Autonomous Vehicle Simulation (AVS) Laboratory, University of Colorado

Basilisk Technical Memorandum

Document ID: Basilisk-test_sunlineUKF

SUNLINE UKF MODULE AND TEST

Prepared by	T. Teil

Status: Initial document
Scope/Contents
This module implements and tests a Unscented Kalman Filter in order to estimate the sunline direction.

Rev:	**Change Description**	**By**
Draft	Initial Revision	T. Teil

Contents

---

1 Introduction

The Unscented Kalman filter (UKF) in the AVS Basilisk simulation is a sequential filter implemented to give the best estimate of the desired states. In this method we estimate the sun heading as well as it's rate of change along the observable axes. The UKF reads in the message written by the coarse sun sensor, and writes a message containing the sun estimate.

This document summarizes the content of the module, how to use it, and the test that was implemented for it. More information on the filter derivation can be found in Reference [**?**], and more information on the square root unscented filter can be found in Reference [**?**] (attached alongside this document).

1.1 Dynamics

The states that are estimated in this filter are the sunline vector, and it's rate of change $\boldsymbol{X}^* = \begin{bmatrix}\boldsymbol{d} & \dot{\boldsymbol{d}}\end{bmatrix}^T$. The star superscript represents that this is the reference state.

The dynamics are given in equation **??**. Given the nature of the filter, there is an unobservable state component: the rotation about the $\boldsymbol{d}$ axis. In order to remedy this, we project the states along this axis and subtract them, in order to measure only observable state components.

$$\boldsymbol{F}(\boldsymbol{X}) = \begin{bmatrix}\boldsymbol{F}_1(\boldsymbol{d}) \\ \boldsymbol{F}_2(\dot{\boldsymbol{d}})\end{bmatrix} = \begin{bmatrix}\dot{\boldsymbol{d}} - \left((\boldsymbol{d}\cdot\dot{\boldsymbol{d}})\frac{\boldsymbol{d}}{||\boldsymbol{d}||^2}\right) \\ -\frac{1}{\Delta t}\left((\boldsymbol{d}\cdot\dot{\boldsymbol{d}})\frac{\boldsymbol{d}}{||\boldsymbol{d}||^2}\right)\end{bmatrix} \tag{1}$$

The measurement model is given in equation **??**, and the H matrix defined as $H = \left[\frac{\partial \boldsymbol{G}(\boldsymbol{X},t_i)}{\partial \boldsymbol{X}}\right]^*$ is given in equation **??**.

In this filter, the only measurements used are from the coarse sun sensor. For the i^{th} sensor, the measurement is simply given by the dot product of the sunline heading and the normal to the sensor. This yields easy partial derivatives for the H matrix, which is a matrix formed of the rows of transposed normal vectors (only for those which received a measurement). Hence the H matrix has a changing size depending on the amount of measurements.

$$\boldsymbol{G}_i(\boldsymbol{X}) = \boldsymbol{n}_i \cdot \boldsymbol{d} \tag{2}$$

$$\boldsymbol{H}(\boldsymbol{X}) = \begin{bmatrix}\boldsymbol{n}_1^T \\ \vdots \\ \boldsymbol{n}_i^T\end{bmatrix} \tag{3}$$

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

2.1 User initialization

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
```

- The angle threshold under which the coarse sun sensors do not read the measurement:
  `FilterContainer.sensorUseThresh = 0.`
- The process noise matrix:
```
qNoiseIn = numpy.identity(5)
qNoiseIn[0:3, 0:3] = qNoiseIn[0:3, 0:3]*0.01*0.01
qNoiseIn[3:6, 3:6] = qNoiseIn[3:6, 3:6]*0.001*0.001
filterObject.qNoise = qNoiseIn.reshape(25).tolist()
```
- The measurement noise value, for instance:
  `FilterContainer.qObsVal = 0.001`
- The initial covariance:
```
Filter.covar =
[0.4, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.4, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.4, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.04, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.04, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.04]
```
- The initial state :
  `Filter.state =[1.0, 0.0, 0.0, 0.0, 0.0, 0.0]`

The messages must also be set as such:

- `filterObject.navStateOutMsgName = "sunline_state_estimate"`
- `filterObject.filtDataOutMsgName = "sunline_filter_data"`
- `filterObject.cssDataInMsgName = "css_sensors_data"`
- `filterObject.cssConfInMsgName = "css_config_data"`

2.2 Inputs and Outputs

The UKF reads in the measurements from the coarse sun sensors. These are under the form of a list of cosine values. Knowing the normals to each of the sensors, we can therefore use them to estimate sun heading.

3 Test Design

The unit test for the sunlineUKF module is located in:

`fswAlgorithms/attDetermination/sunlineUKF/_UnitTest/test_SunlineUKF.py`

As well as another python file containing plotting functions:
`fswAlgorithms/attDetermination/sunlineUKF/_UnitTest/SunlineUKF_test_utilities.py`

The test is split up into 3 subtests. The first test creaks up all of the individual filter methods and tests them individually. These notably go over the square-root unscented filter specific functions. The second test verifies that in the case where the state is zeroed out from the start of the simulation, it remains at zero. The third test verifies the behavior of the time update with a measurement modification in the middle of the run.

3.1 Individual tests

In each of these individual tests, random inputs are fed to the methods and their values are computed in parallel in python. These two values are then compared to assure that the correct computations are taking place.

- QR Decomposition: This tests the QR decomposition function which returns just the R matrix. Tolerance to absolute error $\epsilon = 10^{-15}$.

  Passed

- LU Decomposition: This tests the LU Decomposition accuracy. Tolerance to absolute error $\epsilon = 10^{-14}$.

  Passed

- LU backsolve: This tests the LU Back-Solve accuracy. Tolerance to absolute error $\epsilon = 10^{-14}$.

  Passed

- LU matrix inverse: This tests the LU Matrix Inverse accuracy. Tolerance to absolute error $\epsilon = 10^{-14}$.

  Passed

- Cholesky decomposition: This tests the Cholesky Matrix Decomposition accuracy. Tolerance to absolute error $\epsilon = 10^{-14}$.

  Passed

- L matrix inverse: This tests the L Matrix Inverse accuracy. Tolerance to absolute error $\epsilon = 10^{-14}$.

  Passed

- U matrix inverse: This tests the U Matrix Inverse accuracy. Tolerance to absolute error $\epsilon = 10^{-12}$.

  Passed

3.2 Static Propagation

This test also takes no measurements in, and propagates with the expectation of no change. It then tests that the states and covariance are as expected throughout the time of simulation. Plotted results are seen in Figure **??**. We indeed see that the state and covariance that evolve nominally and without bias .

Tolerance to absolute error: $\epsilon = 10^{-10}$

3.3 Full Filter test

This test the filter working from start to finish. No measurements are taken in for the first 20 time steps. Then a heading is given through the CSS message. Halfway through the simulation, measurements stop, and 20 time steps later a different heading is read. The filter must be robust and detect this change. This test is parametrized for different test lengths, different initial conditions, different measured headings, and with or without measurement noise. All these are successful.

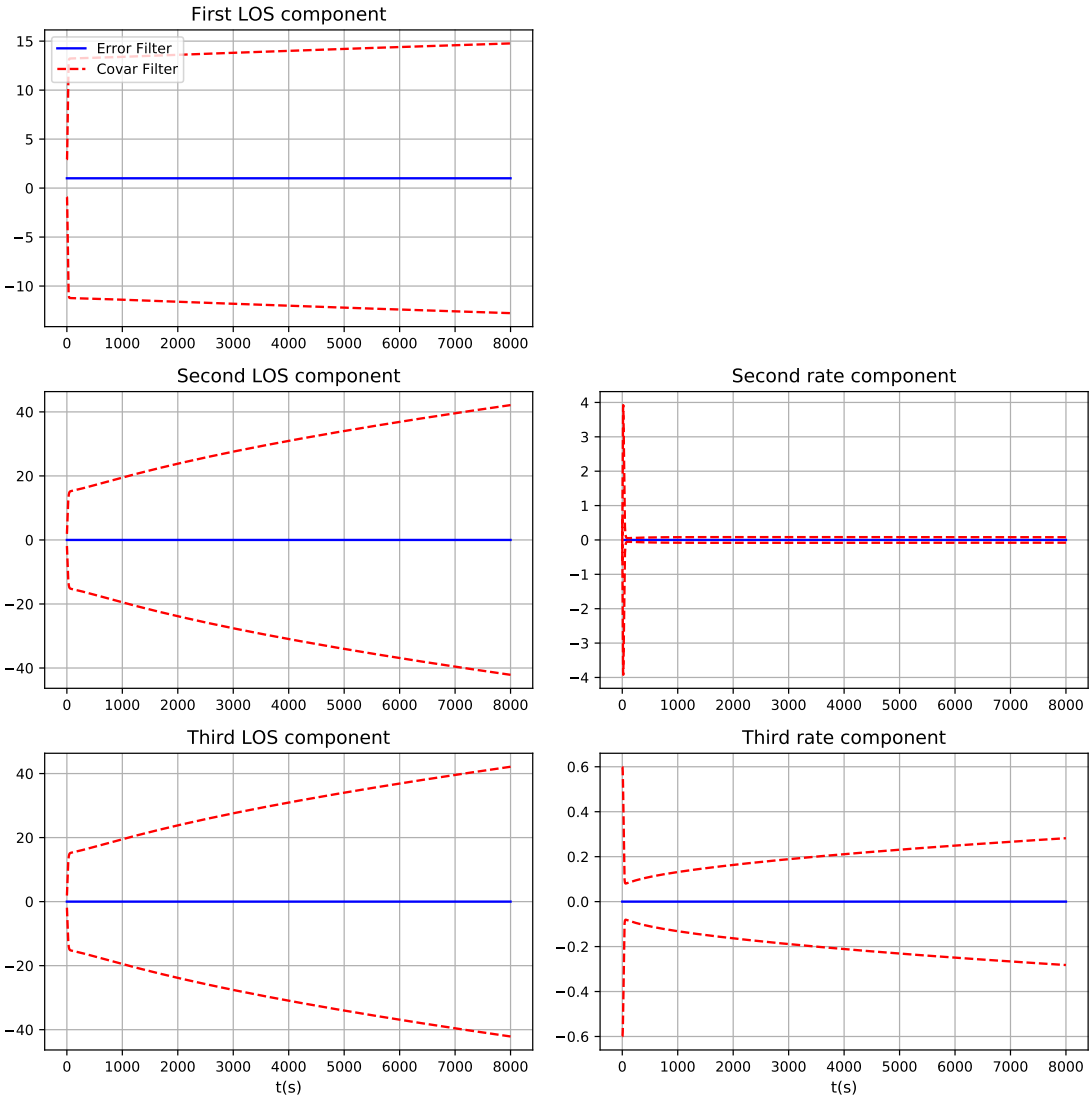

Fig. 1: State error and covariance

Tolerance to absolute error without measurement noise: $\epsilon = 10^{-10}$

Passed

Plotted results are seen in Figures **??**, and **??**. Figure **??** shows the state error and covariance over the run. We see the covariance initially grow, then come down quickly as measurements are used. It grows once again as the measurements stop before bringing the state error back to zero with a change in sun heading.

Figure **??** shows the post fit residuals for the filter, with no measurement noise. We see that the observations are read in well an that the residuals are brought back down to zero.

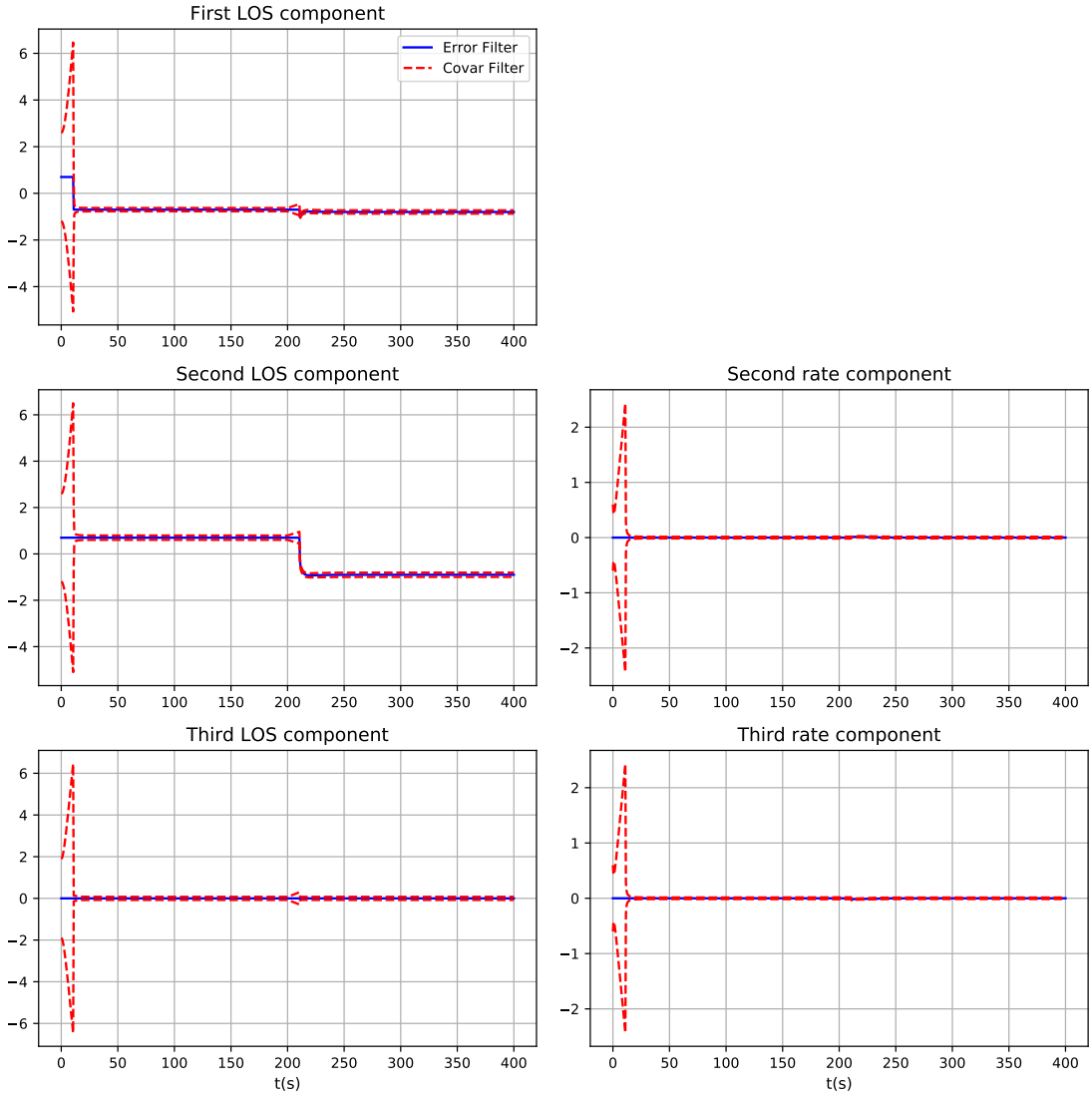

Fig. 2: State error and covariance

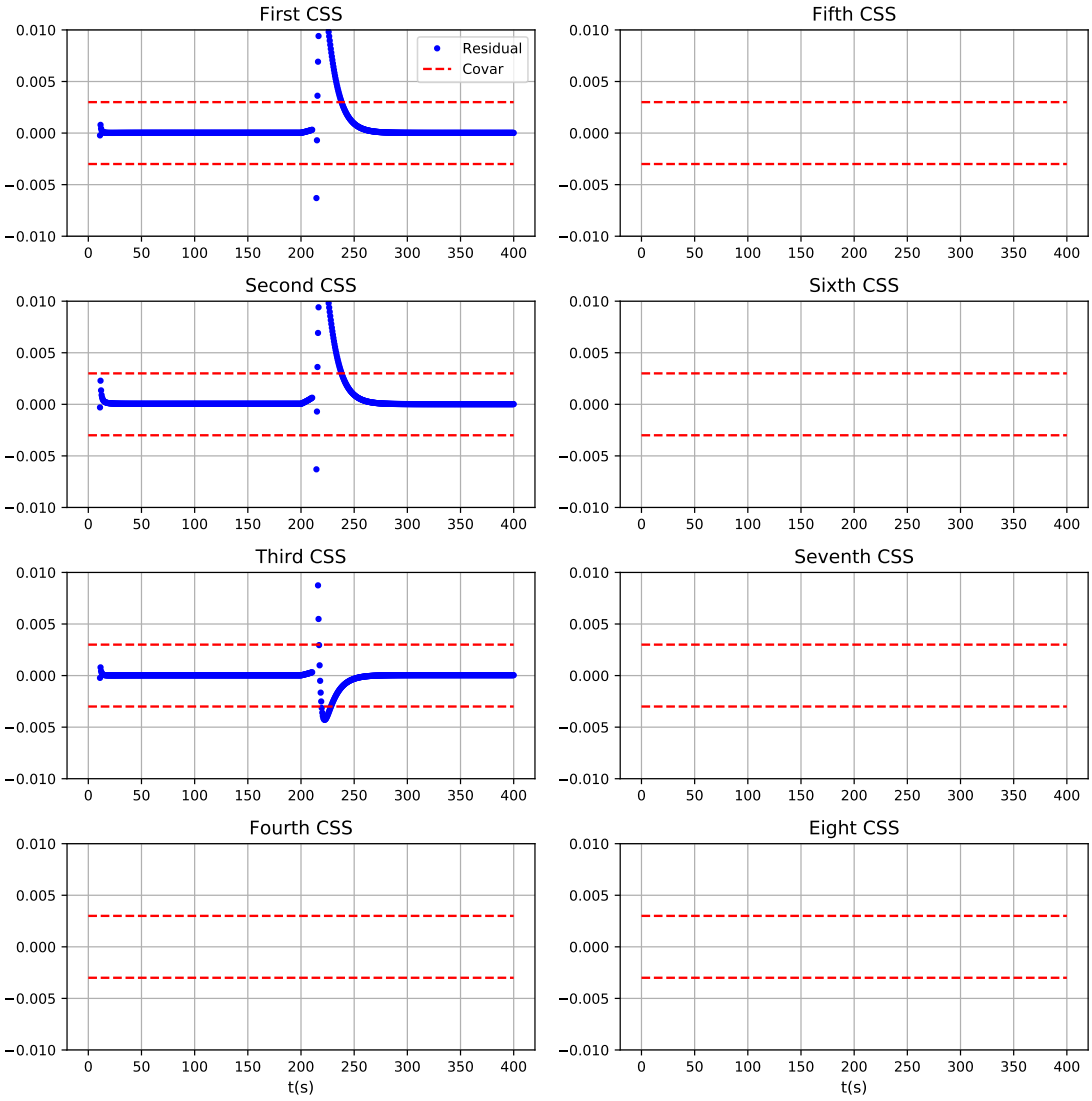

Fig. 3: Post Fit Residuals