Stabilizing a Double Pendulum using MC-PILCO

Alessandro Borgherini

April 27, 2025

1 Introduction

In this project, the Monte Carlo version of PILCO (MC-PILCO), a data-efficient model-based reinforcement learning algorithm, to learn a stabilization policy is employed [1].
The double pendulum system consists of two links and actuated joints. In this project the Pendubot version of the pendulum is used, menaning that only the first joint is actuated. The goal is to bring the second link to a vertically upright position and maintain stability.

1.1 Software Overview

The file `test_mcpilco_double_pendulum.py` is used to assign values to all the hyperparameters and to run the full algorithm.
To plot all the graphs run `log_plot_double_pendulum.py`.
To obtain the final policy parameters run `obtain_final_policy.py`.
To test the final policy on the simulated pendulum run `test_doublependulum_policy.py`.

2 Methodology

2.1 MC-PILCO brief Overview

MC-PILCO (Monte Carlo Probabilistic Inference for Learning COntrol) is a model-based policy search algorithm that builds upon the original PILCO framework. It uses Gaussian Processes (GPs) to learn a probabilistic model of the system dynamics from data. Unlike PILCO, which relies on analytical moment matching, MC-PILCO leverages Monte Carlo rollouts to propagate uncertainty, making it more flexible and applicable to complex or highly nonlinear systems.
The algorithm follows an iterative loop: it begins by collecting trajectory data using an initial exploratory policy. This data is then used to train a GP model of the dynamics. Using the learned model, multiple particles are simulated through Monte Carlo sampling. These simulated rollouts are used to estimate the expected long-term cost, and the policy is updated accordingly. The process then repeats—collecting new data, refining the model, and improving the policy over time.
To cope with the large computational burden due to the high number of collected samples, the GP approximation Subset of Regressors is implemented with a threshold equal to the one shown in table 1, see [2] for a detailed description of the approximation.

2.2 Policy

The policy used is the squashed RBF network, that is a Radial Basis Function (RBF) with outputs limited by an hyperbolic tangent function. It's defined as

$$\pi_\theta(\mathbf{x}) = u_{\max} \tanh\left(\frac{1}{u_{\max}} \sum_{i=1}^{n_b} w_i e^{-\|\mathbf{a}_i - \phi(\mathbf{q})\|^2_{\Sigma_\pi}}\right) \tag{1}$$

$$\phi(\mathbf{q}(\mathbf{t})) = [\dot{\mathbf{q}}(\mathbf{t}), \cos(\mathbf{q}(\mathbf{t})), \sin(\mathbf{q}(\mathbf{t}))]$$

$\mathbf{q}(\mathbf{t})$ are the double pendulum current joints configuration at time t.
The policy parameters to be learned are $\theta = \{\omega, A, \sigma_\pi\}$ where $\omega = [\omega_1, \omega_2, \ \ldots, \omega_{n_b}]$ and $A = [a_1, a_2, \ \ldots, a_{n_b}]$.
u_{max} and n_b are hyperparameters, the value used in this project are shown in table 1.

2.3 Cost function

The cost function used is a commonly used one in the form of

$$c(\mathbf{q}) = 1 - e^{-\|\mathbf{q} - \mathbf{q}_{des}\|^2_{\Sigma_c}} \tag{2}$$

$$\Sigma_c = diag\{\frac{1}{l_1}, \frac{1}{l_2}\}$$

Where l_1 and l_2 are two other hyperparameters.
The desired final configuration is set at $\mathbf{q}_{des} = [\pi, 0]$.

2.4 Hyperparameters

In table 1 are shown the learning algorithm hyperparameters used.

3 Results

3.1 Training Performance

In Figure 1, the plots of the rollout costs for each episode are shown. By "rollout cost", we refer to the average cost across all particles simulated on the learned model.
It can be observed that the policy learning component of the algorithm consistently succeeds in learning a good policy for each model obtained at every episode. However, by comparing the final rollout cost at the end of one episode with the initial cost at the start of the next, it becomes evident that the model is continually improving, requiring the algorithm to learn a new policy each time.
Only after several episodes does the model appear to converge and be accurately learned.

Hyperparameter	Symbol	Value
Number of trials	n	19
Number of particles	M	600
Initial learning rate	α	0.01
Initial dropout probability	p_{drop}	0.25
Sampling time	T_s	0.02
Optimization steps	opt_steps	2000
GR training epochs	N_{epoch}	4000
Maximum input	u_{max}	6
Number of basis	n_b	300
Lengthscales	$[l_1,\ l_2]$	[3, 3]
Subset of Regressors threshold	SOR_threshold	0.5

Table 1: Hyperparameters used in the model

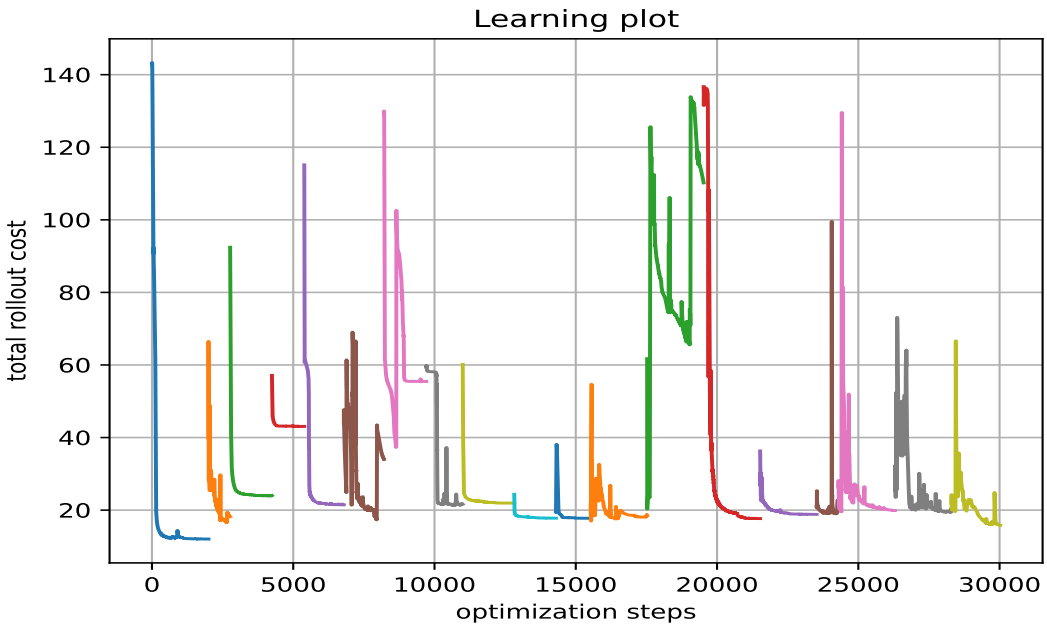

Figure 1: Total rollout cost for each training episode

3.2 Stabilization Behavior

In Figure 2, the performance of the final learned policy on the learned model is shown. As observed, the policy operates correctly, driving the cost to zero. This indicates that the learned model of the double pendulum is successfully stabilized by the policy.
In Figure 3, the performance of the final policy on the true double pendulum system is presented. Although the policy stabilizes the learned model, it fails to fully stabilize the true system.

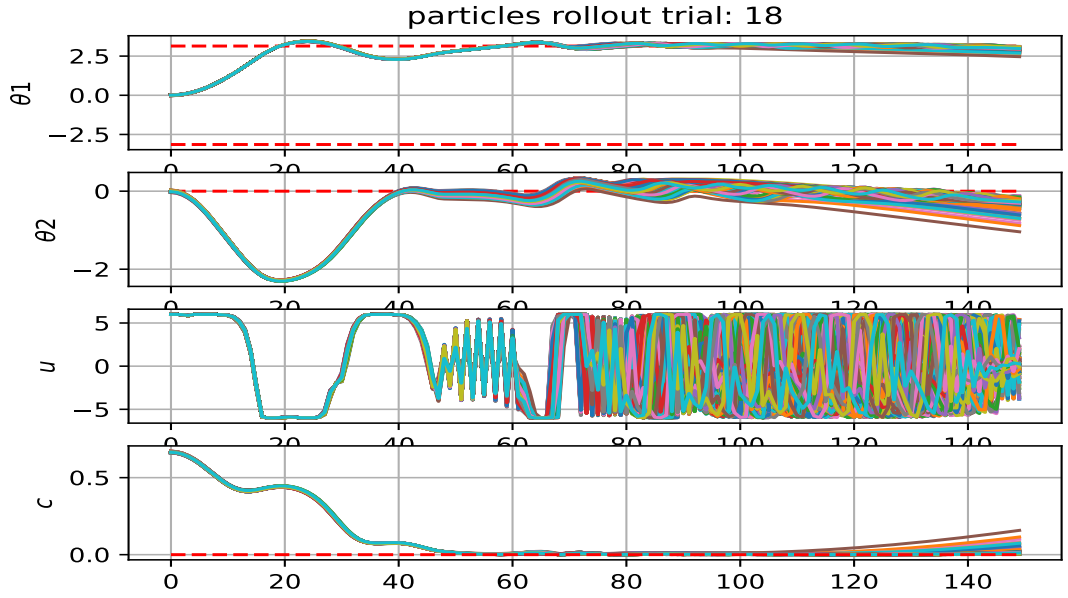

Figure 2: State trajectories for the particles rollouts on the learned model under the final learned policy.

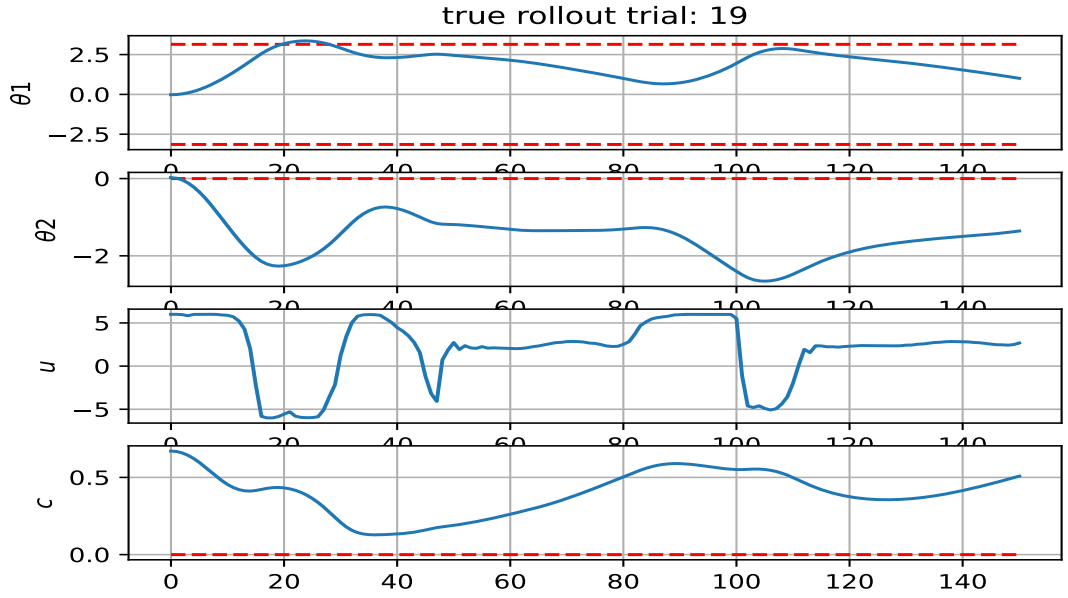

Figure 3: State trajectories under the final learned policy on the true system.

4 Discussion and Conclusion

By examining Figures 2 and 3, it is clear that the algorithm, after 19 episodes, did not fully succeed in learning the true model. Instead, the policy that stabilizes the learned model is unable to stabilize the true physical system.
This is likely caused by the GoR approximation of the model and could potentially be mitigated by tuning the threshold hyperparameter.
Further improvements to the algorithm could also be achieved by adjusting additional hyperparameters.

References

[1] F. Amadio, A. Dalla Libera, R. Antonello, D. Nikovski, R. Carli, and D. Romeres, *"Model-based policy search using monte carlo gradient estimation with real systems application"*, IEEE Transactions on Robotics, vol. 38, no. 6, pp. 3879–3898, 2022.

[2] J. Quinonero-Candela and C. E. Rasmussen, *"A unifying view of sparse approximate gaussian process regression"*, Journal of Machine Learning Research, vol. 6, no. 65, pp. 1939–1959, 2005.