Athletic Intelligence Olympics challenge with Model-Based Reinforcement Learning

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

TR2023-111 September 02, 2023

Abstract

In this report, we describe the solution we propose for the AI Olympics competition held at IJCAI 2023. Our solution is based on a recent Model-Based Reinforcement Learning algorithm named MC-PILCO. Besides briefly reviewing the algorithm, we discuss the most critical aspects of the MC-PILCO implementation in the environments at hand.

International Joint Conference on Artificial Intelligence 2023

Athletic Intelligence Olympics challenge with Model-Based Reinforcement Learning

Alberto Dalla Libera[1] , **Niccolo' Turcato**[1] , **Giulio Giacomuzzo**[1] , **Ruggero Carli**[1] and **Diego Romeres**[2]
[1]Department of Information Engineering, University of Padova, Italy.
[2]Mitsubishi Electric Research Laboratories, Cambridge, MA, USA.
alberto.dallalibera@unipd.it, turcatonic@dei.unipd.it, giulio.giacomuzzo@phd.unipd.it, carlirug@dei.unipd.it, romeres@merl.com

Abstract

In this report, we describe the solution we propose for the AI Olympics competition held at IJCAI 2023. Our solution is based on a recent Model-Based Reinforcement Learning algorithm named MC-PILCO. Besides briefly reviewing the algorithm, we discuss the most critical aspects of the MC-PILCO implementation in the environments at hand.

1 Introduction

In this short paper, we present the Reinforcement Learning (RL) [Sutton and Barto, 2018] approach we implemented to tackle the AI Olympics competition held at IJCAI 2023[1]. Our algorithm, named Monte-Carlo Probabilistic Inference for Learning COntrol (MC-PILCO) [Amadio *et al.*, 2022], is a Model-Based (MB) RL algorithm that proved remarkably data-efficient in several low-dimensional benchmarks, such as a cart-pole, a ball & plate, and a Furuta pendulum, both in simulation and real setups. MC-PILCO exploits data collected by interacting with the system to derive a model of the system dynamics, and optimizes the policy by simulating the system, rather than optimizing the policy directly on the actual system. When considering physical systems, this kind of approach can be highly performing and more data-efficient than Model-Free (MF) solutions.

This paper is organized as follows: Section 2 introduces the goal and the settings of the competition. Section 3 presents the MC-PILCO algorithm. Section 4 reports the experiments that have been performed, finally Section 5 concludes the paper.

2 Goal of the competition

The challenge considers a 2 degrees of freedom (dof) underactuated pendulum [Wiebe *et al.*, 2022; Wiebe *et al.*, 2023] 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 an horizon of $T = 10\,\mathrm{s}$. Controllers are evaluated by a performance and a robustness score.

[1]https://ijcai-23.dfki-bremen.de/competitions/ai_olympics/

3 MC-PILCO for underactuated robotics

In the first part of this section we briefly review MC-PILCO, then, in the second part, we discuss its application to the considered problem.

3.1 MC-PILCO review

MC-PILCO is a MB policy gradient algorithm, in which 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's consists of the succession of several attempts to solve the desired task, also called trials. 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.

Model Learning

MC-PILCO relies on GP Regression (GPR) to learn the system dynamics [Rasmussen, 2003]. In our previous work, [Amadio *et al.*, 2022], 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 means of 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. Given an input-output training dataset $\mathcal{D}^{(i)} = \{\tilde{\boldsymbol{X}}, \boldsymbol{y}^{(i)}\}$, where the inputs are $\tilde{\boldsymbol{X}} = [\tilde{\boldsymbol{x}}_1^T, \dots, \tilde{\boldsymbol{x}}_n^T]^T$, and the outputs $\boldsymbol{y}^{(i)} = [y_1^{(i)}, \dots y_n^{(i)}]^T$ are measurements of $\Delta_t^{(i)}$ at time instants $t = 0, \dots, T_{tr}$, GPR assumes the following probabilistic model,

$$\boldsymbol{y}^{(i)} = f^i(\tilde{\boldsymbol{X}}) + \boldsymbol{e}, \tag{4}$$

where vector $\boldsymbol{e}$ accounts for noise, defined a priori as zero mean independent Gaussian noise with variance σ_i^2. The unknown function f^i is defined a priori as a GP with mean $m_\Delta^{(i)}$ and covariance defined by a kernel function $k(\tilde{\boldsymbol{x}}_{t_i}, \tilde{\boldsymbol{x}}_{t_j})$, namely, $f^i(\tilde{\boldsymbol{X}}) \sim N(m_\Delta^{(i)}, K_{\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}})$, where the element of $K_{\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}}$ at row r and column j is $E[\Delta_{t_r}^{(i)}, \Delta_{t_j}^{(i)}] = k(\tilde{\boldsymbol{x}}_{t_r}, \tilde{\boldsymbol{x}}_{t_j})$. The mean function $m_\Delta^{(i)}$ can be derived from prior knowledge of the system, or can be set as the null function if no information is available. Instead, as regards the kernel function, one typical choice to model continuous functions is the squared-exponential kernel:

$$k(\tilde{\boldsymbol{x}}_{t_i}, \tilde{\boldsymbol{x}}_{t_j}) := \lambda^2 \boldsymbol{e}^{-\|\tilde{\boldsymbol{x}}_{t_i} - \tilde{\boldsymbol{x}}_{t_j}\|_{\Lambda^{-1}}^2} \tag{5}$$

where λ and Λ are trainable hyperparameters tunable by maximizing the marginal likelihood (ML) of the training samples [Rasmussen, 2003].

As explained in [Rasmussen, 2003], the posterior distributions of each $\Delta_t^{(i)}$ given $\mathcal{D}^i$ are Gaussian distributed, with mean and variance expressed as follows:

$$\begin{aligned}
&\mathbb{E}[\hat{\Delta}_t^{(i)}] = m_\Delta^{(i)}(\tilde{\boldsymbol{x}}_t) + K_{\tilde{\boldsymbol{x}}_t\tilde{\boldsymbol{X}}}\Gamma_i^{-1}(\boldsymbol{y}^{(i)} - m_\Delta^{(i)}(\tilde{\boldsymbol{X}}))\\
&var[\hat{\Delta}_t^{(i)}] = k_i(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{x}}_t) - K_{\tilde{\boldsymbol{x}}_t\tilde{\boldsymbol{X}}}\Gamma_i^{-1}K_{\tilde{\boldsymbol{X}}\tilde{\boldsymbol{x}}_t}\\
&\Gamma_i = K_{\tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}} + \sigma_i^2 I
\end{aligned} \tag{6}$$

Then, also the posterior distribution of the one-step ahead transition model in (3) is Gaussian, namely,

$$p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t, \mathcal{D}) \sim \mathcal{N}(\boldsymbol{\mu}_{t+1}, \Sigma_{t+1}) \tag{7}$$

with mean $\boldsymbol{\mu}_{t+1}$ and covariance Σ_{t+1} derived combining (3) and (6).

Policy Update

In the policy update phase, the policy is trained in order to minimize the expected cumulative cost in eq. (2) with the expectation computed w.r.t. the one-step ahead probabilistic model in eq. (7). This requires the computation of long-term distributions starting from the initial distribution $p(\boldsymbol{x}_0)$ and eq. (7), which is not possible in closed form. MC-PILCO resorts to Monte Carlo sampling [Caflisch, 1998] 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. (7) and the current policy. Then, the expectations in eq. (2) are approximated by the mean of the simulated particles costs, namely,

$$\hat{J}(\boldsymbol{\theta}) = \sum_{t=0}^{T}\left(\frac{1}{N}\sum_{n=1}^{N} c\left(\boldsymbol{x}_t^{(n)}\right)\right) \tag{8}$$

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) [Bottou, 2010], applying the reparameterization trick to differentiate stochastic operations [Kingma and Welling, 2013].

The authors of [Amadio *et al.*, 2022] proposed the use of dropout [Srivastava *et al.*, 2014] of the policy parameters $\boldsymbol{\theta}$ to improve exploration and increase the ability to escape from local minima during policy optimization of MC-PILCO.

3.2 MC-PILCO for underactuated robotics

The task in object presents a number of 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, [Amadio *et al.*, 2022] and [Amadio *et al.*, 2023] derived a MBRL controller for a Furuta Pendulum 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 100 Hz.

The second issue is that controllers are evaluated by a performance and robustness score. In the robustness test, the characteristics of the system and data acquisition vary. This is an issue for data-driven solutions like MC-PILCO since retraining of the controller is not allowed. For this reason, we decided to focus only on the performance score, even if in our previous work we showed that MC-PILCO can be robust to noise and filtering by including these effects in the simulation.

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

$$B\boldsymbol{u}_t = M(\boldsymbol{q}_t)\ddot{\boldsymbol{q}}_t + n(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t), \tag{9}$$

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

$$m_\Delta(\tilde{\boldsymbol{x}}_t) = \begin{bmatrix} m_\Delta^{(1)} \\ m_\Delta^{(2)} \end{bmatrix} := T_s \cdot M^{-1}(\boldsymbol{q}_t)(B\boldsymbol{u}_t - n(\boldsymbol{q}_t, \dot{\boldsymbol{q}}_t)) \tag{10}$$

as the mean function in eq. (6). It is important to point out that eq. (10) is nearly perfect to approximate the system when T_s is sufficiently small, but it becomes unreliable as T_s grows. In particular, with $T_s = 0.01\,\text{s}$ the predictions of eq. (10) 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 Regressors, see [Quiñonero-Candela and Rasmussen, 2005] for a detailed description.

An important aspect of policy optimization is the particles initialization, in this case, it is guaranteed that the system will always start at $\boldsymbol{x}_0 = \bar{0}$, therefore the initial distribution can be set to $p(\boldsymbol{x}_0) \sim \mathcal{N}(\bar{0}, \epsilon I)$ with ϵ in the order of 10^{-4}.

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:

$$c_{st}(\boldsymbol{x}_t) = 1 - e^{-\|\boldsymbol{q}_t - \boldsymbol{q}_G\|^2_{\Sigma_c}} \qquad \Sigma_c = \operatorname{diag}\left(\frac{1}{\ell_c}, \frac{1}{\ell_c}\right), \tag{11}$$

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 [Amadio *et al.*, 2022]:

$$\begin{aligned} \pi_{\boldsymbol{\theta}}(\boldsymbol{x}_t) &= u_M \tanh\left(\sum_{i=1}^{N_b} \frac{w_i}{u_M} e^{-\|\boldsymbol{a}_i - \phi(\boldsymbol{x}_t)\|^2_{\Sigma_\pi}}\right) \\ \phi(\boldsymbol{x}_t) &= [\dot{\boldsymbol{q}}_t^T, \cos(\boldsymbol{q}_t^T), \sin(\boldsymbol{q}_t^T)]^T \end{aligned} \tag{12}$$

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: $\Sigma_\pi \in \mathbb{R}^{6\times 6}$, $a_i \in \mathbb{R}^6$, $w_i \in \mathbb{R}$ 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. Given the ideal conditions considered in this simulation, for the purpose of the challenge we switched to an LQR controller after swing-up. Under ideal circumstances, the LQR controller has the capability to stabilize the system at an unstable equilibrium by exerting zero final torque.

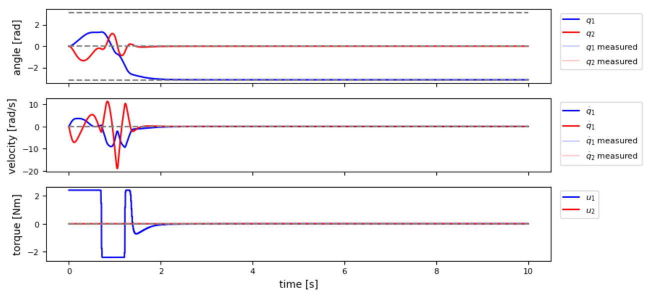

Figure 1: Simulation of the Pendubot system (500 Hz), under control of the policy trained with MC-PILCO.

Controller	Perf. score	Rob. score	Avg. score
TVLQR	0.827	0.95	0.8885
MC-PILCO	0.891	0.871	0.881
iLQR MPC stab.	0.845	0.86	0.8525
iLQR Riccati	0.847	0.592	0.7195
iLQR MPC	0.861	0.2	0.5305
Energy PFL	0.594	0.117	0.3555

Table 1: Pendubot scores comparison.

4 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 [Amadio *et al.*, 2022]. All the code was implemented in Python with the PyTorch [Paszke *et al.*, 2017] library.

For both robots, we use the model described in Section 3.1, with mean function from eq. (10) and kernel function from eq. (5). The max torque u_M was set to conservative values, to optimize the score of the controller. 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.

4.1 Pendubot

The policy for the Pendubot swing-up was optimized for a horizon of $T = 3.0\,\text{s}$, with u_M set to 40% of the torque

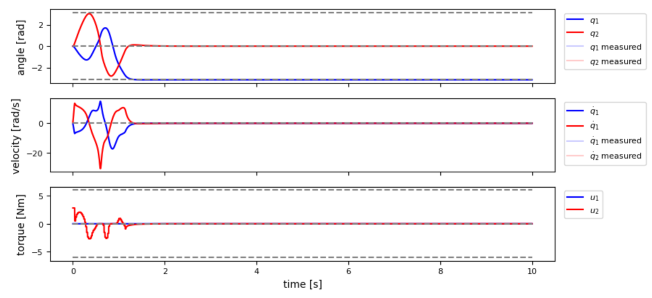

Figure 2: Simulation of the Acrobot system (500 Hz), under control of the policy trained with MC-PILCO.

limit of the actuator. The sampling time for model and policy learning is $0.01\,\mathrm{s}$, thus the control rate is $100\,\mathrm{Hz}$. The condition for switching to the LQR stabilization is $\boldsymbol{q}_1 > 3.1\,\mathrm{rad}$. The Controller's strategy is depicted in fig. 1, in fig. 3 (left) we report the robustness bar charts. This controller has a performance score of 0.891 and a robustness score of 0.852. In table 1 we compare our controller's score with other tested control strategies.

4.2 Acrobot

The policy for the Pendubot swing-up was optimized for a horizon of $T = 3.0\,\mathrm{s}$, with u_M set to 50% of the torque limit of the actuator. The sampling time for model and policy learning is $0.02\,\mathrm{s}$, thus the control rate is $50\,\mathrm{Hz}$. The condition for switching to the LQR stabilization is $\boldsymbol{q}_1 > 2.8\,\mathrm{rad}$. The Controller's strategy is depicted in fig. 2, in fig. 3 (right) we report the robustness bar charts. This controller has a performance score of 0.869 and a robustness score of 0.73. In table 2 we compare our controller's score with other tested control strategies.

Controller	Perf. score	Rob. score	Avg. score
TVLQR	0.783	0.861	0.822
MC-PILCO	0.869	0.634	0.7515
iLQR MPC stab.	0.806	0.685	0.7459
Energy PFL	0.728	0.503	0.6155
iLQR Riccati	0.831	0.298	0.5645
iLQR MPC	0.796	0.089	0.4425

Table 2: Acrobot scores comparison.

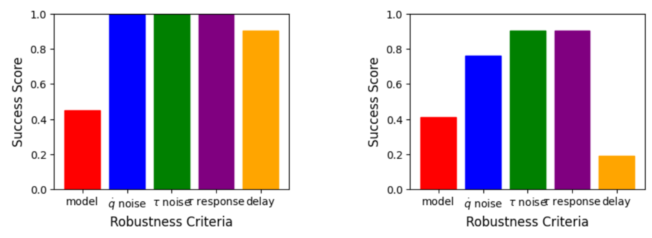

Figure 3: Pendubot (left) and Acrobot (right) robustness bar charts.

5 Conclusions

In both systems, our MBRL approach reaches a performance score higher than other tested approaches, while remaining competitive w.r.t. the robustness score.

References

[Amadio *et al.*, 2022] Fabio Amadio, Alberto Dalla Libera, Riccardo Antonello, Daniel Nikovski, Ruggero Carli, and Diego Romeres. Model-based policy search using monte carlo gradient estimation with real systems application. *IEEE Transactions on Robotics*, 38(6):3879–3898, 2022.

[Amadio *et al.*, 2023] Fabio Amadio, Alberto Dalla Libera, Daniel Nikovski, Ruggero Carli, and Diego Romeres. Learning control from raw position measurements, 2023.

[Bottou, 2010] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proc of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[Caflisch, 1998] Russel E Caflisch. Monte carlo and quasi-monte carlo methods. *Acta numerica*, 7:1–49, 1998.

[Kingma and Welling, 2013] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[Paszke *et al.*, 2017] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[Quiñonero-Candela and Rasmussen, 2005] Joaquin Quiñonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6(65):1939–1959, 2005.

[Rasmussen, 2003] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer, 2003.

[Srivastava *et al.*, 2014] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 15(1):1929–1958, 2014.

[Sutton and Barto, 2018] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[Wiebe *et al.*, 2022] Felix Wiebe, Shubham Vyas, Lasse Jenning Maywald, Shivesh Kumar, and Frank Kirchner. Realaigym: Education and research platform for studying athletic intelligence. In *RSS Online Proceedings, Mind the Gap: Opportunities and Challenges in the Transition Between Research and Industry, New York*, 2022.

[Wiebe *et al.*, 2023] Felix Wiebe, Shivesh Kumar, Lasse Shala, Shubham Vyas, Mahdi Javadi, and Frank Kirchner. An open source dual purpose acrobot and pendubot platform for benchmarking control algorithms for underactuated robotics. *IEEE Robotics and Automation Magazine*, 2023. under review.