Autonomous Vehicle Simulation (AVS) Laboratory, University of Colorado

Basilisk Technical Memorandum

Document ID: Basilisk-test_unitThrusterDynamics

SUNLINE SWITCH-EKF MODULE AND TEST

Prepared by	T. Teil

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

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

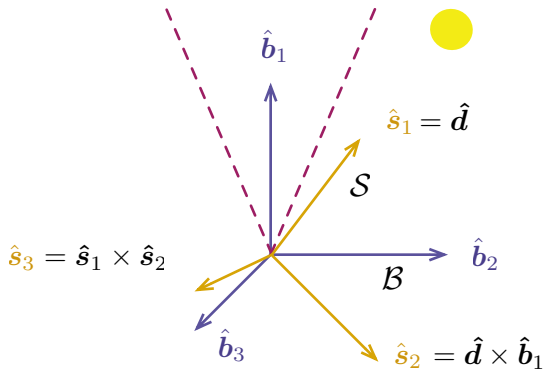

Fig. 1: Frame built off the body frame for Switch filters

Contents

---

1 Introduction

The Switch Extended Kalman filter (SEKF) 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 SEKF 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 EKF can be found in Reference [**?**].

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

2.1 Filter Derivation

The Switch-EKF attempts to avoid subtracting any terms from the state, while still removing the unobservable component of the rate. In order to do this, an appropriate frame must be defined. In order to not track the rate component alongside the sunline direction, that vector needs to be one of the basis vectors of the frame. It is decided to be the first vector for the frame, and therefore in that frame, ω_1 the component of the rotation rate can be removed from the states. This frame is called $\mathcal{S}_1 = \{\hat{\boldsymbol{s}}_1 = \frac{\boldsymbol{d}}{|\boldsymbol{d}|}, \hat{\boldsymbol{s}}_2, \hat{\boldsymbol{s}}_3\}$. This is seen in Figure **??**, where the dotted line represents the 30° threshold cone before switching frames.

The second vector of the frame must be created using only $\boldsymbol{d}$, and the body frame vectors. The first intuitive decision, is to use $\hat{\boldsymbol{b}}_1$ of the body frame and define $\boldsymbol{s}_2$ in Equation (**??**). The third vector $\hat{\boldsymbol{s}}_3$ of the $\mathcal{S}_1$ frame, is naturally created from the first two.

$$\hat{\boldsymbol{s}}_2 = \frac{\hat{\boldsymbol{s}}_1 \times \hat{\boldsymbol{b}}_1}{|\hat{\boldsymbol{s}}_1 \times \hat{\boldsymbol{b}}_1|} \qquad \hat{\boldsymbol{s}}_3 = \frac{\hat{\boldsymbol{s}}_1 \times \hat{\boldsymbol{s}}_2}{|\hat{\boldsymbol{s}}_1 \times \hat{\boldsymbol{s}}_2|} \tag{1}$$

The problem that arises is the singularity that occurs when $\hat{\boldsymbol{b}}_1$ and $\boldsymbol{d}$ become aligned: this frame becomes undefined. In order to counteract this, using a similar process as the shadow set used for Modified Rodrigues Parameters [**?**], a second frame is created. This frame $\mathcal{S}_2 = \{\hat{\bar{\boldsymbol{s}}}_1 = \hat{\boldsymbol{s}}_1, \hat{\bar{\boldsymbol{s}}}_2, \hat{\bar{\boldsymbol{s}}}_3\}$ is created with the same first vector, but constructs $\hat{\bar{\boldsymbol{s}}}_2$ using $\hat{\boldsymbol{b}}_2$ of the body frame as in Equation (**??**).

The last vector, once again, finishes the orthonormal frame.

$$\hat{\tilde{\boldsymbol{s}}}_2 = \frac{\hat{\tilde{\boldsymbol{s}}}_1 \times \hat{\boldsymbol{b}}_2}{|\hat{\tilde{\boldsymbol{s}}}_1 \times \hat{\boldsymbol{b}}_2|} \tag{2}$$

With both these frames, $\mathcal{S}_1$ and $\mathcal{S}_2$, the singularities can always be avoided. Indeed, $\mathcal{S}_1$ becomes singular when $\boldsymbol{d}$ approches $\hat{\boldsymbol{b}}_1$, while $\mathcal{S}_2$ becomes singular when the sunheading approaches $\hat{\boldsymbol{b}}_2$. By changing frames, whenever the sunline gets within a safe cone of 30° (a modifiable value) of $\hat{\boldsymbol{b}}_1$, the frame is rotated into $\mathcal{S}_2$, which is not singular. Similarly, when $\boldsymbol{d}$ approches $\hat{\boldsymbol{b}}_2$ the frame is switched back to $\mathcal{S}_1$.

Because the two frames share the sunline vector $\boldsymbol{d}$, this vector is the same in both frames. This is a clear advantage as this is the vector we desire to estimate, and not having to rotate it avoids numerical issues. The rotation of the rates is done by computing the following DCMs, of which all the vectors are known.

$$[\mathcal{BS}_1] = \begin{bmatrix} {}^{\mathcal{B}}\hat{\boldsymbol{s}}_1 & {}^{\mathcal{B}}\hat{\boldsymbol{s}}_2 & {}^{\mathcal{B}}\hat{\boldsymbol{s}}_3 \end{bmatrix} \qquad [\mathcal{BS}_2] = \begin{bmatrix} {}^{\mathcal{B}}\hat{\tilde{\boldsymbol{s}}}_1 & {}^{\mathcal{B}}\hat{\tilde{\boldsymbol{s}}}_2 & {}^{\mathcal{B}}\hat{\tilde{\boldsymbol{s}}}_3 \end{bmatrix} \qquad [\mathcal{S}_2\mathcal{S}_1] = [\mathcal{BS}_2]^T[\mathcal{BS}_1] \tag{3}$$

2.2 Filter Dynamics

The filter is therefore derived with the states being $\boldsymbol{X} = \begin{bmatrix} {}^{\mathcal{B}}\boldsymbol{d} & \omega_2 & \omega_3 \end{bmatrix}^T$, given that $\boldsymbol{\omega}_{\mathcal{S}/\mathcal{B}} = {}^{\mathcal{S}}\begin{bmatrix} \omega_1 & \omega_2 & \omega_3 \end{bmatrix}^T$. The rates of $\mathcal{S}$ relative to the body and inertial frame are related as such: $\boldsymbol{\omega}_{\mathcal{S}/\mathcal{N}} - \boldsymbol{\omega}_{\mathcal{S}/\mathcal{B}} = \boldsymbol{\omega}_{\mathcal{B}/\mathcal{N}}$. Since ω_1 is unknown, it is set to zero. Furthermore, since the sun heading is considered to be constant in the inertial frame over the period of time required for attitude determination and control, the equation becomes $-\bar{\boldsymbol{\omega}}_{\mathcal{S}/\mathcal{B}} = \bar{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}}$.

$\boldsymbol{\omega}_{\mathcal{S}/\mathcal{B}}$ is estimated directly by the filter, and its skew matrix can be computed by setting ω_1 to zero (in the absence of information). This defines $\tilde{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}}$ as a function of known parameters. The dynamics are therefore given by Equations (**??**) and (**??**), where $[\tilde{\boldsymbol{d}}](2,3)$ corresponds to the 2^{nd} and 3^{rd} columns of the $[\tilde{\boldsymbol{d}}]$ matrix.

$$\boldsymbol{X}' = \boldsymbol{F}(\boldsymbol{X}) = \begin{bmatrix} {}^{\mathcal{B}}\boldsymbol{d}' \\ \omega_2' \\ \omega_3' \end{bmatrix} = \begin{bmatrix} -{}^{\mathcal{B}}\bar{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}} \times {}^{\mathcal{B}}\boldsymbol{d} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} {}^{\mathcal{S}}\begin{bmatrix} 0 \\ \omega_2 \\ \omega_3 \end{bmatrix} \times {}^{\mathcal{B}}\boldsymbol{d} \\ 0 \\ 0 \end{bmatrix} \tag{4}$$

$$[A] = \left[\frac{\partial \boldsymbol{F}(\boldsymbol{d}, t_i)}{\partial \boldsymbol{X}}\right] = \begin{bmatrix} [{}^{\mathcal{B}}\tilde{\bar{\boldsymbol{\omega}}}_{\mathcal{S}/\mathcal{B}}] & -[\tilde{\boldsymbol{d}}][\mathcal{BS}](2,3) \\ [0]_{2\times 3} & [0]_{2\times 2} \end{bmatrix} \tag{5}$$

This formulation leads to simple dynamics, much simpler than those of the filter which subtracts the unobservable states, yet can actually estimate the observable of the rate, instead of using past estimates of $\boldsymbol{d}$.

2.3 Switching Frames

When switching occurs, the switch matrix $[W]$ can be computed in Equation (**??**) using the previously computed DCMs. This equation assumes the switch is going from frame 1 to frame 2 (the reciprocal is equivalent), and $[\mathcal{S}_2\mathcal{S}_1](2,3)$ corresponds to the 2^{nd} and 3^{rd} columns of the $[\mathcal{S}_2\mathcal{S}_1]$ matrix.

$$[W] = \begin{bmatrix} [I]_{3\times 3} & [0]_{3\times 2} \\ [0]_{2\times 3} & [\mathcal{S}_2\mathcal{S}_1](2,3) \end{bmatrix} \tag{6}$$

The new states $\boldsymbol{X}$ and covariance [P] after the switch are therefore given in Equation (**??**)

$$\bar{\boldsymbol{X}} = [W]\boldsymbol{X} \qquad [\bar{P}] = [W][P][W]^T \tag{7}$$

When writing out the values of the state and covariance, it is necessary to bring it back into the body frame, using the $[\mathcal{BS}]$ DCM ($\mathcal{S}$ representing the current frame in use).

2.4 Process Noise for Switch-EKF

The time update of the error covariance matrix from time t_k to t_{k+1} ($\Delta t = t_{k+1} - t_k$)is given in equation (**??**). The process noise matrix $[Q]$ is added via the $[\Gamma]$ matrix defined in equation (**??**). Process noise is only added on the accelerations, meaning that $[B] = \begin{bmatrix}[0]_{3\times3}\\ [I]_{3\times3}\end{bmatrix}$ when there are 6 states.

$$[P]_{k+1} = [\Phi](t_{k+1}, t_k)[P]_k[\Phi](t_{k+1}, t_k)^T + [\Gamma](t_{k+1}, t_k)[Q][\Gamma](t_{k+1}, t_k)^T \tag{8}$$

$$[\Gamma](t_{k+1}, t_k) = \int_{t_k}^{t_{k+1}} [\Phi](t_{k+1}, \tau)[B](\tau)\mathrm{d}\tau \tag{9}$$

In others filters (the EKF and the UKF), the second half of the state vector is a direct derivative of the sun heading vector. Regarding state noise compensation, this allowed the approximation in equation (**??**), along with the fact that measurements are received frequently with regard to the evolution of the dynamics.

$$[\Gamma](t_{k+1}, t_k) = \Delta t \begin{bmatrix}\frac{\Delta t}{2}[I]_{3\times3}\\ [I]_{3\times3}\end{bmatrix} \tag{10}$$

This is not the case for this filter. Indeed, $[\Phi]$ is a 5 by 5 matrix, expanded in equation (**??**) using the fact that $[\Phi](t_{k+1}, \tau) = \frac{\partial \boldsymbol{X}(t_{k+1})}{\partial \boldsymbol{X}(\tau)}$, and that $\boldsymbol{X} = \begin{bmatrix}\boldsymbol{d} & \bar{\boldsymbol{\omega}}\end{bmatrix}^T$. With this, equation (**??**) can be re-written as equation (**??**).

$$[\Phi](t_{k+1}, \tau) = \begin{bmatrix}[\Phi_1]_{3\times3} & [\Phi_2]_{3\times2}\\ [\Phi_3]_{2\times3} & [\Phi_3]_{2\times2}\end{bmatrix} = \begin{bmatrix}\frac{\partial \boldsymbol{d}(t_{k+1})}{\partial \boldsymbol{d}(\tau)} & \frac{\partial \boldsymbol{d}(t_{k+1})}{\partial \bar{\boldsymbol{\omega}}(\tau)}\\ \frac{\partial \bar{\boldsymbol{\omega}}(t_{k+1})}{\partial \boldsymbol{d}(\tau)} & \frac{\partial \bar{\boldsymbol{\omega}}(t_{k+1})}{\partial \bar{\boldsymbol{\omega}}(\tau)}\end{bmatrix} \tag{11}$$

$$[\Gamma](t_{k+1}, t_k) = \int_{t_k}^{t_{k+1}} \begin{bmatrix}[\Phi_1]_{3\times3} & [\Phi_2]_{3\times2}\\ [\Phi_3]_{2\times3} & [\Phi_3]_{2\times2}\end{bmatrix}\begin{bmatrix}[0]_{3\times3}\\ [I]_{3\times3}\end{bmatrix}\mathrm{d}\tau = \int_{t_k}^{t_{k+1}} \begin{bmatrix}[\Phi_2]_{3\times2}\\ [\Phi_4]_{2\times2}\end{bmatrix}\mathrm{d}\tau \tag{12}$$

These submatrices of the state transition matrix now need to be approximated. As before, assuming dense tracking data, $[\Phi_4]_{2\times2} = \frac{\partial \bar{\boldsymbol{\omega}}(t_{k+1})}{\partial \bar{\boldsymbol{\omega}}(\tau)} \approx [I]_{2\times2}$. In order to approximate $[\Phi_2]_{3\times2} = \frac{\partial \boldsymbol{d}(t_{k+1})}{\partial \bar{\boldsymbol{\omega}}(\tau)}$, the discrete state update is used as seen in equation (**??**), where $\boldsymbol{d}_i$ is the sun heading at time τ. It is reminded that $\bar{\boldsymbol{\omega}} = \bar{\boldsymbol{\omega}}_{\mathcal{S}/\mathcal{B}}$, and that $\bar{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}} = -\bar{\boldsymbol{\omega}} = -\bar{\boldsymbol{\omega}}_{\mathcal{S}/\mathcal{B}}$.

$$\boldsymbol{d}_{k+1} = \boldsymbol{d}_i - (t_{k+1} - \tau)[\tilde{\boldsymbol{d}}_i]\bar{\boldsymbol{\omega}}_{\mathcal{B}/\mathcal{N}} \tag{13}$$

$$\boldsymbol{d}_{k+1} = \boldsymbol{d}_i + (t_{k+1} - \tau)[\tilde{\boldsymbol{d}}_i]\bar{\boldsymbol{\omega}} \tag{14}$$

$$\Rightarrow \frac{\partial \boldsymbol{d}(t_{k+1})}{\partial \bar{\boldsymbol{\omega}}(\tau)} = (t_{k+1} - \tau)[\tilde{\boldsymbol{d}}_i] \tag{15}$$

Therefore, assuming the state does not vary over the time between two updates, $[\Phi_2]_{3\times2} = \frac{\partial \boldsymbol{d}(t_{k+1})}{\partial \bar{\boldsymbol{\omega}}(\tau)} = (t_{k+1} - \tau)[\tilde{\boldsymbol{d}}_i]$. This leads to the new $[\Gamma]$ matrix in equation (**??**), which is used for state noise compensation.

$$[\Gamma](t_{k+1}, t_k) = \int_{t_k}^{t_{k+1}} \begin{bmatrix}[\Phi_2]_{3\times2}\\ [\Phi_4]_{2\times2}\end{bmatrix}\mathrm{d}\tau = \Delta t \begin{bmatrix}\frac{\Delta t}{2}[\tilde{\boldsymbol{d}}_k](2,3)\\ [I]_{2\times2}\end{bmatrix} \tag{16}$$

2.5 Measurements

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.

$$\boldsymbol{G}_i(\boldsymbol{X}) = \boldsymbol{n}_i \cdot \boldsymbol{d} \tag{17}$$

$$\boldsymbol{H}(\boldsymbol{X}) = \begin{bmatrix} \boldsymbol{n}_1^T \\ \vdots \\ \boldsymbol{n}_i^T \end{bmatrix} \tag{18}$$

2.6 User initialization

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

- The angle threshold under which the coarse sun sensors do not read the measurement: `FilterContainer.sensorUseThresh = 0.`
- The process noise value, for instance: `FilterContainer.qProcVal = 0.001`
- The measurement noise value, for instance: `FilterContainer.qObsVal = 0.001`
- The threshold in the covariance norm leading to the switch from the EKF update to the linear Kalman Filter update (discussed more closely in the Measurement update part): `FilterContainer.ekfSwitch = 5`
- The initial covariance:

```
Filter.covar =
[0.4, 0., 0., 0., 0., 0.,
0., 0.4, 0., 0., 0., 0.,
0., 0., 0.4, 0., 0., 0.,
0., 0., 0., 0.004, 0., 0.,
0., 0., 0., 0., 0.004, 0.,
0., 0., 0., 0., 0., 0.004]
```

- The initial state : `Filter.state =[0.0, 0.0, 1.0, 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"`

2.7 Inputs and Outputs

The EKF 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 Filter Algorithm

Once the filter has been properly setup in the python code, it can go through it's algorithm. This is done according to the algorithms derived in Reference [**?**].

Initialization

First the filter is initialized. This can be done at any time during a simulation in order to reset the filter.

- Time is set to t_0
- The state $\boldsymbol{X}^*$ is set to the initial state $\boldsymbol{X}_0^*$
- The state error $\boldsymbol{x}$ is set to it's initial value $\boldsymbol{x}_0$
- The covariance P is set to the initial state P_0

Time Update

At some time t_i, if the update filter method is called, a time update will first be executed.

- The state is propagated using the dynamics $\boldsymbol{F}$ with initial conditions $\boldsymbol{X}^*(t_{i-1})$
- Compute the dynamics matrix $A(t) = \left[\frac{\partial \boldsymbol{F}(\boldsymbol{X},t)}{\partial \boldsymbol{X}}\right]^*$ which is evaluated on the reference trajectory
- Integrate the STM, $\dot{\Phi}(t, t_{i-1}) = A(t)\Phi(t, t_{i-1})$ with initial conditions $\Phi(t_{i-1}, t_{i-1}) = I$

  This gives us $\boldsymbol{X}^*(t_i)$ and $\Phi(t_i, t_{i-1})$.

Observation read in

If no measurement is read in at time t_i:

- $\boldsymbol{X}^*(t_i)$ previously computed becomes the most recent reference state
- $\boldsymbol{x}_i = \bar{x}_i = \Phi(t_i, t_{i-1})x_{i-1}$ is the new state error
- $P_i = \bar{P}_i = \Phi(t_i, t_{i-1})P_{i-1}\Phi^T(t_i, t_{i-1})$ becomes the updated covariance

If a measurement is read in, the algorithm computes the observation, the observation state matrix, and the Kalman Gain.

- The observation ($\boldsymbol{Y}_i$) is compared to the observation model, giving the innovation: $\boldsymbol{y}_i = \boldsymbol{Y}_i - G(\boldsymbol{X}_i^*, t_i)$
- Compute the observation matrix along the reference trajectory: $\tilde{H}_i = \left[\frac{\partial \boldsymbol{G}(\boldsymbol{X},t_i)}{\partial \boldsymbol{X}}\right]^*$
- Compute the Kalman Gain $K_i = \bar{P}_i \tilde{H}_i^T \left(\tilde{H}_i \bar{P}_i \tilde{H}_i^T + R_i\right)^{-1}$

Measurement Update

Depending on the covariance, the filter can either update as a classic, linear Kalman Filter, or as the Extended Kalman filter. This is done in order to assure robust and fast filter convergence. Indeed in a scenario with a very large initial covariance, the EKF's change in reference trajectory could delay or inhibit the convergence. In order to remedy this, a few linear updates are performed if the maximum value in the covariance is greater than a user-set threshold.

Linear update:

- The state error is updated using the time updated value: $\boldsymbol{x}_i = \bar{x}_i + K_i \left[\boldsymbol{y}_i - \tilde{H}_i \bar{x}_i\right]$
- The covariance is updated using the Joseph form of the covariance update equation: $P_i = \left(I - K_i \tilde{H}_i\right) \bar{P}_i \left(I - K_i \tilde{H}_i\right)^T + K_i R_i K_i^T$
- The reference state stays the same, and it's propagated value $\boldsymbol{X}^*(t_i)$ becomes $\boldsymbol{X}^*(t_{i-1})$

EKF update:

- The state error is updated using the innovation and the Kalman Gain: $\boldsymbol{x}_i = K_i \boldsymbol{y}_i$
- The reference state is changed by the state error: $\boldsymbol{X}^*(t_i) = \boldsymbol{X}^*(t_i) + \boldsymbol{x}_i$
- The covariance is updated using the Joseph form of the covariance update equation: $P_i = \left(I - K_i \tilde{H}_i\right) \bar{P}_i \left(I - K_i \tilde{H}_i\right)^T + K_i R_i K_i^T$
- The new reference state is now used $\boldsymbol{X}^*(t_i)$ becomes $\boldsymbol{X}^*(t_{i-1})$

4 Test Design

The unit test for the sunlineEKF module is located in:

`fswAlgorithms/attDetermination/sunlineEKF/_UnitTest/test_SunlineEKF.py`

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

The test is split up into 4 subtests, the last one is parametrized in order to test different scenarios. The first test creaks up all of the individual filter methods and tests them individually. 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 in a general case. The final test is a full filter test.

4.1 `sunline_individual_test`

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.

- Dynamics Matrix: This method computes the dynamics matrix A. Tolerance to absolute error $\epsilon = 10^{-10}$.

  Passed

- State and STM propagation: This method propagates the state using the $\boldsymbol{F}$ function as well as the STM using $\dot{\Phi} = A\Phi$. Tolerance to absolute error $\epsilon = 10^{-10}$.

  Passed

- H and y propagation: This method computes the H matrix, and compares the measurements to the expected measurements given the state. Tolerance to absolute error $\epsilon = 10^{-10}$.

  Passed

- Kalman gain: This method computes the K matrix. Tolerance to absolute error $\epsilon = 10^{-10}$.

  Passed

- EKF update: This method performs the measurement update in the case of an EKF. Tolerance to absolute error $\epsilon = 10^{-10}$.

  Passed

- Linear Update: This method performs the measurement update in the linear case. Tolerance to absolute error $\epsilon = 10^{-10}$.

  Passed

4.2 `StatePropStatic`

This test runs the filter with no measurements. It initializes with a zeroed state, and assures that at the end of the simulation all values are still at zero. Plotted results are seen in Figure **??**.

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

Passed

4.3 `StatePropVariable`

This test also takes no measurements in, but gives a random state with rate of 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 for the test and the code overlap perfectly.

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

Passed

4.4 `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.

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

Tolerance to absolute error with measurement noise: $\epsilon = 10^{-2}$

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 evolution of the state vector compared to the true values. The parts were there is a slight delay is due to the fact that no observations are read in.

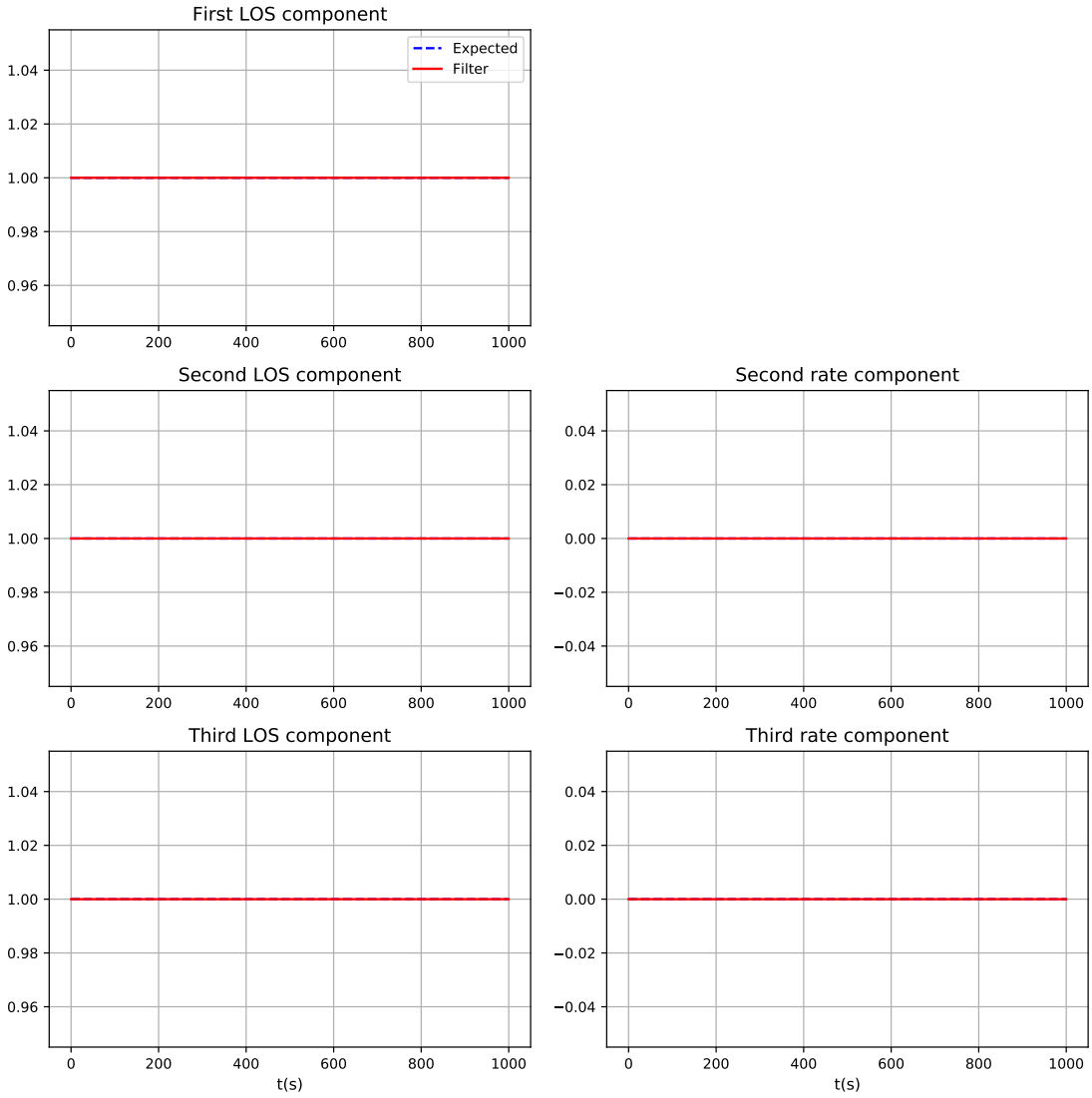

Fig. 2: States vs true states in static case

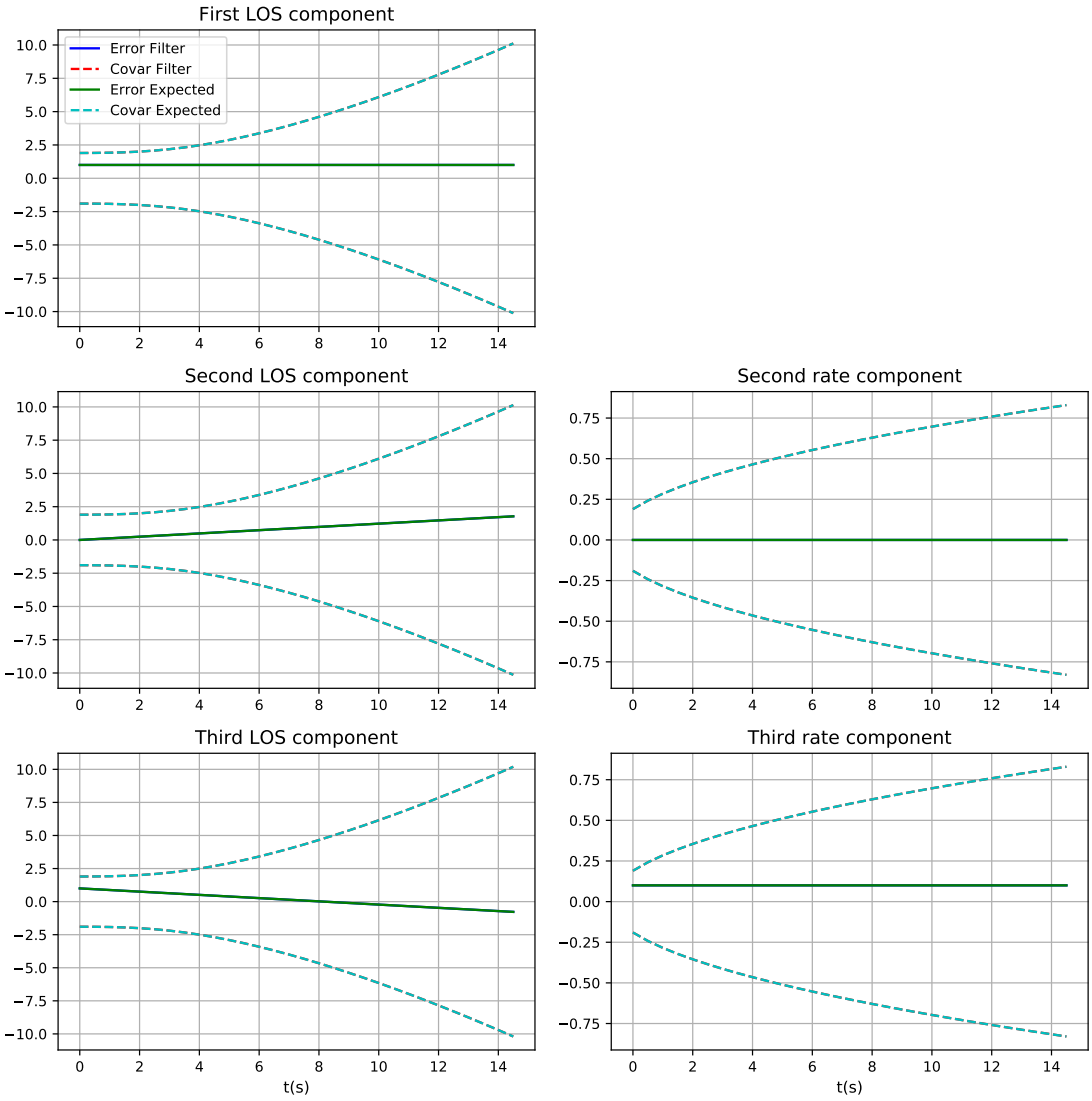

Fig. 3: State error and covariance vs expected Values

Figure **??** shows the post fit residuals for the filter, with the 3σ measurement noise values. We see that the observations are read in well an that the residuals are brought back down to noise. We do observe a slight bias in the noise. This could be due to the equations of motion, and is not concerning.

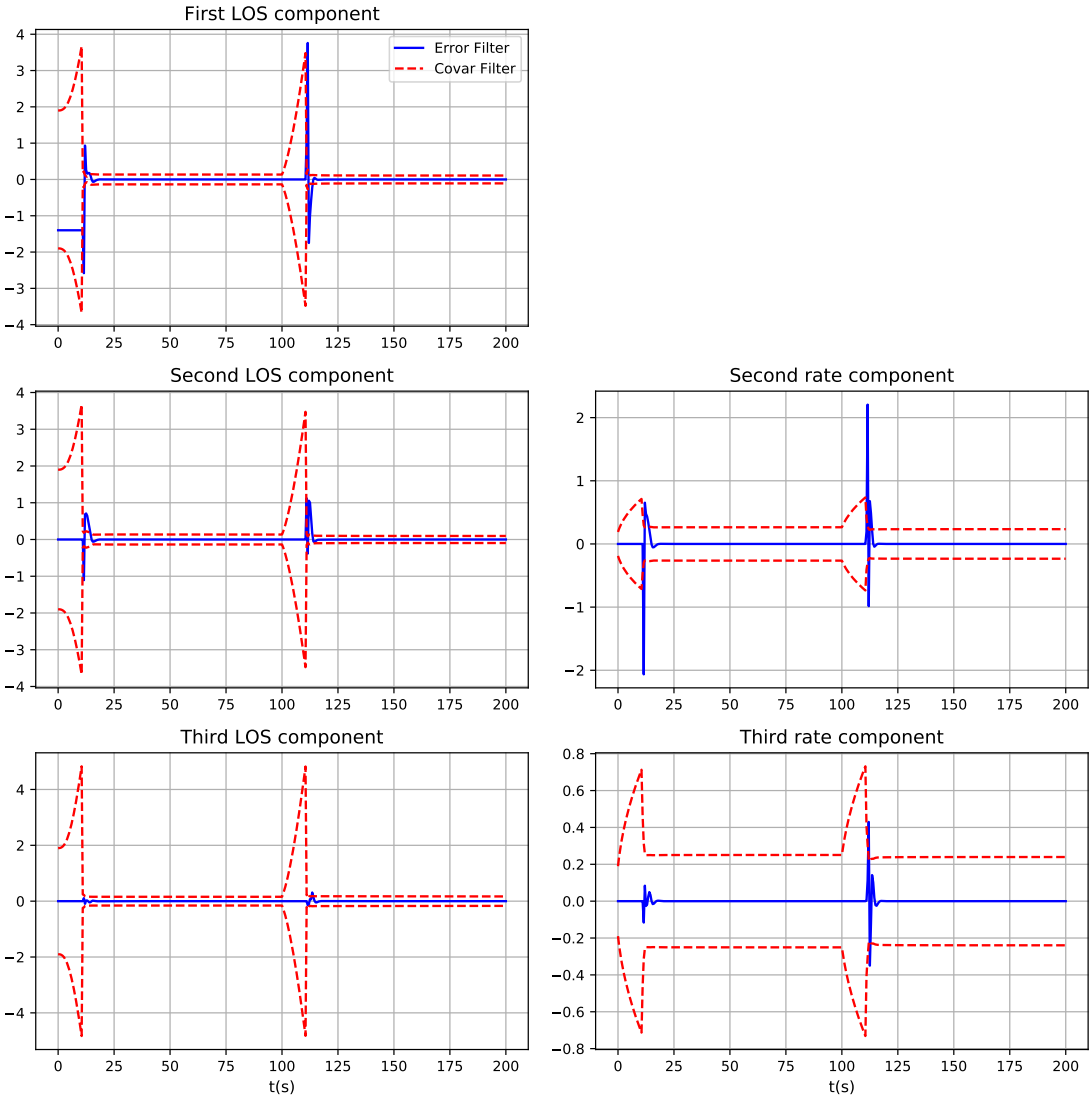

Fig. 4: State error and covariance

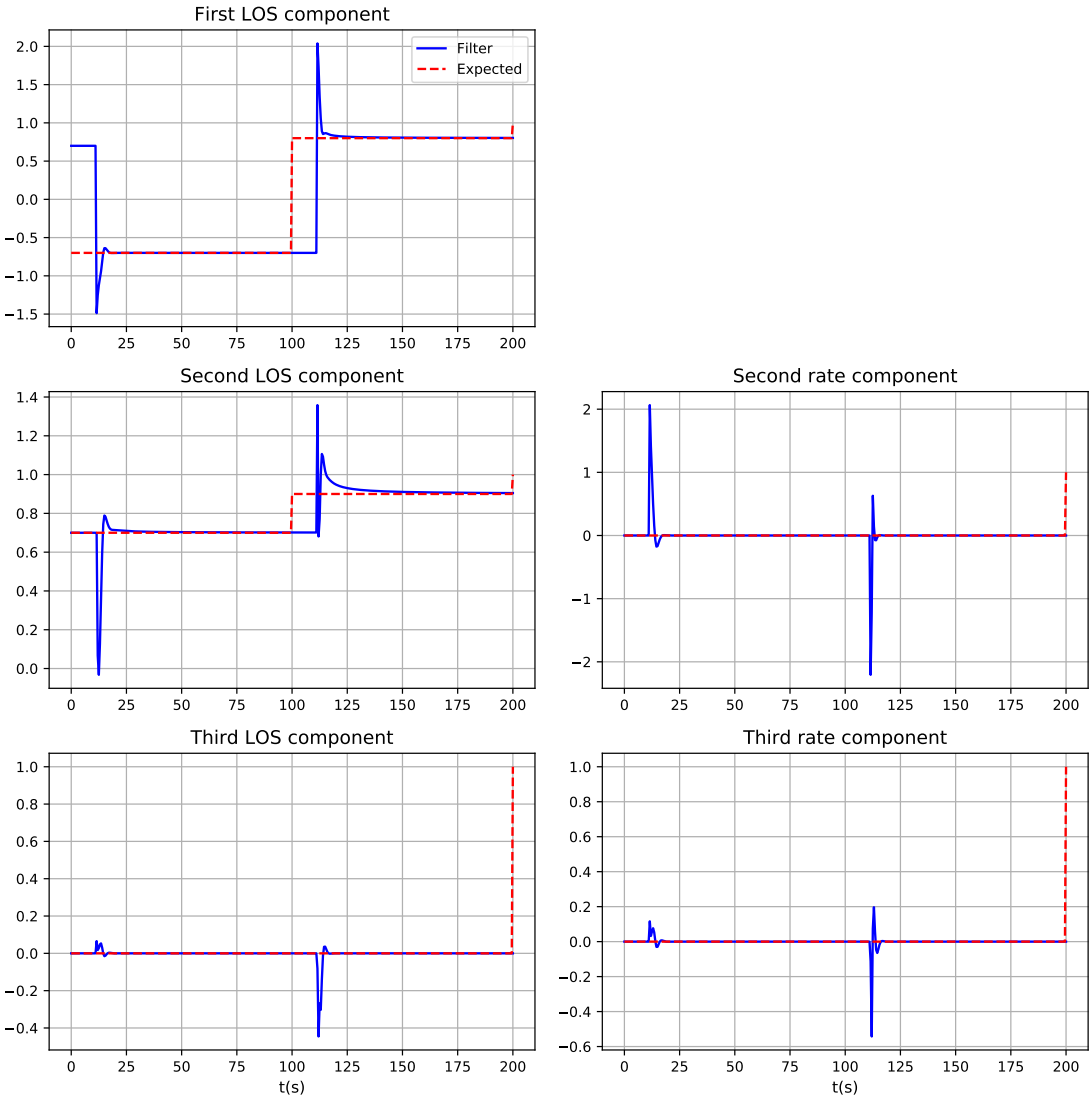

Fig. 5: States tracking target values

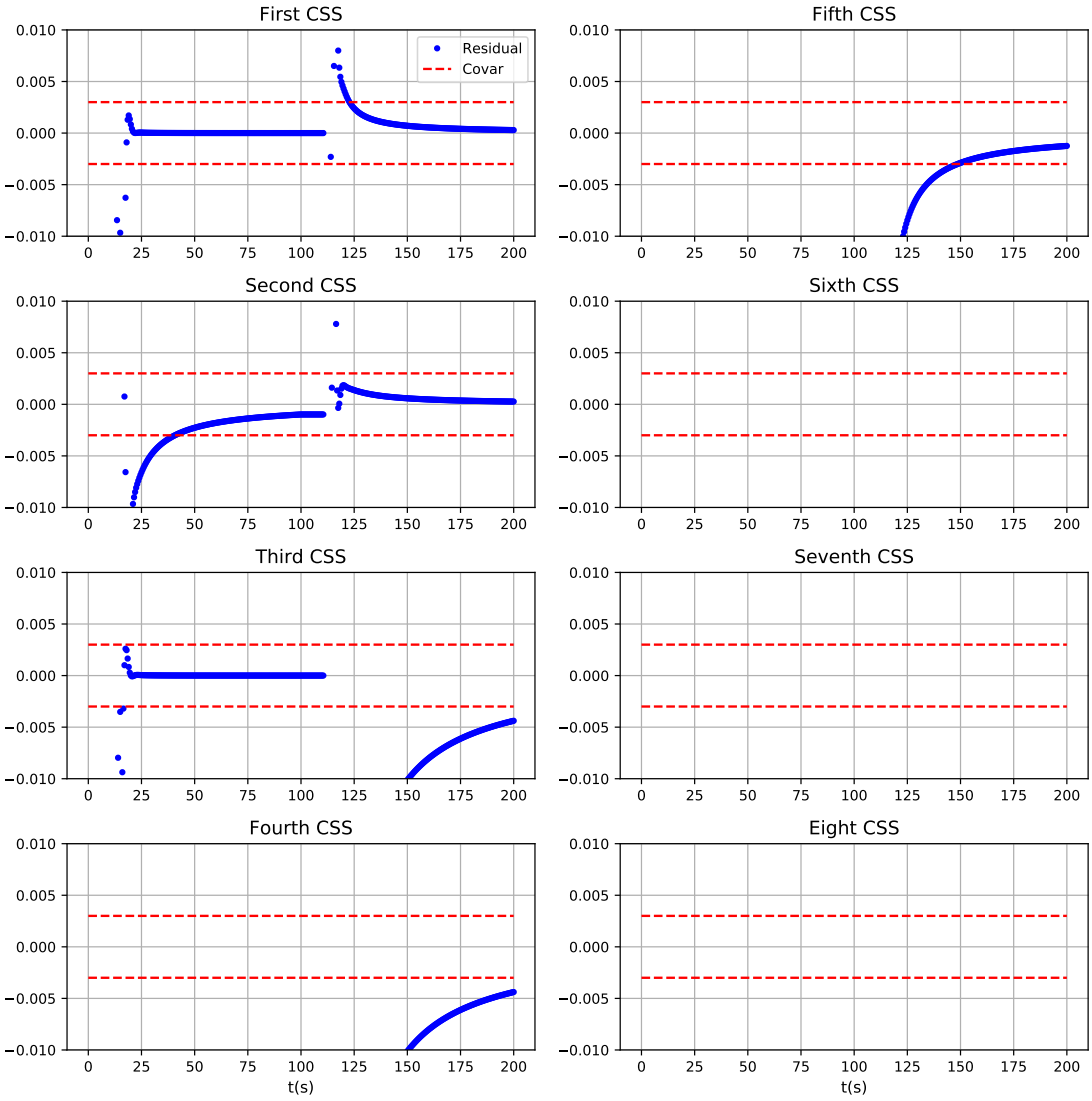

Fig. 6: Post Fit Residuals