
# 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 to produce this filter's measurements. This filter is nearly identical to the relativeOD filter except that it estimates bias in the circle measurements. To do this, it integrates the pixelLine converter module as the measurement model. |

| 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 spacecraft rotation and velocity in the inertial frame, as well as a bias on the measurements in pixels $\boldsymbol{b}$.

**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" [3] alongside this document. The filter is therefore derived with the states being $\boldsymbol{X} = \begin{bmatrix} {}^{\mathcal{N}}\boldsymbol{r} & {}^{\mathcal{N}}\boldsymbol{v} & \boldsymbol{b} \end{bmatrix}^T$

The dynamics of the filter are given in Equations (1). $\tau$ is the total torque read in by the wheels.

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

$$\dot{\boldsymbol{v}} = -\frac{\mu}{|\boldsymbol{r}|^3}\boldsymbol{r}\dot{\boldsymbol{b}} \qquad = \boldsymbol{0} \tag{2}$$

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

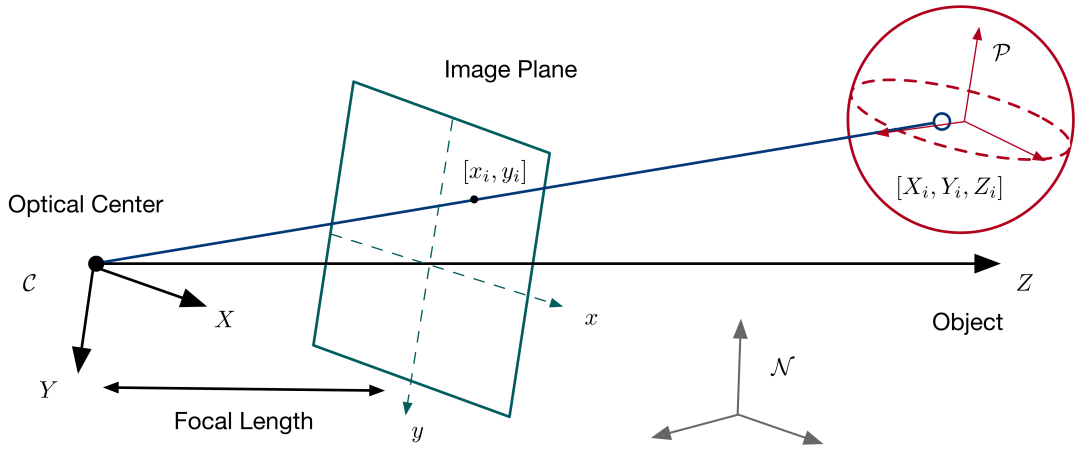


**Fig. 1:** Camera Model

This converter module processes the output of a circle finding method to extract spacecraft inertial position. It does this by reading spacecraft attitude (coming from star tracker or other means), camera parameters, and the circle properties.

Messages read:

- CameraConfigMsg: containing focal length, resolution, and sensor size. These values are needed for the following computations. Notably the camera frame relative to the body frame is used.
- CirclesInMsg: Circle radius, center pixel and line, and uncertainty around these values in pixels.
- NavAttInMsg: Used for the spacecraft attitude. This allows to move from the body frame to the inertial frame.

Message written:

- OpNavFswMsg: Message containing ${}^{\mathcal{N}}\boldsymbol{r}$ and it's covariance.

## 1.3 Position computation

A geometrical method can be used to extract pose information from center and apparent diameter information. The norm of the position vector is given by the apparent size, it's direction is given by the pixel and line data. Using ${}^{\mathcal{C}}\boldsymbol{r}_c = {}^{\mathcal{C}}\begin{bmatrix} r_1 & r_2 & r_3 \end{bmatrix}^T$ as the relative vector of the camera with respect to the celestial center, $A$ as the apparent diameter of the celestial body, $D$ as the actual diameter:

$$|\boldsymbol{r}_c| = \frac{1}{2}\frac{D}{\sin\left(\frac{1}{2}A\right)} \qquad \frac{1}{r_3}\begin{bmatrix} r_1 \\ r_2 \end{bmatrix} = \frac{1}{r_3}\tilde{\boldsymbol{r}} = \frac{1}{f}\begin{bmatrix} x \\ y \end{bmatrix} \tag{3}$$

These equations have been used in multiple instances.[1,2] The third component of $\boldsymbol{r}_c$ provides the range measurement to the body which can be extracted using the apparent diameter measurements. Hence the definition of $\tilde{\boldsymbol{r}}$ which only contains the first two components of $\boldsymbol{r}_c$. The vector components of $\boldsymbol{r}_c$ can be expressed relative to the inertial frame assuming inertial attitude knowledge from other instruments. Using the position of the camera on the spacecraft this provides the measurement value for an orbit determination filter using a circle-finding algorithm.

## 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

This filter builds on the long test suite of other SRuKFs in Basilisk. This test focuses on the differences from other filters: the measurement update. In order to keep a rigorous process, the state propagation is test once more as well.

## 4.1 Test 1: Individual Methods Tests

The first test in this suite runs methods individually:

- pixel Line 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
- pixel Line 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).

Input circle measurement parameters are:

- Input circlesCenters = [100, 200]
- Input circlesRadii = [100]
- Input planetIds = [2]
- Input cirlcesInMsg = msg
- Input planetId = 2
- Input countHalfSPs = 9
- Input numStates =9

Input attitude parameters are:

- Input sigma_BN = [0, 0.2,-0.1]
- Input omega_BN_B = [0.,0.,0.]
- Input sigma_CB = [-0.2, 0., 0.3]
- Input focalLength = 1
- Input sensorSize = [10,10]
- Input resolution = [512, 512]

Figures 2 and 3 show the results for the energy and state errors.
Energy is conserved, and state errors are down to machine precision

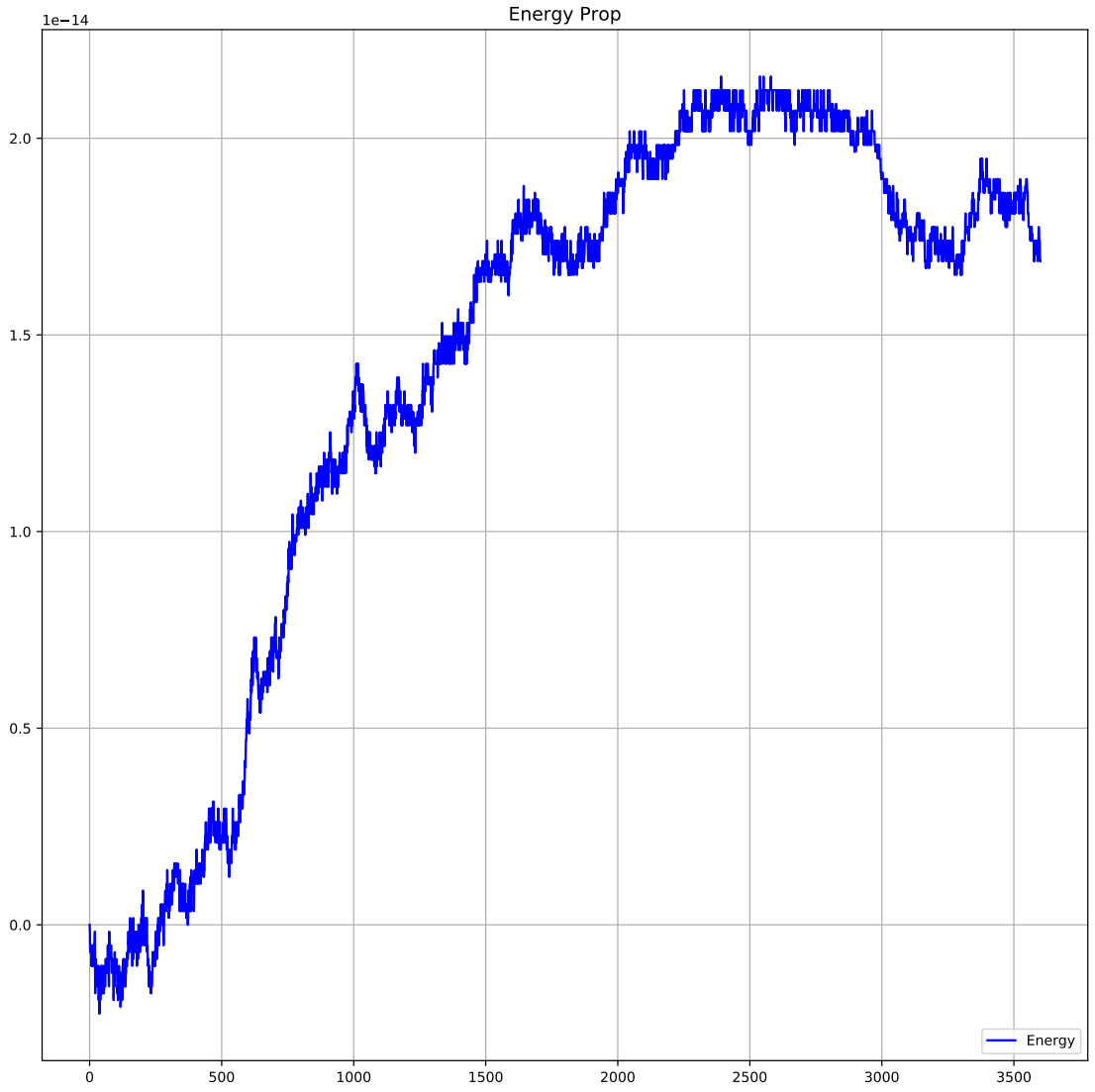


**Fig. 2:** Orbital Energy

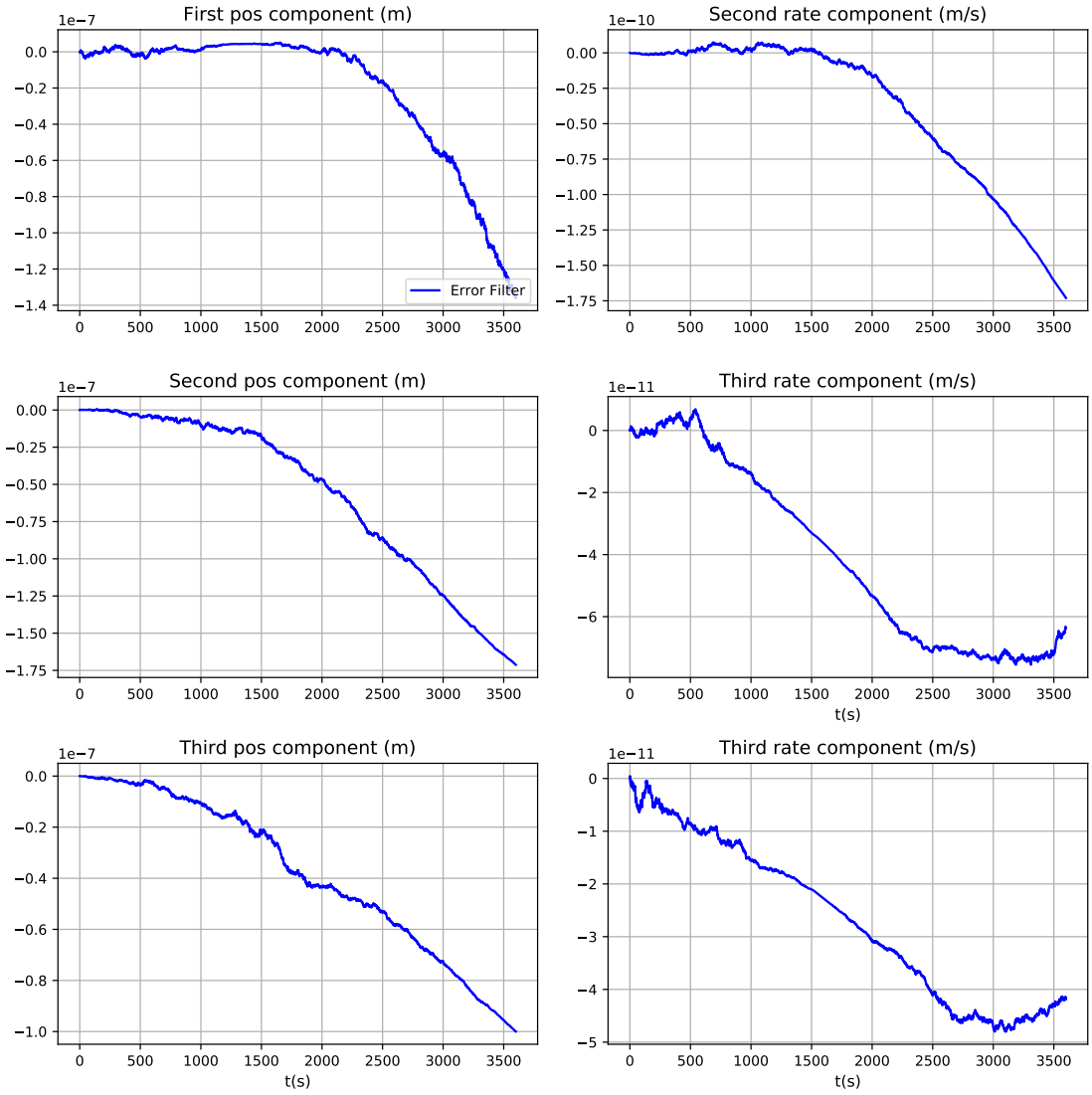


**Fig. 3:** State error

| 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 |

# 5 Test Parameters

# 6 Test Results

**Table 2:** Test results

| Check | Pass/Fail |
|---|---|
| Test 1 | PASS |
| Test 2 | PASS |

# 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, 0.0, 0.0,
0.0, 1000.*1E6, 0.0, 0.0, 0.0, 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,
0.0, 0.0, 0.0, 5.*1E6, 0.0, 0.0,0.0, 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, 0.0, 0.0, 0.0, 5.*1E6, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
5.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 5.0,]
```

- The initial state :
```
bias = [1,1,1]
filterObject.stateInit = r.tolist() + v.tolist() + bias
```

- The process noise :

```
qNoiseIn = np.identity(9)
qNoiseIn[0:3, 0:3] = qNoiseIn[0:3, 0:3]*1E-8*1E-8
qNoiseIn[3:6, 3:6] = qNoiseIn[3:6, 3:6]*1E-7*1E-7
qNoiseIn[6:, 6:]  = qNoiseIn[6:, 6:]*1E-1*1E-1
filterObject.qNoise = qNoiseIn.reshape(9*9).tolist()
```

The messages must also be set as such:

- `filterObject.navStateOutMsgName = "pixelLine_state_estimate"`
- `filterObject.filtDataOutMsgName = "pixelLine_filter_data"`
- `filterObject.circlesInMsgName = "circles_data"`
- `filterObject.cameraConfigMsgName = "camera_config_data"`
- `filterObject.attInMsgName = "simple_att_nav_output"`

# REFERENCES

[1] Richard H. Battin. *An Introduction to the Mathematics and Methods of Astrodynamics, Revised Edition*. American Institute of Aeronautics and Astronautics, 2019/01/10 1999.

[2] Wiliam M. Owen. Optical navigation program mathematical models. Engineering Memorandum 314-513, Jet Propulsion Laboratory, August 1991.

[3] R. van der Merwe. The square-root unscented kalman filter for state and parameter-estimation. Acoustics, Speech, and Signal Processing, 2001.