```
% 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;
    0 0 -K_2 0 0 0;
    0 0 0 delta_t 0 0;
    0 0 -K_1*K_pp -K_1*K_pd 0 0;
    0 0 0 0 0 1;
    0 0 0 0 -K_3*K_ep -K_3*K_ed]*delta_t+eye(6);
B1 = [0 0;
    0 0;
    0 0;
    K_1*K_pp*delta_t 0;
    0 0;
    0 K_3*K_ep*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
x5_0 = 0;
x6_0 = 0;
x0 = [x1_0 x2_0 x3_0 x4_0 x5_0 x6_0]';% Initial values

% Time horizon and initialization
N  = 60;                                  % 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
for i=5:6:N*(mx)
    z0(i) = 0.1;
end

% 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

q1 = 10;
q2 = 10;
alpha = 0.2;
beta = 20;
lambda_t = 2*pi/3;
dt = 0.25;

% 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
Q1(5,5) = 0;
Q1(6,6) = 0;
P1 = diag([q1 q2]);                                 % Weight on input
Q = gen_q(Q1,P1,N,M);                                   % Generate Q, hint: gen_q
c = zeros(5*N, 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 = [A1*x0; zeros(6*(N-1), 1)];% Generate b
%% Solve QP problem with linear model
F_cost = @(x) 1/2*x'*Q*x;
opt = optimoptions('fmincon', 'Algorithm', 'sqp', 'MaxFunEvals', 400000);
tic
[z, lambda] = fmincon(F_cost, z0, [], [], Aeq, beq, vlb, vub, @(z)constraints(z), opt);
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

%% LQR

%Q_lqr = diag([15 25 15 25 15 25]);
% Q_lqr = diag([1 1 1 1 2 1]);
Q_lqr = diag([10 5 1 5 100 1]);
```

```
R_lqr = 1*diag([1 1]);
[K, S, e] = dlqr(A1, B1, Q_lqr, R_lqr);




%% Extract control inputs and states
u = [z(N*mx+1:2:N*mx+M*mu);z(N*mx+M*mu);z(N*mx+M*mu)];
u1  = [z(N*mx+1:2:N*mx+M*mu);z(N*mx+M*mu)]; % Control input from solution
u2  = [z(N*mx+2:2:N*mx+M*mu);z(N*mx+M*mu)];

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
x5 = [x0(5);z(5:mx:N*mx)];
x6 = [x0(6);z(6:mx:N*mx)];

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

u1   = [zero_padding; u1; zero_padding];
u2   = [zero_padding; u2; 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];
x5  = [zero_padding; x5; zero_padding];
x6  = [zero_padding; x6; zero_padding];




function [c, ceq] = constraints(z)
    N = 60;
    alpha = 0.2;
    beta = 20;
    lambda_t = 2*pi/3;
    dt = 0.25;
    c = zeros(N, 1);
    for k = 0:N-1
        c(k+1) = alpha*exp(-beta*(z(1+k*6)-lambda_t).^2)-z(5+k*6);
    end
    ceq = [];
end
```