**Appendix E – Codes accompanying Question 5**

|  |
| --- |
| % Question 5 - Tune EKF  close all, clearvars, clc  load("cooplocalization\_finalproj\_KFdata.mat");    x0 = [10 0 pi/2 -60 0 -pi/2]';  u0 = [2 -pi/18 12 pi/25]';  Dt = 0.1;  n = size(x0,1);  steps = 1000;  seed = 100;  rng(seed);    Q = [ .00001 0 1e-6 0 0 0;  0 .00001 1e-6 0 0 0;  1e-6 1e-6 .001 0 0 0;  0 0 0 .015 0 0e-6;  0 0 0 0 .015 0e-6;  0 0 0 0e-6 0e-6 .008]./18;    kp = 1;  P0 = diag([(kp\*1/2)^2 (kp\*1/2)^2 (kp\*0.25/2)^2 ...  (kp\*2.5/2)^2 (kp\*2.5/2)^2 (kp\*0.25/2)^2]);    runs = 50;  EX = zeros(n, steps+1, runs);  p = 5;  EY = zeros(p, steps+1, runs);  PS = zeros(n, n, steps+1, runs);  SS = zeros(p, p, steps+1, runs);  fig1 = figure;  enablePlotDuring = true;  for run = 1:runs  disp(['run #', num2str(run)]);    % generate truth for run  [x, y] = GenerateTruth(x0, u0, P0, Qtrue, Rtrue, Dt, steps, true);  t = (0:(length(x)-1))\*Dt;    % assume we can get exact measurement noise from  % specifications of sensors  R = Rtrue;    % Run filter for all time-steps of run #k  [x\_est, y\_est, P, S] = EKF(x0, P0, u0, y, Q, R, Dt);    % wrap angle diff too!!  ex = WrapX(x - x\_est);  ey = WrapY(y - y\_est);    % Plot error during monte carlo runs  if enablePlotDuring == true  PlotStates(fig1,t,ex, ['State Errors, Run ',num2str(run)], P);  end    % save run data from NEES/NIS tests  EX(:, :, run) = ex;  EY(:, :, run) = ey;  PS(:, :, :, run) = P;  SS(:, :, :, run) = S;  end    %% Calculate NEES and NIS statistics  [NEES\_bar, NIS\_bar] = CalcStats(EX, EY, PS, SS);    %--------------------------------------------------------------------------  % Plots for (a)  PlotStates(fig1,t,ex, ['State Errors, Run ',num2str(run)], P);    fig2 = figure;  fig3 = figure;  fig4 = figure;  PlotMeasurements(fig2,t,y,'Ground Truth Measurements');  PlotStates(fig3,t,x,'Ground Truth States');  PlotMeasurements(fig4,t,ey,['Ground Truth Measurement Errors, Run ',num2str(run)]);    %--------------------------------------------------------------------------  % Plots for (b)  fig5 = figure;  alpha = 0.05;  PlotNees(fig5, NEES\_bar, runs, n, alpha);  %--------------------------------------------------------------------------  % Plots for (c)  fig6 = figure;  alpha = 0.01;  PlotNis(fig6, NIS\_bar, runs, p, alpha); |
| function [x\_est, y\_est, P, S] = EKF(x0, P0, u, y, Q, R, Dt)    % set simulation tolerances for ode45  opt = odeset('RelTol',1e-6,'AbsTol',1e-6);    n = size(x0, 1); % number of states  p = size(R, 1); % number of measurements  steps = size(y,2); % number of time steps    x\_est = zeros(n, steps); % state estimate vector  y\_est = zeros(p, steps); % measurement estimate vector  P = zeros(n, n, steps); % coveriance  S = zeros(p, p, steps);    % start with initial estimate of total state  % and covariance  x\_p = x0;  P\_p = P0;    for i=1:steps  %----------------------------------------------------------  % Prediction Step  % use full NL equations to estimate state at next time step  % using state at previous time step; since Wk is AWGN,  % its expected value is zero, set input to zero  wk = zeros(1,n);  [~, x\_m] = ode45(@NL\_DynModel, [0.0 Dt], x\_p', opt, u', wk);  x\_m = x\_m(end,:)';  x\_m = WrapX(x\_m);    % to calculate covariance, linearize "online"  % about current state estimate  [A\_t,B\_t,C\_t] = Linearize(x\_m, u);  [F, ~, H] = Discretize(A\_t, B\_t ,C\_t, Dt);  P\_m = F\*P\_p\*F' + Q;    % uese estimated state from NL ODEs; since Wk is AWGN,  % its expected value is zero, set to zero  vk = zeros(p,1);  y\_est(:,i) = NL\_MeasModel(x\_m, vk);  y\_est = WrapY(y\_est);    % calculate innovation vector  e\_y = y(:,i) - y\_est(:,i);  e\_y = WrapY(e\_y);    %----------------------------------------------------------  % Correction Step  % calculate gain using linearized measurement  % matrix H and covariance from Prediction Step  S\_p = H\*P\_m\*H' + R;  K = P\_m\*H'/S\_p;    % calculate posterior state estimate and covariance  x\_p = x\_m + K\*e\_y;  x\_p = WrapX(x\_p);  P\_p = (eye(n) - K\*H)\*P\_m;    % for each time-step, save estimate and covariance  x\_est(:,i) = x\_p;  P(:,:,i) = P\_p;  S(:,:,i) = 0.5\*(S\_p + S\_p');  end    end |
| function [x, y] = GenerateTruth(x0, u, P0, Q, R, Dt, steps, wrapOn)  opt = odeset('RelTol',1e-6,'AbsTol',1e-6);  useChol = true;  n = size(x0,1);  p = size(R,1);    x = zeros(n,steps+1);  y = zeros(p,steps+1);    % set initial conditions  dx = mvnrnd(zeros(1,n),P0);  x(:,1) = x0 + dx';  x(3,1) = wrapToPi(x(3,1));  x(6,1) = wrapToPi(x(6,1));    for i = 2:steps+1    % generate noisy state  if useChol==true  wk = chol(Q)\*randn(n,1);  else  wk = mvnrnd(zeros(1,n),Q)';  end  [~,next\_x] = ode45(@NL\_DynModel, [0 Dt], x(:,i-1)', opt, u', wk);    if wrapOn == true  % wrap angle to [-pi pi]  next\_x(3) = wrapToPi(next\_x(3));  next\_x(6) = wrapToPi(next\_x(6));  end  x(:,i) = next\_x(end,:)';  end    for i = 1:steps+1  % generate noisy measurement  if useChol==true  vk = chol(R)\*randn(p,1);  else  vk = mvnrnd(zeros(1,p),R)';  end  y(:,i) = NL\_MeasModel(x(:,i), vk);    if wrapOn == true  % wrap angle to [-pi pi]  y(1,i) = wrapToPi(y(1,i));  y(3,i) = wrapToPi(y(3,i));  end  end    end |
| function [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 |