Autonomous Vehicle Simulation (AVS) Laboratory, University of Colorado

Basilisk Technical Memorandum

Document ID: Basilisk-pixelLineBiasUKF

RELATIVE OD UNSCENTED FILTER WITH BIAS ESTIMATION

Prepared by	T. Teil

Status: First Version
Scope/Contents
This module filters center and apparent diameter measurements in order to estimate spacecraft relative position in the inertial frame. The filter used is an unscented Kalman filter, and the images are first processed by houghCricles and pixelLineConverter in order to produce this filter's measurements.

Rev	Change Description	By	Date
1.0	First documentation	T. Teil	06/20/2019

Contents

---

1 Model Description

This module implements a square-root unscented Kalman Filter in order to achieve it's best state estimate of the inertial spacecraft attitude states. The estimated state is the attitude (MRPs) and the spacecraft rotation rate in the body frame.

Important: The default units in Basilisk are meters for distance, and meters per second for speed. These are the units to be used for this filter, though the internals use km and km/s for numerical precision.

1.1 Filter Setup

The equations and algorithm for the square root uKF are given in "inertialUKF_DesignBasis.pdf" [**?**] alongside this document. The filter is therefore derived with the states being $\boldsymbol{X} = \begin{bmatrix} {}^{\mathcal{N}}\boldsymbol{r} & {}^{\mathcal{N}}\boldsymbol{v} \end{bmatrix}^T$

The dynamics of the filter are given in Equations (**??**). τ is the total torque read in by the wheels.

$$\dot{\boldsymbol{r}} = \boldsymbol{v} \tag{1}$$

$$\dot{\boldsymbol{v}} = -\frac{\mu}{|\boldsymbol{r}|^3}\boldsymbol{r} \tag{2}$$

The propagation is done using an RK4 integrator. The following square-root uKF coefficients are used: $\alpha = 0.02$, and $\beta = 2$.

1.2 Measurements

The measurement model is simple given the pre-processing by the pixelLineConverter, which extracts the pixel Line transformations from the filter code. This is done to simplify the upkeep and modularity of the filter for Optical Navigation.

$$\boldsymbol{G}_i(\boldsymbol{X}) = \boldsymbol{r} \tag{3}$$

2 Module Functions

- **relativeOD UKF Time Update:** Performs the filter time update as defined in the baseline algorithm
- **relativeOD UKF Meas Update:** Performs the filter measurement update as defined in the baseline algorithm
- **relativeOD UKF Two Body Dynamics:** Provides the function used for integration and incorporates all the known dynamics
- **relativeOD UKF Meas Model:** Predicts the measurements given current state and measurement model $\boldsymbol{G}$
- **relativeOD State Prop:** Integrates the state given the $\boldsymbol{F}$ dynamics of the system
- **relativeOD Clean Update:** Returns filter to a previous state in the case of a bad computation

3 Module Assumptions and Limitations

The assumptions of this module are all tied in to the underling assumptions and limitations to a working filter. In order for a proper convergence of the filter, the dynamics need to be representative of the actual spacecraft perturbations. In this module, the dynamics implemented in the filter are currently just two-body dynamics. Many more perturbations could be added in the future.

Depending on the tuning of the filter (process noise value and measurement noise value), the robustness of the solution will be weighed against it's precision. The number of measurements and the frequency of their availability also influences the general performance.

4 Test Description and Success Criteria

4.1 Test 1: Individual Methods Tests

The first test in this suite runs methods individually:

- relOD uKF Meas Model: This test creates a Sigma Point matrix and predicts the measurements model's computations. It compares the expected output and the actual output down to 1E-15
- relOD State Prop: This test runs the state propagation after one step of simulation. It's main goal is to test the RK4, as it runs one in python and compares them down to 1E-15

4.2 Test 2: State Propagation

This test runs a pure propagation test. The states are set to a fixed value and integrated with the filter. This shows filter stability in the simple case and a very low tolerance for error is permitted (1E-10).

Import sim parameters are:

- **Timestep:** dt $= 1$s
- **Planet:** Mars, $\mu = 42828.314$

Figures **??** and **??** show the results for the energy and state errors.
Energy is conserved, and state errors are down to machine precision

4.3 Test 3: State Update

Given no unknown dynamics, this test runs the filter by giving position measurements every 50s. For the first segment of the simulation, the measurements give the expected orbit with noise. After 250s, the orbit is kicked by a velocity step and the measurement changed.

Import sim parameters are:

- **Timestep:** dt $= 1$s
- **Planet:** Mars, $\mu = 42828.314$
- **Noise on measurements:** $5m$
- **Perturbation:** Velocity kick: $[0., 0., 0., -0.01, 0.01, 0.02]$

Figures **??** shows conservation of energy before and after the perturbation, while showing that there is indeed a unexpected event. Figure **??** show the results of the state errors. It shows the rise in errors when the perturbation is introduced and how the filter resolves it. Similarly, the postfit residuals in Figure **??** show how the filter reconverges after being kicked off track.

5 Test Parameters

6 Test Results

Table 2: Test results

Check	Pass/Fail
Test 1	PASS
Test 2	PASS
Test 3	PASS

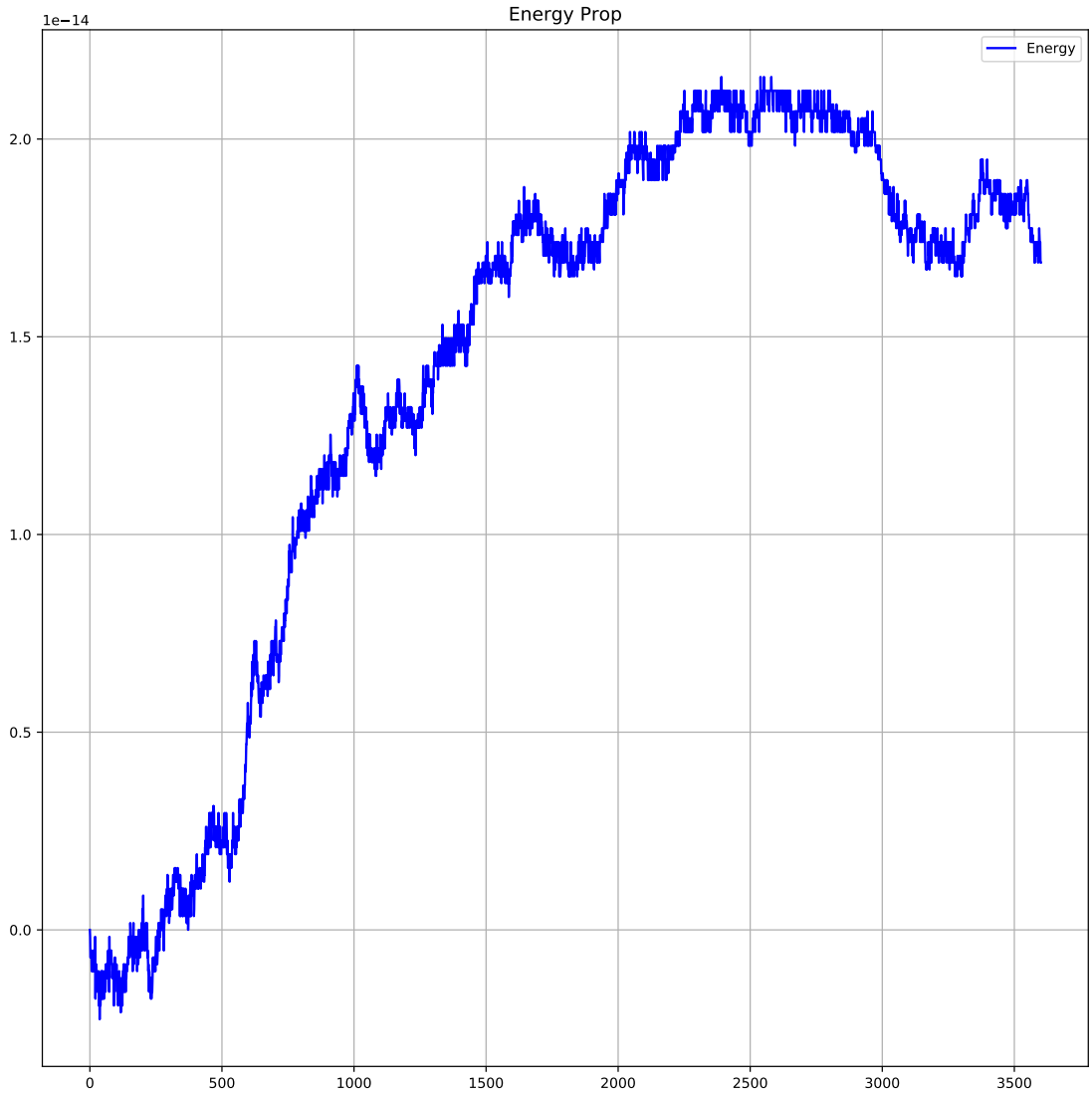

Fig. 1: Orbital Energy

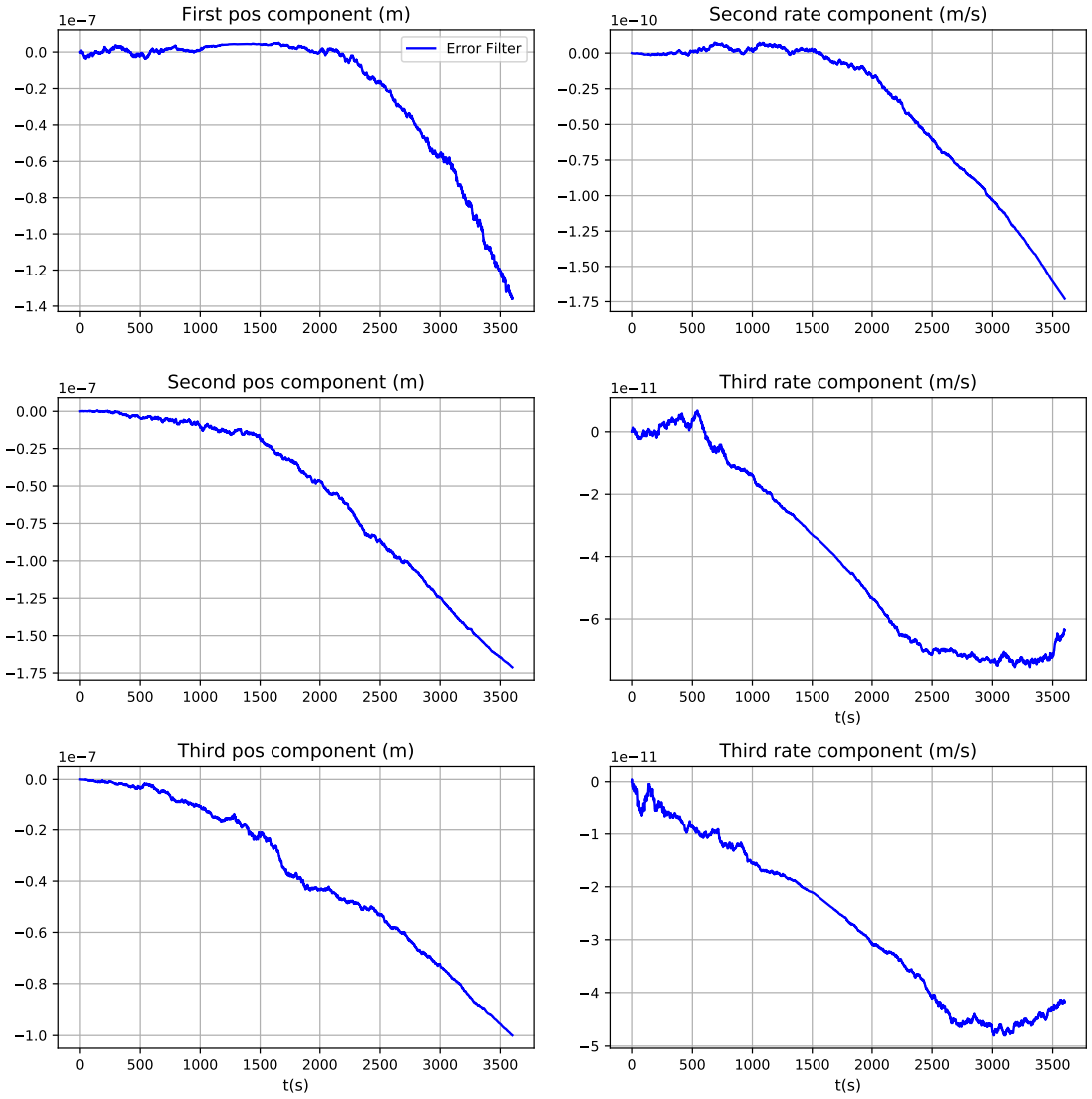

Fig. 2: State error

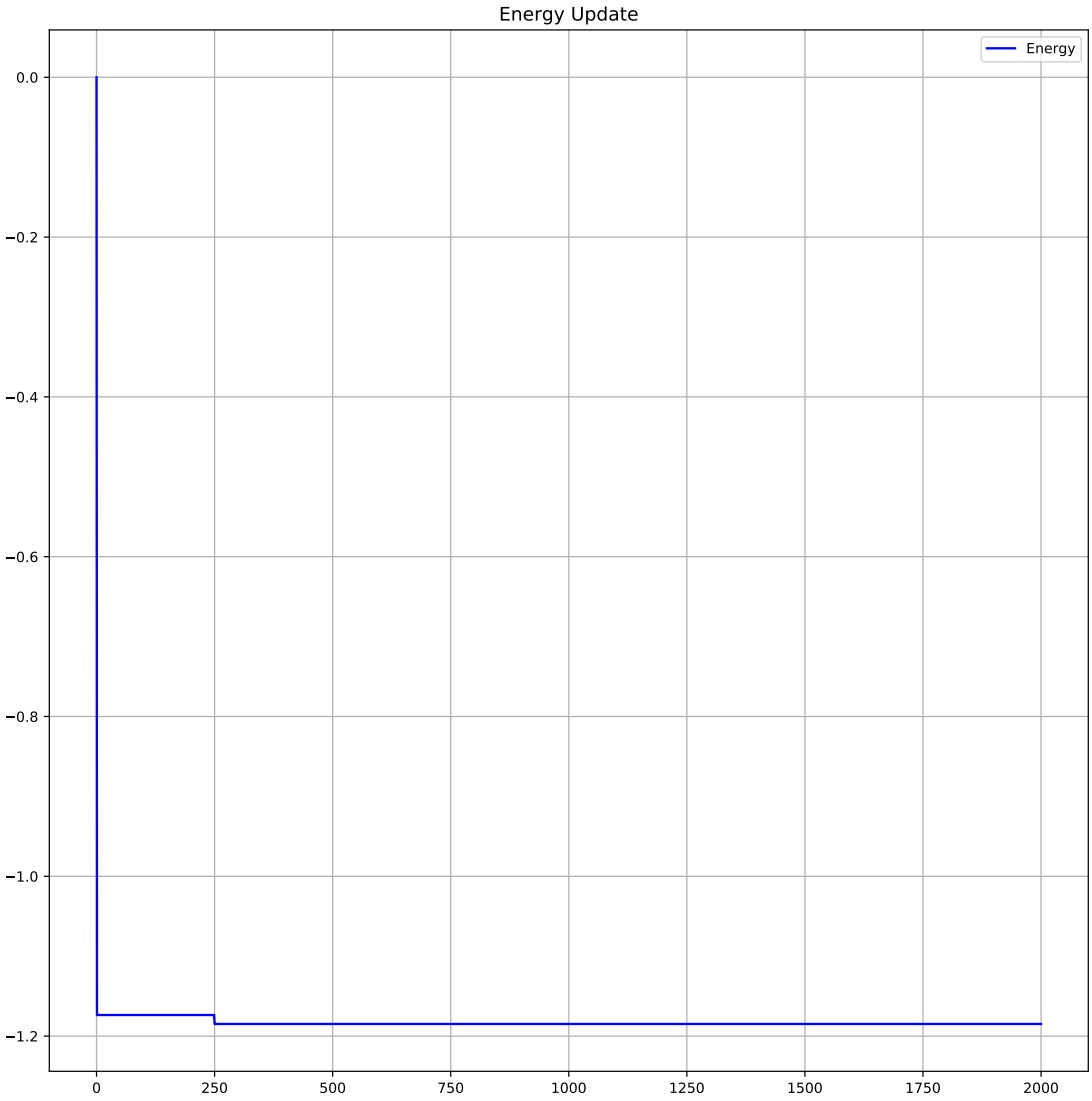

Fig. 3: Orbital Energy

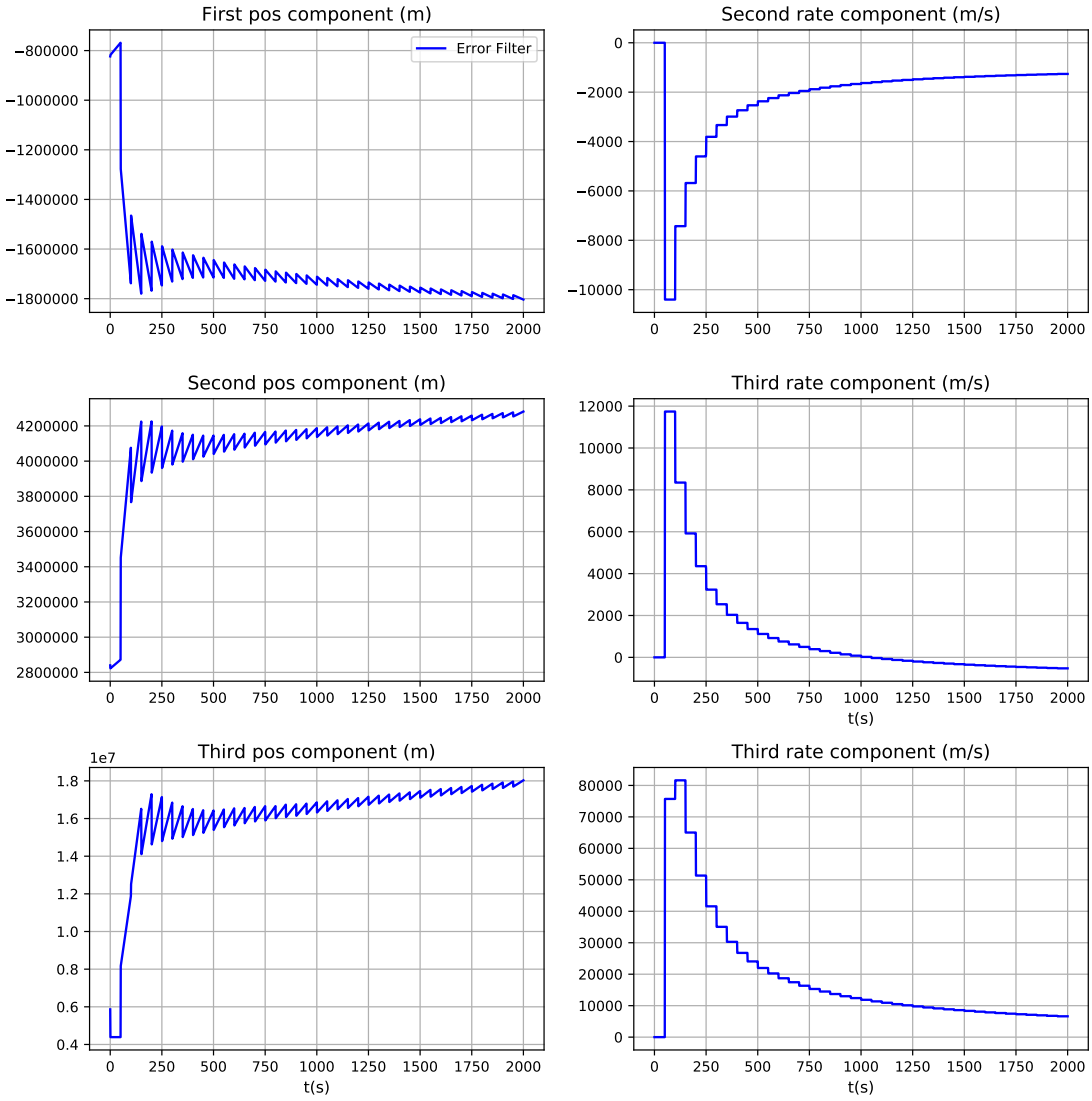

Fig. 4: State error

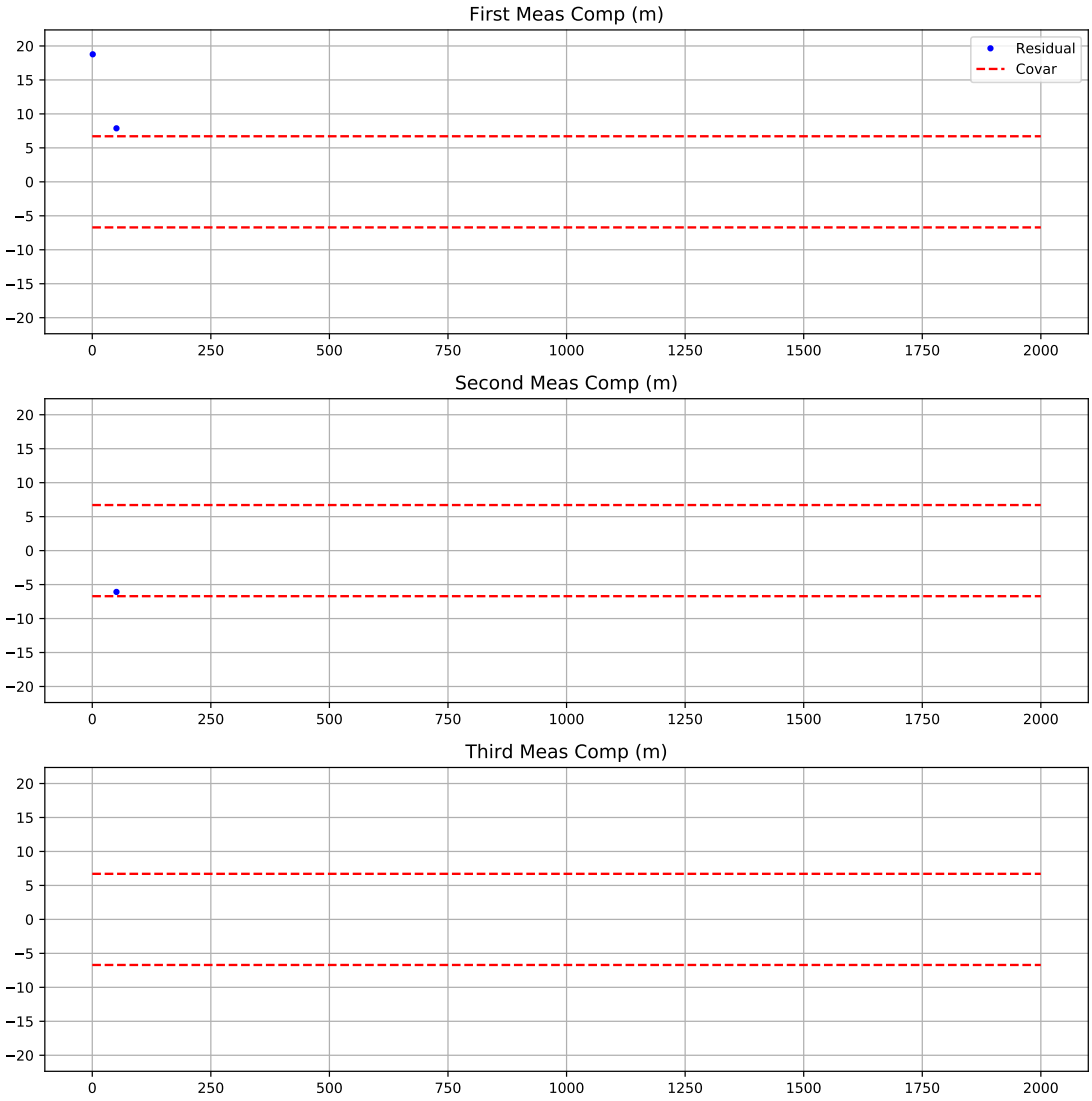

Fig. 5: Post Fit Residuals

Output Value Tested	Tolerated Error
Test 1-Measurement	1E-15
Test 1-Propagation	1E-15
Test 2-Energy	1E-10
Test 2-States	1E-10
Test 3-Energy	1E-10
Test 3-States	1E-10
Test 3-Covariance	1E-10

7 User Guide

7.1 Filter Set-up, initialization, and I/O

In order for the filter to run, the user must set a few parameters:

- The unscented filter has 3 parameters that need to be set, and are best as:
```
filterObject.alpha = 0.02
filterObject.beta = 2.0
filterObject.kappa = 0.0
```

- Initialize orbit:
```
mu = 42828.314*1E9 #m3/s2
elementsInit = orbitalMotion.ClassicElements()
elementsInit.a = 4000*1E3 #meters
elementsInit.e = 0.2
elementsInit.i = 10
elementsInit.Omega = 0.001
elementsInit.omega = 0.01
elementsInit.f = 0.1
r, v = orbitalMotion.elem2rv(mu, elementsInit)
```

- The initial covariance:
```
Filter.covar =
[1000*1E6, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 1000.*1E6, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 1000.*1E6, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 5.*1E6, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 5.*1E6, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 5.*1E6]
```

- The initial state :
```
filterObject.stateInit = r.tolist() + v.tolist()
```

- The process noise :
```
qNoiseIn = np.identity(6)
qNoiseIn[0:3, 0:3] = qNoiseIn[0:3, 0:3]*1E-8*1E-8
qNoiseIn[3:6, 3:6] = qNoiseIn[3:6, 3:6]*1E7*1E7
filterObject.qNoise = qNoiseIn.reshape(36).tolist()
```

The messages must also be set as such:

- `filterObject.navStateOutMsgName = "relod_state_estimate"`
- `filterObject.filtDataOutMsgName = "relod_filter_data"`
- `filterObject.opNavInMsgName = "reold_opnav_meas"`