening much faster than the IMU is called.

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:

4.1 Primary Test Method

In order to thoroughly test arbitrary outputs from the IMU, The equations of motion for the sensor were formulated as functions of the center of mass of the spacecraft as opposed to the equations seen in the model description which were formulated about the body frame. The truth values for the following test are set up in the following way:

$$\boldsymbol{r}_{S/N} = \boldsymbol{r}_{C/N} + \boldsymbol{r}_{S/C} \tag{30}$$

$$\dot{\boldsymbol{r}}_{S/N} = \dot{\boldsymbol{r}}_{C/N} + \boldsymbol{r}'_{S/C} + \boldsymbol{\omega}_{B/N} \times \boldsymbol{r}_{S/C} \tag{31}$$

Knowing that:

$$\boldsymbol{r}'_{S/C} = -\boldsymbol{c}' \tag{32}$$

Eq. 31 becomes:

$$\dot{\boldsymbol{r}}_{S/N} = \dot{\boldsymbol{r}}_{C/N} - \boldsymbol{c}' + \boldsymbol{\omega}_{B/N} \times \boldsymbol{r}_{S/C} \tag{33}$$

So, again substituting in the center of mass velocity in the $\mathcal{B}$ frame,

$$\ddot{\boldsymbol{r}}_{S/N} = \ddot{\boldsymbol{r}}_{C/N} - \boldsymbol{c}'' - 2\boldsymbol{\omega}_{B/N} \times \boldsymbol{c}' + \dot{\boldsymbol{\omega}}_{B/N} \times \boldsymbol{r}_{S/C} + \boldsymbol{\omega}_{B/N} \times \boldsymbol{\omega}_{B/N} \times \boldsymbol{r}_{S/C} \tag{34}$$

$\boldsymbol{c}''$ and $\boldsymbol{c}'$ can be solved for in terms of $\ddot{\boldsymbol{c}}$ and $\dot{\boldsymbol{c}}$ using the transport theorem:

$$\boldsymbol{c}' = \dot{\boldsymbol{c}} - \boldsymbol{\omega}_{B/N} \times \boldsymbol{c} \tag{35}$$

$$\boldsymbol{c}'' = \ddot{\boldsymbol{c}} - 2\boldsymbol{\omega}_{B/N} \times \boldsymbol{c}' - \dot{\boldsymbol{\omega}}_{B/N} \times \boldsymbol{c} - \boldsymbol{\omega}_{B/N} \times \boldsymbol{\omega}_{B/N} \times \boldsymbol{c} \tag{36}$$

Now, given the states at a previous time step, t_{n-1}, and the accelerations (linear and angular), new states can be calculated for t_n:

$$\Delta t = t_n - t_{n-1} \tag{37}$$

$$\dot{\boldsymbol{r}}_{B/N,n} = \dot{\boldsymbol{r}}_{B/N,n-1} + \frac{\ddot{\boldsymbol{r}}_{B/N,n-1} + \ddot{\boldsymbol{r}}_{B/N,n}}{2}\Delta t \tag{38}$$

$$\boldsymbol{r}_{B/N,n} = \boldsymbol{r}_{B/N,n-1} + \frac{\dot{\boldsymbol{r}}_{B/N,n-1} + \dot{\boldsymbol{r}}_{B/N,n}}{2}\Delta t \tag{39}$$

$$\boldsymbol{\omega}_{B/N,n} = \boldsymbol{\omega}_{B/N,n-1} + \frac{\dot{\boldsymbol{\omega}}_{B/N,n-1} + \dot{\boldsymbol{\omega}}_{B/N,n}}{2}\Delta t \tag{40}$$

$$[\boldsymbol{B}] = (1-\sigma^2)[I_{3x3}] + 2[\tilde{\boldsymbol{\sigma}}] + 2\boldsymbol{\sigma}\boldsymbol{\sigma}^T \tag{41}$$

$$\dot{\boldsymbol{\sigma}} = \frac{1}{4}[\boldsymbol{B}]^{\mathcal{B}}\boldsymbol{\omega} \tag{42}$$

$$\boldsymbol{\sigma}_{B/N,n} = \boldsymbol{\sigma}_{B/N,n-1} + \frac{\dot{\boldsymbol{\sigma}}_{B/N,n-1} + \dot{\boldsymbol{\sigma}}_{B/N,n}}{2}\Delta t \tag{43}$$

The same can then be done for the position $\boldsymbol{r}_{C/N}$ with its derivatives. Also, knowing that:

$$\ddot{\boldsymbol{c}} = \ddot{\boldsymbol{r}}_{C/N} - \ddot{\boldsymbol{r}}_{B/N} \tag{44}$$

the same can be done for $\boldsymbol{c}$ and its derivatives. All of this numerical integration must be done in the inertial frame.

At this point, all of the information needed to solve for Eq. 34 is known. Additionally, the delta-v accumulated between t_{n-1} and t_n can be added to the total delta-v

4.2 Test Descriptions

1. Clean The IMU is run with all clean inputs, i.e. nonzero accelerations and angular accelerations of the spacecraft and this is compared to the truth values generated in python. No noise, discretization, saturation, etc. is applied.

   Success Criteria: The outputs match to acceptable tolerance and are visually confirmed.

2. Noise The IMU is run with inputs as in the clean test, i.e. nonzero accelerations and angular accelerations of the spacecraft and this is compared to the truth values generated in python. Gaussian noise and random walk are applied.

   Success Criteria: The output standard deviations match the inputs to acceptable tolerance.

3. Bias The IMU is run with all clean inputs, i.e. nonzero accelerations and angular accelerations of the spacecraft and this is compared to the truth values generated in python. Bias is then added.

   Success Criteria: The outputs match to acceptable tolerance and are visually confirmed to include bias.

4. Saturation The IMU is run with all clean inputs, i.e. nonzero accelerations and angular accelerations of the spacecraft and this is compared to the truth values generated in python. Out of bounds values are floored or ceilinged.

   Success Criteria: The outputs match to acceptable tolerance and are visually confirmed to be capped.

5. Discretization The IMU is run with all clean inputs, i.e. nonzero accelerations and angular accelerations of the spacecraft and this is compared to the truth values generated in python. Outputs are discretized.

   Success Criteria: The outputs match to acceptable tolerance and are visually confirmed to be discretized. Note. Two points in time always fail this test. This has to do with the python generated and c++ generated values being ever-so-slightly off and not discretizing at the same point. They match at the next timesteps and have been ignored for the test.

As an additional check, $[PB]$ is calculated separately for the truth values and yaw, $pitch$, and $roll$ are fed to the IMU which calculates this value independently. In this way, the multiple set-up options for the IMU 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. Note that tolerances are relative $\frac{truth-output}{truth}$

Test	Tolerance	GyroLSB	AccelLSB	RotMax	TransMax	RotNoise	TransNoise	RotBias	TransBias
Clean	1e-08	0e+00	0e+00	1.0e+03	1.0e+03	0.0	0.0	0.0e+00	0.0e+00
Noise	1e-01	0e+00	0e+00	1.0e+03	1.0e+03	0.1	0.1	0.0e+00	0.0e+00
Bias	1e-08	0e+00	0e+00	1.0e+03	1.0e+03	0.0	0.0	1.0e+01	1.0e+01
Sat.	1e-08	0e+00	0e+00	1.0e+00	5.0e+00	0.0	0.0	0.0e+00	0.0e+00
Disc.	1e-08	5e-02	5e-01	1.0e+02	1.0e+03	0.0	0.0	0.0e+00	0.0e+00

For all tests, the gyro has a scale factor of 1 applied to each axis while the accelerometer has a scale factor of two. This functionality is easily verified in the Noise test, which has 2x the standard deviation that was given only for the linear outputs.

6 Test Results

All checks within test_imu_sensor.py passed as expected. Table 3 shows the test results. The figures below the table show that the truth values matched the output values for all values checked.

Table 3: Test results

Test	Pass/Fail
Clean	PASSED
Noise	PASSED
Bias	PASSED
Sat.	PASSED
Disc.	PASSED

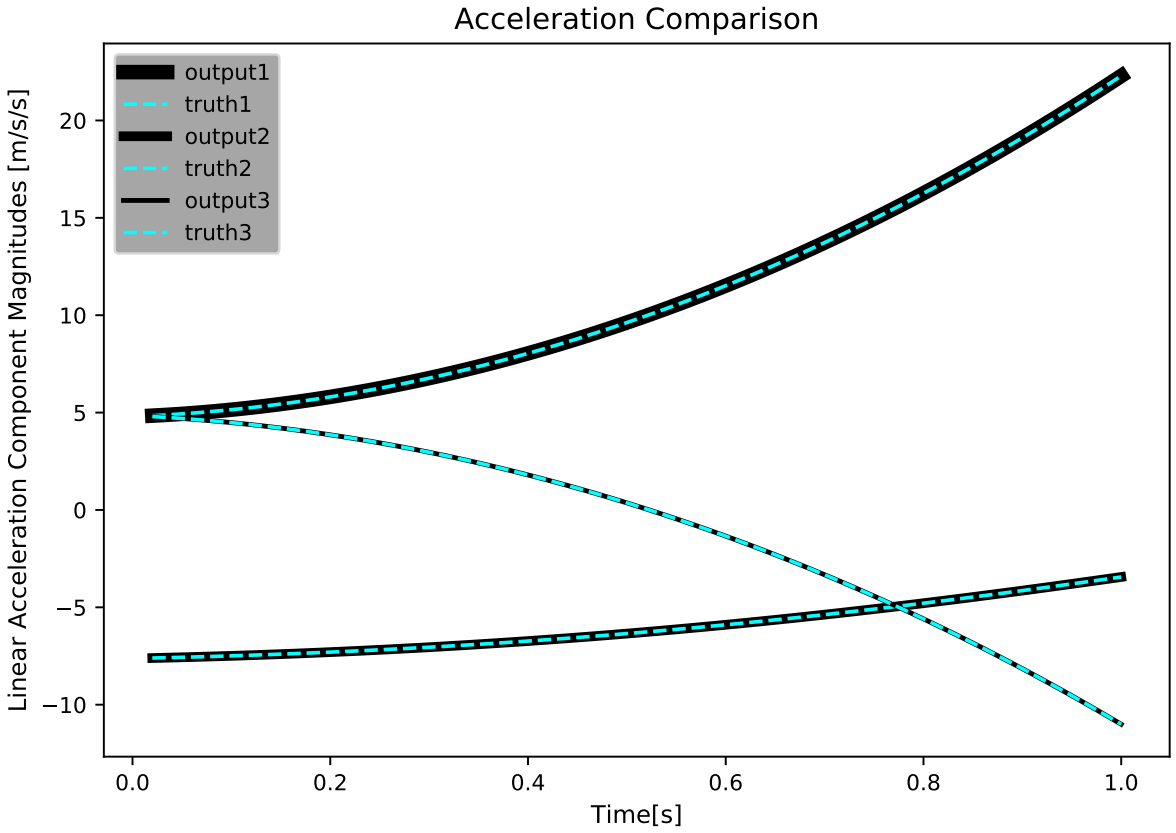

Fig. 1: Plot Comparing Sensor Linear Accelertaion Truth and Output for test: clean. Note that 1, 2, and 3 indicate the components of the acceleration.

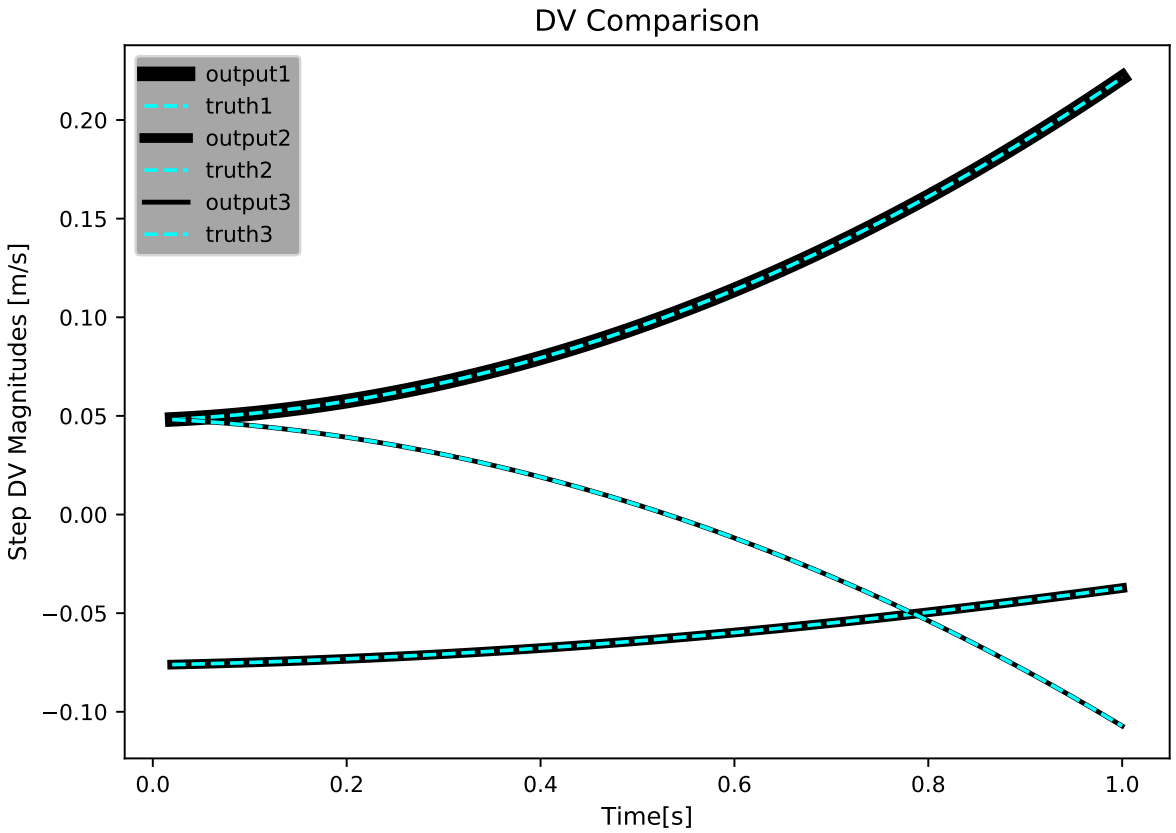

Fig. 2: Plot Comparing Time Step DV Truth and Output for test: clean. Note that 1, 2, and 3 indicate the components of the velocity delta.

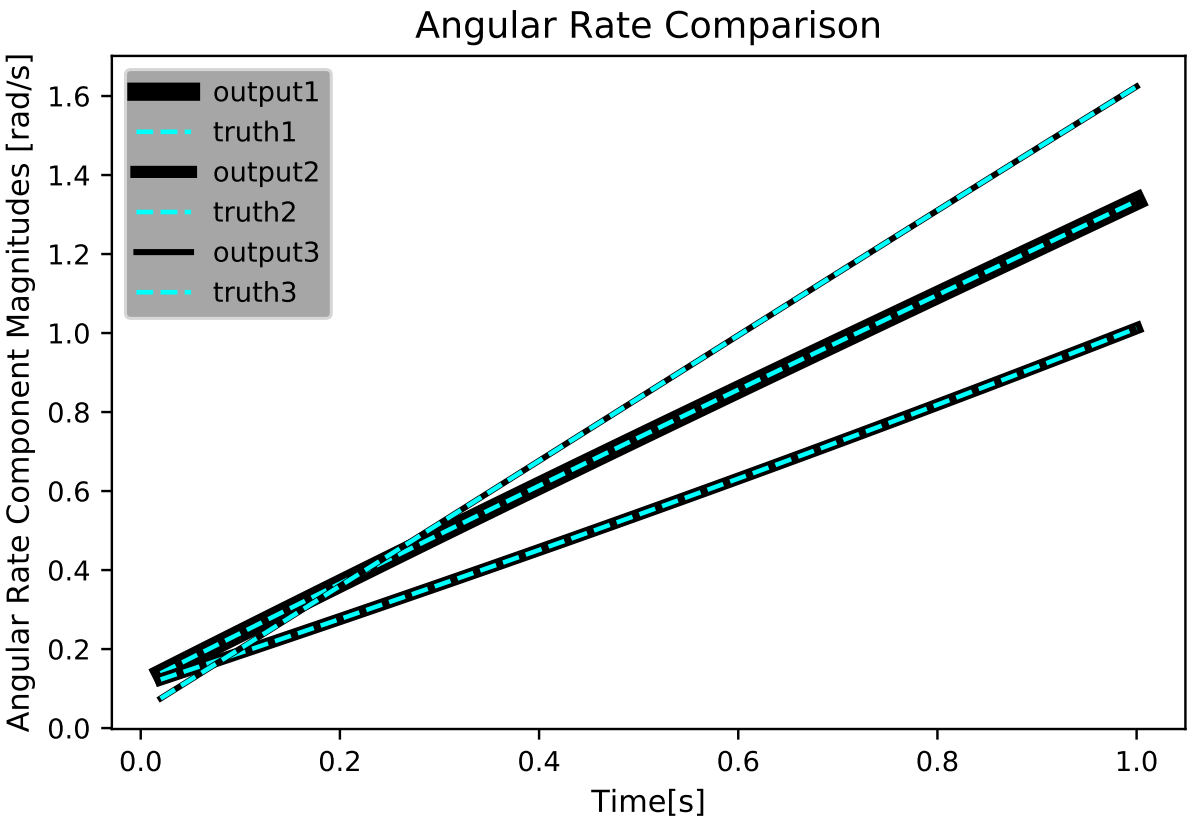

Fig. 3: Plot Comparing Angular Rate Truth and Output for test: clean. Note that 1, 2, and 3 indicate the components of the angular rate.

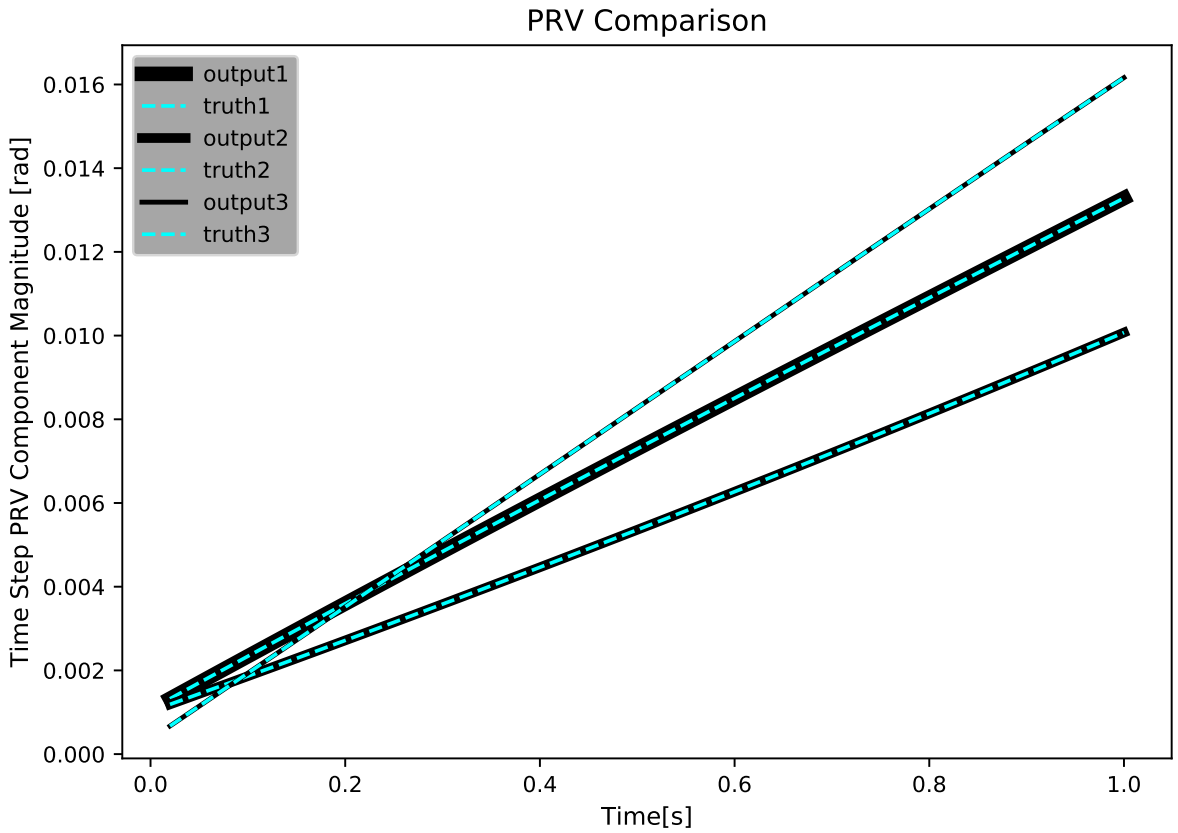

Fig. 4: Plot Comparing Time Step PRV Truth and Output for test: clean. Note that 1, 2, and 3 indicate the components of the principal rotation vector.

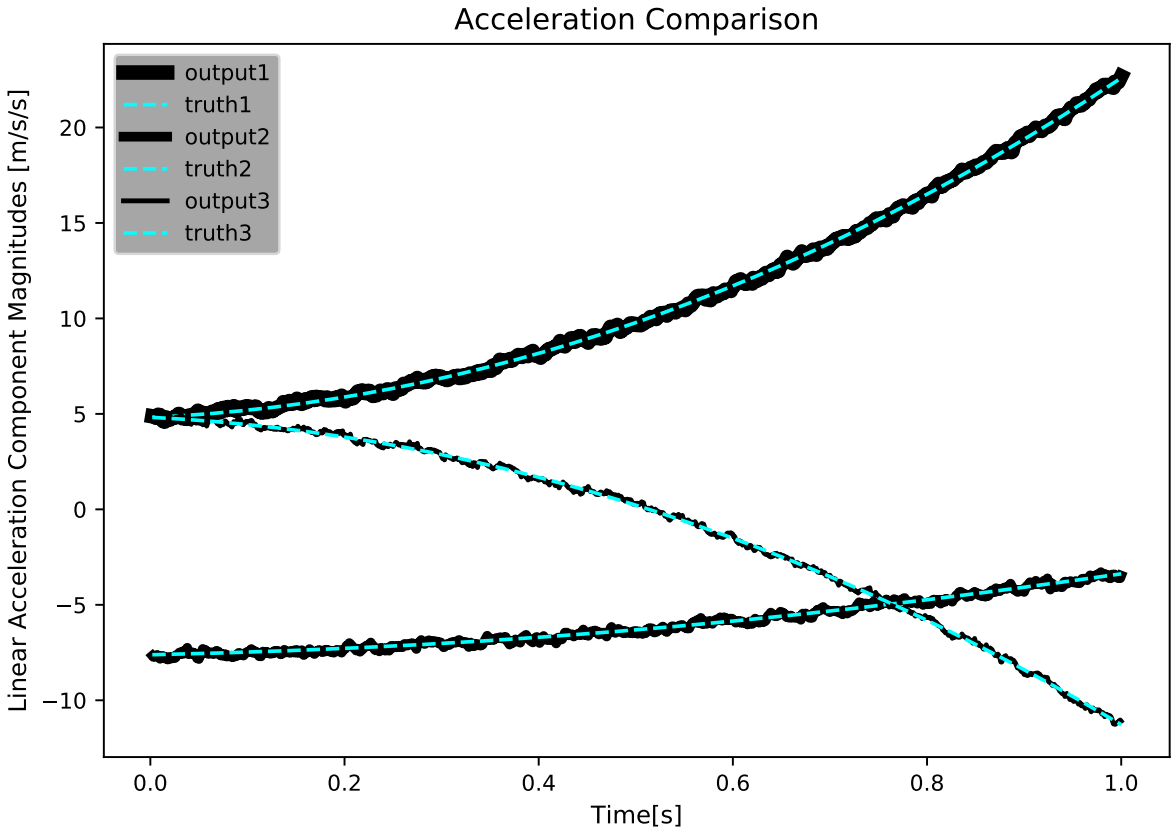

Fig. 5: Plot Comparing Sensor Linear Accelertaion Truth and Output for test: noise. Note that 1, 2, and 3 indicate the components of the acceleration.

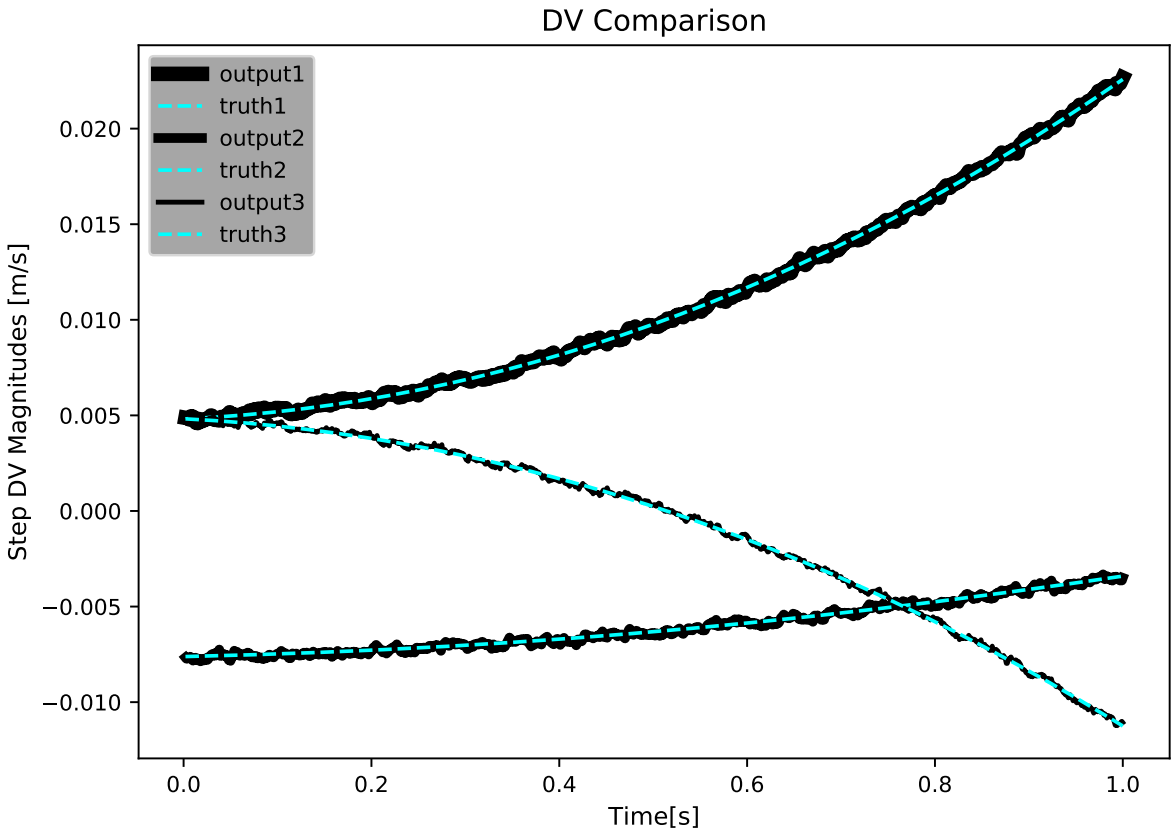

Fig. 6: Plot Comparing Time Step DV Truth and Output for test: noise. Note that 1, 2, and 3 indicate the components of the velocity delta.

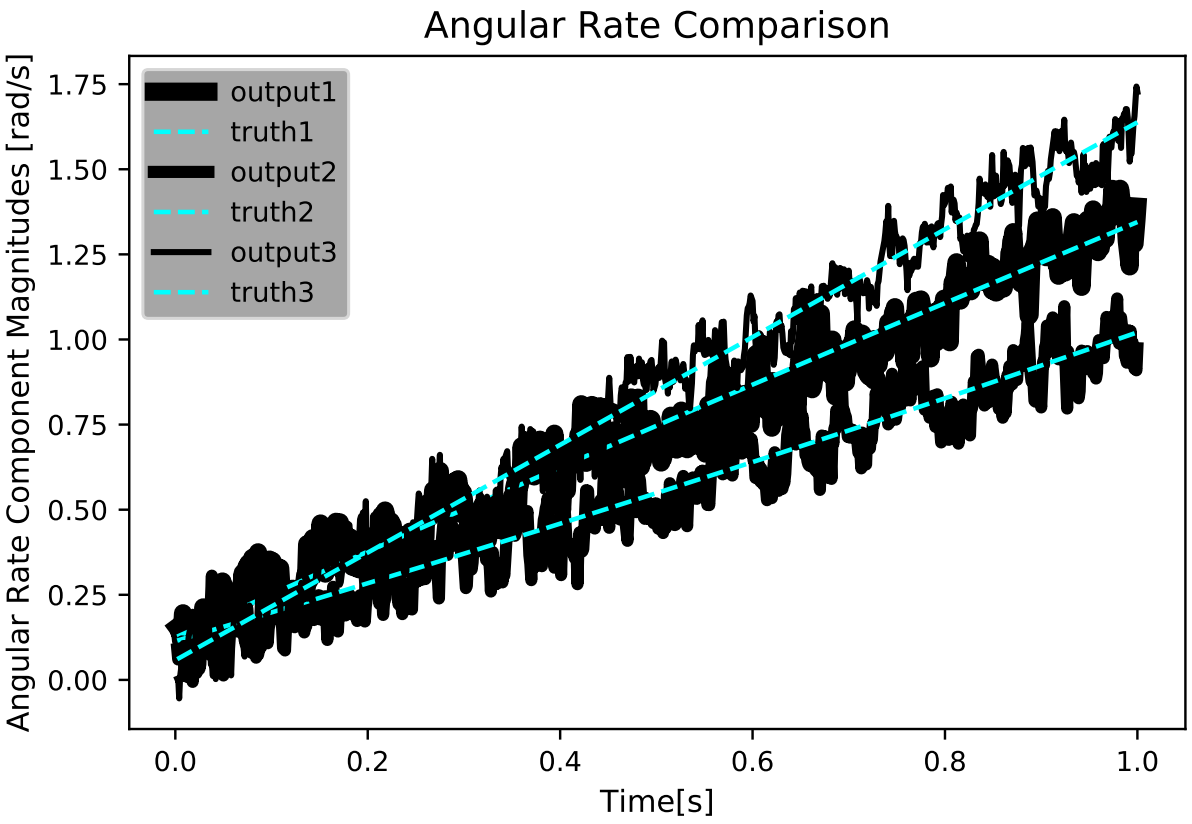

Fig. 7: Plot Comparing Angular Rate Truth and Output for test: noise. Note that 1, 2, and 3 indicate the components of the angular rate.

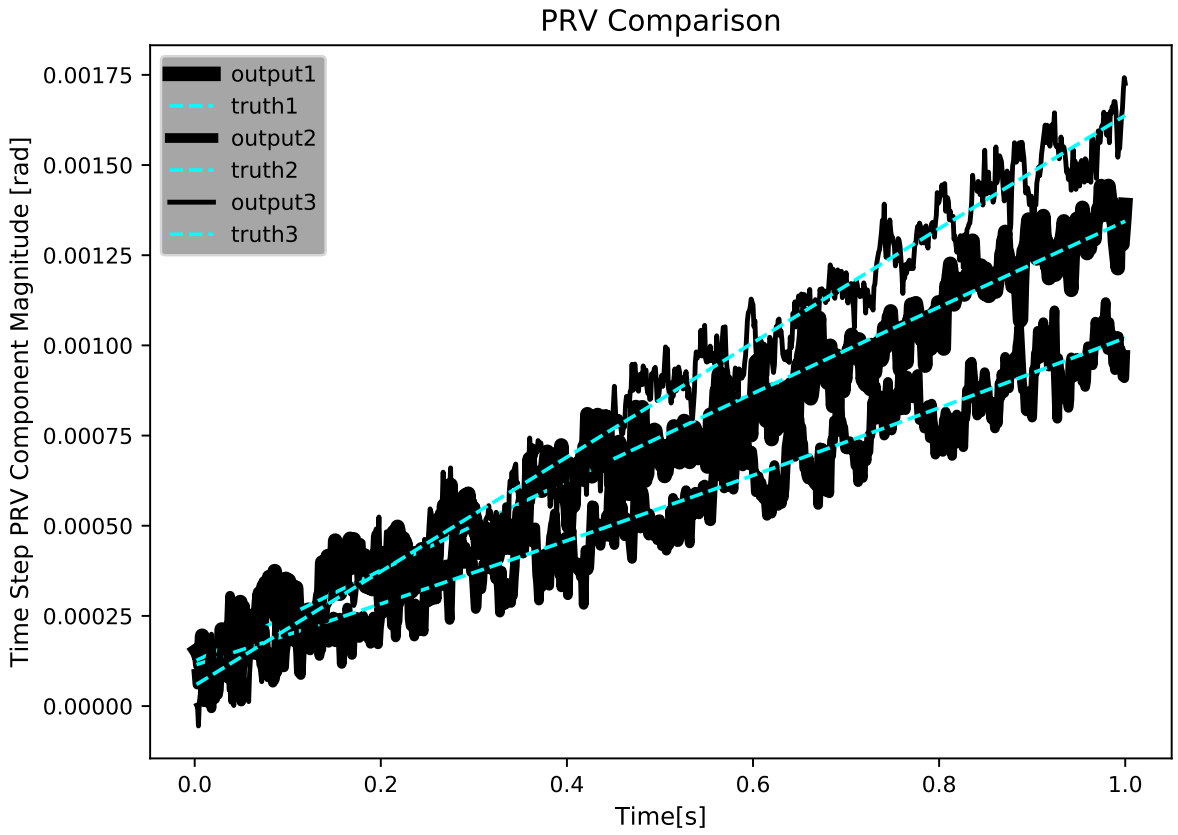

Fig. 8: Plot Comparing Time Step PRV Truth and Output for test: noise. Note that 1, 2, and 3 indicate the components of the principal rotation vector.

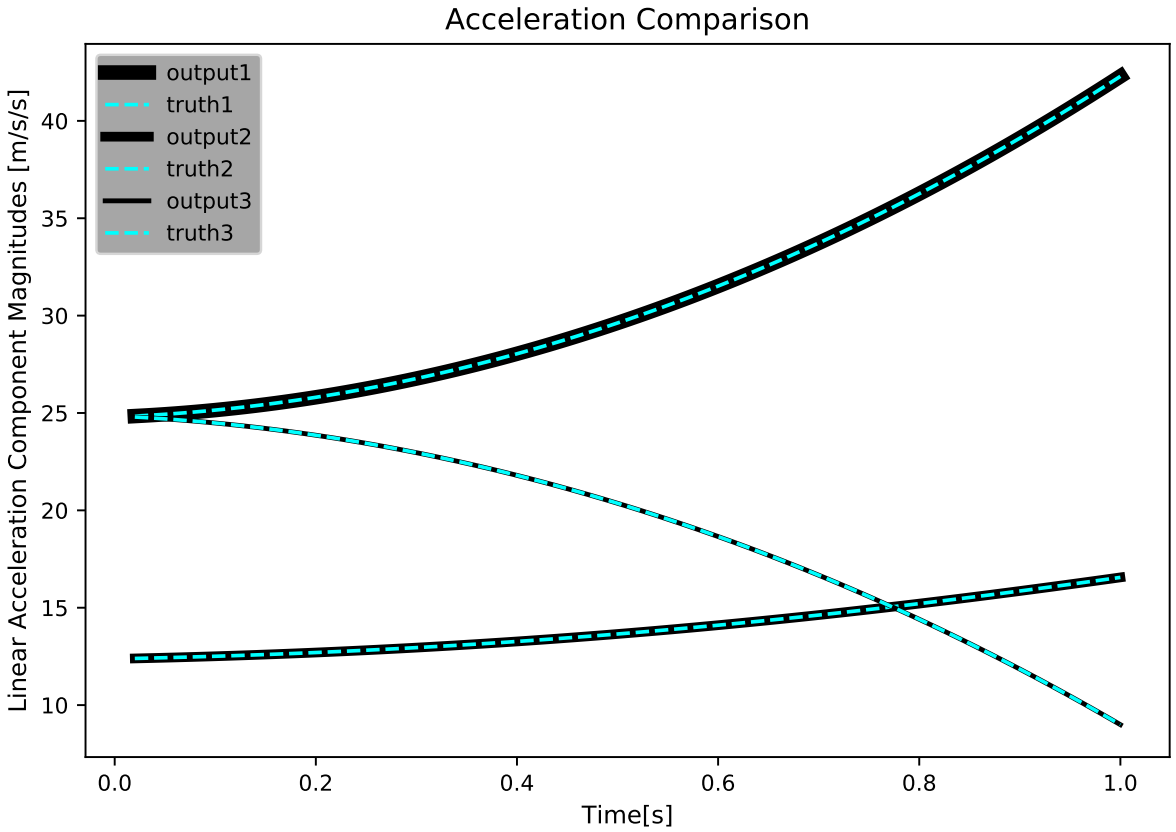

Fig. 9: Plot Comparing Sensor Linear Accelertaion Truth and Output for test: bias. Note that 1, 2, and 3 indicate the components of the acceleration.

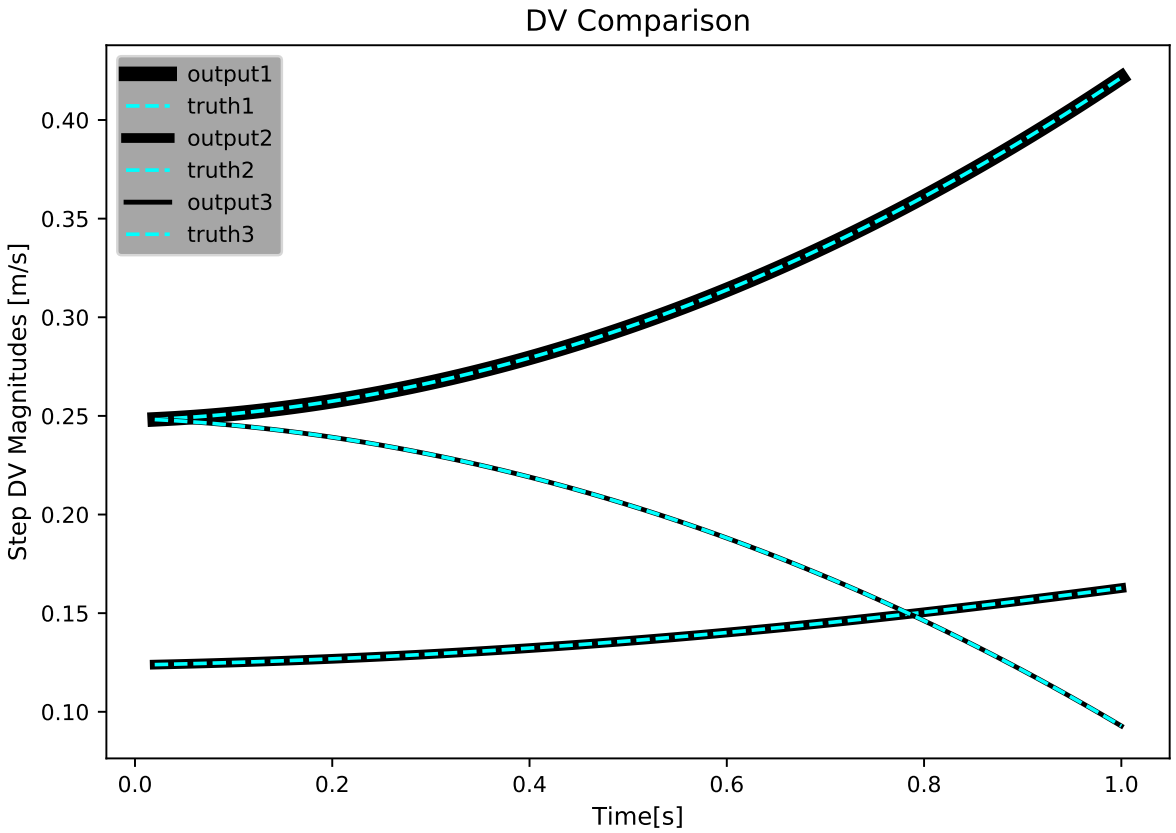

Fig. 10: Plot Comparing Time Step DV Truth and Output for test: bias. Note that 1, 2, and 3 indicate the components of the velocity delta.

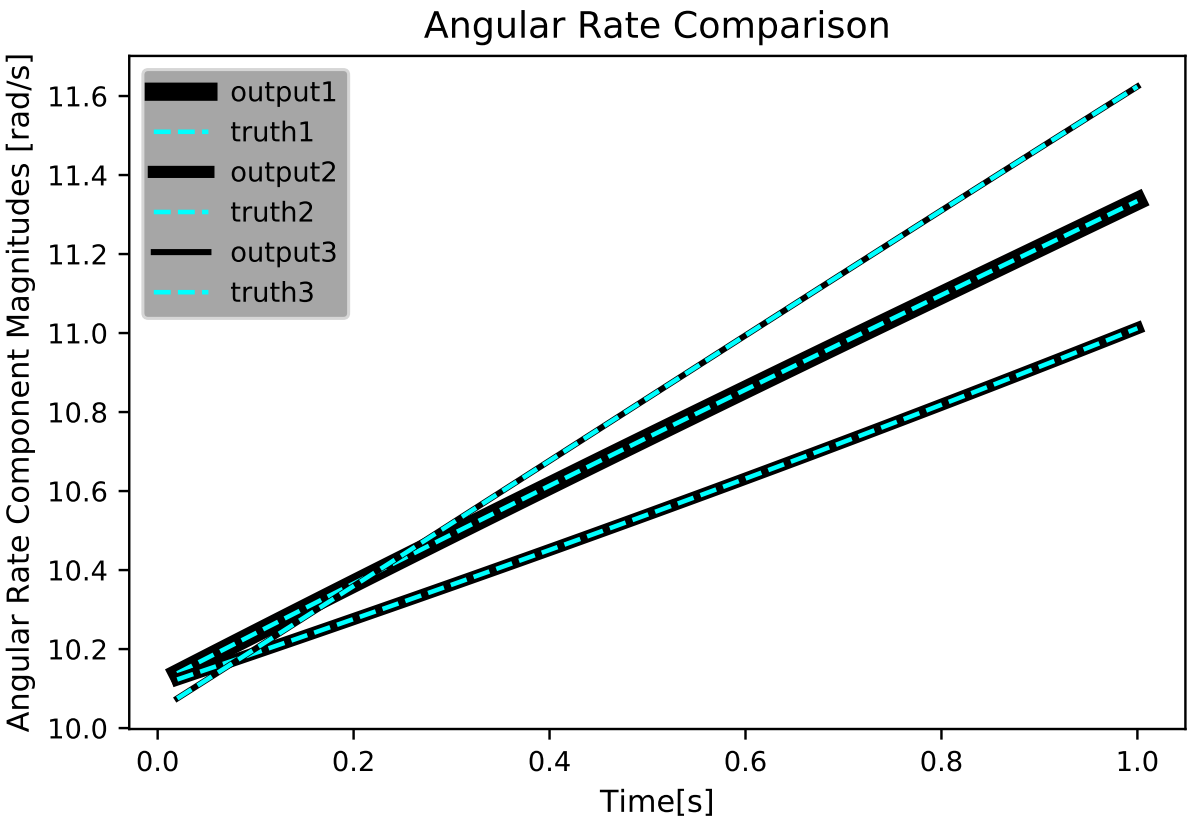

Fig. 11: Plot Comparing Angular Rate Truth and Output for test: bias. Note that 1, 2, and 3 indicate the components of the angular rate.

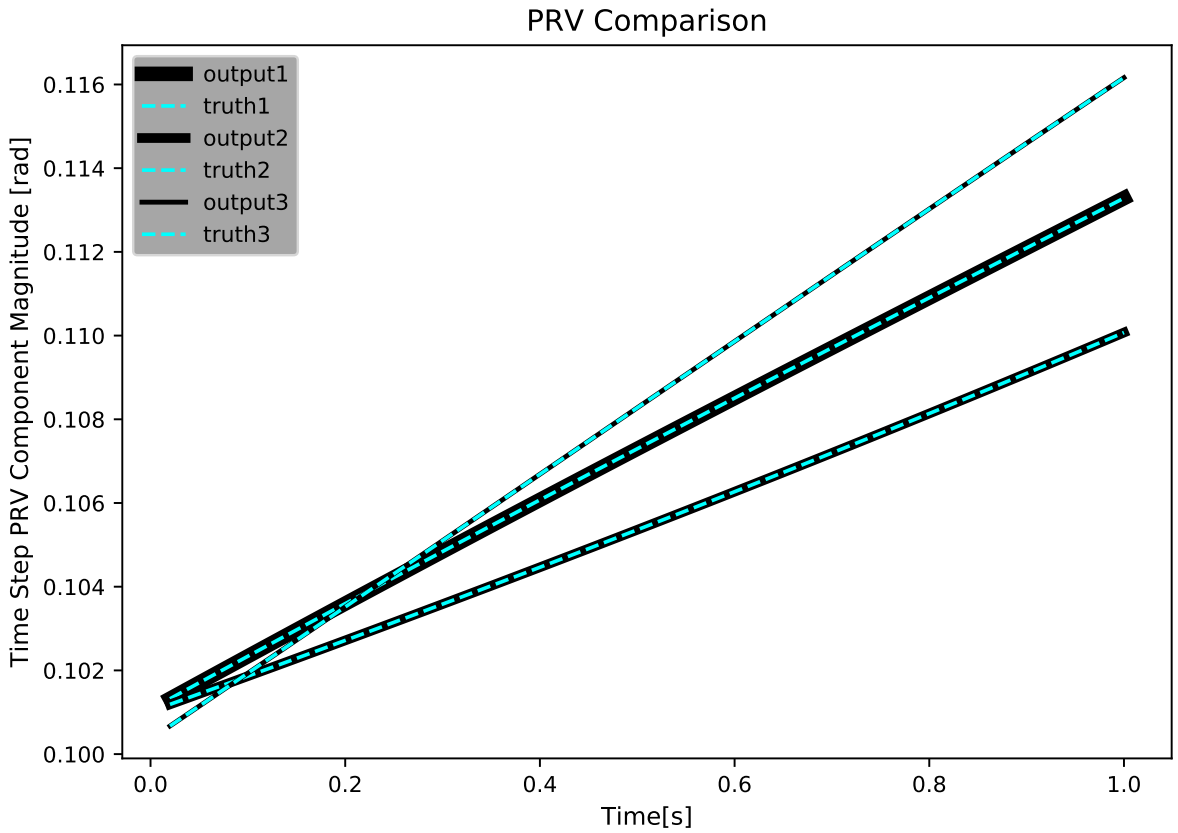

Fig. 12: Plot Comparing Time Step PRV Truth and Output for test: bias. Note that 1, 2, and 3 indicate the components of the principal rotation vector.

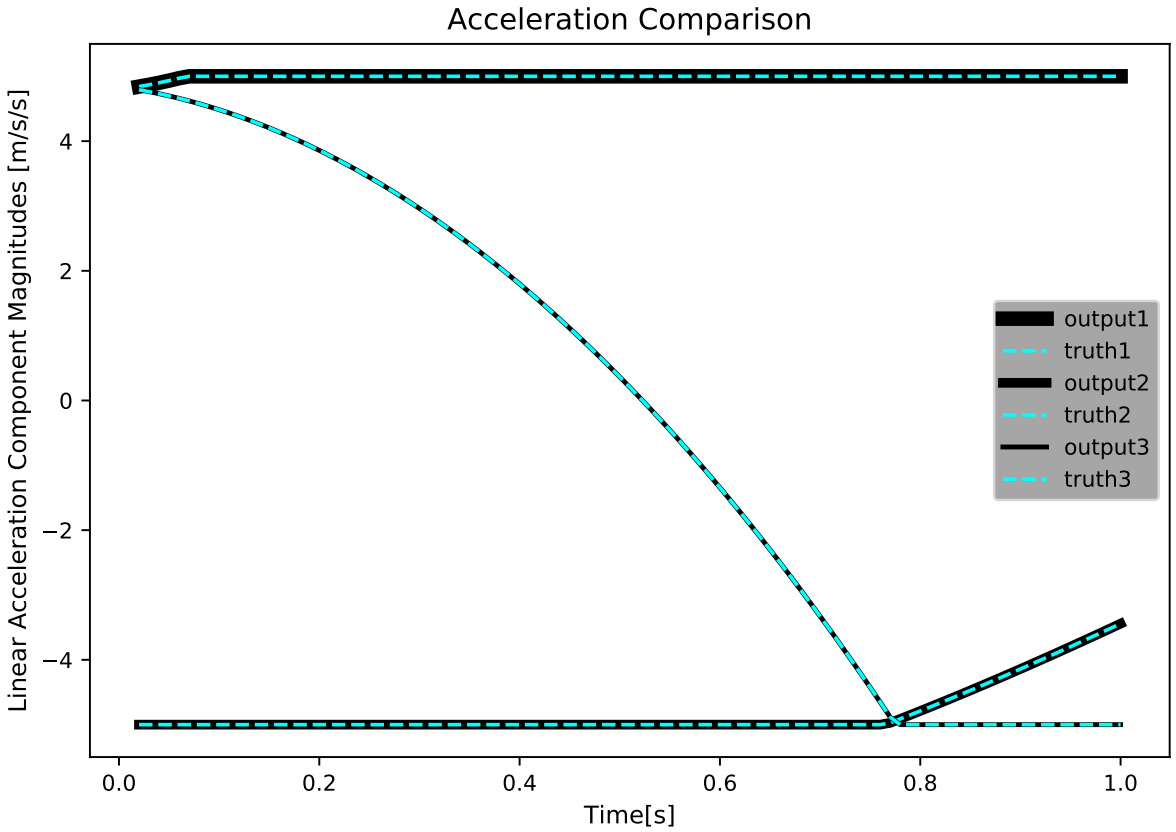

Fig. 13: Plot Comparing Sensor Linear Accelertaion Truth and Output for test: saturation. Note that 1, 2, and 3 indicate the components of the acceleration.

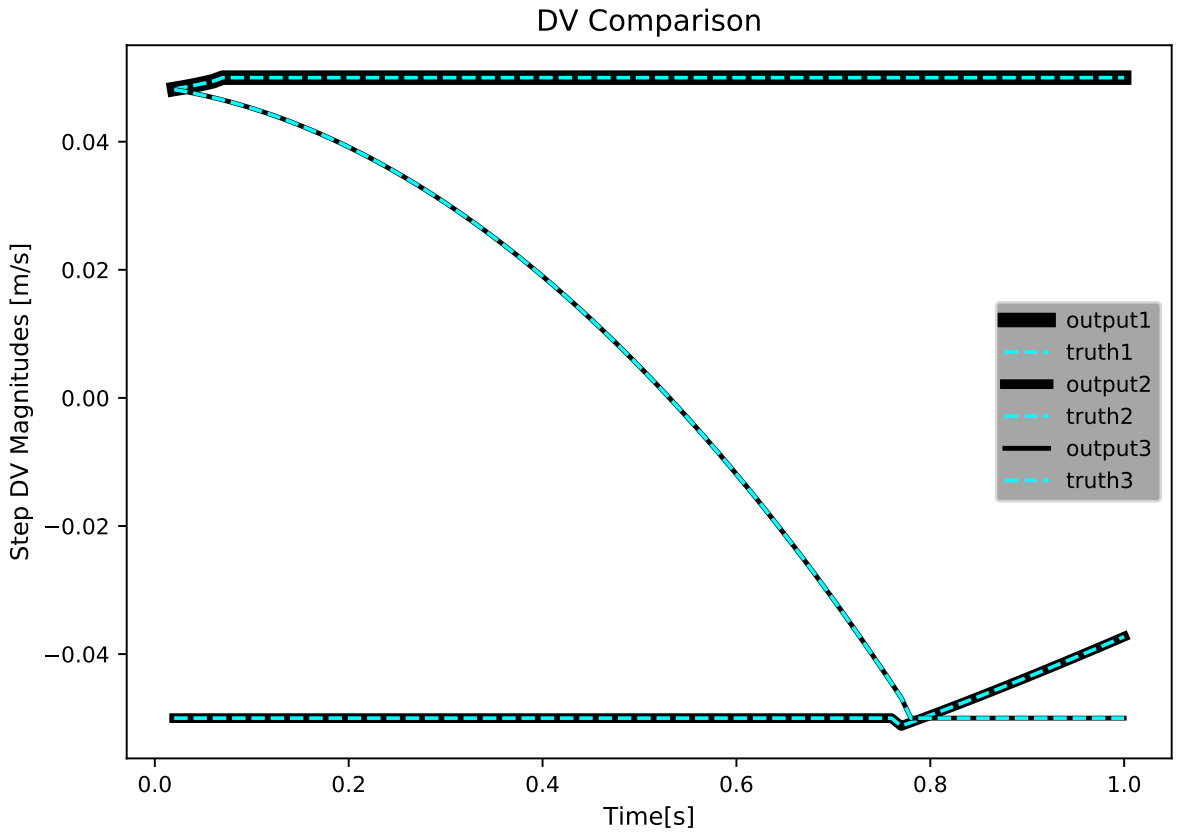

Fig. 14: Plot Comparing Time Step DV Truth and Output for test: saturation. Note that 1, 2, and 3 indicate the components of the velocity delta.

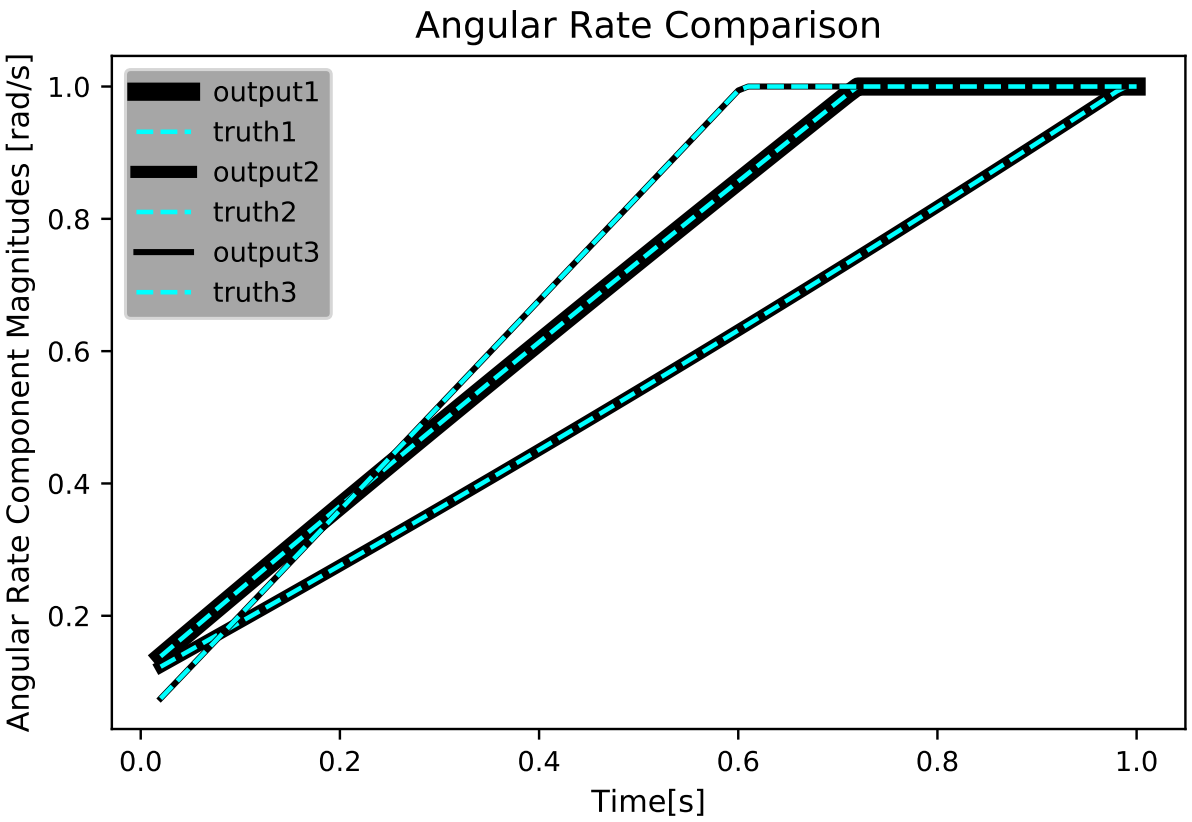

Fig. 15: Plot Comparing Angular Rate Truth and Output for test: saturation. Note that 1, 2, and 3 indicate the components of the angular rate.

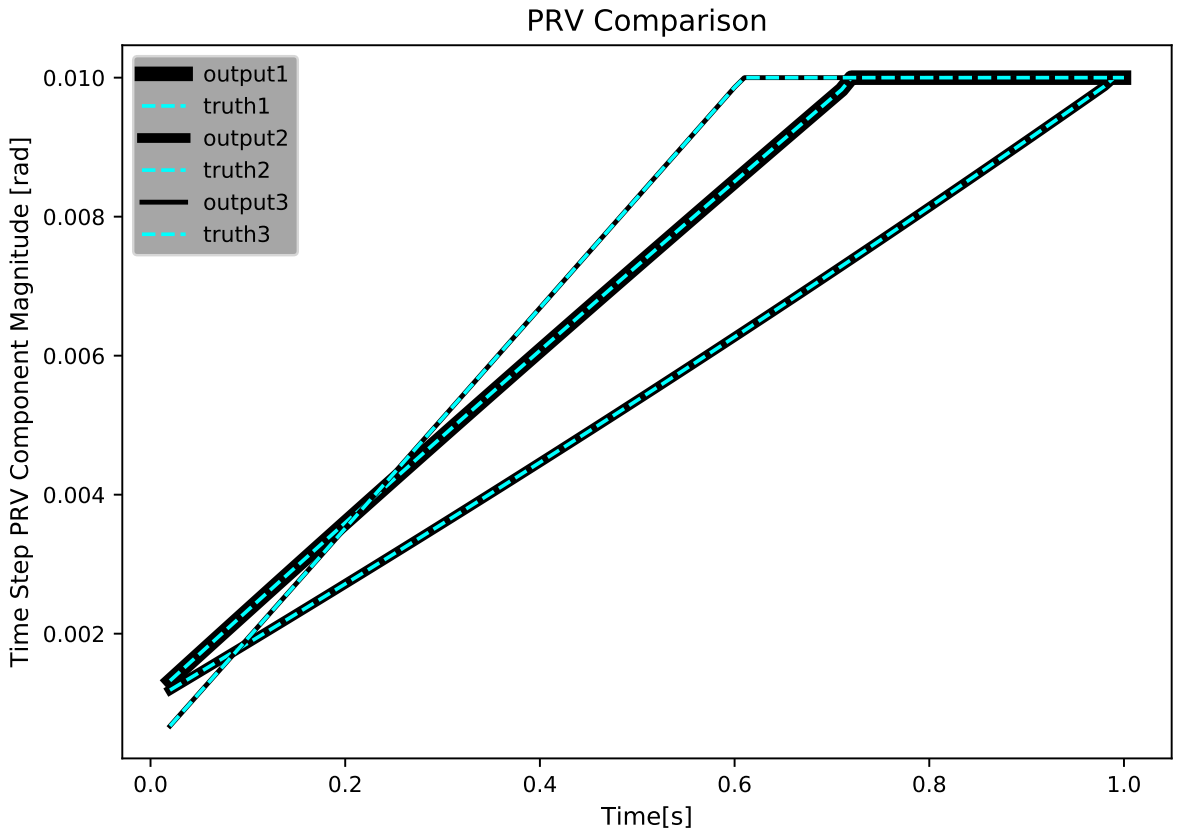

Fig. 16: Plot Comparing Time Step PRV Truth and Output for test: saturation. Note that 1, 2, and 3 indicate the components of the principal rotation vector.

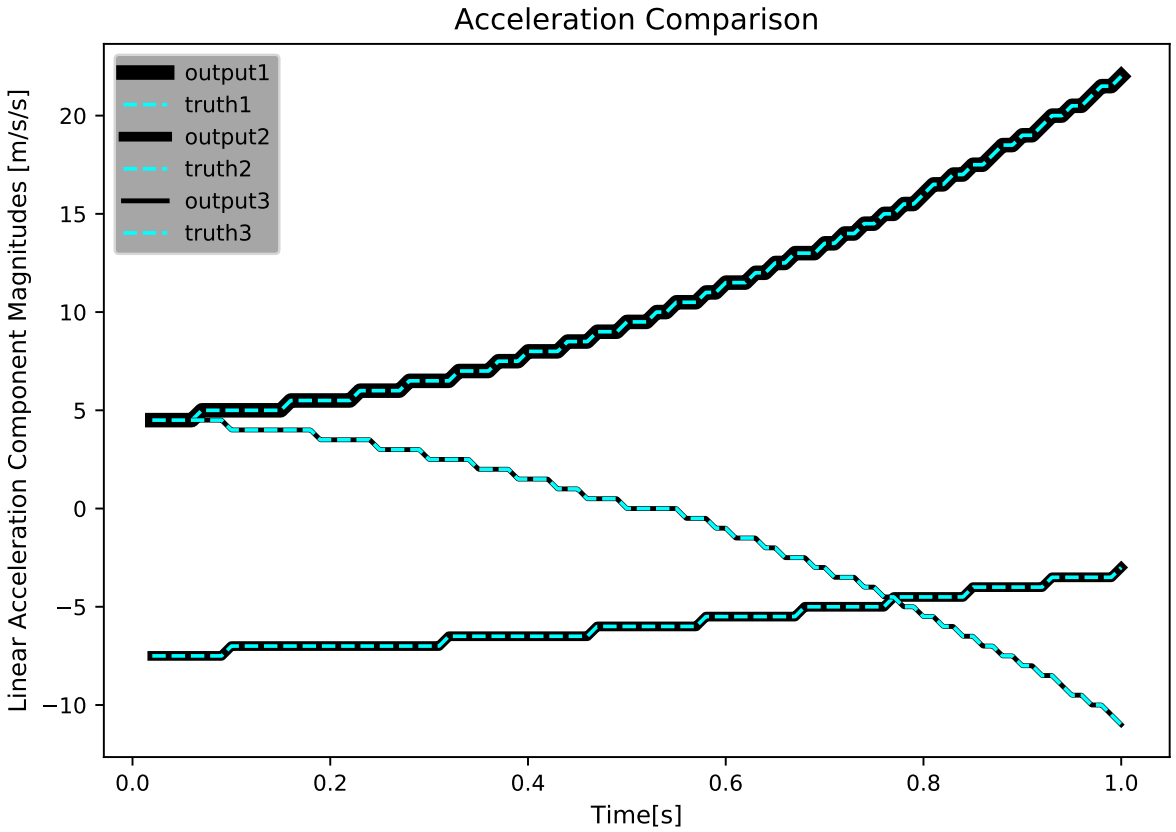

Fig. 17: Plot Comparing Sensor Linear Accelertaion Truth and Output for test: discretization. Note that 1, 2, and 3 indicate the components of the acceleration.

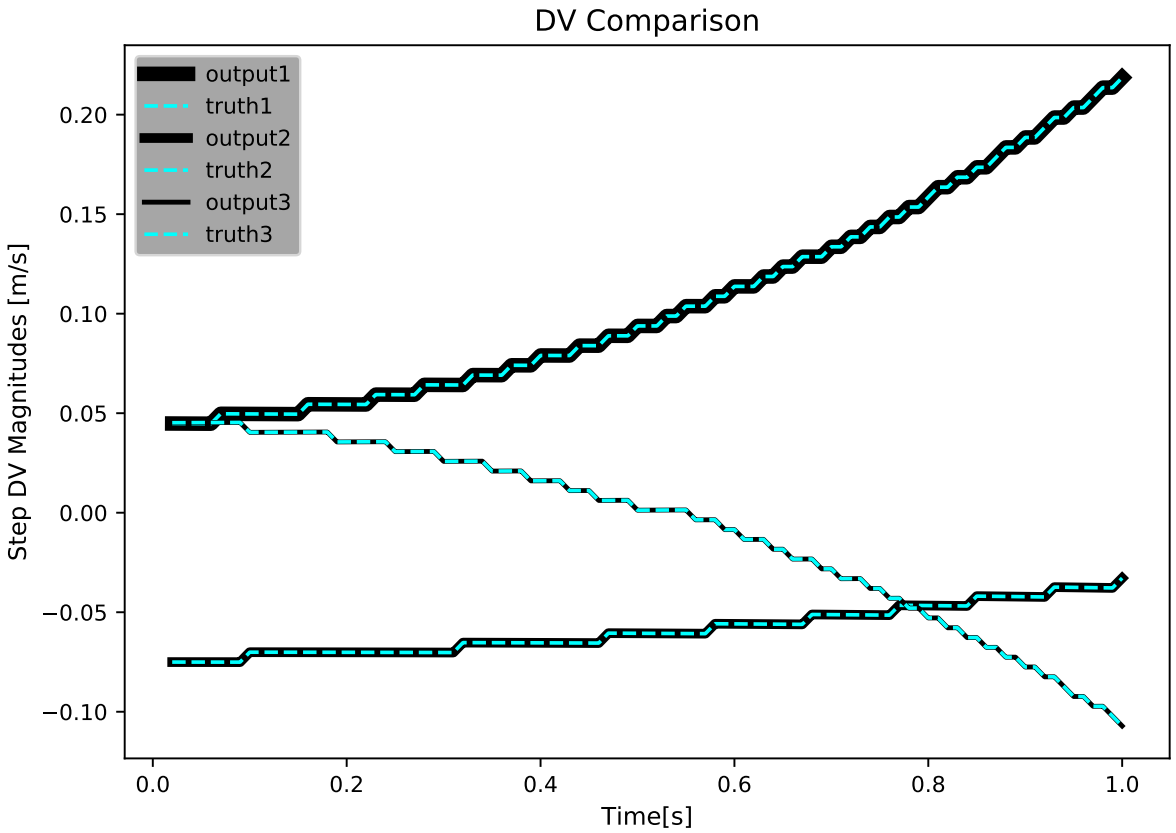

Fig. 18: Plot Comparing Time Step DV Truth and Output for test: discretization. Note that 1, 2, and 3 indicate the components of the velocity delta.

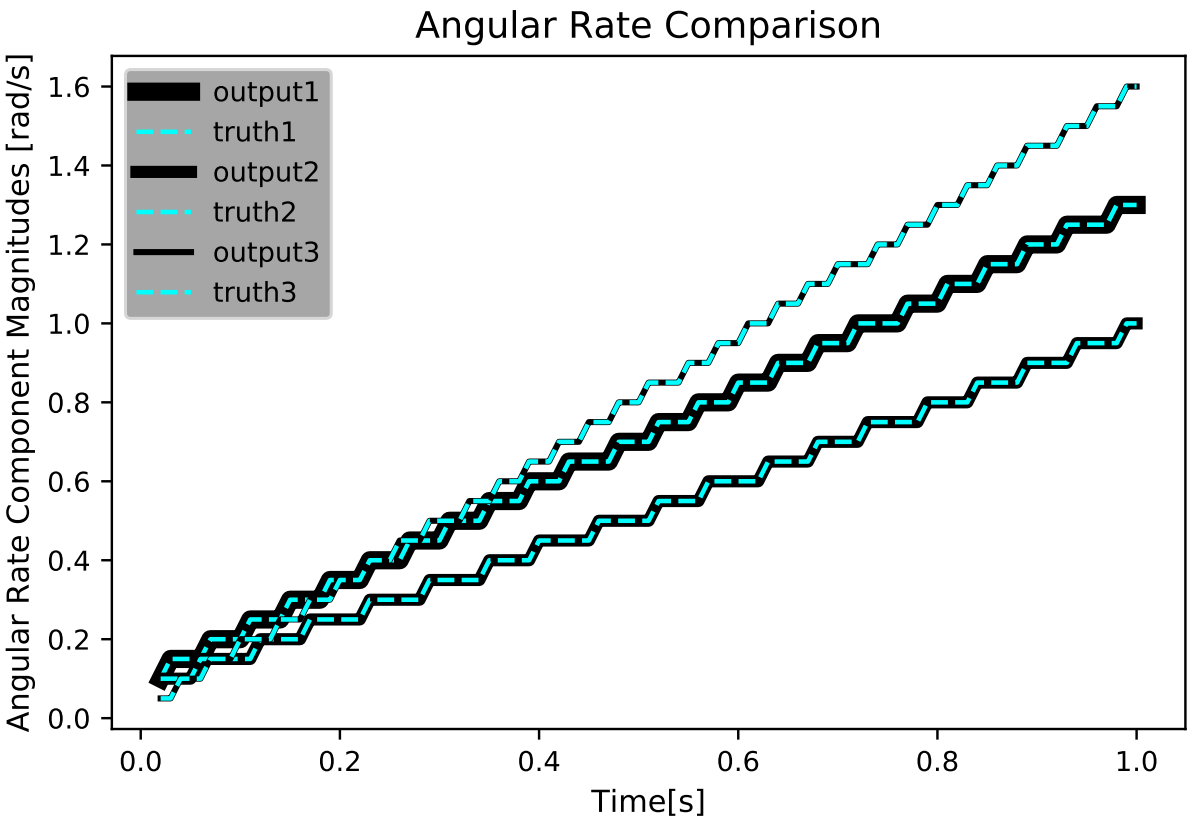

Fig. 19: Plot Comparing Angular Rate Truth and Output for test: discretization. Note that 1, 2, and 3 indicate the components of the angular rate.

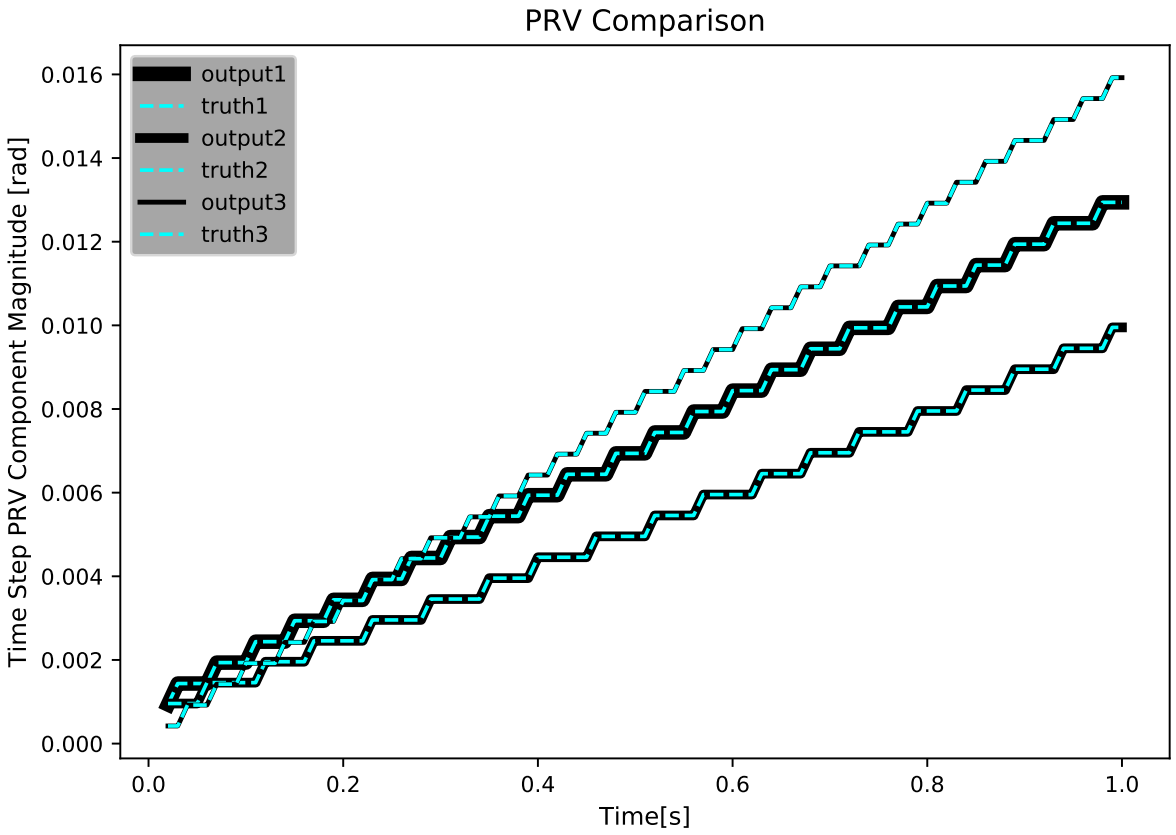

Fig. 20: Plot Comparing Time Step PRV Truth and Output for test: discretization. Note that 1, 2, and 3 indicate the components of the principal rotation vector.

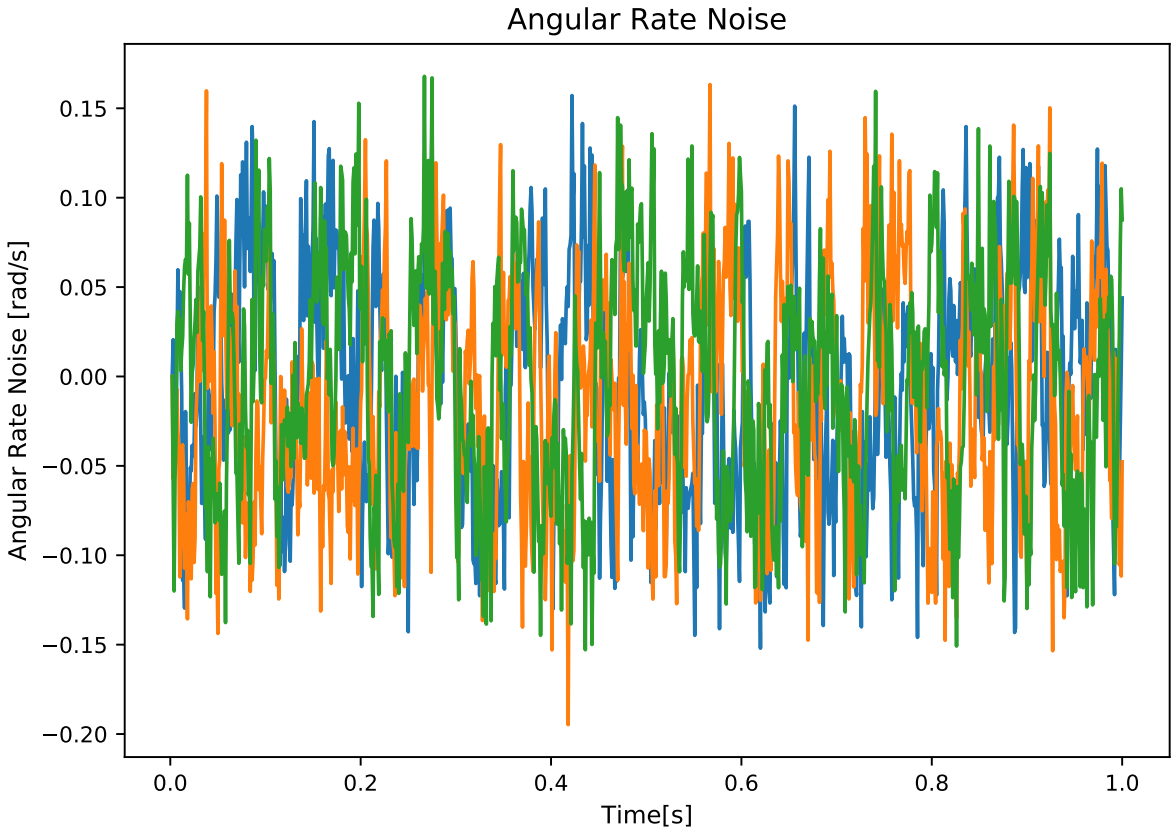

Fig. 21: Plot of Angular Rate noise along each component for the noise test.

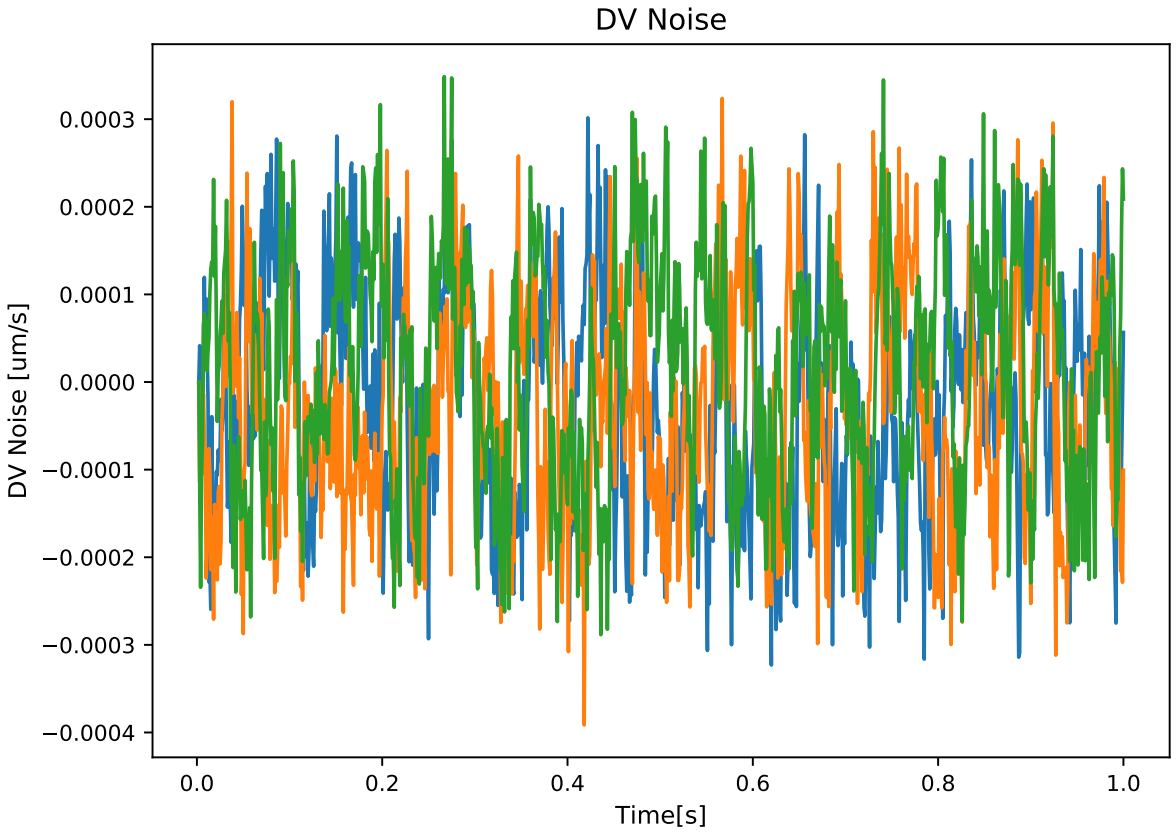

Fig. 22: Plot of DeltaV noise along each component for the noise test.

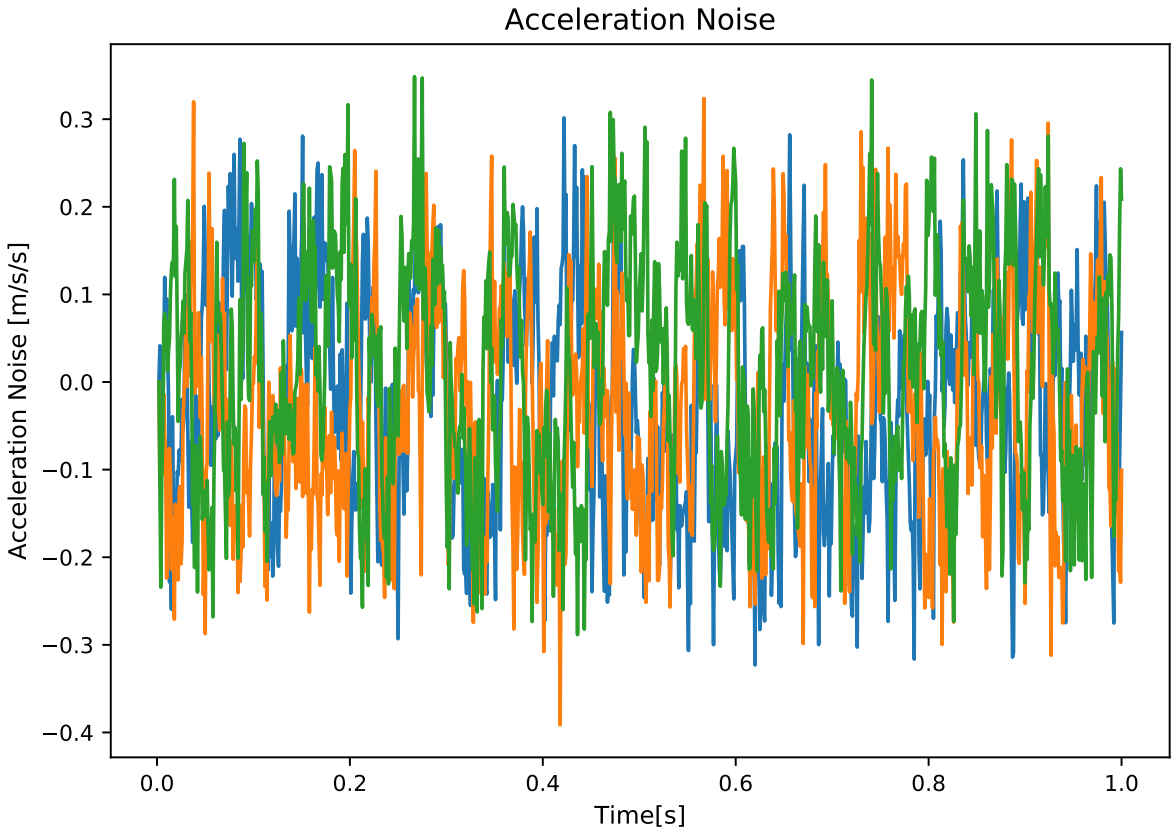

Fig. 23: Plot of acceleration noise along each component for the noise test.

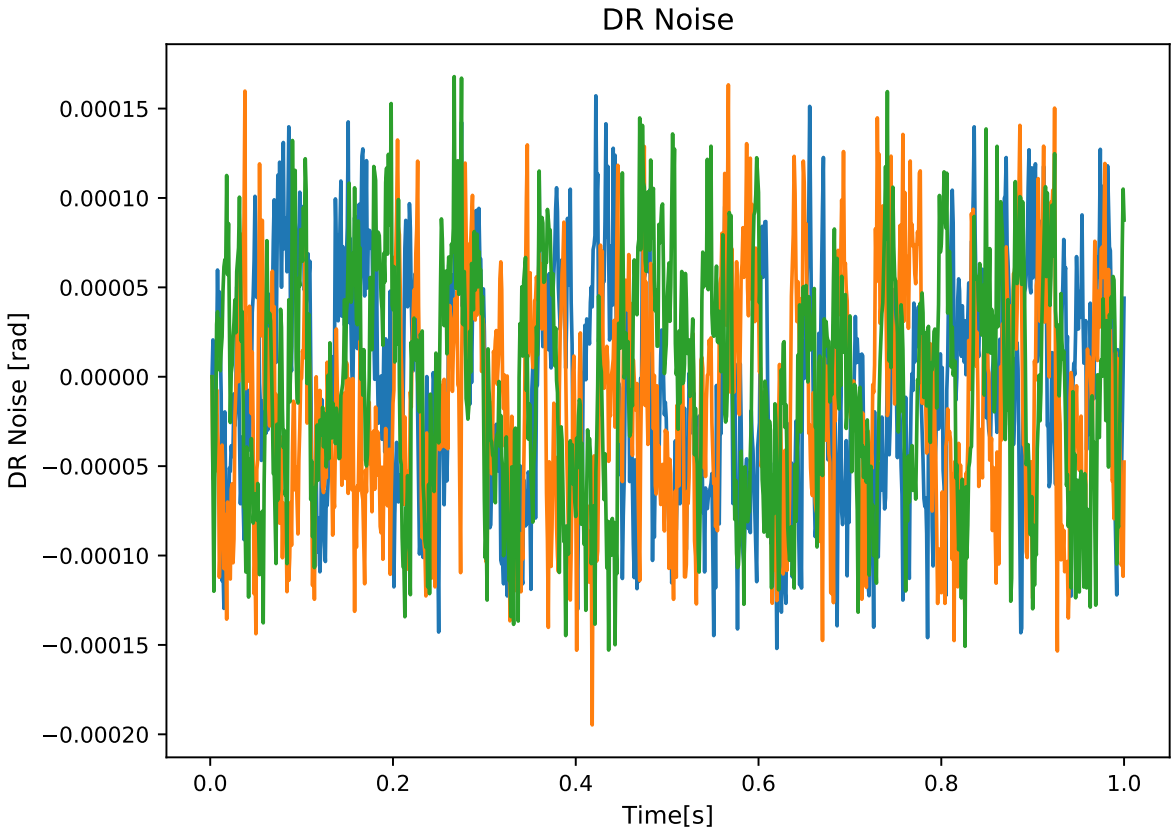

Fig. 24: Plot of PRV noise along each component for the noise test.

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. 25 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.

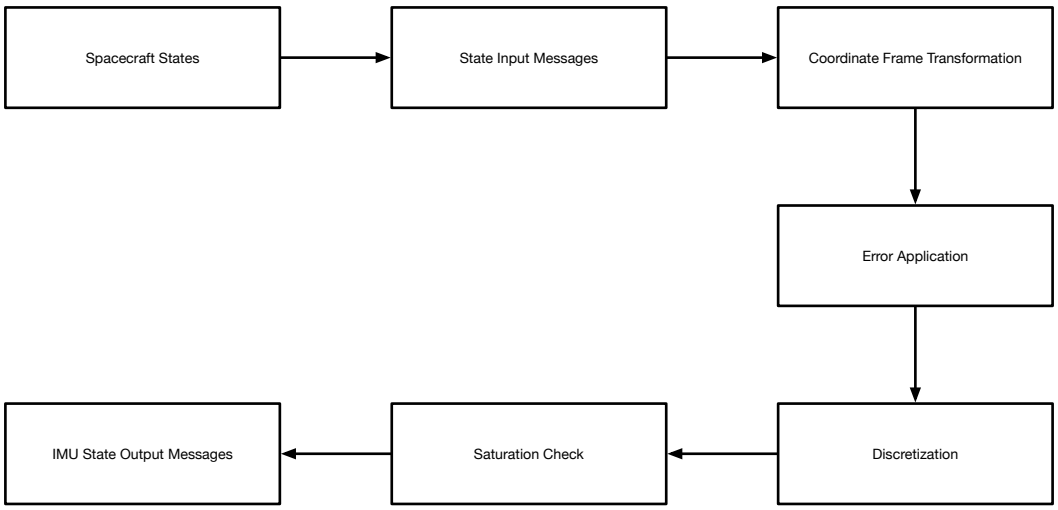

Fig. 25: A pseudo-code diagram demonstrating the flow of inputs and outputs in the IMU module.

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.
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.