MITSUBISHI ELECTRIC RESEARCH LABORATORIES
https://www.merl.com

Learning global control of underactuated double pendulum with Model-Based Reinforcement Learning

Turcato, Niccolò; Cali, Marco; Dalla Libera, Alberto; Giacomuzzo, Giulio; Carli, Ruggero; Romeres, Diego

TR2025-046 April 11, 2025

Abstract

This report describes our proposed solution for the third edition of the "AI Olympics with RealAIGym" com- petition, held at ICRA 2025. We employed Monte-Carlo Proba- bilistic Inference for Learning Control (MC-PILCO), an MBRL algorithm recognized for its exceptional data efficiency across various low-dimensional robotic tasks, including cart-pole, ball & plate, and Furuta pendulum systems. MC-PILCO optimizes a system dynamics model using interaction data, enabling policy refinement through simulation rather than direct system data optimization. This approach has proven highly effective in physical systems, offering greater data efficiency than Model- Free (MF) alternatives. Notably, MC-PILCO has previously won the first two editions of this competition, demonstrating its robustness in both simulated and real-world environments. Besides briefly reviewing the algorithm, we discuss the most critical aspects of the MC-PILCO implementation in the tasks at hand: learning a global policy for the pendubot and acrobot systems.

IEEE International Conference on Robotics and Automation (ICRA) - 3rd AI Olympics with RealAIGym Competition 2025

Mitsubishi Electric Research Laboratories, Inc.
201 Broadway, Cambridge, Massachusetts 02139

Learning global control of underactuated double pendulum with Model-Based Reinforcement Learning

Niccolò Turcato[1], Marco Calì[1], Alberto Dalla Libera[1], Giulio Giacomuzzo[1], Ruggero Carli[1] and Diego Romeres[2]

***Abstract*— This report describes our proposed solution for the third edition of the "AI Olympics with RealAIGym" competition, held at ICRA 2025. We employed Monte-Carlo Probabilistic Inference for Learning Control (MC-PILCO), an MBRL algorithm recognized for its exceptional data efficiency across various low-dimensional robotic tasks, including cart-pole, ball & plate, and Furuta pendulum systems. MC-PILCO optimizes a system dynamics model using interaction data, enabling policy refinement through simulation rather than direct system data optimization. This approach has proven highly effective in physical systems, offering greater data efficiency than Model-Free (MF) alternatives. Notably, MC-PILCO has previously won the first two editions of this competition, demonstrating its robustness in both simulated and real-world environments. Besides briefly reviewing the algorithm, we discuss the most critical aspects of the MC-PILCO implementation in the tasks at hand: learning a global policy for the pendubot and acrobot systems.**

I. Introduction

This report outlines our team's implementation of a Reinforcement Learning (RL) approach to address the third "AI Olympics with RealAIGym" competition at ICRA 2025[1], based on the RealAIGym project [1]. Specifically, we employed Monte-Carlo Probabilistic Inference for Learning Control (MC-PILCO) [2], a Model-Based (MB) RL algorithm known for its exceptional data efficiency in various low-dimensional benchmarks, including cart-pole, ball & plate, and Furuta pendulum systems, both in simulation and real-world environments. Notably, MC-PILCO also secured victory in the first two editions of this competition [3], [4], [5]. MC-PILCO leverages interaction data to optimize a system dynamics model. Instead of directly optimizing the policy using system data, it refines the policy by simulating the system, enhancing data efficiency. Considering physical systems, this approach can be highly performing and more data-efficient than Model-Free (MF) solutions. Examples of MC-PILCO applications and derivations have been reported in [6], [7], [8], [9].

This paper is organized as follows: Section II introduces the goal and the settings of the competition. Section III presents the MC-PILCO algorithm for global policy training. Section IV reports the experiments that have been performed, finally Section V concludes the paper.

[1]Department of Information Engineering, University of Padova, Italy.
[2]Mitsubishi Electric Research Laboratories, Cambridge, MA, USA
Correspondence to niccolo.turcato@phd.unipd.it
[1]https://ai-olympics.dfki-bremen.de/

Compared to the solutions proposed in the first two editions, the solution proposed for this competition integrates a new type of incremental training (Section III-B), which aims at developing a global controller for the system.

II. The competition

The challenge considers a 2 degrees of freedom (dof) underactuated pendulum [10] with two possible configurations. In the first configuration, also called Pendubot, the first joint, namely, the one attached to the base link is active, and the second is passive. Instead, in the second configuration, also named Acrobot, the first joint is passive and the second is actuated. For each configuration, the competition's goal is to derive a controller that performs the swing-up and stabilization in the unstable equilibrium point of the systems. Both robots are underactuated, which makes the task particularly challenging from the control point of view. The systems are simulated at $500\,\mathrm{Hz}$ with a Runge-Kutta 4 integrator for a horizon of $T = 60\,\mathrm{s}$. The competition is composed of 2 stages. In the first stage, namely the simulation stage, controllers are assessed on a simulated system. In the second stage, namely the real hardware stage, the participating teams test their controllers on the real system, with the possibility of retraining their learning-based controllers. The goal of the competition is to develop a global policy, thus the controllers are evaluated by re-initializing the system in random positions sampled from the state space, at random times. The winners of the competition are chosen based on the performance and the reliability of the submitted controllers.

III. Learning a global policy with MC-PILCO

In this section, we first review MC-PILCO, and secondly, we discuss its application to learn a global policy for the underactuated double pendulum.

A. MC-PILCO

MC-PILCO is a Model-Based policy gradient algorithm. Gaussian Processes (GPs) are used to estimate system dynamics and long-term state distributions are approximated with a particle-based method.

Consider a system with evolution described by the discrete-time unknown transition function $f : \mathbb{R}^{d_x} \times \mathbb{R}^{d_u} \to \mathbb{R}^{d_x}$:

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{w}_t, \tag{1}$$

where $\boldsymbol{x}_t \in \mathbb{R}^{d_x}$ and $\boldsymbol{u}_t \in \mathbb{R}^{d_u}$ are respectively the state and input of the system at step t, while $\boldsymbol{w}_t$ is an independent white noise describing uncertainty influencing the system evolution. As usual in RL, a cost function $c(\boldsymbol{x}_t)$ encodes the task to be accomplished. A policy $\pi_{\boldsymbol{\theta}} : \boldsymbol{x} \to \boldsymbol{u}$ that depends on the parameters $\boldsymbol{\theta}$ selects the inputs applied to the system. The objective is to find policy parameters $\boldsymbol{\theta}^*$ that minimize the cumulative expected cost, defined as follows,

$$J(\boldsymbol{\theta}) = \sum_{t=0}^{T} \mathbb{E}[c(\boldsymbol{x}_t)], \tag{2}$$

where the initial state x_0 is sampled according to a given probability $p(\boldsymbol{x}_0)$.

MC-PILCO consists of a series of attempts, known as trials, to solve the desired task. Each trial consists of three main phases: (i) model learning, (ii) policy update, and (iii) policy execution. In the first trial, the GP model is derived from data collected with an exploration policy, for instance, a random exploration policy.

In the model learning step, previous experience is used to build or update a model of the system dynamics. The policy update step formulates an optimization problem whose objective is to minimize the cost in eq. (2) w.r.t. the parameters of the policy $\boldsymbol{\theta}$. Finally, in the last step, the current optimized policy is applied to the system and the collected samples are stored to update the model in the next trials.

In the rest of this section, we give a brief overview of the main components of the algorithm and highlight their most relevant features.

1) Model Learning: MC-PILCO relies on GP Regression (GPR) to learn the system dynamics [11]. For the use of GPs in system identification and control we refer the interested reader to [12]. In our previous work, [2], we presented a framework specifically designed for mechanical systems, named speed-integration model. Given a mechanical system with d degrees of freedom, the state is defined as $\boldsymbol{x}_t = [\boldsymbol{q}_t^T, \dot{\boldsymbol{q}}_t^T]^T$ where $\boldsymbol{q}_t \in \mathbb{R}^d$ and $\dot{\boldsymbol{q}}_t \in \mathbb{R}^d$ are, respectively, the generalized positions and velocities of the system at time t. Let T_s be the sampling time and assume that accelerations between successive time steps are constant. The following equations describe the one-step-ahead evolution of the i-th degree of freedom,

$$\dot{q}_{t+1}^{(i)} = \dot{q}_t^{(i)} + \Delta_t^{(i)} \tag{3a}$$

$$q_{t+1}^{(i)} = q_t^{(i)} + T_s \dot{q}_t^{(i)} + \frac{T_s}{2} \Delta_t^{(i)} \tag{3b}$$

where $\Delta_t^{(i)}$ is the change in velocity. MC-PILCO estimates the unknown function $\Delta_t^{(i)}$ from collected data by GPR. Each $\Delta_t^{(i)}$ is modeled as an independent GP, denoted f^i, with input vector $\tilde{\boldsymbol{x}}_t = [\boldsymbol{x}_t^T, \boldsymbol{u}_t^T]^T$, hereafter referred as GP input.

The posterior distributions of each $\Delta_t^{(i)}$ given $\mathcal{D}^i$ are Gaussian distributed, with mean and variance expressed as follows:

$$\begin{aligned} &\mathbb{E}[\hat{\Delta}_t^{(i)}] = m_\Delta^{(i)}(\tilde{\boldsymbol{x}}_t) + K_{\tilde{\boldsymbol{x}}_t \tilde{\boldsymbol{X}}} \Gamma_i^{-1} (\boldsymbol{y}^{(i)} - m_\Delta^{(i)}(\tilde{\boldsymbol{X}})), \\ &var[\hat{\Delta}_t^{(i)}] = k_i(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{x}}_t) - K_{\tilde{\boldsymbol{x}}_t \tilde{\boldsymbol{X}}} \Gamma_i^{-1} K_{\tilde{\boldsymbol{X}} \tilde{\boldsymbol{x}}_t}, \\ &\Gamma_i = K_{\tilde{\boldsymbol{X}} \tilde{\boldsymbol{X}}} + \sigma_i^2 I, \end{aligned} \tag{4}$$

refer to [11] for the derivation of Equation (4). Then, also the posterior distribution of the one-step ahead transition model in (3) is Gaussian, namely,

$$p(\boldsymbol{x}_{t+1} | \boldsymbol{x}_t, \boldsymbol{u}_t, \mathcal{D}) \sim \mathcal{N}(\boldsymbol{\mu}_{t+1}, \Sigma_{t+1}) \tag{5}$$

with mean $\boldsymbol{\mu}_{t+1}$ and covariance Σ_{t+1} derived combining (3) and (4).

2) Policy Update: In the policy update phase, the policy is trained to minimize the expected cumulative cost in eq. (2) with the expectation computed w.r.t. the one-step ahead probabilistic model in eq. (5). This requires the computation of long-term distributions starting from the initial distribution $p(\boldsymbol{x}_0)$ and eq. (5), which is not possible in closed form. MC-PILCO resorts to Monte Carlo sampling [13] to approximate the expectation in eq. (2). The Monte Carlo procedure starts by sampling from $p(\boldsymbol{x}_0)$ a batch of N particles and simulates their evolution based on the one-step-ahead evolution in eq. (5) and the current policy. Then, the expectations in eq. (2) are approximated by the mean of the simulated particles costs, namely,

$$\hat{J}(\boldsymbol{\theta}) = \sum_{t=0}^{T} \left(\frac{1}{N} \sum_{n=1}^{N} c\left(\boldsymbol{x}_t^{(n)} \right) \right) \tag{6}$$

where $\boldsymbol{x}_t^{(n)}$ is the state of the n-th particle at time t.

The optimization problem is interpreted as a stochastic gradient descend problem (SGD) [14], applying the reparameterization trick to differentiate stochastic operations [15].

The authors of [2] proposed the use of dropout [16] of the policy parameters $\boldsymbol{\theta}$ to improve exploration and increase the ability to escape from local minima during policy optimization of MC-PILCO.

B. Global Policy training

The task in object presents several practical issues when applying the algorithm. The first one is that the control frequency requested by the challenge is quite high for a MBRL approach. Indeed, high control frequencies require a high number of model evaluations which increases the computational cost of the algorithm. Generally, this class of systems can be controlled at relatively low frequencies, for instance, [2] and [17] derived a MBRL controller for a Furuta Pendulum at 33 Hz. Indeed, in the real hardware stage of the first two editions of the competition, the MC-PILCO controller was trained to work at 33 Hz. However, the physical properties of the simulated systems (no friction) make the system particularly sensitive to the system input. For these reasons, we selected a control frequency of 50 Hz.

The second challenge lies in the task requirements. Indeed, the task requires the policy to drive the system to the unstable equilibrium starting from initial state $x_0 = [p_0^T, \dot{p}_0^T]^T$, where

$p_0 \in [-\pi, \pi] \times [-\pi, \pi]$ and $||\dot{p}_0||$ is very small. Thus the initial state distribution of the system can be defined as

$$p(x_0) \sim U\left(-x_M, x_M\right), x_M = \begin{bmatrix} \pi \\ \pi \\ \varepsilon \\ \varepsilon \end{bmatrix} \tag{7}$$

where $\varepsilon > 0$ is a very small constant.

Since the nominal model of the system is available to develop the controller, we use the forward dynamics function of the plant as the prior mean function of the change in velocity for each joint. The available model is

$$B\boldsymbol{u}_t = M(\boldsymbol{q}_t)\ddot{\boldsymbol{q}}_t + n(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t), \tag{8}$$

where $M(\boldsymbol{q}_t)$ is the mass matrix, $n(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t)$ contains the Coriolis, gravitational and damping terms, and B is the actuation matrix, which is $B = \text{diag}([1, 0])$ for the Pendubot and $B = \text{diag}([0, 1])$ for the Acrobot. We define then

$$m_\Delta(\tilde{\boldsymbol{x}}_t) = \begin{bmatrix} m_\Delta^{(1)} \\ m_\Delta^{(2)} \end{bmatrix} := T_s \cdot M^{-1}(\boldsymbol{q}_t)(B\boldsymbol{u}_t - n(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t)) \tag{9}$$

as the mean function in eq. (4). The control input $\boldsymbol{u}_t \in \mathbb{R}$ is a scalar representing the torque given in input to the controlled joint. It is important to point out that eq. (9) is nearly perfect to approximate the system when T_s is sufficiently small, but it becomes unreliable as T_s grows. In particular, with $T_s = 0.02\,\text{s}$ the predictions of eq. (9) are not good enough to describe the behavior at the unstable equilibrium. The inaccuracies of the prior mean are compensated by the GP models. To cope with the large computational burden due to the high number of collected samples, we implemented the GP approximation Subset of Data, see [18] for a detailed description.

An important aspect of policy optimization is the particle initialization, in this case using the initial distribution eq. (7) in eq. (6) at the first trial results in a very unreliable optimization problem, which typically does not converge to acceptable solutions. For this reason, we employ an incremental initialization strategy to learn global control for the system. Namely we use a surrogate initial distribution for both policy execution and particles initialization:

$$p'_k(x_0) \sim U\left(-x_M \cdot \gamma_k, x_M \cdot \gamma_k\right), \tag{10}$$

$$\gamma_k = \text{clip}(\frac{k - k_m}{K}, 0, 1), \tag{11}$$

where $k \in \mathbb{N}$ is the trial index, and $k_m, K \in \mathbb{N}$ regulate the increment in the uniform distribution's bounds. This strategy falls within Curriculum Learning [19], as the policy is trained on a task of increasing difficulty.

The cost function must evaluate the policy performance w.r.t. the task requirements, in this case, we want the system to reach the position $\boldsymbol{q}_G = [\pi, 0]^T$ and stay there indefinitely. A cost generally used in this kind of system is the saturated distance from the target state:

$$c_{st}(\boldsymbol{x}_t) = 1 - e^{-\|\|\boldsymbol{q}_t| - \boldsymbol{q}_G\|^2_{\Sigma_c}} \qquad \Sigma_c = \text{diag}\left(\frac{1}{\ell_c}, \frac{1}{\ell_c}\right), \tag{12}$$

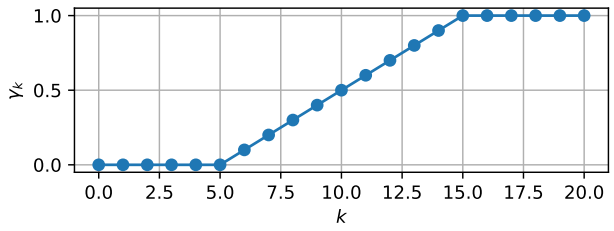

Fig. 1: γ_k scheduling following eq. (11), with $k_m = 5, K = 10$.

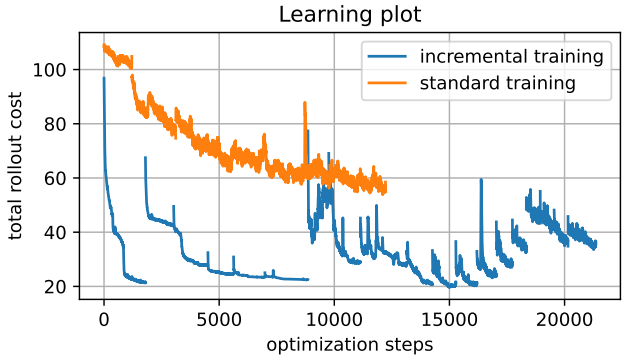

Fig. 2: Total rollout costs in the policy optimization steps of the two MC-PILCO trainings, the first using the incremental initial distribution, the second using the nominal initial distribution in all trials.

with $\ell_c = 3$. Notice that this cost does not depend on the velocity of the system, just on the distance from the goal state, but it does encourage the policy to reach the goal state with zero velocity.

The policy function that is used to learn a control strategy is the general purpose policy from [2]:

$$\pi_{\boldsymbol{\theta}}(\boldsymbol{x}_t) = u_M \tanh\left(\sum_{i=1}^{N_b} \frac{w_i}{u_M} e^{-\|\boldsymbol{a}_i - \phi(\boldsymbol{x}_t)\|^2_{\Sigma_\pi}}\right)$$
$$\phi(\boldsymbol{x}_t) = [\dot{\boldsymbol{q}}_t^T, \cos(\boldsymbol{q}_t^T), \sin(\boldsymbol{q}_t^T)]^T \tag{13}$$

with hyperparameters $\boldsymbol{\theta} = \{\mathbf{w}, A, \Sigma_\pi\}$, where $\mathbf{w} = [w_1, \ldots, w_{N_b}]^T$ and $A = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{N_b}\}$ are, respectively, weights and centers of the N_b Gaussians basis functions, whose shapes are determined by Σ_π. For both robots, the dimensions of the elements of the policy are: $\Sigma_\pi \in \mathbb{R}^{6\times6}$, $a_i \in \mathbb{R}^6$, $w_i \in \mathbb{R}$ for $i = 1, \ldots, N_b$, since the policy outputs a single scalar. In the experiments, the parameters are initialized as follows. The basis weights are sampled uniformly in $[-u_M, u_M]$, the centers are sampled uniformly in the image of ϕ with $\dot{\boldsymbol{q}}_t \in [-2\pi, 2\pi]$ rad/s. The matrix Σ_π is initialized to the identity.

IV. Experiments

In this section, we briefly discuss how the algorithm was applied to both systems and show the main results. We also report the optimization parameters used for both systems, all the parameters not specified are set to the values reported in [2]. All the code was implemented in Python with the PyTorch [20] library.

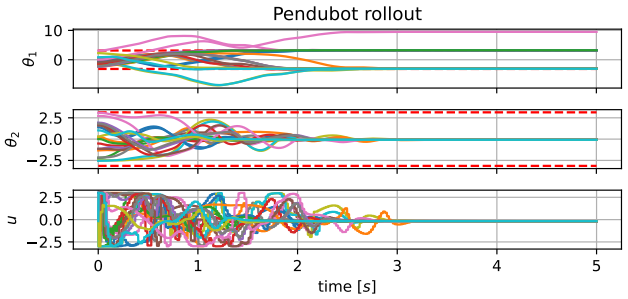

Fig. 3: 20 simulated trials of the Pendubot system (500 Hz), under MC-PILCO's control policy (50 Hz). The initial position for each joint is uniformly sampled from the interval $[-\pi, \pi]$.

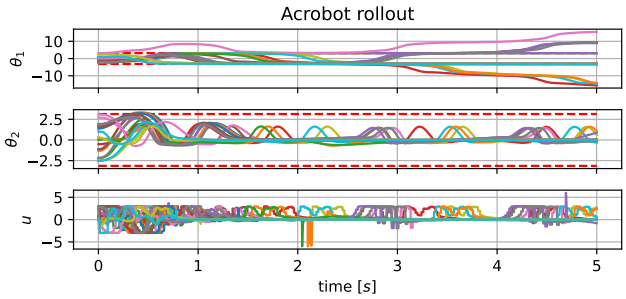

Fig. 4: 20 simulated trials of the Pendubot system (500 Hz), under MC-PILCO's control policy (50 Hz). The initial position for each joint is uniformly sampled from the interval $[-\pi, \pi]$.

Controller	Perf. score Pendubot	Perf. score Acrobot
MC-PILCO (incremental)	0.468	0.292
MC-PILCO (standard)	0.1	0.21
TVLQR	0.094	0.073

TABLE I: Pendubot and Acrobot scores comparison.

For both robots, we use the model described in Section III-A.1, with mean function from eq. (9) and squared-exponential kernel [11]. The policy optimization horizon was set much lower than the horizon required for the competition, this allows to reduce the computational burden of the algorithm, moreover, it pushes the optimization to find policies that can execute a fast swing-up. We exploit dropout in the policy optimization as a regularization strategy, to yield better policies.

Note: The performance score for the simulation stage is computed by simulating the controllers for $60\,\mathrm{s}$, randomly resetting the joints to some position in $[-\pi, \pi]$. The score is proportional to the duration for which the system remains stabilized in the unstable equilibrium[2]. However, this is done with a PID controller activated for $0.2\,\mathrm{s}$, which, when the joints are outside $[-\pi, \pi]$ causes the system to accelerate to high velocity, becoming uncontrollable, due to the lack of friction. This can be partially solved on the pendubot by switching to a damping control when the joint velocity is too high, while it remains unsolved for the acrobot.

A. Pendubot

The policy for the Pendubot swing-up was optimized for a horizon of $T = 3.0\,\mathrm{s}$, with $u_M = 3.0\,\mathrm{N}\cdot\mathrm{m}$. The control switches to a damping controller $\boldsymbol{u} = -D\dot{\boldsymbol{q}}_1$ when $\max(\dot{\boldsymbol{q}}_1, \dot{\boldsymbol{q}}_2) \geq 20\,\mathrm{rad/s}$ to limit uncontrollable maneuvers caused by the resetting PID. The Linear Quadratic Regulator (LQR) controller for stabilization is not used for this system, to avoid additional complexity in the control strategy. The policy is optimized over a total of 20 episodes, using γ_k scheduling reported in fig. 1 ($k_m = 5, K = 10$).

In fig. 2 we compare the total rollout costs of the policy update steps obtained with the proposed incremental training, with the standard approach using the nominal initial distribution eq. (7) in all trials. While the number of episodes of the two trainings is the same, the number of steps in the optimizations is quite different. Specifically, the incremental training allows the policy to first learn the swing-up task, starting from the stable equilibrium, and then gradually adapt the parameters to a wider initial distribution. In contrast, using the nominal uniform distribution since the first trial results in a much more complex optimization, because the initial parameters of the policy are random and the model's prediction is more uncertain (having less data). This results in noisy policy gradient steps, which trigger the exit condition from the optimization sooner than with the proposed strategy. As a result, the final total rollout cost of the policy trained with the incremental initial distribution is lower than with the standard training.

The Controller's strategy is depicted in fig. 3. This controller has a performance score close to 0.5, while the MC-PILCO controller obtained with the standard training achieves a score of 0.1. The baseline controller Time Varing Linear Quadratic Regulator (TVLQR) [3], [10] has a score ≤ 0.1. Table I reports the scores of our controller and the baseline.

B. Acrobot

The policy for the Acrobot swing-up was optimized for a horizon of $T = 2.0\,\mathrm{s}$, with $u_M = 3.0\,\mathrm{N}\cdot\mathrm{m}$. The policy is optimized over a total of 20 episodes, using γ_k scheduling reported in fig. 1 ($k_m = 5, K = 10$). The optimization steps for the acrobot's policy result in a learning plot similar to fig. 2.

Given the ideal conditions considered in this simulation, the control switches to an LQR controller after the swing-up. Under ideal circumstances, the LQR controller can stabilize the system at an unstable equilibrium by exerting zero final torque. The switching condition is obtained by checking if the system's state is within the LQR's region of attraction.

The Controller's strategy is depicted in fig. 4. This controller has a performance score of around 0.3, while with standard training it is slightly lower, around 0.2. The baseline TVLQR has a score lower than 0.1. Table I reports the scores of our controller and the baseline.

[2] https://ai-olympics.dfki-bremen.de/

V. Conclusions

In both systems, our MBRL approach is able to solve the global task with good swing-up time, handling the uniform initial state distribution.

Acknowledgements

Alberto Dalla Libera and Giulio Giacomuzzo were supported by PNRR research activities of the consortium iNEST (Interconnected North-Est Innovation Ecosystem) funded by the European Union Next GenerationEU (Piano Nazionale di Ripresa e Resilienza (PNRR) – Missione 4 Componente 2, Investimento 1.5 – D.D. 1058 23/06/2022, ECS_00000043). This manuscript reflects only the Authors' views and opinions, neither the European Union nor the European Commission can be considered responsible for them.

References

[1] F. Wiebe, S. Vyas, L. J. Maywald, S. Kumar, and F. Kirchner, "Realaigym: Education and research platform for studying athletic intelligence," in *Proceedings of Robotics Science and Systems Workshop Mind the Gap: Opportunities and Challenges in the Transition Between Research and Industry, New York*, 2022.

[2] 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.

[3] F. Wiebe, N. Turcato, A. Dalla Libera, C. Zhang, T. Vincent, S. Vyas, G. Giacomuzzo, R. Carli, D. Romeres, A. Sathuluri, M. Zimmermann, B. Belousov, J. Peters, F. Kirchner, and S. Kumar, "Reinforcement learning for athletic intelligence: Lessons from the 1st "ai olympics with realaigym" competition," in *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24* (K. Larson, ed.), pp. 8833–8837, International Joint Conferences on Artificial Intelligence Organization, 8 2024. Demo Track.

[4] F. Wiebe, N. Turcato, A. Dalla Libera, J. S. B. Choe, B. Choi, T. L. Faust, H. Maraqten, E. Aghadavoodi, M. Cali, A. Sinigaglia, *et al.*, "Reinforcement learning for robust athletic intelligence: Lessons from the 2nd'ai olympics with realaigym'competition," *arXiv preprint arXiv:2503.15290*, 2025.

[5] N. Turcato, A. Dalla Libera, G. Giacomuzzo, R. Carli, and D. Romeres, "Learning control of underactuated double pendulum with model-based reinforcement learning," 2024.

[6] F. Amadio, A. Dalla Libera, D. Nikovski, R. Carli, and D. Romeres, "Learning control from raw position measurements," in *2023 American Control Conference (ACC)*, pp. 2171–2178, IEEE, 2023.

[7] N. Turcato, A. Dalla Libera, G. Giacomuzzo, and R. Carli, "Teaching a robot to toss arbitrary objects with model-based reinforcement learning," in *2023 9th International Conference on Control, Decision and Information Technologies (CoDIT)*, pp. 1126–1131, 2023.

[8] N. Turcato, G. Giacomuzzo, M. Terreran, D. Allegro, R. Carli, and A. Dalla Libera, "Data efficient robotic object throwing with model-based reinforcement learning," *arXiv preprint arXiv:2502.05595*, 2025.

[9] Z. Liang, K. Zhao, J. Xie, and Z. Zhang, "Ultra-fast tuning of neural network controllers with application in path tracking of autonomous vehicle," *ISA Transactions*, vol. 149, pp. 394–408, 2024.

[10] F. Wiebe, S. Kumar, L. J. Shala, S. Vyas, M. Javadi, and F. Kirchner, "Open source dual-purpose acrobot and pendubot platform: Benchmarking control algorithms for underactuated robotics," *IEEE Robotics & Automation Magazine*, vol. 31, no. 2, pp. 113–124, 2024.

[11] C. E. Rasmussen, "Gaussian processes in machine learning," in *Summer school on machine learning*, pp. 63–71, Springer, 2003.

[12] A. Carè, R. Carli, A. Dalla Libera, D. Romeres, and G. Pillonetto, "Kernel methods and gaussian processes for system identification and control: A road map on regularized kernel-based learning for control," *IEEE Control Systems Magazine*, vol. 43, no. 5, pp. 69–110, 2023.

[13] R. E. Caflisch, "Monte carlo and quasi-monte carlo methods," *Acta numerica*, vol. 7, pp. 1–49, 1998.

[14] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proc of COMPSTAT'2010*, pp. 177–186, Springer, 2010.

[15] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[16] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *JMLR*, vol. 15, no. 1, pp. 1929–1958, 2014.

[17] F. Amadio, A. Dalla Libera, D. Nikovski, R. Carli, and D. Romeres, "Learning control from raw position measurements," 2023.

[18] J. Quiñonero-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.

[19] X. Wang, Y. Chen, and W. Zhu, "A survey on curriculum learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 9, pp. 4555–4576, 2022.

[20] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.