```
% TTK4135 - Helicopter lab
% Hints/template for problem 2.
% Updated spring 2018, Andreas L. Flåten

%% Initialization and model definition
init10; % Change this to the init file corresponding to your helicopter

% Discrete time system model. x = [lambda r p p_dot]'
delta_t = 0.25; % sampling time
A1 = [0 1 0 0;
    0 0 -K_2 0;
    0 0 0 delta_t;
    0 0 -K_1*K_pp -K_1*K_pd]*delta_t+eye(4);
B1 = [0;0;0;K_1*K_pp*delta_t];

% Number of states and inputs
mx = size(A1,2); % Number of states (number of columns in A)
mu = size(B1,2); % Number of inputs(number of columns in B)

% Initial values
x1_0 = pi;                              % Lambda
x2_0 = 0;                               % r
x3_0 = 0;                               % p
x4_0 = 0;                               % p_dot
x0 = [x1_0 x2_0 x3_0 x4_0]';           % Initial values

% Time horizon and initialization
N  = 100;                                  % Time horizon for states
M  = N;                                 % Time horizon for inputs
z  = zeros(N*mx+M*mu,1);                % Initialize z for the whole horizon
z0 = z;                                 % Initial value for optimization

% Bounds
ul      = -30*pi/180;                   % Lower bound on control
uu      = 30*pi/180;                   % Upper bound on control

xl      = -Inf*ones(mx,1);              % Lower bound on states (no bound)
xu      = Inf*ones(mx,1);               % Upper bound on states (no bound)
xl(3)   = ul;                           % Lower bound on state x3
xu(3)   = uu;                           % Upper bound on state x3

% Generate constraints on measurements and inputs
[vlb,vub]       = gen_constraints(N,M,xl,xu,ul,uu); % hint: gen_constraints
vlb(N*mx+M*mu)  = 0;                    % We want the last input to be zero
vub(N*mx+M*mu)  = 0;                    % We want the last input to be zero

% Generate the matrix Q and the vector c (objecitve function weights in the QP problem)
Q1 = zeros(mx,mx);
Q1(1,1) = 2;                            % Weight on state x1
Q1(2,2) = 0;                            % Weight on state x2
Q1(3,3) = 0;                            % Weight on state x3
```

```
Q1(4,4) = 0;                            % Weight on state x4
P1 = 10;                                % Weight on input
Q = gen_q(Q1,P1,N,M);                                  % Generate Q, hint: gen_q
c = zeros(N*mx+M*mu,1);                                  % Generate c, this is the linear
constant term in the QP

%% Generate system matrixes for linear model
Aeq = gen_aeq(A1,B1,N,mx,mu);             % Generate A, hint: gen_aeq
beq = Aeq*z;% Generate b
beq(1) = x1_0;
%% Solve QP problem with linear model
tic
[z,lambda] = quadprog(Q,c,[],[],Aeq,beq,vlb,vub); % hint: quadprog. Type 'doc quadprog'
for more info
t1=toc;

% Calculate objective value
phi1 = 0.0;
PhiOut = zeros(N*mx+M*mu,1);
for i=1:N*mx+M*mu
  phi1=phi1+Q(i,i)*z(i)*z(i);
  PhiOut(i) = phi1;
end

%% Extract control inputs and states
u  = [z(N*mx+1:N*mx+M*mu);z(N*mx+M*mu)]; % Control input from solution

x1 = [x0(1);z(1:mx:N*mx)];              % State x1 from solution
x2 = [x0(2);z(2:mx:N*mx)];              % State x2 from solution
x3 = [x0(3);z(3:mx:N*mx)];              % State x3 from solution
x4 = [x0(4);z(4:mx:N*mx)];              % State x4 from solution

num_variables = 5/delta_t;
zero_padding = zeros(num_variables,1);
unit_padding  = ones(num_variables,1);

u   = [zero_padding; u; zero_padding];
x1  = [pi*unit_padding; x1; zero_padding];
x2  = [zero_padding; x2; zero_padding];
x3  = [zero_padding; x3; zero_padding];
x4  = [zero_padding; x4; zero_padding];

%% Plotting
t = 0:delta_t:delta_t*(length(u)-1);
ts_u = timeseries(u,t);

pitchplot1 = load('p_q_01.mat');
pdotplot1 = load('pdot_q_01.mat');
travelplot1 = load('travel_q_01.mat');
rplot1 = load('tr_q_01.mat');
```

```
pitchplot2 = load('p_q_1.mat');
pdotplot2 = load('pdot_q_1.mat');
travelplot2 = load('travel_q_1.mat');
rplot2 = load('r_q_1.mat');

pitchplot3 = load('p_q_10.mat');
pdotplot3 = load('pdot_q_10.mat');
travelplot3 = load('travel_q_10.mat');
rplot3 = load('r_q_10.mat');

pitch = pitchplot1.ans(2, :)*pi/180;
pdot = pdotplot1.ans(2, :)*pi/180;
travel = travelplot1.ans(2, :)*pi/180;
tr = rplot1.ans(2, :)*pi/180;

pitch2 = pitchplot2.ans(2, :)*pi/180;
pdot2 = pdotplot2.ans(2, :)*pi/180;
travel2 = travelplot2.ans(2, :)*pi/180;
tr2 = rplot2.ans(2, :)*pi/180;

pitch3 = pitchplot3.ans(2, :)*pi/180;
pdot3 = pdotplot3.ans(2, :)*pi/180;
travel3 = travelplot3.ans(2, :)*pi/180;
tr3 = rplot3.ans(2, :)*pi/180;

t3 = rplot1.ans(1, :);
t4 = rplot2.ans(1, :);
t5 = rplot3.ans(1, :);
```