Assignment 3

Problem(1): Code-

3.1 a)

prediction horizon: N

$$J^*_k(x_k) = \min_{u_{k|k} \dots u_{k+N-1|k}} (x_N - x_g)^T Q (x_N - x_g) + \sum_{k=0}^{N-1} (x_k - x_g)^T Q (x_k - x_g) + \gamma a_k^2$$

Note $x_k = [p_k, v_k]^T$

subject to constraints :-

$$x_{k+i+1|k} = f(x_{k+i|k}, u_{k+i|k})$$

↳ simplifies to:

$$p_{k+i+1|k} = p_{k+i|k} + v_{k+i+1}$$

$$v_{k+i+1|k} = v_{k+i|k} + 0.001\, a_{k+i|k} - 0.0025 \cos(3\, p_{k+i|k})$$

$$x_{k|k} = x_k.$$

(Note terms can be further rearranged ($p_{k+i+1|k}$)

$$x_g = \begin{bmatrix} 0.5 \\ 0.05 \end{bmatrix} \quad x_0 = \begin{bmatrix} -\pi/6 \\ 0 \end{bmatrix}$$

3.1 b)

Note: Simulation results in appendix:

```
% main_mc_mpc: Main script for Problem 3.1 and Problem 3.2 (a) and (c)
%
% --
% Control for Robotics
% AER1517 Spring 2022
% Assignment 3
%
% --
% University of Toronto Institute for Aerospace Studies
% Dynamic Systems Lab
%
% Course Instructor:
% Angela Schoellig
% schoellig@utias.utoronto.ca
%
% Teaching Assistant:
% SiQi Zhou
% siqi.zhou@robotics.utias.utoronto.ca
% Lukas Brunke
% lukas.brunke@robotics.utias.utoronto.ca
% Adam Hall
% adam.hall@robotics.utias.utoronto.ca
%
% --
% 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 = 110; % 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 = 100;

curr_state_stack =[];
% 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;

%% 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;

flag =0;
load_prev = 0;
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
        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
```

```
    % 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.




    if load_prev == 0 && flag == 0
        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
    x_prev = x;
    end

    if load_prev == 1

    load('nonlin_opt')
    x_prev = x;
    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;
    %%
    % formulate objective function
    S_quad = get_cost_quad_prog(r, Q, n_lookahead);
```

```
        %%
        %% formulate constraints
        % Dynamics Constraint
            % Compute Aeq, Beq

        [Aeq,beq,c] =  compute_constraint_matrices(x_prev,cur_state);

        save('constraint_matrices', 'Aeq', 'beq');
        % Actuation & state constraints (upper and lower bounds)
        %
        %%

        % 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


        % x = ...;
        A_ineq = [];
        b_ineq = [];
        H = S_quad;

        f = zeros(3*n_lookahead-1,1);
        lb = [repmat(action_bound(1),n_lookahead-1,1); ...
        repmat(state_bound(:,1),n_lookahead,1)];

        ub = [repmat(action_bound(2),n_lookahead-1,1); ...
        repmat(state_bound(:,2)+[0.5;0],n_lookahead,1)];

        x = quadprog(H,f,A_ineq,b_ineq,Aeq,beq,lb,ub);
        % ============================================================
    end

    % Separate inputs and states from the optimization variable x
    inputs = x(1:n_lookahead*dim_action -1);
    states_crossterms = x(n_lookahead*dim_action: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
curr_state_stack = [curr_state_stack,cur_state];

% Remove first input
%     cur_mpc_inputs(1) = [];
%     cur_mpc_states(:,1) = [];
x(1) = action;
x(n_lookahead) = cur_state(1);
x(n_lookahead+1) = cur_state(2);
flag = 1;
```

```
    load_prev = 1;
    cur_mpc_inputs(1) = action;
    cur_mpc_states(:,1) = cur_state;
    % 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');
```

```
% compute_constraint_matrices: Function defninig nonlinear system dynamics
constraints
%          (used in fmincon)
%
% Inputs:
%       x:               A vector of decision variables
%                        [input,...,input, state', ..., state']^T
%
% Outputs:
%       c:               Nonlinear inequality constraints
%       ceq:             Nonlinear equality constraints
%
% --
% Control for Robotics
% AER1517 Spring 2022
% Assignment 3
%
% --
% University of Toronto Institute for Aerospace Studies
% Dynamic Systems Lab
%
% Course Instructor:
% Angela Schoellig
% schoellig@utias.utoronto.ca
%
% Teaching Assistant:
% SiQi Zhou
% siqi.zhou@robotics.utias.utoronto.ca
% Lukas Brunke
% lukas.brunke@robotics.utias.utoronto.ca
% Adam Hall
% adam.hall@robotics.utias.utoronto.ca
%
% --
% Revision history
% [20.03.07, SZ]    first version
% [22.03.02, SZ]    second

function [Aeq,beq,c] = compute_constraint_matrices(x,cur_state)
    % Load parameters and current state
    addpath(genpath(pwd));
    load('params');
    load('cur_state');

    % Extract input and state from the vector x
    INPUTS = x(1:n_lookahead*dim_action);
    STATES_CROSSTERMS = x(n_lookahead*dim_action+1:end);
%     odd_idx = 1:2:2*n_lookahead;
%     even_idx = odd_idx + 1;
%     POSITIONS = STATES_CROSSTERMS(odd_idx);
%     VELOCITIES = STATES_CROSSTERMS(even_idx);
```

```
    A_bar = [];
    B_bar = [];
    w = zeros(n_lookahead*2, 1);

    for k = 1:n_lookahead-1

    state_k = [STATES_CROSSTERMS(2*k-1);STATES_CROSSTERMS(2*k)];
    input_k = INPUTS(k);
    %get linearized matrices
    [A_k, B_k] = get_lin_matrices(state_k,input_k);
    %compute w (some terms arising due to linearization at each timesteps)
    x_next_bar = [STATES_CROSSTERMS(2*k+1);STATES_CROSSTERMS(2*k+2)];
    w_k = x_next_bar - A_k*state_k - B_k*input_k;
    w(2*k + 1) = w_k(1);
    w(2*(k+1)) = w_k(2);

    A_bar = blkdiag(A_bar, A_k); %diagonal matrix with A_k as main diagonal
    B_bar = blkdiag(B_bar, B_k); %diagonal matrix with B_k as main diagonal
    end
    % Define equality constraints
    % Have to add additional zeros in the block diagonal matrix
    %ReCompute A_bar
    zero1 = zeros(2*n_lookahead -2 ,2);
    A_bar = [A_bar zero1];
    zero2 = zeros(2,2*n_lookahead);
    A_bar = [zero2;A_bar];
    %done!

    %Compute B_bar
    zero3 = zeros(2,n_lookahead-1);
    B_bar = [zero3;B_bar]

    %compute H_bar
    I = eye(2);
    zero4 = zeros(2*n_lookahead-2,2);
    H_bar = [I;zero4];
    I_bar = eye(2*n_lookahead);
    % Inequality constraints
    Aeq = [-B_bar, (I_bar-A_bar)];
    %beq step 1 computation
    beq1 = H_bar*[cur_state(1); cur_state(2)];
    beq = beq1 + w;
    %beq step 2 computation:


    c = [];
end
```

Assignment 3

2. With shorter lengths of prediction horizon, the robot was not able to reach the peak point.

3. The MPC controller performs poorly when the model deviates from the actual model.

Note: Plots, simulation results in appendix.

Assignment 3

3.2

a) Simulation results in appendix:

b)

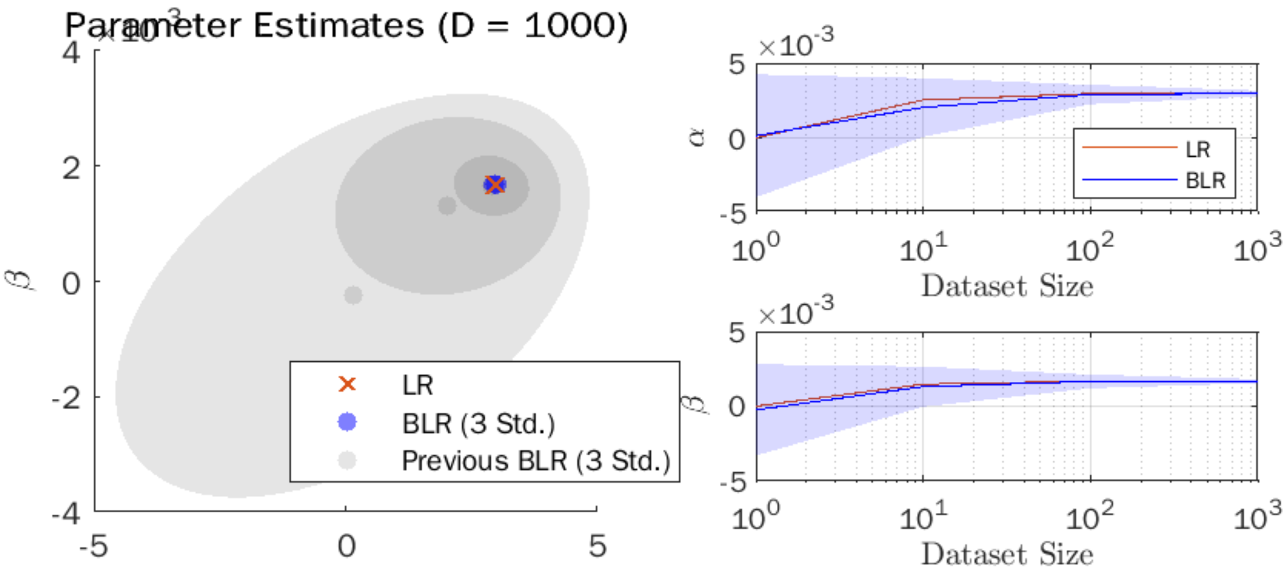
Problem(2): Code-

```
% main_system_id: Main script for Problem 3.2 (b)
%
% --
% Control for Robotics
% AER1517 Spring 2022
% Assignment 3
%
% --
% University of Toronto Institute for Aerospace Studies
% Dynamic Systems Lab
%
% Course Instructor:
% Angela Schoellig
% schoellig@utias.utoronto.ca
%
% Teaching Assistant:
% SiQi Zhou
% siqi.zhou@robotics.utias.utoronto.ca
% Lukas Brunke
% lukas.brunke@robotics.utias.utoronto.ca
% Adam Hall
% adam.hall@robotics.utias.utoronto.ca
%
% --
% Revision history
% [20.03.07, SZ]    first version
% [22.03.02, SZ]    second version

clear all
close all
clc

addpath(genpath(pwd));

%% General
% Result and plot directory
save_dir = './results/';
mkdir(save_dir);

%% Parameter estimation
data_size = [1, 10, 100, 1000];
theta_blr_est = [];
theta_cov_blr_est = [];
theta_lr_est = [];

for D = data_size
    % Generate ID data

    [id_data] = generate_param_iddata(D);

    % ================== [TODO] Parameter Estimation =================
```

```
    % Problem 3.2 (b): Write a 'param_id' function that computes the
    % parameter estimates based on the identification dataset 'id_data'.
    % The structure 'id_data' contains the input-output data generated by
    % the uncertain mountain car environment {state_cur, input_cur,
    % state_nxt}.
    [mu_lr, mu_blr, cov_blr] = param_id(id_data)

    % Record estimation results
    theta_lr_est = [theta_lr_est, mu_lr];
    theta_blr_est = [theta_blr_est, mu_blr];
    theta_cov_blr_est{size(theta_blr_est, 2)} = cov_blr;

    % Plot estimates
    hdl = figure(1);
    hdl.Position(3) = 1155;
    plot_param_est(data_size, theta_lr_est, theta_blr_est, theta_cov_blr_est);

    % Print results
    fprintf('\nD = %d\n', D);
    fprintf('LR: %.4f, %.4f\n', mu_lr(1), mu_lr(2));
    std_blr = sqrt(diag(cov_blr));
    fprintf('BLR: %.4f, %.4f\n', mu_blr(1), mu_blr(2));
    fprintf('BLR Uncertainty: %.4f, %.4f\n', std_blr(1), std_blr(2));
    pause(1);
%     waitforbuttonpress;
end

% Save plot
saveas(gcf, strcat(save_dir, 'Parameter Estimation.png'));


Function (parameter estimation):

function [mu_lr, mu_blr, cov_blr] = param_id(id_data)

x = id_data.state_cur;
u = id_data.input_cur;
x_plus = id_data.state_nxt;

% mu_lr = x;
% mu_blr = u;
% cov_blr = x_plus;

n1 = size(x,2);
%compute v_prior
v_prior = zeros(1,n1);
phi = zeros(2,2);
phi_stack = []
Tau = [];
for i = 1:n1
%p_k = x(1,i)
%v_k = x(2,i)
p_prior(i) = x(1,i) + x(2,i);
v_prior(i) = x(2,i);
tau_v = x_plus(2,i) - v_prior(i); %v_k+1 - v_prior
tau_p = x_plus(1,i) - p_prior(i); %p_k+1 -p_prior
tau = [tau_p;tau_v];
Tau = [Tau;tau]; %compute Tau
phi(1,1) = u(i);
phi(1,2) = -cos(3*x(1,i));
phi(2,1) = u(i);
phi(2,2) = -cos(3*x(1,i));
phi_stack = [phi_stack;phi];
end

mu_lr = inv(phi_stack'*phi_stack)*phi_stack'*Tau; %linear regression
```

```
% Bayesian linear regression:
%compute Sigma_hat_theta_inv
sigma = 0.0015;
Sigma_zero = (sigma^2)*eye(2);
Sigma_hat_theta_inv = inv(Sigma_zero) + (sigma^(-2))*phi_stack'*phi_stack
Sigma_hat_theta = inv(Sigma_hat_theta_inv);
mu_hat_theta = Sigma_hat_theta*(sigma^(-2)*phi_stack'*Tau);

mu_blr = mu_hat_theta;
cov_blr = Sigma_hat_theta;

end
```

3.



BLR

As the size of data set increases, both approaches converge to better parameter estimate values.

LR is a point estimate, while BLR is computes a distribution. This would be advantageous if the data is too noisy, as the point estimate can be false in that case.