ESE 547 Homework 8 /

Seif Elkhashab / 04-12-2024 /

Scalar MRAC Design

Chapter 7. Consider the first order roll dynamics of an aircraft

$$\dot{p} = L_p p + L_{\delta_{ail}} \delta_{ail}$$

where p is the roll rate, δ_{ail} is the control input, L_p is the unknown roll damping, and $L_{\delta_{ail}}$ is the unknown aileron control power. The desired dynamics (model) are given by

$$\dot{p}_m = L_p^m p_m + L_\delta^m r(t)$$

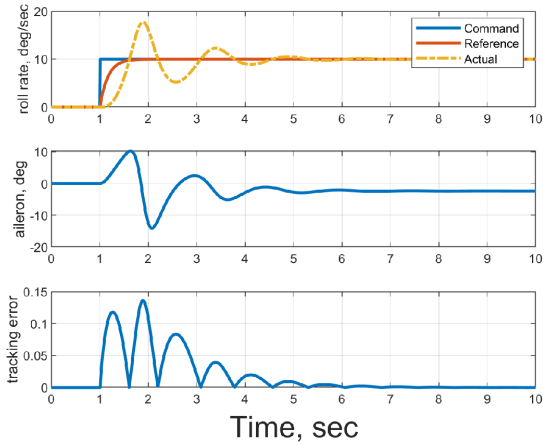
where $r(t)$ is the pilot reference input.

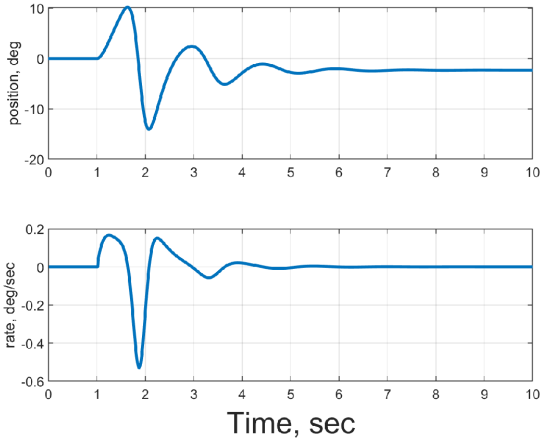
Design an adaptive control law, $\delta_a il = \hat{K}_p p + \hat{K}_r r$ to track the reference model in Eq. (2). You must debug the m-file given to get it to run correctly. Four statements need to be corrected. Test your adaptive controller using $r(t)$ t for a step, sine wave, and triangle wave inputs. The m-file has a menu to select the input and plot the results.

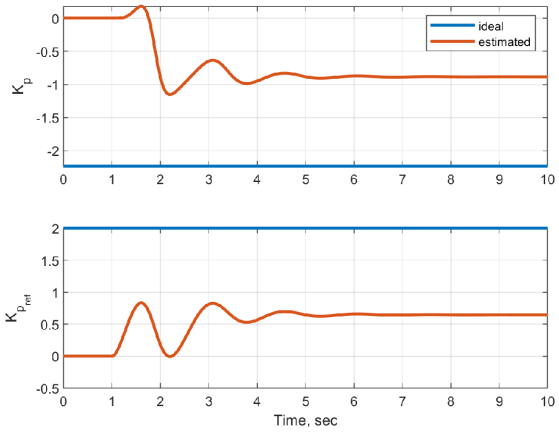
Turn in the plots for the response and gains versus time.

Turn in your m-file that produced the response.

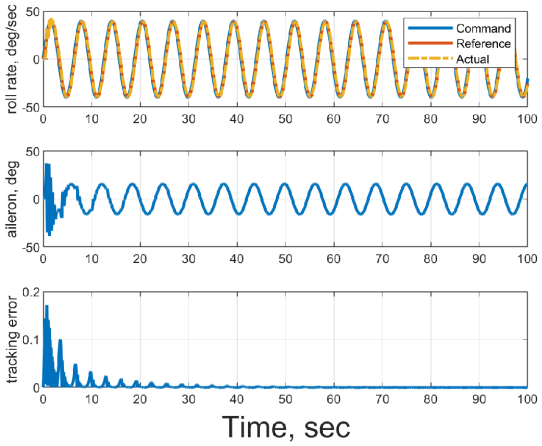
STEP INPUT RESULTS:

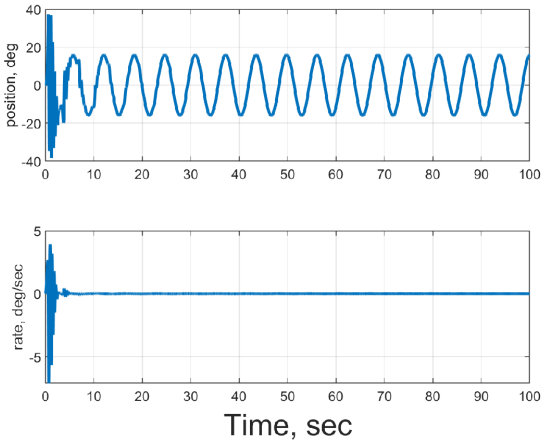


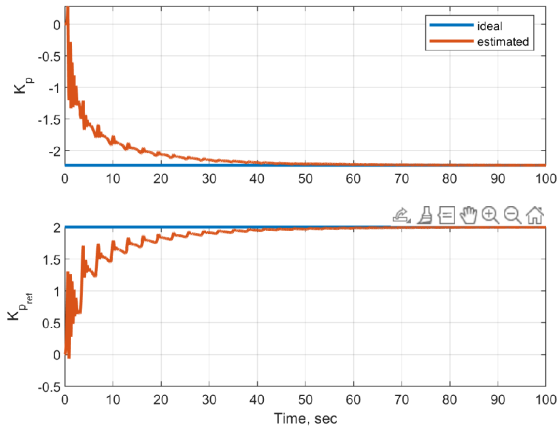




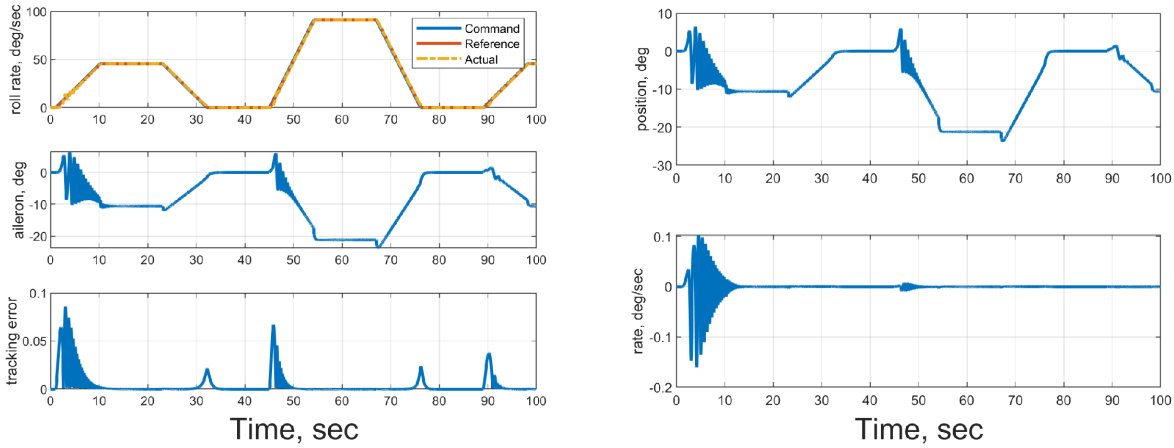
SINE INPUT RESULTS:

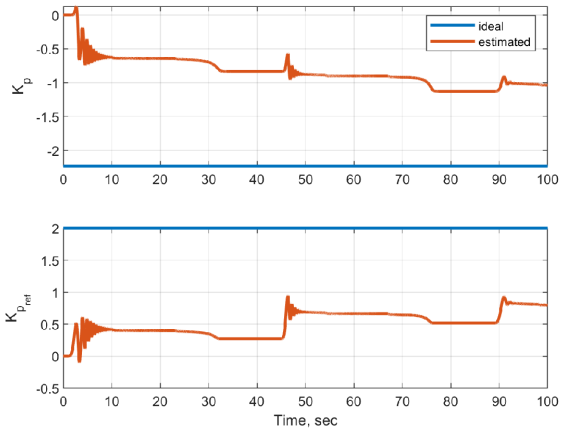






RAMP INPUT RESULTS:





Code:

```
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% Purpose:
%   Examples 11.1, 11.2, 11.3: MRAC design for scalar roll dynamics with robustness
%   modifications: 1) dead-zone, 2) sigma-mod, and 3) e-mod
%
% Created:
%   06-17-11, Eugene Lavretsky
%
% Modified:
%   01-14-12, Eugene Lavretsky
%   1) Changed section #-s
%   2) Converted to Homework 7
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
close all, clear all

d2r = pi/180;
r2d = 180/pi;

% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% Control on/off selection
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% mrac_mod_selection = menu('MRAC Mod-s Selection', 'None', '11.1: Dead-zone',
'11.2: Sigma', '11.3: E') - 1;
mrac_mod_selection = 0.;


input_selection = menu('R-input Selection', 'Step', 'Sine', 'Ramp');

% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% simulation time data
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
dt     = .01; % integration time step, sec
if input_selection == 1
    Tfinal = 10;  % simulation stop time, sec
else
    Tfinal = 100;  % simulation stop time, sec
end

time   = [0:dt:Tfinal]';
npnts  = length(time);

% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% commanded step-input, r(t)
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
r_max = 10*d2r;

% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% Case 1: step input
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
if input_selection == 1
    r = r_max + time*0;
    one_sec = [1:(1/dt + 1)];
    r(one_sec) = r(one_sec)*0;
end

% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% Case 2: sinusoidal inputs
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
if input_selection == 2
    r_om = 1;  % base frequency
    % r  = sin(r_om*time) + sin(r_om*time/2) + sin(r_om*time/4) + sin(r_om*time/8);
    r  = sin(r_om*time);
    r  = r/max(abs(r))*r_max*4;
```

```
end


% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% Case 3: Series of step-inputs
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
if input_selection == 3
t_cmd_step1 = [0, 1, 1.1, 10.1, 10.2, 20];
t_cmd_step2 = t_cmd_step1(end) + 2.0 + t_cmd_step1;
t_cmd_step3 = t_cmd_step2(end) + 2.0 + t_cmd_step1;
t_cmd_step4 = t_cmd_step3(end) + 2.0 + t_cmd_step1;
t_cmd_step5 = t_cmd_step4(end) + 2.0 + t_cmd_step1;
t_cmd_step6 = t_cmd_step5(end) + 2.0 + t_cmd_step1;

 r_cmd_step = [0, 0, 1.0, 1.0, 0.0, 0.0];

t_cmd_table = [t_cmd_step1, t_cmd_step2, t_cmd_step3, t_cmd_step4, t_cmd_step5,
t_cmd_step6];
t_cmd_table(end) = max(t_cmd_table(end), Tfinal);

 r_cmd_table = [0.5*r_cmd_step, -0.5*r_cmd_step, r_cmd_step, -r_cmd_step,
0.5*r_cmd_step, -0.5*r_cmd_step];

 r_step = interp1(t_cmd_table,r_cmd_table,time)*r_max;

  sys_int = tf(1,[1 0]);
  r = lsim(sys_int,r_step,time);
end
%
%  figure, plot(time,y_int)

% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% Nominal system data
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Lp    = 0.7;
Ldela =  3.;  % (dela is in rad)

A = Lp;
B = Ldela;

% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% Reference model data
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Aref = -6;
Bref =  6;
%Aref =  6;
%Bref =  -6;

% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% Ideal / True gains
```

```
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Kx_ideal = (Aref - A)/B;
Kr_ideal = Bref/B;

% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% Adaptive control learning rate
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Gamma_x = 100;  % Kx learning rates
Gamma_r = 100;  % Kr learning rate
%Gamma_x = 0;  % Kx learning rates
%Gamma_r = 0;  % Kr learning rate

% MRAC mod parameters
if mrac_mod_selection == 0,     % all mods are Off
    sigma_mod_gain = 0.0;
    e_mod_gain     = 0.0;
    dead_zone_tol  = 0.0;

    % elseif mrac_mod_selection == 1, % dead-zone is On
    %   sigma_mod_gain = 0.0;
    %   e_mod_gain     = 0.0;
    %   dead_zone_tol  = 3*d2r;
    %
    % elseif mrac_mod_selection == 2, % sigma-mod is On
    %   sigma_mod_gain = 0.1;
    %   e_mod_gain     = 0.0;
    %   dead_zone_tol  = 0.0;
    %
    % elseif mrac_mod_selection == 3, % e-mod is On
    %   sigma_mod_gain = 0.0;
    %   e_mod_gain     = 1;
    %   dead_zone_tol  = 0.0;
end;

% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% Noise data
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 f_min = -10*d2r;
 f_max =  10*d2r;
% f_min = 0.;
% f_max = 0.;

% % save random number generator state to be repeatable
% if ~exist('savedState', 'var')
%   defaultStream = RandStream.getDefaultStream;
%   savedState    = defaultStream.State;
% end;
%
% defaultStream.State = savedState;
%
```

```
% Generate repeatable values from the uniform distribution on the interval [f_min,
f_max].
f_data = f_min + (f_max - f_min).*rand(npnts,1);

% gust filter
s = tf('s');
sys_f = ss(10/(s+10));

f = lsim(sys_f,f_data,time);
f = f/max(f)*f_max;

% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% data pre-allocation for speed
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
x        = zeros(1,npnts);
xref     = zeros(1,npnts);
x_dot    = zeros(1,npnts);
xref_dot = zeros(1,npnts);

u = zeros(1,npnts);

e      = zeros(1,npnts);
norm_e = zeros(npnts,1);

Kx_hat     = zeros(1,npnts);
Kr_hat     = zeros(1,npnts);
Kx_hat_dot = zeros(1,npnts);
Kr_hat_dot = zeros(1,npnts);

% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% Main simulation loop starts here
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
for k = 1:npnts-1,
  % tracking error vector
  e(:,k)  = x(:,k) - xref(:,k);
  e_k     = e(:,k);

  norm_e(k) = norm(e_k);

  % ~~~~~~~~~~~~~~~~~~~~~~~~~
  % adaptive laws with mod-s
  % ~~~~~~~~~~~~~~~~~~~~~~~~~
  if mrac_mod_selection == 0,     % all mods are Off
      Kx_hat_dot(:,k) = -Gamma_x*x(:,k)*e_k;
      Kr_hat_dot(:,k) = -Gamma_r*r(k)*e_k;

  elseif mrac_mod_selection == 1, % dead-zone is On
      if abs(e_k) > dead_zone_tol
          e_dead_zone_k = e_k;
      else
```

```
        e_dead_zone_k = 0;
    end;
    Kx_hat_dot(:,k) = -Gamma_x*x(:,k)*e_dead_zone_k;
    Kr_hat_dot(:,k) = -Gamma_r*r(k)  *e_dead_zone_k;

elseif mrac_mod_selection == 2, % sigma-mod is On
    Kx_hat_dot(:,k) = -Gamma_x*x(:,k)*e_k - sigma_mod_gain*Kx_hat(:,k);
    Kr_hat_dot(:,k) = -Gamma_r*r(k)*e_k   - sigma_mod_gain*Kr_hat(:,k);

elseif mrac_mod_selection == 3, % e-mod is On
    Kx_hat_dot(:,k) = -Gamma_x*x(:,k)*e_k - e_mod_gain*norm_e(k)*Kx_hat(:,k);
    Kr_hat_dot(:,k) = -Gamma_r*r(k)*e_k   - e_mod_gain*norm_e(k)*Kr_hat(:,k);

end;

% ~~~~~~~~~~~~~~~~~~~~~~~~~
% adaptive control
% ~~~~~~~~~~~~~~~~~~~~~~~~~
u(k) = Kx_hat(:,k)'*x(:,k) + Kr_hat(k)*r(k);

% ~~~~~~~~~~~~~~~~~~~~~~~~~
% true plant dynamics
% ~~~~~~~~~~~~~~~~~~~~~~~~~
%x_dot(:,k) = A*x(:,k) + B*(u(k) + f(k));
x_dot(:,k) = A*x(:,k) + B*u(k);

% ~~~~~~~~~~~~~~~~~~~~~~~~~
% reference model dynamics
% ~~~~~~~~~~~~~~~~~~~~~~~~~
xref_dot(:,k) = Aref*xref(:,k) + Bref*r(k);

% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% one-step-ahead integration using AB-2 method
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
if k == 1,
    % true plant dynamics integration
    x(:,k+1) = x(:,k) + dt*x_dot(:,k);

    % reference model dynamics integration
    xref(:,k+1) = xref(:,k) + dt*xref_dot(:,k);

    % adaptive law dynamics
    Kx_hat(:,k+1) = Kx_hat(:,k) + dt*Kx_hat_dot(:,k);
    Kr_hat(k+1)   = Kr_hat(k)   + dt*Kr_hat_dot(k);

else
    % true plant dynamics integration
    x(:,k+1) = x(:,k) + dt*(1.5*x_dot(:,k) - 0.5*x_dot(:,k-1));

    % reference model dynamics integration
```

```
      xref(:,k+1) = xref(:,k) + dt*(1.5*xref_dot(:,k) - 0.5*xref_dot(:,k-1));

      % adaptive laws
      Kx_hat(:,k+1) = Kx_hat(:,k) + dt*(1.5*Kx_hat_dot(:,k) -
0.5*Kx_hat_dot(:,k-1));
      Kr_hat(k+1)   = Kr_hat(k)   + dt*(1.5*Kr_hat_dot(k)   - 0.5*Kr_hat_dot(k-1));
  end;

end;
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% Main simulation loop ends here
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

% end points
u(end)    = u(end-1);
e(:,end)  = e(:,end-1);

x_dot(:,end)    = x_dot(:,end-1);
xref_dot(:,end) = xref_dot(:,end-1);

Kx_hat_dot(:,end) = Kx_hat_dot(:,end-1);
Kr_hat_dot(:,end) = Kr_hat_dot(end-1);

% sim outputs
p_cmd_dps = r   *r2d;
p_dps     = x   *r2d;
p_ref_dps = xref*r2d;

dela_deg     = u*r2d;
dela_dot_dps = [0, diff(dela_deg)]'./time;

% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% Plots
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% Regulated output and control
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
figure,
subplot(311);
plot(time,p_cmd_dps, time,p_ref_dps, time,p_dps,'-.','Linewidth',2), grid;
ylabel('roll rate, deg/sec','fontsize',10);
% title('Tracking Performance');
legend('Command', 'Reference','Actual');

subplot(312);
plot(time, dela_deg,'Linewidth',2), grid;
ylabel('aileron, deg','fontsize',10);

subplot(313);
```

```
plot(time, norm_e,'Linewidth',2), grid;
ylabel('tracking error','fontsize',10);
xlabel('Time, sec','fontsize',20);

% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% Control position and rate
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
figure,
subplot(211);
plot(time, dela_deg,'Linewidth',2), grid;
ylabel('position, deg','fontsize',10);
% title('Differential Aileron');

subplot(212);
plot(time, dela_dot_dps,'Linewidth',2), grid;
ylabel('rate, deg/sec','fontsize',10);
xlabel('Time, sec','fontsize',20);

% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% Adaptive gains
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
figure,
subplot(211);
plot(time, Kx_ideal + time*0, time, Kx_hat,'Linewidth',2), grid;
ylabel('K_p','fontsize',10);
% ylabel('K_x','fontsize',20);
% title('Adaptive Gains');
legend('ideal', 'estimated');

subplot(212);
plot(time, Kr_ideal + time*0, time, Kr_hat,'Linewidth',2), grid;
ylabel('K_{p_{ref}}','fontsize',10);
% ylabel('K_r','fontsize',20);
xlabel('Time, sec','fontsize',10);
close all, clear all
return;
```