Model Predictive Control - EL2700

Assignment 2 : Finite-time Optimal Control with Dynamic Programming

2020

Automatic Control
School of Electrical Engineering and Computer Science
Kungliga Tekniska Högskolan

Task : Optimal control through dynamic programming

In the first part of this assignment, we will design a finite-time optimal controller for stabilizing the inverted pendulum. In this task, we will disregard the dynamics of the cart and consider the simple dynamics of the inverted pendulum. To this end, you will first analyze a dynamic programming technique on the linear dynamics of the inverted pendulum. We will then extend the formulation to design a nonlinear controller by feedback linearization. As the previous task, we will resort to CasADi and Python to complete this assignment.

In the second part of this assignment, we will include cart dynamics and design a finite-time optimal controller for moving the cart from one position to the other while maintaining the pendulum upright. Here you will provide a possible solution and hopefully understand how complex the problem becomes to solve due to the curse of dimensionality.

Before proceeding: Please run `install_deps.py` to install missing packages needed for this Assignment.

Part 1: Inverted Pendulum Model

Inverted pendulum model We have modeled the continuous-time nonlinear dynamics of the inverted pendulum using the following first order ODE system

$$\begin{aligned} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} &= \begin{bmatrix} x_2 \\ -a_0 \sin x_1 - a_1 x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ b_0 u \cos x_1 \end{bmatrix} \\ y &= x_1 \end{aligned} \tag{1}$$

with $a_0 = -\frac{mgl}{I+ml^2}$, $a_1 = \frac{b}{I+ml^2}$ and $b_0 = \frac{ml}{I+ml^2}$, where $x_1 = \theta$ and $x_2 = \dot{\theta}$.

Using dynamic programming, we can find the optimal control u_t by solving the one-step optimization problem

$$J_t(x_t) = \min_{u_t} x_t^T Q x_t + u_t^T R u_t + J_{t+1}(Ax_t + Bu_t). \tag{2}$$

The difficulty lies in calculating the cost-to-go function $J_{t+1}(x_{t+1})$. We can approximate the cost-to-go by gridding the state-space and calculating the value function at these grid points. We will use the CasADi optimization class `nlpsol` and `ca.interpolant` to approximate the cost-to-go.

Under the file `dp.py` you will find the `Gridder` class. This class solves a dynamic programming problem recurring to the gridding method. In this method, a part of the state-space is sampled using a grid, and for each point in the grid, an optimal control input is generated.

The method `create_grid` creates the grid according to the user specifications.

Q1: We sample a grid with 10 and 5 points for x_1 and x_2, respectively. What effect does a higher number of points have? What about the computational complexity?

Q2: The grid is created with a small epsilon away of $\frac{\pi}{2}$. Why?

Now turn your attention to the method `solve_grid`. Here, the gridding method is implemented.

Q3: Motivate and explain how the problem is solved in your own words.

Take a careful look into the function `solve_dp`. In this function a solver is used to obtain the optimal control for the time-step at hand.

Q4: Why do we include constraints on the state transition? Can this have undesired side-effects?

Extra (not mandatory): The used solver supports different options. These can be viewed in `https://web.casadi.org/python-api/#nlp`, under the options for the IPOPT solver. Play with the options of the solver to see what effect they have, especially, the number of maximum iterations and print levels. These will be useful down the road for debugging your implementations.

Finally, run this linear DP control policy by running `task2.py`.

Q5: Tune the matrices Q and R with different weights in the `task2.py` script. Comment on the implication of different gains in the performance of the system.

Nonlinear control design

One advantage of gridding is that we can include the system nonlinearities in the controller design. An alternative approach to linearizing the inverted pendulum model is a technique called feedback linearization. Lets redefine the input as

$$u = \frac{1}{\cos x_1}\Big(\frac{a_0}{b_0}(\sin x_1 - x_1) + v\Big). \tag{3}$$

Then, the nonlinear dynamics of the inverted pendulum (4) become linear for the fictitious control

$$\begin{bmatrix}\dot{x_1}\\ \dot{x_2}\end{bmatrix} = \begin{bmatrix}0 & 1\\ -a_0 & -a_1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix} + \begin{bmatrix}0\\ b_0\end{bmatrix} v. \tag{4}$$

To design v, we need to substitute the expression for u in (3) into the stage cost

$$J_{stage}(x_t, v_t) = x_t^T Q x_t + R\left[\frac{1}{\cos x_1}\Big(\frac{a_0}{b_0}(\sin x_1 - x_1) + v\Big)\right]^2. \tag{5}$$

Finally, we can obtain $v^\star$ by solving the minimization problem

$$J_t(x_t) = \min_{v_t} J_{stage}(x_t, v_t) + J_{t+1}(Ax_t + Bv_t). \tag{6}$$

The optimal control $u^\star$ is then given by (3). Your task is to implement the nonlinear controller in (3) by modifying the stage cost J_{stage} - under the `set_cost_functions` method - according to (5), and incorporating the fictitious control input v - in the `solve_grid` method. You can use the same penalty matrices as in the linear formulation to be able to compare the performance of this controller to its linear counterpart. Now, simulate your controller with the same initial point and save your results.

Performance comparison You will now compare the performance of the linear formulation and the nonlinear formulation in terms of the following performance measures:

1. computation time for offline calculation of the optimal control for same grid size;
2. accumulated cost and spent control energy;
3. convergence rate to the upright position.

Part 2: Cart-Pendulum Model

Dynamic programming is a powerful technique for solving optimal control problems. However, it suffers from the notorious "curse of dimensionality", which prevents its direct application when the dimension of the state space is high. To observe this problem, we will now consider a four dimensional state-space model where we include both cart and pendulum dynamics.

Linear controller design We will now design a finite time optimal controller for moving the cart from one position to another while keeping the pendulum upright. We will reuse the linearized cart-pendulum model around the upright position which you derived in the first design project. The discrete-time linear dynamics of the cart pendulum model can be represented as:

$$\mathbf{x}_{t+1} = A\mathbf{x}_t + Bu_t \tag{7}$$

where $\mathbf{x}_t = \begin{bmatrix}x & \dot{x} & \theta & \dot{\theta}\end{bmatrix} \in \mathbb{R}^4$ and $u_t \in \mathbb{R}$ are the system states and control input respectively. We will now compute an optimal solution to the following tracking problem

$$\min_{u_t} \sum_{t=1}^{N} (\mathbf{x}_t - \mathbf{x}_r)^T Q(\mathbf{x}_t - \mathbf{x}_r) + u_t^T R u_t \tag{8}$$

$$\text{subject to: } \mathbf{x}_{t+1} = A\mathbf{x}_t + Bu_t, \quad t = 1, \ldots, N-1 \tag{9}$$

where $\mathbf{x}_r$ is the reference state we want to track. In this problem, $\mathbf{x}_r = \begin{bmatrix} 10 & 0 & 0 & 0 \end{bmatrix}$. To solve this problem, take inspiration from the class `Gridder` to construct a five dimensional grid, corresponding to four system states and the time dimension, under the class `BiggerGridder`.

Note: in this Part 2 we expect you to provide a tentative solution that conceptually works - even if during the simulation it fails to achieve a solution. Provide a few comments on why this is a more difficult task, and which problems you faced.

The parts you should fill in `BiggerGridder` are:

- Method `create_grid`;
- Method `set_cost_functions`;
- Method `solve_grid`;
- Method `solve_dp`, specifically the solver options.

Tips for this problem:

- Explore the solver options regarding the the maximum number of iterations and print levels to debug your code;
- Also use the variable `self.DEBUG`, by setting it to `True`, to have more information on which problems you might be having.

As always, refer to the Slack workspace `el2700workspace.slack.com` for questions.

Good Luck!