ASEN 5044 Fall 2018	**Project Report**
Mike Miles	**Due Date: Dec 19, 2018**
Andrew Kramer	

Part 1

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

Part a

Find the required CT Jacobians needed to obtain CT linearized model parameters. Show the key steps and variables needed to find the Jacobian matrices and state the sizes of the results. Don't use a symbolic solver.

We've elected to do the cooperative air/ground localization problem. The system is defined as follows:

$$x = [\xi_g \ \eta_g \ \theta_g \ \xi_a \ \eta_a \ \theta_a]^T$$

$$u = [v_g \ \phi_g \ v_a \ \omega_a]^T$$

$$\dot{x} = \begin{bmatrix} u_1 \cos x_3 \\ u_1 \sin x_3 \\ \frac{u_1}{L} \tan u_2 \\ u_3 \cos x_6 \\ u_3 \sin x_6 \\ u_4 \end{bmatrix}$$

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

The system Jacobians are defined as follows:

$$\frac{\partial f}{\partial x} = \begin{bmatrix} 0 & 0 & -u_1 \sin x_3 & 0 & 0 & 0 \\ 0 & 0 & u_1 \cos x_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -u_3 \sin x_6 \\ 0 & 0 & 0 & 0 & 0 & u_3 \cos x_6 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\frac{\partial f}{\partial u} = \begin{bmatrix} \cos x_3 & 0 & 0 & 0 \\ \sin x_3 & 0 & 0 & 0 \\ \frac{1}{L}\tan u_2 & \frac{u_1}{L}(\tan^2 u_2 + 1) & 0 & 0 \\ 0 & 0 & \cos x_6 & 0 \\ 0 & 0 & \sin x_6 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

$$\frac{\partial h}{\partial u} = [0]_{5\times 4}$$

The Jacobian of f with respect to x is $n \times n$ where n is the dimension of the state vector. The Jacobian of f with respect to u is $n \times m$ where m is the dimension of the control vector. The Jacobian of h with respect to x is $p \times n$ where p is the dimension of the measurement vector. The Jacobian of h with respect to u is $p \times m$.
Deriving the Jacobians $\frac{\partial f}{\partial x}$, $\frac{\partial f}{\partial u}$, and $\frac{\partial h}{\partial u}$ are straightforward. The Jacobian $\frac{\partial h}{\partial x}$ is more complicated. Fortunately there are two basic patterns with small variations. The first common pattern is exemplified by $\frac{\partial h_1}{\partial x_1}$:

$$\begin{aligned} \frac{\partial h_1}{\partial x_1} &= \frac{\partial}{\partial x_1}\tan^{-1}\left(\frac{x_5-x_2}{x_4-x_1}\right) - x_3 \\ &= \frac{\partial}{\partial x_1}\left(\frac{x_5-x_2}{x_4-x_2}\right)\frac{1}{\frac{(x_5-x_2)^2}{(x_2-x_1)^2}+1} \\ \frac{\partial}{\partial x_1}\left(\frac{x_5-x_2}{x_4-x_1}\right) &= \frac{x_5-x_2}{(x_4-x_1)^2} \\ \frac{\partial h_1}{\partial x_1} &= \frac{x_5-x_2}{(x_4-x_1)^2+(x_5-x_2)^2} \end{aligned}$$

The second pattern is exemplified by $\frac{\partial h_2}{\partial x_1}$

$$\begin{aligned}\frac{\partial h_2}{\partial x_1} &= \frac{\partial}{\partial x_1}\sqrt{(x_1-x_4)^2+(x_2-x_5)^2} \\ &= \left(\frac{\partial}{\partial x_1}(x_1-x_4)^2\right)\frac{1}{2}\left((x_1-x_4)^2+(x_2-x_5)^2\right)^{-\frac{1}{2}} \\ \frac{\partial}{\partial x_1}(x_1-x_4)^2 &= 2(x_1-x_4) \\ \frac{\partial h_2}{\partial x_1} &= \frac{x_1-x_4}{\sqrt{(x_1-x_4)^2+(x_2-x_5)^2}}\end{aligned}$$

With minor changes of variables the above three patterns can be adapted to any of the more complex terms in $\frac{\partial h}{\partial x}$.

part b

Linearize your system about its specified equilibrium/nominal operating point given in the description. Find the corresponding DT linearized model matrices (from the corresponding DT nonlinear model Jacobians) using $\Delta T = 0.1$. If possible discuss the observability, controllability, and stability properties of your time-invariant system approximation around the linearization point. If your result is time-varying result then skip this analysis.

The discrete time, linearized matrices for the initial conditions $x_0 = [10\ 0\ \pi/2\ -60\ 0\ -\pi/2]^T$ and $u_0 = [2\ -\pi/18\ 12\ \pi/25]$ are as follows:

$$F_0 = \begin{bmatrix} 1 & 0 & -0.2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1.2 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad G_0 = \begin{bmatrix} 0.0035 & -0.0412 & 0 & 0 \\ 0.1 & 0 & 0 & 0 \\ -0.0353 & 0.4124 & 0 & 0 \\ 0 & 0 & 0 & 0.06 \\ 0 & 0 & -0.1 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix}$$

$$H_0 = \begin{bmatrix} 0 & 0.0143 & -1 & 0 & -0.0143 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0.0143 & 0 & 0 & -0.0143 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \qquad M_0 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The system's nominal point changes with time, which means the point about which the system is linearized also changes. This means the linearized, discrete time system matrices are not time-invariant.

part c

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 choose) 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 and measurements using `ode45` in Matlab. Start from

the same initial conditions for the total state vector and again assuming no noise or additional control inputs. Provide suitable labeled plots to report and compare your resulting states and measurements from the linearized DT and full nonlinear DT model. Simulate at least 400 time steps.

Figures 1 and 3 show the states and measurements found by simulating the system using the linearized state transition and measurement equations with an initial state perturbation of $[0\ 1\ 0\ 0\ 0\ 0.1]^T$ of the values of the state at time zero. Figure 2 shows the perturbations of the system states at all timesteps. Figures 4 and 5 show the states and measurements found by simulating the system using `ode45` in Matlab. It can be seen from the figures that the states estimated by the linearized system track the true system state quite well. There are discrepancies between the true measurements and linearized measurements. The linearized measurements track the true measurements quite well. The code used to generate the linearized plots is included in the appendix under Exercise 1 Code.

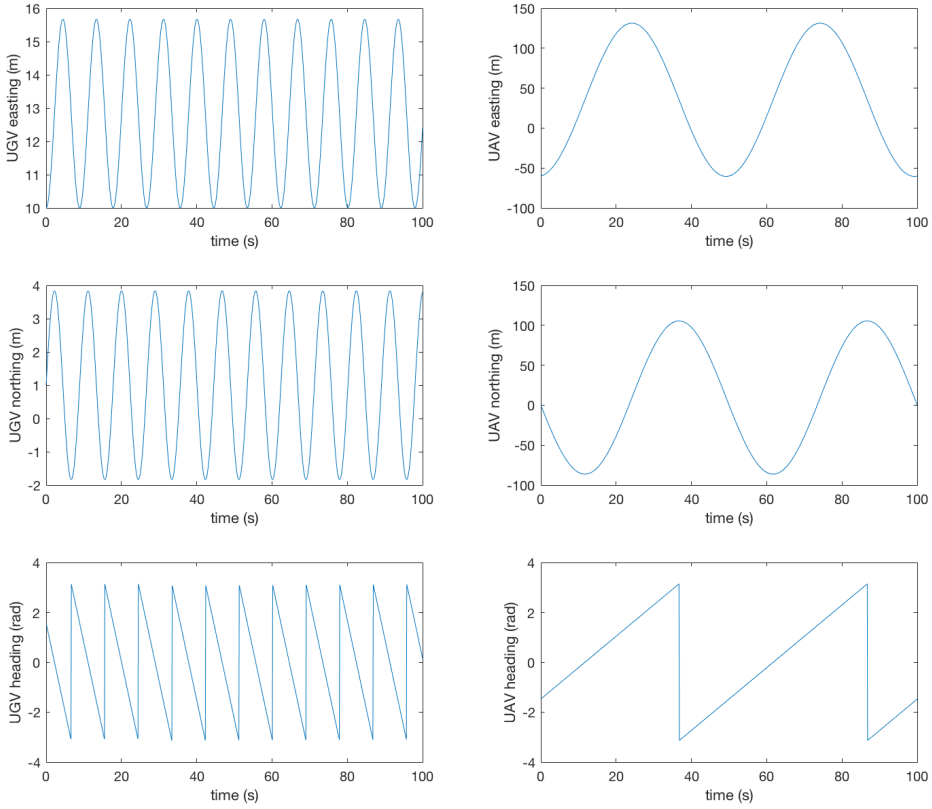

Figure 1: System states simulated by linearized system

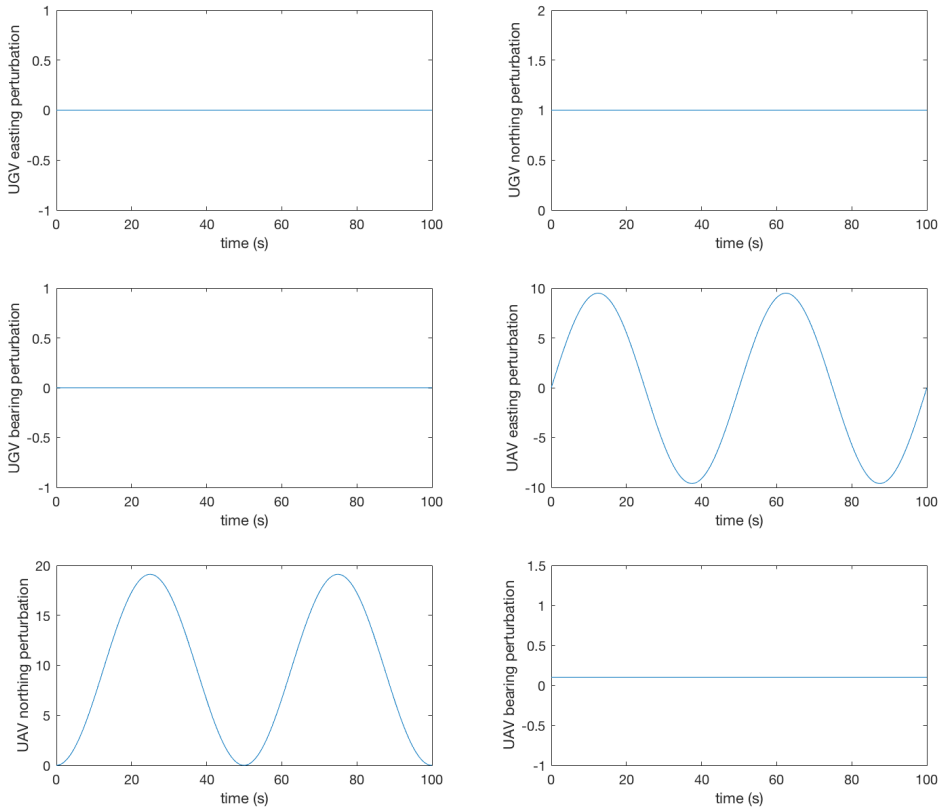

Figure 2: System state perturbations simulated by linearized system

Part 2: Nonlinear Filtering

1: Linearized Kalman Filter

Implement and tune a linearized Kalman filter using the specified nominal state trajectory, control inputs, and covariance matrix values posted on Canvas for the DT nonlinear process and measurement noise. Use NEES and NIS chi-square tests based on Monte Carlo truth model test simulations to tune and validate your linearized Kalman filter's performance. 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.
Explain how you tuned your linearized Kalman filter's process noise and provide appropriate plots/illustrations to show that your filter is working properly:

i. 'Typical' Simulation Instance

Plots for a single typical simulation instance showing the noisy simulated groundtruth states, simulated measurements, and resulting linearized Kalman filter state estimation errors.

Answer:
To assess the performance of the linearized Kalman filter, NEES/NIS tests were performed using

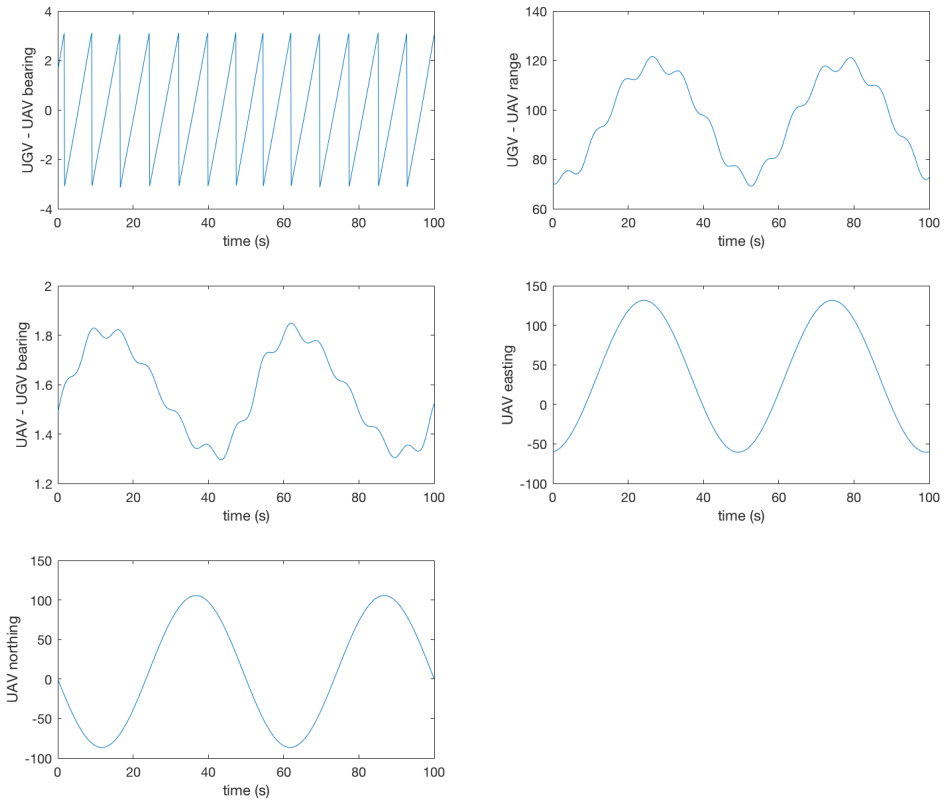

Figure 3: Measurements simulated by linearized system

a sample size of N=50 simulation runs. During each run, "ground-truth" data was simulated via ode45. Data was generated incrementally by including the ode45 solver and AWGN noise generation in a time-iterating loop so that process noise in previous states is compounded, as in reality. In the *linearized* Kalman filter, the states are estimated by using the filter to estimate the perturbation between the true and estimated states. Then the estimated state is simply:

$$\hat{x}(t) = x^*(t) + \delta\hat{x}(t)$$

where $\hat{x}$ and $\delta\hat{x}$ are the estimated state and perturbation, respectively, and x^* is the nominal state. The remainder of the linearized Kalman filter follows (see above for full matrix definitions):

Prediction

$$\delta\hat{x}_{k+1}^- = \widetilde{F}_k\delta\hat{x}_k^+ + \widetilde{G}_k\delta u_k$$

$$\mathrm{P}_{k+1}^- = \widetilde{F}_k P_k^+ \widetilde{F}_k^T + \widetilde{\Omega}_k Q_k \widetilde{\Omega}_k^T$$
$$\delta u_k = u_{k+1} - u_{k+1}^*$$

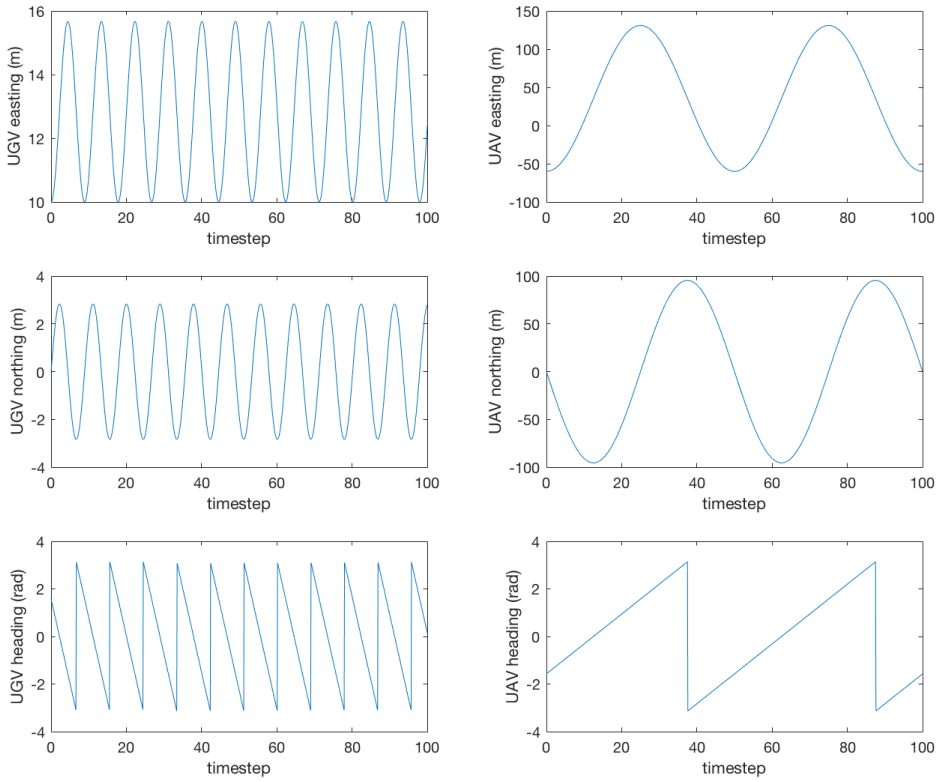

Figure 4: Nominal system states simulated by `ode45`

Correction

$$K_{k+1} = P^-_{k+1}\widetilde{H}^T_{k+1}[\widetilde{H}_{k+1}P^-_{k+1}\widetilde{H}^T_{k+1} + R_{k+1}]^{-1}$$

$$\delta y_{k+1} = y_{k+1} - y^*_{k+1} = y_{k+1} - h(x^*_{k+1})$$
$$\delta\hat{x}^+_{k+1} = \delta\hat{x}^-_{k+1} + K_{k+1}(\delta y_{k+1} - \widetilde{H}_{k+1}\delta\hat{x}^-_{k+1})$$
$$\mathrm{P}^+_{k+1} = (I - K_{k+1}\widetilde{H}_{k+1})P^-_{k+1}$$

where

$$\widetilde{F}_k|_{nom[k]} = I + \Delta t \cdot \frac{\partial f}{\partial x}|_{(x^*,u^*,t=t_k)},\ \widetilde{G}_k|_{nom[k]} = \Delta t \cdot \frac{\partial f}{\partial u}|_{(x^*,u^*,t=t_k)},$$

$\widetilde{\Omega}|_{nom[k]} = \Delta t \cdot \Gamma(t)|_{t=t_k}$, and $\widetilde{H}_{k+1}|_{nom[k+1]} = \frac{\partial h}{\partial x}|_{(x^*,u^*,t=t_{k+1})}$

Figure 6 shows the estimated and groundtruth states and associated errors for the UGV. Figure 7 shows estimated and groundtruth states and associated errors for the UAV. One can see the LKF does a fairly good job of estimating the states for the UAV, but the LKF does not do well when estimating the UGV states.

ii. NEES Test

Plots of the NEES test statistic points from all Monte Carlo simulations vs. time, comparing the resulting averages to computed upper and lower bounds for the NEES χ^2 test.

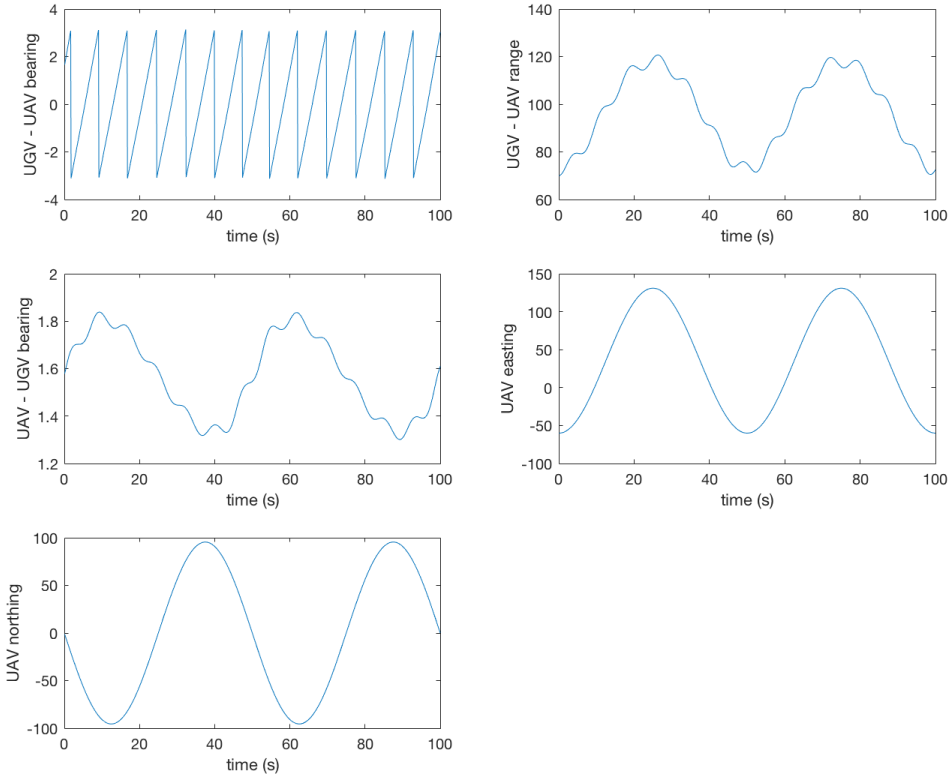

Figure 5: Measurements from system simulated by `ode45`

Answer:
NEES tests were performed for N=50 and calculated as follows:

$$e_{x,k} = x_k - \hat{x}_k^+ \sim N(0, P_k^+)$$

$$\epsilon_{x,k} = e_{x,k}^T (P_k^+)^{-1} e_{x,k} \sim \chi_n^2$$

$$\bar{\epsilon}_{x,k} = \frac{1}{N} \Sigma_{i=1}^N \epsilon_{x,k}^i$$

The resulting NEES plot is shown in figure 8.

Despite decent performance of the linearized filter to track the highly nonlinear dynamics, particularly in the case of the aerial vehicle states, the dynamics seem too nonlinear to resolve the major errors captured by the NEES and NIS scores, resulting in mean NEES scores which are much higher than the r significance bounds, as in Figure 8. To tune the filter we started with the R_{KF} matrix set equal to R_{true}. We tried different power of ten multiples of identity for the Q_{KF} matrix. After we found the best magnitude for the Q_{KF} matrix based on the NEES, NIS, and estimation

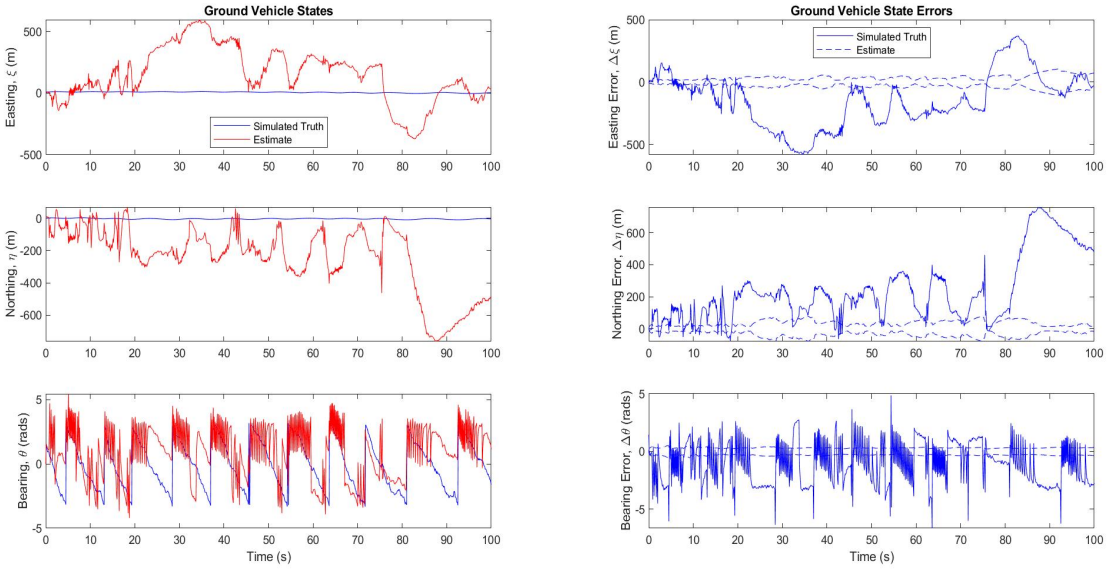

Figure 6: UGV Simulated Ground-truth, Estimated States, and State Errors via Linearized Kalman Filter

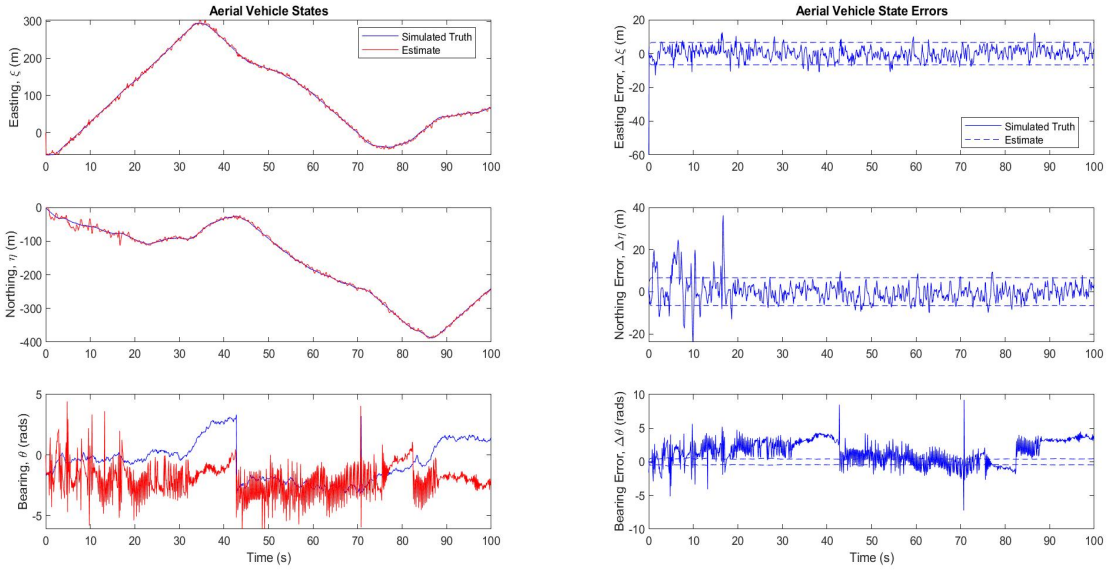

Figure 7: UAV Simulated Ground-truth, Estimated States, and State Errors via Linearized Kalman Filter

errors, we adjusted the diagonal elements of the Q_{KF} matrix to fine tune the linearized KF's performance. Confusingly, we found the best filter performance in terms of estimation errors did not correspond to the best performance in NEES scores. It could be the errors in state estimation prevent a good NEES score.

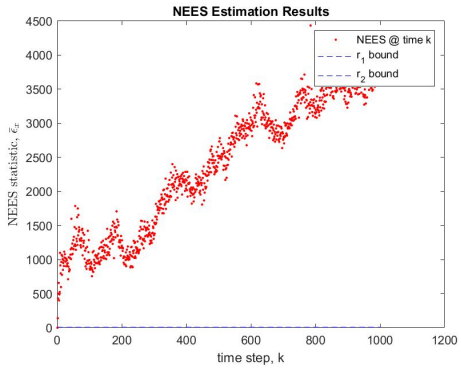

Figure 8: NEES ($\bar{\epsilon}_{x,k}$) for Linearized Kalman Filter

iii. NIS Test

Plots of the NIS test statistic points for all Monte Carlo simulations vs. time, comparing the resulting averages to computed upper and lower bounds for the NIS χ^2 test.

Answer:
Similarly, NIS tests were performed for N=50 and calculated as follows:

$$e_{y,k} = y_k - \hat{y}_k^+ \sim N(0, S_k)$$

$$\epsilon_{y,k} = e_{y,k}^T (S_k)^{-1} e_{y,k} \sim \chi_p^2$$

$$\bar{\epsilon}_{y,k} = \frac{1}{N} \Sigma_{i=1}^{N} \epsilon_{y,k}^i$$

where $S_k = H P_k^- H^T + R$.

Again, the best performance in terms of estimation errors was not obtained with the same tuning as the best NIS score. The NIS scores plotted in figure 9 generally stay within the bounds calculated using `chi2inv`, but they don't stay within the bounds 95% of the time as would be expected with a linear system. It could be these errors are caused by linearization errors, compounded by bad linearization points calculated from the linearized system. This could cause the NIS scores to deviate more from their expected values.

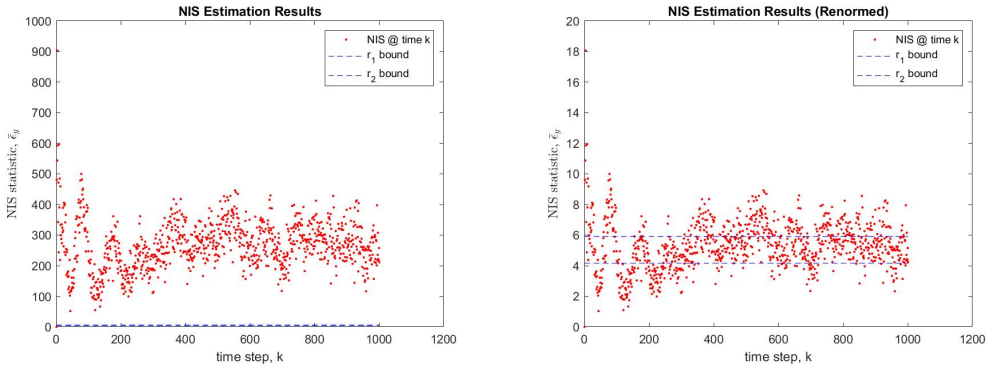

Figure 9: NIS, $\bar{\epsilon}_{y,k}$, (left) & Renormalized NIS, $\frac{\bar{\epsilon}_{y,k}}{N}$, (right) for Linearized Kalman Filter

2: Extended Kalman Filter

Implement and tune an extended Kalman filter using the specified nominal state trajectory along with the control input values and covariance matrix values posted in Canvas for the DT nonlinear process noise and measurement noise for your selected system. Use NEES and NIS χ^2 tests based on Monte Carlo truth model test 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 some justification for your choices).
Explain how you tuned your EKF's guess of the process noise and provide appropriate plots to show that your filter is working properly.

i

Figure 10 shows the estimated states, groundtruth states, and 2σ error bounds for the UGV states over a typical Monte Carlo simulation. Figure 11 shows the same for the UAV states. The UGV and UAV states are plotted separately for readability. Note that for the estimated bearing for both vehicles, the estimate tends to oscillate when the groundtruth switches from $-\pi$ to π or vice-versa. This is because the bearing is constrained to the range $[-\pi, \pi]$, and when the estimate is near the ends of this range, small errors can make it wrap to the other end of the range. Figure 12 shows the logged measurements for a typical Monte Carlo simulation. Finally, figure 13 shows state estimate errors plotted with 2σ bounds. Again, small errors in the bearing estimate when the groundtruth is near the ends of the range can cause the errors in these states to appear much larger than they actually are.
The EKF was tuned per the tuning procedure presented an slides 12 and 13 of lecture 29: R_{KF} was set to R_{true} and the Q_{KF} matrix was tuned until the state estimate errors stayed just within 2σ bounds for a typical Monte Carlo simulation and the distribution of NIS scores was fairly tight and consistent. The `Q` matrix was tuned by first multiplying the `Qtrue` matrix by successive powers of ten, then fine-tuning the values on the diagonal of the `Q` matrix.

ii

Figure 14 shows NEES test statistic points averaged over 100 Monte Carlo simulations vs. time. 100 simulations were done because there was no subjective difference between an average over 50 simulations and an average over 100 simulations. This led us to believe 100 simulations would be sufficient. An α value of 0.05 was chosen to provide the widest acceptable range of NEES scores. Figure 14 shows the NEES statistic falls within the expected range for the first 5 seconds, but it rapidly diverges after that. Interestingly, the NEES statistic seems to fall within several distinct strata. The reasons for this are unclear, however. It is possible the NEES statistic is χ^2 distributed, but the number of degrees of freedom is dependent on the system's linearization point. This would imply the linearization errors occur in discrete modes depending on the linearization point.

iii

Figure 15 shows NIS test statistic points averaged over 100 Monte Carlo simulations vs. time. Figure 15 shows the NIS statistic is consistent over all timesteps, but it does not fall within the expected range. It appears the measurement errors are χ^2 distributed with 3 degrees of freedom, rather than the expected 5 degrees of freedom. In this case linearization errors in propagating the measurement covariance are probably distorting the expected NIS score.

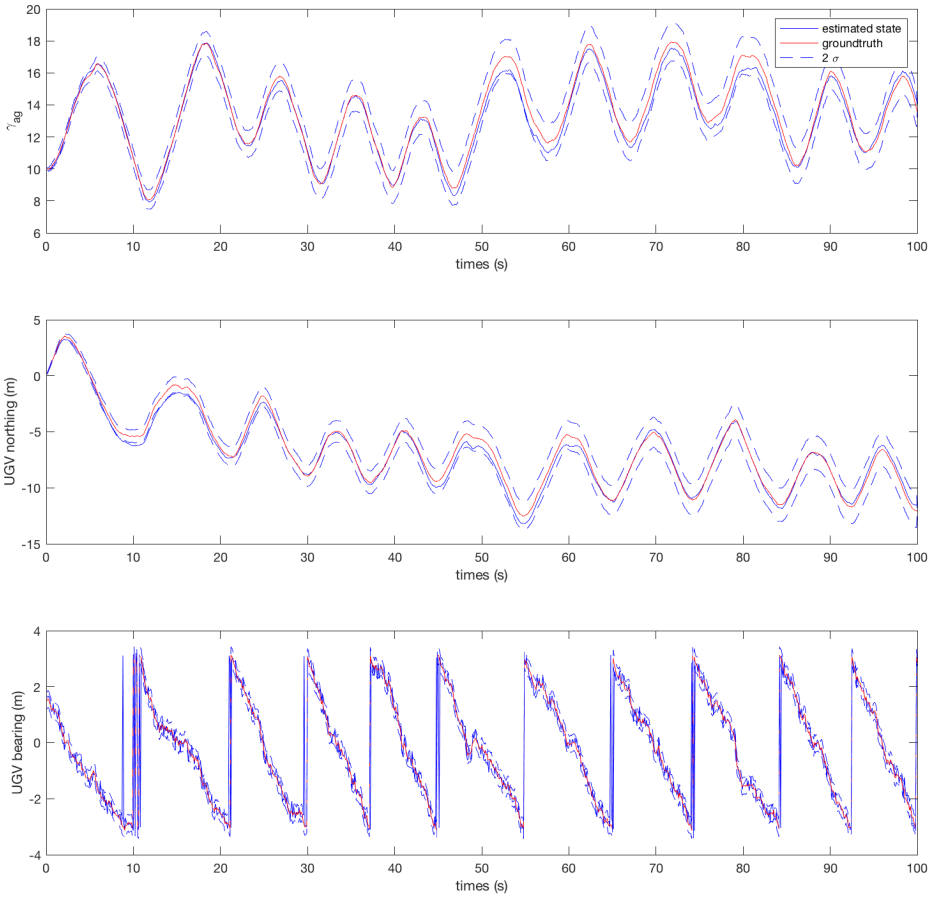

Figure 10: groundtruth and EKF estimated states for the UGV

3

Implement your linearized and extended Kalman filters 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 bounds. Compare your results. Does the linearized or extended Kalman filter perform better and why?

Linearized Kalman Filter

Figures 16 and 17 show the estimated states for the UGV and UAV, respectively, when the linearized Kalman filter is running with logged measurements.

Extended Kalman Filter

Figure 18 shows the estimated states for the UGV when the extended Kalman filter is running with logged measurements. Figure 19 shows the same for the UAV.

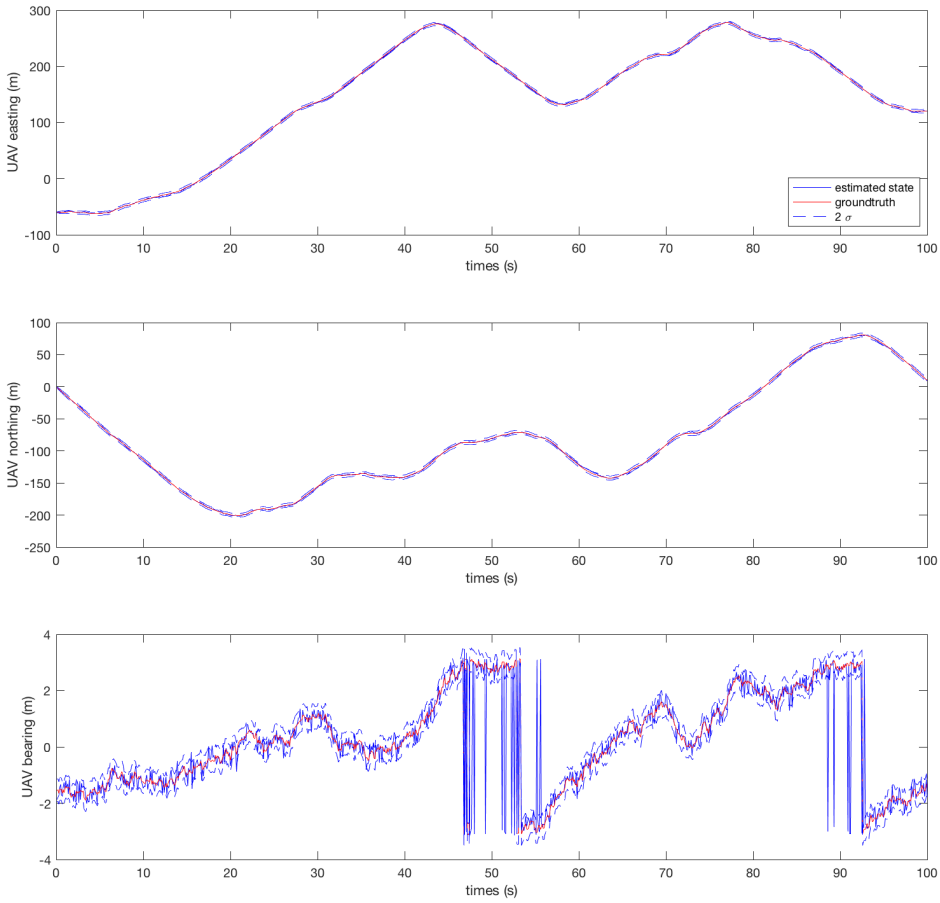

Figure 11: groundtruth EKF and estimated states for the UAV

Comparison

The EKF performs much better than the LKF in terms of estimation accuracy and NEES and NIS scores. The LKF does a reasonably good job of estimating the UAV states but it's estimation of the UGV's states pretty much unusable. This could be because the Euler approximation to the state propagation function does a poor job of propagating the state perturbation *and* consequently provides a poor point about which to linearize and propagate the state covariance. The same could apply to the measurement perturbation and measurement covariance. Meanwhile, the EKF does a very good job of estimating all estimates at most times. This is not surprising because the state propagation function and measurement function are more accurate. This not only provides better estimates in the prediction step, it also provides a better linearization point for propagating the covariances.

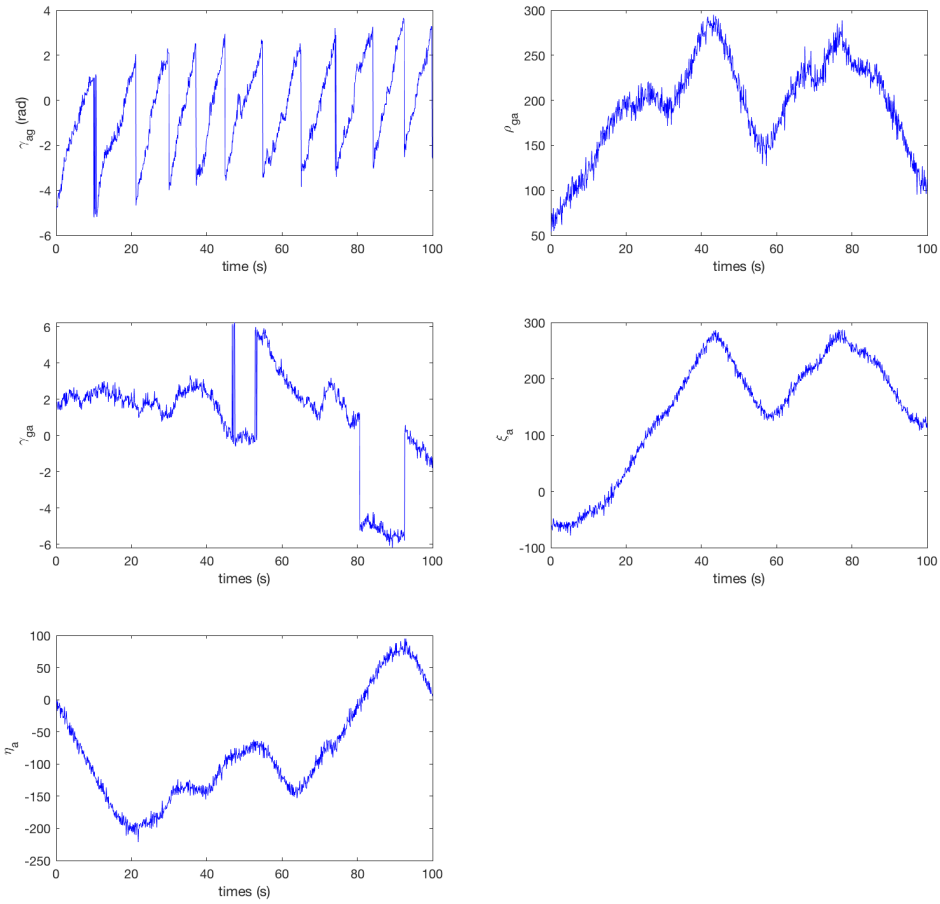

Figure 12: measurements for the EKF

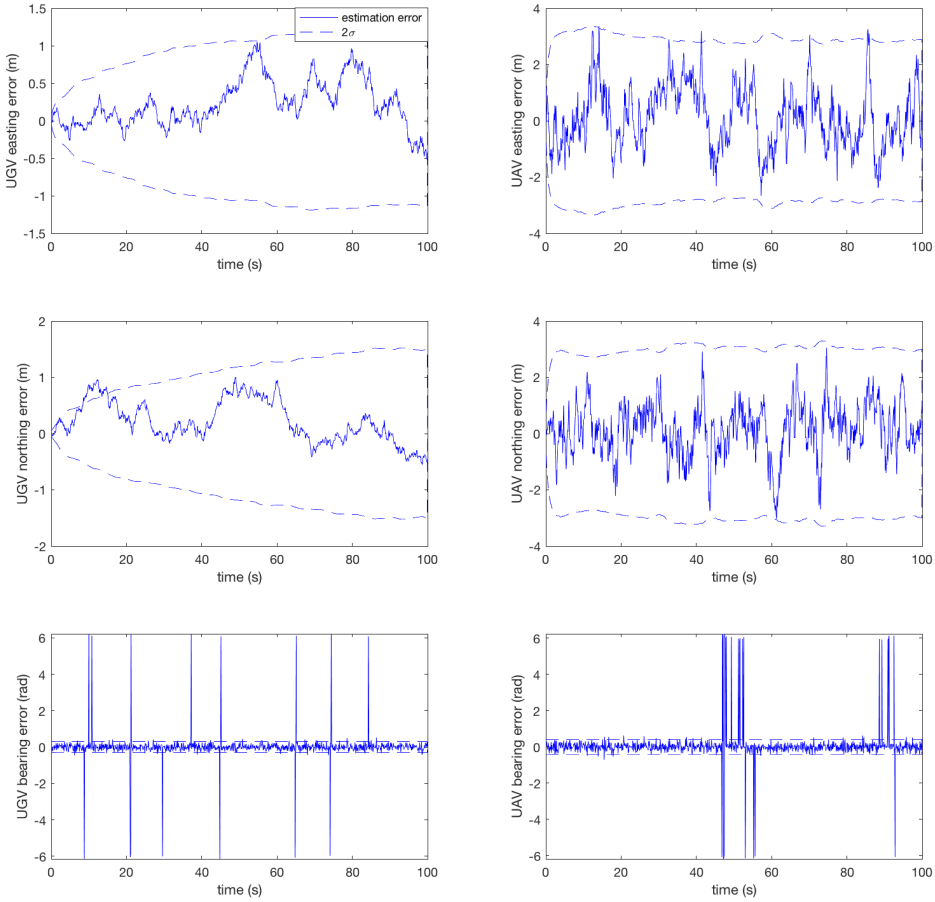

Figure 13: EKF state estimation errors for all states

Figure 14: NEES statistic averaged over 100 Monte Carlo simulations

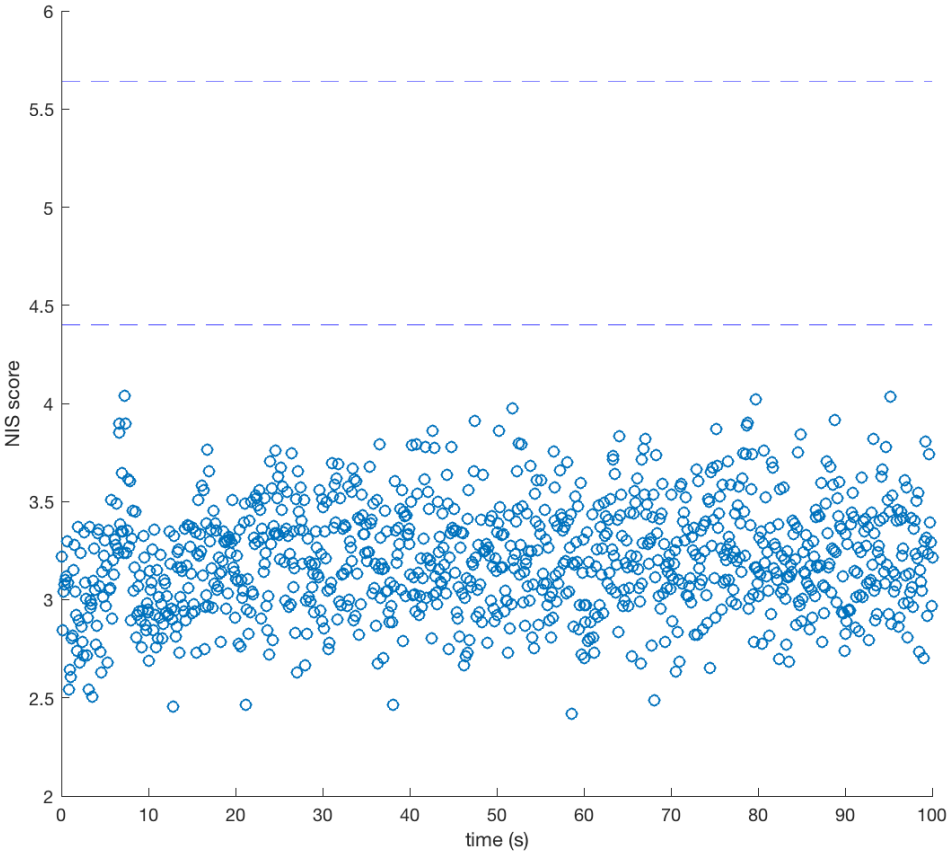

Figure 15: NIS statistic averaged over 100 Monte Carlo simulations

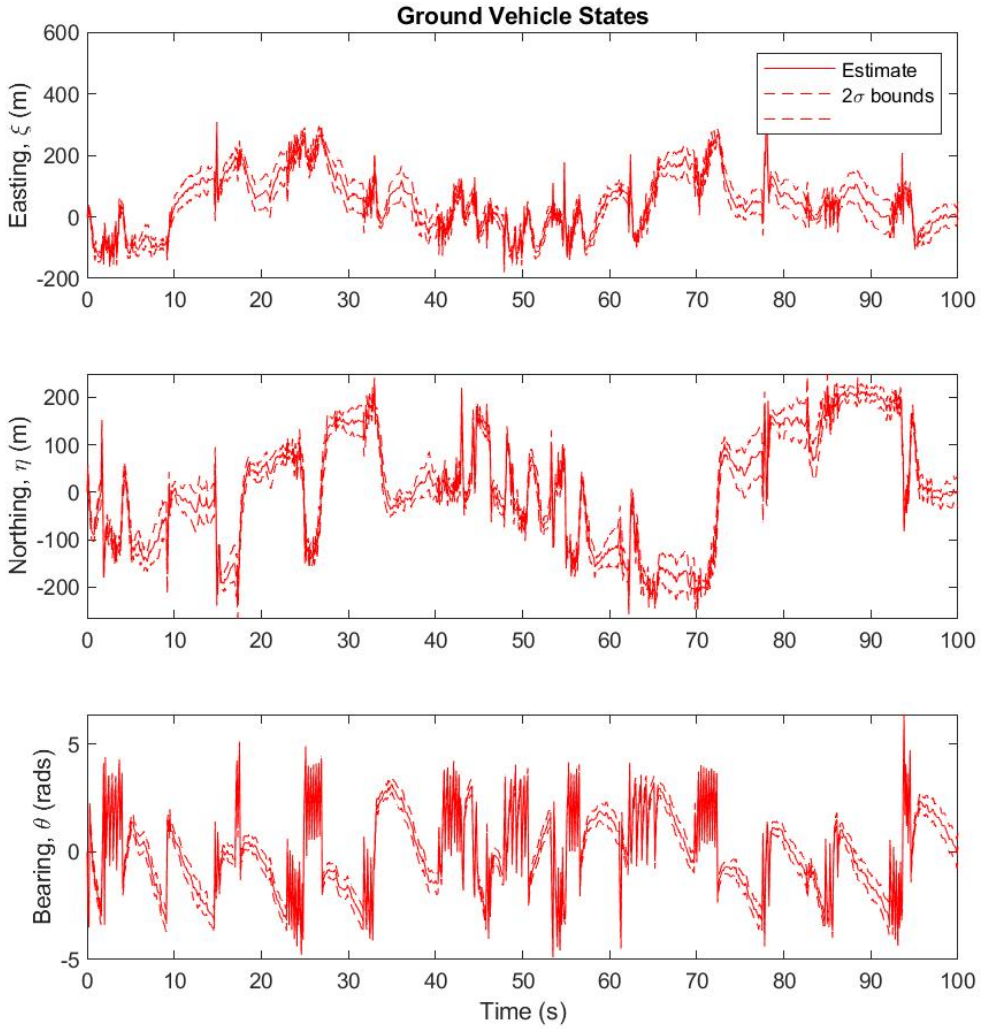

Figure 16: UGV Estimated States with logged measurement via Linearized KF

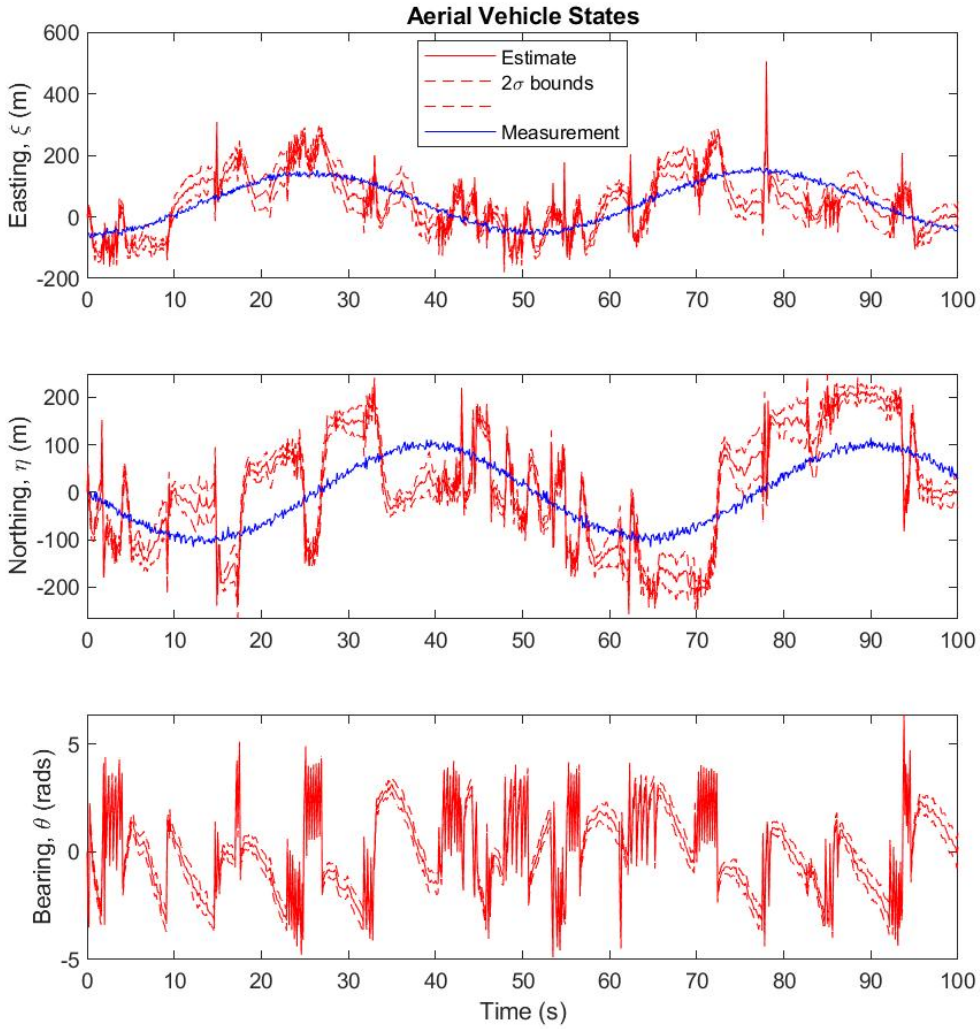

Figure 17: UAV Estimated States with logged measurement via Linearized KF

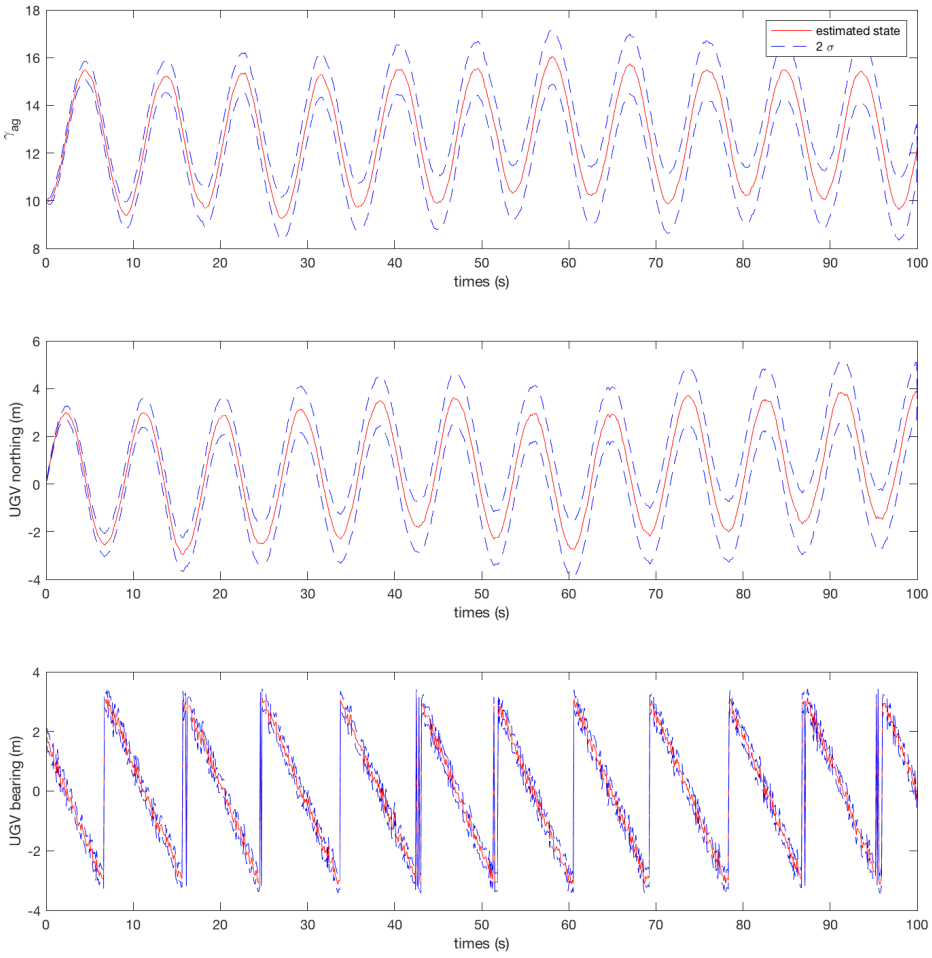

Figure 18: EKF estimated states and error bounds for UGV with logged measurements

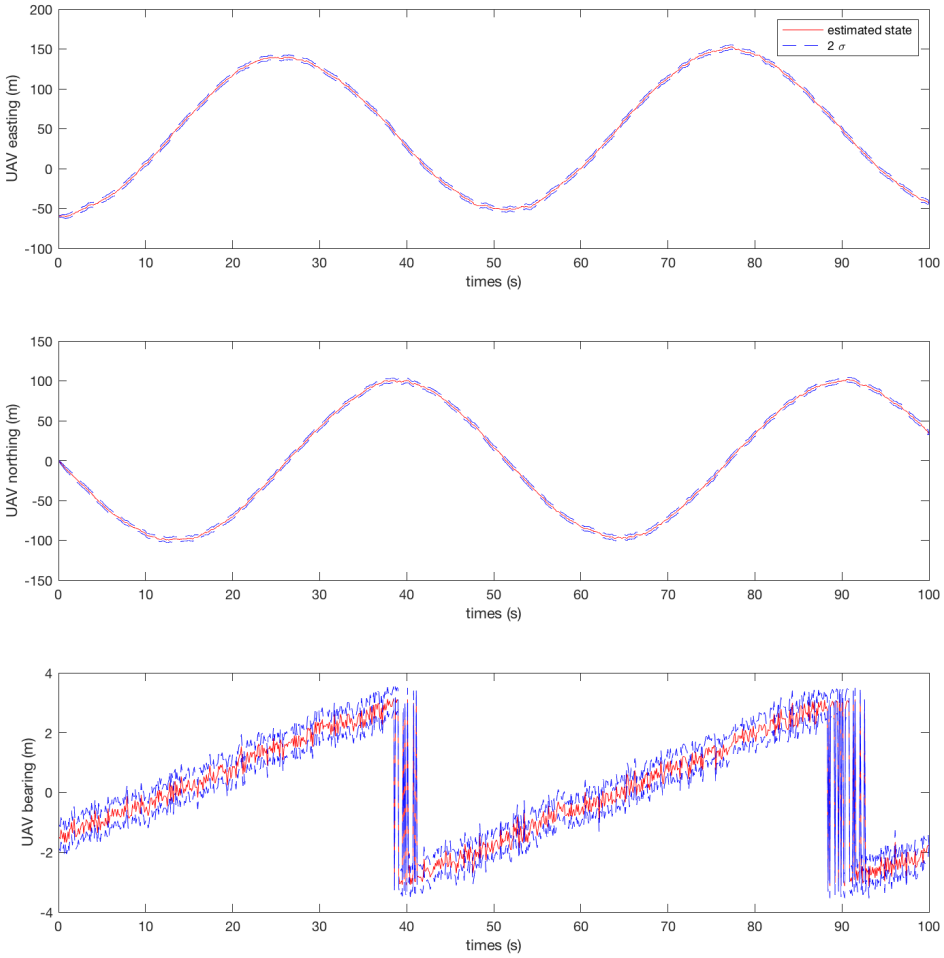

Figure 19: EKF estimated states and error bounds for UAV with logged measurements

Part 1 Code

Listing 1: Calculate And Plot Linearized States

```
x0 = [10; 0; pi/2; -60; 0; -pi/2];
x_tilde = [0;1;0;0;0;0.1];
u = [2; -pi/18; 12; pi/25];
x = x0+x_tilde;
y = [];
t = [0:0.1:100];


for i = 1:1000

    [F,G] = getLinStateMats(xnom(:,i+1),u,0.5,0.1);
    H = getLinHMat(xnom(:,i+1));

    new_x_tilde = F*x_tilde(:,i);
    x_tilde(3) = constrainAngle(x_tilde(3));
    x_tilde(6) = constrainAngle(x_tilde(6));
    x_tilde = [x_tilde new_x_tilde];

    new_x = xnom(:,i+1) + new_x_tilde;
    x = [x new_x];

    y_pret = H*x_tilde(:,i);
    y_nom = getMeas(xnom(:,i+1));
    new_y = y_nom + y_pret;
    new_y(1) = constrainAngle(new_y(1));
    new_y(3) = constrainAngle(new_y(3));

    y = [y new_y];
end


figure;
subplot(3,2,1);
plot(t,x(1,:));
xlabel('time (s)');
ylabel('UGV easting (m)');
subplot(3,2,2);
plot(t,x(4,:));
xlabel('time (s)');
ylabel('UAV easting (m)');
subplot(3,2,3);
plot(t,x(2,:));
xlabel('time (s)');
ylabel('UGV northing (m)');
```

```
subplot(3,2,4);
plot(t,x(5,:));
xlabel('time (s)');
ylabel('UAV northing (m)');
subplot(3,2,5);
plot(t,x(3,:));
xlabel('time (s)');
ylabel('UGV heading (rad)');
subplot(3,2,6);
plot(t,x(6,:));
xlabel('time (s)');
ylabel('UAV heading (rad)');

figure;
subplot(3,2,1);
plot(t(2:end), y(1,:));
xlabel('time (s)');
ylabel('UGV - UAV bearing');
subplot(3,2,2);
plot(t(2:end), y(2,:));
xlabel('time (s)');
ylabel('UGV - UAV range');
subplot(3,2,3);
plot(t(2:end), y(3,:));
xlabel('time (s)');
ylabel('UAV - UGV bearing');
subplot(3,2,4);
plot(t(2:end), y(4,:));
xlabel('time (s)');
ylabel('UAV easting');
subplot(3,2,5);
plot(t(2:end), y(5,:));
xlabel('time (s)');
ylabel('UAV northing');

figure;
subplot(3,2,1);
plot(t,x_tilde(1,:));
xlabel('time (s)');
ylabel('UGV easting perturbation');
subplot(3,2,2);
plot(t,x_tilde(2,:));
xlabel('time (s)');
ylabel('UGV northing perturbation');
subplot(3,2,3);
plot(t,x_tilde(3,:));
xlabel('time (s)');
ylabel('UGV bearing perturbation');
```

```
subplot(3,2,4);
plot(t,x_tilde(4,:));
xlabel('time (s)');
ylabel('UAV easting perturbation');
subplot(3,2,5);
plot(t,x_tilde(5,:));
xlabel('time (s)');
ylabel('UAV northing perturbation');
subplot(3,2,6);
plot(t,x_tilde(6,:));
xlabel('time (s)');
ylabel('UAV bearing perturbation');
```

Listing 2: Calculate Linearized State Matrices At Given Point

```
function [F,G] = getLinStateMats(x,u,L,deltaT)
    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;
         (1/L)*tan(u(2)) (u(1)/L)*(tan(u(2))^2+1) 0 0;
         0 0 cos(x(6)) 0;
         0 0 sin(x(6)) 0;
         0 0 0         1];

    F = eye(size(A)) + deltaT * A;
    G = deltaT * B;
end
```

Listing 3: Calculate Linearized H Matrix At Given Point

```
function H = getLinHMat(x)
    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+d25^2) 0;
        -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
```

Part 2 Code

Listing 4: Get Result Of Single Linearize KF Run NEES/NIS

```
clear all; close all; %clc;
rng(100)

addpath('../ASEN-5044/project'); % load fcns

global n; n = 6; % number of states
global m; m = 4; % number of inputs
global p; p = 5; % number of measurement

load('cooplocalization_finalproj_KFdata');
T = size(ydata,2);
global dt; dt = tvec(2)-tvec(1);
global L; L = 0.5;
x0 = [10, 0, pi/2, -60, 0, -pi/2]';
P0 = 5*Qtrue;
u0 = [2, -pi/18, 12, pi/25]';

% process noise covar (tune)
Qkf = diag([5000 5000 1 500 500 10]);

Rkf = Rtrue;
N = 50;
NEESsamples = zeros(N,T);
NISsamples = zeros(N,T);
for this_run=1:N
    xtrue = zeros(n,T);
    ytrue = zeros(p,T);
    xk = mvnrnd(x0',P0)';
    xtrue_plot = zeros(size(xtrue));
%     [~,xnom] = ode45(@(t,x) nonlinearF(t,x,u0),tvec,x0); xnom = xnom
    ';
    for ii=1:length(tvec)
        wk = mvnrnd(zeros(n,1)',Qtrue)';
        [~,xnom] = ode45(@(t,x) nonlinearF(t,x,u0),[0 dt],xk); xnom =
            xnom';
        xkp1 = xnom(:,end) + wk;
        xtrue_plot(:,ii) = xkp1;
        xkp1(3) = constrainAngle(xkp1(3));
        xkp1(6) = constrainAngle(xkp1(6));
        vkp1 = mvnrnd(zeros(p,1)',Rtrue)';
        ytrue(:,ii) = getMeas(xkp1) + vkp1;

        xtrue(:,ii) = xkp1;
        xk = xkp1;
```

```
    end

    Pk_post = P0;
    dxk_post = mvnrnd(zeros(n,1)',Pk_post)';
    xk_post = x0 + dxk_post;
    x_kf = zeros(n,T); x_kf(:,1) = xk_post;
    x_kf_plot = zeros(size(x_kf));

    for ii=2:T
        % calc matrices
        [F,G,H,M] = getLinearizedMatrices(xk_post,u0,L,dt);
        Omega = getOmega();

        % calc prior measurement from dynamic propagation
        dxkp1_prior = F*dxk_post;% + G*du;
        xkp1_prior = xk_post + dxkp1_prior;
        Pkp1_prior = F*Pk_post*F' + Omega*Qkf*Omega';
%         du = ?

        % calc Kalman gain, K, and use measurement to correct estimate
        K = Pkp1_prior*H'*inv(H*Pkp1_prior*H' + Rkf);
        dy = ytrue(:,ii)-getMeas(xkp1_prior);
        dxkp1_post = dxkp1_prior + K*(dy-H*dxkp1_prior);
        xkp1_post = xk_post + dxkp1_post;
        x_kf_plot(:,ii) = xkp1_post;
        xkp1_post(3) = constrainAngle(xkp1_post(3));
        xkp1_post(6) = constrainAngle(xkp1_post(6));
        Pkp1_post = (eye(n) - K*H)*Pkp1_prior;

        % calc NEES/NIS
        Pykp1 = H*Pkp1_prior*H' + Rkf;
        Pykp1 = 0.5*(Pykp1 + Pykp1');
        NEESsamples(this_run,ii) = (xtrue(:,ii) - xkp1_post)'*inv(
            Pkp1_post)*(xtrue(:,ii) - xkp1_post);
        NISsamples(this_run,ii) = (ytrue(:,ii) - getMeas(xkp1_post))'*
            inv(Pykp1)*(ytrue(:,ii) - getMeas(xkp1_post));

        if( sum(isnan(xkp1_post))>0 )
            disp("xkp1_post is NaN");
            return
        end

        % store and iterate data
        x_kf(:,ii) = xkp1_post;
        P_kf(:,:,ii) = Pkp1_post;
        dxk_post = dxkp1_post;
        xk_post = xkp1_post;
        Pk_post = Pkp1_post;
```

```
end


if this_run==1
% Plot States
    figure
    for ii=1:3
        subplot(3,1,ii)
        plot(tvec,xtrue_plot(ii,:),'b-'), hold on
        plot(tvec,x_kf_plot(ii,:),'r-')
        switch ii
            case 1
                title('Ground Vehicle States');
                ylabel('Easting, \xi (m)');
                legend('Simulated Truth','Estimate','Location','
                    best')
            case 2
                ylabel('Northing, \eta (m)');
            case 3
                xlabel('Time (s)');
                ylabel('Bearing, \theta (rads)');
        end
    end
    set(gcf,'units','normalized','outerposition',[0 0 .5 1])
    saveas(gcf,'linKF_GVStates.jpg')

    figure
    for ii=1:3
        subplot(3,1,ii)
        plot(tvec,xtrue_plot(ii+3,:),'b-'), hold on
        plot(tvec,x_kf_plot(ii+3,:),'r-')
        switch ii
            case 1
                title('Aerial Vehicle States');
                ylabel('Easting, \xi (m)');
                legend('Simulated Truth','Estimate','Location','
                    best')
            case 2
                ylabel('Northing, \eta (m)');
            case 3
                xlabel('Time (s)');
                ylabel('Bearing, \theta (rads)');
        end
    end
    set(gcf,'units','normalized','outerposition',[0 0 .5 1])
    saveas(gcf,'linKF_AVStates.jpg')

    figure
```

```
        for ii=1:3
            subplot(3,1,ii)
            plot(tvec,xtrue_plot(ii,:)-x_kf_plot(ii,:),'b-'), hold on
            plot(tvec,2*sqrt(squeeze(P_kf(ii,ii,:))),'b--')
            plot(tvec,-2*sqrt(squeeze(P_kf(ii,ii,:))),'b--')
            switch ii
                case 1
                    title('Ground Vehicle State Errors');
                    ylabel('Easting Error, \Delta\xi (m)');
                    legend('Simulated Truth','Estimate','Location','
                        best')
                case 2
                    ylabel('Northing Error, \Delta\eta (m)');
                case 3
                    xlabel('Time (s)');
                    ylabel('Bearing Error, \Delta\theta (rads)');
            end
        end
        set(gcf,'units','normalized','outerposition',[0.5 0 .5 1])
        saveas(gcf,'linKF_GVStateErrors.jpg')

        figure
        for ii=1:3
            subplot(3,1,ii)
            plot(tvec,xtrue_plot(ii+3,:)-x_kf_plot(ii+3,:),'b-'), hold
                on
            plot(tvec,2*sqrt(squeeze(P_kf(ii+3,ii+3,:))),'b--')
            plot(tvec,-2*sqrt(squeeze(P_kf(ii+3,ii+3,:))),'b--')
            switch ii
                case 1
                    title('Aerial Vehicle State Errors');
                    ylabel('Easting Error, \Delta\xi (m)');
                    legend('Simulated Truth','Estimate','Location','
                        best')
                case 2
                    ylabel('Northing Error, \Delta\eta (m)');
                case 3
                    xlabel('Time (s)');
                    ylabel('Bearing Error, \Delta\theta (rads)');
            end
        end
        set(gcf,'units','normalized','outerposition',[0.5 0 .5 1])
        saveas(gcf,'linKF_AVStateErrors.jpg')
    end
end

%% NEES/NIS
% NEES test
```

```
NEESmean = mean(NEESsamples,1);
NEEStotalerr = sum(NEESmean)
NEESalpha = 0.05; % significance level
% compute intervals:
NEESr1 = chi2inv(NEESalpha/2, N*n)./ N;
NEESr2 = chi2inv(1-NEESalpha/2, N*n)./ N;
NEESscore = sum(NEESmean>NEESr1 & NEESmean<NEESr2)/T;

figure
plot(NEESmean,'r.'),hold on
plot(NEESr1*ones(size(NEESmean)),'b--')
plot(NEESr2*ones(size(NEESmean)),'b--')
ylabel('NEES statistic, $\bar{\epsilon}_x$','Interpreter','latex')
xlabel('time step, k')
title('NEES Estimation Results')
legend('NEES @ time k', 'r_1 bound', 'r_2 bound')
saveas(gcf,'linKF_NEES.jpg')

figure
plot(NEESmean./N,'r.'),hold on
plot(NEESr1*ones(size(NEESmean)),'b--')
plot(NEESr2*ones(size(NEESmean)),'b--')
ylabel('NEES statistic, $\bar{\epsilon}_y$','Interpreter','latex')
xlabel('time step, k')
title('NEES Estimation Results (Renormed)')
legend('NEES @ time k', 'r_1 bound', 'r_2 bound')
saveas(gcf,'linKF_NEES_renormed.jpg')

% NIS test
NISmean = mean(NISsamples,1);
NISalpha = 0.05; % significance level
NIStotalerr = sum(NISmean)
% compute intervals:
NISr1 = chi2inv(NISalpha/2, N*p)./ N;
NISr2 = chi2inv(1-NISalpha/2, N*p)./ N;
NISscore = sum(NISmean>NISr1 & NISmean<NISr2)/T;

figure
plot(NISmean,'r.'),hold on
plot(NISr1*ones(size(NISmean)),'b--')
plot(NISr2*ones(size(NISmean)),'b--')
ylabel('NIS statistic, $\bar{\epsilon}_y$','Interpreter','latex')
xlabel('time step, k')
title('NIS Estimation Results')
legend('NIS @ time k', 'r_1 bound', 'r_2 bound')
saveas(gcf,'linKF_NIS.jpg')

figure
```

```
plot(NISmean./N,'r.'),hold on
plot(NISr1*ones(size(NISmean)),'b--')
plot(NISr2*ones(size(NISmean)),'b--')
ylabel('NIS statistic, $\bar{\epsilon}_y$','Interpreter','latex')
xlabel('time step, k')
title('NIS Estimation Results (Renormed)')
legend('NIS @ time k', 'r_1 bound', 'r_2 bound')
saveas(gcf,'linKF_NIS_renormed.jpg')
```

Part 3 Code

Listing 5: Get Result Of Single EKF Run

```
function [P_est,x_est,S_log,ey_log,x_gt,y_log] = EKF()
    load("KFdata.mat");

    Q = [0.00035  0       0      0       0       0;
         0       0.00035  0      0       0       0;
         0       0        5.0    0       0       0;
         0       0        0      0.15    0       0;
         0       0        0      0       0.15    0;
         0       0        0      0       0       10.0];

    R = Rtrue;

    % estimated x0
    x0 = [10; 0; pi/2; -60; 0; -pi/2];

    % initial position covariance
    P_0 = [0.0005 0    0    0    0    0;
           0   0.0005 0    0    0    0;
           0   0    0.005   0    0    0;
           0   0    0    0.0005 0    0;
           0   0    0    0    0.0005 0;
           0   0    0    0    0    0.005]; % initial covariance, need to
               tune

    P_p = P_0;          % covariance after update step (P-minus)
    P_m = zeros(6);     % covariance after measurement step (P-plus)
    P_est = zeros(6,6000);        % log of covariance matrices at each
        timestep

    L = 0.5;            % UGV wheelbase (m)
    deltaT = tvec(2)-tvec(1);

    % need to load logged measurements and controls here
    % (or whatever it turns out to be)
    ey_log = zeros(5,1000); % log of measurement errors
    y_log = zeros(5,1000);
```

```
S_log = zeros(5,5000);

x_hat_p = x0; % set initial state estimate equal to initial state
              % may need to actually estimate this later
x_est = zeros(6,1001);
x_est(:,1) = x_hat_p; % log of estimated system states at each
    timestep

% get perturbation to groundtruth x
% sampled using initial covariance
x_perturb = mvnrnd(zeros(1,6),P_0);

x_gt = zeros(6,1001);
x_gt(:,1) = x0 + x_perturb';

u = [2 -pi/18 12 pi/25];

% generate groundtruth states
for i = 2:1001
    wk = mvnrnd(zeros(1,6),Qtrue);
    [~,next_x_gt] = ode45(@motionEqs, [0.0 deltaT], x_gt(:,i-1)',
        [], u');
    next_x_gt = next_x_gt(end,:)' + wk';
    next_x_gt(3) = constrainAngle(next_x_gt(3));
    next_x_gt(6) = constrainAngle(next_x_gt(6));
    x_gt(:,i) = next_x_gt;
end

for i = 2:1001 %2:size(tvec,2)

    % update step
    % calculate new state and covariance from system dynamics

    % get new state estimate using numerical integration
    [~,ode_x] = ode45(@motionEqs, [0.0 deltaT], x_hat_p', [], u');
    x_hat_m = ode_x(end,:)';
    x_hat_m(3) = constrainAngle(x_hat_m(3));
    x_hat_m(6) = constrainAngle(x_hat_m(6));

    % update estimated state covariance
    % Not using omega matrix because it's I in this case

    %[F,G] = getLinStateMats(x_hat_p, u, L, deltaT);
    [F,G] = getLinStateMats(x_hat_m, u, L, deltaT);
    P_m = F*P_p*F' + Q;

    % get actual measurement
    vk = mvnrnd(zeros(1,5),Rtrue);
```

```
        y = getMeas(x_gt(:,i)) + vk';
        y_log(:,i) = y;

        %y = ydata(:,i);

        % get predicted measurement using nonlinear model
        y_hat = getMeas(x_hat_m);

        % get measurement error
        e_y = y - y_hat;

        % get linearized H matrix
        H = getLinHMat(x_hat_m);

        %lin_ey = y - H*x_hat_m;

        ey_log(:,i-1) = e_y;

        S = H*P_m*H' + R;
        S_log(:,5*(i-2)+1:5*(i-2)+5) = S;

        % get the Kalman gain
        % assumes time-invariant R
        K = P_m*H'/S;

        % get corrected state and covariance estimates
        x_hat_p = x_hat_m + K*e_y;
        P_p = (eye(6) - K*H)*P_m;

        x_hat_p(3) = constrainAngle(x_hat_p(3));
        x_hat_p(6) = constrainAngle(x_hat_p(6));

        x_est(:,i) = x_hat_p; % add newly estimated state to log
        P_est(:,6*(i-2)+1:6*(i-2)+6) = P_p;

    end

end
```

Listing 6: Do Truth Model Testing and Calculate NEES and NIS Scores

```
function err = TMT_EKF()
num_runs = 100;
num_steps = 1001;
alpha = 0.05;
n = 6;
p = 5;

eps_x = zeros(num_runs,num_steps-1);
```

```
eps_y = zeros(num_runs,num_steps-1);

for i = 1:num_runs
    [P,x,S,e_y,true_x,~] = EKF();
    Pdim = size(P,1);
    Sdim = size(S,1);
    e_x = true_x - x;
    e_x = e_x(:,2:end);
    for j = 1:num_steps-1
        Pstart = (j-1)*Pdim + 1;
        Pend = j*Pdim;
        Sstart = (j-1)*Sdim + 1;
        Send = j*Sdim;
        eps_x(i,j) = ((e_x(:,j)'/P(:,Pstart:Pend))*e_x(:,j));
        eps_y(i,j) = ((e_y(:,j)'/S(:,Sstart:Send))*e_y(:,j));
    end
end

t = 0.1:0.1:100;

r1_y = chi2inv(alpha/2,num_runs*p)/num_runs;
r1_y = r1_y * ones(size(t));
r2_y = chi2inv(1-(alpha/2),num_runs*p)/num_runs;
r2_y = r2_y * ones(size(t));

r1_x = chi2inv(alpha/2,num_runs*n)/num_runs;
r1_x = r1_x * ones(size(t));
r2_x = chi2inv(1-(alpha/2),num_runs*n)/num_runs;
r2_x = r2_x * ones(size(t));

eps_x = sum(eps_x,1) ./ num_runs;
eps_y = sum(eps_y,1) ./ num_runs;

x_err_sum = 0;
y_err_sum = 0;
for i = 1:size(eps_x,2)
    x_err_sum = x_err_sum + (eps_x(i) - 6)^2;
    y_err_sum = y_err_sum + (eps_y(i) - 5)^2;
end

err = x_err_sum;

x_in_ct = 0;
y_in_ct = 0;

for i=1:size(eps_x,2)
    if (eps_x(i) > r1_x(1) && eps_x(i) < r2_x(1))
        x_in_ct = x_in_ct + 1;
```

```
    end
    if (eps_y(i) > r1_y(1) && eps_y(i) < r2_x(1))
        y_in_ct = y_in_ct + 1;
    end
end

x_in_ct / size(eps_x,2)
y_in_ct / size(eps_y,2)

scatter(t,eps_x);
hold on;
plot(t,r1_x,'b--',t,r2_x,'b--');
ylabel("NEES score");
xlabel("time (s)");
figure;
scatter(t,eps_y);
hold on;
plot(t,r1_y,'b--',t,r2_y,'b--');
ylabel("NIS score");
xlabel("time (s)");

end
```