[ASEN 5044] Statistical Estimation of Dynamical Systems

Final Report

Fall 2020

Group Members:

Joshua Malburg

Junior Sundar

Contents

Contribution to Project

Question Number	Allocated to
1	Joshua
2	Joshua
3	Junior
4	Joshua
5	Junior
6	Joshua (coding) /Junior (analysis)
AQ13	Junior

Question #1

Continuous-time System Equations

The non-linear equations of motions of our UGV-UAV two-agent system are provided in the problem description as:

$$\dot{\xi}_g = \upsilon_g \cos\theta_g + \widetilde{\omega}_{x,g} \quad (1)$$

$$\dot{\eta}_g = \upsilon_g \sin\theta_g + \widetilde{\omega}_{y,g} \quad (2)$$

$$\dot{\theta}_g = \frac{\upsilon_g}{L}\tan\phi_g + \widetilde{\omega}_{\omega,g} \quad (3)$$

$$\dot{\xi}_a = \upsilon_a \cos\theta_a + \widetilde{\omega}_{x,a} \quad (4)$$

$$\dot{\eta}_a = \upsilon_a \sin\theta_a + \widetilde{\omega}_{y,a} \quad (5)$$

$$\dot{\theta}_a = \omega_a + \widetilde{\omega}_{\omega,a} \quad (6)$$

The inputs $[\upsilon_g, \phi_g, \upsilon_a, \omega_a]^T$ to our system are the UGV linear velocity (m/s), UGV steering angle (rad), UAV linear velocity (m/s) and UAV angular rate (rad/s). Our state vector $[\xi_g, \eta_g, \theta_g, \xi_a, \eta_a, \theta_a]^T$ is comprised of the easting position (m), northing position (m) and heading angle (rad) for both the UGV and UAV; each state equation is assumed to be corrupted by AWGN. For measurements we are provide the UAV easting and northing position along with the UGV-UAV relative azimuth angles and range; the output sensing equations are then:

$$\theta_{ga} = \tan^{-1}\left(\frac{\eta_a - \eta_g}{\xi_a - \xi_g}\right) - \theta_g + \tilde{v}_{\theta,ga} \quad (7)$$

$$r = \sqrt{(\xi_g - \xi_a)^2 + (\eta_g - \eta_a)^2} + \tilde{v}_r \quad (8)$$

$$\theta_{ag} = \tan^{-1}\left(\frac{\eta_g - \eta_a}{\xi_g - \xi_a}\right) - \theta_a + \tilde{v}_{\theta,ag} \quad (9)$$

$$\xi_a + \tilde{v}_{\xi,a} \quad (10)$$

$$\eta_a + \tilde{v}_{\eta,a} \quad (11)$$

Our system can be expressed in standard non-linear state-space form by stacking the NL state equations and measurements from above in to $\mathcal{F}$ and h matrices:

$$\dot{x} = \begin{bmatrix} \mathcal{F}_1(x,u,w,t) \\ \vdots \\ \mathcal{F}_n(x,u,w,t) \end{bmatrix} \quad (12)$$

$$y = \begin{bmatrix} h_1(x,u,v,t) \\ \vdots \\ h_n(x,u,v,t) \end{bmatrix} \quad (13)$$

To find the linear CT perturbation model of our system we linearize about the nominal operation point provided in the problem description and find the partial derivatives / Jacobians (see Appendix A for supporting derivations):

$$\left.\frac{\delta \mathcal{F}}{\delta x}\right|_{nom} = \begin{bmatrix} \frac{\delta \mathcal{F}_1}{\delta x_1} & \cdots & \frac{\delta \mathcal{F}_1}{\delta x_n} \\ \vdots & \ddots & \vdots \\ \frac{\delta \mathcal{F}_n}{\delta x_1} & \cdots & \frac{\delta \mathcal{F}_n}{\delta x_n} \end{bmatrix} = \begin{bmatrix} 0 & 0 & -u_1\sin(x_3) & 0 & 0 & 0 \\ 0 & 0 & u_1\cos(x_3) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -u_3\sin(x_6) \\ 0 & 0 & 0 & 0 & 0 & u_3\cos(x_6) \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (14)$$

$$\left.\frac{\delta \mathcal{F}}{\delta u}\right|_{nom} = \begin{bmatrix} \frac{\delta \mathcal{F}_1}{\delta u_1} & \cdots & \frac{\delta \mathcal{F}_1}{\delta u_n} \\ \vdots & \ddots & \vdots \\ \frac{\delta \mathcal{F}_n}{\delta u_1} & \cdots & \frac{\delta \mathcal{F}_n}{\delta u_n} \end{bmatrix} = \begin{bmatrix} \cos(x_3) & 0 & 0 & 0 \\ \sin(x_3) & 0 & 0 & 0 \\ \frac{1}{L}\tan\phi_g & \frac{u_1}{L}(\tan^2(u_2+1)) & 0 & 0 \\ 0 & 0 & \cos(x_6) & 0 \\ 0 & 0 & \sin(x_6) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (15)$$

$$\left.\frac{\delta h}{\delta u}\right|_{nom} = \begin{bmatrix} \frac{\delta h_1}{\delta u_1} & \cdots & \frac{\delta h_1}{\delta u_n} \\ \vdots & \ddots & \vdots \\ \frac{\delta h_n}{\delta u_1} & \cdots & \frac{\delta h_n}{\delta u_n} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (16)$$

$$\left.\frac{\delta h}{\delta x}\right|_{nom} = \begin{bmatrix} \frac{\delta h_1}{\delta x_1} & \cdots & \frac{\delta h_1}{\delta x_n} \\ \vdots & \ddots & \vdots \\ \frac{\delta h_n}{\delta x_1} & \cdots & \frac{\delta h_n}{\delta x_n} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{x_5-x_2}{(x_1-x_4)^2+(x_2-x_5)^2} & \frac{x_1-x_4}{(x_1-x_4)^2+(x_2-x_5)^2} & -1 & \frac{x_2-x_5}{(x_1-x_4)^2+(x_2-x_5)^2} & \frac{x_4-x_1}{(x_1-x_4)^2+(x_2-x_5)^2} & 0 \\ \frac{x_1-x_4}{\sqrt{(x_1-x_4)^2+(x_2-x_5)^2}} & \frac{x_2-x_5}{\sqrt{(x_1-x_4)^2+(x_2-x_5)^2}} & 0 & \frac{x_4-x_1}{\sqrt{(x_1-x_4)^2+(x_2-x_5)^2}} & \frac{x_5-x_2}{\sqrt{(x_1-x_4)^2+(x_2-x_5)^2}} & 0 \\ \frac{x_5-x_2}{(x_1-x_4)^2+(x_2-x_5)^2} & \frac{x_1-x_4}{(x_1-x_4)^2+(x_2-x_5)^2} & 0 & \frac{x_2-x_5}{(x_1-x_4)^2+(x_2-x_5)^2} & \frac{x_4-x_1}{(x_1-x_4)^2+(x_2-x_4)^2} & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad (17)$$

The resulting CT linear matrices F, G and H are $n \times n$, $n \times m$ and $n \times p$, where n equals the number of states (6), m equals the number of inputs (4) and p equals the number of measurements (5).

Question #2

Linear Discrete-time System Equations

If our time-step is small we can use Euler integration to approximate the state transition function which enables us to define the DT linear matrices as a function of the CT Jacobians found in Question #1. For the provided nominal state vector $x_{nom} = \left[10, 0, \frac{\pi}{2}, -60, 0, -\frac{\pi}{2}\right]^T$ and input vector $\left[2, -\frac{\pi}{18}, 12, \frac{\pi}{25}\right]^T$ our DT linearized matrices are then:

$$\tilde{F}_k = I + \triangle T \left.\frac{\delta \mathcal{F}}{\delta x}\right|_{nom[k]} = \begin{bmatrix} 1 & 0 & -0.2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1.2 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (18)$$

$$\tilde{G}_k = \triangle T \left.\frac{\delta \mathcal{F}}{\delta u}\right|_{nom[k]} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0.1 & 0 & 0 & 0 \\ -0.0353 & 0.4124 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -0.1 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix} \quad (19)$$

$$\tilde{H}_k = \left.\frac{\delta h}{\delta x}\right|_{nom[k]} = \begin{bmatrix} 0 & 0.0143 & -1 & 0 & -0.0143 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0.0143 & 0 & 0 & -0.0143 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad (20)$$

Our system is not time-invariant because our matrices are a function of input and state and therefore the nominal point is different at each time step.

Question #3

DT Nonlinear Model

We simulate the nonlinear model using the '`ode45()`' function on MATLAB. We define the nonlinear dynamics model function in the code provided in Appendix C '`NL_DynModel`'. The resulting NL state dynamics simulation assuming no process and measurement noise is shown in Figure 1. Here we set the initial state equal to the specification provided:

$$x(0) = \begin{bmatrix} 10 & 0 & \frac{\pi}{2} & -60 & 0 & -\frac{\pi}{2} \end{bmatrix}^T \quad (21)$$

This represents the nominal state trajectory.

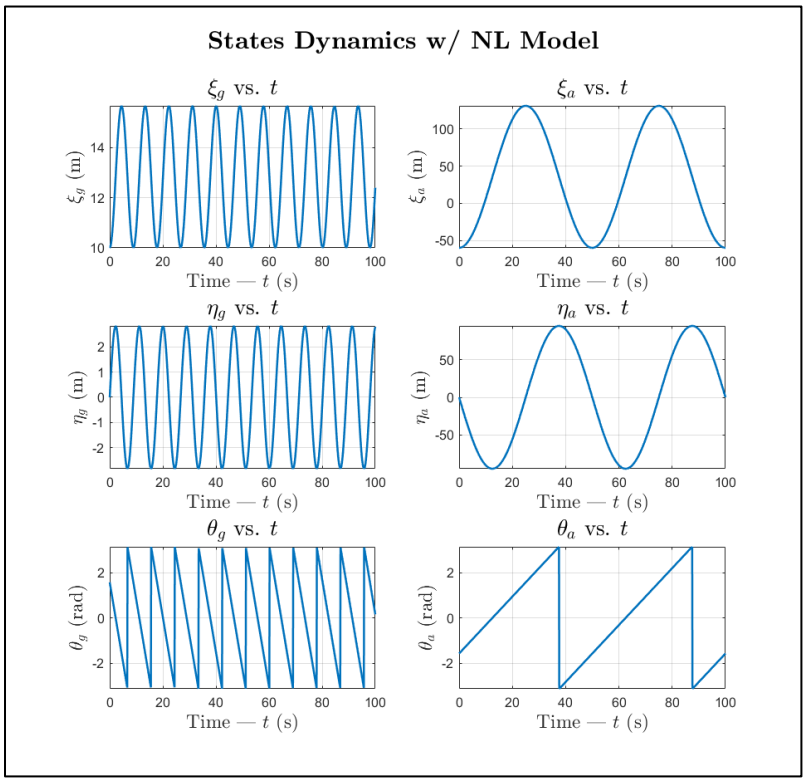

Figure 1 – State dynamics simulation with nonlinear model (using ode45)

Note here that the angles θ_g and θ_a have been wrapped to within $[-\pi, \pi]$.

The nonlinear measurement model function is defined in Appendix C '`NL_MeasModel`'. This function takes in the states at step $'k'$ and outputs the sensor readings.

The resulting NL measurement dynamics without process and measurement noise is shown in Figure 2. This represents the nominal measurements trajectory.

Again the angles γ_{ag} and γ_{ga} have been wrapped within $[-\pi, \pi]$.

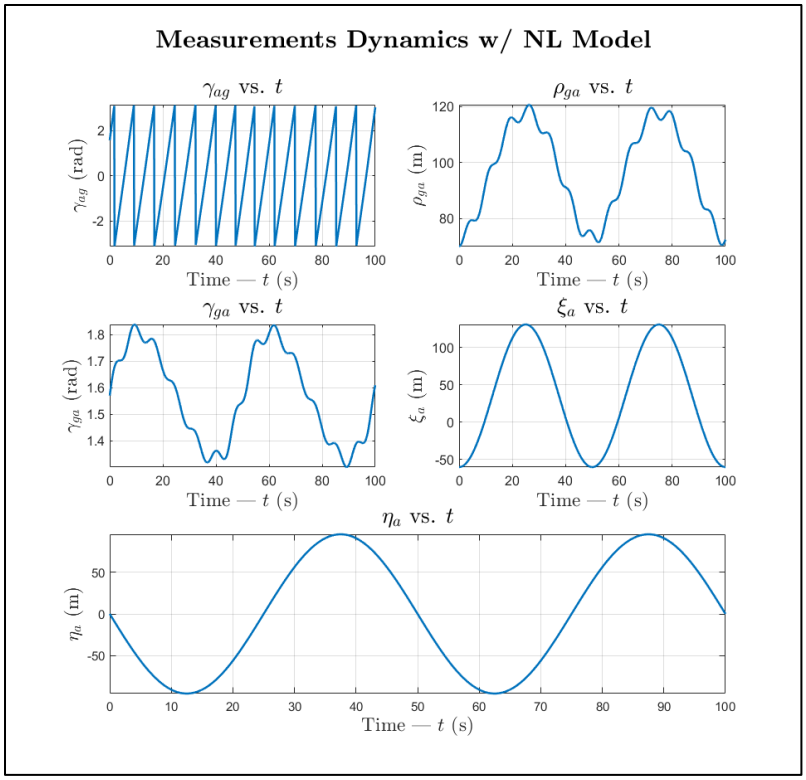

Figure 2 – Measurement dynamics simulation with nonlinear model

DT Linearized Model

We simulate the linearized DT model using the methodology prescribed in Question #1 and Question #2. To perform the linearization, we define the following from specification:

$$x_{nom}(0) = \begin{bmatrix} 10 & 0 & \frac{\pi}{2} & -60 & 0 & -\frac{\pi}{2} \end{bmatrix}^T \quad (22)$$

$$u_{nom} = \begin{bmatrix} 2 & -\frac{\pi}{18} & 12 & \frac{\pi}{25} \end{bmatrix}^T \quad (23)$$

$$\delta x(0) = [0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0.1]^T \quad (24)$$

The transition matrices for the DT linearization model at step 'k' are calculated in Appendix C '`Linearize`' using $x_{nom}(k)$ and u_{nom}. The linearized state and measurement dynamics and perturbations are calculated in Appendix C '`DT_L_Model`'. $\delta x(0)$ in Equation (24) was selected because its small enough to ensure the linearization does not deviate from nominal trajectory

The graph in Figure 3 plots the state evolution of the linearized DT model. Although the state evolution closely matches the NL model's nominal trajectory, there is in fact a perturbation from the nominal trajectory shown in Figure 5. The graph in Figure 4 plots the measurement dynamics of the linearized DT model and the sensor readings perturbations are shown in Figure 6.

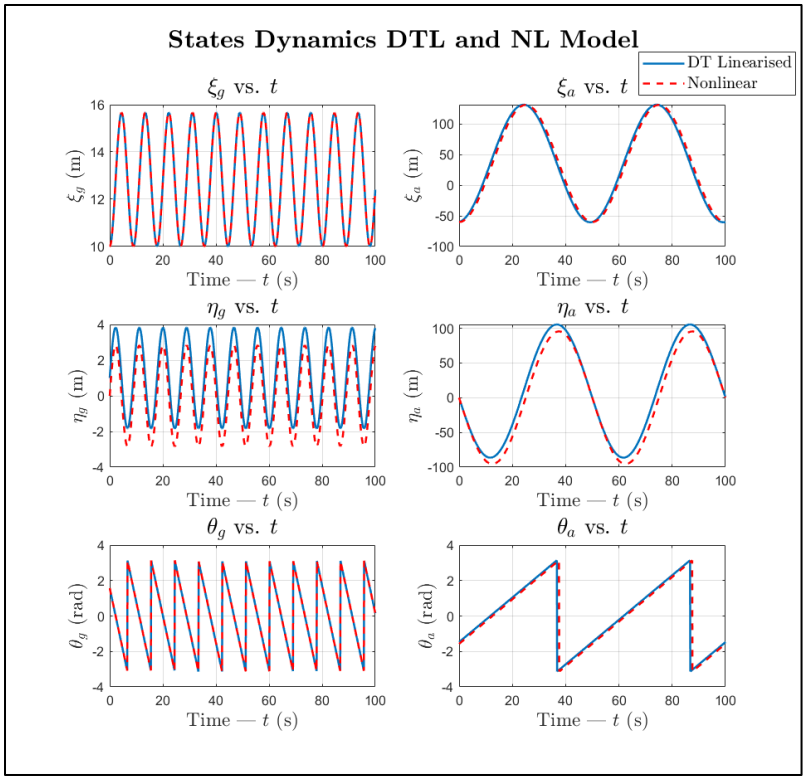

Figure 3 – State dynamics simulation with DT linearized model

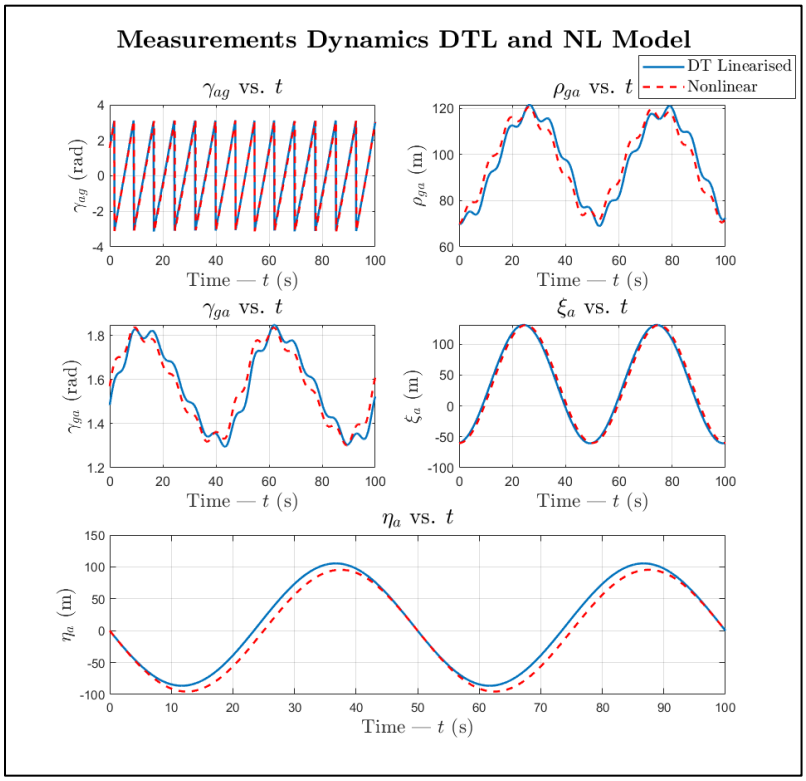

Figure 4 –Measurement dynamics simulation with DT linearized model

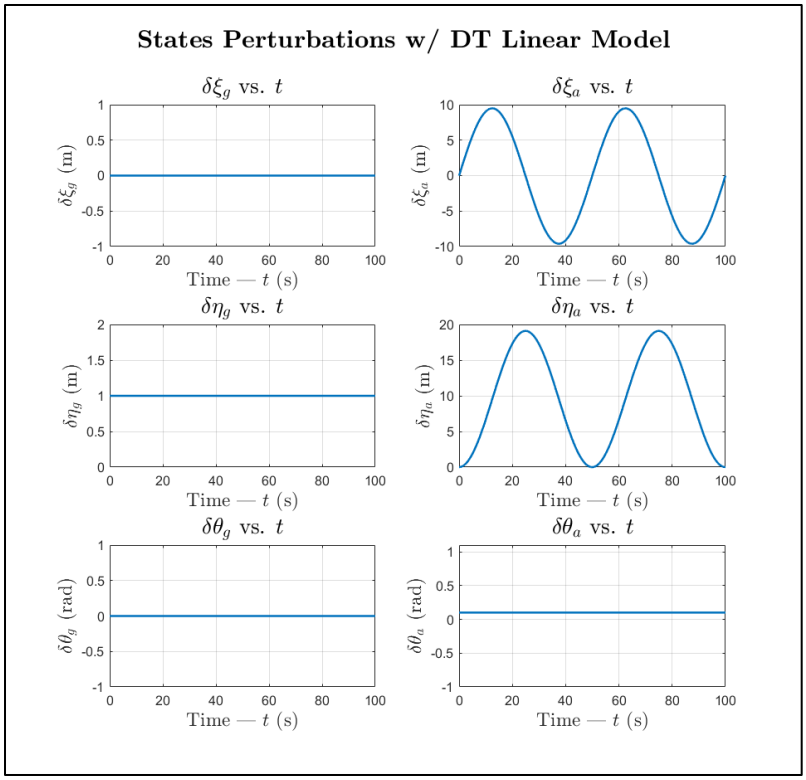

Figure 5 – State dynamics perturbations with DT linearized model

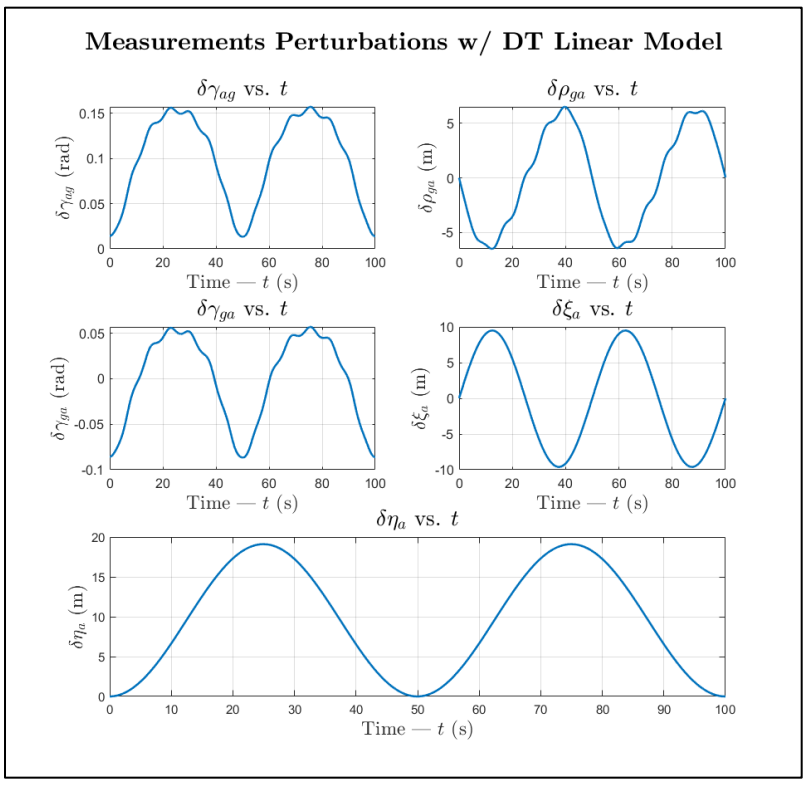

Figure 6 - Measurement dynamics perturbations with DT linearized model\

The state perturbation graphs show that ξ_g, η_g, θ_g and θ_a are constant and unchanging while ξ_a and η_a are varying in an oscillatory manner. This perturbation is transferred through to the sensor readings as well as all the sensor outputs are calculated using ξ_a and/or η_a.

We can conclude that the linearization only results in varying degrees of perturbation in the UAV's states from their nominal trajectory while the UGV's states deviate by a constant value equal to the initial perturbation. And as a result, the sensor readings relating to UAV's states contain major perturbations, while the relative sensor readings have a lower amplitude in their variation.

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 (25)] and correction [Equation (26)] 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 (Q_{KF}) for tuning and,
- initial values for $\delta\hat{x}^{+}(0)$ and $P^{+}(0)$.

The function outputs:

- the state estimates ($x_{estimate} = x_{nominal} + \delta\hat{x}^{+}$),
- measurement estimates $\left(y_{estimate}(k+1) = y_{nominal}(k+1) + \widetilde{H}_{k+1}\delta\hat{x}_{k+1}^{-}\right)$,
- estimation errors ($e_x = x_{truth} - x_{estimate}$ and $e_y = y_{truth} - y_{estimate}$) and,
- sequence of covariance matrices for state and measurement estimates (P^{+} and S^{-})

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

$$\begin{aligned} \delta\hat{x}_{k+1}^{-} &= \tilde{F}_k \delta\hat{x}_k^{+} \\ P_{k+1}^{-} &= \tilde{F}_k P_k^{+} \hat{F}_K^{T} + \hat{\Omega}_k Q_k \hat{\Omega}_k^{T} \end{aligned} \tag{25}$$

$$\begin{aligned} S_{k+1}^{-} &= \widetilde{H}_{k+1} P_{k+1}^{-} \widetilde{H}_{k+1}^{T} + R_{k+1} \\ K_{k+1} &= P_{k+1}^{-} \widetilde{H}_{k+1}^{T} (S_{k+1}^{-})^{-1} \\ \delta y_{k+1} &= y_{truth}(k+1) - y_{nominal}(k+1) \\ \delta\hat{x}_{k+1}^{+} &= \delta\hat{x}_{k+1}^{-} + K_{k+1}\left(\delta y_{k+1} - \widetilde{H}_{k+1}\delta\hat{x}_{k+1}^{-}\right) \\ P_{k+1}^{+} &= \left(I_n - K_{k+1}\widetilde{H}_{k+1}\right) P_{k+1}^{-} \end{aligned} \tag{26}$$

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 ($\tilde{F}_k, \widetilde{H}_k, \widetilde{\Omega}_k$) 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 ($N = 50$) 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 $x_{truth}(0)$ for the ground truth values for every Monte Carlo run from $dx^+(0) \sim \left(0, P^+(0)\right)$ and then $x_{truth}(0) = x_{nominal} + dx^+(0)$ 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:

$$\delta\hat{x}^+(0) = x_{truth}(0) - x_{nominal}(0)$$
$$P^+(0) = diag([1 \quad 1 \quad 0.025 \quad 1 \quad 1 \quad 0.025]) \tag{27}$$

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 ($P^+(0) \neq \infty$). And also, the angles cannot exceed 180^o or go below -180^o in radians.

In addition to this, in the LKF, the covariance matrix for the measurement noise (R) 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 zero and converges, and the 2σ 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 $\alpha = 0.01$. 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 $N = 50$ Monte Carlo runs, $n = 6$ states, $p = 5$ measured values, the chi-square confidence bounds are:

$$\{4.8133\} \leq \bar{\epsilon}_{x,k} \leq \{7.3369\}$$
$$\{3.9232\} \leq \bar{\epsilon}_{y,k} \leq \{6.2269\} \quad (28)$$

The $\bar{\epsilon}_{x,k}$ and $\bar{\epsilon}_{y,k}$ values are obtained by averaging the $\epsilon_{x,k}$ and $\epsilon_{y,k}$ values are all k steps, which are obtained from the following equations:

$$\epsilon_{x,k} = e_{x,k}^T (P^+)^{-1} e_{x,k}$$
$$\epsilon_{y,k} = e_{y,k}^T (S^-)^{-1} e_{y,k}$$

While tuning the LKF we vary our predicted process noise covariance matrix (Q_{KF}) until the two conditions in the beginning of this subsection are satisfied. We first start with $Q_{KF} = I_{6\times6}$ and run the truth model tests and check if the KF works (by looking at the state estimate errors and innovation) and the NEES and NIS tests are satisfied. The result showed that neither of the conditions were met, hence we moved to $Q_{KF} = 0.1I_{6\times6}$ then to $Q_{KF} = 0.01I_{6\times6}$ 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 ξ_g and η_g states as they appeared to be the states most sensitive to changes in Q_{KF}.

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

$$Q_{KF} = diag([0.0001 \quad 0.0001 \quad 0.01 \quad 0.1 \quad 0.1 \quad 0.01]) \quad (29)$$

Before we move forward, the term 'best satisfying Q_{KF}' 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 7 and Figure 8, 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 ξ_a and η_a as well as the relative distance measurement ρ_{ga} are large. Furthermore, the innovation errors corresponding to these sensed values do not average to zero and converge, as we can see that as time progresses, those errors oscillate.

This problem could not be averted regardless of the corresponding Q_{KF} elements selected (here we are referring to elements (4,4) and (6,6) that correspond to ξ_a and η_a), and hence the innovation

errors carried over to the state estimates. For instance, the state estimate errors for ξ_g and η_g (UGV positions) in Figure 9 also grows over time instead of averaging to zero and converging. Just observing the state estimates for ξ_g and η_g in Figure 10, we can see that they maintain a similar behavior to the truth model however as time progresses the state estimates experience significant deviation from the truth model.

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 9 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 ξ_a and η_a are large, i.e. approximately ranging between -100m and 100m, while the errors are miniscule in comparison (in terms of magnitude). 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 θ_g and θ_a are not significantly affected as they are more dependent on the control inputs.

Now while the state estimation errors for η_g, ξ_g, η_a and ξ_a do not converge and average to zero, their covariances do converge. In fact, all the covariances of the state estimates (P^+) and measurement estimates (S^-) converge.

We will now look at the NEES and NIS tests for the LKF. Referring to Figure 11, we can see that the NIS test is, to an extent, being satisfied, however as time progresses the NIS data exceed the confidence intervals. This overlaps with the previous observation of how the measurement estimates for ξ_a and η_a as well as the relative distance measurement ρ_{ga} 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 exceed the bounds.

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 found that for the time spent in fine tuning, the gains were indistinguishable. Hence an alternate estimator would be preferred.

Discussion on LKF for Estimating Nonlinear System

The tuning procedure was arduous. 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 ξ_a and η_a as well as the relative distance measurement ρ_{ga} 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 is reflected on the NEES and NIS tests because while the system passes the NIS tests in the beginning, the NEES test completely fails. Hence, we can conclude that the LKF fails and is statistically inconsistent in estimating the states. On a side note, 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.

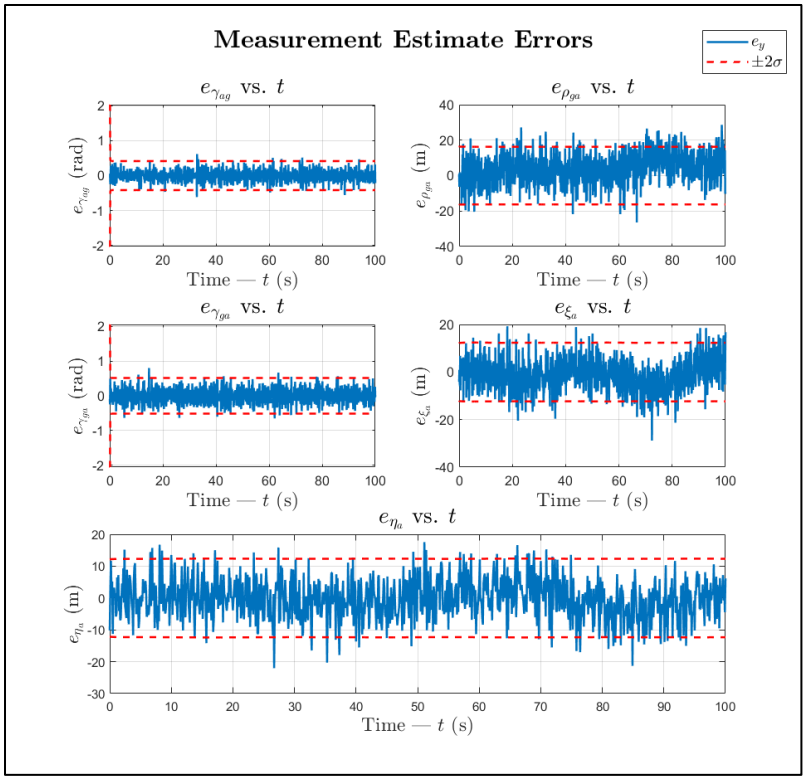

Figure 7 – Innovstion errors and 2σ bounds of sample LKF Monte Carlo run with Equations (27)(28)(29) applied

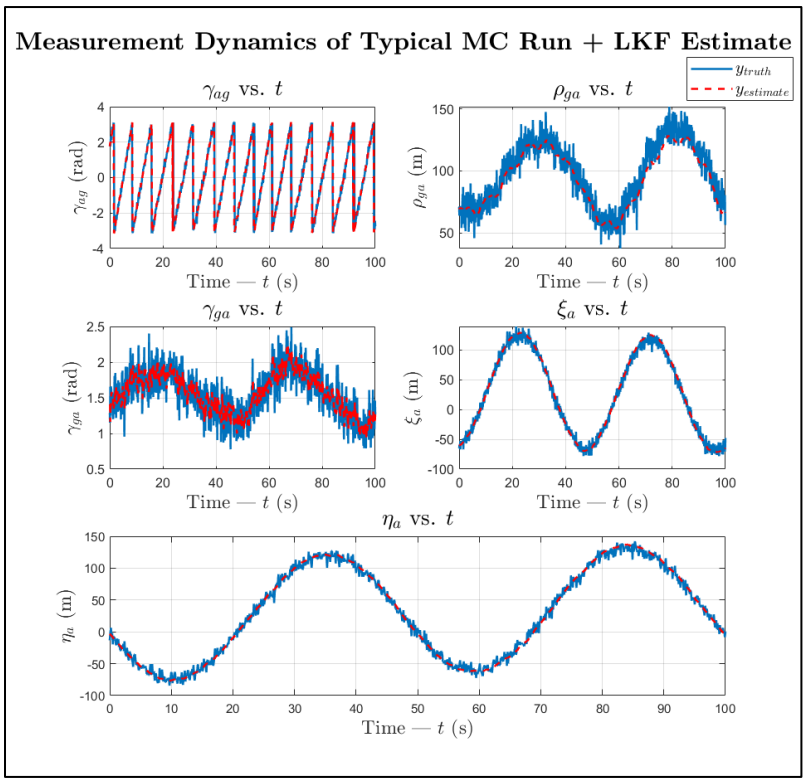

Figure 8 – Measurement estimates and ground truth of sample LKF Monte Carlo run with Equations (27)(28)(29) applied

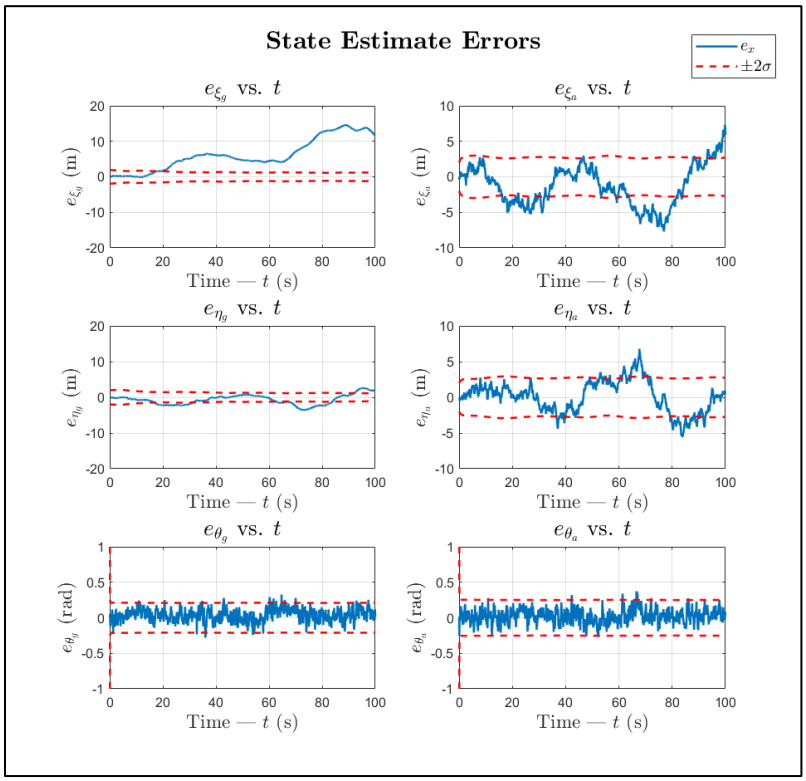

Figure 9 – State estimate errors and 2σ bounds of sample LKF Monte Carlo run with Equations (27)(28)(29) applied

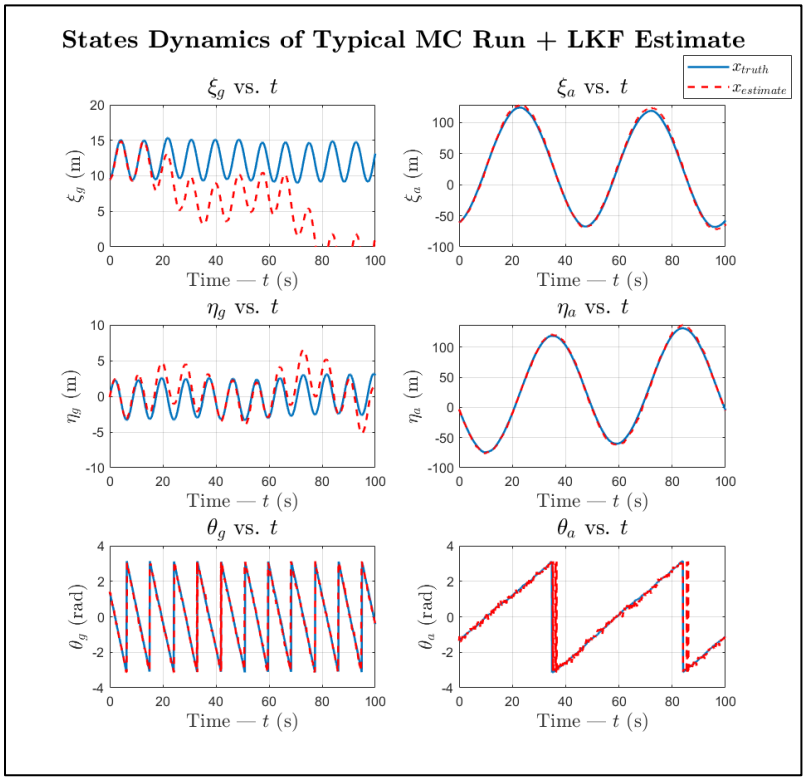

Figure 10 – State estimates and state ground truth of sample LKF Monte Carlo run with Equations (27)(28)(29) applied

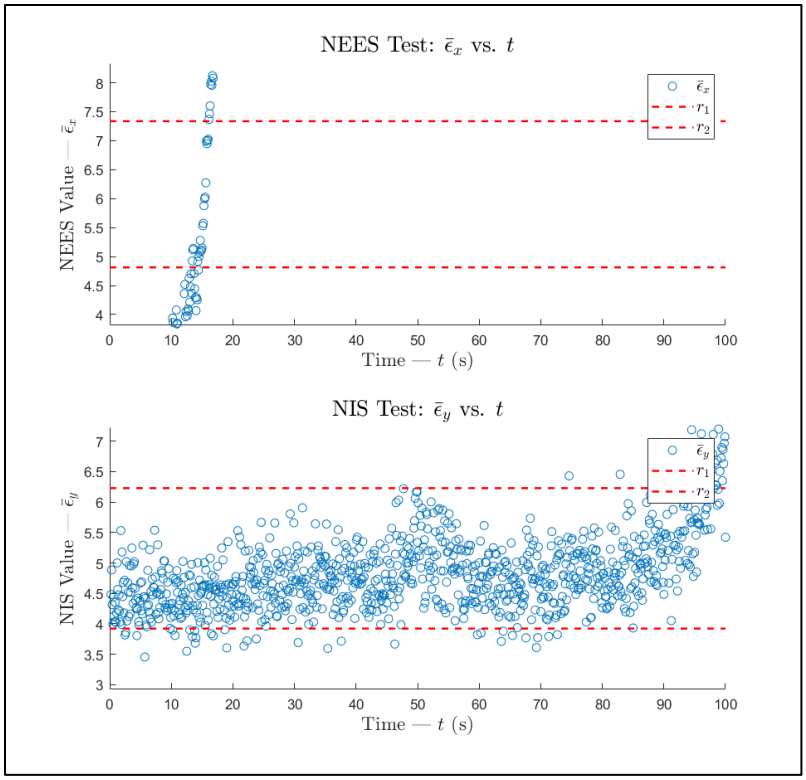

Figure 11 – NEES and NIS chi-square test for $N = 50$ LKF Monte Carlo runs with Equations (27)(28)(29) applied

Question #5

Extended Kalman Filter (EKF)

The EKF starts off at time-step zero (0) with an estimate of the total state and covariance. At each time-step k, during the prediction step the EKF uses the non-linear equations to calculate the estimated state which in turn is used to approximate the predicted covariance matrix.

$$x_{k+1}^{-} = f[\hat{x}_k^{+}, u_k, w_k = 0]$$

$$P_{k+1}^{-} = \tilde{F}_k P_k^{+} \tilde{F}_k^{T} + \tilde{\Omega}_k Q_k \tilde{\Omega}_k^{T},$$

where $\tilde{F}_k\big|_{\hat{x}_k^{+}, u_k, w_k=0} \approx I + \Delta T \cdot \tilde{A}\big|_{(\hat{x}_k^{+}, u_k(t_k), w_k(t_k)=0)}$ and

$$\tilde{\Omega}_k \approx \Delta T \cdot \Gamma(t)|_{(t=t_k)}$$

During the correction step the EKF calculates the innovation vector $(y_{k+1} - \hat{y}_{k+1}^{-})$ using the measurement estimate derived from the non-linear measurement equation.

$$\hat{y}_{k+1}^{-} = h[\hat{x}_{k+1}^{-}, v_{k+1} = 0]$$

$$\tilde{e}_{y_{k+1}} = y_{k+1} - \hat{y}_{k+1}^{-}$$

The filter then generates a new DT System (H) matrix, linearized about the new state estimate, to calculate the Kalman gain, update the total state estimate and update the covariance matrix.

$$\tilde{K}_{k+1} = P_{k+1}^{-} \tilde{H}_{k+1}^{T} \left[\tilde{H}_{k+1} P_{k+1}^{-} \tilde{H}_{k+1}^{T} + R_{k+1}\right]^{-1}$$

$$x_{k+1}^{+} = x_{k+1}^{-} + \tilde{K}_{k+1} \tilde{e}_{y_{k+1}}$$

$$P_{k+1}^{+} = \left(I - \tilde{K}_{k+1} \tilde{H}_{k+1}\right) P_{k+1}^{-},$$

where $\tilde{H}_{k+1} = \frac{\delta h}{\delta x}\Big|_{\hat{x}_{k+1}^{-}}$

MATLAB code for implementation and tuning of the EKF can be found in Appendix E.

Tuning Approach

Tuning the filter involved generating randomized ground truth data for Monte Carlo simulations and calculating the mean NEES and NIS scores at each time step. The EKF's process noise matrix Q was then tweaked until the NEES and NIS consistency tests were within the desired confidence bounds (alpha = ±0.01). The list below captures guidelines for and lessons learned during tuning.

Tuning Guidelines / Lessons Learned

- Use State Error plots to tune individual Q diagonal gains. If more than 3% of signal exceeds 2-sigma bounds, increase Q gain (filter is over-confident on state). If less than 3% the filter is too pessimistic, decrease Q.
- If periodicity of error signal is slow need to increase time duration of test or increase number of Monte Carlo runs, else testing might not be capturing long-term behavior of filter. The time duration was set to 100 seconds so multiple UAV orbits were included in a run and because the UGV error signal has low-frequency content (likely related to UAV orbit).
- The initial position should be randomized using the initial covariance matrix (x0 + randn(0,P0) to prevent odd NEES/NIS behavior at the start. Use the error bounds to estimate the average covariance and use this as an initial guess. The initial variance of [25 m 25 m 0.25 rad 156 m 156 m 0.25 rad]^T proved to be adequate.
- If the NEES scores are above the upper bound the Q values are too small (over-confident) and if the Q values are too small the NEES scores will be below the lower bound.
- The alpha value—desired filter consistency value—is dependent on the use cause but 5-10% is generally acceptable.
- The chi-square scores seemed to be most sensitive to the UGV and UAV angle noise and least sensitive to the UAV north/east position noise.
- Since the vehicle position/velocity are a function of the heading angle, there's some coupling between these states so their off-diagonals were made non-zero. This proved to be of little benefit though.
- Creating a plot with state error plots for all runs was informative for making tuning decisions.
- The true noise measurement covariance was used as is under the assumption that these sensor error values could be extracted from the sensor specification.

Typical Simulation Plots

The section shows plots from a typical simulation run. Figure 12 captures an example of the randomly generated UGV and UAV state data.

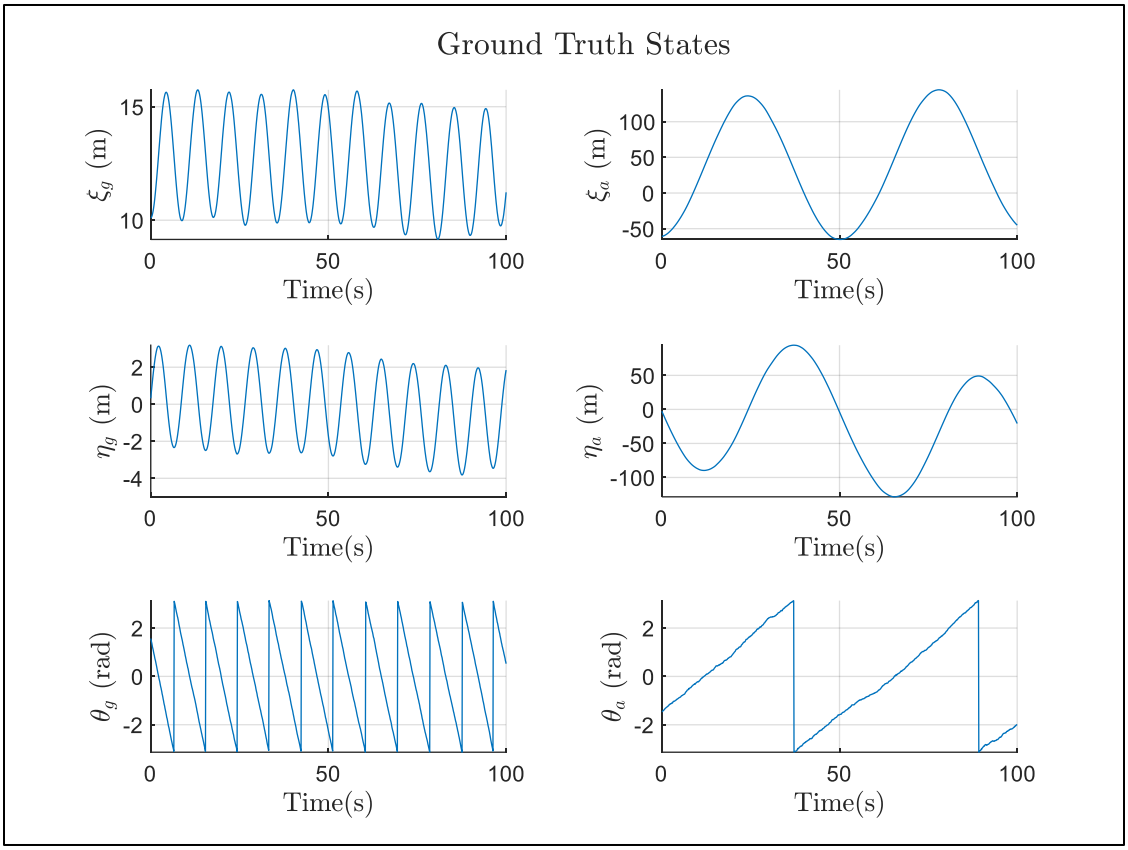

Figure 12 - Simulation Run - Ground Truth States

Figure 13 shows an example of the randomly generated noisy measurement data.

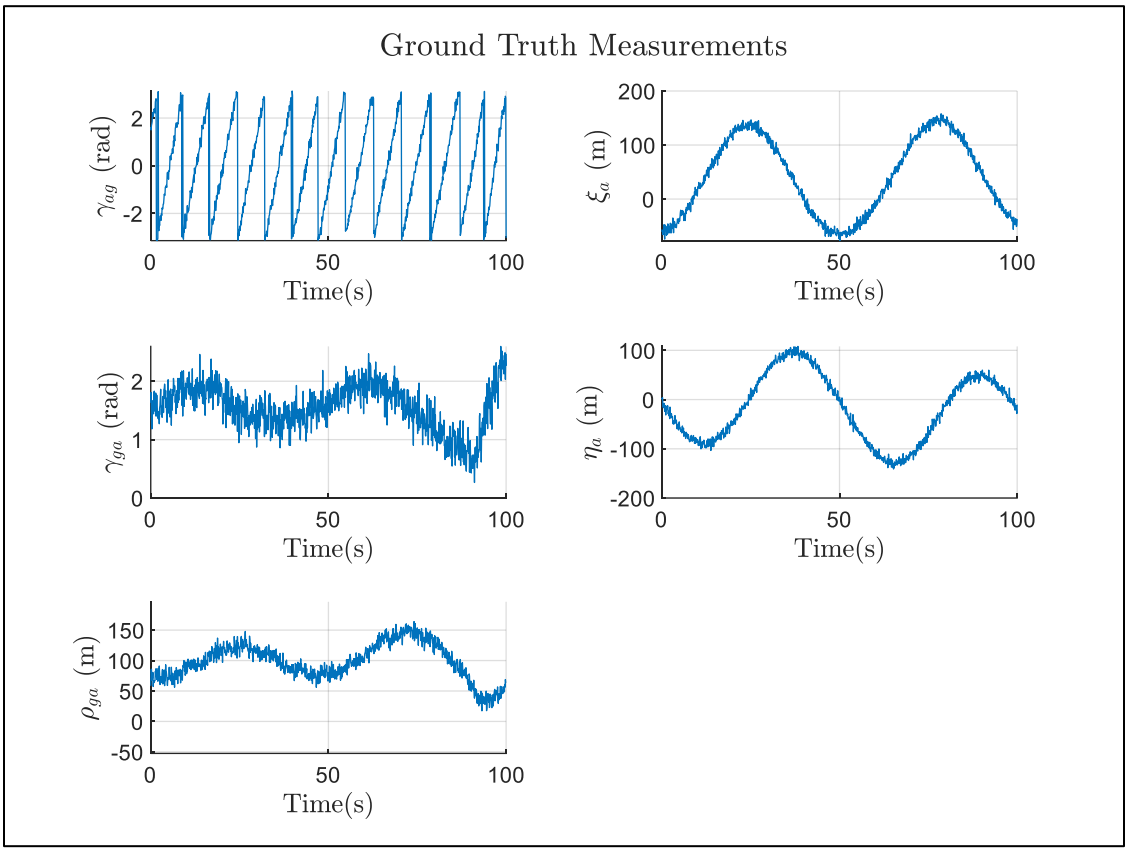

Figure 13 - Simulation Run - Simulated Measurement Data

Figure 14 is an example of the state estimate errors; the ±2σ bounds calculated using the covariance matrix are shown as red dotted lines.

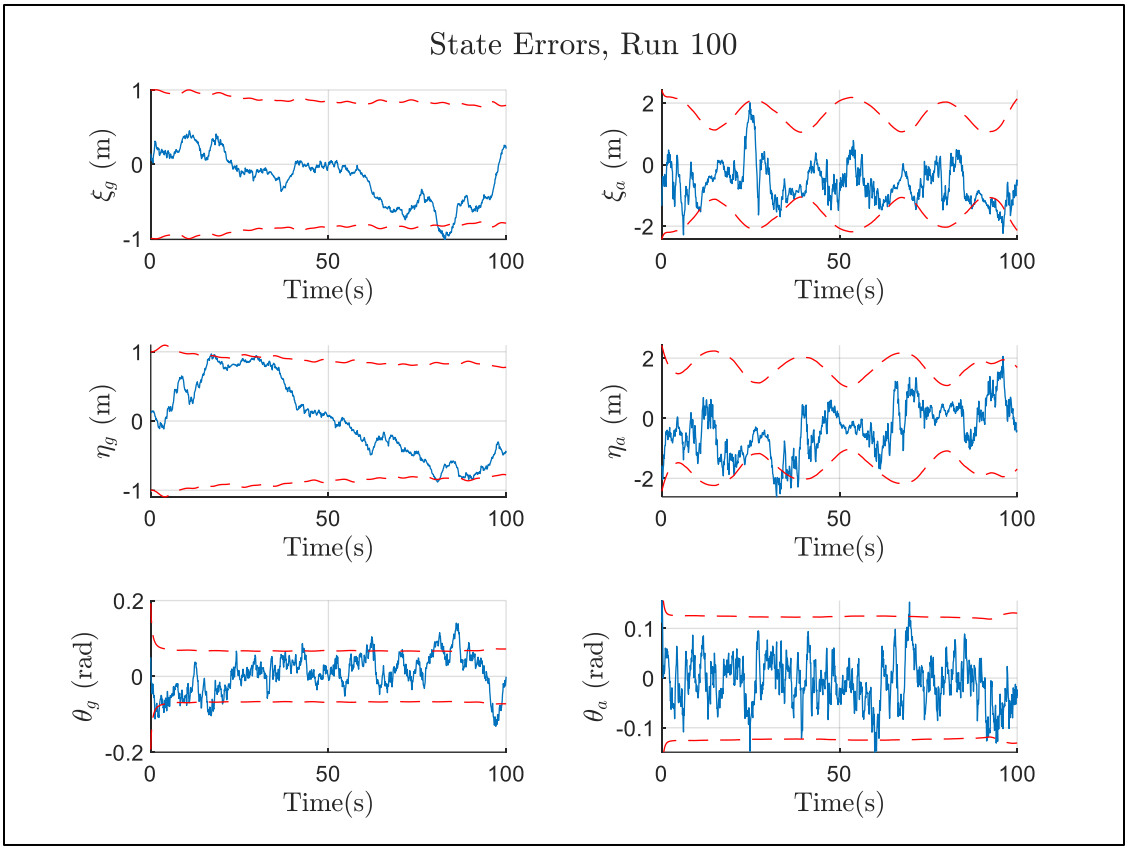

Figure 14 – Estimated State Errors

An example of the typical measurement error is shown in Figure 15.

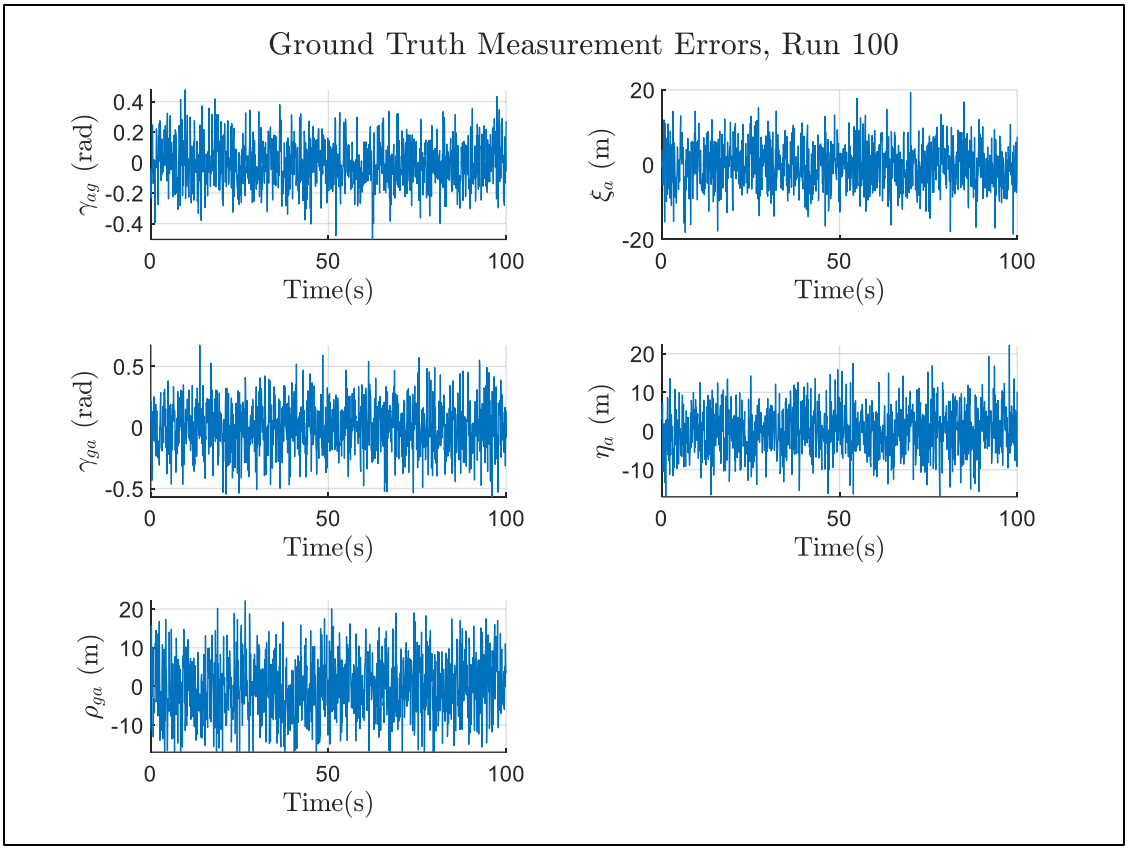

Figure 15 - Estimated Measurement Errors

Chi-Square Test Results

This section shows the final NEES and NIS test results (referring to Figure 16 and Figure 17) for 100 Monte Carlo simulation runs and an alpha value of 5%. Our filter passes the NEES test criteria as the mean score is approximately equal to the number of states (n = 6) and approximately 5% of samples are beyond the limits. The NIS test was easier to pass, with a mean equal to the number of measurements (p=5) and approximately 5% of samples beyond the limits.

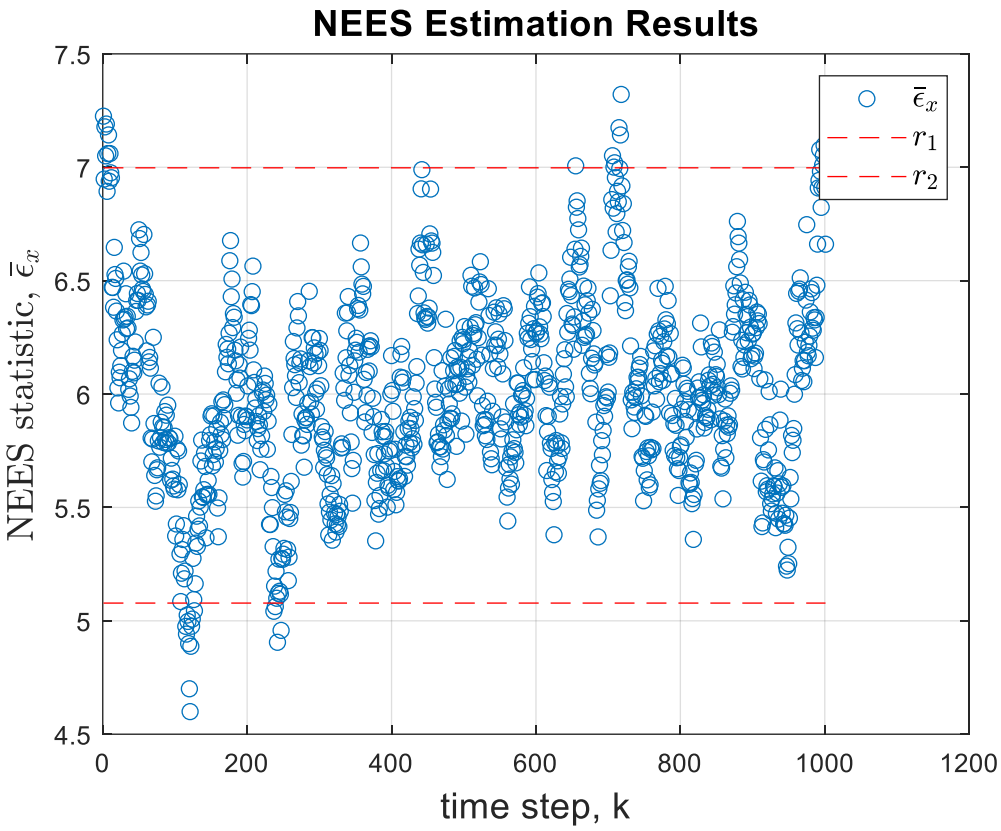

Figure 16 - Final NEES results

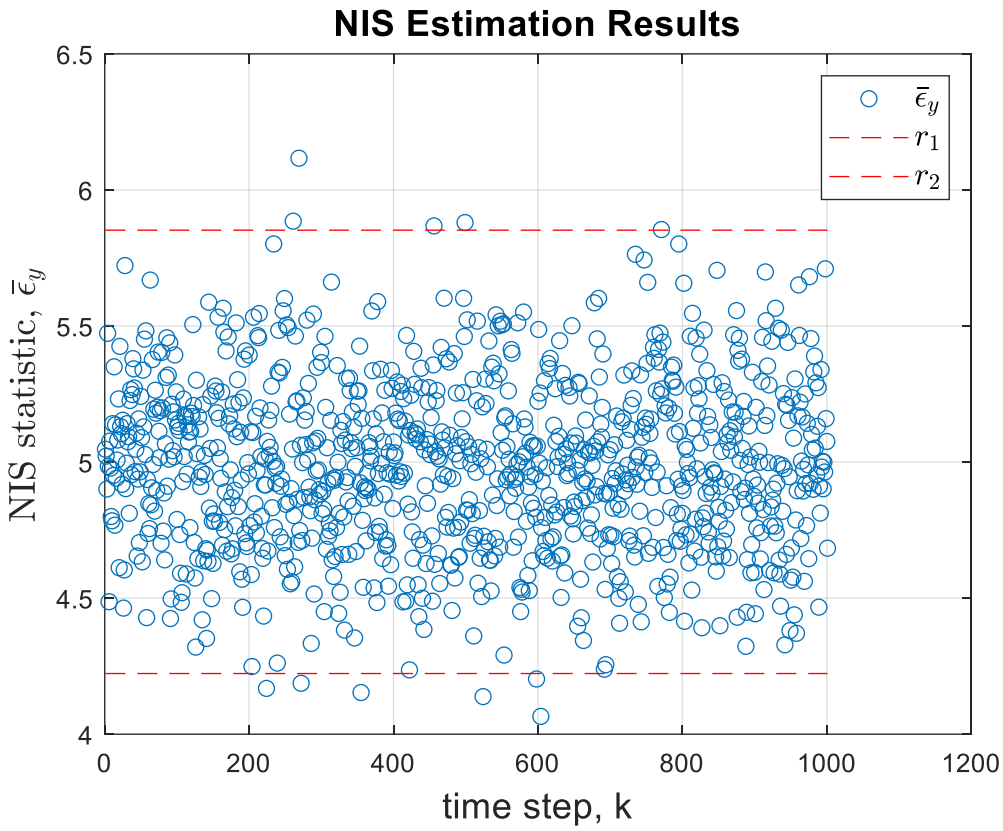

Figure 17 - Final NIS Results

Question #6

Filter Comparison – LKF vs EKF

For this series of measurements and initial conditions the filters had no significant difference performance in predicting the measurements. This is known be referring to the Figure 18 which shows the NIS test results. We can see that since we only have the ground truth measurement data to work with, the LKF and EKF perform comparably and effectively within the given time limit of the operation to estimate the sensor datas. This is known because the LKF and the EKF both pass the NIS tests within the time limit.

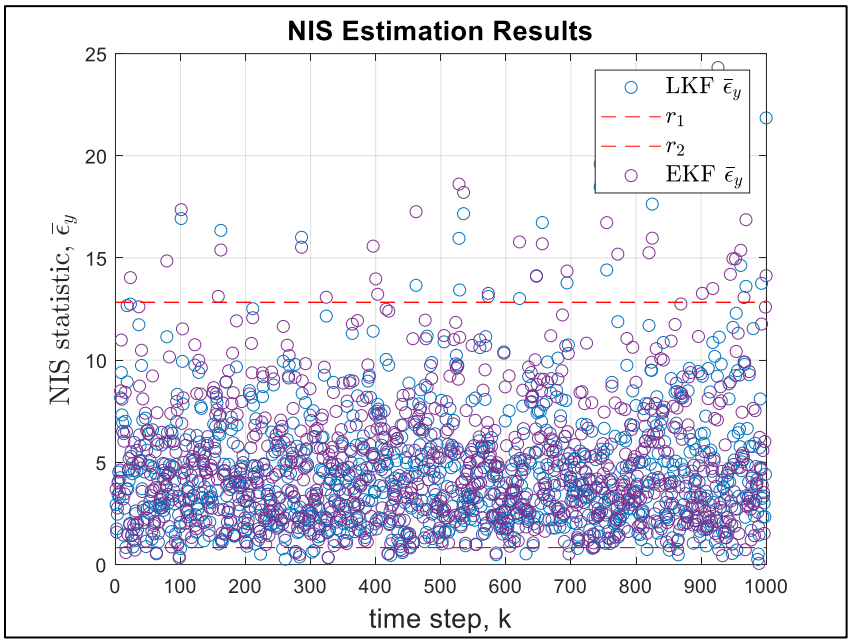

Figure 18 – NIS test results for LKF and EKF given ground truth measurement data

Next, we compare the state trajectory estimates obtained by running the LKF and EKF estimators. For this, we keep the finalized tuned parameters the same as in Question 4 and Question 5. Since we do not have ground truth data for the states, we cannot compare the state estimate errors and only the state estimate error covariances. The basis of comparison here will be how well the state estimates are bounded within the 2σ bounds: are the state estimates smooth, are the bounds converging etc.

Referring to Figure 19 and Figure 20 we see that EKF is superior in predicting the states compared to the LKF because the 2σ bounds converge faster, and if we draw our attention to the graphs for θ_a we see that the LKF's the estimate is not stable while the EKF's is. By this, we mean that at time 90s, the state estimate for θ_a with the LKF is an outlier and forms a discontinuity with the

current state trajectory. This results in the sudden ‘jump’ in the LKF’s graph while this is not present in the EKF’s graph.

If we go back to the conclusions derived in the previous sections (Question 4 and Question 5) we know that the EKF is superior to the LKF for this nonlinear system. We found that although the LKF and EKF both provide an accurate estimate for the measurements, the EKF performs far better in predicting the system’s states because the state estimate errors were averaging and converging to zero, and the covariance bounds for the estimates were also stable and converging. This was not the case with the LKF as the estimate errors for the position states for the UGV and the UAV did not average to zero and deviated as time progressed.

Returning to the current problem, since we do not have a ground truth state trajectory to compare our results, with the limited information for comparison, we can conclude that the EKF performs better than the LKF for this cooperative localization nonlinear problem.

Refer to Appendix F for codes adhering to this question.

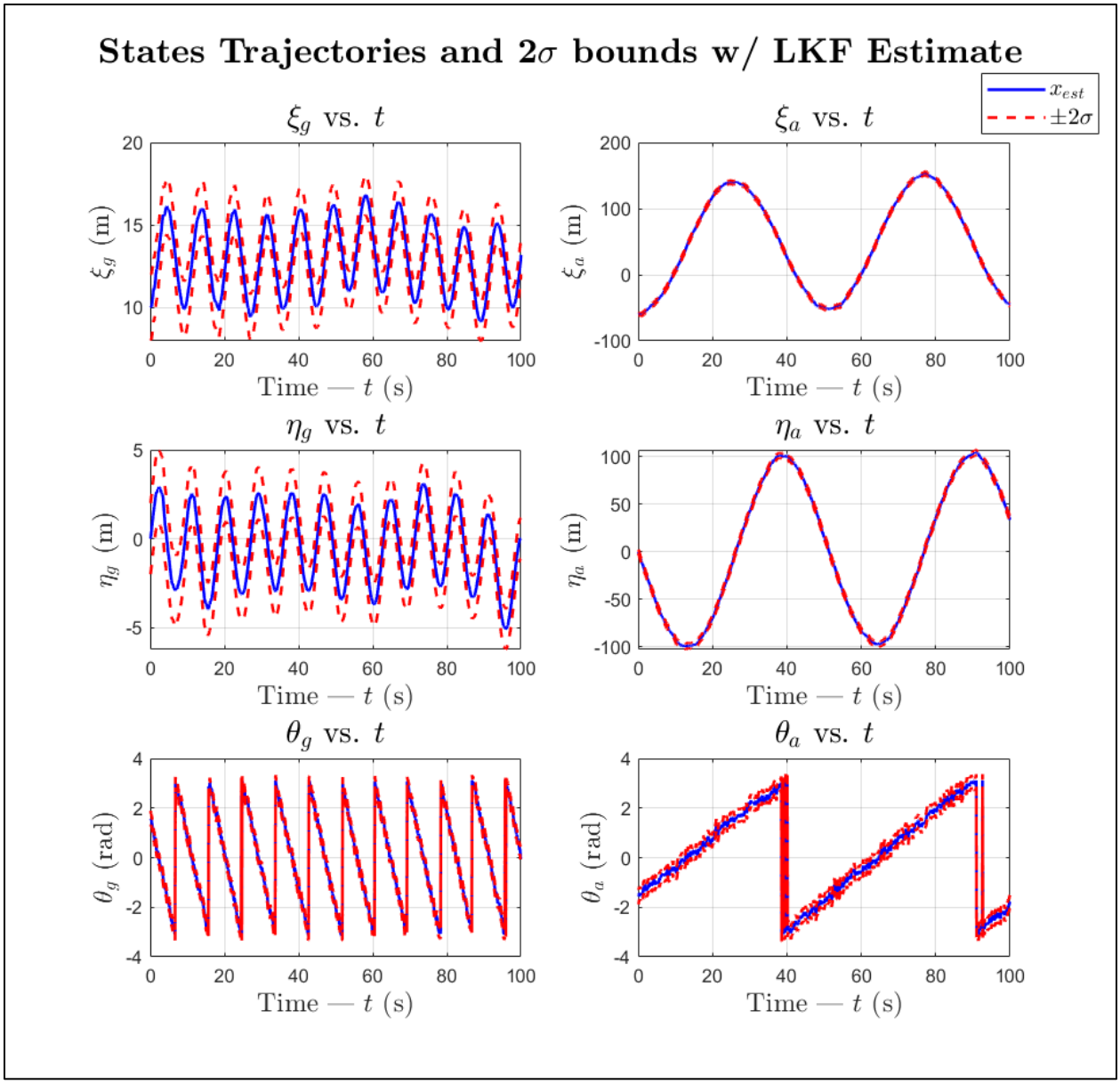

Figure 19 – Estimated state trajectories and 2σ bounds using the LKF estimator

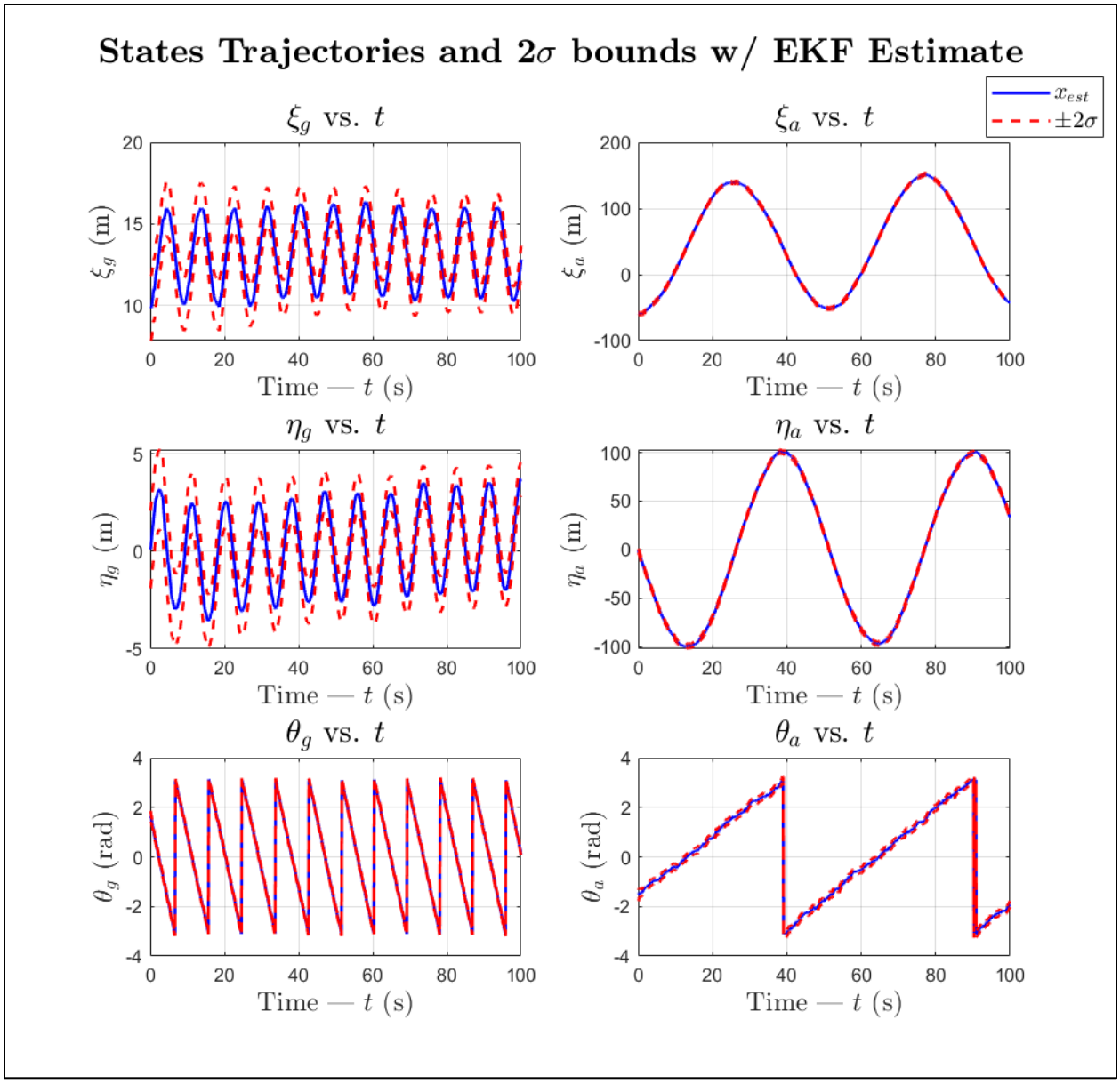

Figure 20 – Estimated state trajectories and 2σ bounds using the EKF estimator

AQ #13

Haiku

5	With ground truth data
7	Kalman Filters Estimate
5	Noisy system states

APPENDIX

Appendix A – Supporting derivation for Jacobians

$\frac{\partial h_1}{dx_1} = \frac{\partial}{dx_1}\left(\tan^{-1}\left(\frac{x_5 - x_2}{x_4 - x_1}\right) - x_3 + \tilde{v}_{\theta,ga}\right) = \frac{\partial}{dx_1}\left(\tan^{-1}\left((x_5 - x_2)(-x_1 + x_4)^{-1}\right)\right)$	(A.1)
$\frac{\partial h_1}{dx_2} = \frac{\partial}{dx_2}\left(\tan^{-1}\left(\frac{x_5 - x_2}{x_4 - x_1}\right) - x_3 + \tilde{v}_{\theta,ga}\right) = \frac{\partial}{dx_2}\left(\tan^{-1}\left(\frac{-1}{x_4 - x_1}x_2 + \frac{x_5}{x_4 - x_1}\right)\right)$	(A.2)
$\frac{\partial h_1}{dx_4} = \frac{\partial}{dx_4}\left(\tan^{-1}\left(\frac{x_5 - x_2}{x_4 - x_1}\right) - x_3 + \tilde{v}_{\theta,ga}\right) = \frac{\partial}{dx_4}\left(\tan^{-1}\left((x_5 - x_2)(x_4 - x_1)^{-1}\right)\right)$	(A.3)
$\frac{\partial h_1}{dx_5} = \frac{\partial}{dx_5}\left(\tan^{-1}\left(\frac{x_5 - x_2}{x_4 - x_1}\right) - x_3 + \tilde{v}_{\theta,ga}\right) = \frac{\partial}{dx_5}\left(\tan^{-1}\left(\frac{1}{x_4 - x_1}x_5 + \frac{-x_2}{x_4 - x_1}\right)\right)$	(A.4)
$\frac{\partial h_2}{dx_1} = \frac{\partial}{dx_1}\left(\sqrt{(x_1 - x_4)^2 + (x_2 - x_5)^2} + \tilde{v}_r\right) = \frac{\partial}{dx_1}\left(\sqrt{(x_1 - x_4)^2 + (x_2 - x_5)^2}\right)$	(A.5)
$\frac{\partial h_2}{dx_1} = \frac{\partial}{dx_2}\left(\sqrt{(x_1 - x_4)^2 + (x_2 - x_5)^2} + \tilde{v}_r\right) = \frac{\partial}{dx_2}\left(\sqrt{(x_1 - x_4)^2 + (x_2 - x_5)^2}\right)$	(A.6)
$\frac{\partial h_2}{dx_1} = \frac{\partial}{dx_4}\left(\sqrt{(x_1 - x_4)^2 + (x_2 - x_5)^2} + \tilde{v}_r\right) = \frac{\partial}{dx_4}\left(\sqrt{(-x_4 + x_1)^2 + (x_2 - x_5)^2}\right)$	(A.7)
$\frac{\partial h_2}{dx_1} = \frac{\partial}{dx_5}\left(\sqrt{(x_1 - x_4)^2 + (x_2 - x_5)^2} + \tilde{v}_r\right) = \frac{\partial}{dx_5}\left(\sqrt{(x_1 - x_4)^2 + (-x_5 + x_2)^2}\right)$	(A.8)
$\frac{\partial h_3}{dx_1} = \frac{\partial}{dx_1}\left(\tan^{-1}\left(\frac{x_2 - x_5}{x_1 - x_4}\right) - x_6 + \tilde{v}_{\theta,ag}\right) = \frac{\partial}{dx_1}\left(\tan^{-1}\left((x_2 - x_5)(x_1 - x_4)^{-1}\right)\right)$	(A.9)
$\frac{\partial h_3}{dx_2} = \frac{\partial}{dx_2}\left(\tan^{-1}\left(\frac{x_2 - x_5}{x_1 - x_4}\right) - x_6 + \tilde{v}_{\theta,ag}\right) = \frac{\partial}{dx_2}\left(\tan^{-1}\left(\frac{1}{x_1 - x_4}x_2 + \frac{-x_5}{x_1 - x_4}\right)\right)$	(A.10)
$\frac{\partial h_3}{dx_4} = \frac{\partial}{dx_4}\left(\tan^{-1}\left(\frac{x_2 - x_5}{x_1 - x_4}\right) - x_6 + \tilde{v}_{\theta,ag}\right) = \frac{\partial}{dx_4}\left(\tan^{-1}\left((x_2 - x_5)(-x_4 + x_1)^{-1}\right)\right)$	(A.11)
$\frac{\partial h_3}{dx_5} = \frac{\partial}{dx_5}\left(\tan^{-1}\left(\frac{x_2 - x_5}{x_1 - x_4}\right) - x_6 + \tilde{v}_{\theta,ag}\right) = \frac{\partial}{dx_5}\left(\tan^{-1}\left(\frac{-1}{x_1 - x_4}x_5 + \frac{x_2}{x_1 - x_4}\right)\right)$	(A.12)

All solutions for equations A.2, A.4, A.10, A.12 all are of the generic form:

$$\frac{\partial}{dx_a}\left(\tan^{-1}(ax_a + b)\right) = \frac{1}{1 + (ax_a + b)^2}\frac{\partial}{dx_a}(ax_a + b) = \frac{a}{1 + (ax_a + b)^2}$$

All solutions for equations A.1, A.3, A.9, A.11 all are of the generic form:

$$\frac{\partial}{dx_a}\left(\tan^{-1}(c(ax_a + b)^{-1})\right) = \frac{1}{1 + (c(ax_a + b)^{-1})^2}\frac{\partial}{dx_a}(c(ax_a + b)^{-1}) =$$

$$\frac{-ac}{1+(c(ax_a+b)^{-1})^2}(ax_a+b)^{-2} = \frac{\frac{-ca}{(ax_a+b)^2}}{1+\left(\frac{c}{ax_a+b}\right)^2} = \frac{\frac{-ca}{(ax_a+b)^2}}{\left(\frac{(ax_a+b)^2+c}{ax_a+b}\right)^2}$$

$$= \frac{-ca}{(ax_a+b)^2+c^2}$$

All solutions for equations A5 through A8 all are of the generic form:

$$\frac{\partial}{dx}(c+(ax+b)^2)^{1/2} = \frac{(c+(ax+b)^2)^{-1/2}}{2}\frac{\partial}{dx}((ax+b)^2)$$

$$= \frac{(c+(ax+b)^2)^{-1/2}}{2}(2a)(ax+b) = \frac{a(ax+b)}{\sqrt{c+(ax+b)^2}}$$

The rest of the math was done by hand, substituting the parameters (a,b,c) and simplifying:

$$\frac{a}{1+(ax_a+b)^2}$$

A.2 $a = \frac{-1}{x_4-x_1}$ $b = \frac{x_5}{x_4-x_1}$ $x_a = x_2$

$$\frac{\frac{-1}{x_4-x_1}}{1+\left[\left(\frac{-1}{x_4-x_1}\right)x_2+\frac{x_5}{x_4-x_1}\right]^2} = \frac{\frac{-1}{x_4-x_1}}{1+\left(\frac{x_5-x_2}{x_4-x_1}\right)^2} = \frac{\frac{-1}{(x_4-x_1)}}{\frac{(x_4-x_1)^2+(x_5-x_2)^2}{(x_4-x_1)^2}}$$

$$= \frac{-(x_4-x_1)}{(x_4-x_1)^2+(x_5-x)^2}$$

A.4 $a = \frac{1}{x_4-x_1}$ $b = \frac{-x_2}{x_4-x_1}$ $x_a = x_5$

$$\frac{\frac{1}{x_4-x_1}}{1+\left[\frac{1}{(x_4-x_1)}x_5+\left(\frac{-x_2}{x_4-x_1}\right)\right]^2} = \frac{\frac{1}{x_4-x_1}}{1+\frac{(x_5-x_2)^2}{(x_4-x_1)^2}} = \frac{\frac{1}{(x_4-x_1)}}{\frac{(x_4-x_1)^2+(x_5-x_2)^2}{(x_4-x_1)^2}} = \frac{(x_4-x_1)}{(x_4-x_1)^2+(x_5-x_2)^2}$$

A.10 $a = \frac{1}{x_1-x_4}$ $b = \frac{-x_5}{x_1-x_4}$ $x = x_2$

$$\frac{\frac{1}{x_1-x_4}}{1+\left[\frac{1}{(x_1-x_4)}x_2+\frac{-x_5}{(x_1-x_4)}\right]^2} = \frac{\frac{1}{x_1-x_4}}{1+\frac{(x_2-x_5)^2}{(x_1-x_4)^2}} = \frac{\frac{1}{x_1-x_4}}{\frac{(x_1-x_4)^2+(x_2-x_5)^2}{(x_1-x_4)^2}} = \frac{x_1-x_4}{(x_1-x_4)^2+(x_2-x_5)^2}$$

A.12 $a = \left(\frac{-1}{x_1-x_4}\right)$ $b = \frac{x_2}{x_1-x_4}$ $x = x_5$

$$\frac{\frac{-1}{x_1-x_4}}{1+\left[\left(\frac{-1}{x_1-x_4}\right)x_5+\left(\frac{x_2}{x_1-x_4}\right)\right]^2} = \frac{\frac{-1}{x_1-x_4}}{1+\frac{(x_2-x_5)^2}{(x_1-x_4)^2}} = \frac{\frac{-1}{x_1-x_4}}{\frac{(x_1-x_4)^2+(x_2-x_5)^2}{(x_1-x_4)^2}} = \frac{-(x_1-x_4)}{(x_1-x_4)^2+(x_2-x_5)^2}$$

$$\frac{-ca}{(ax+b)^2 + c^2}$$

#1] $a = -1 \quad b = x_4 \quad c = (x_5 - x_2) \quad x = x_1$

$$\frac{+(y_5 - y_2)(+1)}{(-x_1 + x_4)^2 + (x_5 - x_2)^2} = \frac{y_5 - x_2}{(x_4 - x_1)^2 + (x_5 - x_2)^2}$$

#3] $a = 1 \quad b = -x_1 \quad c = y_5 - y_2 \quad x = x_4$

$$\frac{(-1)(x_5 - y_2)(1)}{(x_4 + -x_1)^2 + (y_5 - y_2)^2} = \frac{x_2 - x_5}{(x_4 - x_1)^2 + (x_5 - x_2)}$$

A.9 / $a = 1 \quad b = -x_4 \quad c = y_2 - x_5 \quad x = x_1$

$$\frac{(-1)(x_2 - x_5)(1)}{(x_1 - x_4)^2 + (y_2 - x_5)^2} = \frac{y_5 - x_2}{(x_1 - x_4)^2 + (x_2 - x_5)^2}$$

A.11 $a = -1 \quad b = x_1 \quad c = (x_2 - x_5) \quad x = x_4$

$$\frac{(-1)(y_2 - x_5)(-1)}{(-1x_4 + x_1)^2 + (y_2 - x_5)^2} = \frac{(x_2 - x_5)}{(x_1 - x_4)^2 + (x_2 - x_5)^2}$$

$$\frac{a(ax+b)}{\sqrt{c+(ax+b)^2}}$$

A5 | $a = 1 \quad b = -x_4 \quad c = (x_2 - x_5)^2 \quad x = y_1$

$$\frac{(1)(x_1 + -x_4)}{\sqrt{(x_2 - x_5)^2 + (x_1 - x_4)^2}}$$

A6 | $a = 1 \quad b = -x_5 \quad c = (x_1 - x_4)^2 \quad x = x_2$

$$\frac{(1)(x_2 + -x_5)}{\sqrt{(x_1 - x_4)^2 + (x_2 - x_5)^2}}$$

A7 | $a = -1 \quad b = x_1 \quad c = (x_2 - x_5)^2 \quad x = x_4$

$$\frac{(-1)((-1)x_4 + x_1)}{\sqrt{(x_2 - x_5)^2 + (y_4 - y_1)^2}} = \frac{y_4 - x_1}{\sqrt{(x_2 - x_5)^2 + (x_4 - x_1)^2}}$$

A8 | $a = -1 \quad b = x_2 \quad c = (x_1 - x_4)^2 \quad x = x_5$

$$\frac{(-1)((-1)x_5 + x_2)}{\sqrt{(x_1 - x_4)^2 + (x_5 - x_2)^2}} = \frac{x_5 - x_2}{\sqrt{(x_1 - x_4)^2 + (x_5 - x_2)^2}}$$

Appendix C – Codes accompanying Question 3

```
function xdot = NL_DynModel(t,x,u,Pnoise)
%NL_DynModel
%   input: t - time model; x - state vector; u - control input vector;
%          Pnoise - process noice vector
%   output: ___
%   Function input for ode45 for NL dynamics model

% u = [v_g, phi_g, v_a, w_a]';
% x = [xi_g eta_g theta_g xi_a eta_a theta_a]';
% ~w = [w_x,g w_y,g w_w,g w_x,a w_y,a w_w,a]';
L = 0.5;

xdot = [u(1)*cos(x(3)) + Pnoise(1);
        u(1)*sin(x(3)) + Pnoise(2);
        (u(1)/L)*(tan(u(2))) + Pnoise(3);
        u(3)*cos(x(6)) + Pnoise(4);
        u(3)*sin(x(6)) + Pnoise(5);
        u(4) + Pnoise(6)];
end
```

```
function [y] = NL_MeasModel(x,Mnoise)
%NL_MeasModel
%   input: x - state vector; Mnoise - measurement noice vector
%   output: y - sensor readings;
%   Uses NL state inputs and measurement noise vector to get sensor
%   readings

% y = [gamma_ag rho_ga gamma_ga xi_a eta_a]';
% x = [xi_g eta_g theta_g xi_a eta_a theta_a]';
% x = [1    2     3       4    5     6      ]';

y = [atan2(x(5)-x(2),x(4)-x(1)) - x(3);
     sqrt((x(1)-x(4))^2 + (x(2)-x(5))^2);
     atan2(-x(5)+x(2),-x(4)+x(1)) - x(6);
     x(4);
     x(5)];

y = y + Mnoise;

end
```

```
function [A_t,B_t,C_t] = Linearize(x,u)
%Linearize
%   input: x - nominal state vector; u - nominal control input;
%   output: A_t - A tilde Matrix; B_t - B tilde Matrix; C_t - C tilde
%           Matrix
%   Obtain the CT linearized state perturbation matrices

% u = [v_g, phi_g, v_a, w_a]';
% x = [xi_g eta_g theta_g xi_a eta_a theta_a]';

L = 0.5;

A_t = [0 0 -u(1)*sin(x(3)) 0 0 0;
       0 0 u(1)*cos(x(3))  0 0 0;
       0 0 0               0 0 0;
       0 0 0               0 0 -u(3)*sin(x(6));
       0 0 0               0 0 u(3)*cos(x(6));
       0 0 0               0 0 0];


B_t = [cos(x(3))    0                    0         0;
```

```
        sin(x(3))    0                       0          0;
        tan(u(2))/L  (u(1)/L)*sec(u(2))^2 0          0;
        0            0                       cos(x(6))  0;
        0            0                       sin(x(6))  0;
        0            0                       0          1];

% x = [xi_g eta_g theta_g xi_a eta_a theta_a]';

C11 = (x(5)-x(2))/((x(5)-x(2))^2 + (x(4)-x(1))^2);
C12 = -(x(4)-x(1))/((x(5)-x(2))^2 + (x(4)-x(1))^2);
C13 = -1;
C14 = -(x(5)-x(2))/((x(5)-x(2))^2 + (x(4)-x(1))^2);
C15 = (x(4)-x(1))/((x(5)-x(2))^2 + (x(4)-x(1))^2);
C21 = (x(1)-x(4))*((x(1)-x(4))^2 + (x(2)-x(5))^2)^-0.5;
C22 = (x(2)-x(5))*((x(1)-x(4))^2 + (x(2)-x(5))^2)^-0.5;
C24 = -(x(1)-x(4))*((x(1)-x(4))^2 + (x(2)-x(5))^2)^-0.5;
C25 = -(x(2)-x(5))*((x(1)-x(4))^2 + (x(2)-x(5))^2)^-0.5;
C31 = -(x(2)-x(5))/((x(2)-x(5))^2 + (x(1)-x(4))^2);
C32 = (x(1)-x(4))/((x(2)-x(5))^2 + (x(1)-x(4))^2);
C34 = (x(2)-x(5))/((x(2)-x(5))^2 + (x(1)-x(4))^2);
C35 = -(x(1)-x(4))/((x(2)-x(5))^2 + (x(1)-x(4))^2);
C36 = -1;
C44 = 1;
C55 = 1;

C_t = [C11 C12 C13 C14 C15 0;
       C21 C22 0   C24 C25 0;
       C31 C32 0   C34 C35 C36;
       0   0   0   C44 0   0;
       0   0   0   0   C55 0];
end
```

```
function [x_DTL, dx_DTL, y_DTL, dy_DTL, F, H, O] = DT_L_Model(t,Dt,x_NL,y_NL,x_pert,u_nom)
%DT_L_Model
%   input: t - time; Dt - time increment; x_NL - nominal state trajectory;
%          y_NL - nominal sensor readings; x_pert - initial state
%          perturbation; u_nom = nominal control input;
%   output: x_DTL = discrete time state; dx_DTL = discrete time state
%           perturbation; y_DTL = discrete time sensor; dy_DTL = discrete
%           time sensor perturbation
%   Function for linear DT model

% u = [v_g, phi_g, v_a, w_a]';
% x = [xi_g eta_g theta_g xi_a eta_a theta_a]';
% ~w = [w_x,g w_y,g w_w,g w_x,a w_y,a w_w,a]';

x_nominal = [x_NL(1,:); x_NL(2,:);
             wrapToPi(x_NL(3,:)); x_NL(4,:);
             x_NL(5,:); wrapToPi(x_NL(6,:))];
y_nominal = y_NL;

F = zeros(6,6,length(t));
H = zeros(5,6,length(t));
O = zeros(6,6,length(t));

% Evaluate F and H matrices with predef nominal state trajectories
for i=1:length(t)
    [A_t, ~, C_t] = Linearize(x_nominal(:,i),u_nom);
    F(:,:,i) = eye(6) + A_t*Dt;
    H(:,:,i) = C_t;
    O(:,:,i) = Dt*eye(6,6);
end
```

```
dx_DTL = zeros(6,length(t));
dy_DTL = zeros(5,length(t));
x_DTL = zeros(6,length(t));
y_DTL = zeros(5,length(t));
dx_DTL(:,1) = x_pert;
dy_DTL(:,1) = H(:,:,1)*x_pert;
x_DTL(:,1) = x_nominal(:,1)+dx_DTL(:,1);
y_DTL(:,1) = y_nominal(:,1)+dy_DTL(:,1);

for i=2:length(t)
    dx_DTL(:,i) = F(:,:,i)*dx_DTL(:,i-1);
    dy_DTL(:,i) = H(:,:,i)*dx_DTL(:,i);
    x_DTL(:,i) = x_nominal(:,i)+dx_DTL(:,i);
    y_DTL(:,i) = y_nominal(:,i)+dy_DTL(:,i);
end
end
```

Appendix D – Codes accompanying Question 4

```
function                          [x_est,y_est,dx,P,S,e_x,e_y]                    =
LKF(dx_init,P_init,x_nom,y_nom,x,y,Fk,Hk,Ok,Q,R, wrapOn)

n = size(Fk(:,:,1),2);          % No. of States
p = size(Hk(:,:,1),1);          % No. of Measurements
steps = length(x_nom(1,:))-1;   % No. of k steps
dx = zeros(n,steps+1);
P = zeros(n,n,steps+1);
S = zeros(p,p,steps+1);
K = zeros(n,p,steps+1);

dx(:,1) = dx_init;              % initial dx
dy = y - y_nom;                 % ground truth dx
if wrapOn == true
    dx = WrapX(dx);
    dy = WrapY(dy);
end

P(:,:,1) = P_init;              % inital P
I = eye(n);

y_est = zeros(5,steps+1);
y_est(:,1) = y_nom(:,1) + Hk(:,:,1)*(dx(:,1));

e_y = zeros(p,steps+1);
e_y(:,1) =  y(:,1) - y_est(:,1);% error in measurement estimates
if wrapOn == true
    e_y = WrapY(e_y);
end

S(:,:,1) = Hk(:,:,1)*P(:,:,1)*Hk(:,:,1)' + R;
K(:,:,1) = P(:,:,1)*Hk(:,:,1)'*inv(S(:,:,1));

for i=1:steps
    % Prediction Step
    dx(:,i+1) = Fk(:,:,i)*dx(:,i);
    if wrapOn == true
        dx = WrapX(dx);
    end
    P(:,:,i+1) = Fk(:,:,i)*P(:,:,i)*Fk(:,:,i)' + Ok(:,:,i)*Q*Ok(:,:,i)';

    % Correction Step
    S(:,:,i+1) = Hk(:,:,i+1)*P(:,:,i+1)*Hk(:,:,i+1)' + R;
    K(:,:,i+1) = P(:,:,i+1)*Hk(:,:,i+1)'*inv(S(:,:,i+1));
    y_est(:,i+1) = y_nom(:,i+1) + Hk(:,:,i+1)*(dx(:,i+1));

    e_y(:,i+1) = y(:,i+1) - y_est(:,i+1);
    if wrapOn == true
        e_y = WrapY(e_y);
    end
    P(:,:,i+1) = (I - K(:,:,i+1)*Hk(:,:,i+1))*P(:,:,i+1);

    dx(:,i+1) = dx(:,i+1) + K(:,:,i+1)*(dy(:,i+1) - Hk(:,:,i+1)*dx(:,i+1));
    if wrapOn == true
        dx = WrapX(dx);
    end
end

x_est = x_nom + dx;
e_x = x - x_est;
if wrapOn == true
```

```
    x_est = WrapX(x_est);
    e_x = WrapX(e_x);
end
end
```

```
function [x_truth,y_truth,x_est,y_est,dx,P,S,e_x,e_y] = LKF_MonteCarlo(Q,R,steps)
load("cooplocalization_finalproj_KFdata.mat");

x_nom = [10 0 pi/2 -60 0 -pi/2]';
u_nom = [2 -pi/18 12 pi/25]';
x_pert = [0 1 0 0 0 0.1]';
Dt = 0.1;

n = size(x_nom,1);
P_init = diag([1 1 0.025 1 1 0.025]);

t = 0:Dt:steps*Dt;
Rtrue(2,2)=1;Rtrue(4,4)=1;Rtrue(5,5)=1;
% Generate truth model outputs for nominal trajectories
[x_truth, y_truth] = GenerateTruth(x_nom, u_nom, P_init, Qtrue, Rtrue, Dt, steps,
true);

% Generate nominal trajectories
[~,x_NL] = ode45(@(t,x) NL_DynModel(t,x,u_nom,zeros(6,1)),t,x_nom);
x_NL = x_NL';
y_NL = zeros(5,length(t));
for i=1:length(t)
    y_NL(:,i) = NL_MeasModel(x_NL(:,i),zeros(5,1));
end

% Generate DT matrices along nominal trajectory
x_nominal = x_NL;
y_nominal = y_NL;

x_nominal = WrapX(x_NL);
y_nominal = WrapY(y_NL);

Fk = zeros(6,6,length(t));
Hk = zeros(5,6,length(t));
Ok = zeros(6,6,length(t));

for i=1:length(t)
    [A_t, B_t, C_t] = Linearize(x_nominal(:,i),u_nom);
    [Fk(:,:,i), ~ , Hk(:,:,i)] = Discretize(A_t,B_t,C_t, Dt);
    Ok(:,:,i) = eye(6);
end

% Run LKF on Data
dx_init = x_truth(:,1)-x_nominal(:,1);
P_init = eye(6);

[x_est,y_est,dx,P,S,e_x,e_y]                                                    =
LKF(dx_init,P_init,x_nominal,y_nominal,x_truth,y_truth,Fk,Hk,Ok,Q,R, true);
end
```

```
clc
clear
load('cooplocalization_finalproj_KFdata.mat')
seed = 100;
rng(seed);
Dt = 0.1;
steps = 1000;
n = 6; p = 5; t = 0:Dt:steps*Dt;

N = 50;         % No. of Monte Carlo runs
```

```
NEES_all = zeros(N,steps+1);
NIS_all = zeros(N,steps+1);

% Tuning parameters
Q = diag([0.0001 0.0001 0.01 0.1 0.1 0.01]);
R = Rtrue;
for i=1:N
    disp(i)
    NEES = zeros(1,steps+1);
    NIS = zeros(1,steps+1);

    [x_truth,y_truth,x_est,y_est,dx,P,S,e_x,e_y] = LKF_MonteCarlo(Q,R,steps);

    for k=1:steps+1
        NEES(:,k) = e_x(:,k)'*inv(P(:,:,k))*e_x(:,k);
        NIS(:,k) = e_y(:,k)'*inv(S(:,:,k))*e_y(:,k);
    end

    NEES_all(i,:) = NEES;
    NIS_all(i,:) = NIS;
end

NEES_bar = zeros(1,steps+1);
NIS_bar = zeros(1,steps+1);

for i=1:steps+1
    NEES_bar(1,i) = mean(NEES_all(:,i));
    NIS_bar(1,i) = mean(NIS_all(:,i));
end

alpha = 0.01;
rx1 = (chi2inv(alpha/2,N*n))./N;
rx2 = (chi2inv(1-alpha/2,N*n))./N;
ry1 = (chi2inv(alpha/2,N*p))./N;
ry2 = (chi2inv(1-alpha/2,N*p))./N;
```

Appendix E – Codes accompanying Question 5

```
% Question 5 - Tune EKF
close all, clearvars, clc
load("cooplocalization_finalproj_KFdata.mat");

x0 = [10 0 pi/2 -60 0 -pi/2]';
u0 = [2 -pi/18 12 pi/25]';
Dt = 0.1;
n = size(x0,1);
steps = 1000;
seed = 100;
rng(seed);

Q = [  .00001     0             1e-6    0       0       0;
        0           .00001      1e-6    0       0       0;
        1e-6        1e-6        .001    0       0       0;
        0           0           0       .015    0       0e-6;
        0           0           0       0       .015    0e-6;
        0           0           0       0e-6    0e-6    .008]./18;

kp = 1;
P0 = diag([(kp*1/2)^2 (kp*1/2)^2 (kp*0.25/2)^2 ...
    (kp*2.5/2)^2 (kp*2.5/2)^2 (kp*0.25/2)^2]);

runs = 50;
EX = zeros(n, steps+1, runs);
p = 5;
EY = zeros(p, steps+1, runs);
PS = zeros(n, n, steps+1, runs);
SS = zeros(p, p, steps+1, runs);
fig1 = figure;
enablePlotDuring = true;
for run = 1:runs
    disp(['run #', num2str(run)]);

    % generate truth for run
    [x, y] = GenerateTruth(x0, u0, P0, Qtrue, Rtrue, Dt, steps, true);
    t = (0:(length(x)-1))*Dt;

    % assume we can get exact measurement noise from
    % specifications of sensors
    R = Rtrue;

    % Run filter for all time-steps of run #k
    [x_est, y_est, P, S] = EKF(x0, P0, u0, y, Q, R, Dt);

    % wrap angle diff too!!
    ex = WrapX(x - x_est);
    ey = WrapY(y - y_est);

    % Plot error during monte carlo runs
    if enablePlotDuring == true
        PlotStates(fig1,t,ex, ['State Errors, Run ',num2str(run)], P);
    end
```

```
    % save run data from NEES/NIS tests
    EX(:, :, run) = ex;
    EY(:, :, run) = ey;
    PS(:, :, :, run) = P;
    SS(:, :, :, run) = S;
end

%% Calculate NEES and NIS statistics
[NEES_bar, NIS_bar] = CalcStats(EX, EY, PS, SS);

%-------------------------------------------------------------------------
% Plots for (a)
PlotStates(fig1,t,ex, ['State Errors, Run ',num2str(run)], P);

fig2 = figure;
fig3 = figure;
fig4 = figure;
PlotMeasurements(fig2,t,y,'Ground Truth Measurements');
PlotStates(fig3,t,x,'Ground Truth States');
PlotMeasurements(fig4,t,ey,['Ground Truth Measurement Errors, Run
',num2str(run)]);

%-------------------------------------------------------------------------
% Plots for (b)
fig5 = figure;
alpha = 0.05;
PlotNees(fig5, NEES_bar, runs, n, alpha);
%-------------------------------------------------------------------------
% Plots for (c)
fig6 = figure;
alpha = 0.01;
PlotNis(fig6, NIS_bar, runs, p, alpha);
```

```
function [x_est, y_est, P, S] = EKF(x0, P0, u, y, Q, R, Dt)

% set simulation tolerances for ode45
opt = odeset('RelTol',1e-6,'AbsTol',1e-6);

n = size(x0, 1);            % number of states
p = size(R, 1);             % number of measurements
steps = size(y,2);          % number of time steps

x_est = zeros(n, steps);    % state estimate vector
y_est = zeros(p, steps);    % measurement estimate vector
P = zeros(n, n, steps);     % coveriance
S = zeros(p, p, steps);

% start with initial estimate of total state
% and covariance
x_p = x0;
P_p = P0;

for i=1:steps
```

```
    %-------------------------------------------------------------
    % Prediction Step
    % use full NL equations to estimate state at next time step
    % using state at previous time step; since Wk is AWGN,
    % its expected value is zero, set input to zero
    wk = zeros(1,n);
    [~, x_m] = ode45(@NL_DynModel, [0.0 Dt], x_p', opt, u', wk);
    x_m = x_m(end,:)';
    x_m = WrapX(x_m);

    % to calculate covariance, linearize "online"
    % about current state estimate
    [A_t,B_t,C_t] = Linearize(x_m, u);
    [F, ~, H] = Discretize(A_t, B_t ,C_t, Dt);
    P_m = F*P_p*F' + Q;

    % uese estimated state from NL ODEs; since Wk is AWGN,
    % its expected value is zero, set to zero
    vk = zeros(p,1);
    y_est(:,i) = NL_MeasModel(x_m, vk);
    y_est = WrapY(y_est);

    % calculate innovation vector
    e_y = y(:,i) - y_est(:,i);
    e_y = WrapY(e_y);

    %-------------------------------------------------------------
    % Correction Step
    % calculate gain using linearized measurement
    % matrix H and covariance from Prediction Step
    S_p = H*P_m*H' + R;
    K = P_m*H'/S_p;

    % calculate posterior state estimate and covariance
    x_p = x_m + K*e_y;
    x_p = WrapX(x_p);
    P_p = (eye(n) - K*H)*P_m;

    % for each time-step, save estimate and covariance
    x_est(:,i) = x_p;
    P(:,:,i) = P_p;
    S(:,:,i) = 0.5*(S_p + S_p');
end

end
```

```
function [x, y] = GenerateTruth(x0, u, P0, Q, R, Dt, steps, wrapOn)
opt = odeset('RelTol',1e-6,'AbsTol',1e-6);
useChol = true;
n = size(x0,1);
p = size(R,1);

x = zeros(n,steps+1);
y = zeros(p,steps+1);
```

```
% set initial conditions
dx = mvnrnd(zeros(1,n),P0);
x(:,1) = x0 + dx';
x(3,1) = wrapToPi(x(3,1));
x(6,1) = wrapToPi(x(6,1));

for i = 2:steps+1

    % generate noisy state
    if useChol==true
        wk = chol(Q)*randn(n,1);
    else
        wk = mvnrnd(zeros(1,n),Q)';
    end
    [~,next_x] = ode45(@NL_DynModel, [0 Dt], x(:,i-1)', opt, u', wk);

    if wrapOn == true
    % wrap angle to [-pi pi]
    next_x(3) = wrapToPi(next_x(3));
    next_x(6) = wrapToPi(next_x(6));
    end
    x(:,i) = next_x(end,:)';
end

for i = 1:steps+1
    % generate noisy measurement
    if useChol==true
        vk = chol(R)*randn(p,1);
    else
        vk = mvnrnd(zeros(1,p),R)';
    end
    y(:,i) = NL_MeasModel(x(:,i), vk);

    if wrapOn == true
    % wrap angle to [-pi pi]
    y(1,i) = wrapToPi(y(1,i));
    y(3,i) = wrapToPi(y(3,i));
    end
end
end
```

```
function [F, G, H] = Discretize(A_t,B_t,C_t, Dt)
%Linearize
%   Inputs:
%   A_t - linearized CT system matrix
%   B_t - lizearized CT input matrix
%   C_t - linearized CT measurement matrix
%   Output
%   F - state transition matrix
%   G -control effect matrix
%   H - sensing matrix
%
%   Discretizze the CT state perturbation matrices

    H = C_t;
    F = eye(size(A_t)) + Dt * A_t;
```

```
    G = Dt * B_t;
end
```

```
function [NEES, NIS] = CalcStats(EX, EY, P, S)

steps = size(EX, 2);
runs = size(EX, 3);

NEES_all = zeros(runs,steps);
NIS_all = zeros(runs,steps);
NEES = zeros(1,steps);
NIS = zeros(1,steps);

for run=1:runs
    for step=1:steps
        NEES(step) = EX(:,step, run)' / P(:,:,step, run) * EX(:,step, run);
        NIS(step) = EY(:,step, run)' / S(:,:,step, run) * EY(:,step, run);
    end

    NEES_all(run,:) = NEES;
    NIS_all(run,:) = NIS;
end

% calculate mean at each time step
for i=1:steps
    NEES(i) = mean(NEES_all(:,i));
    NIS(i) = mean(NIS_all(:,i));
end
end
```

```
function PlotMeasurements(hdl,t,y, title, S)

if nargin > 4
    s = zeros(size(y));
    for ind = 1:size(y,2)
        s(:,ind) = 2*sqrt(diag(S(:,:,ind)));
    end
    displayError = true;
else
    displayError = false;
end

figure(hdl)
if isempty(hdl.Children)
    tiledlayout(3,2);
    ax1 = nexttile;
    ax2 = nexttile;
    ax3 = nexttile;
    ax4 = nexttile;
    ax5 = nexttile;
else
    thdl = hdl.Children;
    ax1 = thdl.Children(5,1);
    ax2 = thdl.Children(4,1);
    ax3 = thdl.Children(3,1);
    ax4 = thdl.Children(2,1);
    ax5 = thdl.Children(1,1);
```

```
end

hold([ax1 ax2 ax3 ax4 ax5],'on');
ftSize = 10;
sgtitle(title,'FontSize',ftSize+2,'Interpreter','latex')
output = 1;
plot(ax1, t,y(output,:))
if displayError == true
    plot(ax1, t, s(output,:),'r--'), plot(ax1, t, -s(output,:),'r--')
end
ylabel(ax1, '$\gamma_{ag}$ (rad)','FontSize',ftSize,'Interpreter','latex')
xlabel(ax1, 'Time(s)','FontSize',ftSize,'Interpreter','latex')
axis([min(t) max(t) min(y(output,:)) max(y(output,:))])
grid on

output = output + 1;
plot(ax5, t,y(output,:))
if displayError == true
    plot(ax5, t, s(output,:),'r--'), plot(ax5, t, -s(output,:),'r--')
end
ylabel(ax5, '$\rho_{ga}$ (m)','FontSize',ftSize,'Interpreter','latex')
xlabel(ax5, 'Time(s)','FontSize',ftSize,'Interpreter','latex')
axis([min(t) max(t) min(y(output,:)) max(y(output,:))])
grid on

output = output + 1;
plot(ax3, t,y(3,:))
if displayError == true
    plot(ax3, t, s(output,:),'r--'), plot(ax3, t, -s(output,:),'r--')
end
ylabel(ax3, '$\gamma_{ga}$ (rad)','FontSize',ftSize,'Interpreter','latex')
xlabel(ax3, 'Time(s)','FontSize',ftSize,'Interpreter','latex')
axis([min(t) max(t) min(y(output,:)) max(y(output,:))])
grid on

output = output + 1;
plot(ax2, t,y(output,:))
if displayError == true
    plot(ax2, t, s(output,:),'r--'), plot(ax2, t, -s(output,:),'r--')
end
ylabel(ax2, '$\xi_a$ (m)','FontSize',ftSize,'Interpreter','latex')
xlabel(ax2, 'Time(s)','FontSize',ftSize,'Interpreter','latex')
axis([min(t) max(t) min(y(output,:)) max(y(output,:))])
grid on

output = output + 1;
plot(ax4, t,y(output,:))
if displayError == true
    plot(ax4, t, s(output,:),'r--'), plot(ax4, t, -s(output,:),'r--')
end
ylabel(ax4, '$\eta_a$ (m)','FontSize',ftSize,'Interpreter','latex')
xlabel(ax4, 'Time(s)','FontSize',ftSize,'Interpreter','latex')
axis([min(t) max(t) min(y(output,:)) max(y(output,:))])
grid on
```

```
grid([ax1 ax2 ax3 ax4 ax5],'on');
hold([ax1 ax2 ax3 ax4 ax5],'off');
end
```

```
function PlotNees(hdl, epsNEESbar, Nsimruns, n, alpha)

figure(hdl);
Nnx = Nsimruns*n;

%%compute intervals:
r1x = chi2inv(alpha/2, Nnx )./ Nsimruns;
r2x = chi2inv(1-alpha/2, Nnx )./ Nsimruns;

figure(hdl)
plot(epsNEESbar,'o','MarkerSize',6),hold on
plot(r1x*ones(size(epsNEESbar)),'r--')
plot(r2x*ones(size(epsNEESbar)),'r--')
ylabel('NEES                                                    statistic,
$\bar{\epsilon}_x$','FontSize',14,'Interpreter','latex')
xlabel('time step, k','FontSize',14)
title('NEES Estimation Results','FontSize',14)
legend('$\bar{\epsilon}_x$','$r_1$','$r_2$','FontSize',12,'Interpreter','lat
ex')
grid on;
end
```

```
function PlotNis(hdl, epsNISbar, Nsimruns, p, alpha)

figure(hdl);
Nny = Nsimruns*p;

%%compute intervals:
r1y = chi2inv(alpha/2, Nny )./ Nsimruns;
r2y = chi2inv(1-alpha/2, Nny )./ Nsimruns;

plot(epsNISbar,'o','MarkerSize',6),hold on
plot(r1y*ones(size(epsNISbar)),'r--')
plot(r2y*ones(size(epsNISbar)),'r--')
ylabel('NIS                                                     statistic,
$\bar{\epsilon}_y$','FontSize',14,'Interpreter','latex')
xlabel('time step, k','FontSize',14)
title('NIS Estimation Results','FontSize',14)
legend('$\bar{\epsilon}_y$','$r_1$','$r_2$','FontSize',12,'Interpreter','lat
ex')
grid on;
end
```

```
function PlotStates(hdl,t,x, title, P)

if nargin > 4
    p = zeros(size(x));
    for ind = 1:size(x,2)
        p(:,ind) = 2*sqrt(diag(P(:,:,ind)));
    end
    displayError = true;
else
    displayError = false;
end
```

```
figure(hdl)
if isempty(hdl.Children)
    tiledlayout(3,2);
    ax1 = nexttile;
    ax2 = nexttile;
    ax3 = nexttile;
    ax4 = nexttile;
    ax5 = nexttile;
    ax6 = nexttile;
else
    thdl = hdl.Children;
    ax1 = thdl.Children(6,1);
    ax2 = thdl.Children(5,1);
    ax3 = thdl.Children(4,1);
    ax4 = thdl.Children(3,1);
    ax5 = thdl.Children(2,1);
    ax6 = thdl.Children(1,1);
end

hold([ax1 ax2 ax3 ax4 ax5 ax6],'on');

ftSize = 10;
sgtitle(title,'FontSize',ftSize+2,'Interpreter','latex')
state = 1;

plot(ax1, t,x(state,:))
if displayError == true
    plot(ax1, t, p(state,:),'r--'), plot(ax1, t, -p(state,:),'r--')
end
ylabel(ax1, '$\xi_g$ (m)','FontSize',ftSize,'Interpreter','latex')
xlabel(ax1, 'Time(s)','FontSize',ftSize,'Interpreter','latex')
axis([min(t) max(t) min(x(state,:)) ...
max(x(state,:))])

state = state + 1;
plot(ax3, t,x(state,:))
if displayError == true
    plot(ax3, t, p(state,:),'r--'), plot(ax3, t, -p(state,:),'r--')
end
ylabel(ax3, '$\eta_g$ (m)','FontSize',ftSize,'Interpreter','latex')
xlabel(ax3, 'Time(s)','FontSize',ftSize,'Interpreter','latex')
axis([min(t) max(t) min(x(state,:)) ...
max(x(state,:))])

state = state + 1;
plot(ax5, t,wrapToPi(x(state,:)))
if displayError == true
    plot(ax5, t, p(state,:),'r--'), plot(ax5, t, -p(state,:),'r--')
end
ylabel(ax5, '$\theta_g$ (rad)','FontSize',ftSize,'Interpreter','latex')
xlabel(ax5, 'Time(s)','FontSize',ftSize,'Interpreter','latex')
axis([min(t) max(t) min(wrapToPi(x(state,:))) ...
max(wrapToPi(x(state,:)))])
```

```
state = state + 1;
plot(ax2, t,x(state,:))
if displayError == true
    plot(ax2, t, p(state,:),'r--'), plot(ax2, t, -p(state,:),'r--')
end
ylabel(ax2, '$\xi_a$ (m)','FontSize',ftSize,'Interpreter','latex')
xlabel(ax2, 'Time(s)','FontSize',ftSize,'Interpreter','latex')
axis([min(t) max(t) min(x(state,:)) ...
max(x(state,:))])

state = state + 1;
plot(ax4, t,x(state,:))
if displayError == true
    plot(ax4, t, p(state,:),'r--'), plot(ax4, t, -p(state,:),'r--')
end
ylabel(ax4, '$\eta_a$ (m)','FontSize',ftSize,'Interpreter','latex')
xlabel(ax4, 'Time(s)','FontSize',ftSize,'Interpreter','latex')
axis([min(t) max(t) min(x(state,:)) ...
max(x(state,:))])

state = state + 1;
plot(ax6, t,wrapToPi(x(state,:)))
if displayError == true
    plot(ax6, t, p(state,:),'r--'), plot(ax6, t, -p(state,:),'r--')
end
ylabel(ax6, '$\theta_a$ (rad)','FontSize',ftSize,'Interpreter','latex')
xlabel(ax6, 'Time(s)','FontSize',ftSize,'Interpreter','latex')
axis([min(t) max(t) min(wrapToPi(x(state,:))) ...
max(wrapToPi(x(state,:)))])

grid([ax1 ax2 ax3 ax4 ax5 ax6],'on');
hold([ax1 ax2 ax3 ax4 ax5 ax6],'off');
end
```

```
function [out] = WrapX(in)
    out = in;
    out(3,:) = wrapToPi(out(3,:));
    out(6,:) = wrapToPi(out(6,:));
end
```

```
function [out] = WrapY(in)
    out = in;
    out(1,:) = wrapToPi(out(1,:));
    out(3,:) = wrapToPi(out(3,:));
end
```

Appendix F – Codes accompanying Question 6

```
% Question 6 - Compare Filter Performance
close all, clearvars, clc
load("cooplocalization_finalproj_KFdata.mat");

useUnwrappedY = false;       % wrap Y for LKF
runMonteCarlo = false;      % run MC or use Mat data
overlayAllRuns = true;      % plot all runs on top of each other

if runMonteCarlo == true
    x0 = [10 0 pi/2 -60 0 -pi/2]';
    u0 = [2 -pi/18 12 pi/25]';
    steps = 1000;
    runs = 50;

    Q_lkf = diag([0.0001 0.0001 0.01 0.1 0.1 0.01]);
    Q_ekf = diag([.0015, .0015, 0.01, 0.001, 0.005, 0.01]);

    Q_ekf = [  .000001  0       1e-6    0       0       0;
               0       .000001 1e-6    0       0       0;
               1e-6    1e-6    .001    0       0       0;
               0       0       0       .015    0       0e-6;
               0       0       0       0       .015    0e-6;
               0       0       0       0e-6    0e-6    .008]./18;
    kp = 1;
    P0 = diag([(kp*1/2)^2 (kp*1/2)^2 (kp*0.25/2)^2 ...
        (kp*2.5/2)^2 (kp*2.5/2)^2 (kp*0.25/2)^2]);
else
    x0 = [10 0 pi/2 -60 0 -pi/2]';
    u0 = [2 -pi/18 12 pi/25]';
    steps = size(ydata,2) - 2;
    runs = 1;
    Q_lkf = diag([0.0001 0.0001 0.01 0.1 0.1 0.01]);
    Q_ekf = diag([.0015, .0015, 0.01, 0.001, 0.005, 0.01]);

    Q_ekf = [  .000001  0       1e-6    0       0       0;
            0       .000001 1e-6    0       0       0;
            1e-6    1e-6    .001    0       0       0;
            0       0       0       .015    0       0e-6;
            0       0       0       0       .015    0e-6;
            0       0       0       0e-6    0e-6    .008]./18;
    P0 = diag([1 1 0.025 1 1 0.025]);
end

p = 5;
Dt = 0.1;
n = size(x0,1);
seed = 100;
rng(seed);

EX_lkf = zeros(n, steps+1, runs);
EY_lkf = zeros(p, steps+1, runs);
PS_lkf = zeros(n, n, steps+1, runs);
SS_lkf = zeros(p, p, steps+1, runs);
```

```
EX_ekf = EX_lkf;
EY_ekf = EY_lkf;
PS_ekf = PS_lkf;
SS_ekf = SS_lkf;

if runMonteCarlo == true
    fig1 = figure;
    fig3 = figure;
    fig5 = figure;
end

fig2 = figure;
fig4 = figure;
fig6 = figure;

for run = 1:runs
    disp(['run #', num2str(run)]);
    %-------------------------------------------------------------------------
    % generate data
    if runMonteCarlo == true
       if useUnwrappedY == true
           [x, y] = GenerateTruth(x0, u0, P0, Qtrue, Rtrue, Dt, steps, false);
       else
           [x, y] = GenerateTruth(x0, u0, P0, Qtrue, Rtrue, Dt, steps, true);
       end
        t = (0:(length(x)-1))*Dt;
        unwrapped_y = y;
    else
        y = ydata(:,2:end);
        unwrapped_y = y;
        if useUnwrappedY == true
            unwrapped_y(1,:) = unwrap(y(1,:));
            unwrapped_y(3,:) =  unwrap(y(3,:));
        end
        x = ones(size(x0,1),size(y,2));
        x(:,1) = x0;
        t = tvec(:,2:end);
        steps = size(y,2) - 1;
    end
    if overlayAllRuns == true
        if runMonteCarlo == true
            PlotStates(fig1,t,x,'Ground Truth States, All Runs');
        end
        PlotMeasurements(fig2,t,y,'Ground Truth Measurements, All Runs');
    end

    % assume we can get exact measurement noise from
    % specifications of sensors
    R = Rtrue;

    %-------------------------------------------------------------------------
    %-------------------------------------------------------------------------
    % LKF

    % generate nominal trajectory for run, along with DT matrices
```

```
    [x_nom,y_nom] = GenerateNom(x0, u0, steps, Dt);
    x_nom = WrapX(x_nom);
    y_nom = WrapY(y_nom);
    [Fk, Hk, Ok] = GenerateLKFMats(u0, x_nom, steps+1, p, Dt);
    dx_init = x0 - x_nom(:,1);
    dx_init = WrapY(dx_init);

    % Run filter for all time-steps of run #k
    [x_est_lkf,y_est_lkf,~,P_lkf,S_lkf,~,~] = LKF(dx_init, P0, x_nom, y_nom,
x, unwrapped_y, ...
        Fk, Hk, Ok, Q_lkf, R, true);

    % wrap angle diff too!!
    ex_lkf = x - x_est_lkf;
    ex_lkf(3,:) = angdiff(x_est_lkf(3,:),x(3,:));
    ex_lkf(6,:) = angdiff(x_est_lkf(6,:),x(6,:));
    ey_lkf = y - y_est_lkf;
    ey_lkf(1,:) = angdiff(y_est_lkf(1,:),y(1,:));
    ey_lkf(3,:) = angdiff(y_est_lkf(3,:),y(3,:));

    % save run data from NEES/NIS tests
    EX_lkf(:, :, run) = ex_lkf;
    EY_lkf(:, :, run) = ey_lkf;
    PS_lkf(:, :, :, run) = P_lkf;
    SS_lkf(:, :, :, run) = S_lkf;
    %-----------------------------------------------------------------------
    %-----------------------------------------------------------------------
    % EKF
    [x_est_ekf, y_est_ekf, P_ekf, S_ekf] = EKF(x0, P0, u0, y, Q_ekf, R, Dt);

    % wrap angle diff too!!
    ex_ekf = x - x_est_ekf;
    ex_ekf(3,:) = angdiff(x_est_ekf(3,:),x(3,:));
    ex_ekf(6,:) = angdiff(x_est_ekf(6,:),x(6,:));
    ey_ekf = y - y_est_ekf;
    ey_ekf(1,:) = angdiff(y_est_ekf(1,:),y(1,:));
    ey_ekf(3,:) = angdiff(y_est_ekf(3,:),y(3,:));

    % save run data from NEES/NIS tests
    EX_ekf(:, :, run) = ex_ekf;
    EY_ekf(:, :, run) = ey_ekf;
    PS_ekf(:, :, :, run) = P_ekf;
    SS_ekf(:, :, :, run) = S_ekf;
    %-----------------------------------------------------------------------
    %-----------------------------------------------------------------------
    % Plot error during monte carlo runs
    if overlayAllRuns == true
        if runMonteCarlo == true
            PlotStates(fig3,t,ex_lkf,     ['LKF     State     Errors,     Runs
',num2str(run)], P_lkf);
            PlotStates(fig5,t,ex_ekf,     ['EKF     State     Errors,     Runs
',num2str(run)], P_ekf);
        end
        PlotMeasurements(fig4,t,ey_lkf,['LKF Ground Truth Measurement Errors,
Runs ',num2str(run)], S_lkf);
```

```
        PlotMeasurements(fig6,t,ey_ekf,['EKF Ground Truth Measurement Errors,
Runs ',num2str(run)], S_ekf);
    end
end

if overlayAllRuns == false
    if runMonteCarlo == true
        PlotStates(fig1,t,x,'Ground Truth States, All Runs');
        PlotStates(fig3,t,ex_lkf, ['LKF State Errors, Runs ',num2str(run)],
P_lkf);
        PlotStates(fig5,t,ex_ekf, ['EKF State Errors, Runs ',num2str(run)],
P_ekf);
    end
    PlotMeasurements(fig2,t,y,'Ground Truth Measurements, All Runs');
    PlotMeasurements(fig4,t,ey_lkf,['LKF Ground Truth Measurement Errors,
Runs ',num2str(run)], S_lkf);
    PlotMeasurements(fig6,t,ey_ekf,['EKF Ground Truth Measurement Errors,
Runs ',num2str(run)], S_ekf);
end

%% Calculate NEES and NIS statistics
[NEES_lkf, NIS_lkf] = CalcStats(EX_lkf, EY_lkf, PS_lkf, SS_lkf);
[NEES_ekf, NIS_ekf] = CalcStats(EX_ekf, EY_ekf, PS_ekf, SS_ekf);

%-------------------------------------------------------------------------
% NEES Plot
alpha = 0.05;
if runMonteCarlo == true
    fig7 = figure;
    PlotNees(fig7, NEES_lkf, runs, n, alpha);
    hold all;
    PlotNees(fig7, NEES_ekf, runs, n, alpha);
    hold off;
    legend('LKF $\bar{\epsilon}_x$','$r_1$','$r_2$',...
        'EKF $\bar{\epsilon}_x$','FontSize',12,'Interpreter','latex')
end
%-------------------------------------------------------------------------
% NIS Plot
fig8 = figure;
PlotNis(fig8, NIS_lkf, runs, p, alpha);
hold all;
PlotNis(fig8, NIS_ekf, runs, p, alpha);
hold off;
legend('LKF $\bar{\epsilon}_y$','$r_1$','$r_2$',...
    'EKF $\bar{\epsilon}_y$','FontSize',12,'Interpreter','latex')

%-------------------------------------------------------------------------
%% Comparison Plots
if runMonteCarlo == true
    fig9 = figure;
    fig11 = figure;
end

fig10 = figure;
fig12 = figure;
```

```
if runMonteCarlo == true
    x(3,:) = wrapToPi(x(3,:));
    x(6,:) = wrapToPi(x(6,:));
    PlotStates(fig9,t,x,'Ground Truth States');
    PlotStates(fig9,t,x_est_ekf,'Ground Truth States');
    PlotStates(fig9,t,x_est_lkf,'Ground Truth States');
    legend('Truth','LKF','EKF');
end

y(3,:) = wrapToPi(y(3,:));
y(1,:) = wrapToPi(y(1,:));
PlotMeasurements(fig10,t,y,'Ground Truth Measurements');
y_est_lkf(3,:) = wrapToPi(y_est_lkf(3,:));
y_est_lkf(1,:) = wrapToPi(y_est_lkf(1,:));
PlotMeasurements(fig10,t,y_est_lkf,'Ground Truth Measurements');
PlotMeasurements(fig10,t,y_est_ekf,'Ground Truth Measurements');
legend('Truth','LKF','EKF');

if runMonteCarlo == true
    PlotStates(fig11,t,ex_lkf, ['State Errors, Run ',num2str(run)]);
    PlotStates(fig11,t,ex_ekf, ['State Errors, Run ',num2str(run)]);
    legend('LKF','EKF');
end
PlotMeasurements(fig12,t,ey_lkf,['Ground  Truth  Measurement  Errors,  Run
',num2str(run)]);
PlotMeasurements(fig12,t,ey_ekf,['Ground  Truth  Measurement  Errors,  Run
',num2str(run)]);
legend('LKF','EKF');
```