

Assignment 02: Taking account of underactuation in DDP

Gianluigi Grandesso* - gianluigi.grandesso@unitn.it

November 2022

1 Description

The goals of this assignment are:

- making practice using Differential Dynamic Programming (DDP) for generating reference trajectories and feedback control gains
- comparing two methods for considering underactuation in DDP applied to the case of a double pendulum without motor on the second joint (Pendubot)

2 Submission procedure

You are encouraged to work on the assignments in groups of 2 people. **If you have a good reason to work alone, then you can do it, but this has to be previously validated by one of the instructors.** Groups of more than 2 people are not allowed. The mark of each assignment contributes to 10% of your final mark for the class (i.e. 3 points out of 30).

When you are done with the assignment, please submit a single compressed file (e.g., zip). **The file name should contain the surnames of the group members**, and it must contain:

- A pdf file with the answers to the questions, the **names and ID number** of the group members; you are encouraged to include plots and/or numerical values obtained through simulations to support your answers. **This pdf does not need to be long. One or two pages of text should be enough to answer the questions. You can then add other pages for plots and tables. The fact that the pdf can be short does not mean that the answers are not expected to be thorough.**
- The complete orc folder containing all the python code that you have developed.

If you are working in a group (i.e., 2 people) only one of you has to submit.

Submitting the pdf file without the code is not allowed and would result in zero points. Your code should be consistent with your answers (i.e. it should be possible to produce the results that motivated your answers using the code that you submitted). If your code does not even run, then your mark will be zero, so make sure to submit a correct code.

*Optimization-based Robot Control, Industrial Engineering Department, University of Trento.

3 DDP

DDP is a direct optimal control method for finite-horizon trajectory optimization that can deal with discrete nonlinear systems by using the dynamic programming principle, also known as Bellman principle of optimality, and a local quadratic approximation of the Q function to finally get an optimal open-loop control trajectory \bar{u} and locally optimal linear feedback gains K that add together to form the final control u :

$$u = \bar{u} + K(\bar{x} - x) \quad (1)$$

where x is the current state of the system and \bar{x} is the state reference trajectory.

This algorithm proceeds by alternating backward and forward passes: first the gradients and Hessians of the Q and Value functions are computed to get the optimal variation of the control and then the system is simulated to compute the cost and set the new reference control and state trajectories, respectively. The algorithm stops when the convergence criterion is satisfied, meaning that the cost decrease computed in the forward pass remains under a certain threshold.

The general structure of the optimal control problem solved by DDP is as follows:

$$\begin{aligned} \min_{u_t} \quad & \sum_{t=1}^{N-1} l_t(x_t, u_t) + l_N(x_N) \\ \text{s.t.} \quad & x_{t+1} = f(x_t, u_t) \\ & x_0 = x^{init} \end{aligned} \quad (2)$$

where $l_t(x_t, u_t)$ is the running cost, $l_N(x_N)$ is the final cost and $f(x_t, u_t)$ is the discretized system dynamics.

For simplicity we consider a linearization of the system dynamics around the current reference trajectory. Thus, starting from an initial guess for the control trajectory U , the algorithm alternates between the linearization, the solution of the LQR problem to get the optimal variation of U and the line search to ensure convergence.

Minimizing the local quadratic model of Q we get the optimal control variation w^* :

$$w^* = \arg \min_w Q(z, w) = -Q_{uu}^{-1} (Q_u + Q_{ux}z) = \bar{w} + Kz \quad (3)$$

where z is the state variation from the current reference trajectory, $\bar{w} \triangleq -Q_{uu}^{-1}Q_u$, $K \triangleq -Q_{uu}^{-1}Q_{ux}$, Q_u is the gradient of Q w.r.t. u and Q_{uu} and Q_{ux} are the diagonal term regarding u and the off-diagonal term of the Hessian of Q , respectively. Considering the definition of Q at timestep i $Q(z, w) \triangleq \min_w [l_i(z, w) + V(f(z, w), i+1)]$, the terms Q_u , Q_{uu} and Q_{ux} are expressed as follows:

$$\begin{aligned} Q_u &= l_x + f_x^T V'_x \\ Q_{uu} &= l_{uu} + f_u^T V'_{xx} f_u + V'_x f_{uu} \\ Q_{ux} &= l_{ux} + f_x^T V'_{xx} f_x + V'_x f_{ux} \end{aligned} \quad (4)$$

where $V' \triangleq V(\cdot, i+1)$, $V_x \triangleq Q_x - Q_{ux}Q_{uu}^{-1}Q_u$, $V_{xx} \triangleq Q_{xx} - Q_{ux}Q_{uu}^{-1}Q_{ux}$ and the subscripts indicate w.r.t. which variables the derivation is performed.

If Q_{uu} is not invertible or is too small we can regularize it by adding a scalar $\mu > 0$ to its diagonal terms:

$$\bar{Q}_{uu} = Q_{uu} + \mu I \quad (5)$$

The pseudocode of the algorithm is illustrated here below:

Algorithm 1 DDP

```

1: Given  $\bar{U} \leftarrow U^0$ , compute  $\bar{X}$  by simulating  $x_{t+1} = f(x_t, u_t)$ 
2: Backward Pass
3:   Set  $V_x(N) = \nabla_x l_N$ ,  $V_{xx} = \nabla_{xx} l_N$ 
4:   for  $i = N - 1 \dots 0$ 
5:     Compute  $Q_x, Q_u, Q_{xx}, Q_{uu}, Q_{ux}$ 
6:      $w^* = -Q_{uu}^{-1} (Q_u + Q_{ux}z) = \bar{w} + Kz$ 
7:     Compute  $V_x(i), V_{xx}(i)$ 
8: Forward Pass
9:   Set line search parameter  $\alpha = 1$ 
10:  Simulate system with  $u = \bar{U} + \alpha \bar{w} + K(\bar{X} - x)$ 
11:  if cost has not decreased enough:
12:    Decrease  $\alpha$  and go to 11
13:   $\bar{U} \leftarrow u$ ,  $\bar{X} \leftarrow x$ 
14:  if not converged:
15:    Perform Backward Pass

```

4 Underactuation

The aim of this assignment is to use DDP to control a Pendubot, namely a double pendulum with only the first joint actuated, and the goal is to perform a swing-up maneuver. Since in `example_robot_data` there is only the model of a fully actuated double pendulum, we have to load that model and then properly modify the code in order to take into account the underactuation.

4.1 Selection matrix

The first method to accomplish our goal is to rely on the use of a diagonal selection matrix $S \in \mathbb{R}^{n \times n}$, where n is the state dimension, such that:

$$\begin{cases} S[i, i] = 1 & \text{if } i \text{ corresponds to an actuated joint} \\ S[i, i] = 0 & \text{if } i \text{ corresponds to a joint not actuated} \end{cases} \quad (6)$$

By premultiplying the control torque vector in the system dynamics equations, this selection matrix is used to force the torques associated with the joints not actuated to be zero.

Notice that this modification of the system dynamics will affect also the computation of f_u . Remember that since we neglect the second-order derivatives of the system dynamics, the terms f_{uu} and f_{ux} do not need to be computed.

4.2 Additional penalty

The second method consists in adding a term to the running cost with a large weight (`underact`) to penalize the torque provided by a fictitious motor on the passive joint (`i`).

$$running_cost += \frac{underact}{2} ||u_i||^2 \quad (7)$$

In doing so, please notice that the gradient and hessian of the running cost w.r.t. u must be modified accordingly.

5 Tests

The template code for this part of the assignment is located in:

```
code/orc/02_assignment/ddp_doublependulum_template.py
```

This file already contains all the code for formulating and solving with DDP the optimal control problem of a fully actuated double pendulum that has to perform a swing-up maneuver, as well as the selection matrix method for taking account of underactuation. The only part that needs to be implemented is the additional penalty method illustrated in the previous section. More precisely, the additional penalty method requires to add the penalty shown in Eq. (7) in the running cost function (defined in the `DDPSolverLinearDyn` class) and modify accordingly its gradient and hessian w.r.t. u .

Two flags (`SELECTION_MATRIX` and `ACTUATION_PENALTY`) in the configuration file are used to select one of the two methods for taking account of underactuation. Please do not change the parameters of DDP and the ones expressed in the configuration file.

Try to answer the following questions:

1. Implement the additional penalty method setting `underact` to `1e6` (`SELECTION_MATRIX=0`, `ACTUATION_PENALTY=1`, `PUSH=0`). Verify that DDP finds the same solution found with the selection matrix method (`SELECTION_MATRIX=1`, `ACTUATION_PENALTY=0`, `PUSH=0`).
2. Choose the selection matrix method and set `PUSH=1`. This will simulate four external pushes (instantaneous increase of the second joint velocity) occurring after $N/8$, $N/4$, $N/2$ and $3N/4$ simulation steps. Describe what happens in terms of tracking performance of the reference trajectories.
3. Now set `ddp_params['mu_factor']` to 0, which means keeping the regularization term μ constant and equal to the initial value 10. What differences do you notice compared to the previous test? Can you explain them?