# Question #4

## Implementing Linearized Kalman Filter (LKF)

To implement the LKF for this Cooperative Localization problem we define an LKF function which performs the prediction [Equation ( 1 )] and correction [Equation ( 2 )] step.

The function takes in:

* the ground truth values (state dynamics and measurements),
* the nominal state and measurement trajectories without process or measurement noise,
* the process noise covariance () for tuning and,
* initial values for and .

The function outputs:

* the state estimates (),
* measurement estimates ,
* estimation errors ( and ) and,
* sequence of covariance matrices for state and measurement estimates ( and)

The LKF runs the following set of equations in a loop for steps:

|  |  |  |
| --- | --- | --- |
|  |  | ( ) |

|  |  |  |
| --- | --- | --- |
|  |  | ( ) |

This function is included in a Monte Carlo run which feeds the ground truth models, nominal state and measurement trajectories, and the DT state space matrices linearized around the nominal trajectories () according to the methods from Question #2 and Question #3.

Within the LKF function, we wrapped the nominal trajectories, truth models, and estimated state perturbations corresponding to angles.

## Truth Model Testing (TMT) for LKF

For the TMT we do 50 Monte Carlo () runs. This is an appropriate number of Monte Carlo runs as we will have enough data sets for performing an unbiased NEES and NIS test.

The truth model is a simulated run of the nonlinear model with the process and measurement noise generated using the covariances uploaded on Canvas. We seed for the ground truth values for every Monte Carlo run from and then and then for every subsequent value the process noise is obtained from covariance matrix ‘Qtrue’.

Additionally, for the multiple runs in the TMT we feed the LKF the following initial values:

|  |  |  |
| --- | --- | --- |
|  |  | ( ) |

These initial conditions were selected for the following reasons: we are given the nominal and truth state trajectories, and hence the initial perturbation can be derived from those values; furthermore, we have a sufficient degree of certainty of our initial state perturbation such that the state perturbation covariance is finite (). And also, the angles cannot exceed 180o or go below -180o in radians.

In addition to this, in the LKF, the covariance matrix for the measurement noise () is set to be the one uploaded on canvas, i.e. Rtrue, as that information is generally known to us from the sensors being used.

## Tuning

There are two aspects that we need to keep in mind while tuning the Kalman filter. First, we need to make sure that the Kalman filter ‘works properly’, i.e. the estimate error averages at 0, and the bounds are converging. Second, we also look at the NEES and NIS chi-square tests and make sure they are within the confidence intervals.

We calculate the confidence intervals on MATLAB with significance level . We chose this significance level to provide a less stringent condition for proving whether the LKF is doing its job (i.e. having a low enough false-alarm probability). This was because we found that the LKF is not good enough for estimating this nonlinear system. The bounds are calculated using the MATLAB function ‘chi2inv’. As a result, for Monte Carlo runs, states, measured values, the chi-square confidence bounds are:

|  |  |  |
| --- | --- | --- |
|  |  | ( ) |

The and values are obtained by averaging the and values are all steps, which are obtained from the following equations:

While tuning the LKF we vary our predicted process noise covariance matrix () until the two conditions in the beginning of this subsection are satisfied. We first start with and run the truth model tests and check if the KF works (by looking at the state and measurement estimate errors) and the NEES and NIS tests are satisfied. The result showed that neither of the conditions were met, hence we moved to then to and then fine-tuned the main diagonal parameters in this way until the conditions were sufficiently satisfied.

In tuning, we focused on the consistency of the estimates for the and states as they appeared to be the states most sensitive to changes in .

In the end, the process noise covariance matrix for the LKF that best satisfied the conditions was found to be:

|  |  |  |
| --- | --- | --- |
|  |  | ( ) |

Before we move forward, the ‘best satisfying ’ is used lightly because, over multiple trials, it was found that the LKF cannot successfully estimate the states for this problem even after multiple variations. It is predicted that more fine tuning is required, however with 6 states to work with and limited time, we think that it is better to look for alternate options.

Referring to Figure 4 and Figure 5, we see that the LKF has done a decent job in estimating the measurements however the range of errors for the positions of the UAV and as well as the relative distance measurement are large. Furthermore, these measurement estimate errors do not average to zero and converge, as we can see that as time progresses, the errors oscillate.

This error could not be reduced regardless of the corresponding elements selected, and this resulted in the estimation errors carrying over to the state estimates. For instance, the state estimate errors for and (UGV positions) in Figure 2 also grows over time instead of averaging to zero and converging. Just observing the state estimates for and in Figure 3, we can see that they maintain similar behavior to the truth model however as time progresses the state estimates experience significant deviation.

On another note, the state estimates for the UAV’s positions appear to match the truth model, however a closer look at the state estimate errors in Figure 2 shows that the estimate is getting further from the truth model as time progresses. We do not see this in the state estimate curves because the magnitudes of and are large, i.e. approximately ranging between -100m and 100m. We predict that we will observe a significant deviation in state estimates from the truth model if the duration of the operation is increased.

Additionally, the angle estimates and are not significantly affected as they are more dependent on the control inputs.

Now while the state estimation errors for , , and do not converge and average to zero, their covariances do converge. In fact, all the covariances of the state estimates () and measurement estimates () converge.

We will now look at the NEES and NIS tests for the LKF. Referring to Figure 1, we can see that the NIS test is, to and extent, being satisfied, however as the time progresses the NIS data exceed the confidence intervals. This overlaps with the previous observation of how the measurement estimates for and as well as the relative distance measurement deviate significantly as time progresses.

We can also see how the effect of this deviation carries over to the state estimate errors by looking at the NEES test where the points are within the confidence interval in the beginning but very quickly, they go beyond the limit.

This tells us that the current configuration for the LKF always fails the NEES test but passes the NIS tests for a limited period. To reiterate, there is still room for more fine tuning of the LKF, but we believe that rather than wasting more time in tuning this further, a better alternative would be preferred.

## Discussion on LKF for Estimating Nonlinear System

The tuning procedure was arduous and complicated. However, after multiple attempts, we can conclude that the LKF is not a good enough filter to estimate the nonlinear system for cooperative localization.

We derived this conclusion because we were not able to tune the LKF to satisfy the two conditions: the LKF ‘works properly’ and satisfies the NEES and NIS chi-square tests.

First we can conclude that the LKF fails to estimate the states for the system for an extended period of time. We tested this by increasing the time duration to >100s and found that not only did the NIS values exceed the bounds, but the state estimates also showed significant deviation from the truth model. This issue may be because the estimates are significantly deviating from the nominal trajectories calculated through linearization (first order approximates). This is probably because the measurement noise covariances corresponding to and as well as the relative distance measurement are 36, 36 and 64 respectively causing the truth model values to be exceedingly noisy. This means that calculating the nominal trajectory once in the beginning provides a poor basis for comparison. Hence, we believe that the EKF would perform better for this problem where me calculate the nominal trajectory every time step.

We modified the internals of the truth model generator to evaluate the effect of having a more precise sensor, but it was found that a better sensor could not improve the state estimates obtained from the LKF. This tells us that the LKF is not an appropriate estimator for this nonlinear system.

This reflects on the NEES and NIS tests because while the system passes the NIS tests in the beginning, the NEES test completely fails. Hence, we know that the LKF completely fail and is statistically inconsistent in estimating the states. Therefore, if we only have the truth model sensor data, the LKF can provide a reasonable estimate for the states but only for the earlier time ranges.

The MATLAB codes for the LKF implementation, the Monte Carlo runs, and the NEES and NIS tests are attached in Appendix D.

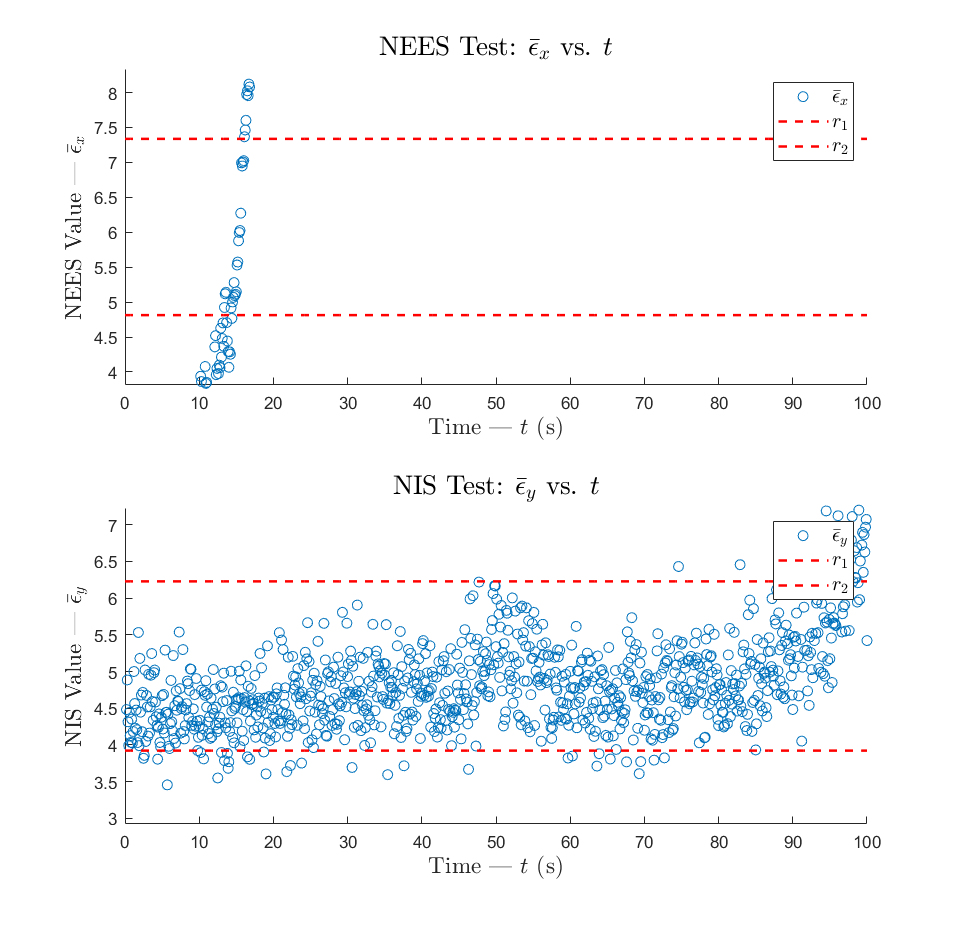


Figure – NEES and NIS chi-square test for LKF Monte Carlo runs with Equations ( 3 )( 4 )( 5 ) applied

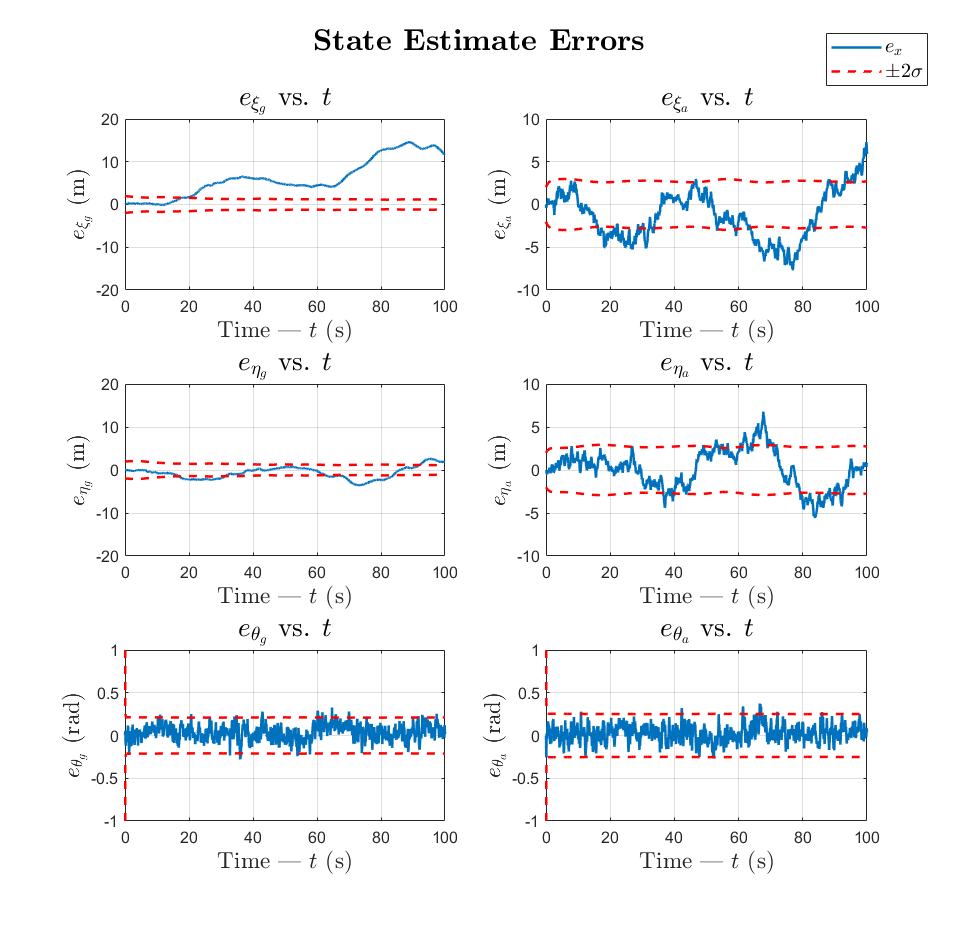


Figure – State estimate errors and bounds of sample LKF Monte Carlo run with Equations ( 3 )( 4 )( 5 ) applied

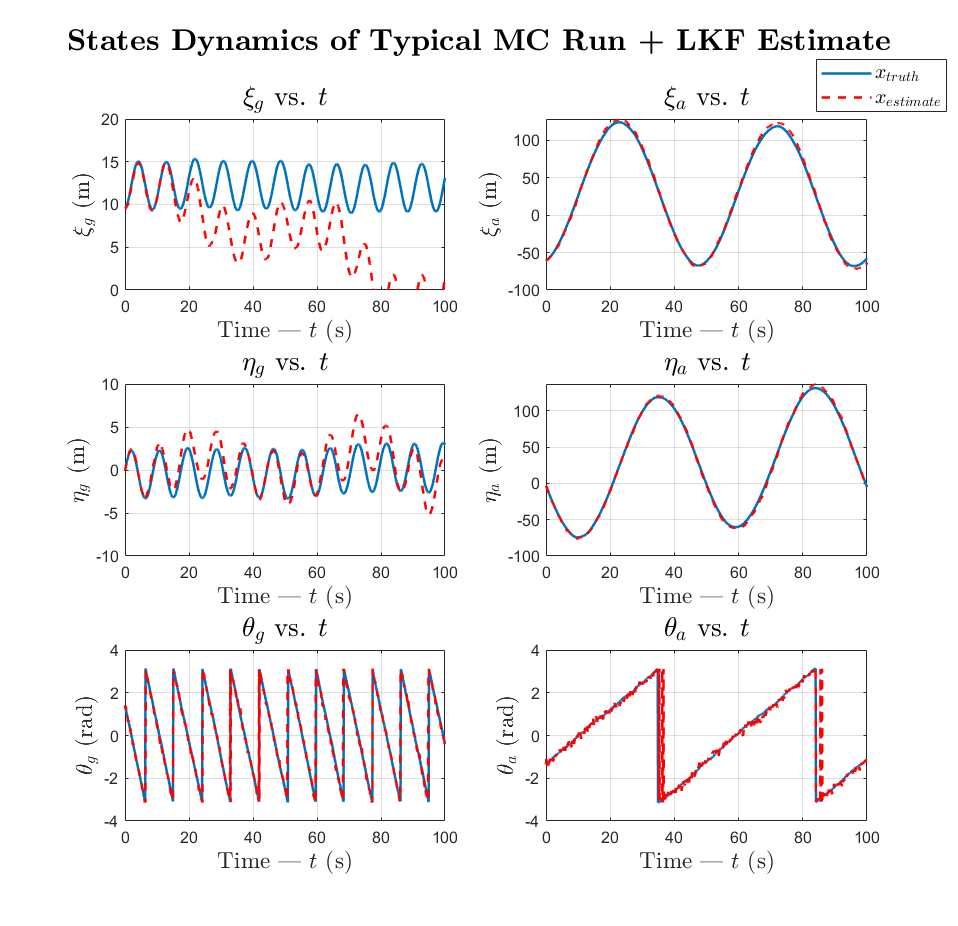


Figure – State estimates and state ground truth of sample LKF Monte Carlo run with Equations ( 3 )( 4 )( 5 ) applied

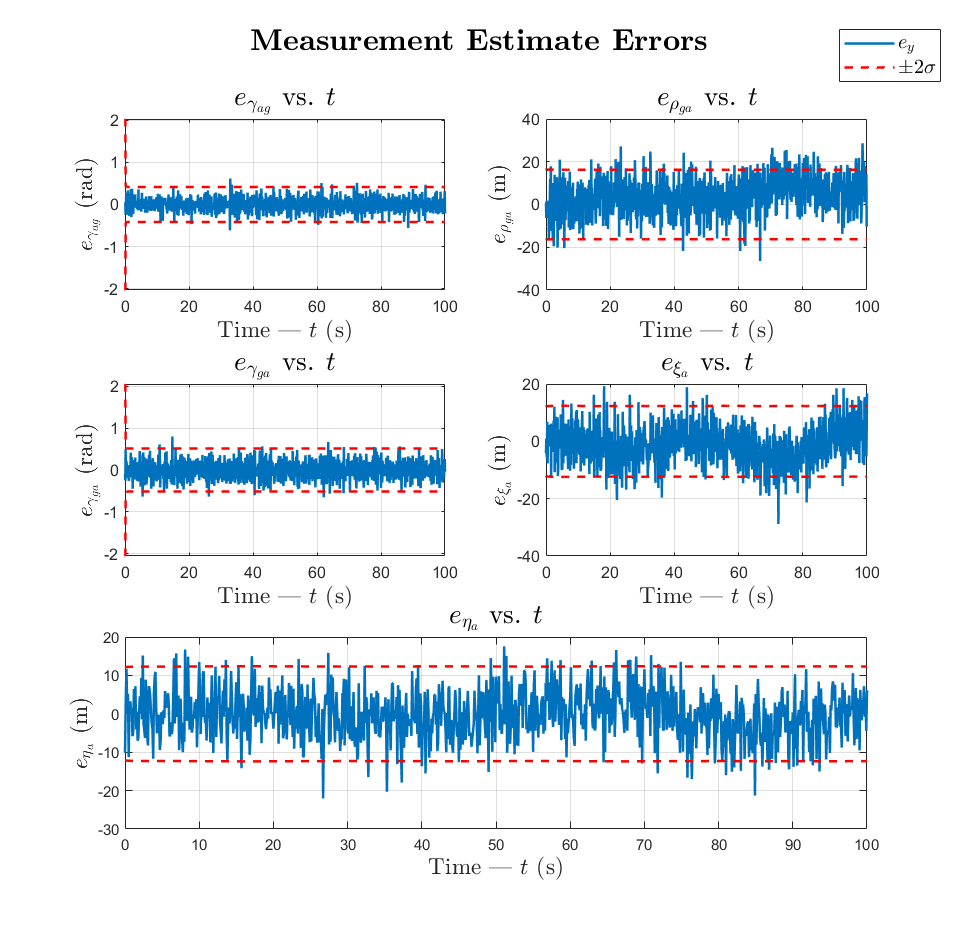


Figure – Measurement estimates and ground truth of sample LKF Monte Carlo run with Equations ( 3 )( 4 )( 5 ) applied

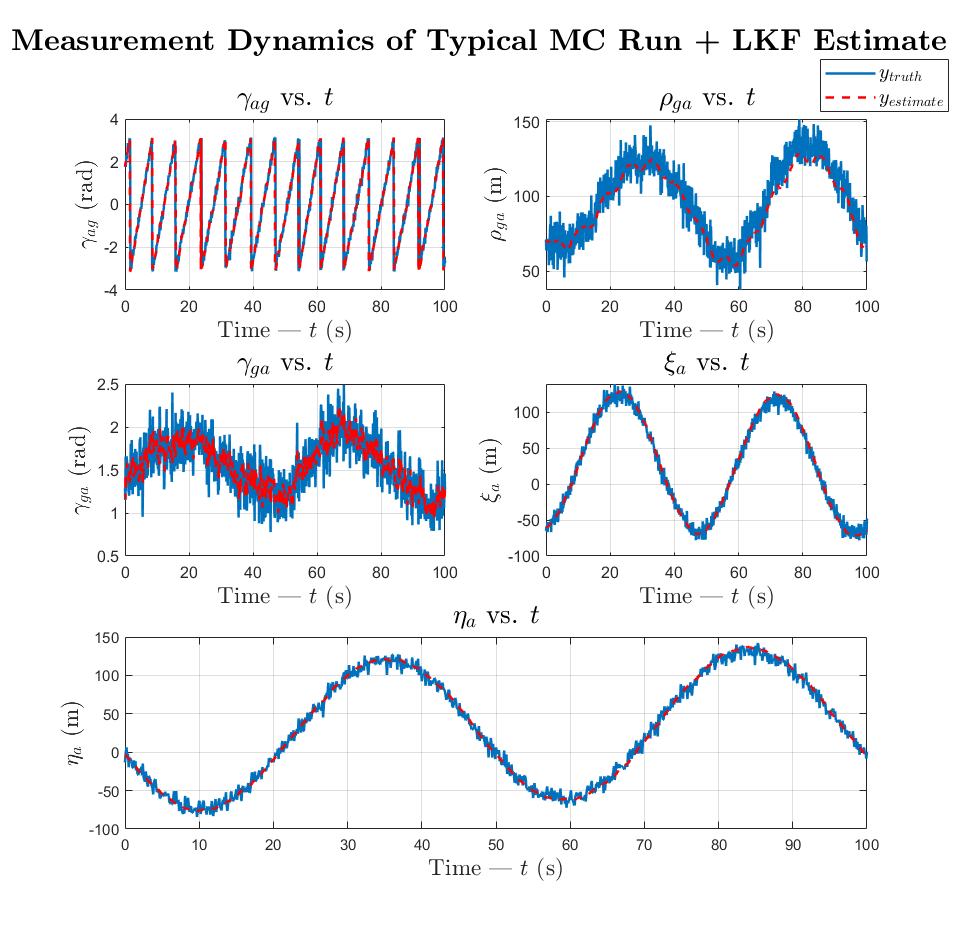


Figure – Measurement estimates and ground truth of sample LKF Monte Carlo run with Equations ( 3 )( 4 )( 5 ) applied