Cooperative localization
ASEN 5044 Fall 2023

Pravesh Vadapalli Venkata Guddeti Aniket Anbhule

Due Date: December 16, 2023

1 Part I: Deterministic System Analysis

Select a system from the updated final project system descriptions posted on Canvas. Report which system you selected. Then do the following:

1.1 Question 1

Look at the data file for your project and identify the full duration of the data set and the interval between measurements. Choose a time step suitable for your problem (ΔT = 0.1 for cooperative localization and ΔT = 60 s or 600 s for orbit determination). Generate a full nonlinear simulation of the system dynamics for the specified equilibrium/nominal motion using ode45 in MATLAB (or a similar numerical integration routine), assuming no initial condition errors, no process noise, and no additional control inputs. Use the full nonlinear measurement model to generate simulated perfect measurements with no measurement noise. Plot the time history of each state and measurement for the specified duration.

Given, dynamical equations of motions:

UGV:

$$\dot{\xi}_g = v_g \cos\theta_g + \tilde{w}_{x,g}$$
$$\dot{\eta}_g = v_g \sin\theta_g + \tilde{w}_{y,g}$$
$$\dot{\theta}_g = \frac{v_g}{L} \tan\phi_g + \tilde{w}_{w,g}$$

Where, $\tilde{w_g}(t) = [\tilde{w_{x,g}}\ \tilde{w_{y,g}}\ \tilde{w_{w,g}}]^T$

UAV:

$$\dot{\xi}_a = v_a \cos\theta_a + \tilde{w}_{x,a}$$
$$\dot{\eta}_a = v_a \sin\theta_a + \tilde{w}_{y,a}$$
$$\dot{\theta}_a = w_a \tan\phi_a + \tilde{w}_{w,a}$$

Where, $\tilde{w_a}(t) = [\tilde{w_{x,a}}\tilde{w_{y,a}}\tilde{w_{w,a}}]^T$

Given, the combined system state, control inputs, and disturbance inputs are as follows:

$$x(t) = \left[\xi_g\ \eta_g\ \theta_g\ \xi_a\ \eta_a\ \theta_a\right]^T$$

$$u(t) = \begin{bmatrix} v_g & \phi_g & v_a & \omega_a \end{bmatrix}^T$$

$$\dot{x} = \begin{bmatrix} u_1 \cos_{x_3} \\ u_1 \sin_{x_3} \\ \frac{u_1}{L} \tan_{u_2} \\ u_3 \cos_{x_6} \\ u_3 \sin_{x_6} \\ u_4 \end{bmatrix}$$

$$y(t) = \begin{bmatrix} \tan^{-1}\left(\frac{x_5 - x_2}{x_4 - x_1}\right) - x_3 \\ \sqrt{(x_1 - x_4)^2 + (x_2 - x_5)^2} \\ \tan^{-1}\left(\frac{x_2 - x_5}{x_1 - x_4}\right) - x_6 \\ x_4 \\ x_2 \end{bmatrix}$$

Full Nonlinear Dynamical States Simulation

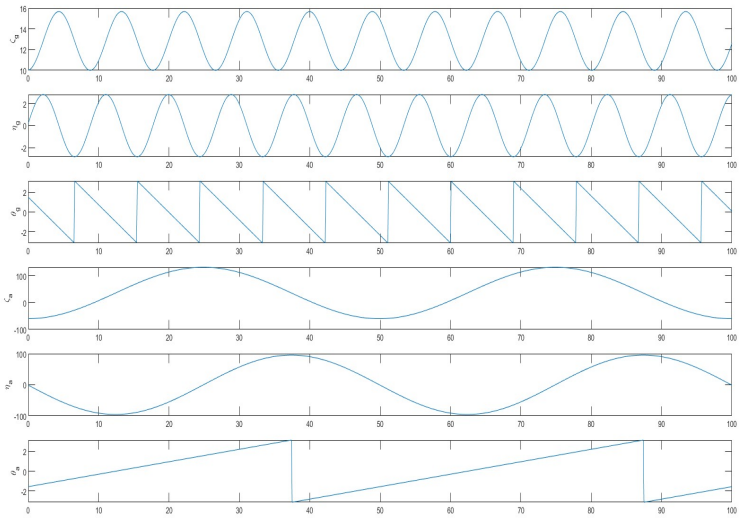

Figure 1: Full Nonlinear State Vs. Time

Refer to Figure 1 and Figure 2 for Full Nonlinear Dynamical state and measurement simulation propagated using ode45 solver in MATLAB.

1.2 Question 2

Derive the CT Jacobians needed to obtain the CT linearized model parameters. Show the key steps and variables needed to find the Jacobian matrices and state the sizes of the results. DO NOT use a symbolic solver or software to find the Jacobians (though you may use these to do a final check of answers).

Linearized/Jacobian CT matrices:

Full Nonlinear Measurement Simulation

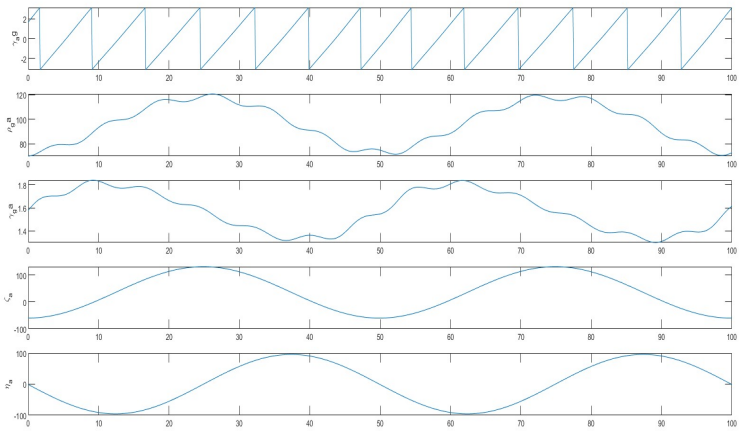

Figure 2: Full Nonlinear Measurement Vs. Time

$$\tilde{A} = \begin{bmatrix} 0 & 0 & -u_1 \sin x_3 & 0 & 0 & 0 \\ 0 & 0 & -u_1 \cos x_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -u_3 \sin x_6 \\ 0 & 0 & 0 & 0 & 0 & -u_3 \cos x_6 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\tilde{B} = \begin{bmatrix} \cos x_3 & 0 & 0 & 0 \\ \sin x_3 & 0 & 0 & 0 \\ \frac{1}{L} \tan u_2 & \frac{u_1}{L} \sec^2 u_2 & 0 & 0 \\ 0 & 0 & \cos x_6 & 0 \\ 0 & 0 & \sin x_6 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\tilde{C} = \begin{bmatrix} \frac{x_5 - x2}{(x_4 - x_1)^2 + (x_5 - x_2)^2} & -\frac{x_4 - x1}{(x_4 - x_1)^2 + (x_5 - x_2)^2} & -1 & -\frac{x_5 - x2}{(x_4 - x_1)^2 + (x_5 - x_2)^2} & \frac{x_4 - x1}{(x_4 - x_1)^2 + (x_5 - x_2)^2} & 0 \\ \frac{x_1 - x4}{\sqrt{(x_1 - x_4)^2 + (x_2 - x_5)^2}} & -\frac{x_2 - x5}{\sqrt{(x_1 - x_4)^2 + (x_2 - x_5)^2}} & 0 & \frac{x_4 - x1}{\sqrt{(x_1 - x_4)^2 + (x_2 - x_5)^2}} & \frac{x_5 - x2}{\sqrt{(x_1 - x_4)^2 + (x_2 - x_5)^2}} & 0 \\ -\frac{x_2 - x5}{(x_1 - x_4)^2 + (x_2 - x_5)^2} & \frac{x_1 - x4}{(x_1 - x_4)^2 + (x_2 - x_5)^2} & 0 & \frac{x_2 - x5}{(x_1 - x_4)^2 + (x_2 - x_5)^2} & \frac{x_1 - x4}{(x_1 - x_4)^2 + (x_2 - x_5)^2} & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

The Jacobian $\tilde{A}$ of f with respect to x is an $n * n$ matrix where, n = number of states of the system. Jacobian $\tilde{B}$ of f with respect to u is an $n * m$ matrix where, m = number of input/control parameters. Jacobian $\tilde{C}$ of h with respect to x is an $p * n$ matrix where, p = number of measurements. Jacobian $\tilde{D}$ of h with respect to u is a null matrix with dimensions of $p * m$.
Jacobian matrix for $\tilde{A}$ and $\tilde{B}$ are fairly straightforward partial derivatives unlike for $\tilde{C}$. The Approach used for solving $\tilde{C}$ is given below:

Let $t_1 = \frac{x_5 - x_2}{x_4 - x_1}$ and $t_2 = \frac{x_2 - x_5}{x_1 - x_4}$ then,

$$\frac{\partial}{\partial x_1}\tan^{-1}(t_1) = \frac{1}{1+t_1^2} * \frac{\partial}{\partial x_1}t_1$$
$$\frac{\partial}{\partial x_1}t_1 = \frac{x_5-x_2}{(x_4-x_1)^2} \quad \text{... (By Quotient rule)}$$

Therefore,

$$\frac{\partial}{\partial x_1}\tan^{-1}(t_1) = \frac{x_5-x_2}{(x_4-x_1)^2+(x_5-x_2)^2}$$
$$\frac{\partial}{\partial x_1}\tan^{-1}(t_2) = \frac{1}{1+t_2^2} * \frac{\partial}{\partial x_1}t_2$$
$$\frac{\partial}{\partial x_1}t_2 = \frac{x_2-x_5}{(x_1-x_4)^2} \quad \text{... (By Quotient rule)}$$

Therefore,

$$\frac{\partial}{\partial x_1}\tan^{-1}(t_2) = \frac{x_2-x_5}{(x_1-x_4)^2+(x_2-x_5)^2}$$

The second partial derivative case:

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

1.3 Quesion 3

Derive the DT linearized model by linearizing your system about its specified equilibrium/nominal motion (given in the description) and find the corresponding DT linearized model matrices for a suitable sampling time Δ T. If applicable, discuss the observability and stability properties of your time-invariant system approximation around the linearization point (if you have a time-varying result, then note this and skip the analysis).

The discrete time, linearized matrices for the initial conditions $x_0 = [10\ 0\ \frac{\pi}{2}\ -60\ 0\ -\frac{\pi}{2}]^T$ and $u_0 = [2\ -\frac{\pi}{2}\ 12\ \frac{\pi}{25}]^T$, at $\Delta T = 0.1$ are as follows:

$$F_0 = \begin{bmatrix} 1 & 0 & \frac{-1}{5} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & \frac{6}{5} \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$G_0 = \begin{bmatrix} 0.0035 & -0.0412 & 0 & 0 \\ 0.1 & 0 & 0 & 0 \\ -0.0353 & 0.4124 & 0 & 0 \\ 0 & 0 & 0 & 0.06 \\ 0 & 0 & -0.1 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix}$$

$$H_0 = \begin{bmatrix} 0 & \frac{1}{70} & -1 & 0 & \frac{-1}{70} & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & \frac{1}{70} & 0 & 0 & \frac{-1}{70} & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

The system is observable since rank of matrix = number of states. The nominal state matrix continuously with each time step, which means the point about which the system is linearized also changes. Hence, the system in time - variant.

1.4 Quesion 4

Simulate the linearized DT dynamics and measurement models near the linearization point for your system, assuming a reasonable initial state perturbation from the linearization point (report the perturbation you chose) and assuming no process noise, measurement noise, or control input perturbations. Use the results to compare and validate your Jacobians and DT model against a full nonlinear simulation of the system dynamics starting from the same initial conditions for the total state vector and again assuming no process noise and no additional control inputs. Compare the linearized measurements to the full nonlinear measurements. Provide suitable labeled plots to report and compare your resulting states and measurements from the linearized DT and full nonlinear DT model. (Run your simulation for the full length of time).

Figure 3 and 4 shows linearized state and measurement simulation of the system with initial state perturbation as $[0; 1; 0; 0; 0; 0.1]^T$. The simulation of initial state perturbations is depicted in Figure 5. The figures show that the states estimated by the linearized system follow the actual system state quite closely. The linearized measurements show a good agreement with the true measurements. States $x_3 and x_6$ and measurments $y_1 and y_3$ were constrained from $[\pi to - \pi]$ or vice-a-versa.

Linearized States Simulation

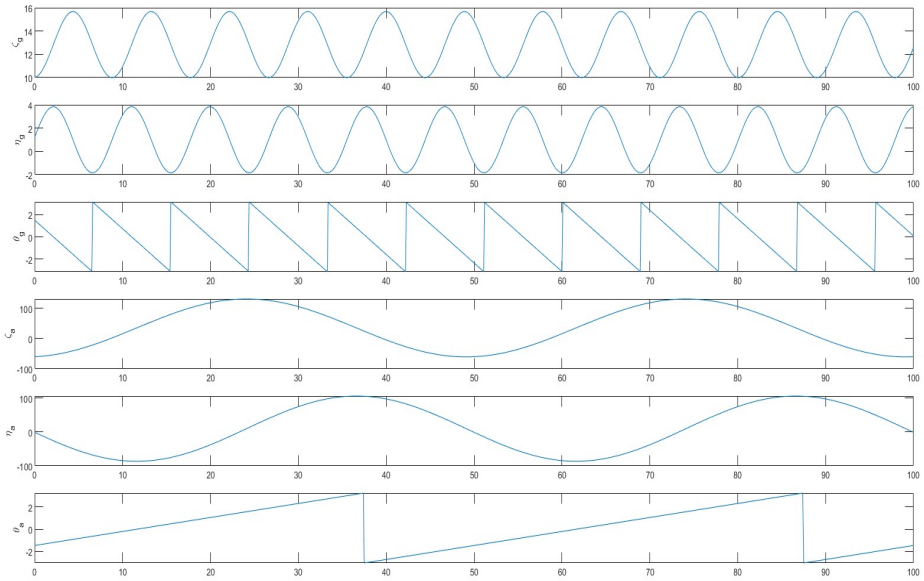

Figure 3: Linearized State Vs. Time

Linearized Measurement Simulation

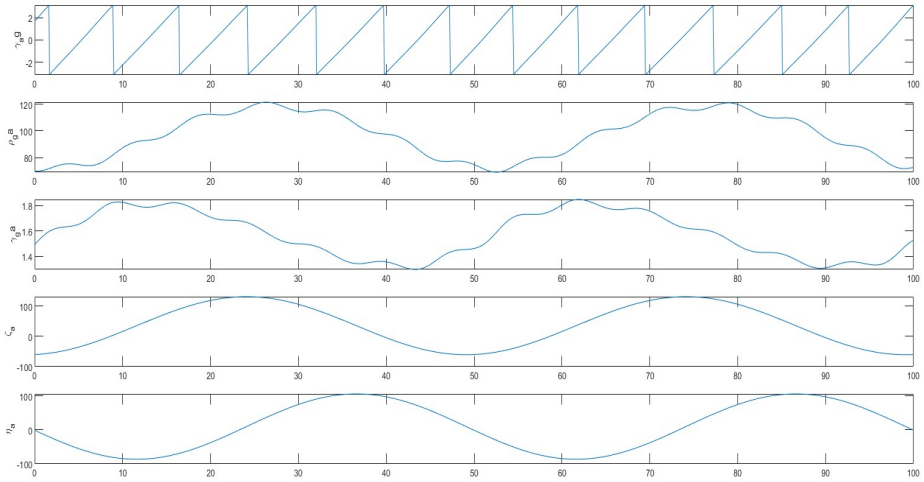

Figure 4: Linearized Measurement Vs. Time

State Perturbation Simulation

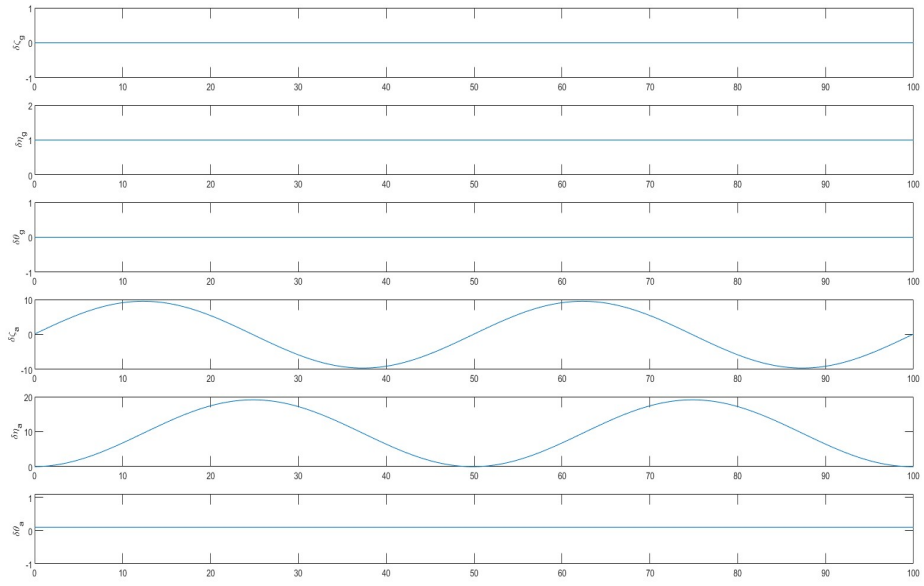

Figure 5: State Perturbations Vs. Time

2 Part II: Stochastic Nonlinear Filtering

2.1 Question 5

Implement and tune a linearized KF using the specified nominal state trajectory (with nominal control inputs) and covariance matrix values posted on Canvas for the DT AWGN process noise and measurement noise for your selected nonlinear system. Use NEES and NIS chi-square tests based on Monte Carlo truth model test (TMT) simulations to tune and validate your linearized KF's performance (note: be sure to explain how the relevant variables in each test can be adapted to the linearized KF). Choose a sufficiently large number of Monte Carlo runs and sample trajectory simulation length to perform the tests, and choose the α value for running each test (provide justification for your choices). Explain how you tuned your linearized KF's guess of the process noise, and provide appropriate plots/illustrations to show that your filter is working properly, namely:
a. Plots for a single 'typical' simulation instance, showing the noisy simulated ground truth states, noisy simulated data, and resulting linearized KF state estimation errors.
b. Plots of the NEES test statistic points from all Monte Carlo simulations vs. time, comparing resulting averages to computed upper/lower bounds for the NEES chi-square test. Explain precisely how the test was set up and how you interpreted the results.
c. Plots of the NIS test statistic points from all Monte Carlo simulations vs. time, comparing resulting averages to computed upper/lower bounds for the NIS chi-square test. Explain precisely how the test was set up and how you interpreted the results.

Q. Explain how you tuned your linearized KF's guess of the process noise, and provide appropriate plots/illustrations to show that your filter is working properly, namely:

Answer: Before tuning the Linearized Kalman Filter, Monte Carlo Truth Model Test (TMT) simulations were performed on the system to analyze the results of Normalized Estimation Error Squared (NEES) and Normalized Innovation Squared (NIS) tests. Initially, N = 50 Monte Carlo TMT simulations were performed, with $Q_{kf} = 1 * Q_{true}$ which resulted in High NEES and low NIS values which means that the filter needs to be appropriately tuned. The team tuned the filter by starting with $Q_{kf} = Q_{true}$ and changed the values by factors of ten till we got decent NEES results. We have fine-tuned by changing the individual values of each diagonal element corresponding to each state of the process noise covariance matrix. Refer to Figure 10 and Figure 11 for NEES and NIS results respectively. We used "*groundtruth*" data obtained from the ode45 solver for each run and included AWGN for each iterative time loop. The algorithm and MATLAB code used for this assignment is included in the CODE section of this Report. Here $\hat{x}, \hat{\delta x}$, and x^* are estimated states, perturbation, and nominal states respectively. Figures 8 and 9 shows estimated states and state estimation error simulation using LKF. Different values of dx were used to get the best results for the least state errors.

LKF Algorithm

$$\hat{x}_{k+1}^{+} = x_{k+1}^{*} + \hat{\delta x}_{k+1}^{+}$$

Time Update/Prediction step for time k+1:

$$\hat{\delta x}_{k+1}^{-} = \tilde{F}_k \hat{\delta x}_k^{+} + \tilde{G}_k \delta u_k$$
$$P_{k+1}^{-} = \tilde{F}_k P_k^{+} \tilde{F}_k^{T} + \tilde{\omega}_k Q_k \tilde{\omega}_k^{T}$$
$$\delta u_{k+1} = u_{k+1} - u_{k+1}^{*}$$

Measurement Update/Correction step for time k+1:

$$K_{k+1} = P_{k+1}^{-} \tilde{H}_{k+1}^{T} [\tilde{H}_{k+1} P_{k+1}^{-} \tilde{H}_{k+1}^{T} + R_{k+1}]^{-1}$$
$$\delta y_{k+1} = y_{k+1} - y_{k+1}^{*}$$

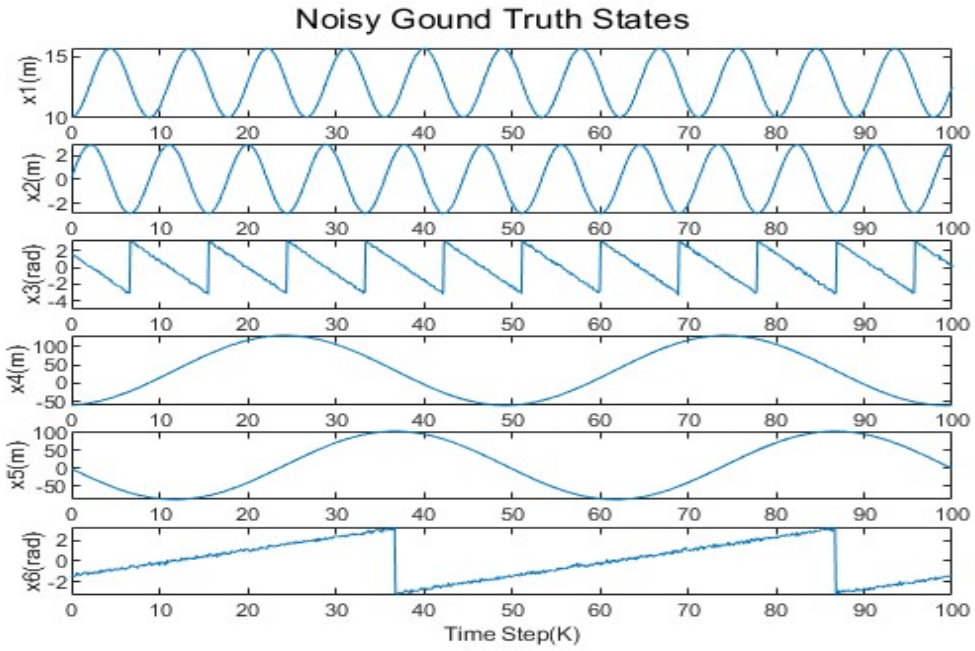

Figure 6: Simulated States Vs. Time

$$\hat{\delta x}^{+}_{k+1} = \hat{\delta x}^{-}_{k+1} + K_{k+1}(\delta y_{k+1} - \tilde{H}_{k+1}\hat{\delta x}^{-}_{k+1})$$
$$P^{+}_{k+1} = (I - K_{k+1}\tilde{H}_{k+1})P^{-}_{k+1}$$

Where,

$$\tilde{\omega_k}| = \Delta T.\Gamma_t|_{t=t_k}$$

$$\tilde{F_k}|_{\hat{x}_k, u_k, w_k=0} = I + \Delta T.\tilde{A}|_{\hat{x}^{+}_k, u_{t_k}, w_{t_k=0}}$$

The Simulated States and simulated measurements using the given process noise covariance matrix Q_{truth} and Measurement noise covariance matrix R_{truth} are given in Figures 6 and 7.

2.1.1

NEES: We calculated NEES values for each iteration time-loop for each Monte Carlo TMT simulation, plot for the same is provided below. We also used chi-sqaure tests to generate bounds using $r_1 = chi2inv(\frac{\alpha}{2}, N.n)/N and r_2 = chi2inv(1 - \frac{\alpha}{2}, N.n)/N$
Where, $\alpha = 0.05$

$$e_{x,k} = x_k - \hat{x}^{+}_k \sim N(0, P^{+}_k)$$
$$\epsilon_{x,k} = e^{T}_{x,k}(P^{+}_k)^{-}1e_{x,k} \sim \chi^2_n$$
$$\bar{\epsilon_{x,k}} = \frac{1}{N}\Sigma^{N}_{i=1}\epsilon^{i}_{x,k}$$

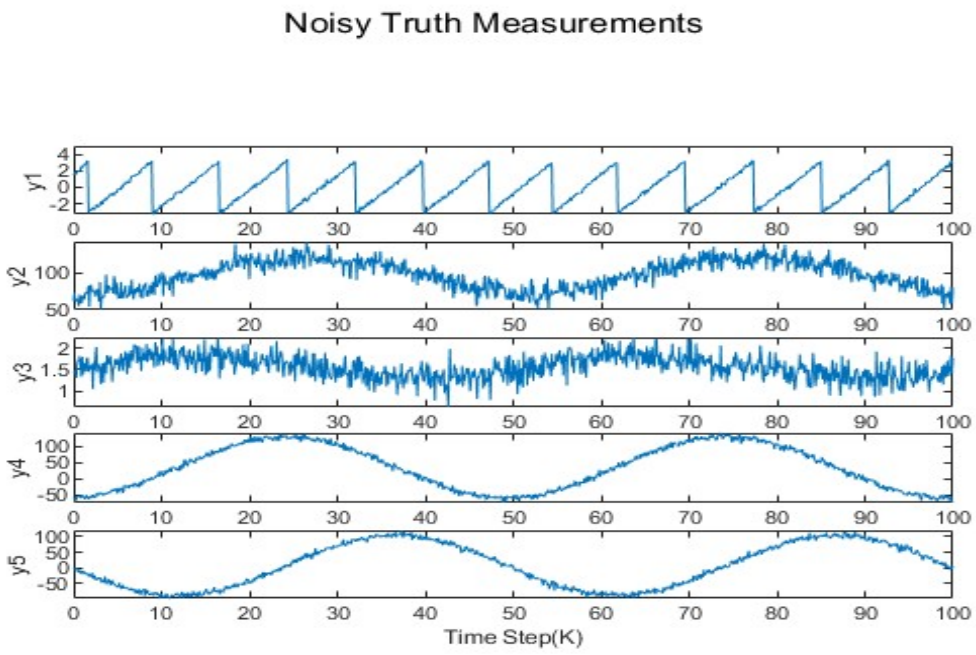

Figure 7: Simulated Measurements Vs. Time

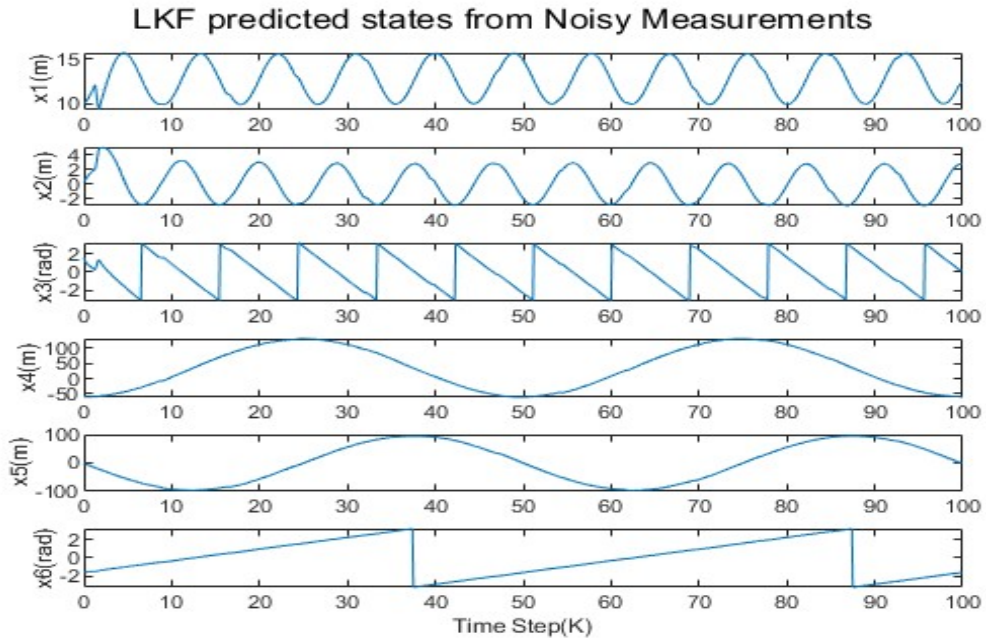

Figure 8: Estimated states LKF Vs. Time

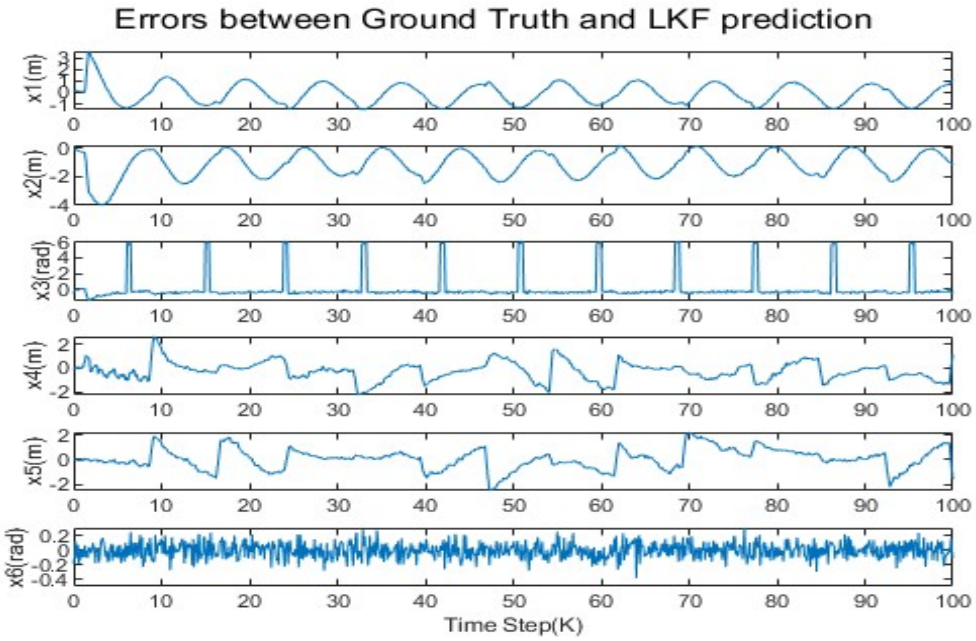

Figure 9: State estimation error Vs. Time

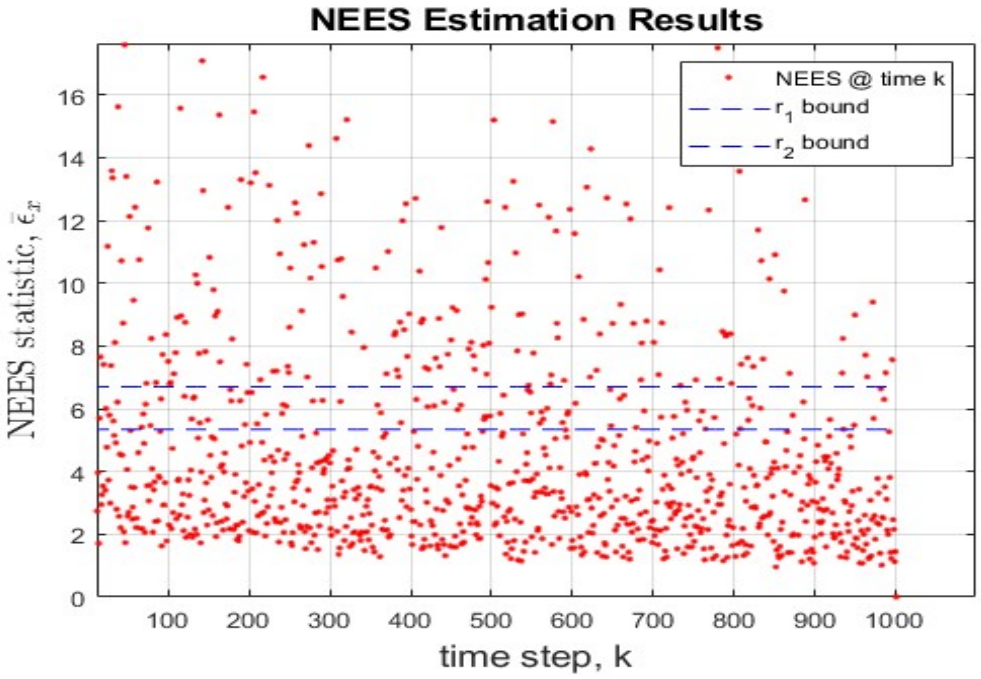

Figure 10: NEES for LKF

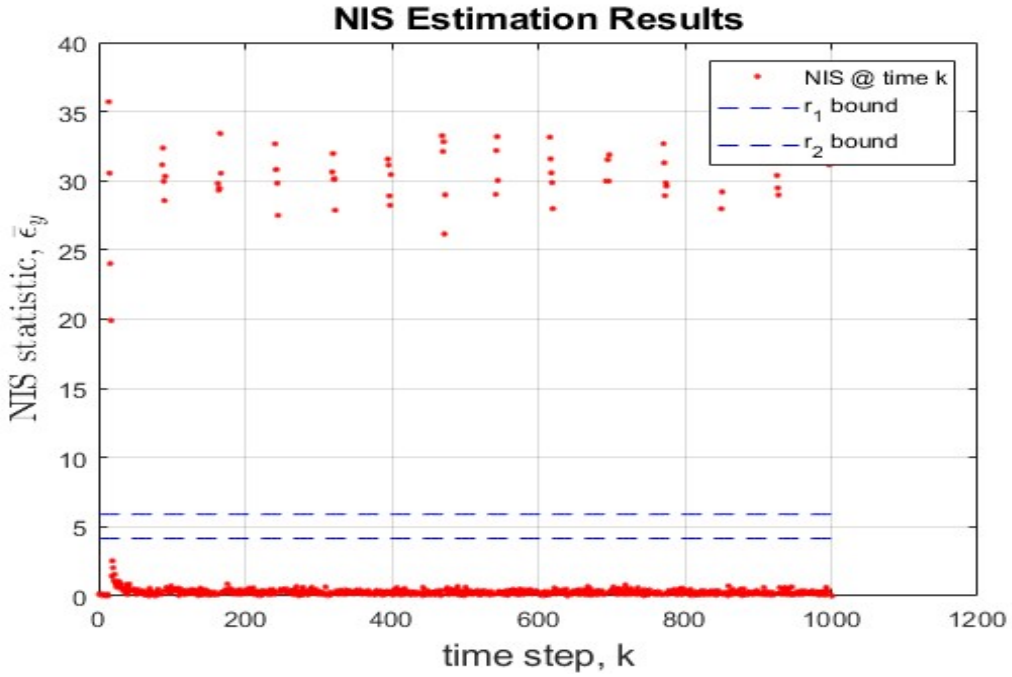

Figure 11: NIS for LKF

2.1.2

NIS: We calculated NIS values for each iteration time-loop for each Monte Carlo TMT simulation, plot for the same is provided below. We also used chi-sqaure tests to generate bounds using $r_1 = chi2inv(\frac{\alpha}{2}, N.n)/N and r_2 = chi2inv(1-\frac{\alpha}{2}, N.n)/N$
Where, $\alpha = 0.05$

$$e_{y,k} = y_k - \hat{y}_k^+ \sim N(0, P_k^+)$$
$$\epsilon_{y,k} = e_{y,k}^T (S_k)^-1 e_{y,k} \sim \chi_n^2$$
$$\bar{\epsilon_{y,k}} = \frac{1}{N} \Sigma_{i=1}^N \epsilon_{y,k}^i$$

Where $S_k = H P_k^- H^T + R$

The dynamics appear too nonlinear to resolve the large mistakes reflected by the NEES and NIS scores, even though the linearized filter does reasonably well in tracking the nonlinear dynamics, especially in the case of the aerial vehicle states. Figure 10 shows NEES results for LKF the points are dispersed but fairly within bounds r_1 and r_2. we adjusted the diagonal elements of the QKF matrix to fine tune the linearized KF's performance. Figure 11 shows NIS scores for LKF system which does not looks very good. It can be due to approximation/Linearization of the system. One way to resolve this issue is to tuning R (measurement covariance matrix) along with Q (process covariance matrix) which results in tuning number of variables which may not a good practice. A large number of points are concentrated below the r1 and r2 lines which means that the filter might overcompensate but given the large distribution of the points, the filter is not as consistent as we want it to be. The value of α for both the NEES and NIS tests was set to 0.05 to accommodate the most points possible.

2.2 Question 6

Implement and tune an extended KF (EKF) using the specified nominal state trajectory along with the control input values and covariance matrix values posted on Can- vas for the DT nonlinear process noise and measurement noise for your selected system. Use NEES and NIS chi-square tests based on Monte Carlo truth model test (TMT) simulations to tune and validate your EKF's performance (be sure to explain how the relevant variables in each test can be adapted to the EKF). Choose a sufficiently large number of Monte Carlo runs and sample trajectory simulation length to perform the tests, and choose the α value for running each test (provide justification for your choices). Explain how you tuned your EKF's guess of the process noise, and provide appropriate plots/illustrations to show that your filter is working properly, namely:
a. Plots for a single 'typical' simulation instance, showing the noisy simulated ground truth states, noisy simulated data, and resulting EKF state estimation errors for each state (with 2σ bounds).
b. Plots of the NEES test statistic points from all Monte Carlo simulations vs. time, comparing the resulting averages to computed upper/lower bounds for the NEES chi-square test.
c. Plots of the NIS test statistic points from all Monte Carlo simulations vs. time, com- paring the resulting averages to computed upper/lower bounds for the NIS chi-square test.

EKF Algorithm

Initialization: $\hat{x}_0^+, \hat{P}_0^+$

Time Update/Prediction step for time k+1:

$$\hat{x}_{k+1}^- = f[\hat{x}_k^+, u_k, w_k = 0]$$
$$P_{k+1}^- = \tilde{F}_k P_k^+ \tilde{F}_k^T + \tilde{\omega}_k Q_k \tilde{\omega}_k^T$$

Measurement Update/Correction step for time k+1:

$$\hat{y}_{k+1}^- = h[\hat{x}_{k+1}^-, V_{k+1} = 0]$$
$$\tilde{H}_{k+1} = \frac{\partial h}{\partial x}|_{\hat{x}_{k+1}^-}$$
$$\tilde{e}_{y,k+1} = y_{k+1} - \hat{y}_{k+1}^-$$
$$\tilde{K}_{k+1} = P_{k+1}^- \tilde{H}_{k+1}^T [\tilde{H}_{k+1} P_{k+1}^- \tilde{H}_{k+1}^T + R_{k+1}]^{-1}$$
$$\hat{x}_{k+1}^+ = \hat{x}_{k+1}^- + \tilde{K}_{k+1} \tilde{e}_{y,k+1}$$
$$P_{k+1}^+ = (I - \tilde{K}_{k+1} \tilde{H}_{k+1}) P_{k+1}^-$$

Where,
$\tilde{\omega}_k| = \Delta T.\Gamma_t|_{t=t_k}$
$\tilde{F}_k|_{\hat{x}_k, u_k, w_k=0} = I + \Delta T.\tilde{A}|_{\hat{x}_k^+, u_{t_k}, w_{t_{k=0}}}$

Figure 12 shows the groundtruth states for the UGV and UAV with the noise in the system. Figure 13 shows the true measurements for UGV and UAV with the noise in the system. $Q_t true$ and $R_t true$ matrices were used for generating process noise and measurement noise. Note that when the groundtruth varies from $-\pi$ to π or vice versa, the estimated bearing for both vehicles tends to fluctuate. This is due to the fact that the bearing is limited to the interval $[\pi, -\pi]$, and slight inaccuracies may lead the estimate to wrap to the other end of the range when it approaches the ends of the range. Figure 14 shows the states predicted by EKF from noisy measurements, the predicted states by the EKF closely match with the ground truth states. Figure 15 shows state estimate errors plotted with 2σ bounds. The procedure used to tune the filter is similar to that used in tuning the LKF refer to section 2.1. Figure 16 and Figure 17 depicts results for NEES and NIS results. From Figure 16 we can say that the filter is optimistic i.e. the P_e matrix is not too small or not too big, since most of the NEES points are between bounds $r_1 and r_2$ but graphs for state $x_1 and x_2$ shows conservative behavior of the filter this is because the first two diagonal elements of Q_{kf} matrix are larger than expected so we tried reducing those elements which resulted in throwing the state propagation out of the bounds. Therefore, we

decided to keep the dynamics as it is since it gives good results. From Figure 17 we can see that most of the points are below the lower bounds which means that the filter is overestimating the measurement noise and the R matrix (measurement noise covariance matrix) is larger than expected. This can be tuned/adjusted by reducing the R matrix. The α value for this is the same as in LKF to accommodate maximum points.

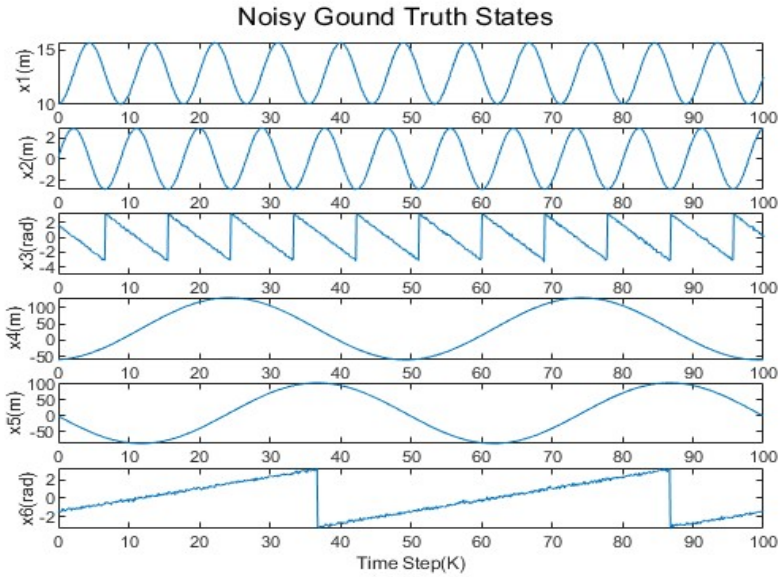

Figure 12: True States EKF Vs. Time

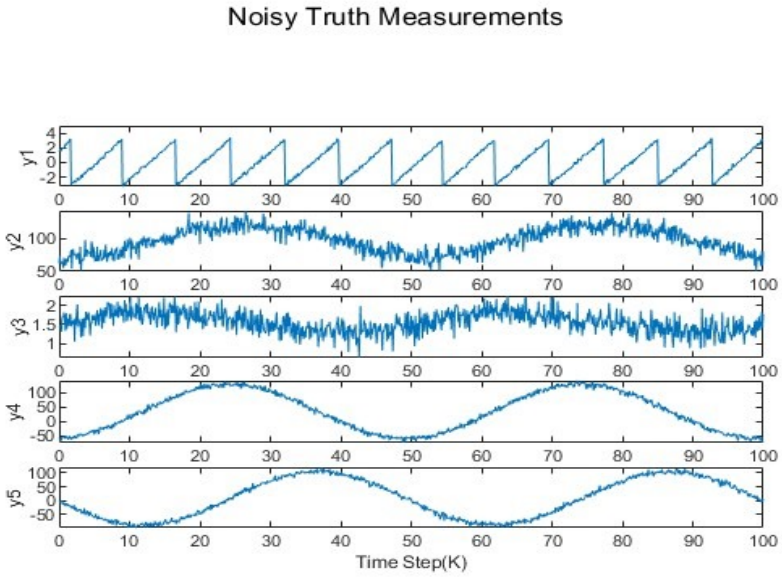

Figure 13: True Measurements Vs. Time

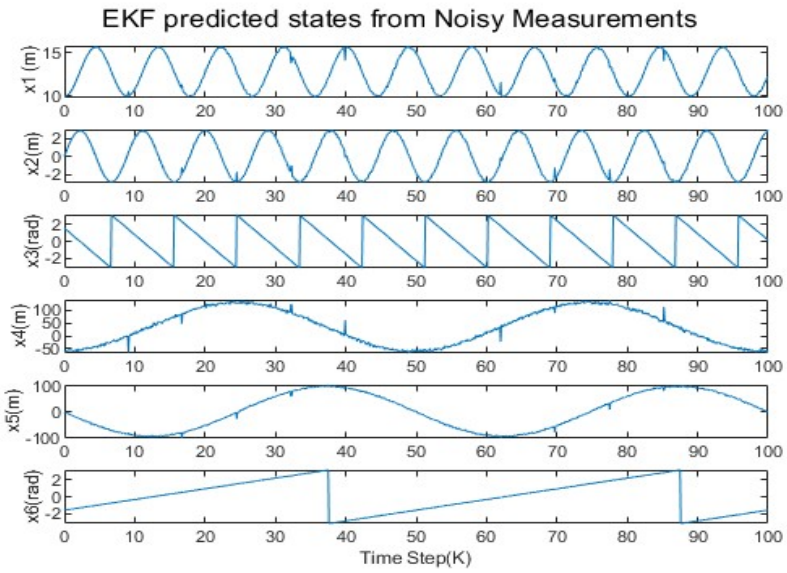

Figure 14: EKF State Estimates Vs. Time

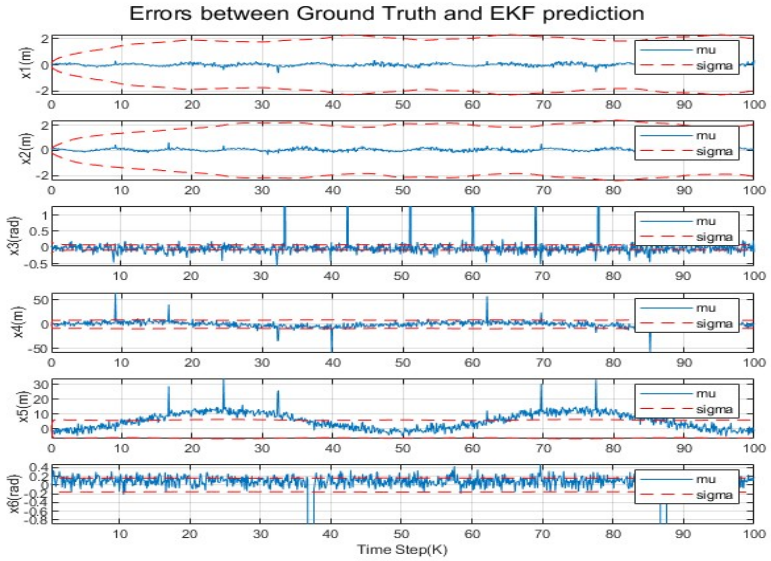

Figure 15: State Error Vs. Time

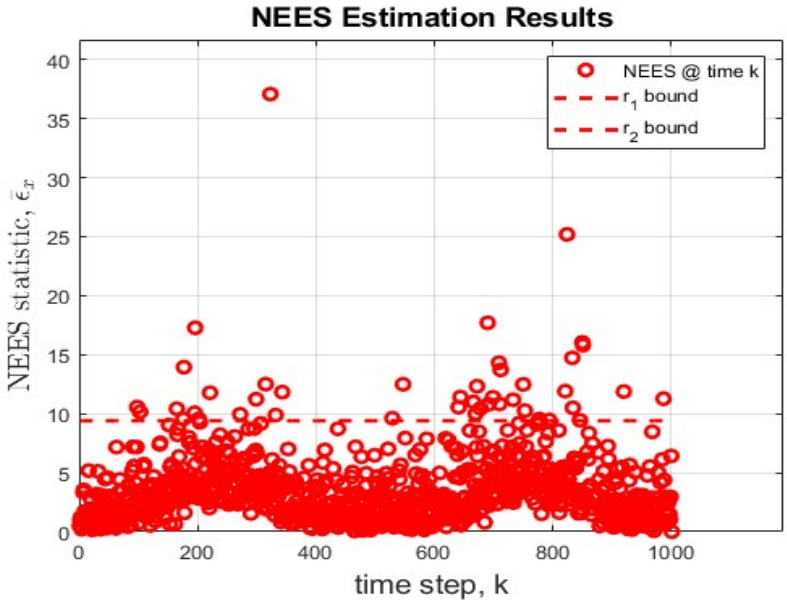

Figure 16: NEES for Extended Kalman Filter

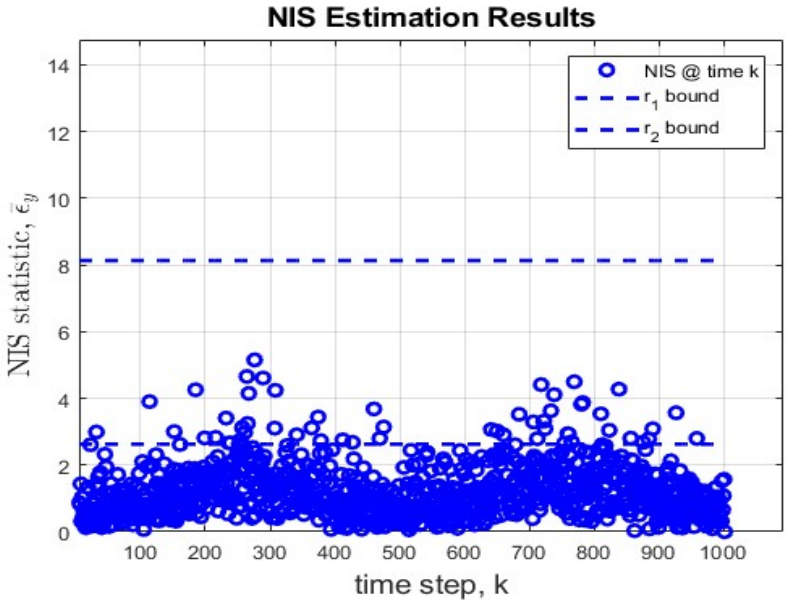

Figure 17: NIS for Extended Kalman Filter

2.3 Question 7

Implement your linearized KF and EKF to estimate the state trajectory for the observation data log posted for your system on Canvas. Turn in plots of the estimated states and 2σ (make sure these are legible/readable). Compare your results – do you think the linearized KF or EKF does a better job (justify)?

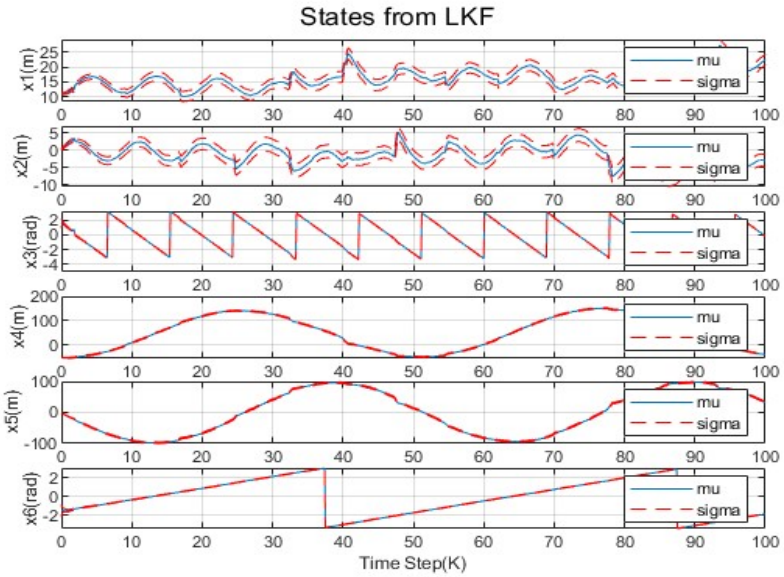

Figure 18: LKF states using observation data log Vs. Time step

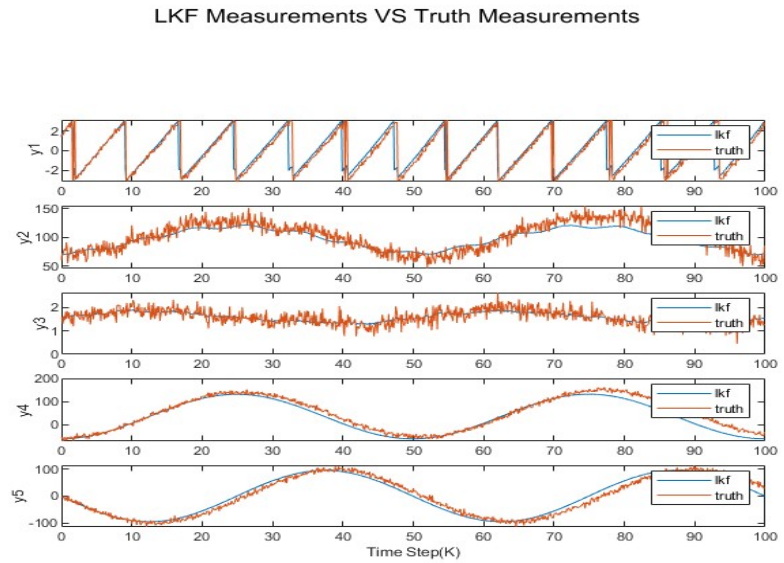

Figure 19: LKF measurements using observation data log Vs. Time step

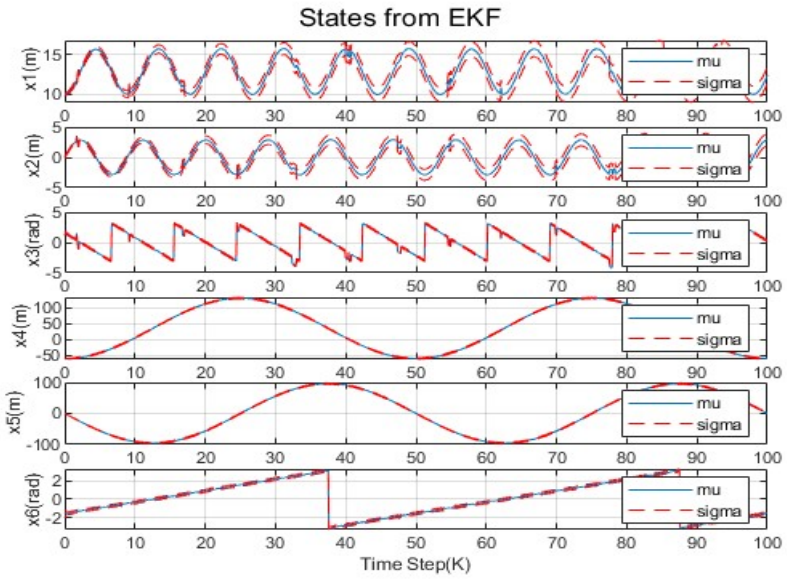

Figure 20: EKF states using observation data log Vs. Time step

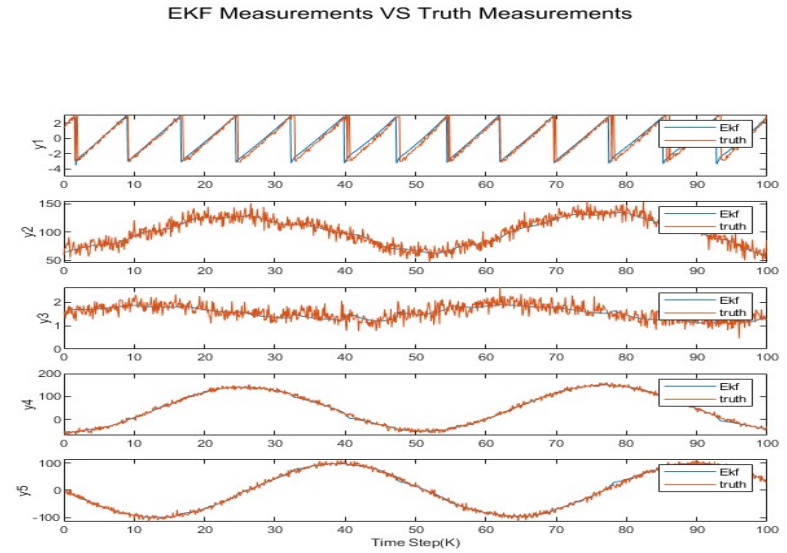

Figure 21: EKF measurements using observation data log Vs. Time step

Comparison Between LKF and EKF: The EKF is noted to perform much better in terms of estimation accuracy compared to the LKF. This implies that the EKF provides more accurate and reliable state estimates for both UAV and UGV. NEES and NIS Scores: The EKF consistently outperforms the LKF in these metrics. This suggests that the EKF's predictions align better with the actual system behavior. The LKF does a reasonably good job of estimating UAV states but falls short in providing usable estimates for UGV states. This discrepancy suggests that the linearization process might be less effective for UGV dynamics. This could be because of Euler approximation

used in the state propagation function and measurement propagation function of the LKF might be contributing to the challenges. The linearization point for the state perturbation may be inaccurate, affecting the propagation of the state covariance. Similar challenges could be pvaresent in the handling of measurement perturbation and measurement covariance, further impacting the accuracy of UGV state estimates. Since, EKF uses nonlinear functions for states and measurement propagation without Euler's approximation it provides more accurate estimation compared to LKF. Figures 18, 19, 20, and 21, shows Observation data log for states and measurements plotted using LKF and EKF.

3 Advanced Questions

3.1 AQ1

Implement the Unscented Kalman Filter (UKF, aka the sigma point filter or SPF) for your system, using the data log provided. Explain how you tuned the filter, and perform NIS tests to validate the performance. Compare the results to those obtained using the linearized KF and EKF, and comment on the results.

Given below are the results obtained from UKF. Please note that despite our best efforts within our time constraints, the UKF has performed worse than EKF. This could be due to the filter needing better tuning.

Tuning: We have tuned the UKF by changing the value of α. We have started with 10^{-1} and changed the value by factors of 10 and have settled on 10^{-4}. This gave the best results and also coincided with the values given in a few papers. We have also changed the initial value of P to better the UKF results.

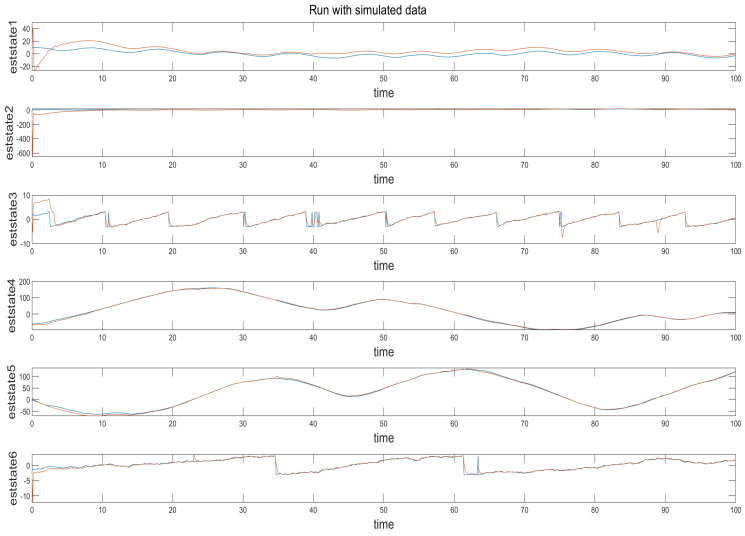

Figure 22: UKF states Vs. Time

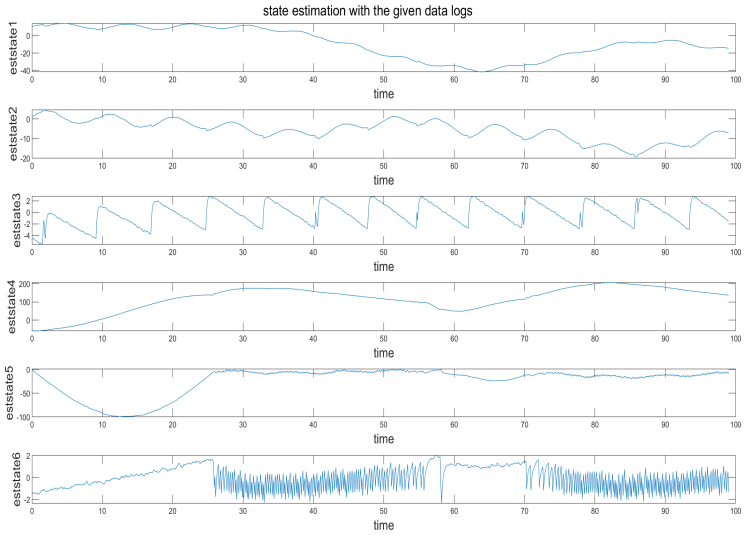

Figure 23: UKF observation data log Vs. Time

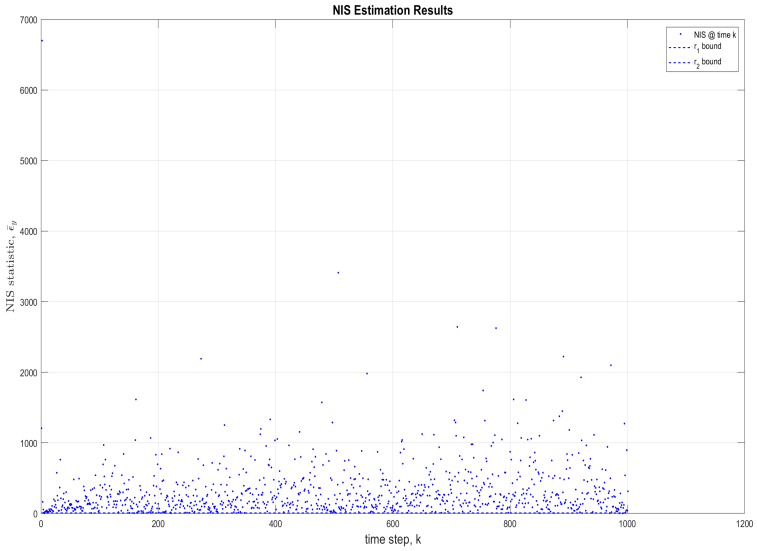

Figure 24: NIS for UKF

UKF:

$$\textit{Dynamics Prediction Step from time step } k \rightarrow k{+}1\textit{:}$$
$$\hat{x}^-_{k+1} \approx \sum_{i=0}^{2n} w^i_m . \chi^{-i}_{k+1}$$
$$P^-_{k+1} \approx \sum_{i=0}^{2n} w^i_c . (\chi^{-i}_{k+1} - \hat{x}^-_{k+1})(\chi^{-i}_{k+1} - \hat{x}^-_{k+1})^T + Q_k$$

$$\textit{Measurement Update Step at time } k{+}1 \textit{ given observation } y(k{+}1)\textit{:}$$
$$\hat{x}^+_{k+1} = \hat{x}^-_{k+1} + K_{k+1}(y_{k+1} - \hat{y}^-_{k+1})$$
$$P^+_{k+1} = P^-_{k+1} - K_{k+1} P_{yy,k+1} K^T_{k+1}$$
$$P^+_{k+1} = P^-_{k+1} - P_{xy,k+1}[P_{yy,k+1}]^{-1} P^T_{xy,k+1}$$

It can be noted that the UKF fails considerably after the first sharp drop of $x6$. Further debugging is required to pinpoint the issue.

3.2 AQ2

Using either your linearized or EKF implementation, investigate and discuss the effect of reducing the number of measurements on your solution accuracy. You can limit the number either by increasing the time gaps between measurements or by reducing the field of view of your sensors (if applicable to your project).

We have used LKF to test the effect of reduced measurements using the simulated data. This was done by taking calculating the measurement step once every 10 timesteps and once every 20 steps. The estimated outputs from LKF were saved and plotted together with the default (measurement every time step). The results are shown in figure 24.

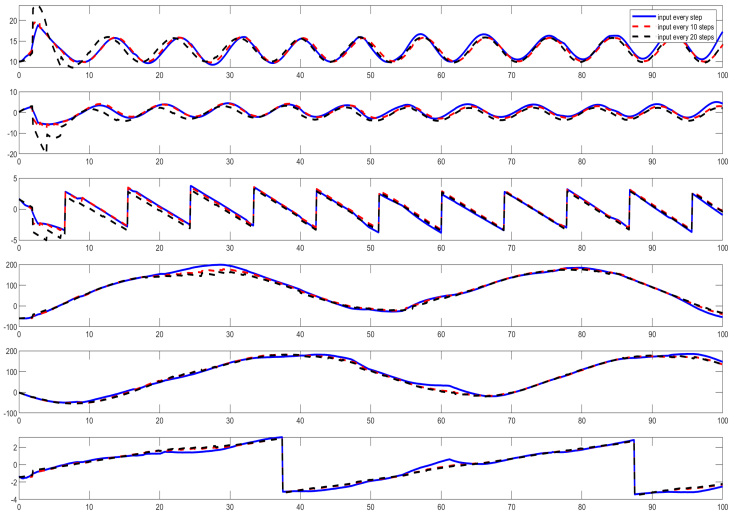

Figure 25: LKF estimates Vs. time

The LKF estimates were surprisingly more accurate when skipping measurements. Not skipping the measurements was, however, more accurate at the initial time steps. This can be due to the accumulation of errors happening at each measurement step outweighing the process noise, meaning that the filter needs more tuning.

4 CODE

Part 1

```
%Part 1
%Non-Linear Propogation of states
dt = 0.1;
L = 0.5;
t = 0:dt:100;
n = length(t);
xnom = [10;0;pi/2;-60;0;-pi/2];
dx0 =  [0;1;0;0;0;0.1];
%dx0 = [1;0;0.1;6;0.1;0.1];
unom = [2,-pi/18, 12, pi/25];
%options = odeset('RelTol', 1e-12);
xdot = @(x, u)[u(1)*cos(x(3)); u(1)*sin(x(3));u(1)*tan(u(2))/L;u(3)*cos(x(6));u(3)*sin(x
xdotwrap = @(t, x) xdot(x, [2,-pi/18, 12, pi/25]);

xold = xnom;
xmat = [];
delxold = dx0;
delxmat = [];
ydall = [];
yfmat = [];
ytota = [];
for i = 0:dt:100

    [tode, y] = ode45(xdotwrap, [0 dt], xold);
    xnew = y(end, :);
    xnew(3) = constrainAngle(xnew(3));
    xnew(6) = constrainAngle(xnew(6));
    xmat = [xmat; xnew];
    xold = xnew;

    [F, G, H] = linStateMats(xnew, unom, L, dt);
    delxnew = F*delxold;
    delxmat = [delxmat delxnew];
    delxold = delxnew;

    yfvar = getY(xnew');
    yfdall = H*delxnew;
    ytot = yfvar + yfdall;
    ytot(1) = constrainAngle(ytot(1));
    ytot(3) = constrainAngle(ytot(3));
    yfvar(1) = constrainAngle(yfvar(1));
    yfvar(3) = constrainAngle(yfvar(3));
    ytota = [ytota ytot];
    yfmat = [yfmat yfvar];
end
yfmat
%q1 NL plots for x
figure(1)
```

```
subplot(6,1,1),plot(t, xmat(:,1)),ylabel('x1');
subplot(6,1,2),plot(t, xmat(:,2)),ylabel('x2');
subplot(6,1,3),plot(t, xmat(:,3)),ylabel('x3');
subplot(6,1,4),plot(t, xmat(:,4)),ylabel('x4');
subplot(6,1,5),plot(t, xmat(:,5)),ylabel('x5');
subplot(6,1,6),plot(t, xmat(:,6)),ylabel('x6');

%q1 NL plots for y
figure(2)
subplot(5,1,1),plot(t,yfmat(1,:)), ylabel("y1");
subplot(5,1,2),plot(t,yfmat(2,:)), ylabel("y2");
subplot(5,1,3),plot(t,yfmat(3,:)), ylabel("y3");
subplot(5,1,4),plot(t,yfmat(4,:)), ylabel("y4");
subplot(5,1,5),plot(t,yfmat(5,:)), ylabel("y5");


%linearized plots

%plots for linearized y
figure(3)
subplot(5,1,1),plot(t,ytota(1,:)), ylabel("y1");
subplot(5,1,2),plot(t,ytota(2,:)), ylabel("y2");
subplot(5,1,3),plot(t,ytota(3,:)), ylabel("y3");
subplot(5,1,4),plot(t,ytota(4,:)), ylabel("y4");
subplot(5,1,5),plot(t,ytota(5,:)), ylabel("y5");


%plots for linearized total x
xfull = xmat + delxmat';
figure(4)
subplot(6,1,1),plot(t, xfull(:,1)),ylabel('x1');
subplot(6,1,2),plot(t, xfull(:,2)),ylabel('x2');
subplot(6,1,3),plot(t, xfull(:,3)),ylabel('x3');
subplot(6,1,4),plot(t, xfull(:,4)),ylabel('x4');
subplot(6,1,5),plot(t, xfull(:,5)),ylabel('x5');
subplot(6,1,6),plot(t, xfull(:,6)),ylabel('x6');

%plots for perturbations
figure(5)
subplot(6,1,1),plot(t, delxmat(1,:)),ylabel('dx1');
subplot(6,1,2),plot(t, delxmat(2,:)),ylabel('dx2');
subplot(6,1,3),plot(t, delxmat(3,:)),ylabel('dx3');
subplot(6,1,4),plot(t, delxmat(4,:)),ylabel('dx4');
subplot(6,1,5),plot(t, delxmat(5,:)),ylabel('dx5');
subplot(6,1,6),plot(t, delxmat(6,:)),ylabel('dx6');
```

PART 2

Q5

```
    %Q5
% Linearized Kalman Filter
```

```
%dx0 = [0;0;0.1;0;0;0.1]; %3
dx0 =  [0;1;0;0;0;0.1];
x0 = xnom;
N = 50; % number of monte carlo simulations
%LKF
rng(100);
for j = 1:N
    xwn = [];
    ywn = [];
    Rkf = Rtrue; %1
    %Qkf = 0.5*Qtrue; %2
    %Qkf = diag([0.005*0.001,0.01*0.001,0.01*0.01,10*0.001,30*0.001,0.01*0.01]);
    Q_ek = diag([500*0.001,500*0.001,10*0.1,5000,500*1,10*0.01]);
    x_all = zeros(6,n);
    x_all(:,1) = dx0;
    sg = dt*eye(6);
    [F, G, H] = linStateMats(xmat(1,:), unom, L, dt);
    xlkt = zeros(6,n);
    xlkt(:,1) = transpose(xmat(1,:)) + x_all(:,1);
    %p = inv(transpose(Hall)*inv(Rall)*Hall);
    p = 5*Qtrue; %4
    x0int = mvnrnd(x0, p);
    nees_samp = zeros(N,n);
    nis_samp = zeros(N,n);
    for i = 0:dt:100
        %Noisy Ground states
        [tode, y1] = ode45(xdotwrap, [0 dt], x0int);
        xn1 = y1(end, :);
        xn1(3) = constrainAngle(xn1(3));
        xn1(6) = constrainAngle(xn1(6));
        xn2 = xn1 + mvnrnd(zeros(1,6), Qtrue);
        xwn = [xwn ;xn2];
        x0int = xn1;

        %Noisy Measurements
        yw1 = getY(xn1');
        yw1(1) = constrainAngle(yw1(1));
        yw1(3) = constrainAngle(yw1(3));
        yw1 = yw1 + transpose(mvnrnd(zeros(1,5), Rtrue));
        ywn = [ywn yw1];
    end


    for i = 1:1000
        %prediction step
        x_all(:,i+1) = F*x_all(:,i);
        p = F*p*transpose(F) + sg*Qkf*transpose(sg);
        [F, G, H] = linStateMats(xmat(i+1,:), unom, L, dt);
        S = H*p*transpose(H)+Rkf;
        ypred = H*x_all(:,i+1);
        K = p*transpose(H)*inv(H*p*transpose(H)+Rkf);
        %measurement update
```

```
        x_all(:,i+1) = x_all(:,i+1) + K*((ywn(:,i)-yfmat(:,i)) - ypred);
        p = (eye(6) - K*H)*p;
        xlkt(:,i+1) = transpose(xmat(i+1,:)) + x_all(:,i+1);

        %NEES/NIS
        S = 0.5*(S+transpose(S));
        nees_samp(j,i) = transpose(transpose(xwn(i,:)) - xlkt(:,i))*inv(p)*(transpose(xw
        nis_samp(j,i) = transpose(ywn(:,i)-(ypred+yfmat(:,i)))*inv(S)*(ywn(:,i)-(ypred+y

    end
end


%NEES test
nees_bar = mean(nees_samp,1);
alpha_nees = 0.05;
Nnx = N*6;
r1x = chi2inv(alpha_nees/2, Nnx )/ N
r2x = chi2inv(1-alpha_nees/2, Nnx )/ N

figure(60)
plot(nees_bar,'r.'),hold on
plot(r1x*ones(size(nees_bar)),'b--')
plot(r2x*ones(size(nees_bar)),'b--'),hold off
ylabel('NEES statistic, $\bar{\epsilon}_x$','Interpreter','latex', 'FontSize',14)
xlabel('time step, k','FontSize',14)
title('NEES Estimation Results','FontSize',14)
legend('NEES @ time k', 'r_1 bound', 'r_2 bound'),grid on

%NIS Test
epsNISbar = mean(nis_samp,1);
alphaNIS = 0.05;
Nny = N*5;
%%compute intervals:
r1y = chi2inv(alphaNIS/2, Nny )./ N
r2y = chi2inv(1-alphaNIS/2, Nny )./ N
f=figure(90)
plot(epsNISbar,'r.'),hold on
plot(r1y*ones(size(epsNISbar)),'b--')
plot(r2y*ones(size(epsNISbar)),'b--'),hold off
ylabel('NIS statistic, $\bar{\epsilon}_y$','Interpreter','latex','FontSize',14)
xlabel('time step, k','FontSize',14)
title('NIS Estimation Results','FontSize',14)
legend('NIS @ time k', 'r_1 bound', 'r_2 bound'),grid on

ywn
xwn

%plot noisy ground truth and measurements
figure(7)
subplot(6,1,1),plot(t, xwn(:,1)),ylabel('noisy x1');
subplot(6,1,2),plot(t, xwn(:,2)),ylabel('noisy x2');
```

```
subplot(6,1,3),plot(t, xwn(:,3)),ylabel('noisy x3');
subplot(6,1,4),plot(t, xwn(:,4)),ylabel('noisy x4');
subplot(6,1,5),plot(t, xwn(:,5)),ylabel('noisy x5');
subplot(6,1,6),plot(t, xwn(:,6)),ylabel('noisy x6');

figure(8)
subplot(5,1,1),plot(t, ywn(1,:)),ylabel('noisy y1');
subplot(5,1,2),plot(t, ywn(2,:)),ylabel('noisy y2');
subplot(5,1,3),plot(t, ywn(3,:)),ylabel('noisy y3');
subplot(5,1,4),plot(t, ywn(4,:)),ylabel('noisy y4');
subplot(5,1,5),plot(t, ywn(5,:)),ylabel('noisy y5');


figure(9)
subplot(6,1,1),plot(t, xlkt(1,:)),ylabel(' x1(m)');
subplot(6,1,2),plot(t, xlkt(2,:)),ylabel('x2(m)');
subplot(6,1,3),plot(t, xlkt(3,:)),ylabel('x3(rad)');
subplot(6,1,4),plot(t, xlkt(4,:)),ylabel('x4(m)');
subplot(6,1,5),plot(t, xlkt(5,:)),ylabel('x5(m)');
subplot(6,1,6),plot(t, xlkt(6,:)),ylabel('x6(rad)');
xlabel('Time Step(K)');
sgtitle("LKF predicted states from Noisy Measurements");

figure(10)
subplot(6,1,1),plot(t,xwn(:,1)'-xlkt(1,:)),ylabel(' x1(m)');
subplot(6,1,2),plot(t, xwn(:,2)'-xlkt(2,:)),ylabel('x2(m)');
subplot(6,1,3),plot(t, xwn(:,3)'-xlkt(3,:)),ylabel('x3(rad)');
subplot(6,1,4),plot(t,xwn(:,4)'-xlkt(4,:)),ylabel('x4(m)');
subplot(6,1,5),plot(t, xwn(:,5)'-xlkt(5,:)),ylabel('x5(m)');
subplot(6,1,6),plot(t,xwn(:,6)'-xlkt(6,:)),ylabel('x6(rad)');
xlabel('Time Step(K)');
sgtitle('Errors between Ground Truth and LKF prediction');
```

Q6

```
    %Q6
%EKF
dt = 0.1;
L = 0.5;
t = 0:dt:100;
n = length(t);
xnom = [10;0;pi/2;-60;0;-pi/2];
%dx0 =  [0;1;0;0;0;0.1];
dx0 = [1;0;0.1;6;0.1;0.1];
unom = [2,-pi/18, 12, pi/25];
%options = odeset('RelTol', 1e-12);
xdot = @(x, u)[u(1)*cos(x(3)); u(1)*sin(x(3));u(1)*tan(u(2))/L;u(3)*cos(x(6));u(3)*sin(x
xdotwrap = @(t, x) xdot(x, [2,-pi/18, 12, pi/25]);
```

```
N = 50;
rng(100);
for j = 1:N
    xwn = [];
    ywn = [];
    p_ek = 5*Qtrue; %
    x0int = mvnrnd(x0, p_ek);
    x0ek = xnom;
    x_ekf = zeros(6,n);
    x_ekf(:,1) = x0ek;
    p_a = zeros(6,n);
    p_a(:,1) = diag(p_ek);
    sg = dt*eye(6);
    %Q_ek = 50*Qtrue; %
    Q_ek = diag([50*0.001,50*0.001,0.01,5000,500*1,10*0.01]);
    [F, G, H] = linStateMats(x0ek, unom, L, dt);
    Rkf2 = Rtrue; %
    nees_samp = zeros(N,n);
    nis_samp = zeros(N,n);
    for i = 0:dt:100
        %Noisy Ground states
        [tode, y1] = ode45(xdotwrap, [0 dt], x0int);
        xn1 = y1(end, :);
        xn1(3) = constrainAngle(xn1(3));
        xn1(6) = constrainAngle(xn1(6));
        xn2 = xn1 + mvnrnd(zeros(1,6), Qtrue);
        xwn = [xwn ;xn2];
        x0int = xn1;

        %Noisy Measurements
        yw1 = getY(xn1');
        yw1(1) = constrainAngle(yw1(1));
        yw1(3) = constrainAngle(yw1(3));
        yw1 = yw1 + transpose(mvnrnd(zeros(1,5), Rtrue));
        ywn = [ywn yw1];
    end


    for i = 1:1000
        %prediction
        [tode, ypk] = ode45(xdotwrap, [0 dt], x0ek);
        xnek = ypk(end, :);
        xnek(3) = constrainAngle(xnek(3));
        xnek(6) = constrainAngle(xnek(6));
        x_ekf(:,i+1) = xnek;
        x0ek = xnek;
        p_ek = F*p_ek*transpose(F) + sg*Q_ek*transpose(sg);

        %measurement update
        [F, G, H] = linStateMats(xnek, unom, L, dt);
        S = H*p_ek*transpose(H)+Rkf2;
        yk = getY(xnek);
```

```
        yk(1) = constrainAngle(yk(1));
        yk(3) = constrainAngle(yk(3));
        eyk = ywn(:,i) - yk;
        K = p_ek*transpose(H)*inv(H*p_ek*transpose(H)+Rkf2);
        x_ekf(:,i+1) = x_ekf(:,i+1)+K*eyk;
        p_ek = (eye(6) - K*H)*p_ek;
        p_a(:,i+1) = diag(p_ek);

        %NEES/NIS
        S = 0.5*(S+transpose(S));
        nees_samp(j,i) = transpose(transpose(xwn(i,:)) - x_ekf(:,i))*inv(p_ek)*(transpos
        nis_samp(j,i) = transpose(eyk)*inv(S)*(eyk);
    end

end


%NEES test
nees_bar = mean(nees_samp,1);
alpha_nees = 0.05;
Nnx = N*6;
r1x = chi2inv(alpha_nees/2, Nnx )./ N
r2x = chi2inv(1-alpha_nees/2, Nnx )./ N

figure(30)
plot(nees_bar,'ro','MarkerSize',6,'LineWidth',2),hold on
plot(r1x*ones(size(nees_bar)),'b--','LineWidth',2)
plot(r2x*ones(size(nees_bar)),'b--','LineWidth',2),hold off
ylabel('NEES statistic, $\bar{\epsilon}_x$','Interpreter','latex', 'FontSize',14)
xlabel('time step, k','FontSize',14)
title('NEES Estimation Results','FontSize',14)
legend('NEES @ time k', 'r_1 bound', 'r_2 bound'),grid on

%NIS Test
epsNISbar = mean(nis_samp,1);
alphaNIS = 0.05;
Nny = N*5;
%%compute intervals:
r1y = chi2inv(alphaNIS/2, Nny )./ N
r2y = chi2inv(1-alphaNIS/2, Nny )./ N
f=figure(32)
plot(epsNISbar,'bo','MarkerSize',6,'LineWidth',2),hold on
plot(r1y*ones(size(epsNISbar)),'b--','LineWidth',2)
plot(r2y*ones(size(epsNISbar)),'b--','LineWidth',2),hold off
ylabel('NIS statistic, $\bar{\epsilon}_y$','Interpreter','latex','FontSize',14)
xlabel('time step, k','FontSize',14)
title('NIS Estimation Results','FontSize',14)
legend('NIS @ time k', 'r_1 bound', 'r_2 bound'),grid on


x_err = xwn' - x_ekf
sig = sqrt(p_a)
```

```
sig2p =  + 2*sig
sig2n =  - 2*sig

%plot noisy ground truth and measurements
figure(33)
subplot(6,1,1),plot(t, xwn(:,1)),ylabel(' x1(m)');
subplot(6,1,2),plot(t, xwn(:,2)),ylabel('x2(m)');
subplot(6,1,3),plot(t, xwn(:,3)),ylabel('x3(rad)');
subplot(6,1,4),plot(t, xwn(:,4)),ylabel('x4(m)');
subplot(6,1,5),plot(t, xwn(:,5)),ylabel('x5(m)');
subplot(6,1,6),plot(t, xwn(:,6)),ylabel('x6(rad)');
xlabel('Time Step(K)');
sgtitle("Noisy Gound Truth States")

figure(34)
subplot(5,1,1),plot(t, ywn(1,:)),ylabel('y1');
subplot(5,1,2),plot(t, ywn(2,:)),ylabel('y2');
subplot(5,1,3),plot(t, ywn(3,:)),ylabel('y3');
subplot(5,1,4),plot(t, ywn(4,:)),ylabel('y4');
subplot(5,1,5),plot(t, ywn(5,:)),ylabel('y5');
xlabel('Time Step(K)');
sgtitle("Noisy Truth Measurements")

%Plot EKF state prediction for simulated Y
figure(35)
subplot(6,1,1),plot(t,x_ekf(1,:)),ylabel(' x1 (m)');
subplot(6,1,2),plot(t, x_ekf(2,:)),ylabel(' x2(m)');
subplot(6,1,3),plot(t, x_ekf(3,:)),ylabel('x3(rad)');
subplot(6,1,4),plot(t,x_ekf(4,:)),ylabel('x4(m)');
subplot(6,1,5),plot(t, x_ekf(5,:)),ylabel('x5(m)');
subplot(6,1,6),plot(t,x_ekf(6,:)),ylabel('x6(rad)');
xlabel('Time Step(K)');
sgtitle("EKF predicted states from Noisy Measurements");

%Plot Errors vs time
figure(36)
subplot(6,1,1),plot(t,x_err(1,:)),ylabel(' x1(m)')
hold on
plot(t,sig2p(1,:),'r--'),grid on
plot(t,sig2n(1,:),'r--'),grid on
legend("mu","sigma")
hold off;
subplot(6,1,2),plot(t, x_err(2,:)),ylabel(' x2(m)'),hold on
plot(t,sig2p(2,:),'r--'),grid on
plot(t,sig2n(2,:),'r--'),grid on
legend("mu","sigma")
hold off;
subplot(6,1,3),plot(t, x_err(3,:)),ylabel('x3(rad)'),hold on
plot(t,sig2p(3,:),'r--'),grid on
plot(t,sig2n(3,:),'r--'),grid on
legend("mu","sigma")
hold off;
```

```
subplot(6,1,4),plot(t,x_err(4,:)),ylabel('x4(m)'),hold on
plot(t,sig2p(4,:),'r--'),grid on
plot(t,sig2n(4,:),'r--'),grid on
legend("mu","sigma")
hold off;
subplot(6,1,5),plot(t,x_err(5,:)),ylabel(' x5(m)'),hold on
plot(t,sig2p(5,:),'r--'),grid on
plot(t,sig2n(5,:),'r--'),grid on
legend("mu","sigma")
hold off;
subplot(6,1,6),plot(t,x_err(6,:)),ylabel('x6(rad)'),hold on
plot(t,sig2p(6,:),'r--'),grid on
plot(t,sig2n(6,:),'r--'),grid on
legend("mu","sigma")
hold off;
xlabel('Time Step(K)');
sgtitle('Errors between Ground Truth and EKF prediction')
```

Q7

```
%LKF

Rkf = Rtrue; %1
%Qkf = 0.5*Qtrue; %2
Qkf = diag([500*0.001,500*0.001,0.01*0.01,500*0.01,500*0.01,0.01*0.01]);
x_all = zeros(6,n);
x_all(:,1) = dx0;
sg = dt*eye(6);
[F, G, H] = linStateMats(xmat(1,:), unom, L, dt);
[F2, G2, H2] = linStateMats(xmat(2,:), unom, L, dt);
Hall = [H;H2];
Rall = blkdiag(Rtrue,Rtrue);
xlkt = zeros(6,n);
xlkt(:,1) = transpose(xmat(1,:)) + x_all(:,1);
%p = inv(transpose(Hall)*inv(Rall)*Hall);
p = 10*Qtrue; %4
p_lkf = zeros(6,n);
p_lkf(:,1) = diag(p);
dy_lkf = zeros(5,n);

for i = 1:1000
        %prediction step
        x_all(:,i+1) = F*x_all(:,i);
        p = F*p*transpose(F) + sg*Qkf*transpose(sg);
        [F, G, H] = linStateMats(xmat(i+1,:), unom, L, dt);
        ypred = H*x_all(:,i+1);
        K = p*transpose(H)*inv(H*p*transpose(H)+Rkf);
        %measurement update
        x_all(:,i+1) = x_all(:,i+1) + K*((ydata(:,i+1)-yfmat(:,i)) - ypred);
        dy_lkf(:,i) = H*x_all(:,i+1);
        p = (eye(6) - K*H)*p;
```

```
        xlkt(:,i+1) = transpose(xmat(i+1,:)) + x_all(:,i+1);
        p_lkf(:,i+1) = diag(p);
end

y_lkf = yfmat+dy_lkf;

sig_lkf = sqrt(p_lkf)
sig2p_lkf = xlkt + 2*sig_lkf
sig2n_lkf = xlkt - 2*sig_lkf



figure(70)
subplot(6,1,1),plot(t, xlkt(1,:)),ylabel('x1(m)')
hold on
plot(t,sig2p_lkf(1,:),'r--'),grid on
plot(t,sig2n_lkf(1,:),'r--'),grid on
legend("mu","sigma")
hold off;
subplot(6,1,2),plot(t, xlkt(2,:)),ylabel('x2(m)')
hold on
plot(t,sig2p_lkf(2,:),'r--'),grid on
plot(t,sig2n_lkf(2,:),'r--'),grid on
legend("mu","sigma")
hold off;
subplot(6,1,3),plot(t, xlkt(3,:)),ylabel('x3(rad)')
hold on
plot(t,sig2p_lkf(3,:),'r--'),grid on
plot(t,sig2n_lkf(3,:),'r--'),grid on
legend("mu","sigma")
hold off;
subplot(6,1,4),plot(t, xlkt(4,:)),ylabel('x4(m)')
hold on
plot(t,sig2p_lkf(4,:),'r--'),grid on
plot(t,sig2n_lkf(4,:),'r--'),grid on
legend("mu","sigma")
hold off;
subplot(6,1,5),plot(t, xlkt(5,:)),ylabel('x5(m)')
hold on
plot(t,sig2p_lkf(5,:),'r--'),grid on
plot(t,sig2n_lkf(5,:),'r--'),grid on
legend("mu","sigma")
hold off;
subplot(6,1,6),plot(t, xlkt(6,:)),ylabel('x6(rad)')
hold on
plot(t,sig2p_lkf(6,:),'r--'),grid on
plot(t,sig2n_lkf(6,:),'r--'),grid on
legend("mu","sigma")
hold off;
xlabel('Time Step(K)');
sgtitle('States from LKF');
```

```
t1 = t(1:n-1);
figure(61)
subplot(5,1,1),plot(t1, y_lkf(1,1:n-1)),ylabel(' y1')
hold on
plot(t1,ydata(1,2:n))
legend('Lkf','truth')
hold off;
subplot(5,1,2),plot(t1, y_lkf(2,1:n-1)),ylabel('y2')
hold on
plot(t1,ydata(2,2:n))
legend('Lkf','truth')
hold off;
subplot(5,1,3),plot(t1, y_lkf(3,1:n-1)),ylabel('y3')
hold on
plot(t1,ydata(3,2:n))
legend('Lkf','truth')
hold off;
subplot(5,1,4),plot(t1, y_lkf(4,1:n-1)),ylabel('y4')
hold on
plot(t1,ydata(4,2:n))
legend('Lkf','truth')
hold off;
subplot(5,1,5),plot(t1, y_lkf(5,1:n-1)),ylabel('y5')
hold on
plot(t1,ydata(5,2:n))
legend('Lkf','truth')
hold off;
xlabel('Time Step(K)');
sgtitle("LKF Measurements VS Truth Measurements");

%Q7 with Ydata EKF

p_ek = 5*Qtrue;
x0ek = xnom;
x_ekf = zeros(6,n);
x_ekf(:,1) = x0ek;
p_a = zeros(6,n);
p_a(:,1) = diag(p_ek);
sg = dt*eye(6);
Q_ek = 10*Qtrue;
[F, G, H] = linStateMats(x0ek, unom, L, dt);
Rkf2 = Rtrue;
y_ekf = zeros(5,n);

for i = 1:1000
        %prediction
        [tode, ypk] = ode45(xdotwrap, [0 dt], x0ek);
        xnek = transpose(ypk(end, :)); %%
        xnek(3) = constrainAngle(xnek(3));
        xnek(6) = constrainAngle(xnek(6));
        x_ekf(:,i+1) = xnek;
        x0ek = xnek;
```

```
        p_ek = F*p_ek*transpose(F) + sg*Q_ek*transpose(sg);

        %measurement update
        [F, G, H] = linStateMats(xnek, unom, L, dt);
        yk = getY(xnek);
        yk(1) = constrainAngle(yk(1));
        yk(3) = constrainAngle(yk(3));
        eyk = ydata(:,i+1) - yk;
        K = p_ek*transpose(H)*inv(H*p_ek*transpose(H)+Rkf2);
        x_ekf(:,i+1) = x_ekf(:,i+1)+K*eyk; %%
        p_ek = (eye(6) - K*H)*p_ek;
        p_a(:,i+1) = diag(p_ek);
        y_ekf(:,i) = getY(x_ekf(:,i+1));
        y_ekf(1,i) = constrainAngle(y_ekf(1,i));
        y_ekf(3,i) = constrainAngle(y_ekf(3,i));

end

sig = sqrt(p_a)
sig2p = x_ekf + 2*sig
sig2n = x_ekf - 2*sig

figure(62)
subplot(6,1,1),plot(t,x_ekf(1,:)),ylabel(' x1(m)')
hold on
plot(t,sig2p(1,:),'r--'),grid on
plot(t,sig2n(1,:),'r--'),grid on
legend("mu","sigma")
hold off;
subplot(6,1,2),plot(t, x_ekf(2,:)),ylabel(' x2(m)'),hold on
plot(t,sig2p(2,:),'r--'),grid on
plot(t,sig2n(2,:),'r--'),grid on
legend("mu","sigma")
hold off;
subplot(6,1,3),plot(t, x_ekf(3,:)),ylabel(' x3(rad)'),hold on
plot(t,sig2p(3,:),'r--'),grid on
plot(t,sig2n(3,:),'r--'),grid on
legend("mu","sigma")
hold off;
subplot(6,1,4),plot(t,x_ekf(4,:)),ylabel(' x4(m)'),hold on
plot(t,sig2p(4,:),'r--'),grid on
plot(t,sig2n(4,:),'r--'),grid on
legend("mu","sigma")
hold off;
subplot(6,1,5),plot(t,x_ekf(5,:)),ylabel('x5(m)'),hold on
plot(t,sig2p(5,:),'r--'),grid on
plot(t,sig2n(5,:),'r--'),grid on
legend("mu","sigma")
hold off;
subplot(6,1,6),plot(t,x_ekf(6,:)),ylabel('x6(rad)'),hold on
plot(t,sig2p(6,:),'r--'),grid on
plot(t,sig2n(6,:),'r--'),grid on
```

```
legend("mu","sigma")
hold off;
xlabel('Time Step(K)');
sgtitle('States from EKF')


t1 = t(1:n-1);
figure(63)
subplot(5,1,1),plot(t1, y_ekf(1,1:n-1)),ylabel(' y1')
hold on
plot(t1,ydata(1,2:n))
legend('Ekf','truth')
hold off;
subplot(5,1,2),plot(t1, y_ekf(2,1:n-1)),ylabel('y2')
hold on
plot(t1,ydata(2,2:n))
legend('Ekf','truth')
hold off;
subplot(5,1,3),plot(t1, y_ekf(3,1:n-1)),ylabel('y3')
hold on
plot(t1,ydata(3,2:n))
legend('Ekf','truth')
hold off;
subplot(5,1,4),plot(t1, y_ekf(4,1:n-1)),ylabel('y4')
hold on
plot(t1,ydata(4,2:n))
legend('Ekf','truth')
hold off;
subplot(5,1,5),plot(t1, y_ekf(5,1:n-1)),ylabel('y5')
hold on
plot(t1,ydata(5,2:n))
legend('Ekf','truth')
hold off;
xlabel('Time Step(K)');
sgtitle("EKF Measurements VS Truth Measurements");
```

Functions

```
    function trp = constrainAngle(arf)
trp = arf;
while trp > pi
    trp = trp - (2*pi);
end
while trp < -pi
    trp = trp + (2*pi);
end
end

function [F, G, H] = linStateMats(x, u, L, dt)

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

F = eye(size(A)) + A*dt;
G = B*dt;

d14 = x(1) - x(4);
d41 = -d14;
d25 = x(2) - x(5);
d52 = -d25;

H = [d52/(d41^2+d52^2) -d41/(d41^2+d52^2) -1 -d52/(d41^2+d52^2)
     d41/(d41^2+d52^2) 0;
     d14/sqrt(d14^2+d25^2) d25/sqrt(d14^2+d25^2) 0 d41/sqrt(d14^2+d25^2) d52/sqrt(d14^2+
    -d25/(d14^2+d25^2) d14/(d14^2+d25^2) 0 d25/(d14^2+d25^2) -
     d14/(d14^2+d25^2) -1;
     0 0 0 1 0 0;
     0 0 0 0 1 0];

end

function y1 = getY(x)
    d52 = x(5) - x(2);
    d41 = x(4) - x(1);
    d25 = -d52;
    d14 = -d41;
    y1 = [atan2(d52,d41)-x(3);
          sqrt(d14^2+d25^2);
          atan2(d25,d14)-x(6);
          x(4);
          x(5)];
end
```

Additional Question

```
clc;
clear all;
close all;
rng(1000)
%% params
load('cooplocalization_finalproj_KFdata.mat')
```

```
dt = 0.1;
L = 0.5;
xnom = [10; 0 ; pi/2; -60; 0; -pi/2];
dx0 =  [0;1;0;0;0;0.1];
% xnom = [-60; 0; -pi/2];
% dx0 =  [0;0;0.1];
% xdot = @(x, u)[u(1)*cos(x(3));u(1)*sin(x(3));u(2)];
% xdotwrap = @(t, x) xdot(x, [12, pi/25]);
% Q = 0.1*eye(3);
% R = 0.1*eye(3);
Q = Qtrue;
R = Rtrue;
% P = 7000*eye(6);
% P = Q;

%% try 1
u = [2, pi/18, 12, pi/25];
% f = @(x)[x(1) + 1*cos(x(3))*dt; x(2)+1*sin(x(3))*dt; x(3)+1*dt];
% h = @(x)[x(1); x(3)];

%% main
f = @(x)[x(1)+u(1)*cos(x(3))*dt;
         x(2)+u(1)*sin(x(3))*dt;
         x(3)+u(1)*tan(u(2))/L*dt;
         x(4)+u(3)*cos(x(6))*dt;
         x(5)+u(3)*sin(x(6))*dt;
         x(6)+u(4)*dt];

h = @(x)[atan2(x(5) - x(2), x(4) - x(1))-x(3);
         sqrt((-(x(4) - x(1)))^2+(-(x(5) - x(2)))^2);
         atan2(-(x(5) - x(2)),-(x(4) - x(1)))-x(6);
         x(4);
         x(5)];
xold = xnom + dx0;
dt = 0.1;
xmat = [];
ymat = [];

%% simulate ground truth state and measurement vectors
for i = 0:dt:100
    xnew = f(xold);
    xnew(3) = constrainAngle(xnew(3));
    xnew(6) = constrainAngle(xnew(6));
    xnew = xnew + mvnrnd(zeros(length(f(xold)), 1), Q)';
    xold = xnew;
    xmat = [xmat xnew];
    ynew = h(xnew) + mvnrnd(zeros(1, length(h(xnew))), R)';
    ymat = [ymat ynew];
end

%% call and est using ukf
N = 10; %number of runs
```

```
for this_run = 1:N
    P = 1*eye(6);
%     P =  [40 0 0 0 0 0; 0 40 0 0 0 0; 0 0 40 0 0 0; 0 0 0 40 0 0; 0 0 0 0 40 0; 0 0 0
    x = xnom+dx0;
iter = 1;
xest = [];
for i = 0:dt:99
%     z = ymat(:, iter);
z = ydata(:, iter+1);
    [F, G, H] = linStateMats(x, u, L, dt);
    [x, P] = ukf(f, x, P, h, z, Q, R);
    S = H*P*H';
    nis_samp(this_run, iter) = (z-h(x))'*inv(S)*(z-h(x));
    iter = iter+1;
    xest = [xest x];
end
end

epsNISbar = mean(nis_samp,1);
alphaNIS = 0.05;
Nny = N*5;
%%compute intervals:
r1y = chi2inv(alphaNIS/2, Nny )./ N;
r2y = chi2inv(1-alphaNIS/2, Nny )./ N;
f=figure(1);
plot(epsNISbar,'bo','MarkerSize',1,'LineWidth',1),hold on
plot(r1y*ones(size(epsNISbar)),'b--','LineWidth',1)
plot(r2y*ones(size(epsNISbar)),'b--','LineWidth',1),hold off
ylabel('NIS statistic, $\bar{\epsilon}_y$','Interpreter','latex','FontSize',14)
xlabel('time step, k','FontSize',14)
title('NIS Estimation Results','FontSize',14)
legend('NIS @ time k', 'r_1 bound', 'r_2 bound'),grid on

%% plots
% t = 0:dt:100;
% f = figure(2);
% sgtitle('Run with simulated data','FontSize',14)
% subplot(6, 1, 1), plot(t, xmat(1,:)), hold on, plot(t, xest(1, :)),xlabel('time', 'Fon
% subplot(6, 1, 2), plot(t, xmat(2,:)), hold on, plot(t, xest(2, :)),xlabel('time','Font
% subplot(6, 1, 3), plot(t, xmat(3,:)), hold on, plot(t, xest(3, :)),xlabel('time','Font
% subplot(6, 1, 4), plot(t, xmat(4,:)), hold on, plot(t, xest(4, :)),xlabel('time','Font
% subplot(6, 1, 5), plot(t, xmat(5,:)), hold on, plot(t, xest(5, :)),xlabel('time','Font
% subplot(6, 1, 6), plot(t, xmat(6,:)), hold on, plot(t, xest(6, :)),xlabel('time','Font

t = 0:dt:99;
figure()
sgtitle('state estimation with the given data logs')
subplot(6, 1, 1), plot(t, xest(1, :)),xlabel('time', 'FontSize',14), ylabel('eststate1',
subplot(6, 1, 2), plot(t, xest(2, :)),xlabel('time', 'FontSize',14), ylabel('eststate2',
subplot(6, 1, 3),plot(t, xest(3, :)),xlabel('time', 'FontSize',14), ylabel('eststate3','
subplot(6, 1, 4), plot(t, xest(4, :)),xlabel('time', 'FontSize',14), ylabel('eststate4',
subplot(6, 1, 5), plot(t, xest(5, :)),xlabel('time', 'FontSize',14), ylabel('eststate5',
```

```
subplot(6, 1, 6), plot(t, xest(6, :)),xlabel('time', 'FontSize',14), ylabel('eststate6',


function [F, G, H] = linStateMats(x, u, L, dt)

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

F = eye(size(A)) + A*dt;
G = B*dt;

d14 = x(1) - x(4);
d41 = -d14;
d25 = x(2) - x(5);
d52 = -d25;

H = [d52/(d41^2+d52^2) -d41/(d41^2+d52^2) -1 -d52/(d41^2+d52^2) d41/(d41^2+d52^2) 0;
     d14/sqrt(d14^2+d25^2) d25/sqrt(d14^2+d25^2) 0 d41/sqrt(d14^2+d25^2) d52/sqrt(d14^2+d
     -d25/(d14^2+d25^2) d14/(d14^2+d25^2) 0 d25/(d14^2+d25^2) -d14/(d14^2+d25^2) -1;
     0 0 0 1 0 0;
     0 0 0 0 1 0];

end
function th = constrainAngle(x)
th = x;
while th > pi
    th = th - (2*pi);
end
while th < -pi
    th = th + (2*pi);
end
end

function [x,P]=ukf(fstate,x,P,hmeas,z,Q,R)

% disp('i got here')
L=numel(x);
m=numel(z);
alpha=1e-4;
kap=0;
beta=2;
```

```
lambda=alpha^2*(L+kap)-L;
c=L+lambda;
Wm=[lambda/c  0.5/c+zeros(1,2*L)];
Wc=Wm;
Wc(1)=Wm(1)+(1-alpha^2+beta);
c=sqrt(c);
X=sigmas(x,P,c);
[x1,X1,P1,X2]=ut(fstate,X,Wm,Wc,L,Q);

[z1,Z1,P2,Z2]=ut(hmeas,X1,Wm,Wc,m,R);
P12=X2*diag(Wc)*Z2';
K=P12*inv(P2);
x=x1+K*(z-z1);
P = P1-P12*inv(P2)*P12';
end
```