Control for Robotics: From Optimal Control to Reinforcement Learning

Assignment 3 Model Predictive Control and System Identification

General Information

Due date:	You can find the due date in the syllabus handout. Your solution must be submitted before 23h59 on the due date.
Submission:	Please submit your solution and the requested Matlab scripts (highlighted in `blue`) as a single PDF document. Both typed and scanned handwritten solutions are accepted. It is your responsibility to provide enough detail such that we can follow your approach and judge your solution. Students may discuss assignments. However, each student must code up and write up their solutions independently. We will check for plagiarism. The points for each question are shown in the left margin.

Introduction

In this assignment, we will implement model predictive control (MPC) to solve a standard benchmark problem in control and reinforcement learning, the mountain car [1]. This assignment consists of two marked problems. In the first problem, we assume that a model of the system is given and we will use MPC to solve the problem. In the second problem, we examine how model uncertainty can impact the solution of the MPC and explore two offline parameter identification approaches for closed-loop control.

Problem 3.1 Mountain Car

In this problem, we consider the mountain car problem that was initially proposed in [1]. The goal of the mountain car problem is to drive an under-powered car to the top of a hill (see Figure 1). Since the car is under-powered, it cannot drive straight up to the top of the hill; instead, it must swing back and forth to gain enough momentum to climb up the hill. We consider a two-dimensional state with position p and velocity v. The position and velocity of the car are bounded between $[-1.2, 0.5]$ and $[-0.07, 0.07]$, respectively. The car's input is the acceleration a, which is bounded between $[-1, 1]$. The dynamics of the car is given by

$$v_{k+1} = v_k + 0.001a_k - 0.0025\cos(3p_k), \quad p_{k+1} = p_k + v_{k+1}, \tag{1}$$

where k is the discrete-time index. Whenever the car reaches the position limits, it comes to an immediate stop and its velocity is set to zero so that it remains there indefinitely. The initial state and the goal state of the system are $[-\pi/6,\, 0]^T$ and $[0.5,\, 0.05]^T$, respectively. We will solve the mountain car problem with MPC in this assignment and use a classical RL approach in the next assignment.

To solve the mountain car problem with MPC, we first formulate it as an optimal control problem with the following cost function:

$$\min_{a_1,..,a_{T-1}} \quad (x_T - x_g)^T Q (x_T - x_g) + \sum_{k=0}^{T-1} (x_k - x_g)^T Q (x_k - x_g) + r a_k^2, \tag{2}$$

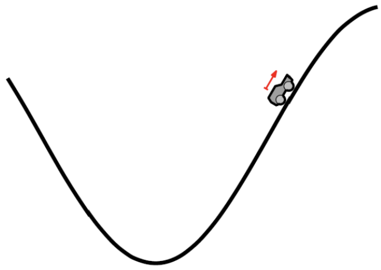

Figure 1: The mountain car problem [2]: The goal is to drive an under-powered car to the top of the hill. Since the car is under-powered, it does not have enough power to directly drive to the goal position at the right edge. Instead, it must build up momentum by swinging back and forth.

where $x_k = [p_k,\ v_k]^T$ is the state, a_k is the input, x_g is the goal state, Q is a symmetric positive semi-definite matrix, r is a non-negative scalar, and T is the length of the trajectory. Note that r is typically required to be strictly positive to penalize large inputs; however, here we will set r to zero and explicitly limit the values of the inputs as constraints in our MPC formulation.

15 (a) Assuming a prediction horizon of N timesteps, write out the MPC optimization problem using the nonlinear system dynamics given in (1).

35 (b) We obtain a nonlinear MPC optimization problem. Such problems are typically difficult to solve in real-time (using, for example, a standard nonlinear optimization solver). To speed up the computation, we can instead use a sequential quadratic programming (SQP) approach to solve the optimization problem at each time step more efficiently. The idea is similar to the ILQC implementation in Assignment 2. Here, we assume that we solve the nonlinear optimization problem only once at the initial time step and rollout a sequence of predicted states $\{\bar{x}_{k+r}\}_{r=1}^{N}$ and inputs $\{\bar{a}_{k+r}\}_{r=0}^{N-1}$ over the prediction horizon. At each subsequent time step k, we linearize the system dynamics and quadratize the cost around the predicted trajectory from the previous time step (i.e., $\{\bar{x}_{k+r}\}_{r=1}^{N}$ and $\{\bar{a}_{k+r}\}_{r=0}^{N-1}$). Define $\delta x_{k+r} = x_{k+r} - \bar{x}_{k+r}$ and $\delta a_{k+r} = a_{k+r} - \bar{a}_{k+r}$, formulate the Quadratic Program (QP) to be solved at each time step. Complete the implementation in main_mc_mpc.m and run the main script. Note that, in typical SQP, we usually solve the QP multiple times in each time step until the solution converges or a maximum number of iterations is reached; for this problem, it suffices to solve the QP once at each time step.

   (i) Report the simulation results and comment on the behaviour of the MPC controller. Both the main script `main_mc_mpc.m` and any added helper functions should be submitted.

   (ii) What effect do different lengths of the prediction horizon have? Explain your observations.

   (iii) Test the MPC controller with noisy measurements. To do so, change `cur_state_mpc_update` in `main_p1_mc_mpc.m` from `cur_state` to `cur_state_noisy`. Try different values of the additive noise standard deviation specified by the array `noise`. Is the car still able to reach the goal state? Explain your observations.

Problem 3.2 Mountain Car with Unknown Parameters

Now suppose that the dynamics of the mountain car has changed. We only know that the dynamics of the system has the following form:

$$v_{k+1} = v_k + \alpha\, a_k - \beta\, \cos(3p_k), \quad p_{k+1} = p_k + v_{k+1}, \tag{3}$$

where α and β are two unknown parameters, and the other variables are the same as in Problem 3.1.

5 (a) Test your solution from Problem 3.1 using the updated mountain car dynamics model by setting the boolean variable `use_uncertain_sim` in the main script `main_mc_mpc.m` to `true`. The noise parameters and prediction horizon parameter should be set to the default values given in the starter code. Report and comment on the simulation results.

35 (b) The given Matlab function `generate_param_iddata.m` generates the input-output datasets $\mathcal{D} = \{x^{(d)}, u^{(d)}, x_+^{(d)}\}_{d=1}^{D}$ from the uncertain mountain car environment, where the superscript (d) denotes samples from the offline dataset with $(x^{(d)}, u^{(d)})$ being a state-input pair at a particular time step and $x_+^{(d)}$ being the corresponding next state, and D is the dataset length. We will identify the parameters α and β using two approaches: *(i)* Linear Regression (LR) that leads to a single-point parameter estimate and *(ii)* Bayesian Linear Regression (BLR) that leads to a parameter distribution estimate. The main script for this part is `main_system_id.m`. You are expected to write a function `[mu_rl, mu_blr, cov_blr] = param_id(id_data)` that accepts the system identification data structure and outputs the parameter estimates and the covariance matrix.

   (i) Formulate the problem as LR. In particular, define the input, output, and the basis functions of the LR model that has α and β as the parameters to be estimated. Write down the objective function and find the optimal parameters (`mu_rl`) as the maximum likelihood estimate.

   (ii) Formulate the problem as BLR. Assume that the prior distribution of the parameters has a zero mean and a diagonal covariance $\text{diag}(\sigma^2, \sigma^2)$. Set $\sigma = 0.0015$ and find the optimal parameters (`mu_blr`) and the corresponding covariance matrix (`cov_blr`).

   (iii) Submit the plot generated by the main script `main_system_id.m` and provide the values of `mu_rl`, `mu_blr`, and `cov_blr` for $D = 1000$. Compare the quality of the estimates using the two approaches above as the size of the dataset varies. What is an advantage of the BLR approach as compared to the LR approach?

   (iv) Comment on the impact of the prior parameter distribution on the posterior estimate using the BLR approach. Support your conclusions with empirical results.

10 (c) Set the boolean variable `use_uncertain_control` to `true` and modify your MPC implementation correspondingly using the parameters identified in part (b). Use either `mu_rl` or `mu_blr` estimated from the largest dataset. Test the controller in the uncertain mountain car environment by running the main script `main_mc_mpc.m`.

References

[1] Andrew William Moore. *Efficient Memory-Based Learning for Robot Control*, 1990. Technical Report UCAM-CL-TR-209, University of Cambridge. Accessed on: Mar. 08, 2020. [Online]. Available: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-209.pdf.

[2] Jonas Buchli. *Course on Optimal and Learning Control for Autonomous Robots*, April 2015. Course Number 151-0607-00L, Swiss Federal Institute of Technology in Zurich (ETH Zurich). Accessed on: Mar. 07, 2020. [Online]. Available: http://www.adrlab.org/doku.php/adrl:education:lecture:fs2015.