```matlab
% main_mc_mpc: Main script for Problem 3.1 and Problem 3.2 (a) and (c)
%
% --
% Control for Robotics
% Assignment 3
%
% --
% Technical University of Munich
% Learning Systems and Robotics Lab
%
% Course Instructor:
% Angela Schoellig
% angela.schoellig@tum.de
%
% Teaching Assistants:
% SiQi Zhou: siqi.zhou@tum.de
% Lukas Brunke: lukas.brunke@tum.de
%
% --
% Revision history
% [20.03.07, SZ] first version
% [22.03.02, SZ] second version
clear all
close all
clc
addpath(genpath(pwd));
%% General
% MPC parameters
n_lookahead = 100; % MPC prediction horizon
n_mpc_update = 1; % MPC update frequency
% Cost function parameters
Q = diag([100, 0]); % not penalizing velocity
r = 0;
% Initial state
cur_state = [-pi/6; 0]; % [-pi/6; 0];
goal_state = [0.5; 0.05];
state_stack = cur_state;
input_stack = [];
% State and action bounds
pos_bounds = [-1.2, 0.5]; % state 1: position
vel_bounds = [-0.07, 0.07]; % state 2: velocity
acc_bounds = [-1, 1]; % action: acceleration
% Plotting parameters
linecolor = [1, 1, 1].*0.5;
fontcolor = [1, 1, 1].*0.5;
fontsize = 12;
% Max number of time steps to simulate
max_steps = 500;
% Standard deviation of simulated Gaussian measurement noise
noise = [1e-3; 1e-5];
% Set seed
rng(0);
% Use uncertain parameters (set both to false for Problem 3.1)
```

```
use_uncertain_sim = false;
use_uncertain_control = false;
% Result and plot directory
save_dir = './results/';
mkdir(save_dir);
% If save video
save_video = false;
load("nonlin_opt.mat");
load('cur_state.mat');
load('mu_lr.mat');
%% Solving mountain car problem with MPC
% State and action bounds
state_bound = [pos_bounds; vel_bounds];
action_bound = [acc_bounds];
% Struct used in simulation and visualization scripts
world.param.pos_bounds = pos_bounds;
world.param.vel_bounds = vel_bounds;
world.param.acc_bounds = acc_bounds;
% Action and state dimensions
dim_state = size(state_bound, 1);
dim_action = size(action_bound, 1);
% Video
if save_video
      video_hdl = VideoWriter('mpc_visualization.avi');
open(video_hdl);
end
% MPC implmentation
tic;
for k = 1:1:max_steps
if mod(k, n_mpc_update) == 1 || n_mpc_update == 1
fprintf('updating inputs...\n');
% Get cost Hessian matrix
S = get_cost(r, Q, n_lookahead);
% Lower and upper bounds
% 3nx1 vector for state and input constraints
lb = [repmat(action_bound(1),n_lookahead,1); ...
repmat(state_bound(:,1),n_lookahead,1)];
ub = [repmat(action_bound(2),n_lookahead,1); ...
repmat(state_bound(:,2)+[0.5;0],n_lookahead,1)];
% Optimize state and action over prediction horizon
% if k <= 1
if false
% Solve nonlinear MPC at the first step
if k == 1
initial_guess = randn(n_lookahead*(dim_state+dim_action), 1);
else
initial_guess = x;
end
% Cost function
sub_states = [repmat(0,n_lookahead,1); ...
repmat(goal_state, n_lookahead,1)];
fun = @(x) (x - sub_states)'*S*(x - sub_states);
% Temporary variables used in 'dyncons'
```

```
save('params', 'n_lookahead', 'dim_state', 'dim_action');
save('cur_state', 'cur_state');
% Solve nonlinear MPC
% x is a vector containing the inputs and states over the
% horizon [input,..., input, state', ..., state']^T
% Hint: For Problem 3.1 (b) and 3.2 (c), to make it easier to
% debug the QP implementation, you may consider load the
% nonlinear optimization solution 'nonlin_opt' or
% 'nonlin_opt_uncert' instead of recomputing the trajectory
% everytime running the code. The optimization needs to run
% once initially and rerun if the time horizon changes.
options = optimoptions(@fmincon, 'MaxFunctionEvaluations', ...
1e5, 'MaxIterations', 1e5, 'Display', 'iter');
if ~use_uncertain_control
[x,fval] = fmincon(fun, initial_guess, [], [], [], [], ...
lb, ub, @dyncons, options);
save('nonlin_opt', 'x', 'fval');
else
[x,fval] = fmincon(fun, initial_guess, [], [], [], [], ...
lb, ub, @dyncons_uncert, options);
save('nonlin_opt_uncert', 'x', 'fval');
end
else
% ================== [TODO] QP Implementation =================
% Problem 3.1 (b): Quadratic Program optimizing state and
% action over prediction horizon
% Problem 3.2 (c): Update the QP implementation using the
% identified system parameters. You can use the boolean
% variable 'use_uncertain_control' to switch between the two
% cases.
% feedback state used in MPC updates
% 'cur_state' or 'cur_state_noisy'
cur_state_mpc_update = cur_state;
% Solve QP (e.g., using Matlab's quadprog function)
% Note 1: x is a vector containing the inputs and states over
% the horizon [input,..., input, state', ..., state']^T
% Note 2: The function 'get_lin_matrices' computes the
% Jacobians (A, B) evaluated at an operation point
% quadprog(H,f,A,b,Aeq,beq,lb,ub)
u = x(1:n_lookahead*dim_action);
state = x(n_lookahead*dim_action+1:end);
u = [u(2:end); u(end)]; % Kick the first term out and added the last term
again to the bottom
state = [state(3:end); state(end-1:end)];
x = [u; state];
% u_ap = repmat(x(1), n_lookahead, 1);
% x_ap = repmat(cur_state_mpc_update, n_lookahead, 1);
% [A, B] = get_lin_matrices(x_ap,u_ap);
% x = x - [u_ap;x_ap];
A = cell(1,n_lookahead);
B = cell(1,n_lookahead);
for i=1:n_lookahead
u_lin = x(i);
```

```
x_lin = x(n_lookahead+2*i-1:n_lookahead+2*i);
[A{i}, B{i}] = get_lin_matrices(x_lin,u_lin);
end
f1 = -B{i};
f2 = [];
f3 = eye(2);
for i = 1:1:n_lookahead-1
f1 = blkdiag(f1, -B{i+1});
f2 = blkdiag(f2, -A{i});
f3 = blkdiag(f3, eye(2));
end
f2 = cat(1, zeros(2, 2*n_lookahead-2), f2);
f2 = cat(2, f2, zeros(2*n_lookahead, 2));
A_eq = [f1, (f2+f3)];
b_eq = zeros(size(A_eq, 1), 1);
A_0 = get_lin_matrices(cur_state_mpc_update,0);
b_eq(1:2,1) = A_0*cur_state_mpc_update;
x_cur = x;
f = 2*S*(x_cur-x);
x = quadprog(4*S, f, [], [], A_eq, b_eq, lb, ub);
x = x + x_cur;
% ==============================================================
end
% Separate inputs and states from the optimization variable x
inputs = x(1:n_lookahead*dim_action);
states_crossterms = x(n_lookahead*dim_action+1:end);
position_indeces = 1:2:2*n_lookahead;
velocity_indeces = position_indeces + 1;
positions = states_crossterms(position_indeces);
velocities = states_crossterms(velocity_indeces);
% Variables if not running optimization at each time step
cur_mpc_inputs = inputs';
cur_mpc_states = [positions'; velocities'];
end
% Propagate
action = cur_mpc_inputs(1);
      if ~use_uncertain_sim
[cur_state, cur_state_noisy, ~, is_goal_state] = ...
one_step_mc_model_noisy(world, cur_state, action, noise);
else
[cur_state, cur_state_noisy, ~, is_goal_state] = ...
one_step_mc_model_uncert(world, cur_state, action, noise);
end
% Remove first input
cur_mpc_inputs(1) = [];
cur_mpc_states(:,1) = [];
% Save state and input
state_stack = [state_stack, cur_state];
input_stack = [input_stack, action];
% Plot
grey = [0.5, 0.5, 0.5];
hdl = figure(1);
hdl.Position(3) = 1155;
```

```
clf;
subplot(3,2,1);
      plot(state_stack(1,:), 'linewidth', 3); hold on;
plot(k+1:k+length(cur_mpc_states(1,:)), cur_mpc_states(1,:), 'color', grey);
ylabel('Car Position');
set(gca, 'XLim', [0,230]);
set(gca, 'YLim', pos_bounds);
subplot(3,2,3);
plot(state_stack(2,:), 'linewidth', 3); hold on;
plot(k+1:k+length(cur_mpc_states(2,:)), cur_mpc_states(2,:), 'color', grey);
ylabel('Car Velocity');
      set(gca, 'XLim', [0,230]);
set(gca, 'YLim', vel_bounds);
subplot(3,2,5);
plot(input_stack(1,:), 'linewidth', 3); hold on;
plot(k:k+length(cur_mpc_inputs)-1, cur_mpc_inputs, 'color', grey);
xlabel('Discrete Time Index');
ylabel('Acceleration Cmd');
set(gca, 'XLim', [0,230]);
set(gca, 'YLim', acc_bounds);
subplot(3,2,[2,4,6]);
      xvals = linspace(world.param.pos_bounds(1),
world.param.pos_bounds(2));
      yvals = get_car_height(xvals);
      plot(xvals, yvals, 'color', linecolor, 'linewidth', 1.5); hold on;
plot(cur_state(1), get_car_height(cur_state(1)), 'ro', 'linewidth', 2);
axis([pos_bounds, 0.1, 1]);
xlabel('x Position');
ylabel('y Position');
axis([world.param.pos_bounds, min(yvals), max(yvals) + 0.1]);
pause(0.1);
% Save video
if save_video
frame = getframe(gcf);
writeVideo(video_hdl, frame);
end
% Break if goal reached
if is_goal_state
fprintf('goal reached\n');
break
end
end
compute_time = toc;
% Close video file
if save_video
close(video_hdl);
end
% Visualization
plot_visualize = false;
plot_title = 'Model Predictive Control';
hdl = visualize_mc_solution_mpc(world, state_stack, input_stack, ...
plot_visualize, plot_title, save_dir);
% Save results
```

```
save(strcat(save_dir, 'mpc_results.mat'), 'state_stack', 'input_stack');
```