```
clc;
clear;

% time-step and # of segments
tf = 20;
S.h = .05;
S.N = tf/S.h;
% system mass
S.m = 2;


% cost function specification
S.Q = 0.01*diag([1,1,.5,.5]);
S.R = diag([.1, .1]);
S.Pf = diag([1,1,5,5]);
S.f = @pnt_f;
S.L = @pnt_L;
S.Lf = @pnt_Lf;
S.mu = 1;
S.ko = 5;
%obstacle
S.os(1).p = [-2.5;-2.5];
S.os(1).r = 1;
S.ko = 5;

% initial state
x0 = [-5; -5; .3; 0];

% initial control
us = [repmat([.1;.05], 1, S.N/2), repmat(-[.1;.05], 1, S.N/2)]/2;

% added constraints of control
S.lb = [-1; -1];
S.ub = [1; 1];

xs = ddp_traj(x0, us, S);
V = ddp_cost(xs, us, S);

subplot(1,2,1)
plot(xs(1,:), xs(2,:), '-b')
hold on
if isfield(S, 'os')
    da = .1;
    a = -da:da:2*pi;
    for i=1:length(S.os)
    % draw obstacle
        plot(S.os(i).p(1) + cos(a)*S.os(i).r, S.os(i).p(2)+
 sin(a)*S.os(i).r, '-r','LineWidth',2);
    end
    axis equal
end
```

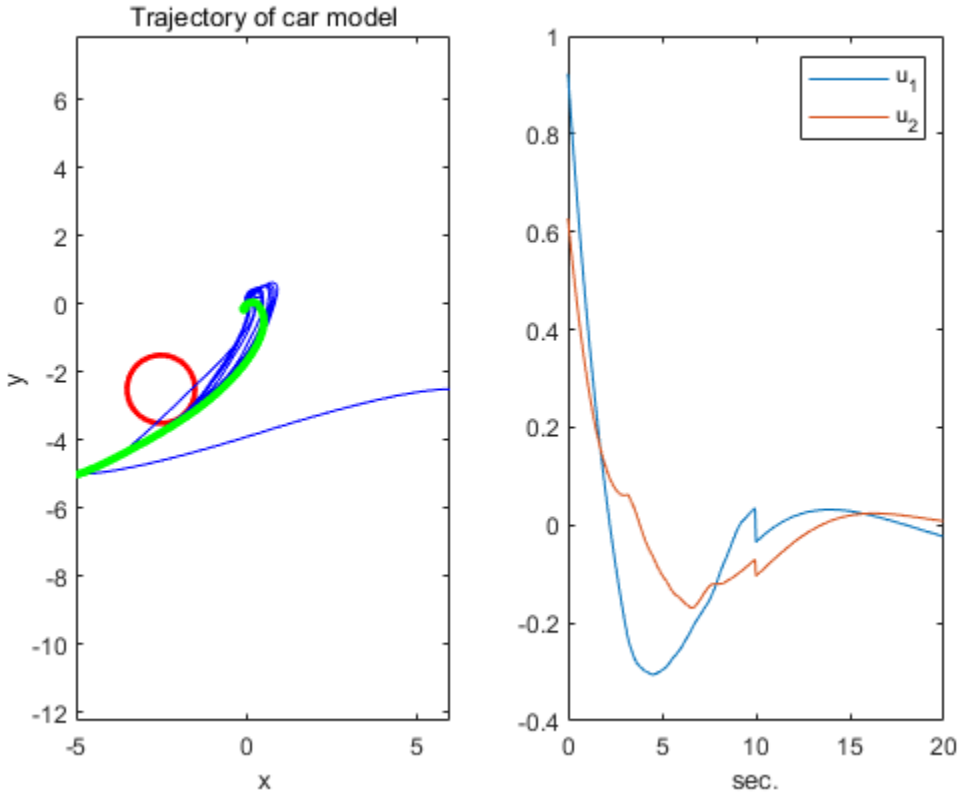
```
% two iteration should be enough for a linear-quadratic problem
for i=1:20
    [dus, V, Vn, dV, a] = ddp(x0, us, S);
    % update control
    us = us + dus;
    S.a = a;
    xs = ddp_traj(x0, us, S);
    plot(xs(1,:), xs(2,:), '-b');
    % S.ko = S.ko*10; % optionally obstacle gain be increased to
    % ensure strict avoidance
end

plot(xs(1,:), xs(2,:), '-g', 'LineWidth',3);
xlabel('x');
ylabel('y');
title(sprintf('Trajectory of car model'));
subplot(1,2,2)
plot(0:S.h:tf-S.h, us(1,:),0:S.h:tf-S.h, us(2,:));
xlabel('sec.')
legend('u_1','u_2')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%
function [L, Lx, Lxx, Lu, Luu] = pnt_L(k, x, u, S)
    if k < S.N+1
        L = S.h*(x'*S.Q*x/2 + u'*S.R*u/2);
        Lx = S.h*S.Q*x;
        Lxx = S.h*S.Q;
        Lu = S.h*S.R*u;
        Luu = S.h*S.R;
    else
        L = x'*S.Pf*x/2;
        Lx = S.Pf*x;
        Lxx = S.Pf;
        Lu = zeros(S.m,1);
        Luu = zeros(S.m,S.m);
    end

    % quadratic penalty term
    if isfield(S, 'os')
        for i=1:length(S.os)
            g = x(1:2) - S.os(i).p;
            c = S.os(i).r - norm(g);
            if c < 0
                continue
            end
            L = L + S.ko/2*c^2;
            v = g/norm(g);
            Lx(1:2) = Lx(1:2) - S.ko*c*v;
            Lxx(1:2,1:2) = Lxx(1:2,1:2) + S.ko*v*v';
        end
    end
end
```

```
function [x, fx, fu] = pnt_f(k, x, u, S)
    A = [eye(2), eye(2)*S.h;
    zeros(2), eye(2)];
    B = [zeros(2);
         S.h*eye(2)];
    x = A*x + B*u;
    fx = A;
    fu = B;
end
```



Published with MATLAB® R2021b