Assignment 3

Path Integral Policy Improvement

General Information

Due date:	You can find the due date in the syllabus handout, on *Gradescope*, and in the course calendar. Your solution must be submitted on *Gradescope* before 23h59.
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 Assignment 1, we designed an ILQC controller that was able to control a quadrotor to perform agile maneuvers such as flying to a defined point and/or passing through some predefined via-points. ILQC is a model-based optimal control approach, which solves both the trajectory and the controller design in a single problem. In Assignment 1, we assumed complete knowledge about the system model, without any uncertainty in the system parameters. In general, model-based algorithms such as ILQC show superior performance in simulation, but their performance in real physical applications is limited by the accuracy of the system model. In fact, our commonly used system models are usually intentionally simple. This is done to keep the complexity and the dimensions of the state-space low while maintaining useful mathematical properties. These limitations typically restrict the performance of model-based methods on real robots.

Another approach to simultaneously finding reference trajectories and controllers is to use model-free machine learning techniques such as the Path Integral Policy Improvement methods presented in [1]. However, major drawbacks of existing learning algorithms are that their success highly depends on the quality of the initial guess since they are all local methods. They only partially exploit the domain knowledge of the system designer—i.e., a known system model.

In this assignment[1], we will combine the benefits of model-based and model-free approaches. In particular, we will adapt the model-based ILQC optimal controller derived in Assignment 1 and use the Path Integral Policy Improvement algorithm (PI2) (Algorithm 10 in [1]) to account for model uncertainties.

Problem 3.1 Agile Quadrotor Maneuvers with PI2

We use the same quadrotor platform as in Assignment 1. Details about the system description, state, inputs, and dynamics can be found in the Assignment 1 handout. The main script is `main_p1_pi2.m` in the

[1] This assignment is adapted from [2].

main directory. It outlines the main steps of the PI2 learning algorithm design. As detailed below, your task is to complete the PI2 algorithm in `PI2_Update.m`.

Connecting ILQC with PI2

In Assignment 1, you completed the functions `[Initial_Controller, Cost_LQR] = LQR_Design(Model, Task)` and `ILQC_Design(Model, Task, Initial_Controller, @Quad_Simulator)`, which calculate the LQR and ILQC controllers for the specific problem. In the first part of this assignment, use your own LQR and ILQC controller implementations and the nominal quadrotor model to generate a suitable initial guess for the learning algorithm. As a backup, we provide protected versions of the LQR and ILQC implementations; however, we strongly encourage you to use your own design.

PI2 Update Function Implementation

Now, we put aside the assumption of complete model knowledge. The 'real' robotic system is now simulated by a model that is perturbed in the parameters. In the learning part, those true system parameters are unknown to the algorithm, however, it can draw samples from the simulator imitating the 'real' robot.

5 (a) Complete the implementation of the General PI2 algorithm (Algorithm 10 in [1]). The protected function `[LearnedController,..., ...] = PIs_Learning(Model_perturbed, Task, ReducedController)` provides the framework for the algorithm, including the initialization, the exploration-noise annealing, and drawing samples from the perturbed system. However, the function `delta_theta = PI2_Update(Task, batch_sim_out, batch_cost)` must be completed. It is called by `PIs_Learning` and includes the missing part of the PI2 algorithm described in [1]. For convenience, the missing part to be implemented for this assignment is outlined in Algorithm 1 of this handout.

Note: The provided script contains the code for calculating the exponentiated cost, as well as the two additional functions `vec2mat(vec)` and `mat2vec(mat)`. The latter two functions are provided because the quadrotor simulator requires the parameter matrix $\boldsymbol{\theta}$ to be given in a different shape than in Algorithm 10 in the lecture notes. Therefore, conversions are required at the beginning and at the end of the function you are to implement. For example, calling something like `temp = sim_out.Controller.BaseFnc(sim_out.t,sim_out.x)` will give a vector-like parameter representation, which first needs to be converted by calling `vec2mat(temp)`. Similarly, at the end of your implementation, your `delta_theta` needs to be converted back to the vector-like representation by calling `mat2vec(delta_theta)`.

5 (b) Based on your implementation, answer the following questions:

(i) How much cost improvement did you obtain using PI2 learning? Please answer in one sentence and attach one of your cost-plots.

(ii) How does the exploration noise (`Task.std_noise`) affect the learning curve? What happens if you decrease/increase it?

(iii) The tuning parameter `Task.num_reuse` specifies how many (of the best) rollouts are saved, carried over, and reused in the next learning iteration. Why does it make sense to keep some of the best rollouts for the next update?

(iv) How does the quality of your initial guess affect the PI2 learning? For example, what happens if you limit your ILQC iterations to only one?

(v) While executing your program, you might have noticed that the cost is not always strictly decreasing during learning. What is your explanation for this behaviour?

Algorithm 1 Excerpt of General PI2 Algorithm [1]

for the ith control input **do**
  for each time s **do**
    Calculate the Return starting from s for the kth rollout
      $R(\tau^k(s)) = \Phi(\mathbf{x}(t_f)) + \int_s^{t_f} \left(q(t,\mathbf{x}), \frac{1}{2}\mathbf{u}^T\mathbf{R}\mathbf{u}\right)\, dt$
    Calculate α starting from s for the kth rollout
      $\alpha^k(s) = \exp\left(-\frac{1}{\lambda}R(\tau^k(s))\right) / \sum_{k'=1}^{K}\ R(\tau^{k'}(s))$
    Calculate the time-varying parameter increment $\Delta\boldsymbol{\theta}_i(s)$
      $\Delta\boldsymbol{\theta}_i(s) = \sum_{k=1}^{K}\ \alpha^k(s)\frac{\boldsymbol{\Upsilon}(s)\boldsymbol{\Upsilon}(s)^T}{\boldsymbol{\Upsilon}(s)^T\boldsymbol{\Upsilon}(s)}\boldsymbol{\epsilon}_i^k(s)$
  end for

  for the jth column of $\Delta\boldsymbol{\theta}_i$ matrix, $\Delta\boldsymbol{\theta}_{i,j}$, **do**
    Calculate the time-averaged parameter vector
      $\Delta\boldsymbol{\theta}_{i,j} = \left(\int_{t_0}^{t_f} \Delta\boldsymbol{\theta}_{i,j} \circ \boldsymbol{\Upsilon}(s)\, ds\right) \cdot / \int_{t_0}^{t_f} \boldsymbol{\Upsilon}(s)\, ds,$
    where $\circ$ and $\cdot/$ denote element-wise multiplication and division.
  end for
end for
return $\Delta\boldsymbol{\theta}$

References

[1] Jonas Buchli. *Optimal and Learning Control for Autonomous Robots*, August 2017. Course Notes, Swiss Federal Institute of Technology in Zurich (ETH Zurich). Accessed on: Mar. 22, 2020. [Online]. Available: https://arxiv.org/abs/1708.09342.

[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.