**Appendix F – Codes accompanying Question 6**

|  |
| --- |
| % Question 6 - Compare Filter Performance  close all, clearvars, clc  load("cooplocalization\_finalproj\_KFdata.mat");    useUnwrappedY = false; % wrap Y for LKF  runMonteCarlo = false; % run MC or use Mat data  overlayAllRuns = true; % plot all runs on top of each other    if runMonteCarlo == true  x0 = [10 0 pi/2 -60 0 -pi/2]';  u0 = [2 -pi/18 12 pi/25]';  steps = 1000;  runs = 50;    Q\_lkf = diag([0.0001 0.0001 0.01 0.1 0.1 0.01]);  Q\_ekf = diag([.0015, .0015, 0.01, 0.001, 0.005, 0.01]);    Q\_ekf = [ .000001 0 1e-6 0 0 0;  0 .000001 1e-6 0 0 0;  1e-6 1e-6 .001 0 0 0;  0 0 0 .015 0 0e-6;  0 0 0 0 .015 0e-6;  0 0 0 0e-6 0e-6 .008]./18;  kp = 1;  P0 = diag([(kp\*1/2)^2 (kp\*1/2)^2 (kp\*0.25/2)^2 ...  (kp\*2.5/2)^2 (kp\*2.5/2)^2 (kp\*0.25/2)^2]);  else  x0 = [10 0 pi/2 -60 0 -pi/2]';  u0 = [2 -pi/18 12 pi/25]';  steps = size(ydata,2) - 2;  runs = 1;  Q\_lkf = diag([0.0001 0.0001 0.01 0.1 0.1 0.01]);  Q\_ekf = diag([.0015, .0015, 0.01, 0.001, 0.005, 0.01]);    Q\_ekf = [ .000001 0 1e-6 0 0 0;  0 .000001 1e-6 0 0 0;  1e-6 1e-6 .001 0 0 0;  0 0 0 .015 0 0e-6;  0 0 0 0 .015 0e-6;  0 0 0 0e-6 0e-6 .008]./18;  P0 = diag([1 1 0.025 1 1 0.025]);  end    p = 5;  Dt = 0.1;  n = size(x0,1);  seed = 100;  rng(seed);    EX\_lkf = zeros(n, steps+1, runs);  EY\_lkf = zeros(p, steps+1, runs);  PS\_lkf = zeros(n, n, steps+1, runs);  SS\_lkf = zeros(p, p, steps+1, runs);  EX\_ekf = EX\_lkf;  EY\_ekf = EY\_lkf;  PS\_ekf = PS\_lkf;  SS\_ekf = SS\_lkf;    if runMonteCarlo == true  fig1 = figure;  fig3 = figure;  fig5 = figure;  end    fig2 = figure;  fig4 = figure;  fig6 = figure;    for run = 1:runs  disp(['run #', num2str(run)]);  %----------------------------------------------------------------------  % generate data  if runMonteCarlo == true  if useUnwrappedY == true  [x, y] = GenerateTruth(x0, u0, P0, Qtrue, Rtrue, Dt, steps, false);  else  [x, y] = GenerateTruth(x0, u0, P0, Qtrue, Rtrue, Dt, steps, true);  end  t = (0:(length(x)-1))\*Dt;  unwrapped\_y = y;  else  y = ydata(:,2:end);  unwrapped\_y = y;  if useUnwrappedY == true  unwrapped\_y(1,:) = unwrap(y(1,:));  unwrapped\_y(3,:) = unwrap(y(3,:));  end  x = ones(size(x0,1),size(y,2));  x(:,1) = x0;  t = tvec(:,2:end);  steps = size(y,2) - 1;  end  if overlayAllRuns == true  if runMonteCarlo == true  PlotStates(fig1,t,x,'Ground Truth States, All Runs');  end  PlotMeasurements(fig2,t,y,'Ground Truth Measurements, All Runs');  end    % assume we can get exact measurement noise from  % specifications of sensors  R = Rtrue;    %----------------------------------------------------------------------  %----------------------------------------------------------------------  % LKF    % generate nominal trajectory for run, along with DT matrices  [x\_nom,y\_nom] = GenerateNom(x0, u0, steps, Dt);  x\_nom = WrapX(x\_nom);  y\_nom = WrapY(y\_nom);  [Fk, Hk, Ok] = GenerateLKFMats(u0, x\_nom, steps+1, p, Dt);  dx\_init = x0 - x\_nom(:,1);  dx\_init = WrapY(dx\_init);    % Run filter for all time-steps of run #k  [x\_est\_lkf,y\_est\_lkf,~,P\_lkf,S\_lkf,~,~] = LKF(dx\_init, P0, x\_nom, y\_nom, x, unwrapped\_y, ...  Fk, Hk, Ok, Q\_lkf, R, true);    % wrap angle diff too!!  ex\_lkf = x - x\_est\_lkf;  ex\_lkf(3,:) = angdiff(x\_est\_lkf(3,:),x(3,:));  ex\_lkf(6,:) = angdiff(x\_est\_lkf(6,:),x(6,:));  ey\_lkf = y - y\_est\_lkf;  ey\_lkf(1,:) = angdiff(y\_est\_lkf(1,:),y(1,:));  ey\_lkf(3,:) = angdiff(y\_est\_lkf(3,:),y(3,:));    % save run data from NEES/NIS tests  EX\_lkf(:, :, run) = ex\_lkf;  EY\_lkf(:, :, run) = ey\_lkf;  PS\_lkf(:, :, :, run) = P\_lkf;  SS\_lkf(:, :, :, run) = S\_lkf;  %----------------------------------------------------------------------  %----------------------------------------------------------------------  % EKF  [x\_est\_ekf, y\_est\_ekf, P\_ekf, S\_ekf] = EKF(x0, P0, u0, y, Q\_ekf, R, Dt);    % wrap angle diff too!!  ex\_ekf = x - x\_est\_ekf;  ex\_ekf(3,:) = angdiff(x\_est\_ekf(3,:),x(3,:));  ex\_ekf(6,:) = angdiff(x\_est\_ekf(6,:),x(6,:));  ey\_ekf = y - y\_est\_ekf;  ey\_ekf(1,:) = angdiff(y\_est\_ekf(1,:),y(1,:));  ey\_ekf(3,:) = angdiff(y\_est\_ekf(3,:),y(3,:));    % save run data from NEES/NIS tests  EX\_ekf(:, :, run) = ex\_ekf;  EY\_ekf(:, :, run) = ey\_ekf;  PS\_ekf(:, :, :, run) = P\_ekf;  SS\_ekf(:, :, :, run) = S\_ekf;  %----------------------------------------------------------------------  %----------------------------------------------------------------------  % Plot error during monte carlo runs  if overlayAllRuns == true  if runMonteCarlo == true  PlotStates(fig3,t,ex\_lkf, ['LKF State Errors, Runs ',num2str(run)], P\_lkf);  PlotStates(fig5,t,ex\_ekf, ['EKF State Errors, Runs ',num2str(run)], P\_ekf);  end  PlotMeasurements(fig4,t,ey\_lkf,['LKF Ground Truth Measurement Errors, Runs ',num2str(run)], S\_lkf);  PlotMeasurements(fig6,t,ey\_ekf,['EKF Ground Truth Measurement Errors, Runs ',num2str(run)], S\_ekf);  end  end    if overlayAllRuns == false  if runMonteCarlo == true  PlotStates(fig1,t,x,'Ground Truth States, All Runs');  PlotStates(fig3,t,ex\_lkf, ['LKF State Errors, Runs ',num2str(run)], P\_lkf);  PlotStates(fig5,t,ex\_ekf, ['EKF State Errors, Runs ',num2str(run)], P\_ekf);  end  PlotMeasurements(fig2,t,y,'Ground Truth Measurements, All Runs');  PlotMeasurements(fig4,t,ey\_lkf,['LKF Ground Truth Measurement Errors, Runs ',num2str(run)], S\_lkf);  PlotMeasurements(fig6,t,ey\_ekf,['EKF Ground Truth Measurement Errors, Runs ',num2str(run)], S\_ekf);  end    %% Calculate NEES and NIS statistics  [NEES\_lkf, NIS\_lkf] = CalcStats(EX\_lkf, EY\_lkf, PS\_lkf, SS\_lkf);  [NEES\_ekf, NIS\_ekf] = CalcStats(EX\_ekf, EY\_ekf, PS\_ekf, SS\_ekf);    %--------------------------------------------------------------------------  % NEES Plot  alpha = 0.05;  if runMonteCarlo == true  fig7 = figure;  PlotNees(fig7, NEES\_lkf, runs, n, alpha);  hold all;  PlotNees(fig7, NEES\_ekf, runs, n, alpha);  hold off;  legend('LKF $\bar{\epsilon}\_x$','$r\_1$','$r\_2$',...  'EKF $\bar{\epsilon}\_x$','FontSize',12,'Interpreter','latex')  end  %--------------------------------------------------------------------------  % NIS Plot  fig8 = figure;  PlotNis(fig8, NIS\_lkf, runs, p, alpha);  hold all;  PlotNis(fig8, NIS\_ekf, runs, p, alpha);  hold off;  legend('LKF $\bar{\epsilon}\_y$','$r\_1$','$r\_2$',...  'EKF $\bar{\epsilon}\_y$','FontSize',12,'Interpreter','latex')    %--------------------------------------------------------------------------  %% Comparison Plots  if runMonteCarlo == true  fig9 = figure;  fig11 = figure;  end    fig10 = figure;  fig12 = figure;    if runMonteCarlo == true  x(3,:) = wrapToPi(x(3,:));  x(6,:) = wrapToPi(x(6,:));  PlotStates(fig9,t,x,'Ground Truth States');  PlotStates(fig9,t,x\_est\_ekf,'Ground Truth States');  PlotStates(fig9,t,x\_est\_lkf,'Ground Truth States');  legend('Truth','LKF','EKF');  end    y(3,:) = wrapToPi(y(3,:));  y(1,:) = wrapToPi(y(1,:));  PlotMeasurements(fig10,t,y,'Ground Truth Measurements');  y\_est\_lkf(3,:) = wrapToPi(y\_est\_lkf(3,:));  y\_est\_lkf(1,:) = wrapToPi(y\_est\_lkf(1,:));  PlotMeasurements(fig10,t,y\_est\_lkf,'Ground Truth Measurements');  PlotMeasurements(fig10,t,y\_est\_ekf,'Ground Truth Measurements');  legend('Truth','LKF','EKF');    if runMonteCarlo == true  PlotStates(fig11,t,ex\_lkf, ['State Errors, Run ',num2str(run)]);  PlotStates(fig11,t,ex\_ekf, ['State Errors, Run ',num2str(run)]);  legend('LKF','EKF');  end  PlotMeasurements(fig12,t,ey\_lkf,['Ground Truth Measurement Errors, Run ',num2str(run)]);  PlotMeasurements(fig12,t,ey\_ekf,['Ground Truth Measurement Errors, Run ',num2str(run)]);  legend('LKF','EKF'); |
| function [x, y] = GenerateTruth(x0, u, P0, Q, R, Dt, steps, wrapOn)  opt = odeset('RelTol',1e-6,'AbsTol',1e-6);  useChol = true;  n = size(x0,1);  p = size(R,1);    x = zeros(n,steps+1);  y = zeros(p,steps+1);    % set initial conditions  dx = mvnrnd(zeros(1,n),P0);  x(:,1) = x0 + dx';  x(3,1) = wrapToPi(x(3,1));  x(6,1) = wrapToPi(x(6,1));    for i = 2:steps+1    % generate noisy state  if useChol==true  wk = chol(Q)\*randn(n,1);  else  wk = mvnrnd(zeros(1,n),Q)';  end  [~,next\_x] = ode45(@NL\_DynModel, [0 Dt], x(:,i-1)', opt, u', wk);    if wrapOn == true  % wrap angle to [-pi pi]  next\_x(3) = wrapToPi(next\_x(3));  next\_x(6) = wrapToPi(next\_x(6));  end  x(:,i) = next\_x(end,:)';  end    for i = 1:steps+1  % generate noisy measurement  if useChol==true  vk = chol(R)\*randn(p,1);  else  vk = mvnrnd(zeros(1,p),R)';  end  y(:,i) = NL\_MeasModel(x(:,i), vk);    if wrapOn == true  % wrap angle to [-pi pi]  y(1,i) = wrapToPi(y(1,i));  y(3,i) = wrapToPi(y(3,i));  end  end    end |
| function [x\_nom,y\_nom] = GenerateNom(x0, u0, steps, Dt)    t = 0:Dt:steps\*Dt;  [~,x\_nom] = ode45(@(t,x) NL\_DynModel(t,x,u0,zeros(6,1)),t,x0);  x\_nom = x\_nom';  y\_nom = zeros(5,length(t));    for i=1:length(t)  y\_nom(:,i) = NL\_MeasModel(x\_nom(:,i),zeros(5,1));  end    end |
| function [Fk, Hk, Ok] = GenerateLKFMats(u0, x\_nom, steps, p, Dt)    Fk = zeros(size(x\_nom, 1), size(x\_nom, 1), steps);  Hk = zeros(p, size(x\_nom, 1), steps);  Ok = zeros(size(x\_nom, 1), size(x\_nom, 1), steps);    for i=1:steps  [A\_t, B\_t, C\_t] = Linearize(x\_nom(:,i), u0);  [Fk(:,:,i), ~ , Hk(:,:,i)] = Discretize(A\_t,B\_t,C\_t, Dt);  Ok(:,:,i) = eye(size(x\_nom, 1));  end    end  function [A\_t,B\_t,C\_t] = Linearize(x,u)  %Linearize  % input: x - nominal state vector; u - nominal control input;  % output: A\_t - A tilde Matrix; B\_t - B tilde Matrix; C\_t - C tilde  % Matrix  % Obtain the CT linearized state perturbation matrices    % u = [v\_g, phi\_g, v\_a, w\_a]';  % x = [xi\_g eta\_g theta\_g xi\_a eta\_a theta\_a]';    L = 0.5;    A\_t = [0 0 -u(1)\*sin(x(3)) 0 0 0;  0 0 u(1)\*cos(x(3)) 0 0 0;  0 0 0 0 0 0;  0 0 0 0 0 -u(3)\*sin(x(6));  0 0 0 0 0 u(3)\*cos(x(6));  0 0 0 0 0 0];      B\_t = [cos(x(3)) 0 0 0;  sin(x(3)) 0 0 0;  tan(u(2))/L (u(1)/L)\*sec(u(2))^2 0 0;  0 0 cos(x(6)) 0;  0 0 sin(x(6)) 0;  0 0 0 1];    % x = [xi\_g eta\_g theta\_g xi\_a eta\_a theta\_a]';    C11 = (x(5)-x(2))/((x(5)-x(2))^2 + (x(4)-x(1))^2);  C12 = -(x(4)-x(1))/((x(5)-x(2))^2 + (x(4)-x(1))^2);  C13 = -1;  C14 = -(x(5)-x(2))/((x(5)-x(2))^2 + (x(4)-x(1))^2);  C15 = (x(4)-x(1))/((x(5)-x(2))^2 + (x(4)-x(1))^2);  C21 = (x(1)-x(4))\*((x(1)-x(4))^2 + (x(2)-x(5))^2)^-0.5;  C22 = (x(2)-x(5))\*((x(1)-x(4))^2 + (x(2)-x(5))^2)^-0.5;  C24 = -(x(1)-x(4))\*((x(1)-x(4))^2 + (x(2)-x(5))^2)^-0.5;  C25 = -(x(2)-x(5))\*((x(1)-x(4))^2 + (x(2)-x(5))^2)^-0.5;  C31 = -(x(2)-x(5))/((x(2)-x(5))^2 + (x(1)-x(4))^2);  C32 = (x(1)-x(4))/((x(2)-x(5))^2 + (x(1)-x(4))^2);  C34 = (x(2)-x(5))/((x(2)-x(5))^2 + (x(1)-x(4))^2);  C35 = -(x(1)-x(4))/((x(2)-x(5))^2 + (x(1)-x(4))^2);  C36 = -1;  C44 = 1;  C55 = 1;    C\_t = [C11 C12 C13 C14 C15 0;  C21 C22 0 C24 C25 0;  C31 C32 0 C34 C35 C36;  0 0 0 C44 0 0;  0 0 0 0 C55 0];  end |
| function [F, G, H] = Discretize(A\_t,B\_t,C\_t, Dt)  %Linearize  % Inputs:  % A\_t - linearized CT system matrix  % B\_t - lizearized CT input matrix  % C\_t - linearized CT measurement matrix  % Output  % F - state transition matrix  % G -control effect matrix  % H - sensing matrix  %  % Discretizze the CT state perturbation matrices    H = C\_t;  F = eye(size(A\_t)) + Dt \* A\_t;  G = Dt \* B\_t;  end |
| function [NEES, NIS] = CalcStats(EX, EY, P, S)    steps = size(EX, 2);  runs = size(EX, 3);    NEES\_all = zeros(runs,steps);  NIS\_all = zeros(runs,steps);  NEES = zeros(1,steps);  NIS = zeros(1,steps);    for run=1:runs  for step=1:steps  NEES(step) = EX(:,step, run)' / P(:,:,step, run) \* EX(:,step, run);  NIS(step) = EY(:,step, run)' / S(:,:,step, run) \* EY(:,step, run);  end    NEES\_all(run,:) = NEES;  NIS\_all(run,:) = NIS;  end    % calculate mean at each time step  for i=1:steps  NEES(i) = mean(NEES\_all(:,i));  NIS(i) = mean(NIS\_all(:,i));  end  end |
| function PlotMeasurements(hdl,t,y, title, S)    if nargin > 4  s = zeros(size(y));  for ind = 1:size(y,2)  s(:,ind) = 2\*sqrt(diag(S(:,:,ind)));  end  displayError = true;  else  displayError = false;  end    figure(hdl)  if isempty(hdl.Children)  tiledlayout(3,2);  ax1 = nexttile;  ax2 = nexttile;  ax3 = nexttile;  ax4 = nexttile;  ax5 = nexttile;  else  thdl = hdl.Children;  ax1 = thdl.Children(5,1);  ax2 = thdl.Children(4,1);  ax3 = thdl.Children(3,1);  ax4 = thdl.Children(2,1);  ax5 = thdl.Children(1,1);  end    hold([ax1 ax2 ax3 ax4 ax5],'on');  ftSize = 10;  sgtitle(title,'FontSize',ftSize+2,'Interpreter','latex')  output = 1;  plot(ax1, t,y(output,:))  if displayError == true  plot(ax1, t, s(output,:),'r--'), plot(ax1, t, -s(output,:),'r--')  end  ylabel(ax1, '$\gamma\_{ag}$ (rad)','FontSize',ftSize,'Interpreter','latex')  xlabel(ax1, 'Time(s)','FontSize',ftSize,'Interpreter','latex')  axis([min(t) max(t) min(y(output,:)) max(y(output,:))])  grid on    output = output + 1;  plot(ax5, t,y(output,:))  if displayError == true  plot(ax5, t, s(output,:),'r--'), plot(ax5, t, -s(output,:),'r--')  end  ylabel(ax5, '$\rho\_{ga}$ (m)','FontSize',ftSize,'Interpreter','latex')  xlabel(ax5, 'Time(s)','FontSize',ftSize,'Interpreter','latex')  axis([min(t) max(t) min(y(output,:)) max(y(output,:))])  grid on    output = output + 1;  plot(ax3, t,y(3,:))  if displayError == true  plot(ax3, t, s(output,:),'r--'), plot(ax3, t, -s(output,:),'r--')  end  ylabel(ax3, '$\gamma\_{ga}$ (rad)','FontSize',ftSize,'Interpreter','latex')  xlabel(ax3, 'Time(s)','FontSize',ftSize,'Interpreter','latex')  axis([min(t) max(t) min(y(output,:)) max(y(output,:))])  grid on    output = output + 1;  plot(ax2, t,y(output,:))  if displayError == true  plot(ax2, t, s(output,:),'r--'), plot(ax2, t, -s(output,:),'r--')  end  ylabel(ax2, '$\xi\_a$ (m)','FontSize',ftSize,'Interpreter','latex')  xlabel(ax2, 'Time(s)','FontSize',ftSize,'Interpreter','latex')  axis([min(t) max(t) min(y(output,:)) max(y(output,:))])  grid on    output = output + 1;  plot(ax4, t,y(output,:))  if displayError == true  plot(ax4, t, s(output,:),'r--'), plot(ax4, t, -s(output,:),'r--')  end  ylabel(ax4, '$\eta\_a$ (m)','FontSize',ftSize,'Interpreter','latex')  xlabel(ax4, 'Time(s)','FontSize',ftSize,'Interpreter','latex')  axis([min(t) max(t) min(y(output,:)) max(y(output,:))])  grid on    grid([ax1 ax2 ax3 ax4 ax5],'on');  hold([ax1 ax2 ax3 ax4 ax5],'off');  end |
| function PlotNees(hdl, epsNEESbar, Nsimruns, n, alpha)    figure(hdl);  Nnx = Nsimruns\*n;    %%compute intervals:  r1x = chi2inv(alpha/2, Nnx )./ Nsimruns;  r2x = chi2inv(1-alpha/2, Nnx )./ Nsimruns;    figure(hdl)  plot(epsNEESbar,'o','MarkerSize',6),hold on  plot(r1x\*ones(size(epsNEESbar)),'r--')  plot(r2x\*ones(size(epsNEESbar)),'r--')  ylabel('NEES statistic, $\bar{\epsilon}\_x$','FontSize',14,'Interpreter','latex')  xlabel('time step, k','FontSize',14)  title('NEES Estimation Results','FontSize',14)  legend('$\bar{\epsilon}\_x$','$r\_1$','$r\_2$','FontSize',12,'Interpreter','latex')  grid on;  end |
| function PlotNis(hdl, epsNISbar, Nsimruns, p, alpha)    figure(hdl);  Nny = Nsimruns\*p;    %%compute intervals:  r1y = chi2inv(alpha/2, Nny )./ Nsimruns;  r2y = chi2inv(1-alpha/2, Nny )./ Nsimruns;    plot(epsNISbar,'o','MarkerSize',6),hold on  plot(r1y\*ones(size(epsNISbar)),'r--')  plot(r2y\*ones(size(epsNISbar)),'r--')  ylabel('NIS statistic, $\bar{\epsilon}\_y$','FontSize',14,'Interpreter','latex')  xlabel('time step, k','FontSize',14)  title('NIS Estimation Results','FontSize',14)  legend('$\bar{\epsilon}\_y$','$r\_1$','$r\_2$','FontSize',12,'Interpreter','latex')  grid on;  end |
| function PlotStates(hdl,t,x, title, P)    if nargin > 4  p = zeros(size(x));  for ind = 1:size(x,2)  p(:,ind) = 2\*sqrt(diag(P(:,:,ind)));  end  displayError = true;  else  displayError = false;  end    figure(hdl)  if isempty(hdl.Children)  tiledlayout(3,2);  ax1 = nexttile;  ax2 = nexttile;  ax3 = nexttile;  ax4 = nexttile;  ax5 = nexttile;  ax6 = nexttile;  else  thdl = hdl.Children;  ax1 = thdl.Children(6,1);  ax2 = thdl.Children(5,1);  ax3 = thdl.Children(4,1);  ax4 = thdl.Children(3,1);  ax5 = thdl.Children(2,1);  ax6 = thdl.Children(1,1);  end    hold([ax1 ax2 ax3 ax4 ax5 ax6],'on');    ftSize = 10;  sgtitle(title,'FontSize',ftSize+2,'Interpreter','latex')  state = 1;    plot(ax1, t,x(state,:))  if displayError == true  plot(ax1, t, p(state,:),'r--'), plot(ax1, t, -p(state,:),'r--')  end  ylabel(ax1, '$\xi\_g$ (m)','FontSize',ftSize,'Interpreter','latex')  xlabel(ax1, 'Time(s)','FontSize',ftSize,'Interpreter','latex')  axis([min(t) max(t) min(x(state,:)) ...  max(x(state,:))])    state = state + 1;  plot(ax3, t,x(state,:))  if displayError == true  plot(ax3, t, p(state,:),'r--'), plot(ax3, t, -p(state,:),'r--')  end  ylabel(ax3, '$\eta\_g$ (m)','FontSize',ftSize,'Interpreter','latex')  xlabel(ax3, 'Time(s)','FontSize',ftSize,'Interpreter','latex')  axis([min(t) max(t) min(x(state,:)) ...  max(x(state,:))])    state = state + 1;  plot(ax5, t,wrapToPi(x(state,:)))  if displayError == true  plot(ax5, t, p(state,:),'r--'), plot(ax5, t, -p(state,:),'r--')  end  ylabel(ax5, '$\theta\_g$ (rad)','FontSize',ftSize,'Interpreter','latex')  xlabel(ax5, 'Time(s)','FontSize',ftSize,'Interpreter','latex')  axis([min(t) max(t) min(wrapToPi(x(state,:))) ...  max(wrapToPi(x(state,:)))])    state = state + 1;  plot(ax2, t,x(state,:))  if displayError == true  plot(ax2, t, p(state,:),'r--'), plot(ax2, t, -p(state,:),'r--')  end  ylabel(ax2, '$\xi\_a$ (m)','FontSize',ftSize,'Interpreter','latex')  xlabel(ax2, 'Time(s)','FontSize',ftSize,'Interpreter','latex')  axis([min(t) max(t) min(x(state,:)) ...  max(x(state,:))])    state = state + 1;  plot(ax4, t,x(state,:))  if displayError == true  plot(ax4, t, p(state,:),'r--'), plot(ax4, t, -p(state,:),'r--')  end  ylabel(ax4, '$\eta\_a$ (m)','FontSize',ftSize,'Interpreter','latex')  xlabel(ax4, 'Time(s)','FontSize',ftSize,'Interpreter','latex')  axis([min(t) max(t) min(x(state,:)) ...  max(x(state,:))])    state = state + 1;  plot(ax6, t,wrapToPi(x(state,:)))  if displayError == true  plot(ax6, t, p(state,:),'r--'), plot(ax6, t, -p(state,:),'r--')  end  ylabel(ax6, '$\theta\_a$ (rad)','FontSize',ftSize,'Interpreter','latex')  xlabel(ax6, 'Time(s)','FontSize',ftSize,'Interpreter','latex')  axis([min(t) max(t) min(wrapToPi(x(state,:))) ...  max(wrapToPi(x(state,:)))])    grid([ax1 ax2 ax3 ax4 ax5 ax6],'on');  hold([ax1 ax2 ax3 ax4 ax5 ax6],'off');  end |
| function [out] = WrapX(in)  out = in;  out(3,:) = wrapToPi(out(3,:));  out(6,:) = wrapToPi(out(6,:));  end |
| function [out] = WrapY(in)  out = in;  out(1,:) = wrapToPi(out(1,:));  out(3,:) = wrapToPi(out(3,:));  end |