

Reward-Driven Hierarchical Learning for Visual Tracking and Control Tasks

Akihiro Kubo

A dissertation submitted for the degree of
Doctor of Informatics

Kyoto University
2024

Supervised by:

Prof. Shin Ishii

Examination committee:

Prof. Shin Ishii (Chair)

Prof. Jun Morimoto

Prof. Taishin Nomura

Abstract

Real-world applications of reinforcement learning (RL) face two major challenges: *sample efficiency*—the need to learn effective policies from limited interactions—and *adaptability*—the ability to cope with different or changing environments. This thesis tackles these challenges through reward-driven hierarchical learning in control and visual tracking tasks, where “hierarchical” refers not to subgoal-based RL architectures or learned internal rewards, but rather to layered policy structures and meta-level optimization guided by reward signals.

First, to enhance both sample efficiency and adaptability across multiple environments with different physical properties (e.g., friction), we proposed an RL framework that integrates central pattern generators (CPGs) in a hierarchically layered policy architecture. By embedding biological knowledge of periodic motion, the exploration space is effectively narrowed, enabling faster learning with fewer samples (Chapter 2). We also introduce a robustness-enhancing extension that alleviates the risk of overfitting to specific training samples or conditions (Chapter 3).

Second, aiming to improve sample efficiency without relying on prior knowledge, we developed a model-based RL (MBRL) method (Chapter 4). By learning and simulating environment dynamics, this approach reduces the need for extensive real-world trials. We propose a strategy to mitigate the trade-off between model bias and dependence on outdated samples, and to achieve efficient policy optimization within limited runtime cost.

Third, in Chapter 5, we address adaptability in a highly non-stationary domain: visual object tracking (VOT). Here, targets and backgrounds show continuous appearance changes, posing a persistent challenge for real-time tracking. We presented a reward-driven method that adaptively adjusts a tracker’s learning rates within a two-stage meta-learning framework: the system first learns a general learning-rate adjustment strategy (meta-training) and then applies it to individual targets (meta-testing). This effectively constitutes a form of hierarchical learning driven by reward signals, improving tracking performance under complex change patterns.

Collectively, these studies show how reward-driven hierarchical approaches can

tackle sample efficiency and adaptability from different angles. The results serve as a foundation for broader real-world applications, paving the way for efficient and adaptive policy learning under realistic conditions.

Acknowledgments

I would like to express my heartfelt gratitude to the following individuals.

First, I am deeply grateful to Professor Ishii for his insightful guidance, patience, and planning, which enabled me to pursue my research directions. During my time in his laboratory, I had the privilege of learning from exceptional researchers. I especially thank Dr. Meshgi for his mentorship in visual object tracking, which led to the idea of combining it with reinforcement learning.

In my doctoral program, discussions with Dr. Parmas, a leading researcher in model-based reinforcement learning, helped refine my ideas and encouraged me to delve into foundational studies in the field.

I am also thankful for the quadruped robot control project guided by Professor Ishii, through which I met many inspiring colleagues. Notably, I appreciate Mr. Nakanishi for broadening my perspective through our discussions and Professor Nakamura for his expert advice and support during paper writing. Dr. Koyamada provided me very important directions to polish up the doctoral dissertation and social skills. Mr. Matsuoka met well my request to develop foundational systems such as simulating multiple robots. Secretaries in the lab so far have supported my office work, what I am not good at. Finally, I would like to take this opportunity to express my gratitude to everyone who has been involved in my research activities so far.

Akihiro Kubo

Contents

1	Introduction	1
2	Hierarchical Reinforcement Learning with Central Pattern Generators	6
2.1	Introduction	6
2.2	Backgrounds	9
2.2.1	Oscillator Model	10
2.2.2	Reinforcement Learning	11
2.3	Hierarchical Reinforcement Learning for a CPG Controller	13
2.3.1	Control Signal	14
2.3.2	Joint Policy	16
2.3.3	SAC Algorithm for HRL-CPG	16
2.4	Experiments and Results	17
2.4.1	Task Description	18
2.4.2	Main Results	20
2.4.3	Comparisons	23
2.5	Concluding Remarks	25
3	Orthogonal Adversarial Hierarchical Reinforcement Learning with CPGs	27
3.1	Introduction	27
3.2	Orthogonal Adversarial Method	28
3.3	Application of OA to HRL-CPG	29
3.3.1	CPG-OA-SAC Algorithm Overview	29
3.4	Experiments and Results	30
3.4.1	Sample Efficiency in Low-Friction Environments	31
3.4.2	Robustness to Adversarial and Random Noise	31
3.5	Conclusion	33

4	Double Horizon Model-based Policy Optimization	35
4.1	Introduction	35
4.2	Background	37
4.2.1	Notation and Problem Setting	37
4.2.2	Model-based Actor-Critic Methods	38
4.2.3	Model-based Policy Optimization (MBPO)	39
4.2.4	Training Rollout-based Methods	40
4.3	Double Horizon Model-Based Policy Optimization	41
4.4	Experiments	42
4.4.1	Comparison of DHMBPO with Ablated Algorithms	43
4.4.2	Hyper-parameter Sensitivity Analysis	46
4.4.3	Comparison with State-of-The-Art Algorithms	46
4.5	Related Work	50
4.6	Conclusion	51
5	A Meta-Q-Learning Approach to Visual Object Tracking	52
5.1	Introduction	52
5.2	Related Studies	54
5.3	Background	55
5.3.1	Discriminative Correlation Filter	55
5.3.2	Reinforcement Learning	56
5.4	Proposed Tracking Framework	57
5.4.1	How to Weight Samples in a DCF	57
5.4.2	Necessity of Adaptation	58
5.4.3	Q-Correlation Filter	58
5.4.4	Adaptive update through Deep Q-learning	60
5.5	Evaluation	60
5.5.1	Definitions of the RL Environment	61
5.5.2	Training and Evaluation	62
5.5.3	Implementation Details	63
5.5.4	Quantitative Results	63
5.5.5	Qualitative Results	65
5.5.6	Attribute Based Evaluation	66
5.6	Discussion	67
5.7	Conclusion	68

6	Conclusion, Limitations, and Future Directions	69
6.1	Limitations	69
6.1.1	Limited Real-World Validation and the Simulation Gap	69
6.1.2	Overfitting and Limited Exploration	69
6.1.3	Inductive Bias in Model Classes	70
6.1.4	Lack of Integrated Systems	70
6.2	Future Directions	71
6.2.1	Applying HRL-CPG Methods to a Real Quadruped Robot and Identifying Challenges	71
6.2.2	Integrating CPG Frameworks and Model-Based RL for Enhanced Control	72
6.2.3	Pursuing Autonomous Systems through Specialized Recognition Modules	72
A	Appendix of Chapter 2	74
A.1	Soft Actor-Critic Algorithm	74
A.2	Storoboscopic Photography while Walking	76
A.3	Evaluation of Learning Performance: Failure Rate and Progression Dis- tance	77
A.4	Supplementary Discussion on Hyperparameters	78
A.5	Deep Transition	79
B	Appendix of Chapter 4	81
B.1	Toy Problem: Policy Evaluation on Random Tabular MDPs	81
B.2	Implementation Detail	82
B.2.1	Neural network architecture	82
B.2.2	Model Learning	85
B.2.3	Early Stopping	85
B.3	Experiment Detail	86
B.4	Result Detail	87
B.5	Additional Experiments for Discussion	87
B.5.1	Detailed Results of Hyper-parameter Sensitivity Analysis	87
B.5.2	Measurements during Training Rollout	88
B.5.3	Application of Distribution Rollout to SAC-SVG(H) Implemen- tation	89
B.5.4	Comparison of Model Learning Performance	89
	Bibliography	98

List of Figures

2.1	Schematic diagram of the applied phase oscillator network (PON). . .	10
2.2	Behavior of the phase oscillator network	11
2.3	Schematic diagrams of the CPG-based control	11
2.4	Joint policy	14
2.5	Quadruped robot REX	18
2.6	Learning curves for each five environments	22
3.1	Learning curves in a low-friction environment	31
3.2	Performance under adversarial noise in five environments.	32
3.3	Performance under random Gaussian noise in five environments.	33
4.1	Comparison of value estimation errors obtained with different lengths of Distribution Rollout and Training Rollout	37
4.2	Comparison of variant DHMBPO algorithm on GYM tasks and DMC tasks	45
4.3	Development of the model's RMedSE for DHMBPO and SVG	46
4.4	Aggregation metrics on 10 tasks for variants of different DR horizon and variants of different TR horizon.	47
4.5	Comparison of sample efficiency with state-of-the-art methods on GYM tasks	48
4.6	Aggregation metrics normalized by DHMBPO's scores	49
5.1	DCF learning is self-contained	53
5.2	Snapshots of tracking results and a plot for a typical profile of the learn- ing rate over time	58
5.3	Schematic of proposed Meta-Q-learning of DCF-based tracker	59
5.4	Success plots comparing our DQCF with five state-of-the-art DCF-based trackers on OTB-2015	66
5.5	Snapshots of tracking results and a plot of a typical learning rate profile	67

A.1	Storoboscopic photographs: two side-view images of successful and failed cases (a and b), and a front-view image of a failure case (c). The vertical red bars are displayed at 1-meter intervals.	76
A.2	A screen shot of a preliminary experiment on surfaces with zebra-patterned bands. The white bands have a lower friction coefficient than the black bands.	76
A.3	Failure rate and the progression distance when succeeded.	77
A.4	Fail rate and locomotion distance when succeeded at final interaction steps.	77
A.5	Learning curves in environment 4 with tuned hyperparameters.	79
B.1	Ratios of root mean squared error for the MVE estimator compared to the target critic	82
B.2	Runtime comparison on GYM tasks.	87
B.3	Comparison of sample efficiency with latent model-based methods on DMC tasks	92
B.4	Runtime comparison with latent model-based methods on DMC tasks. .	93
B.5	Development of test return and the norm of policy gradient over environment steps.	94
B.6	Measurements at each step during training rollout	95
B.7	Sample efficiency curve of DMSAC-SVG(H).	95
B.8	Development of critic loss over the number of environment steps of DA+SAC-SVG(H).	96
B.9	RMSE of model trained on 5K environment steps dataset	96
B.10	RMSE of model trained on 20K environment steps dataset	97
B.11	RMSE of model trained on 50K environment steps dataset	97

List of Tables

2.1	Terrain configurations	19
2.2	Characteristics of comparative methods	24
2.3	Overview of performance after learning	24
4.1	Runtime on GYM and its ratio to DHMBPO’s runtime (in hours). . . .	49
5.1	Average AUCs for the DQCF and BACF on two benchmark datasets .	64
5.2	Comparison of AUC scores for each attribute	68
A.1	Hyperparameters commonly used in experiments in Section 2.4.	78
B.1	Hyper-parameters for DHMBPO used across all experiments in this study	86
B.2	Summary of RMSE values after 100K epochs.	91

Abbreviations

Abbreviation	Definition
AUC	Area Under Curve
CPG	Central Pattern Generator
DCF	Discriminative Correlation Filter
DNN	Deep Neural Network
DRL	Deep Reinforcement Learning
GPU	Graphics Processing Unit
HOG	Histogram of Oriented Gradient
HRL	Hierarchical Reinforcement Learning
MBRL	Model-Based Reinforcement Learning
MDP	Markov Decision Process
MVE	Model-based Value Expansion
PID	Proportional-Integral-Differential
PON	Phase Oscillator Network
PPO	Proximal Policy Optimization
PSR	Peak-to-Sidelobe Ratio
RL	Reinforcement Learning
SAC	Soft Actor-Critic
SVG	Stochastic Value Gradient
UTD	Update-To-Data
VOT	Visual Object Tracking

Copyright Notice. This thesis contains the Author's Accepted Manuscript of the article entitled "A Meta-Q-Learning Approach to Discriminative Correlation Filter based Visual Tracking" by Kubo, A., Meshgi, K. & Ishii, S. This work was published in the Journal of Intelligent & Robotic Systems, Volume 101, Issue 11, 2021, with DOI: <https://doi.org/10.1007/s10846-020-01273-2>. In accordance with Springer's self-archiving policy, this version has been deposited in the repository and is made publicly available 12 months after official publication. The final publication is available at <http://link.springer.com>.

Chapter 1

Introduction

Reinforcement learning (RL) [Sutton, 1991] is a framework in which an agent autonomously acquires an optimal policy mostly from reward signals, and it has attracted substantial attention across various domains. The integration of RL with deep learning technologies demonstrated the potential of RL especially when superhuman performance was achieved in Atari video games [Mnih et al., 2015]. Subsequent research has expanded and diversified RL applications, encompassing perfect-information board games such as Go and Shogi [Silver et al., 2018, Schrittwieser et al., 2020], imperfect card games such as Poker [Yakovenko et al., 2016] and Contract Bridge [Kita et al., 2024], simulation-based robot control tasks [Janner et al., 2019, Hansen et al., 2024], and even real-world robotic systems [Kalashnikov et al., 2018, Agarwal et al., 2023]. This has led to multifaceted developments including hierarchical approaches [Pateria et al., 2021], enhanced representation learning [Grill et al., 2020, Schwarzer et al., 2021], and integrations with optimal control techniques [Hansen et al., 2024].

However, many of these successful cases depend on computational environments where conducting a large number of trials is relatively easy. In real-world applications, especially in robotics, performing a vast number of real-world trials or collecting extensive data can be time-consuming, costly, and raise safety concerns. For instance, there is a risk of damaging the robot itself or harming human operators.

Moreover, the real world is inevitably non-stationary, meaning that conditions change over time. Even if sufficient training data under fixed conditions could be obtained, once those conditions change, further large-scale data collection and re-training would be required. Under such circumstances, improving “sample efficiency,” i.e., the ability to acquire useful policies from limited trials, is indispensable. Additionally, ensuring adaptability to changing conditions is also critical. Together, these factors make improving sample efficiency and ensuring adaptability to non-stationary environments central challenges for obtaining effective real-world control policies, and these

challenges form key topics of this thesis.

One straightforward way to approach these issues is to minimize direct interaction with the real environment and instead leverage simulated robot or dynamical systems. Simulations allow for fast and safe trials, partially mitigating the sample efficiency problem. Furthermore, advances in sophisticated simulation environments, such as those enabling highly parallelized simulations [Makoviychuk et al., 2021, Freeman et al., 2021], have reduced the cost of large-scale data collection and simulation-driven technological development [Bellegarda and Ijspeert, 2022, Agarwal et al., 2023]. Additionally, various techniques aim to bridge the simulation-to-reality gap: domain randomization diversifies environment parameters to familiarize the agent with a wide range of conditions [Tobin, 2019], domain adaptation corrects differences in feature distributions between the simulator (source domain) and the real world (target domain) [Bousmalis et al., 2018], and meta-RL [Arndt et al., 2020, Wang et al., 2016] learns from diverse task distributions to rapidly adapt to new conditions. All these efforts actively work to narrow the gap between simulation and reality.

Nevertheless, even combining these techniques does not fully ensure adaptability to non-stationary environments nor guarantee sample efficiency improvements independent of real-world interaction [Zhao et al., 2020]. Thus, directly improving these properties in RL remains an important research direction. In particular, this thesis explores approaches that can be seen as forms of reward-driven hierarchical learning—not in the sense of subgoal decomposition or learned internal rewards, but rather through layered policy architectures and meta-level optimization driven by reward signals.

One such direction involves incorporating prior knowledge. For example, by introducing the biological principle of central pattern generators (CPGs) [Whelan, 1996, Matsuoka, 1985] into robotic control tasks, the exploration space can be effectively reduced [Mori et al., 2004, Matsubara et al., 2006]. Unlike domain randomization, which requires complex, task-specific randomization designs, leveraging general biological insights such as periodicity provides a means to improve sample efficiency. However, relying on prior knowledge may limit exploration, and it is essential to define a parametric model class for periodic signals (e.g., parametric periodic waveforms) and combine it with RL or reward-driven policy optimization. Hence, we employ a hierarchical (layered) policy structure that integrates CPGs with RL, allowing fast adaptation driven by reward feedback.

On the other hand, model-based RL (MBRL) reduces reliance on real samples and can improve sample efficiency by learning a model of an environment by data-driven manner and simulating policy behaviors within the model space. Ideally, if the model is highly accurate, good policies could be obtained without real-world interaction. In

practice, however, especially in the early stages of RL, this assumption does not hold, leading to policy degradation due to model bias and reliance on samples obtained via premature policies. Consequently, diverse studies have explored various ways of defining and utilizing models [Plaat et al., 2023]. Recent deep MBRL methods have demonstrated success in high-dimensional control tasks, such as humanoid robots in simulation [Janner et al., 2019, Hansen et al., 2024]. These approaches can be roughly divided into two categories: (1) generating numerous virtual samples from learned state transitions and applying model-free techniques to calculate policy gradient [Janner et al., 2019, Frauenknecht et al., 2024], and (2) using predictions from differentiable models to compute policy gradient [Amos et al., 2021, Deisenroth and Rasmussen, 2011, Parmas et al., 2018]. Each approach has its trade-offs in terms of sample efficiency and optimization stability.

MBRL can build task-agnostic models without relying on prior knowledge, contrasting with the CPG-based approach that uses biological insights like periodicity to restrict the exploration space.

In the research presented in this thesis, based on the above considerations, we primarily focus on improving sample efficiency and adaptability in different / non-stationary environments. First, in the RL domain, we conduct three studies: two leveraging CPG-based methods and one employing MBRL. CPG-based methods that utilize hierarchically arranged policies can improve sample efficiency by narrowing the exploration space, but there is a risk of overfitting to the limited samples obtained under certain conditions, potentially reducing generality. Indeed, in the CPG-based hierarchical approach examined in Chapter 2, adapting to an environment with low friction (making it slippery) proved difficult. This indicates that while exploiting control periodicity and policy hierarchy enhances sample efficiency, it can also lead to overfitting and instability in challenging conditions. To address this, Chapter 3 builds on the Chapter 2 method and investigates an approach to improve robustness under observational noise, mitigating instability caused by overfitting. Furthermore, Chapter 4 employs a new MBRL method to pursue sample efficiency without relying on prior knowledge, attempting to ease the trade-off between the model bias and the old-sample dependence.

Meanwhile, visual object tracking (VOT) is a fundamental task in computer vision that sequentially estimates the position of a target object. It has a wide range of applications, including surveillance [Eisenbach et al., 2012], automotive driver assistance [Van Brummelen et al., 2018], and robot vision [Gross et al., 2009, Gad et al., 2022]. VOT is challenging because the target and background change non-stationarily, and multiple factors such as occlusions, illumination shifts, and shape deformations

act simultaneously. Moreover, high-speed and stable tracking is required [Yeong et al., 2021]. The necessity of adapting to non-stationary conditions in VOT makes it a well-suited example to revisit the importance of adaptability, one of central themes of this thesis. Discriminative correlation filter (DCF)-based methods [Bolme et al., 2010] have been widely adopted for their real-time capability, but most rely on fixed learning rates, making rapid adaptation to sudden appearance changes difficult. If the target or background changes abruptly, overfitting or insufficient updates can occur, degrading tracking performance. As an alternative, instead of designing rules in advance or annotating optimal learning rates frame-by-frame, in Chapter 5, we propose a reward-driven approach for an adaptive adjustment of learning rates within a two-stage meta-learning, which can be viewed as a hierarchical procedure—meta-training learns a general update strategy, then meta-testing applies it to individual targets, all guided by reward signals.

In summary, this thesis presents four studies:

- Chapter 2: Integrating CPG into a hierarchically layered policy architecture in RL, to tackle a quadruped robot locomotion task in multiple environments with different physical properties. A baseline gait is provided, and the method is tested on how efficiently it adapts with limited trials, demonstrating that leveraging periodicity can provide sample efficiency.
- Chapter 3: Building on the Chapter 2 approach, this study introduces adversarial samples in a challenging environment to enhance robustness, mitigating instability and overfitting issues.
- Chapter 4: Employing MBRL to improve sample efficiency without relying on prior knowledge such as periodicity. By adjusting prediction horizons in a two-stage modeling approach, the trade-off between model bias and outdated sample dependence is alleviated. The effectiveness is confirmed through a diverse benchmark set.
- Chapter 5: In the VOT domain, a reward-driven dynamic learning rate optimization is proposed, overcoming the difficulties caused by fixed parameters. The method leverages a meta-learning framework to adaptively tune the update rule, improving tracking performance in non-stationary environments.

We position these four studies as complementary efforts to partially address sample efficiency and adaptability in real-world-like conditions. In the final chapter, we conclude the thesis by discussing the main findings, their limitations, and possible future directions. The findings contribute useful insights into multi-faceted challenges related

to sample efficiency, adaptability, robustness, and non-stationary environments in both RL and VOT fields, advancing the applicability of reward-driven learning toward real-world scenarios.

Chapter 2

Hierarchical Reinforcement Learning with Central Pattern Generators

In this chapter, we present a data-driven deep reinforcement learning (DRL) method for the optimization of a hierarchically structured control policy that includes Central Pattern Generators (CPGs). This method, which is as a whole referred to as the hierarchical reinforcement learning with the CPG (HRL-CPG), is then evaluated with the expectation of its applicability in real robot controls. We observed that stable gait motions were gained in a reasonably small number of trials and errors for four out of the five environments. The modification of one hyperparameter allowed for appropriate adaptability to the remaining environment. Thus, it can be deduced that our HRL-CPG can be a candidate DRL method that enables dynamical systems such as real or realistic robots to adapt to a variety of environments within a moderate physical time.

2.1 Introduction

Autonomous mobile robots are anticipated to play a crucial role in addressing labor shortages in the near-future society, particularly in factories specializing in high-mix, low-volume production and in personal logistics. This issue is expected to be particularly acute in developed countries. These robots must possess navigation capabilities not only in structured environments such as factories and flat roads but also in unstructured environments common in daily life, including slopes, steps, and gravel roads. In this context, quadruped robots are gaining significant attention for their potential to transport heavy loads while overcoming slopes and steps [Raibert et al., 2008, Hutter et al., 2016, Hwangbo et al., 2019, Hutter et al., 2017]. Due to the environment-

dependent nature of efficient gait patterns, there is a need for mechanisms that can autonomously and efficiently optimize gait patterns to adapt to new environments.

Reinforcement Learning (RL) is a machine learning framework that enables robots to autonomously learn control policies through trial and error [Sutton and Barto, 1998]. It differs from Model Predictive Control (MPC), which is an optimization-based control approach using an accurate target model [Wieber, 2014, Bledt and Kim, 2020]. With advancements in deep learning, Deep Reinforcement Learning (DRL) has been successfully applied to various tasks [Kalashnikov et al., 2018, Zhou et al., 2021, Jin et al., 2018]. The basic principle of DRL is straightforward: represent the control policies and their value functions using deep neural networks [Mnih et al., 2015]. This approach can also be applied to model-based RL, where the model is explicitly approximated, offering methods similar to MPC [Moerland et al., 2020, Kaiser et al., 2019, Ha and Schmidhuber, 2018, Hafner et al., 2019]. However, learning policies and value functions for combinations of high degrees of freedom in state and control requires substantial trial and error, which is why RL and DRL are primarily used for tasks executed on computers, such as in robot simulators [Brockman et al., 2016, Engstrom et al., 2020].

Unlike computer simulations, conducting a large number of operations with an actual robot in each environment is often impractical. This is due to the necessity for human intervention, such as resetting the robot to its initial state and managing any breakdowns [Levine et al., 2018]. To address this issue and improve the efficiency of robotic operations, apart from advancements in RL algorithms [Schulman et al., 2017, 2015, Haarnoja et al., 2018, Abdolmaleki et al., 2018], two main strategies have emerged.

The first approach, known as *sim2real*, involves training a robot in a simulated environment and then transferring the learned behaviors to a real-world robot [Peng et al., 2017, James et al., 2018, Tan et al., 2018, Zhao et al., 2020]. This method takes advantage of the flexibility and safety of simulation to improve the real robot's performance. However, this technique is challenging because it requires highly accurate simulators that can mimic complex real-world phenomena such as contact interactions and material deformations. Effective calibration techniques [Janner et al., 2019, Chua et al., 2018, Chow et al., 2021, Clavera et al., 2018] are essential to bridge the gap between the dynamics and kinematics of simulated models and real-world conditions, and domain randomization techniques are widely employed to provide robots with diverse experiences [Tobin et al., 2017, Chebotar et al., 2019]. These issues must be addressed to ensure that policies trained in a simulator can perform reliably when deployed on physical robots.

An alternative approach is to incorporate prior knowledge into the control policy

function, effectively narrowing down the search space [Uchendu et al., 2022, Kober and Peters, 2010, Nakamura et al., 2007, Schaal et al., 2005, Harada et al., 2009]. One typical method for achieving this is through the use of central pattern generator (CPG)-based methods, which are inspired by the locomotion mechanisms observed in animals [Whelan, 1996, Matsuoka, 1985]. CPG-based methods are distinguished by their ability to achieve robust locomotion that effectively adapts to environmental changes and disturbances, utilizing the characteristic entrainment provided by CPGs [Fukuoka and Kimura, 2006, Ijspeert et al., 2007, Taga et al., 1991].

For instance, Nakamura et al. [Nakamura et al., 2007] illustrated how the integration of RL with CPGs can produce a well-suited gait for simplified biped robot simulations, adaptable to diverse terrains. The pursuit of strategies for developing multiple gaits continues to progress, as evidenced by recent works [Wang et al., 2021, Deshpande et al., 2023, Campanaro et al., 2021]. A notable contribution in this field is by Bellegarda et al. [Bellegarda and Ijspeert, 2022], who utilized an innovative domain randomization technique. Through extensive trial and error in simulated environments, they achieved robust walking capabilities for robots navigating unstructured spaces populated with obstacles. These advances mark significant steps towards more adaptable and resilient robotic locomotion.

This study aims to develop a mechanism enabling a quadruped robot to adapt to new environments with a minimal number of trials and errors in the target environment. To achieve this, it is essential to easily design initial gait parameters and employ sample-efficient algorithms that adjust those parameters to the environment. A CPG controller is characterized by three key factors: the timing of leg movements (phase), movement patterns (waveform), and the speed of joint motions (frequency). These factors can be empirically derived, for example, by referencing model animals [Hoyt and Taylor, 1981, Taga et al., 1991]. However, when implementing a CPG controller with neural oscillators, determining initial parameters that approximately reproduce the given features is not easy, due to the unknown influence of each parameter on the output, and the expected performance relies heavily on expert intuition [Matsuoka, 2011]. Therefore, we adopt phase oscillators, whose parameters directly correspond to the three properties above. This study proposes a framework for tuning these interdependent parameters through RL.

We evaluate the performance of the proposed method using a quadruped robot simulator on a physics engine Bullet [Coumans and Bai, 2016–2019]. The aim of the control task evolves walking without deviating from an aisle of a certain width. The performance was tested on five types of floor conditions: even surface, up and down slopes with different friction values. The experimental results showed that suitable

gait patterns were obtained within 5.0×10^5 control steps (corresponding to walking 3.0×10^4 steps and 1.0×10^4 seconds for 1,500 trials) on average.

2.2 Backgrounds

In the spinal cord in vertebrates and invertebrates, there is a hypothetical neural circuit referred to as the central pattern generator (CPG). Referring to this mechanism, studies have been conducted on rhythmic movements such as locomotion and swimming [Taga et al., 1991, Whelan, 1996, Ijspeert et al., 2007]. The use of CPGs is advantageous even for artificial systems because the induced entrainment to low-dimensional limit-cycles would make the motions robust against the perturbations. For example, even when an external perturbation (e.g., misalignment of landing timing) is applied, the motion immediately recovers to the periodic one thanks to the interactions between the oscillators.

Figure 2.1 (upper left) shows the block diagram of the control using a CPG. In this study, the robot is assumed to change its state according to the dynamics

$$\frac{d\mathbf{s}_r}{dt} = f_r(\mathbf{s}_r, \mathbf{a}_r), \quad (2.1)$$

where $\mathbf{s}_r$ and $\mathbf{a}_r$ denote a state and control signal of the robot, respectively. $\mathbf{s}_r$ consists of physical quantities such as posture and position. In this study, sensor observations are used instead of state variables, and for simplicity, partial observability is not considered.

In CPG-based control, control signals to the robot, $\mathbf{a}_r$ are output from the CPG controller. A CPG is a dynamic system that changes its state $\mathbf{s}_p$ according to the following equation

$$\frac{d\mathbf{s}_p}{dt} = f_p(\mathbf{s}_p, \mathbf{a}_p). \quad (2.2)$$

Here, $\mathbf{a}_p$ is the feedback signal to adjust the phase of the oscillator in the CPG. The controller outputs the control signal to the robot $\mathbf{a}_r$ as

$$\mathbf{a}_r = g_p(\mathbf{s}_p).$$

In this study, the output of the CPG represents the target posture of the robot. That is, one oscillator corresponds to one joint, and the output of the oscillator is input to the robot as the target joint angle. After the robot receives the target angle, a PID control with a higher control frequency is employed to drive the low-level actuator of each joint.

Figure 2.1 (bottom) shows the temporal change of the state of CPG. The horizontal

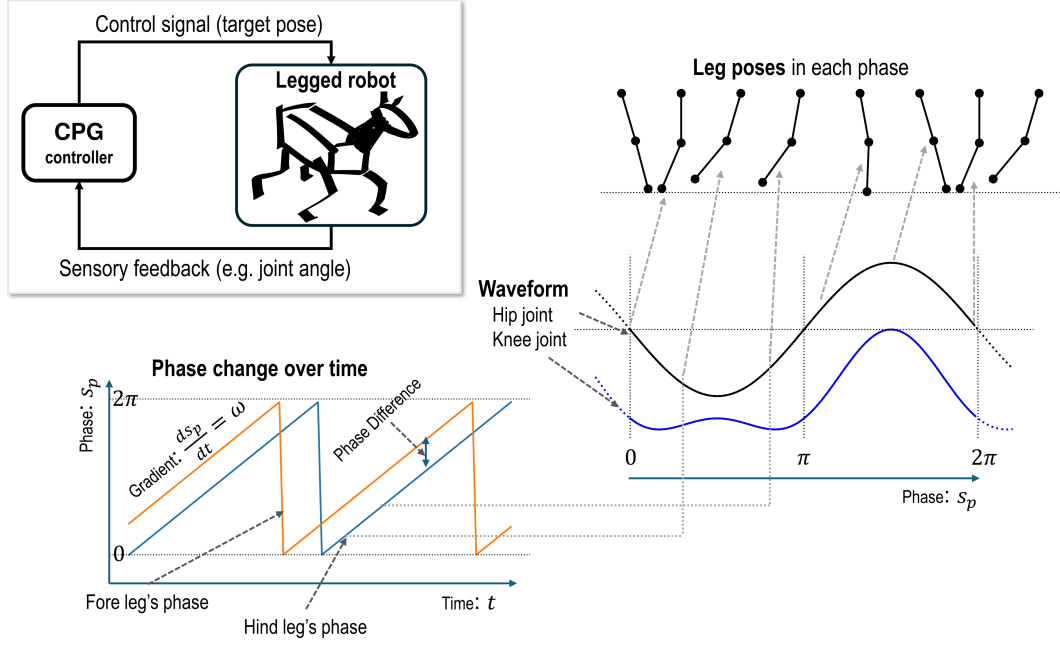


Figure 2.1: Schematic diagram of the applied phase oscillator network (PON).

and vertical axes are the time and phase, respectively. The phase is converted into the target joint angle of each joint by a waveform modulation function. The upper right figure indicates the target pose of a leg when the phase is given.

The gait of the robot changes depending on the following factors. 1) the phase relationship between joints, 2) the length of the gait cycle, and 3) the waveform. By adjusting these characteristics, the robot can modify its gait to adapt to its environment.

2.2.1 Oscillator Model

In this study, we employ a phase oscillator model presented by Kuramoto [Kuramoto, 1975]; because the Kuramoto model is simpler than other oscillator models, such as that presented by Matsuoka [Matsuoka, 1987]. Although both the Kuramoto and Matsuoka models have been used to generate the locomotive motions of animal-like architects [Sajjad and Youngjin, 2016, Owaki et al., 2017, Maeda et al., 2021, Taga et al., 1991, Nakamura et al., 2007], the parameters that determine the three aforementioned characteristics are clear in the former model. With the usage of the former model, our HRL-CPG, which will be introduced in the next section, becomes simpler.

In a phase oscillator network (PON), there are multiple oscillators (phasers), each of which has a characteristic frequency, and their phases are affected by the interaction expressed as a sinusoidal function between the oscillators. Let $s_p^{(i)}$ be the phase of the i -th oscillator, $i = 1, \dots, N$, where N is the number of oscillators, then their network

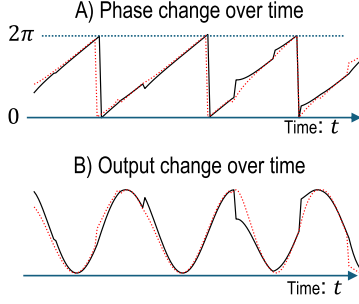


Figure 2.2: Behavior of the PON. The dashed black line shows example behavior of one unit (phaser) in PON; the dotted line shows another. (A) State. (B) Output.

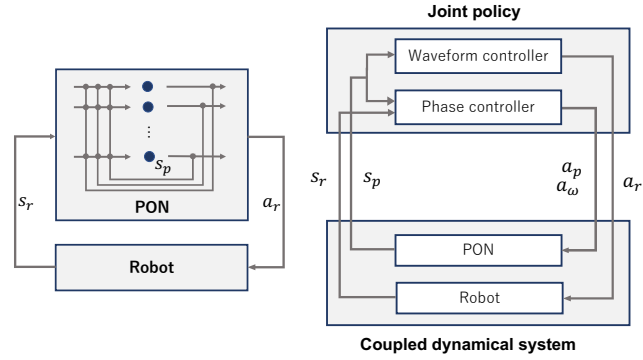


Figure 2.3: Schematic diagrams of the CPG-based control. (a) Control architecture for the robot using a phase oscillator network (PON). (b) Control architecture for the coupled dynamical system using a feed-forward neural network.

PON is expressed by

$$\frac{ds_p^{(i)}}{dt} = \omega^{(i)} + \frac{K}{N} \sum_{j=1}^N \sin(s_p^{(j)} - s_p^{(i)}) + a_p^{(i)}, \quad (2.3)$$

for $i = 1, \dots, N$, where $\omega^{(i)}$ corresponds to the characteristic frequency of the i -th oscillator, and K is the coupling strength. $a_p^{(i)}$ denotes the external input to the i -th oscillator that coordinates the state of the oscillator with the external system.

Figure 2.2 shows the behavior of a PON with two oscillators. (A) shows the temporal change of the states of the oscillators, i.e., phases $s_p^{(i)}$. When an intermittent impulsive external input is applied, a phase shift occurs between the two oscillators. However, they then behave as if the phase difference converges to 0 due to the entrainment. (B) shows the temporal change of the output. The output of the oscillator is defined by a periodic function with the internal state as input, for which a trigonometric function is used. In this example, the waveform function is defined as $\mu^{(i)} = \sin(s_p^{(i)})$.

2.2.2 Reinforcement Learning

RL is a machine learning framework that allows a learning agent to make optimal decisions. The agent takes an action at the observed state, then observes a new state and a consequent reward. The objective of RL is to identify the optimal decision-making agent, i.e., the optimal policy, that maximizes the expectation of the reward summation (return) at any state.

Although the robot and CPG are treated as continuous-time differential dynamical

systems i.e., eqs. (2.1) and (2.2), they are approximated as discrete-time difference dynamical systems, for the sake of simplicity. That is, the state transition is defined by the following probability:

$$p(\mathbf{s}'|\mathbf{s}, \mathbf{a}),$$

where $\mathbf{s}'$, $\mathbf{s}$, and $\mathbf{a}$ are the destination state, the current state and the action taken, respectively.

The agent starts from an initial state $\mathbf{s}_0 \sim p_0(\mathbf{s}_0)$. At each state $\mathbf{s}_t$ at a discretized time t , it takes an action $\mathbf{a}_t$ sampled from a probabilistic policy $\pi(\mathbf{a}_t | \mathbf{s}_t)$, and receives a reward $r(\mathbf{s}_t, \mathbf{a}_t)$. Then, the system transitions to a new state $\mathbf{s}_{t+1}$ according to its dynamics $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$.

A trajectory of states and actions $\tau := (\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, \dots)$ is thus produced by its probability density

$$\rho_\pi(\tau) = p_0(\mathbf{s}_0) \prod_t \pi_t(\mathbf{a}_t | \mathbf{s}_t) p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t).$$

While the standard RL often optimizes the expected return $\mathbb{E}_{\tau \sim \rho_\pi} [\sum_t r(\mathbf{s}_t, \mathbf{a}_t)]$, in this study, we consider an entropy-based regularized version, such as the one used in the soft Q-learning [Haarnoja et al., 2017] and the soft actor-critic (SAC) [Haarnoja et al., 2018] algorithms:

$$J(\pi) := \mathbb{E}_{\tau \sim \rho_\pi} \left[\sum_{t=0}^{\infty} \gamma^t \{r(\mathbf{s}_t, \mathbf{a}_t) - \alpha \log \pi(\mathbf{a}_t | \mathbf{s}_t)\} \right], \quad (2.4)$$

where $0 \leq \gamma < 1$ is a discount factor.

In eq. (2.4), the first term is the expected return and the second one is the entropy which encourages the agent to try exploratory behaviors that have a relatively low chance according to the policy without the entropy term. α is a temperature hyperparameter that controls the relative importance of the exploration against the exploitation in terms of return maximization. In the limit of $\alpha \rightarrow 0$, the above objective is reduced to the conventional RL objective. The details of the RL algorithm used in this study, Soft Actor-Critic (SAC) algorithm [Haarnoja et al., 2017], is explained in Appendix A.1.

One of the important challenges in applying RL to robot control problems is sample efficiency. Especially in the early stages of learning, collisions and falls are expected to occur frequently. In such cases, human intervention and even repair would be required. Bellow, we propose an efficient learning method by constraining the action space by prior knowledge and discuss its effectiveness.

2.3 Hierarchical Reinforcement Learning for a CPG Controller

To allow a quadruped robot to produce stable locomotions, we propose a hierarchical learning framework of the controller, inspired by the periodic motion generating mechanism in animals. We refer to the framework as the hierarchical RL with central pattern generator (HRL-CPG), because it is a hierarchical RL for the coupled dynamical system which involves a PON, a type of CPG.

By naively introducing a CPG network to the RL-based controller as in Figure 2.3(a), the policy becomes a non-stationary stochastic policy:

$$\pi_r(\mathbf{a}_r|\mathbf{s}_r) \sim p(\mathbf{a}_r|\mathbf{s}_p),$$

since it depends on the internal state of the policy, i.e., the state of the CPG network $\mathbf{s}_p$. Note that this state $\mathbf{s}_p$ depends on the history of state of robot $\mathbf{s}_r$, because of the CPG dependence on the sensory feedback from the robot. Thus, the Markov property, one of the assumptions normally used in RL, is no longer satisfied.

To address this non-Markovian character, the PON is divided into two parts, i.e., the internal state and the input. The former is treated as a part of the control target along with the robot and constitutes a coupled dynamical system. The RL is thus used to optimize the behavior of the coupled system that is the combination of the physical system to be controlled and the CPG network, as shown in Figure 2.3(b).

The latter becomes the external input of the oscillators $\mathbf{a}_p$, the third term of the right-hand side of eq. (2.3). To coordinate the robot state with the state of the CPG controller and between the oscillators inside the CPG, both the sensory feedback from the robot $\mathbf{s}_r$ and the PON state $\mathbf{s}_p$ are used to determine $\mathbf{a}_p$. In other words, the coupled dynamical system defined by the state transition probability

$$p(\mathbf{s}'_r, \mathbf{s}'_p | \mathbf{s}_r, \mathbf{s}_p, \mathbf{a}_r, \mathbf{a}_p, a_\omega) \quad (2.5)$$

is controlled by the joint policy

$$\pi(\mathbf{a}_r, \mathbf{a}_p, a_\omega | \mathbf{s}_r, \mathbf{s}_p),$$

where a_ω is a control signal to determine the length of the gait cycle, i.e., the first term of the right-hand side of eq. (2.3). Note that, the state and action of the coupled dynamical system are defined as $\mathbf{s} = [\mathbf{s}_r, \mathbf{s}_p]$ and $\mathbf{a} = [\mathbf{a}_r, \mathbf{a}_p, a_\omega]$, respectively. The above state transition probability and joint policy can be conceptually rewritten as

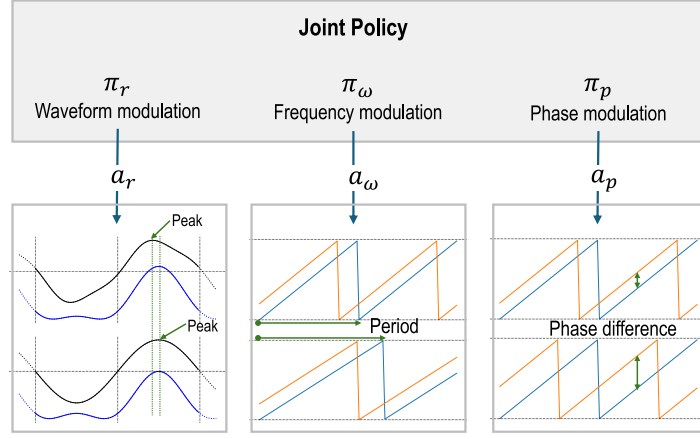


Figure 2.4: Joint policy

$p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ and $\pi(\mathbf{a}|\mathbf{s})$.

2.3.1 Control Signal

$\mathbf{a}_r$, $\mathbf{a}_p$, and a_ω are the modulation signals of the waveform, the phase control signal to the PON, and the base frequency modulation signal, respectively. While each element of $\mathbf{a}_p$, i.e., $a_p^{(i)}$, adjusts the phase of the i -th oscillator in an individual manner, a_ω adjusts the time length of the cycle for all the oscillators. Both of which determine the right-hand side of eq. (2.3); $\mathbf{a}_r$ determines the waveform.

Phase Control

$\mathbf{a}_p$ is the phase control signal to the PON, the third term of the right-hand side of eq. (2.3), and adjusts the phase relationship among oscillators in PON. The phase control policy for the i -th joint is implemented as a neural network:

$$\pi_p(a_p^{(i)}|\mathbf{s}; \phi_p) := \mathcal{N}(\mu_p^{(i)}, \sigma_p^{(i)}), \quad (2.6)$$

where $\mu_p^{(i)}$ is the mean value of the i -th component of output $\mu_p(\mathbf{s}; \phi_{p,\mu}) : \mathbf{s} \rightarrow \mu_p \in \mathcal{R}^N$ calculated by a neural network with the parameter $\phi_{p,\mu}$. $\sigma_p(\mathbf{s}; \phi_{p,\sigma}) : \mathbf{s} \rightarrow \sigma_p \in (0, \infty)^N$ is the variance vector where $\phi_{p,\sigma}$ is the parameter to produce the variance.

Base Frequency Modulation

We assume that the characteristic frequency of each oscillator $\omega^{(i)}$, i.e., the first term of the right-hand side of eq. (2.3), is common for all the oscillators, but is modulated as

$$\omega^{(i)}(t) := \omega_0 + \omega_1 a_\omega(t) \quad (i = 1, \dots, N), \quad (2.7)$$

where ω_0 is the pre-determined basic frequency and $\omega_1 > 0$ is a scale factor for the frequency modulation.

$a_\omega(t)$ is the base frequency modulation signal that depends on the state $\mathbf{s}$ at time t , and is determined by the following procedure. First, we sample a $\tilde{a}_\omega \in \mathcal{R}$ from the standard normal distribution $\mathcal{N}(\mu_\omega, \sigma_\omega)$, which is defined in the same way as in eq. (2.6), based on the neural network’s outputs, μ_ω and σ_ω , with the parameter ϕ_ω . Then we apply the hyperbolic tangent function to $\tilde{a}_\omega$ so that it takes a value between -1 and 1:

$$a_\omega := \tanh(\tilde{a}_\omega), \quad \tilde{a}_\omega \sim \mathcal{N}(\mu_\omega, \sigma_\omega).$$

This implementation is similar to that in [Haarnoja et al., 2018] and the distribution of the random variable a_ω determined by this process is denoted as $\pi_\omega(a_\omega|\mathbf{s}; \phi_\omega)$.

Waveform Modulation

Considering the waveform should have an oscillatory nature, the waveform of the i -th joint is defined as:

$$\mathcal{W}^{(i)}(s_p^{(i)}) = \phi_r^{(i,1)} + \sum_{j=1}^K \left(\phi_r^{(i,2j)} \sin(2\pi j s_p^{(i)} + \delta^{(i)}) + \phi_r^{(i,2j+1)} \cos(2\pi j s_p^{(i)} + \delta^{(i)}) \right)$$

where $\phi_r^{(\cdot)}$ is the parameter adjusted by RL. $\delta^{(i)}$ is a constant set to emit a gait in which the two diagonal legs are swung out at the same timing; it was set to 0 for the left foreleg and right hind leg, and π for the right foreleg and left hind leg.

The waveform modulation policy is given by

$$\pi_r(a_r|\mathbf{s}_p; \phi_r) := \mathcal{N}(\mu_r(\mathbf{s}_p), \sigma_r(\mathbf{s}_p)),$$

where μ_r and σ_r are the mean and variance, respectively. The i -th element of mean and variance are calculated as

$$\mu_r^{(i)}(\mathbf{s}_p) := \mathcal{W}^{(i)}(s_p^{(i)})$$

and

$$\sigma_r^{(i)}(\mathbf{s}_p) := e^{\phi_r^{(i,0)}}, \tag{2.8}$$

respectively.

2.3.2 Joint Policy

The joint policy is defined as the product of the above three policies:

$$\pi(\mathbf{a}|\mathbf{s}) := \pi_r(\mathbf{a}_r|\mathbf{s}_p; \phi_r)\pi_p(\mathbf{a}_p|\mathbf{s}; \phi_p)\pi_\omega(a_\omega|\mathbf{s}; \phi_\omega).$$

HRL-CPG optimizes the joint policy for the coupled dynamical system, according to a single objective function, eq. (2.4). This optimization problem can be seen approximately as an MDP.

Each policy has different temporal and structural impacts on different scales. For example, base frequency modulation changes the overall speed of the gait, while wave-form modulation changes the shape of the entire cycle. The proposed method conducts simultaneous optimization of multiple policies at different scales.

2.3.3 SAC Algorithm for HRL-CPG

Here, we describe an SAC-based optimization for tuning the parameters in our HRL-CPG. There are two critic neural networks, both with the same architecture but different parameters: one network, $Q_\theta(\mathbf{s}, \mathbf{a})$, with parameter θ , is used for approximating the soft Q-function, while the other network uses an exponential moving average of θ (denoted as $\bar{\theta}$) to approximate the soft V-function. The network Q_θ is trained to minimize the following loss functions:

$$J_Q(\theta) := \mathbb{E}_{(\mathbf{s}, \mathbf{a}, \mathbf{s}') \sim \mathcal{D}} \left[\left(Q_\theta(\mathbf{s}, \mathbf{a}) - \left(r(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma \hat{V}(\mathbf{s}') \right) \right)^2 \right]. \quad (2.9)$$

The soft V-function here is computed as

$$\hat{V}(\mathbf{s}) := \mathbb{E}_{\mathbf{a} \sim \pi_\phi(\cdot|\mathbf{s})} [Q_{\bar{\theta}}(\mathbf{s}, \mathbf{a}) + H(\mathbf{s}, \mathbf{a})],$$

where

$$H(\mathbf{s}, \mathbf{a}) := -\alpha_p \log \pi_p(\mathbf{a}_p | \mathbf{s}_p, \mathbf{s}_r) - \alpha_\omega \log \pi_\omega(a_\omega | \mathbf{s}_p, \mathbf{s}_r) - \alpha_r \log \pi_r(\mathbf{a}_r | \mathbf{s}_p). \quad (2.10)$$

To follow the description in Section 2.3.2, the actor is a set of three policy networks, whose parameters are given by $\phi := (\phi_r, \phi_p, \phi_\omega)$. They are trained to decrease the loss function

$$J_\pi(\phi) := - \mathbb{E}_{\mathbf{s} \sim \mathcal{D}, \mathbf{a} \sim \pi_\phi(\cdot|\mathbf{s})} \left[Q_\theta(\mathbf{s}, \mathbf{a}) + H(\mathbf{s}, \mathbf{a}) \right]. \quad (2.11)$$

In our implementation, we set different pre-defined parameter values for α_p and α_r .

Their settings are described in Section 2.4.2.

The entire algorithm for optimizing the joint policy is shown in Algorithm 1. Note that $\mathbf{s}_t$ stands for the state of the coupled dynamical system, eq. (2.5), at the environment step t . As a detailed implementation, we followed that of the original SAC algorithm [Haarnoja et al., 2018], such as minibatches and the double Q-network method [Van Hasselt et al., 2016]. This algorithm is a type of coordinate optimization similar to the expectation-maximization (EM) algorithm [Dempster et al., 1977], well-established in the field of machine learning. That is, we perform stochastic gradient descent steps for the critic (lowering eq. (2.9)) and the actor (lowering eq. (2.11)) in a step-by-step manner and iteratively.

Algorithm 1 HRL-CPG Algorithm

Input: $\theta, \phi, \lambda_Q, \lambda_{\pi_p}, \lambda_{\pi_{\mu_r}}, \lambda_{\pi_{\sigma_r}}, \omega_0, \omega_1$
 $\bar{\theta} \leftarrow \theta$: Initialize target network weights
 $\mathcal{D} \leftarrow \emptyset$: Initialize an empty buffer
for each iteration **do**
 for each environment step **do**
 $\mathbf{a}_t \sim \pi_\phi(\cdot|\mathbf{s}_t)$: Sample action from the joint policy
 $\mathbf{s}_{t+1} \sim p(\cdot|\mathbf{s}_t, \mathbf{a}_t)$: Step forward the coupled dynamical system (2.5)
 $\mathcal{D} \leftarrow \mathcal{D} \cup (\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})$: Store the samples into the buffer
 for each gradient step **do**
 $\theta \leftarrow \theta - \lambda_Q \nabla_\theta J_Q(\theta)$
 $\phi_p \leftarrow \phi_p - \lambda_{\pi_p} \nabla_{\phi_p} J_\pi(\phi)$
 $\phi_\omega \leftarrow \phi_\omega - \lambda_{\pi_p} \nabla_{\phi_\omega} J_\pi(\phi)$
 $\phi_{\mu_r} \leftarrow \phi_{\mu_r} - \lambda_{\pi_{\mu_r}} \nabla_{\phi_{\mu_r}} J_\pi(\phi)$
 $\phi_{\sigma_r} \leftarrow \phi_{\sigma_r} - \lambda_{\pi_{\sigma_r}} \nabla_{\phi_{\sigma_r}} J_\pi(\phi)$
 $\bar{\theta} \leftarrow \tau\theta + (1 - \tau)\bar{\theta}$: Update target network weights
Output: ϕ

2.4 Experiments and Results

In this section, we examine the applicability of our HRL-CPG framework in generating locomotion of a quadruped robot in a variety of environments.

We conducted experiments with consistent hyperparameters for the five environments, and the proposed method exhibited high performance in four out of the five environments, which suggested the stability of our RL method.

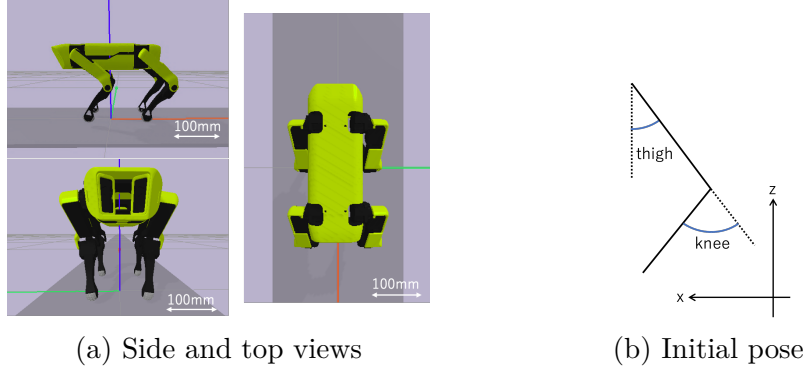


Figure 2.5: Quadruped robot REX (a) Side and top views of the quadruped robot with its scales. (b) The initial pose which is depicted in terms of the left foreleg.

2.4.1 Task Description

We used Bullet Physics SDK for physics simulation and Rex model as a simulator of the quadruped robot, both of which are open-source [Russo, 2021, Coumans and Bai, 2016–2019].

Quadruped Robot

Figure 2.5b shows the quadruped robot we used in the experiments. The length (head to tail) and the width (shoulder to shoulder) of the robot are about 0.35 m and 0.20 m, respectively. The height of the robot with an initial pose shown in the figure is about 0.24 m. Let x -, y -, and z -axes denote the proceeding direction, the left-hand-side, and the up-side of the robot simulator, respectively; they are Cartesian and fixed irrespective to the robot’s posture. Each leg consists of two joints, a thigh and knee, arranged to a sagittal plane that is parallel to the xz -plane.

The robot’s state is represented as a vector of 18 dimensions ($\mathbf{s}_r \in \mathcal{S} \subset \mathcal{R}^{18}$): angles and angular velocities of thighs and knees (16 variables), the rotation angle of the robot’s torso around the z -axis, and the displacement of the torso along the y -axis.

Eight joints (the thigh and knee of each leg: $N = 8$) in total are controlled by our method. Our policy emitted the target joint angles every 20 ms. (i.e., 50 Hz control frequency). The low-level PID controller regulates each joint angle to the target angle specified by the policy, with a control frequency of 1,000 Hz.

Environments

The task for the robot in this experiment is to move forward without deviating from a straight aisle with a width of 0.4 meters. The robot needs to acquire a policy to navigate a passageway which is as narrow as approximately twice its total width.

Table 2.1: Terrain configurations

Environment	1	2	3	4	5
Gradient (degree)	0.0	-3.0	3.0	0.0	0.0
Friction coefficient	1.0	1.0	1.0	0.5	2.0

To demonstrate that learning is possible in a variety of environments, we prepared five kinds of terrain with different gradients and friction coefficients. The configurations of each environment are listed in Table 2.1. When changing the friction coefficient, we altered the value of the lateral friction.

Reward Function

Regardless of the environment, the same reward function:

$$r := k_x \Delta_x - \delta_t (k_y y^2 - k_\varepsilon |\varepsilon|) + p \quad (2.12)$$

was used. Here, Δ_x is the distance traveled from the previous control cycle along with the x -axis, and corresponds to the average speed of each control step. The faster the robot moves forward, the greater the reward. y is the displacement from the center line along the y -axis and the corresponding reward is greater when staying at the center of the aisle. ε is the sum of the products of torque and rotational speed at each joint. k_x , k_y and k_ε are the constants that determine the contributions of the three factors, and were empirically set to 50.0, 1.0, and 0.0025, respectively. δ_t ($=0.02$) is the constant used to take a balance between the instantaneously observed reward and the cumulative rewards per second.

p is a negative reward (penalty, -3) and is applied when the robot tips over or gets off the aisle; when the robot received this, the control episode was terminated. The maximum time length of each episode was set to 20 s (1,000 control steps since the control cycle is 20 ms); even when the robot performed walking for 20 s, the episode was terminated. It was considered as tipping over when the robot's torso tilted more than $\cos^{-1}(0.85)$ rad. Since both sides of the aisle are cliffs, getting off the aisle will result in a fall. The reward function, eq. (2.12), encouraged the robot to walk along the aisle faster and stabler without falling for 20 s.

Initial Pose

Figure 2.5b(b) depicts the initial pose, which was set empirically; the joint angles were set to 0.904 rad for the thigh parts and 1.48 rad for the knee parts. That is,

$s_{r0}^{(i)} \in \{0.904, 1.48\}$ is the initial state of the robot's i -th joint angle. The target angles are determined based on the policy's output and those values as $0.904 + a_r^{(i)}$ for the thighs and $1.48 + a_r^{(i)}$ for the knees.

Initial Policy Parameters

ϕ_p and ϕ_ω , the parameters of neural networks for the phase control policy π_p and the base frequency modulation policy π_ω , respectively, were initialized with Kaiming initialization [He et al., 2015].

The policy parameters of the waveform policy $\pi_r(a_r|s_p)$ were set as:

$$\begin{aligned}\phi_r^{(i,0)} &= 0.01 \quad \forall i \\ \phi_r^{(i,1)} &= 0 \quad \forall i \\ \phi_r^{(i,3)} &= -0.1 \quad \text{if the } i\text{-th oscillator controls a thigh} \\ \phi_r^{(i,3)} &= 0.2 \quad \text{if the } i\text{-th oscillator controls a knee} \\ \phi_r^{(i,j)} &= 0 \quad \text{if } j \text{ is not 1 or 3}\end{aligned}$$

Training and Test Performance

This study aims to assess how quickly the proposed method enables the robot to adapt to individual environments. To ensure a consistent evaluation of this adaptation process, both training and testing were performed in the same environment.

On the other hand, to evaluate whether stable policy can be learned, the behavior is designed to exhibit consistent control during testing, while stochastic behavior is encouraged during training to facilitate exploration. During training, actions are computed through a sampling procedure represented by the joint policy to encourage exploration. The randomness involved in this process is controlled by the parameters σ_p of the phase control policy (eq. (2.6)), σ_ω of the base frequency modulation policy, and σ_r of the waveform modulation policy (eq. (2.8)), which are dependent on the state at each step. During testing, these values are set to zero, and actions are applied to the environment (and the CPGs) using the same procedure.

2.4.2 Main Results

We used consistent hyperparameters. The base modulation frequency was $\omega_0 = 6\pi$ and $\omega_1 = 2\pi$. The temperature parameters, α_r and α_p , were determined at 0.00001 and 0.1, respectively, such to allow the robot to generate good walks in environment 1 (flat surface). Although the relatively large α_p seems to introduce a significant fluctuation to the PON via the phase modulation policy π_p , this is not the case; even if we apply

large noises to the PON units independently, effective change in the oscillator’s state remains small due to compensation of independent perturbations within the whole PON dynamics. Making α_p large thus enables RL on the phase modulation policy to perform effective exploration under such noise compensation. To examine the robustness of the parameters across different environments, we adopted the temperature parameters determined in environment 1 regardless of the target environment.

Figure 2.6(a) shows the learning curves, i.e., the test performance averaged over five training runs. In each test, the robot walks through the same environment where the controller was trained. The vertical axis denotes the accumulated reward for a roll-out episode, and the horizontal axis denotes the learning step. Differences in line color indicate differences in the five environments (Table 2.1).

A single-episode evaluation roll-out was performed every 5,000 learning steps. In each panel, the solid line and shaded area are the mean and 90 and 10 percentiles, respectively.

Figure 2.6(a) indicates that learning in environment 4 was unsuccessful. Further analysis on the progression of learning, detailed in Appendix C, evaluated the failure rate (the frequency of falls during test episodes before the termination step) and the progression distance during successful episodes. In environment 4, the failure rate did not decrease significantly, unlike in other environments, where the failure rate steadily declined and the progression distance increased. However, even in environment 4, learning became feasible with hyperparameter adjustments, resulting in a decrease in the failure rate and an increase in progression distance, similar to the patterns observed in other environments. These findings suggest that under the current reward setting, where significant penalties are applied for frequent “ failure ” cases, the learning process may become unstable. This warrants further investigation.

Finally, we report the number of samples that were needed to achieve suitable gait patterns. On average, in all the environments (except for environment 4), walking was achieved in approximately 3.0×10^4 walking steps, which corresponded to approximately 1.0×10^4 physical seconds (≈ 2.7 hours) and 1.5×10^3 trials (each trial was up to 20 s), implying that our method is applicable to autonomous adaptation by a realistic quadruped robot with a reasonable number of learning steps and/or physical time.

There can be two possible reasons for the unfavorable result in environment 4. First, the initial set of policy parameters was not suitable for the environment. Since the purpose of our study is to develop an RL method which adjusts the policy to adapt to its uniform environment, we implicitly assumed that the optimal policy is not significantly differ from the initial policy (as a kind of prior knowledge). However, the initial policy parameters are empirically determined based on the test runs on the flat

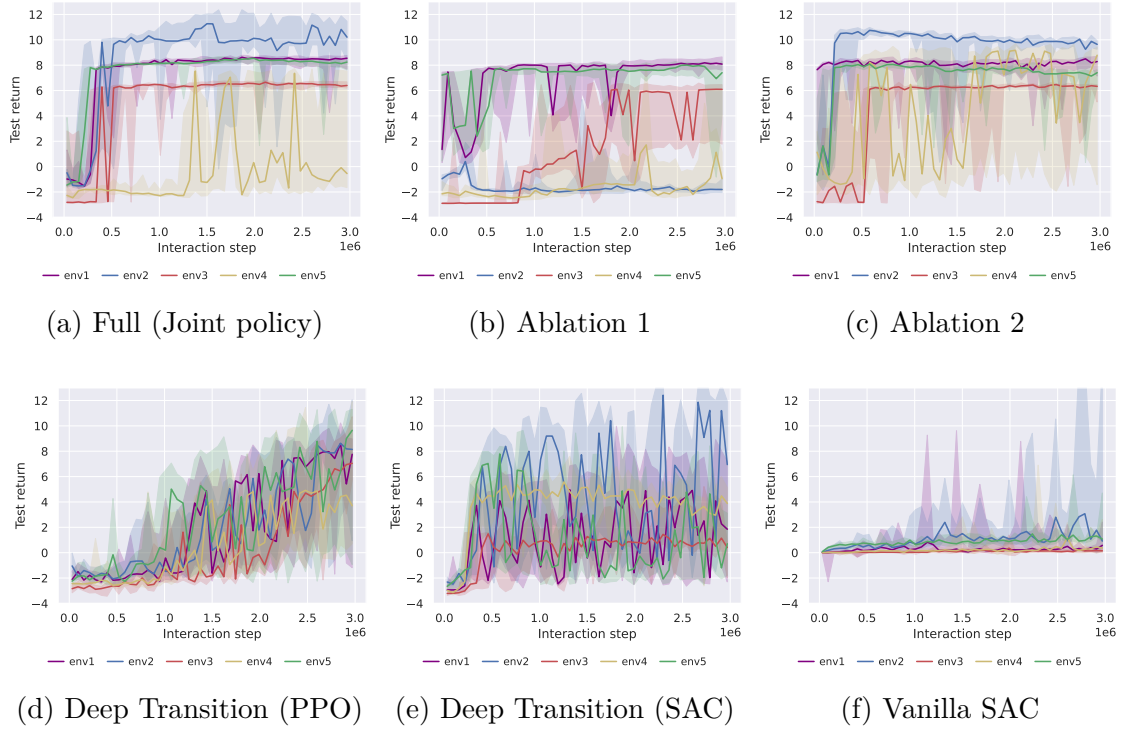


Figure 2.6: Learning curves; the horizontal and vertical axes denote the number of interaction steps with the environment and the test performance (the total return of one roll-out), respectively. The test was performed every 5,000 interaction steps. “Vanilla SAC” stands for a naive SAC where no CPG was used.

and relatively high friction aisle. This case may not have been as it was supposed to be.

Second, the learning in the earlier stage had been trapped in a local optimum due to the biased sampling toward the experiences in which the robot slipped to get off the aisle. This could have happened, especially when the learning rate for the variance of the robot control signal σ_r was relatively high; notably, this learning rate is related to the degree of exploitation, such that highly setting of it would have led to highly exploiting behaviors. In fact, stable walking was achieved as shown in Figure 11 in Appendix D when two (hyper)parameters were adjusted; the learning rate for σ_r was changed from 3.0×10^{-4} to 1.0×10^{-5} and the initial value of σ_r from $e^{0.01}$ to $e^{0.1}$ for all joints. In Appendix D, an additional experiment in environment 4 is shown.

2.4.3 Comparisons

We compared our proposed method with two ablation models and three baseline methods. The differences are summarized in Table 2.2. In Ablation 1, the base frequency modulation policy π_ω is excluded from the joint policy, meaning that ω in eq. (2.7) is always ω_0 . In Ablation 2, the waveform modulation policy π_r is excluded from the learning targets, meaning that π_r is fixed at its initial setting described in Section 2.4.1. The hyperparameters were set to the identical values as those in the proposed method (Full).

As one of the state-of-the-art methods that are mostly related to our method, Deep Transition was adopted [Bellegarda and Ijspeert, 2022, Shafiee et al., 2024], since this method is designed for RL of a CPG controller. According to this method, the sensory feedback from the robot to the CPG controller and the amplitude of the motion (gait pattern) are adjusted. The former and the latter correspond to π_p and π_r in our method, respectively. However, the tunable characteristics of the waveform modulation in our method are wider, i.e., not only the amplitude but also the shape can be tuned. The original Deep Transition used the proximal policy optimization (PPO) for RL, which might be slower than SAC, then we also tested its variation with the SAC-based RL to make the comparison fair.

Additionally, we also tested a naive SAC where no CPG was used. In this case, a neural network where the input and output were the sensory feedback from the robot and the target joint angles, respectively, was trained by SAC.

These ablation models and baseline methods were evaluated following the same training-testing scheme as that for the proposed method. The detail of experiments for the Deep Transition is described in Appendix E.

In the case of Ablation 1 (Fig. 2.6(b)), there has been an overall decline in per-

Table 2.2: Characteristics of comparative methods. “DT” stands for Deep Transition and “Vanilla SAC” stands for a naive SAC where no CPG was used.

Optimize	π_p	π_ω	π_r
Full	Yes	Yes	Yes
Ablation 1	Yes	No	Yes
Ablation 2	Yes	Yes	No
DT (PPO)	Yes	No	Yes [†]
DT (SAC)	Yes	No	Yes [†]
Vanilla SAC	N/A	N/A	N/A

Table 2.3: Overview of performance after learning. Method name is the same as in Table 2.2

Method	Env.1	Env.2	Env.3	Env.4	Env.5
Full	8.32	9.64	6.40	1.39	8.06
Ablation 1	8.01	-1.76	4.11	-0.51	7.16
Ablation 2	7.85	9.24	5.86	5.47	6.81
DT (PPO)	6.26	6.67	5.86	4.08	5.83
DT (SAC)	2.22	5.58	1.09	3.16	0.49
Vanilla SAC	0.71	3.76	0.20	0.67	1.42

formance as well as an increase in the number of cases that do not adapt well. This suggests π_ω (base frequency modulation) is important for the adaptation. Actually, the effect of this policy can be realized by adding a constant to the phase modulation policy π_p , since the both outputs are just added together in eq. (2.3). In other words, they are redundant parameters. However, the result would suggest that having an explicit parameter that controls the average period of gait across all four legs is important for adaptation.

In the case of Ablation 2 (Fig. 2.6(c)), it can be seen that gait patterns suited to each environment were obtained as well as in the proposed method’s case. In environment 4, the average performance was better than that by our method, though the robot still got off the aisle frequently. In the other environments, the averaged performances are slightly worse, and the returns of successful roll-outs were lower than those by the proposed one, especially in environment 2. This suggests that waveform modulation π_r would have committed to obtain better gait patterns thanks to the high degree of freedom.

Deep Transition (PPO) (Fig. 2.6(d)) adapts steadily in all environments, where Deep Transition (SAC) (Fig. 2.6(d)) exhibited better learning. However, both methods failed to achieve stable gaits. The returns of successful roll-outs, especially by Deep Transition (SAC), are high, suggesting good policies could be obtained if sample efficiency is not considered. The improvement in average return has been stable, especially when PPO is used, and this may be useful as a preliminary training in situations like sim2real.

On the contrary to the other methods, the vanilla SAC (without CPG) did not allow the robot to obtain any good gait patterns. This suggests that the use of CPG is crucial for the gait adaptation to the environment.

Table 2.3 shows the average performance after learning. The values denote the

average of returns of test roll-outs during 2.5M steps and 3M steps and over 5 seeds.

2.5 Concluding Remarks

In this study, we developed a data-driven DRL method for optimizing the CPG-involving policies and evaluated its performance in a practical manner with the expectation of its applicability in real robot controls. To this end, we proposed a hierarchical RL framework, called HRL-CPG, which allows a coupled system of the target robot simulator and a CPG network to autonomously adapt to various kinds of terrains to walk through.

In the HRL-CPG, we prepared a hierarchically arranged control policy consisting of a waveform modulation policy, a phase modulation policy, and a base frequency modulation policy, and presented an SAC-based optimization method to tune them. When the HRL-CPG was applied to acquisition of the adaptive gait by a four-legged robot simulator in five kinds of aisles with different gradients and different friction coefficients, we observed that stable gait motions were acquired within a reasonably small number of trials and errors for four out of the five environments.

After tuning one hyperparameter, it could also adapt to the remaining one environment within a relatively small number of trials and errors. These numbers corresponded to 10,000 physical seconds (≈ 2.7 hours). Thus, we conclude that our HRL-CPG can be a candidate for a DRL method that allows real or realistic robots to adapt to a variety of environments within a moderate physical time.

In the proposed method, we suggested a learning approach for the CPG controller under the condition of uniform terrains. However, since uniform terrain cannot be necessarily assumed in real-world applications, extensions would be demanded. For example, there can be various types of non-uniform ground surfaces, such as local unevenness (bumps and potholes) or stairs. In such an environment, incorporating sensor information from a vision-equipped robot into the policy must be crucial and remains a challenge for future work. As a preliminary experiment, we applied our HRL-CPG method to train a controller for the robot’s walking task on surfaces with zebra-patterned bands of varying friction. We provided the robot with the distance between its swung foot and the nearest boundary of a higher-friction zebra band in the forward direction. The robot learned a gait that placed its feet on the boundary of zebra bands, maintaining consistent friction for forward movement (Fig. 8 in Appendix B). This suggests that our method could be applicable to such tasks if appropriate visual features are provided.

The experiments suggest that our method achieves higher sample efficiency in adap-

tation compared to the Deep Transition, one of the state-of-the-art methods. Unlike our proposed method, which learns variations of a single gait, Deep Transition explores various gait patterns, resulting in different adaptive characters. The need for extensive exploration may impact the learning speed. In our proposed method, expanding adaptation to diverse situations using various gaits is a future challenge. Developing a higher-level mechanism to switch between gaits, such as trot, pace, and bounce, based on the situation, would be essential.

We observe that the *OA* regularization appears to help the hierarchical agent adapt more quickly, potentially by discouraging local policies that are overly tuned to a single friction parameter.

Chapter 3

Orthogonal Adversarial Hierarchical Reinforcement Learning with CPGs

3.1 Introduction

In Chapter 2, we investigated a CPG-based hierarchical approach, which showed limited adaptability to low-friction (slippery) environment. Although harnessing control periodicity and policy hierarchy can improve sample efficiency, such approaches may suffer from overfitting and become unstable under challenging conditions [Pinto et al., 2017].

Furthermore, observation noise has been identified as a significant factor that exacerbates these instabilities. Data-driven controllers are often reported to be vulnerable to even small perturbations in inputs [Huang et al., 2017], posing a serious concern in real-world tasks where human co-existence or unpredictable environmental noise is inevitable. Motivated by these challenges, recent studies have applied adversarial learning techniques to reinforcement learning to enhance robustness [Pinto et al., 2017, Huang et al., 2017]. Originally proposed in supervised classification contexts [Goodfellow et al., 2015, Miyato et al., 2018], adversarial learning has been extended to the RL domain as Deep Adversarial RL (DARL), enabling controllers to maintain performance under worst-case scenarios.

In this chapter, we aim to integrate the DARL concept into the HRL-CPG framework to improve robustness against observation noise. Specifically, we adopt the Orthogonal Adversarial (OA) method [Ohashi et al., 2024], which exposes the policy to worst-case observation perturbations while avoiding an undesired bias that conflicts with the original RL objective. Such an OA-based strategy is expected to alleviate

overfitting and enhance stability. As demonstrated later in this chapter, our proposed method can maintain the benefits of the hierarchical design in terms of sample efficiency, while significantly improving performance in low-friction and noisy environments.

3.2 Orthogonal Adversarial Method

In this section, we briefly describe the Orthogonal Adversarial (OA) method proposed by Ohashi et al. [2024]. The core idea of OA is to introduce adversarial noise into the agent’s observation in order to regularize policy optimization, thereby improving robustness to worst-case perturbations. Unlike standard adversarial training, OA applies an orthogonal decomposition to the original policy gradient, ensuring consistent and stable regularization.

Key Steps of the OA Method

We assume a Soft Actor-Critic (SAC) framework [Haarnoja et al., 2017], where a policy π_ϕ is trained to maximize the soft value function estimate

$$\hat{V}(\mathbf{s}) := \mathbb{E}_{\mathbf{a} \sim \pi_\phi(\cdot|\mathbf{s})} [Q_\theta(\mathbf{s}, \mathbf{a}) + H(\mathbf{s}, \mathbf{a})],$$

with Q_θ being the critic and $H(\mathbf{s}, \mathbf{a})$ the entropy term. In OA, we inject a small perturbation η into the state $\mathbf{s}$ such that it reduces the above objective as much as possible:

$$\eta := \arg \min_{\|\eta\|=\epsilon} \mathbb{E}_{\mathbf{a} \sim \pi_\phi(\cdot|\mathbf{s}+\eta)} [Q_\theta(\mathbf{s}, \mathbf{a}) + H(\mathbf{s}, \mathbf{a})].$$

Practically, one uses a first-order approximation [Huang et al., 2017] to compute η by taking the gradient of the expected soft Q-value with regard to the state, normalized and scaled by ϵ .

Once this adversarial noise η is determined, the policy gradient under the noisy state,

$$\delta\phi^{\text{adv}} := \nabla_\phi \mathbb{E}_{\mathbf{a} \sim \pi_\phi(\cdot|\mathbf{s}+\eta)} [Q_\theta(\mathbf{s}, \mathbf{a}) + H(\mathbf{s}, \mathbf{a})],$$

serves as a regularization term that encourages the policy to remain robust to the worst-case perturbation. The OA method further orthogonalizes this adversarial gradient with regard to the original gradient $\delta\phi$:

$$\delta\phi^{\text{OA}} := \delta\phi + \lambda \left(\delta\phi^{\text{adv}} - \frac{\delta\phi \cdot \delta\phi^{\text{adv}}}{\delta\phi \cdot \delta\phi} \delta\phi \right), \quad (3.1)$$

where $\delta\phi$ is the standard SAC policy gradient and λ is a non-negative coefficient. The algorithm with OA method is called *OA-SAC* algorithm. This formulation ensures that the adversarial gradient only contributes the component orthogonal to $\delta\phi$, providing more consistent regularization.

3.3 Application of OA to HRL-CPG

In Chapter 2.3.3, we introduced a hierarchical RL approach called *CPG-SAC*, which exploits Central Pattern Generators (CPGs) in conjunction with Soft Actor-Critic (SAC). Although CPG-SAC showed promising potential in standard conditions, it failed to adapt adequately in low-friction environment (environment 4). One open question is whether CPG-SAC remains robust when the agent’s observations are perturbed by noise. To address these limitations, we now propose *CPG-OA-SAC*, an algorithm that integrates the OA method into CPG-SAC.

3.3.1 CPG-OA-SAC Algorithm Overview

The core difference from OA-SAC lies in how we incorporate adversarial noise into the robot’s state ($\mathbf{s}_r$) during policy optimization. Specifically, at each update, we generate a perturbation η_r by approximating the worst-case noise via the first-order gradient of the soft Q-value (and entropy term) with respect to $\mathbf{s}_r$. Concretely, let

$$\hat{V}(\mathbf{s}) := \mathbb{E}_{\mathbf{a} \sim \pi_\phi(\cdot|\mathbf{s})} [Q_\theta(\mathbf{s}, \mathbf{a}) + H(\mathbf{s}, \mathbf{a})],$$

where $\mathbf{s} = (\mathbf{s}_r, \mathbf{s}_p)$ is the joint state of the robot and the CPG phase variables and $H(\mathbf{s}, \mathbf{a})$ is the entropy term defined as eq. (2.10). We then define

$$\eta_r := -\epsilon \frac{\nabla_{\mathbf{s}'_r} \mathbb{E}_{\mathbf{a} \sim \pi_\phi(\cdot|\mathbf{s}'_r, \mathbf{s}_p)} [Q_\theta(\mathbf{s}, \mathbf{a}) + H(\mathbf{s}, \mathbf{a})] \Big|_{\mathbf{s}'_r = \mathbf{s}_r}}{\left\| \nabla_{\mathbf{s}'_r} \mathbb{E}_{\mathbf{a} \sim \pi_\phi(\cdot|\mathbf{s}'_r, \mathbf{s}_p)} [Q_\theta(\mathbf{s}, \mathbf{a}) + H(\mathbf{s}, \mathbf{a})] \Big|_{\mathbf{s}'_r = \mathbf{s}_r} \right\|_2}, \quad (3.2)$$

which is added to the robot’s state $\mathbf{s}_r$ during the next policy update step. In other words, we do *not* add any adversarial noise to the CPG’s internal phase variables $\mathbf{s}_p$, thereby preserving the inherent oscillatory mechanism of the CPG.

Given this noise vector η_r , the adversarial gradient for CPG-SAC is computed as

$$\delta\phi^{\text{adv}} := \nabla_\phi \mathbb{E}_{\mathbf{a} \sim \pi_\phi(\cdot|\mathbf{s}_r + \eta_r, \mathbf{s}_p)} [Q_\theta(\mathbf{s}, \mathbf{a}) + H(\mathbf{s}, \mathbf{a})].$$

The OA method orthogonalizes this gradient with respect to the CPG-SAC gradi-

Algorithm 2 CPG-OA-SAC Algorithm

Input: $\theta, \phi, \lambda_Q, \lambda_{\pi_p}, \lambda_{\pi_{\mu_r}}, \lambda_{\pi_{\sigma_r}}, \omega_0, \omega_1$
 $\bar{\theta} \leftarrow \theta$: Initialize target network weights
 $\mathcal{D} \leftarrow \emptyset$: Initialize an empty buffer
for each iteration **do**
 for each environment step **do**
 $\mathbf{a}_t \sim \pi_\phi(\cdot | \mathbf{s}_t)$: Sample action from the joint policy
 $\mathbf{s}_{t+1} \sim p(\cdot | \mathbf{s}_t, \mathbf{a}_t)$: Step forward the coupled dynamical system (2.5)
 $\mathcal{D} \leftarrow \mathcal{D} \cup (\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})$: Store the samples into the buffer
 for each gradient step **do**
 Calculate adversarial noise (3.2)
 Calculate the OA gradient (3.1) for ϕ_p and ϕ_ω respectively: $\delta\phi_p^{\text{OA}}$ and $\delta\phi_\omega^{\text{OA}}$
 $\theta \leftarrow \theta - \lambda_Q \nabla_\theta J_Q(\theta)$
 $\phi_p \leftarrow \phi_p - \lambda_{\pi_p} \delta\phi_p^{\text{OA}}$
 $\phi_\omega \leftarrow \phi_\omega - \lambda_{\pi_p} \delta\phi_\omega^{\text{OA}}$
 $\phi_{\mu_r} \leftarrow \phi_{\mu_r} - \lambda_{\pi_{\mu_r}} \nabla_{\phi_{\mu_r}} J_\pi(\phi)$
 $\phi_{\sigma_r} \leftarrow \phi_{\sigma_r} - \lambda_{\pi_{\sigma_r}} \nabla_{\phi_{\sigma_r}} J_\pi(\phi)$
 $\bar{\theta} \leftarrow \tau\theta + (1 - \tau)\bar{\theta}$: Update target network weights
Output: ϕ

ent $\delta\phi := \nabla_\phi J_\pi(\phi)$ via eq. (3.1). Here $J_\pi(\phi)$ is from eq. (2.11). Algorithm 2 outlines the overall procedure for CPG-OA-SAC. The differences from Algorithm 1 are indicated in red text.

In summary,

- **CPG-SAC** updates the policy solely via the original SAC gradient $\delta\phi$.
- **CPG-OA-SAC** modifies the update by adding the orthogonalized adversarial gradient to $\delta\phi$.

3.4 Experiments and Results

We evaluate whether CPG-OA-SAC can overcome the shortcomings of CPG-SAC in two main respects: (1) sample efficiency in low-friction environments, and (2) robustness to observation noise. As baselines for comparison, we also include SAC (without CPG) and OA-SAC (SAC with orthogonal adversarial regularization) in our plots. We set $\epsilon = 12.8$ as noise magnitude and $\lambda = 0.3$ as coefficient for OA regularization during policy optimization.

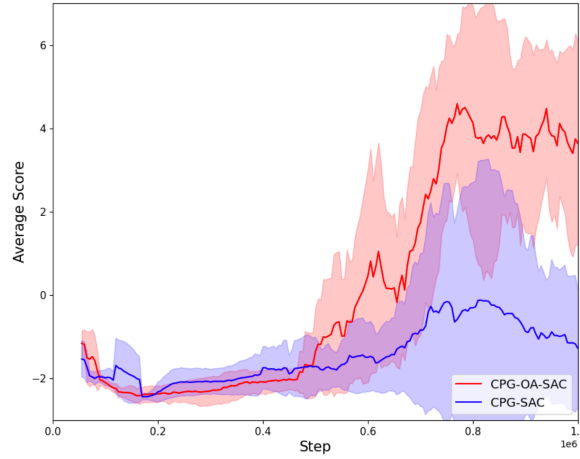


Figure 3.1: Learning curves in a low-friction environment. We compare CPG-SAC and CPG-OA-SAC. The x -axis denotes the number of environment steps (up to 1×10^6), and the y -axis indicates the average return over 5 runs with different seeds. Shaded regions show the standard deviation. Notably, CPG-SAC struggles to gain any reward within these steps, while CPG-OA-SAC improves more rapidly. This suggests that CPG-OA-SAC is far more sample-efficient under low-friction conditions.

3.4.1 Sample Efficiency in Low-Friction Environments

Figure 3.1 presents the learning curves in a low-friction environment, comparing CPG-SAC and CPG-OA-SAC. Because friction is significantly lower than other environments, CPG-SAC fails to make measurable progress within one million environment steps. By contrast, CPG-OA-SAC begins to achieve non-trivial walking (approximately 4 meters in a 20-second episode) after around 80 k environment steps. We observe that the OA regularization appears to help the hierarchical agent adapt more quickly while avoiding overfitting to limited experience.

3.4.2 Robustness to Adversarial and Random Noise

We next assess the learned policies under artificial noise added to the robot’s state. All policies are trained for up to 3×10^6 environment steps and then evaluated under two types of noise: *adversarial noise* (the same formulation as training but with varying magnitude ϵ) and *random Gaussian noise* ($\mathcal{N}(0, \sigma^2)$). We consider the same five environments described in Chapter 2, each having different terrain or friction properties.

Figure 3.2 shows how the four methods fare under adversarial noise of increasing magnitude. CPG-OA-SAC consistently demonstrates higher average returns compared to the baselines, suggesting that the hierarchical CPG structure combined with orthogonal adversarial regularization effectively boosts robustness. However, given the overlapping standard deviation bands across different environments, we refrain from

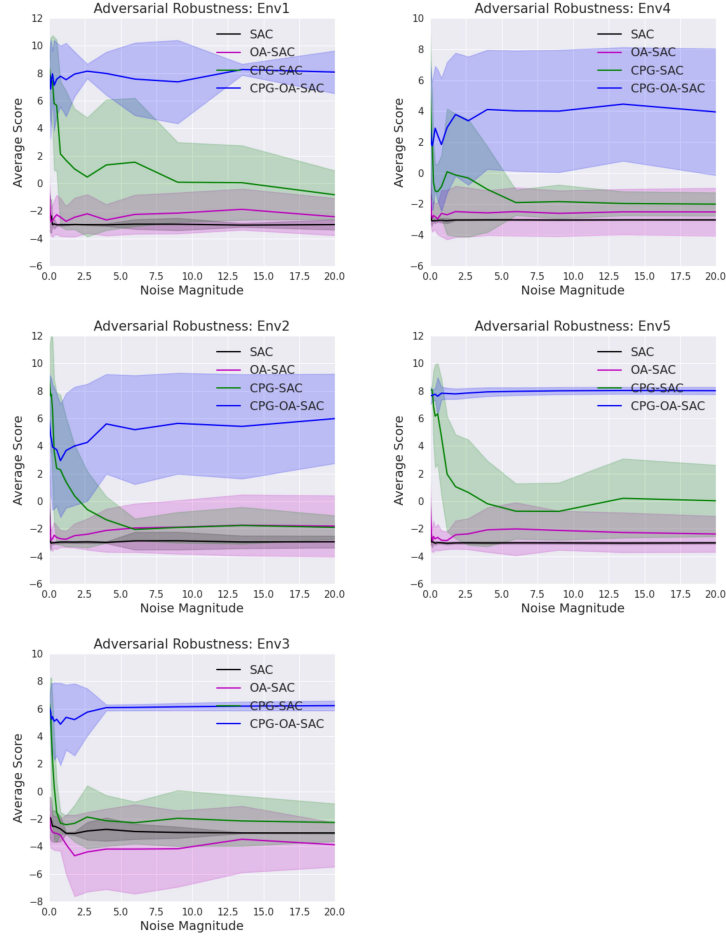


Figure 3.2: Performance under adversarial noise in five environments. We show the average return (lines) and standard deviation (shaded area) across 5 runs, as a function of the adversarial noise strength ϵ . While the results differ slightly across individual environments, we plot the averages here for brevity; the CPG-OA-SAC curve remains higher than all other methods in most of the tested range.

making a strict claim of statistical significance. In practice, this still indicates a clear trend that *CPG-OA-SAC* outperforms the other three methods in challenging adversarial scenarios.

In Figure 3.3, we report results under random Gaussian noise. While the noise tolerance advantage of CPG-OA-SAC is less pronounced here than in the adversarial setting, the average performance remains higher in all tested σ . This outcome implies that even if the noise is not specifically crafted as an adversarial attack, the additional robustness imparted by OA regularization still benefits the CPG-based agent.

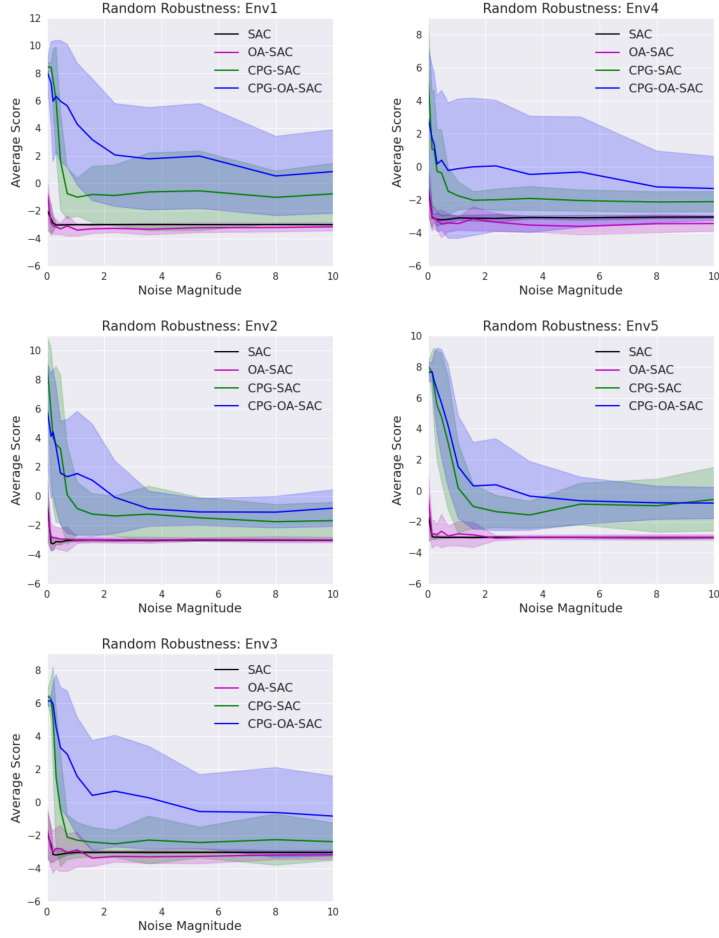


Figure 3.3: Performance under random Gaussian noise in five environments. Each policy is tested by adding $\mathcal{N}(0, \sigma^2)$ noise to the robot’s state, and we plot the average return versus σ . Again, the mean results across all five environments indicate that CPG-OA-SAC achieves higher returns than the other algorithms at most σ values. Due to large variability and overlapping standard deviations, we present these curves primarily as visual evidence that the CPG-OA-SAC approach tends to be more tolerant to random disturbances.

3.5 Conclusion

We introduced CPG-OA-SAC, an approach that combines Central Pattern Generator-based hierarchical RL with the Orthogonal Adversarial (OA) method to address instability and noise vulnerabilities. Building upon CPG-SAC, our method leverages adversarial perturbations to regularize the policy update, thereby improving sample efficiency and robustness across multiple challenging conditions. Experiments in a low-friction locomotion task show that CPG-OA-SAC learns to walk in fewer steps and maintains high performance under adversarial and random noise. By restricting the adversarial noise to the external robot state, we preserve the benefits of hierarchical

structure while mitigating the most harmful perturbations. Overall, our findings show that OA-based adversarial regularization can significantly enhance the robustness of hierarchical reinforcement learning in certain settings. Although other robustification techniques remain to be explored, these results provide a tangible step toward safer and more efficient learning of control policies in real-world robotics.

Chapter 4

Double Horizon Model-based Policy Optimization

4.1 Introduction

Reinforcement learning (RL) is a framework in which an agent improves its policy through repeated interactions with the environment. However, in many applications, it is challenging to continue interacting with the real environment until a sufficient number of samples has been collected. To address this issue, model-based reinforcement learning (MBRL) has emerged as an important strategy: it leverages a learned model to generate virtual samples from past experiences, thereby reducing the need for extensive real-world interactions while improving the policy.

Nonetheless, when the model’s predictive performance is inadequate, especially in the early stages of training, the generated virtual samples become biased, potentially disrupting policy updates. Moreover, data gathered initially under less competent policies often diverges from what the current policy would encounter, resulting in so-called “off-policy” samples. Evaluating and improving the current policy solely on such off-policy data may fail to yield effective information. Additionally, RL commonly involves simultaneous updates to both the policy (actor) and the value function (critic), making it even more challenging to accurately and consistently learn the critic.

Existing MBRL methods have addressed these difficulties through different approaches. For example, the MBPO algorithm [Janner et al., 2019] employs a model simulation called “Branched Rollout” to generate as many states as possible that the current policy is likely to visit, thereby reducing reliance on off-policy data and enabling learning on a state distribution that better reflects the current policy. In this study, we refer to Branched Rollout as Distribution Rollout (DR). On the other hand, algorithms based on Stochastic Value Gradient (SVG) methods [Heess et al., 2015, Amos et al.,

2021, Hafner et al., 2019, Deisenroth and Rasmussen, 2011] use short-horizon model simulations, referred to as Training Rollout (TR), to compute a Model-based Value Expansion (MVE) estimator [Feinberg et al., 2018]. The MVE estimator combines the relatively unbiased sequence of rewards obtained from the short TR trajectory with the value function’s prediction at the final state. Using this estimator as a training target helps correct the value function and also guides policy updates. By effectively utilizing on-policy reward sequences, policy improvement can progress smoothly, even with limited optimization steps. However, until now, DR and TR have been used independently, and no attempt has been made to simultaneously ensure conditions that closely match the current policy on both the state distribution and value estimation fronts.

In this work, we propose a new approach that combines DR and TR to align both the state distribution and the value estimation with the current policy. DR reconstructs a state distribution on the model that better reflects the current policy, thereby reducing the reliance on previously collected off-policy data. Meanwhile, TR leverages the reward sequence to compute the MVE estimator, enhancing the accuracy of the value estimate. However, increasing the TR horizon excessively leads to instability in the gradient computation (backpropagation through time) and risks gradient explosion [Parmas et al., 2018]. Our method starts TR from already on-policy-like states provided by DR, allowing us to keep the TR horizon short while still achieving adequate value estimation accuracy and effective policy improvement. This approach stabilizes gradient computation and, as a result, enables more sample-efficient and stable policy learning.

To illustrate this effect, we first conducted experiments on a small-scale problem (Tabular MDP) as shown in Figure 4.1. In these experiments, we varied the number of episodes in which random actions were executed throughout, as well as the lengths of DR and TR, and then compared the resulting value estimation errors. The results show that, regardless of whether many random actions were taken initially, using sufficiently long DR and TR together significantly reduced the estimation error. Conversely, when many random actions were taken and DR was insufficient (i.e., the state distribution was not adequately shaped to match the current policy), lengthening TR alone did not decrease the error and sometimes even worsened it. This suggests that performing long TR from off-policy states makes it difficult to evaluate the policy accurately, highlighting the importance of DR in approximating an on-policy state distribution.

Based on these observations, we propose a Deep MBRL algorithm called Double Horizon Model-Based Policy Optimization (DHMBPO) that simultaneously employs both rollouts. DR simulates from states sampled out of the replay buffer, generating

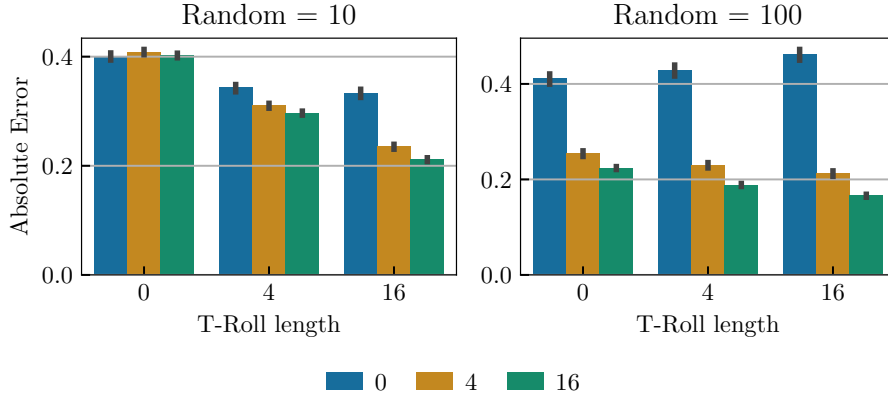


Figure 4.1: Comparison of value estimation errors for different lengths of Distribution Rollout and Training Rollout. The vertical axis shows the absolute difference between the estimated objective function and the true objective function. “Random” indicates the number of initial episodes with random actions, the legend represents the length of the Distribution Rollout, and “T-Roll length” corresponds to the length of the Training Rollout.

samples closer to the state distribution induced by the current policy. This reduces dependence on off-policy data and allows learning under conditions that align better with the current policy. TR, starting from DR-generated samples, performs a short-horizon rollout and computes the MVE estimator from its reward sequence and the critic’s output. By using the MVE estimator, DHMBPO achieves high sample efficiency with a lower Update-To-Data (UTD) ratio, enabling effective policy optimization with fewer updates. A lower UTD ratio also reduces the overall computation time, and indeed, we demonstrate that DHMBPO attains about one-sixteenth the execution time compared to an MBPO-based algorithm without MVE estimation (see Section 4.4).

In summary, this study integrates DR and TR, which had previously been used separately, into a single framework that improves the balance between sample efficiency and computational cost, even in the presence of model bias and off-policy data. The following sections provide the necessary background (Section 4.2), detail the proposed method (Section 4.3), and demonstrate DHMBPO’s effectiveness through evaluations on two standard RL benchmark sets (Section 4.4).

4.2 Background

4.2.1 Notation and Problem Setting

Let $M = \{\mathcal{S}, \mathcal{A}, T, r, \gamma, \rho\}$ represent the Markov Decision Process, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, the next states are sampled from the transition probability

$s' \sim p(s'|s, a)$, r is the reward function, the actions are sampled from the stochastic policy $a \sim \pi(a|s)$, and γ is the discount factor. With the value function for policy π :

$$V^\pi(s) := \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s \right] \quad (4.1)$$

and the distribution of states induced by π : $\rho_\pi(s)$, the objective in this work is to find a policy π^* which maximizes

$$J(\pi) := \mathbb{E}_{s \sim \rho_\pi} [V^\pi(s)]. \quad (4.2)$$

under the condition that the transition probability $p(s'|s, a)$ is unknown.

On the other hand, in entropy-regularized RL where $r(s_t, a_t)$ is replaced by $r(s_t, a_t) - \alpha \log \pi(a_t|s_t)$ in eq. (4.1), as a formulation of the Soft Actor-Critic (SAC) algorithm, the soft value function

$$V_\alpha^\pi(s) := \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s \right], \quad (4.3)$$

and its associated soft Q -function

$$Q_\alpha^\pi(s, a) := r(s, a) + \gamma \mathbb{E}_{s' \sim T(\cdot|s, a)} [V_\alpha^\pi(s')]$$

are optimized. Here, we assume that

$$V_\alpha^\pi(s) = \mathbb{E}_{a \sim \pi(\cdot|s)} [Q_\alpha^\pi(s, a) - \alpha \log \pi(a|s)]$$

holds for all $s \in \mathcal{S}$. The temperature parameter α controls the degree of exploration. If $\alpha = 0$, eq. (4.3) is equivalent to eq. (4.1), and hence, in what follows, we do not strictly differentiate between the standard and soft value functions. For simplicity, we write $V^\pi(s)$ for the value function and $Q^\pi(s, a)$ for the state-action value function.

4.2.2 Model-based Actor-Critic Methods

In this study, we consider a deep model-based actor-critic architecture. Namely, we have a policy (actor) $\pi_\theta(a|s)$, a critic $Q_\phi(s, a)$ approximating $Q^\pi(s, a)$, and a pair of learned models $p_\psi(s'|s, a)$ and $r_\psi(s, a)$. The parameters θ , ϕ , and ψ refer to the respective neural network weights.

During critic training, for an input (s, a) , a target signal $\hat{y}$ is computed, and the critic Q_ϕ is updated by minimizing the squared difference between its output and $\hat{y}$. Meanwhile, the actor π_θ is updated to maximize the sample average (with respect to a

given state distribution $\rho(s)$) of the estimated value $V^{\pi_\theta}(s) \approx \mathbb{E}_{a \sim \pi_\theta(\cdot|s)} [Q_\phi(s, a) - \alpha \log \pi_\theta(a|s)]$. Although in principle the critic should be updated until convergence at each policy improvement step, we follow existing works [Janner et al., 2019, Hafner et al., 2019, Hansen et al., 2022, 2024] by performing alternating updates of the critic and actor within policy optimization loop. This alternation is explicitly shown in line 8 of the for loop in Algorithm 3.

Among deep model-based RL algorithms with excellent performance, we can categorize the uses of model rollouts into two primary approaches: (i) the *distribution rollout* (DR), which approximates the expectation with respect to the on-policy state distribution $\rho^\pi(s)$ induced by π_θ , (ii) the *training rollout* (TR), which is used to compute the approximate value function $\hat{V}(s)$ for critic and / or actor updates.

DR was proposed by Janner et al. [2019] (referred to in their work as Branched rollout). It begins by sampling initial states from a replay buffer that stores transitions from the real environment, then performs multi-step rollouts using the learned model. The resulting trajectories are split into 1-step transitions, which are stored in a model buffer and used by standard policy optimization (e.g., the SAC algorithm’s update methods for actor and critic).

TR, on the other hand, is common to many MBRL algorithms [Feinberg et al., 2018, Hafner et al., 2019, Hansen et al., 2022, Amos et al., 2021]. It also starts from states drawn from the replay buffer but is used differently: the generated trajectories from a short-horizon model rollout are utilized for critic training [Feinberg et al., 2018, Hafner et al., 2019, Hansen et al., 2024] or actor updates [Amos et al., 2021, Hafner et al., 2019]. In this study, following Hafner et al. [2019], we use TR samples for both the critic and actor updates.

4.2.3 Model-based Policy Optimization (MBPO)

The MBPO algorithm, proposed by Janner et al. [2019], applies the policy optimization procedure from SAC [Haarnoja et al., 2018] to fictitious samples generated via model rollouts. Here, we call these rollouts *distribution rollouts* (DR). To execute DR, we randomly select initial states from the replay buffer and then simulates D steps using the learned model. The resulting trajectories are stored in a *model buffer*, $\mathcal{D}_m^D$, which is separate from the replay buffer. Here D denotes the DR horizon and indicates that the data distribution in $\mathcal{D}_m^D$ depends on D .

The critic Q_ϕ is updated to minimize:

$$\mathcal{L}_D(\phi) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}_m^D} \left[\left(Q_\phi(s, a) - \hat{y} \right)^2 \right],$$

where the target signal $\hat{y}$ is given by

$$\hat{y} := r(s, a) + \gamma \mathbb{E}_{a' \sim \pi_\theta(\cdot|s')} [Q_{\bar{\phi}}(s', a') - \alpha \log \pi_\theta(a'|s')].$$

Here, $Q_{\bar{\phi}}$ is a target critic with parameters $\bar{\phi}$, which are an exponentially moving average of ϕ [Mnih et al., 2015]. $(s, a, r, s') \sim \mathcal{D}_m^D$ and $s \sim \mathcal{D}_m^D$ denote independent sampling operations from the model buffer.

The actor is then updated to maximize

$$\mathcal{J}_D(\theta) := \mathbb{E}_{s \sim \mathcal{D}_m^D} [V_{\phi, \theta}(s)],$$

where

$$V_{\phi, \theta}(s) := \mathbb{E}_{a \sim \pi_\theta(\cdot|s)} [Q_{\phi}(s, a) - \alpha \log \pi_\theta(a|s)].$$

4.2.4 Training Rollout-based Methods

In this work, we refer to methods that employ the Model-based Value Expansion (MVE) estimator [Feinberg et al., 2018] as *TR-based* methods. The MVE estimator for a state s uses a T -step TR trajectory $\tau_T = (s_0 = s, a_0, \hat{s}_1, a_1, \dots, \hat{s}_T)$ generated by the policy π_θ and the model p_ψ , to compute:

$$\hat{V}_{\phi, \theta, T}(s) := \sum_{t=0}^{T-1} \gamma^t r_\psi(\hat{s}_t, a_t) + \gamma^T \hat{V}_\phi(\hat{s}_T), \quad (4.4)$$

where

$$\hat{V}_\phi(\hat{s}_T) := Q_{\bar{\phi}}(\hat{s}_T, a_T) - \alpha \log \pi_\theta(a_T|\hat{s}_T) \approx \mathbb{E}_{a \sim \pi_\theta(\cdot|\hat{s}_T)} [Q_{\bar{\phi}}(\hat{s}_T, a) - \alpha \log \pi_\theta(a|\hat{s}_T)].$$

Even when $\hat{V}_\phi(\hat{s}_T)$ is imperfect due to the critic struggling to keep pace with a rapidly changing policy, the partial on-policy reward sequence up to $T - 1$ steps improves the overall value estimate at s [Feinberg et al., 2018]. Although Feinberg et al. [2018] assumed the reward function to be known, learning a reward model r_ψ typically poses smaller difficulties than learning a dynamics model or Q -function, so similar benefits are expected.

The MVE estimator is used exclusively for training the critic [Feinberg et al., 2018] or exclusively for training the actor [Deisenroth and Rasmussen, 2011, Heess et al., 2015, Amos et al., 2021]. The latter approaches are often referred to as Stochastic Value Gradient (SVG) methods. In this study, however, we follow Hafner et al. [2019] and utilize the MVE estimator for both the critic and the actor. Specifically, the critic

is trained to approximate the Q function $Q^\pi(s, a)$ by minimizing

$$\mathcal{L}_Q(\phi) := \mathbb{E}_{(s,a) \sim \mathcal{D}_e} \left[\left(Q_\phi(s, a) - \hat{Q}_{\bar{\phi}, \theta, T}(s, a) \right)^2 \right],$$

where

$$\hat{Q}_{\bar{\phi}, \theta, T}(s, a) := r_\psi(s, a) + \sum_{h=1}^{T-1} \gamma^h (r_\psi(\hat{s}_h, a_h) - \alpha \log \pi_\theta(a_h | \hat{s}_h)) + \gamma^T \hat{V}_{\bar{\phi}}(\hat{s}_T)$$

For the actor, we maximize

$$\mathcal{J}_T(\theta) := \mathbb{E}_{s \sim \mathcal{D}_e} \left[\hat{V}_{\phi, \theta, T}(s) \right].$$

A key point here is that the gradient $\nabla_\theta \hat{V}_{\phi, \theta, T}(s)$ is computed by combining the reparametrization trick [Kingma and Welling, 2013] with backpropagation through time. It has shown that when T becomes large, this process can be prone to gradient explosion [Doya, 1992, Parmas et al., 2018]. Hence, in practice, T is often limited to about 5 steps to maintain stable optimization [Amos et al., 2021].

4.3 Double Horizon Model-Based Policy Optimization

This section introduces Double Horizon Model-Based Policy Optimization (DHMBPO), which integrates DR and TR to approximate both the state distribution and the value function in a manner closer to the current policy. By doing so, DHMBPO enables efficient policy improvement even under model bias and off-policy conditions.

As defined in eq. (4.2), the objective in RL is to maximize the expected value $V^{\pi_\theta}(s)$ under the state distribution $\rho_{\pi_\theta}(s)$ induced by the policy π_θ :

$$\mathbb{E}_{s \sim \rho_{\pi_\theta}(s)} [V^{\pi_\theta}(s)].$$

However, it is difficult to handle $\rho_{\pi_\theta}(s)$ and $V^{\pi_\theta}(s)$ directly. Existing methods approximate them through alternative distributions and estimators.

MBPO approximates $\rho_{\pi_\theta}(s)$ via model-buffer samples:

$$\mathbb{E}_{s \sim \rho_{\pi_\theta}(s)} [\dots] \approx \mathbb{E}_{s \sim \mathcal{D}_m^D} [\dots],$$

and replaces $V^{\pi_\theta}(s)$ with the output of the critic $Q_\phi(s, a)$. This approach, based on DR, helps reduce reliance on off-policy data, allowing learning to proceed on a state

distribution that is more reflective of the current policy’s behavior.

On the other hand, TR-based methods approximate $\rho_{\pi_\theta}(s)$ with samples from the replay buffer $\mathcal{D}_e$:

$$\mathbb{E}_{s \sim \rho_{\pi_\theta}(s)}[\cdots] \approx \mathbb{E}_{s \sim \mathcal{D}_e}[\cdots],$$

and replace $V^{\pi_\theta}(s)$ with the T -step MVE estimator, eq. (4.4). Even if the critic Q_ϕ cannot fully keep pace with a rapidly changing policy, the MVE estimator provides a more accurate approximation of Q^π . Especially when the actor and critic are updated alternately, the MVE estimator adapts to the evolving policy more quickly, requiring fewer optimization steps for effective improvement.

In summary, DR produces a more on-policy-like state distribution, while TR (through MVE) mitigates errors arising from an imperfect critic Q_ϕ . However, MBPO uses DR without MVE, and TR-based algorithms such as SAC-SVG(H) [Amos et al., 2021] uses MVE without reshaping the state distribution to be closer to the current policy. This work combines DR and TR to achieve

$$\mathbb{E}_{s \sim \rho_{\pi_\theta}(s)}[\cdots] \approx \mathbb{E}_{s \sim \mathcal{D}_m^D}[\cdots] \quad \text{and} \quad V^{\pi_\theta}(s) \approx \text{eq. (4.4)},$$

thereby reducing off-policy dependence and improving value approximation for the current policy when the policy is updated so rapidly that the critic has not yet converged to accurately estimate its corresponding Q -function.

Concretely, the critic loss function is defined as

$$\mathcal{L}_{D,T}(\phi) := \mathbb{E}_{(s,a) \sim \mathcal{D}_m^D} \left[\left(Q_\phi(s, a) - \hat{Q}_{\bar{\phi}, \theta, T}(s, a) \right)^2 \right], \quad (4.6)$$

and the actor objective is

$$\mathcal{J}_{D,T}(\theta) := \mathbb{E}_{s \sim \mathcal{D}_m^D} \left[\hat{V}_{\bar{\phi}, \theta, T}(s) \right]. \quad (4.7)$$

Algorithm 3 presents the detailed procedure for DHMBPO. Here L is length of an episode and repeating policy optimization L times corresponds to a UTD ratio of 1.

4.4 Experiments

To verify the general effectiveness of the proposed combination of DR and TR, we selected a variety of continuous control tasks based on MuJoCo [Todorov et al., 2012] from Gymnasium (GYM) [Towers et al., 2023] and DMControl (DMC) [Tunyasuvunakool et al., 2020] as test beds, covering tasks ranging from classical control tasks to multi-degree-of-freedom robot locomotion tasks.

Algorithm 3 Double Horizon Model-Based Policy Optimization

Require: Models p_ψ and r_ψ , actor network π_θ , critic network Q_ϕ , target critic network $Q_{\bar{\phi}}$, replay buffer $\mathcal{D}_e$, model buffer $\mathcal{D}_m^D$

- 1: **for** $e = 0, 1, 2, \dots$ **do**
- 2: Interact with the environment using π_θ and add observed transitions to $\mathcal{D}_e$
- 3: Fit the models p_ψ and r_ψ to samples from $\mathcal{D}_e$
- 4: Sample states from $\mathcal{D}_e$; then perform D -step DR and store the generated transitions in $\mathcal{D}_m^D$
- 5: **for** $i = 0$ to $L - 1$ **do** $\triangleright$ Policy optimization loop
- 6: Sample states from $\mathcal{D}_m^D$ and execute T -step TR
- 7: Compute the MVE estimates (4.4) and (4.5)
- 8: Update ϕ to minimize (4.6) and then update θ to maximize (4.7)
- 9: Clear $\mathcal{D}_m^D$

The objectives of the experiments presented in this section are as follows:

- Does the combination of DR and TR contribute to improving sample efficiency?
- Does the combination contribute to the efficient learning of the critic?
- How does DHMBPO perform compared to other methods?

All experiments were conducted using common hyperparameters, the details of which are provided in Appendix B.3. Following [Janner et al., 2019], we used variant tasks from GYM for the Humanoid and Ant environments, with some state dimensions truncated to lower dimensions. In this section, we define the agent’s performance after interacting with the environment for x steps as the *test return*, which is the sample mean of the cumulative rewards over 10 episodes.

In Section 4.4.1, we confirm the effect of the combination of DR and TR, and in Section 4.4.3, we compare the proposed method to the high performance MBRL algorithms.

4.4.1 Comparison of DHMBPO with Ablated Algorithms

Sample Efficiency

Figure 4.2 shows the experimental results evaluating the sample efficiency of the proposed DHMBPO algorithm and its variants on a set of 10 continuous control benchmark tasks. We selected 5 representative tasks from the GYM suite (upper row) and 5 relatively challenging tasks from the DMC suite (lower row). All compared methods share the same model-learning modules and implementation details, differing only in how the DR and TR are combined.

We consider three configurations of DHMBPO corresponding to different (DR length, TR length) settings:

- (20, 5): The proposed DHMBPO algorithm, which uses both DR and TR.
- (0, 5): DHMBPO without DR, i.e., it uses only TR. This setup is conceptually equivalent to SAC-SVG(H) [Amos et al., 2021].
- (20, 0): DHMBPO without TR, i.e., it uses only DR. This setup is conceptually equivalent to MBPO [Janner et al., 2019].

For each configuration, we ran experiments with 8 different random seeds. The solid lines in Figure 4.2 represent the mean performance across these seeds, while the shaded areas indicate the 95% confidence intervals.

The results show that combining DR and TR outperforms using either DR or TR alone. In particular, the DHMBPO configuration (20, 5) achieves higher returns and better sample efficiency compared to the two reduced versions. For example, the TR-only variant (0, 5) corresponds to SAC-SVG(H); while it benefits from improved value estimation, the absence of DR leads to reliance on off-policy samples, resulting in less sample efficiency in challenging tasks like Humanoid, Walker2d, and quadruped-run. Meanwhile, the DR-only variant (20, 0) corresponds to MBPO; although it can approximate the on-policy state distribution, the lack of TR’s MVE estimation fails to provide effective value corrections, leading to notably inferior performance on all tasks.

Overall, these results demonstrate that the synergy achieved by combining DR and TR in DHMBPO leads to faster learning than either technique alone. In other words, the proposed DHMBPO setup (20, 5) effectively balances the need to approximate on-policy state distributions and refine value estimation, thereby achieving superior sample efficiency across a wide range of continuous control tasks.

Efficient Critic Learning

Experimental Setup We investigated how DR affects the learning of a critic in DHMBPO. Concretely, we ran DHMBPO for 100K environment steps on five environments from the Gym suite. After this training, the learned dynamics model and actor network were fixed. We then re-initialized a critic network from scratch and compared two variants for critic learning:

- **with DR:** Use model-generated rollouts (stored in a model buffer) to train the critic,
- **without DR:** Use only samples from the replay buffer, without additional model-based rollouts.

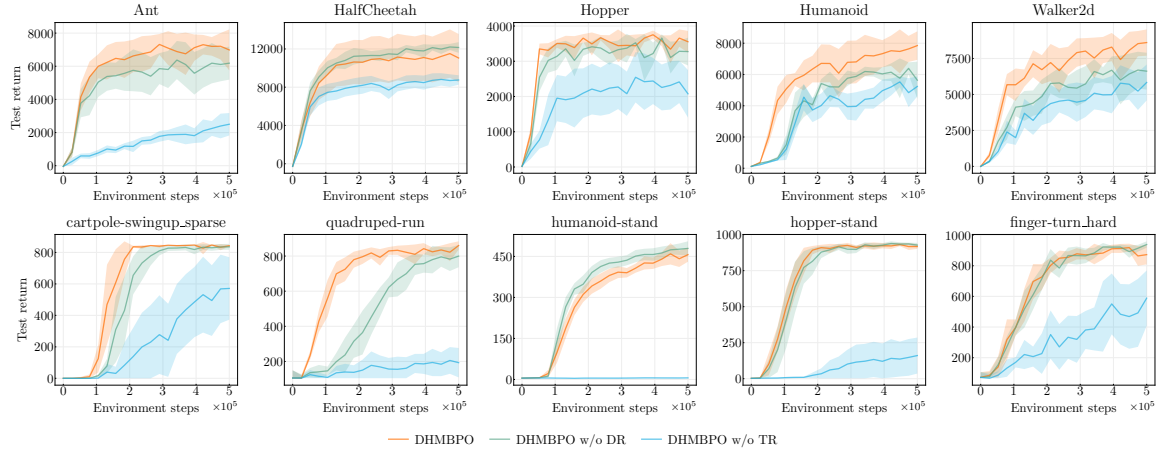


Figure 4.2: Performance comparison of DHMBPO (orange) against its two ablated variants: DHMBPO without Distribution Rollout (w/o DR, corresponding to SAC-SVG(H), green) and DHMBPO without Training Rollout (w/o TR, corresponding to MBPO, blue). The top row shows results on five representative tasks from the GYM suite, and the bottom row shows results on five more challenging tasks from the DMC suite. Each solid line represents the mean test return over 8 random seeds, and the shaded regions denote the 95% confidence intervals. The proposed DHMBPO configuration (DR length = 20, TR length = 5) consistently outperforms both variants that omit one of the rollouts, demonstrating the importance of combining DR and TR for more efficient learning.

As ground truth values, we computed a Monte Carlo return $\hat{Q}_g(s, a)$ from 2,048 rollouts in the real environment, for each of 256 randomly sampled state-action pairs (s, a) taken from the replay buffer. We then calculated MVE estimate $\hat{Q}_{\bar{\phi}, \theta, 5}(s, a)$ for the 256 state-action pairs, and measured its discrepancy from $\hat{Q}_g(s, a)$ via a normalized Root Median Squared Error (RMedSE):

$$E(i) := \sqrt{\text{Median}_{(s,a) \sim \mathcal{D}} \left[\left(\frac{\hat{Q}_g(s, a) - \hat{Q}_{\bar{\phi}, \theta, 5}(s, a)}{\hat{Q}_g(s, a)} \right)^2 \right]},$$

where i is the critic optimization step. “Median” denotes the median over the selected 256 state-action pairs. For each setup, we repeated the procedure with 32 different random seeds for the critic’s initialization and plotted the mean (solid line) and the 95% confidence intervals (shaded area) of $E(i)$ as a function of i (see Figure 4.3).

In both the Humanoid and Hopper environments, we observe that *with DR* exhibits a significantly lower estimation error, particularly at the early stages of training. This difference indicates that the MVE estimator is more accurately approximating the true value function. Since the policy is continuously updated throughout training, an early advantage in value estimation can accumulate and benefit each subsequent

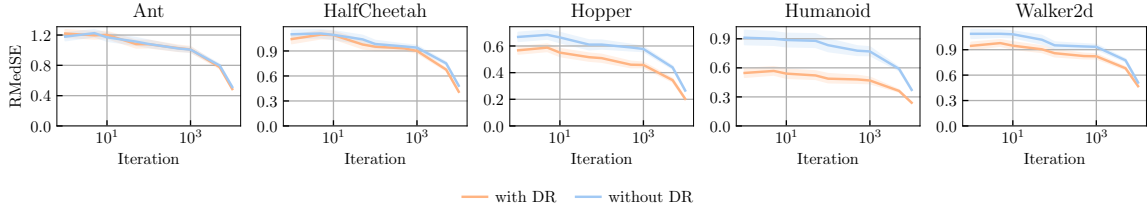


Figure 4.3: Development of the RMedSE for DHMBPO (Orange) and DHMBPO without DR (Blue). The x -axis is the number of iteration for update of critic parameter while all other models are fixed. RMedSE in the y -axis is the normalized root-median-squared-error about Monte Carlo return from the target environment.

optimization step. In the Humanoid environment specifically, we believe this improved value estimation under DR leads to the notable gains in sample efficiency shown in Figure 4.2.

4.4.2 Hyper-parameter Sensitivity Analysis

Figure 4.4 illustrates our sensitivity analysis of DR and TR horizon hyperparameters. We analyzed test returns after 500K environment steps. To avoid bias toward any single metric, the figure reports three summary statistics—median, inter-quartile mean (IQM), and mean—along with 95% confidence intervals. The IQM score is the mean over samples between the first and third quartiles.

We chose DR=20 and TR=5 as our baseline. In Figure 4.4a, we varied only the DR horizon (0, 10, 20, 40), while keeping TR=5 fixed. In Figure 4.4b, we varied only the TR horizon (1, 3, 5, 7, 9), while keeping DR=20 fixed. The results show that: 1). DR benefits from being relatively long (e.g., 20) compared to short horizons (e.g., 0). 2). TR=5 offers a good balance between sample efficiency and gradient stability. Short TR reduces sample efficiency (similar to “DHBPO w/o TR” in Figure 4.2), while too long TR can cause gradient norms to explode (see Appendix B.5.1, Figure B.5b), destabilizing learning.

Overall, these findings suggest setting DR to a moderately long horizon and TR to a shorter, carefully chosen horizon to prevent gradient explosion and maintain robust performance.

4.4.3 Comparison with State-of-The-Art Algorithms

In this section, we compare the proposed DHMBPO algorithm with several high performance deep MBRL algorithms whose implementations are publicly available.

For the comparison on the GYM suite, we selected five tasks and compared DHMBPO

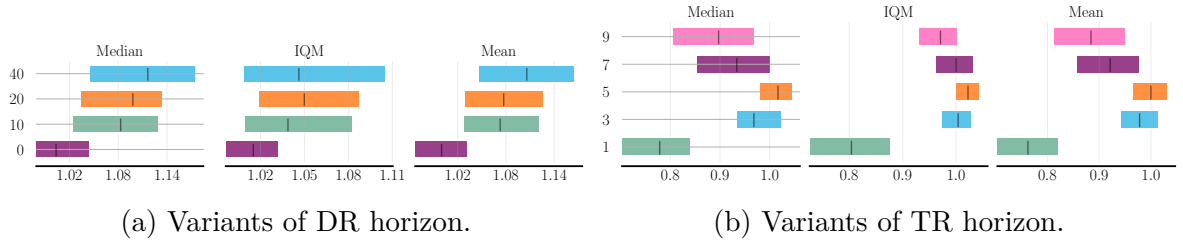


Figure 4.4: Aggregation metrics on 10 tasks for (a) variants of different DR horizon and (b) variants of different TR horizon.

with MBPO [Janner et al., 2019], SAC-SVG(H) [Amos et al., 2021], and MACURA [Frauenknecht et al., 2024]. In addition to MBPO, SAC-SVG(H) is an important baseline algorithm, which uses SVG method, a kind of TR-based algorithms. The primary difference between DHMBPO without DR and SAC-SVG(H) is that DHMBPO uses Deep Ensemble models [Lakshminarayanan et al., 2017], whereas SAC-SVG(H) uses a GRU [Cho, 2014] as the deterministic recurrent neural network. Additionally, SAC-SVG(H) learns the critic using one-step on the TR.

MACURA is based on the MBPO algorithm but adaptively adjusts the length of the DR and reported to achieve state-of-the-art performance on the GYM suite.

For the comparison on DMC suite, we selected 18 tasks and compared DHMBPO with TD-MPC2 [Hansen et al., 2024] and Dreamer v3 [Hafner et al., 2023]. Both of these are latent model-based actor-critic algorithms that have demonstrated exceptional performance across multiple benchmark suites, including DMC, using common hyperparameters. TD-MPC2 combines representation learning and reward learning, and during behavior execution, it performs model predictive control (MPC). In this study, however, we report results without MPC for a fair comparison. Dreamer v3 combines representation learning with SVG and has shown high control performance even on image-based tasks.

The results on GYM tasks are shown in Figure 4.5, where the sample mean over eight random seeds is plotted as the solid line and 95% credible intervals are shown as the shaded area. For MBPO, five random seeds were used. It is confirmed that DHMBPO achieves the highest sample efficiency across all tasks even without per-task tuning. This is practical but perhaps the most important contribution of this work.

Next, we present the results of the comparison on DMC tasks in Figure 4.6. Here, we use the statistical aggregation tool rliable Agarwal et al. [2021] to normalize the test return values between 490K and 500K environment steps based on the sample mean of DHMBPO’s test return. The three metrics for each method are shown, where the vertical bars represent the metrics, and the width of the bars corresponds to the 95% confidence intervals. The IQM score shows that DHMBPO exhibited significant

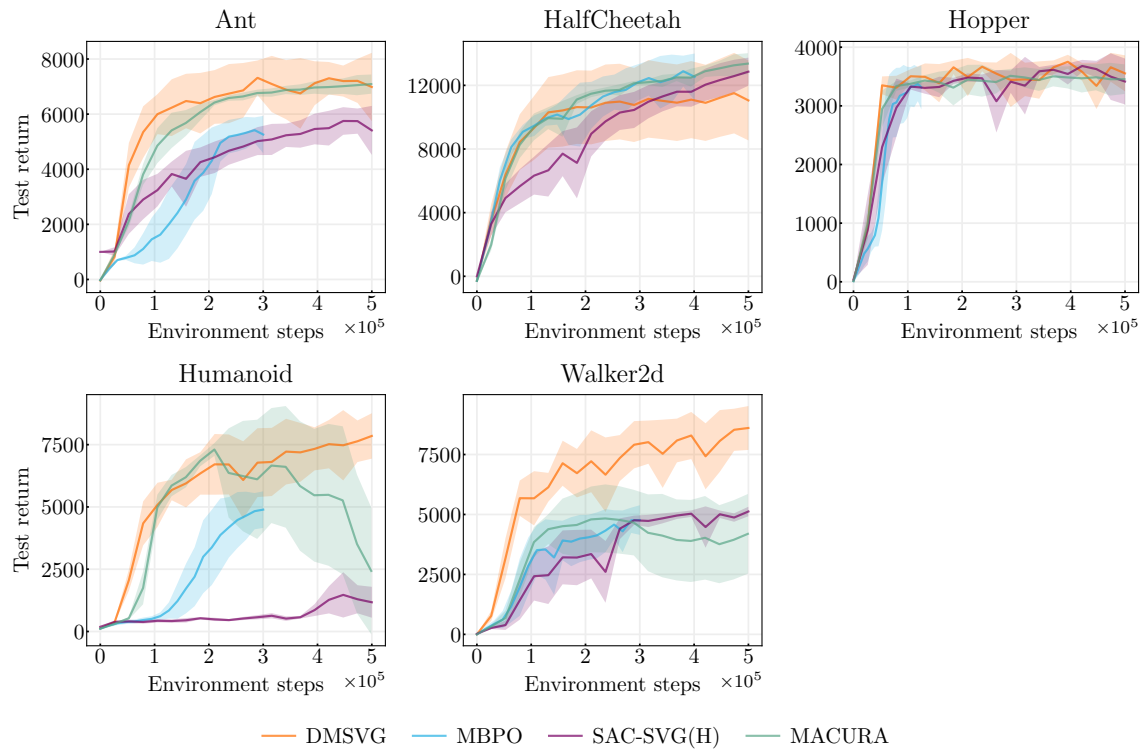


Figure 4.5: Comparison of sample efficiency with state-of-the-art methods on GYM tasks. The solid line represents the sample mean over eight random seeds, and the shaded area indicates the 95% confidence intervals. DHMBPO shows superior sample efficiency, particularly excelling on the Walker2d task.

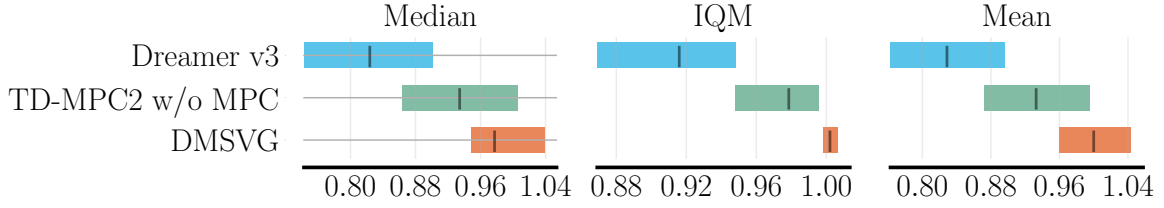


Figure 4.6: Aggregation metrics normalized by DHMBPO’s scores. The IQM score shows significant improvement in performance, demonstrating DHMBPO’s advantage over other methods on the DMC tasks.

Table 4.1: Runtime on GYM and its ratio to DHMBPO’s runtime (in hours).

Task	DHMBPO	MACURA	SAC-SVG(H)
		[Frauenknecht et al., 2024]	[Amos et al., 2021]
Ant	3.6	58.3	5.2
HalfCheetah	3.3	32.6	6.7
Hopper	3.7	73.3	6.7
Humanoid	5.2	104.3	6.9
Walker2d	4.0	63.0	6.6
Ratio	1.0	16.8	1.6

performance improvement over the other two methods.

Additional task-specific learning curves and runtime results are provided in Section B.4.

Runtime Comparison

As shown in Table 4.1, we compare runtime of DHMBPO, MACURA and SAC-SVG(H) GYM tasks. Each experiment was executed until 500K environment steps. The computer configuration was NVIDIA RTX A4000 16GB $\times$ 8, AMD EPYC 7513 32-Core Processor @ 1.5GHz $\times$ 2 or NVIDIA RTX A4000 16GB $\times$ 8, Intel(R) Xeon(R) Silver 4310 CPU @ 2.10GHz $\times$ 2. We believe that there are not so much difference about their runtime because the difference is configuration of CPU but the most process are on GPU.

Taking into account the aforementioned results on sample efficiency, it can be seen that the DHMBPO algorithm reached 500K steps in less than one-sixteenth of the runtime, while achieving the same or higher degree of sample efficiency as the MACURA algorithm. The main reason why MACURA took relatively long time is its UTD ratio, the number of update of the policy per observing a data from the environment. For example, the UTD ratio for MBPO on HalfCheetah task was 40, while that of DHMBPO (and SAC-SVG(H)) was 1. Although a high UTD ratio promotes higher

sample efficiency [Chen et al., 2020, D’Oro et al., 2023], we observe DHMBPO worked well with the UTD ratio of 1, enabling us to save its runtime.

4.5 Related Work

Many practical off-policy actor-critic methods, including the SVG algorithm, rely on uniform sampling from the replay buffer. While importance sampling corrections have been proposed in model-free reinforcement learning (MFRL) [Hallak and Mannor, 2017, Zhang et al., 2020], they are prone to issues such as high variance in the likelihood ratios [Liu et al., 2019].

In MBRL, MBPO [Janner et al., 2019] introduced DR, achieving high sample efficiency. However, MBPO suffers from the issue of requiring extensive runtime. In contrast, while MBPO uses the model solely for generating virtual data, the SVG method is expected to leverage the differentiability of the model directly in the optimization of the policy, enabling more efficient learning.

PILCO [Deisenroth and Rasmussen, 2011], which employs Gaussian processes (GPs) as a model, demonstrates very high sample efficiency by performing policy optimization using SVG on the analytical distribution of the cumulative reward. However, when using sample-based predictions instead of assuming an analytical distribution, the gradient estimation becomes unstable, especially for long prediction horizons, as errors accumulate over time [Parmas et al., 2018]. This issue is particularly pronounced in tasks where accurate long-term predictions are crucial.

Furthermore, as discussed in Section 4.3, SVG methods have not adequately addressed the problem of distribution mismatch in the objective function. Specifically, Feinberg et al. [2018] explored the usefulness of model-based value estimation and derived conditions for an on-policy stationary distribution, but the utility of combining these conditions with SVG for policy optimization was not made clear. In this study, we introduce a novel approach that uses the MVE estimator for both actor and critic training, resolving the distribution mismatch issue. This constitutes the key novelty of our work.

Finally, while it is possible to consider using IS to resolve the distribution mismatch, as seen in the flow of MFRL, our study focuses on achieving practical performance improvements. We demonstrate that simply adding the Distribution Rollout used in MBPO leads to significant performance gains. This simple yet effective approach proves to be highly beneficial in real-world RL problems.

4.6 Conclusion

In this study, we introduced DHMBPO, a new MBRL algorithm that integrates two separate model rollouts—a long-horizon Distribution Rollout (DR) and a short-horizon Training Rollout (TR)—to tackle the trade-off between limited data (and thus potential model-bias) and reliance on off-policy samples in MBRL. DR resamples from the model to better approximate the on-policy state distribution, whereas TR uses a shorter horizon to compute the Model-based Value Expansion (MVE) estimator, helping to refine value function updates. By assigning different horizon lengths to DR and TR, we preserve sample efficiency without incurring excessive model-bias.

Experimental results on multiple continuous-control tasks indicate that DHMBPO consistently achieves faster learning and reduced runtime compared to existing MBRL baselines. We observe that the synergy of a long DR with a short TR enables early improvements in value estimation, which accumulate over successive policy updates. Although the method provides a more balanced solution to the model-bias versus off-policy-data challenge, exploration remains an open issue. For instance, certain tasks such as “finger-spin” can still lead to local optima. Nonetheless, the relatively short runtime of DHMBPO facilitates iterative refinement and experimentation, thereby offering a cost-effective path toward practical performance gains in development of real-world reinforcement learning applications.

Chapter 5

A Meta-Q-Learning Approach to Visual Object Tracking

Visual object tracking remains a challenging computer vision problem with numerous real-world applications. Discriminative correlation filter (DCF)-based methods are a recent state-of-the-art approach for dealing with this problem. The learning rate when applying a DCF is typically fixed, regardless of the situation. However, this rate is important for robust tracking, insofar as real-world video sequences include a variety of dynamical changes, such as occlusions, motion blur, and deformations. In this study, we propose a method for dynamically determining the learning rate by means of reinforcement learning, in which we train a function for an image patch that outputs a situation-dependent learning rate in an autonomous fashion. We evaluated this method using two open benchmarks, namely, OTB-2015 and VOT-2105, and found that it outperforms the original DCF tracker. Our results demonstrate the advantages of the so-called meta-learning with DCF-based visual object tracking.

5.1 Introduction

Visual object tracking is an important but challenging task in the field of computer vision. In a visual tracking task, we specify a target at a certain frame of a video and determine its location in the successive frames. This task is important owing to its wide range of real-world applications, such as visual surveillance, robot navigation, and sports analysis [Poppe, 2010]. Visual tracking still includes difficulty, however, because such applications require online tracking in real-time, as well as a high efficiency and performance.

A discriminative correlation filter (DCF) has been used in visual object tracking tasks [Bolme et al., 2010] and has proven to have high computational efficiency and

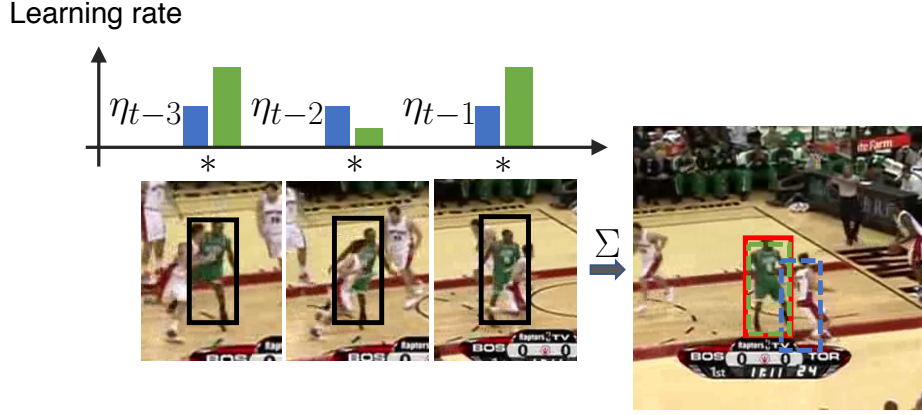


Figure 5.1: DCF learning is self-contained. When a typical DCF tracker takes a new input patch at time $k = t - i$ ($i = 1, 2, 3$), it assigns a prefixed learning rate η_{t-i} , shown with the blue bar. When the target has been occluded, however, this strategy can fail to track at time $k = t$ due to an over-fitting. By contrast, an adaptive learning rate (shown with the green bar) avoids such tracking failures. Here, the red bounding box denotes the ground-truth, a blue dashed bounding box is a possible prediction using a typical DCF tracker, and the green dashed bounding box represents a DCF tracker with an adaptive learning rate.

accuracy. A filter-based method like DCF obtains a heatmap that finds the target pattern by applying a convolution operation within the frequency domain, which is computationally efficient. The convolution kernel used during this process, called a correlation filter, is constructed to discriminate foreground and/or background patterns with high accuracy. To adapt to changes in the foreground and/or background, the correlation filter is updated in an incremental manner. The DCF has been extended for use with kernel tricks [Henriques et al., 2015], adaptation to changes in the scale of the target [Danelljan et al., 2014, 2017, Li and Zhu, 2014], the handling of deformations and partial occlusions of the target [Liu et al., 2015, Bibi et al., 2016], a reduction in the level of confusion [Possegger et al., 2015], and the application of boundary effects [Galoogahi et al., 2015, 2017, Danelljan et al., 2015]. A sequential update of the correlation filter can decrease the robustness against changes in the target appearance, particularly when we consider natural video sequences (see Fig. 5.1). There is a trade-off between robustness and an over-fitting, which is controlled by tuning the learning rate for each dataset [Bolme et al., 2010, Danelljan et al., 2014, 2015, 2016, Galoogahi et al., 2017]. However, when the condition changes across frames in a video sequence, the optimal setting of the learning rates may differ among frames.

In this study, we establish the advantages of adaptive learning rates for the sequential learning of a DCF-based tracker. In effect, we propose the use of a meta-learning technique for the tracking; we call this technique meta-learning, because the tracking procedure like DCF itself includes online learning. Our meta-learning is performed by

offline reinforcement learning (RL), in which we train a Q-function that determines the value of taking an action given a particular state, thus maximizing the total reward given in a sequence of actions and states. We adopted the RL framework to the visual object tracking problem by relating the state with the image patch and the action with the learning rate. Moreover, a reward is offered when a DCF tracker performs well. Based on the RL, we can then expect that the tracker will achieve a good performance under similar dynamical situations in other video sequences. In addition, we used a deep neural network (DNN) to represent the Q-function. We compared our results with and without the use of a DNN, and the results demonstrate a significant improvement when a DNN is incorporated.

Our contributions provided in this study are as follows:

- We show the benefits of a dynamical adaptation of the learning rate when training a DCF-based tracker.
- We propose an RL framework for training a controller that outputs an adaptive learning rate as a meta-learning of the DCF-based tracker, which is done outside of the tracking task.
- We show the advantages of meta-learning in combination with deep learning.

5.2 Related Studies

Because a tracker used in visual object tracking should be self-contained, too much adaptability to the new sample can be harmful, which is a phenomenon called an over-fitting. Such an adaptation, however, is necessary in certain cases of a sudden change, e.g., a change in illumination. Therefore, controlling the trade-off between adaptation and over-fitting is one of the issues for achieving a more robust tracking.

One major approach to this issue is the regularization of model learning [Bolme et al., 2010, Danelljan et al., 2015, Galoogahi et al., 2017] in an incremental manner to reduce an over-fitting while saving memory. Such regularization has been shown to sufficiently alleviate an over-fitting to make the trackers more robust, even in an incremental manner, with greater memory efficiency. A background-aware DCF [Galoogahi et al., 2017] trains the tracker using negative examples as well, which are densely extracted from the wider background.

Although the learning rate for a filter update is basically fixed, it can be controlled adaptively depending on the peak-to-sidelobe ratio (PSR) [Liu et al., 2015]. A higher PSR value denotes more confidence in the target detection. This idea of an adaptive learning rate is based on the intuition that the more the target changes, the more

rapidly the model should adapt to the new sample even under the risk of an over-fitting.

Another way to mitigate the unwanted effects of an over-fitting is to learn at the lowest risk of such an occurrence to the samples applied [Danelljan et al., 2016]. Potentially corrupted samples are down-weighted while increasing the impact of correct samples. There is a relatively low risk of an over-fitting to the correct samples, and therefore these samples are stored in memory for use in tracking.

RL is a machine-learning framework that allows a system to conduct sequential decision-making, even without any knowledge of the system dynamics. Recently, the combination of a tracking algorithm and RL has been proposed, in which the incorporation of DNNs into the algorithm has also been discussed [Zhang et al., 2017, Yun et al., 2017, Choi et al., 2017, Dong et al., 2018]. To achieve a better prediction of the target’s location, in ADNets [Yun et al., 2017] actions were selected along the video sequence from a set of options, including translation moves, scale changes, and stopping, and the policy to select them was trained by RL. Choi et al. [Choi et al., 2017] proposed a visual tracking algorithm with a template selection policy trained by RL. The recently proposed HP tracker [Dong et al., 2018] used RL for dynamically tuning five hyperparameters and demonstrated an improvement in terms of the tracking accuracy.

In this study, on the other hand, we propose the use of a meta-learned policy that solely controls the learning rate of a DCF-based tracker. Unlike the method described in [Danelljan et al., 2016] used for controlling the unwanted effects of an over-fitting according to the samples in the video sequence the tracker is tackling, our method reduces such effects by capturing patterns of changes in the target using a policy learned from other video sequences.

5.3 Background

First, we introduce a DCF-based tracking framework on top of which we apply a learning rate controller. We then present the use of RL, which provides DCF-based trackers with such control.

5.3.1 Discriminative Correlation Filter

A discriminative model used in visual object tracking is a classifier that classifies a given image region or patch as either the target or background. The classifier consists of a discriminatively trained convolution filter f .

Provided that we have a set of training samples $\{x_k, y_k\}_{k=1}^t$ at time t . A label map $y_k \in \mathbb{R}^{M \times N}$ is the desired output of the convolution operation, which takes a

peak positive value around the target region. In addition, x_k is a feature map at time k consisting of $x_k^l \in \mathbb{R}^{M \times N}, l = 1, \dots, L$ extracted from an image patch, where L denotes the number of channels. We may use arbitrarily pre-defined image features such as histogram of oriented gradient (HOG) [Dalal and Triggs, 2005] features. By defining a convolution filter of channel l as a matrix $f^l \in \mathbb{R}^{M \times N}$, we derive an output that integrates the L filters as follows:

$$S(x; f) = \sum_{l=1}^L x^l * f^l, \quad (5.1)$$

where $*$ denotes a cyclic convolution. The filter f is optimized according to the regularized least squares method, whose error is the L_2 error between the filter heatmap matrix $S(x_k; f)$ and the desired output y :

$$\epsilon_t(f) = \sum_{k=1}^t \alpha_k \|S(x_k; f) - y_k\|^2 + \lambda \sum_{l=1}^L \|f^l\|^2, \quad (5.2)$$

where the weight $\alpha_k \geq 0$ adjusts the effect of each training sample, and $\lambda \geq 0$ is the strength of the ridge regularization.

According to the DCF-based tracking, the filter f_t predicts the next target's location using the index at which the filter heatmap matrix is maximal.

5.3.2 Reinforcement Learning

An agent in RL takes an action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$, moves to a new state $s' \in \mathcal{S}$, and simultaneously receives a reward $r(s, a)$ from the environment. Here, a (possibly probabilistic) mapping from state s to action a to be taken in that state is called a policy, which is denoted as $\pi(a|s)$. In this study, we assume that the environmental dynamics and reward delivery are deterministic. That is, given action a_t and state s_t , the reward r_t and the next state s_{t+1} are uniquely determined. The goal of RL is to find a policy π that maximizes the expectation of cumulative rewards, or the Q-function:

$$Q^\pi(s, a) = \mathbb{E} \left[\sum_{i=0}^{\infty} \gamma^i r(s_i, a_i) \middle| s_0 = s, a_0 = a \right],$$

where $0 < \gamma < 1$ is a discount factor emphasizing recent rewards. A policy is said optimal, when it is associated with the optimal Q-function: $Q^*(s, a) = r(s, a) + \gamma \max_{a'} Q^*(s', a')$. That is, $\pi^*(s) = \arg \max_a Q^*(s, a)$ for any state $s \in \mathcal{S}$. When we sample an action a during RL, we do not use the optimal policy induced from the optimal Q-function, but use the following behavior policy:

$$\pi_\epsilon(a|s) = \frac{\exp(\epsilon Q(s, a))}{\sum_{a'} \exp(\epsilon Q(s, a'))} \quad (5.3)$$

where ϵ is a parameter that incorporates an exploration and exploitation into a policy by setting ϵ close to 0 and a relatively large value, respectively.

5.4 Proposed Tracking Framework

In this section, we first argue that in a DCF it is desirable for its learning rate to be dynamically controlled according to the pattern of changes in the target image patches. A policy is therefore required to control this learning rate. In this study, we thus propose the use of RL to identify a policy for a control of the situation-dependent learning rate for DCF-based trackers. In addition, we combine the RL using a data-driven feature extraction technique with deep learning.

5.4.1 How to Weight Samples in a DCF

In many DCF-based trackers, α_i in eq. (5.2), which indicates the weight of each sample in the objective function of learning, is incrementally set based on a fixed strategy; that is, the more recent the observation is, the more it is emphasized in learning. In an online learning setting, we solve the ridge regression at time t as follows:

$$\hat{\epsilon}_t(f) = \|S(x_t; f) - y_t\|^2 + \lambda \sum_{l=1}^d \|f^l\|^2. \quad (5.4)$$

Therefore, $\epsilon_t(f) = \sum_{k=1}^t \alpha_k \hat{\epsilon}_k(f)$. Along the time-series, we incrementally apply an interpolation to the filter parameters $\dot{f}_t$ as follows:

$$\dot{f}_t = (1 - \eta) \dot{f}_{t-1} + \eta f_t^*, \quad (5.5)$$

where $f_t^* := \arg \max_f \hat{\epsilon}_t(f)$ is a local model in terms of time; $\dot{f}_{t-1}$, the previous filter, is a global model at $t - 1$; and η the fixed learning rate.

The learning rate is important for robust tracking because it controls the balance between newly observed features and past estimations, i.e., the local model and the global model. Thus, if the target changes its appearance rapidly, the tracker should focus on the more recent features of the target, whereas if the target changes only slightly, the tracker can retain the information with the global model. We explore this in greater detail below.

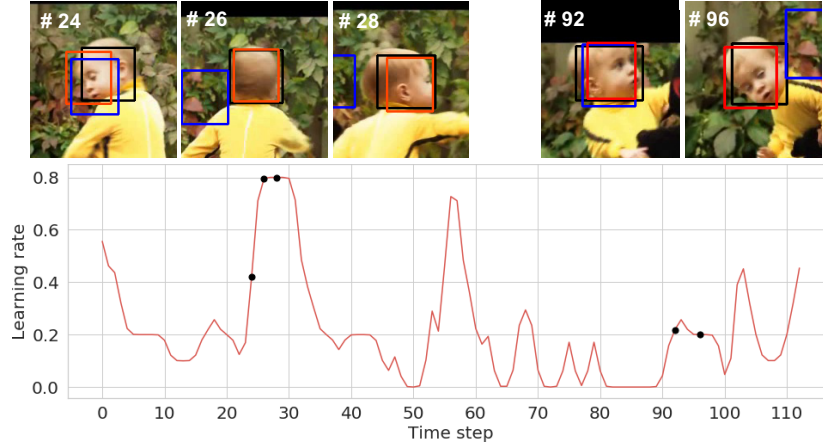


Figure 5.2: Snapshots of tracking results (top) and a plot for a typical profile of the learning rate η over time (bottom). The tracking results are those using BACF with a fixed parameter (blue) and manually set parameter time-series (red). Here, black indicates a human-annotated bounding box, and is regarded as ground-truth. The bottom plot shows changes in the learning rate, which was annotated manually. With the rapid changes of the z-rotation of the target, shown in snapshots, higher learning rates were required near the time indicated by the black dots.

5.4.2 Necessity of Adaptation

Figure 5.2 shows snapshots of tracking with a baseline DCF-based tracker (in particular, we used BACF, whose tracking is shown as the blue bounding box, where AUC, a tracking performance indicator described in Sec 5.5.4, was 0.485), with a plot for a typical profile of the learning rates over time.

Here, we describe the result of a simple experiment. For each frame of the video clip, we manually tuned the learning rate parameter that was used for the model update in eq. (5.5); this manual tuning was based on the intuition discussed above. The plot under the snapshots shows a series of these manually tuned learning rates, with which the tracker achieved a much better performance (red bounding box in the snapshots, with $\text{AUC} = 0.568$). This observation encourages the idea that the adaptive tuning of η in eq. (5.5) will improve the performance of the tracker.

5.4.3 Q-Correlation Filter

The observation above suggests that a controller, or policy, is required to control the learning rate in a situation-dependent manner.

This requirement is difficult to be attained, because there is no systematic method to annotate an ideal learning rate to each state, due to the dynamic nature of general visual object tracking problems. The concept of RL is viable, however, because it has the potential to solve a credit assignment problem in dynamic and even unknown

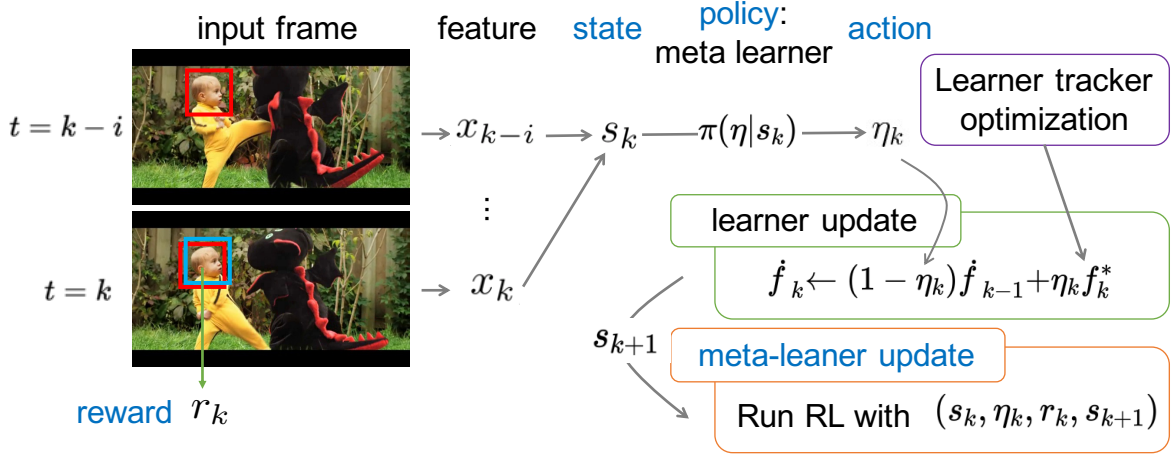


Figure 5.3: Schematic of our proposal. At time $t = k$, we derive state s_k consisting of past and current observation features $\{x_{k-i}, \dots, x_k\}$. The red bounding box denotes the region from which these features are extracted. Our policy π outputs an action, which is the learning rate for the learner tracker update based on state s_k . The model is updated using η_k . For RL, the reward is calculated by comparing the tracker's estimation (red bounding box) with the ground-truth (blue bounding box). Blue words correspond to the components used during RL

environments.

To incorporate RL into the tracking algorithms, we consider a policy $\pi(\eta|s)$, whose input s is a series of observed features and whose output η is the learning rate used for incremental learning in eq. (5.5).

We assume that up to time $t = k$, we have a series of observed features $\{x_{k-i}, \dots, x_k\}$, which are the tracking results. Note that x_{k-j} is dependent on $\dot{f}_{k-j-1}$. We obtain a state at time k : $s_k = g(x_{k-i}, \dots, x_k)$, where $g(\cdot)$ is an arbitrary function, e.g., the Euclidian distance between x_k and x_{k-1} with $i = 1$, or a representation obtained through a data-driven feature extraction technique such as deep learning.

Given a reward function and an environment with an embedded tracker, we can learn a policy of the learning rate by RL. That is, an incorporation of a tracker into RL enables us a meta-learning of the learner tracker in a sole data-driven fashion, without any need to set a pre-determined rule to dynamically control the learning rates. The definition of a reward can be an overlap between the bounding box that has been output by the learner tracker and the ground-truth. Please refer to Sec. 5.5.1 for the definitions of a state and reward function.

For the RL algorithm we define the Q-function, and thus we call the meta-learning algorithm for the DCF-based tracker Q-Correlation Filter (QCF). Since we expect the generalization ability of the RL-based meta-learning, we applied the meta-learning in an offline manner, as outside of the target dataset; that is, we did not use the videos

to be tested when meta-training the QCF.

To investigate the applicability of the meta-learning to the tracking, we first introduce a simple implementation based on the SARSA algorithm [Rummery and Niranjan, 1994]. Then, in the following section, we extend the algorithm to equip the policy with larger generalizability, with the help of deep neural networks (DNNs). The overall tracking and meta-learning schemes are shown in Fig. 5.3.

5.4.4 Adaptive update through Deep Q-learning

Next, we discuss how to represent a state that becomes an input for the policy. One simple way to represent a state, as used with the QCF (in Sec. 5.5.1), is to use the Euclidean distance between two consecutive image patches, reflecting the idea that the policy is dependent on changes in the target behavior.

In this study, as an extension of the QCF above, we propose the use of a DNN to capture the patterns of appearance changes in a data-driven fashion, and such our proposal is called Deep QCF (DQCF). Among the family of methods that combine DNNs and RL, the Deep Q-network (DQN) [Mnih et al., 2015] algorithm provides a suitable architecture for our purpose owing to its inclusion of a convolutional neural network (CNN) [LeCun et al., 1998]; CNNs are known to be robust to image blur, rotations, shifts, and other factors.

Our DQN takes a sequence of i image patches as an input, and outputs a d -dimensional vector $\mathbf{a} \in R^d$. The output $\mathbf{a}$ is a vector of Q-values, each of which represents the goodness of the corresponding discretized learning rate.

The pseudo-code is shown in Algorithm 4. Here, for the purpose of illustration, y and x have ambiguities in the image patch matrix or its location. The function *local_opt* returns a solution to the minimization problem in eq. (5.4), *detect* outputs the location where the heatmap in eq. (5.1) is maximal, *sample_action* obtains a learning rate by sampling an action along the distribution in eq. (5.3), *global_opt* applies a new parameter by updating eq. (5.5), *get_state* applies a new state by cutting off the oldest patch and adding the newest patch, and *get_reward* returns a reward. Lines 4 through 7 are the initialization of the RL environment. After that an execution of action from the agent and an observation of a new state follow each other.

5.5 Evaluation

We first compared our QCF and DQCF to a baseline DCF method, in particular, BACF [Galoogahi et al., 2017]. For this baseline comparison we used benchmark dataset OTB-2015 [Wu et al., 2015] and divided it into two datasets of OTB50 and

Algorithm 4 DQCF Tracker

Input: 1: $y_1, I_1, \dots, I_T$ and $y_2, \dots, y_T$ (when meta-learning) Output: 2: $x_1, \dots, x_T$ 3: procedure QCF($\mathbf{y}, \mathbf{I}$) 4: $x_1 \leftarrow y_1$ 5: $\hat{f}_1 \leftarrow local_opt(x_1)$ $\triangleright$ eq. (5.4) 6: $x_2 \leftarrow detect(\hat{f}_1, I_2)$ $\triangleright$ eq. (5.1) 7: $s_1 \leftarrow [0, x_1, x_2]$ 8: for $i \leftarrow 1$ to T do 9: $\eta_i \leftarrow sample_action(s_i)$ 10: $s_{i+1} \leftarrow step(\eta_i)$ $\triangleright$ eq. (5.3)	1: function $step(\eta_i)$ 2: $f_i^* \leftarrow local_opt(x_i)$ 3: $\hat{f}_{i+1} \leftarrow global_opt(\eta_i, \hat{f}_i, f_i^*)$ $\triangleright$ eq. (5.5) 4: $x_{i+1} \leftarrow detect(\hat{f}_{i+1}, I_{i+1})$ 5: $s_{i+1} \leftarrow get_state(s_i, x_{i+1})$ 6: if training then 7: $r_i \leftarrow get_reward(y_i)$ $\triangleright$ eq. (5.6) 8: Store an experience tuple $(s_i, \eta_i, r_i, s_{i+1})$ 9: Update Q-function 10: return s_{i+1}
--	---

OTB100. OTB50 is a set of such videos that share with OTB-2013 [Wu et al., 2013] and OTB100 is a set of such videos that are not shared with OTB-2013. Second, we compared DQCF with the other state-of-the-art trackers on OTB-2015 [Wu et al., 2015] and another dataset, VOT-2015 [Kristan et al., 2015].

5.5.1 Definitions of the RL Environment

To apply the RL framework, we define the state and reward and clarify the environment used in our case.

Reward function

We assume that the tracker receives a reward r_k from the environment at each time k during meta-learning based on annotated video sequences.

As the immediate reward r_k , we define the intersection over union (IoU):

$$r_k = \frac{|Y_k \cap G_k|}{|Y_k \cup G_k|}, \quad (5.6)$$

where Y_k is the set of pixels included in the estimated bounding box; G_k is the pixel set belonging to the ground-truth bounding box; and $|\cdot|$ denotes the number of pixels. Note that, during the meta-learning stage, the tracker was trained based on annotated video sequences, and as such it can access the ground-truth bounding box in each image frame. By definition, the reward in eq. (5.6) measures the closeness between the estimated target region Y_k and the ground-truth target region G_k .

State

Because the learning rate for the filter update should reflect changes in the target’s appearance, we first employed an extremely simple definition of the state, which becomes an input for the policy of the QCF tracker.

At time k , based on two consecutive image patches, x_k and x_{k-1} , we calculate the Euclidian distance between the two patches, $\|x_k - x_{k-1}\|$, and divide it by MN , where M and N denote the height and width of the extracted patch, respectively, i.e., $s_k = \|x_k - x_{k-1}\|/MN$ and $s_1 = 0$. This is because the size of patch $M \times N$ is variable over the sequences and should thus be normalized.

By contrast, we used a stack of I image patches $\{x_{k-I}, \dots, x_k\}$ as the input to the DQN. In other words, such image patches are defined as a state $\mathbf{s}_k \in R^{I \times M \times N \times L}$, where I is the time length and L is the number of channels for each image patch.

Tracker inside Environment

We regard the DCF-based tracking as a part of the environment for RL, which takes the learning rate as an input and then outputs a state and a reward defined above. In this study, as the baseline tracker, we employed the BACF [Galoogahi et al., 2017], which has demonstrated a state-of-the-art tracking accuracy on several benchmark datasets.

5.5.2 Training and Evaluation

Meta-learning using RL is performed only offline using multiple annotated videos, whereas tracking coupled with learning the target appearance is performed online for each video sequence. Please note that the word “annotation” here means attaching a bounding box for a single target. We do not need to label a ground-truth learning rate for each image frame.

During the meta-learning stage, we avoided situations in which the tracker apparently failed, because training based on such poorly rewarded situations will hinder the reward-based learning. Thus, we stopped the RL process when the tracker failed to track the target. We regarded the tracker as having failed when it had a moving average AUC of below 0.2 over the most recent 100 image frames. To avoid a sampling bias from the earliest parts of the training videos, the initial frame of each training video was randomly set, and subsequent image frames and concatenated earlier frames were used in this order for RL. Moreover, to introduce robustness to DQN, we added pixel-wise independent Gaussian noise to each of the image patches during the meta-learning stage.

5.5.3 Implementation Details

The DNN architecture used in this study is basically the same as the standard architecture proposed as DQN [Mnih et al., 2015], where a CNN and a fully connected network are connected into a sequential (i.e., hierarchical) architecture. In our particular implementation, we used three consecutive patches, namely, the current, previous, and two-steps before, as a state to be input into the DQN. The DQN output was a vector whose element each corresponds to the Q-value for a discretized learning rate; the rate can take any of 20 discretized values, which were arranged at a log-linear scale of between 0 to 1. Regardless of the size of the image patch cut from each frame, the image patch was resized to 200 pixels $\times$ 200 pixels, and then multiplied with a Hann window, as described in [Bolme et al., 2010].

The discount factor γ for the RL was set to 0.99. The learning rate for the stochastic gradient optimization was set to 2.5×10^{-4} . We utilized the RMSProp optimizer with a momentum of 0.95. For the training of the DQN, experience replay buffer and fixed target Q-network techniques were incorporated [Mnih et al., 2015]. The parameters were synchronized with that of the target Q-network during a 1,000 step period, and the experience tuples were randomly sampled. The mini-batch size for DQN training was 32, and the hyperparameter ϵ in eq. (5.3) was decayed exponentially, starting at 1.0 and eventually approaching 0.1. The number of learning episodes was 5,000, where an episode denotes DQN training with a single annotated video.

Other parameters for the tracker were the same as those in BACF. The image features used by BACF were 31-channel HOGs, in line with the original study [Dalal and Triggs, 2005]. We used the default settings for these hyperparameters, and did not tune them.

5.5.4 Quantitative Results

First, we evaluated our trackers on two datasets: OTB50 and OTB100, in a severe manner, i.e., leave-one-dataset-out cross-validation. OTB50 is a subset of OTB-2015 [Wu et al., 2015] that share with OTB-2013 [Wu et al., 2013] and OTB100 is a subset that are not shared with OTB-2013. Each dataset contains 50 sequences respectively. That is, the meta-learning was performed on OTB50 and tested on OTB100, and vice versa. Second, we compared our DQCF tracker with the other state-of-the-art trackers on OTB-2015 and VOT-2015. Over all of these evaluations we used the common architecture for the DQCF, including the size of image patch for the input, regardless of the used database.

	OTB50	OTB100
baseline (BACF [Galoogahi et al., 2017])	69.7	56.1
QCF	67.3	58.0
DQCF	71.5 ± 1.19	58.0 ± 2.10

Table 5.1: Average AUCs for the DQCF and BACF on two benchmark datasets. Owing to the stochastic nature of our proposed trackers, QCF and DQCF, we conducted ten individual runs with different random seeds. The table shows the mean and standard deviation of the DQCF performance for all videos over the ten runs. The standard deviation over the ten runs for the QCF was omitted in the table because it was small (below 0.01) for both datasets. The meta-learning of QCF and DQCF was performed in a leave-one-dataset-out cross-validation.

Q-Correlation Filter

Table 5.1 shows the results, average performance for the two datasets, in terms of the area-under-the-curve (AUC) [Wu et al., 2013]. The AUC measures an area under the success plot curve showing the percentage of bounding boxes whose IoU scores are larger than a given threshold. Therefore, the larger the AUC is, the better the tracking performance. Although the QCF utilized the simplest RL based on SARSA, incorporating the naive state representation of difference between two consecutive image patches, our meta-trained tracker showed a better performance for OTB100 than the baseline. However, this improvement was not as significant as expected. One reason for this moderate improvement might be the state representation. Because we relied on RL, the trained policy should output an appropriate learning rate for the filter update, in which the target situations in the test videos are similar to those in the training videos. This similarity is solely attributed to the features extracted from the image patches. If these features cannot extract the similarity between the training videos and the test videos, the entire RL scenario will collapse. This observation motivated us to utilize more generalizable features, rather than patch difference-based features, to allow RL to be more feasible when meta-training the DCF-based trackers; thus, we proposed DQCF.

Deep Q-Correlation Filter

In this section, we first show a comparison of DQCF in the same experimental setting as in Table 5.1 except for the test dataset and the definition of the state, as explained above. Fig. 5.4 shows a comparison on OTB-2015 between our DQCF and five state-of-the-art DCF-based trackers, namely, DSST [Danelljan et al., 2014], SRDCF [Danelljan et al., 2015], CSK [Henriques et al., 2015], STAPLE [Bertinetto et al., 2016], and

BACF [Galoogahi et al., 2017]. Moreover, as an important related method [Danelljan et al., 2016], we included BACFdecon in this comparison, which jointly optimized the sample weights and the model filter of the BACF tracker. Although the original study [Danelljan et al., 2016] used the SRDCF, instead of BACF, as the baseline DCF-based tracker, we implemented BACFdecon for making the comparison with our RL-based meta-learning (which also used BACF as the baseline) fair. Note that our RL-based method and the adaptive sample weighting scheme, called decon in [Danelljan et al., 2016], are basically modifications of DCF-based trackers. We here report the results of the DQCF, BACF, and BACFdecon, which were implemented by ourselves, while the other trackers’ results are publicly available. When meta-training DQCF, we used a subset of video sequences registered in Temple-Color [Liang et al., 2015]; this subset did not include the video sequences that are shared with OTB-2015.

In this configuration, our DQCF tracker outperformed all of these state-of-the-art DCF-based trackers including BACFdecon.

Moreover, we conducted experiments on VOT-2015 [Kristan et al., 2015] and observed that our DQCF outperforms BACF in terms of accuracy, robustness, and the expected average overlap (EAO); the accuracy of DQCF was 0.546, with a robustness of 41.2 and EAO 0.194, whereas values of 0.547, 53.7, and 0.179 were obtained by BACF, respectively. Here, the EAO measure is based on an empirically estimated average overlap (as a function of the sequence length) and the typical-sequence-length distribution. The larger the EAO is, the better the tracking performance.

These results demonstrate the effectiveness of meta-learning for an adaptive learning rate for a DCF-based tracker, and the achieved robustness is generalizable to a wide range of video sequences registered in these benchmark datasets.

5.5.5 Qualitative Results

To demonstrate that the DQCF could acquire a reasonable policy for controlling the learning rate, we examined the similarity of the learning rates output from DQCF to those attached by human annotator (Section 5.4.2). Figure 5.5 shows typical results; the adaptive learning rate attained by DQCF was effective especially when the target appearance was drastically altered. For example, from frames #24 to #28, the target exhibited rapid changes in the z-rotation, during which the baseline failed to track the target whereas our DQCF could track the target by increasing the learning rate for those frames. This tendency was consistent with human strategy, although our DQCF seemed more sensitive to the dynamics in the video frames than human annotations. Furthermore, DQCF’s AUC score for this sequence was 0.616, much higher than those of the manual adjustment (0.568) and the baseline tracker (0.485). These results sug-

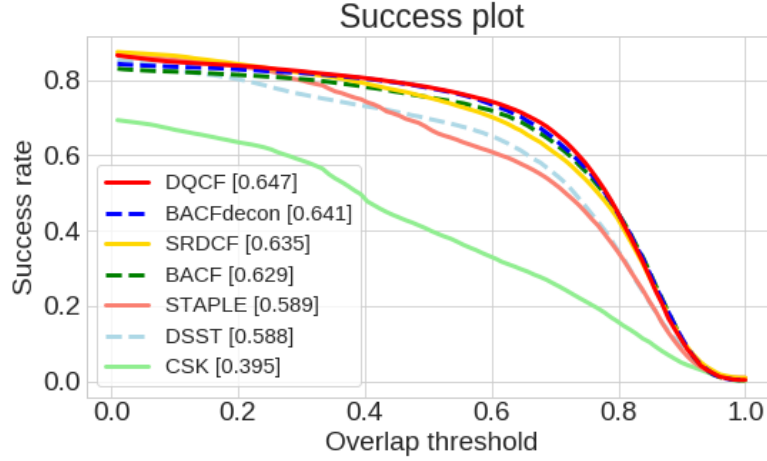


Figure 5.4: Success plots comparing our DQCF with five state-of-the-art DCF-based trackers and a BACF modified with a dynamical setting of the learning rate (i.e., BACFdecon), on OTB-2015. The AUCs are in brackets. Please note that BACF was the common baseline tracker of DQCF and BACFdecon.

gest that our DQCF was able to reproduce human annotation of the adjustable learning rate, and moreover, to attach better annotations than by the human annotator.

5.5.6 Attribute Based Evaluation

Each video sequence in the OTB-2015 dataset is provided 11 labels to describe different challenges in the tracking problem, such as variations in illumination, out-of-plane rotation, and fast motion.

Table 5.2 lists an attribute-dependent comparison between our DQCF and other DCF-based trackers on the OTB-2015 dataset. Like in Sec. 5.5.4, our DQCF did not use the video sequences in OTB-2015 for its meta-learning.

As the results, our DQCF improved the tracking performance over the baseline BACF for all attributes, except for the in-plane rotation and out-of-view attributes. For some attributes such as occlusions, deformations, and motion blur, our tracker showed a substantial improvement because the target changes are not negligible in such cases.

These analyses demonstrate empirically that RL-based meta-learning for a filter learning renders DCF-based tracking more robust under such difficult situations that include unpredictable changes in the target.

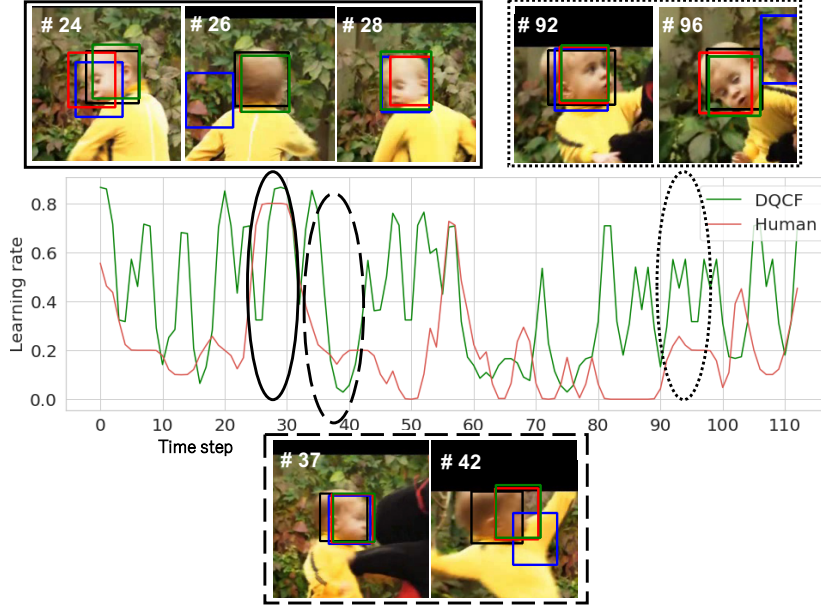


Figure 5.5: Snapshots of tracking results (top and bottom) and a plot of a typical learning rate profile (middle). The green plots and bounding boxes correspond to our DQCF, red indicates a human annotation described in Section 5.4.2, and blue bounding boxes are by the baseline tracker with the fixed learning rate. The black bounding boxes are the ground-truth for the target’s location. DQCF tended to change the learning rates especially when the target appearance was drastically altered; for example, from frames #24 to #28, the target exhibited rapid changes in the z-rotation, so that DQCF increased the learning rate for those frames. It also kept the learning rate to avoid drifts when the appearance was rather consistent.

5.6 Discussion

In Sec. 5.5, we demonstrated that our RL-based meta-learning improves the baseline tracker, when immediate rewards were defined based on ground-truth annotations of the target location and given for every frame of the videos used for the meta-learning. Note that our meta-learned policy could capture the general strategy for dynamically adjusting the online learning rate for the DCF-based trackers. Indeed, the good tracking performance obtained by our very severe meta-learning method, leave-one-dataset-out cross-validation, provides the evidence. We consider our method to be applicable to sequences even without annotations if the reward can be well defined. For example, the mimicking of some majority-of-vote results by multiple existing trackers can be a reward. By contrast, the PSR in the DCF-based trackers can be an alternative because a higher PSR value denotes more confidence in the target detection, and a high confidence correlates with better tracking. In these cases, we do not require any ground-truth annotations at all. Moreover, our method may be able to provide a good performance improvement when some portion of an extremely large database is an-

	IV	SV	OCC	DEF	MB	FM	IPR	OPR	OV	BC	LR
CSK	39	34	36	35	30	37	42	38	32	42	29
DSST	66	57	54	48	52	56	59	55	49	57	52
STAPLE	63	55	57	52	56	57	61	57	53	55	49
BACF	72	60	55	51	64	63	62	58	56	63	61
SRDCF	71	62	57	49	62	65	62	58	55	65	60
BACFdecon	71	63	57	54	69	64	62	59	57	64	62
DQCF	72	63	59	57	69	66	61	61	55	65	61

Table 5.2: Comparison of AUC scores (%) for each attribute of illumination variation (IV), scale variation (SV), occlusion (OCC), deformation (DEF), motion blur (MB), fast motion (FM), in-plane rotation (IPR), out-of-plane rotation (OPR), out-of-view (OV), background clutter (BC), and low resolution (LR) on OTB-2015.

notated, including a weak or semi-supervised annotation [Real et al., 2017]. These extensions of our meta-learning idea remain for future studies.

5.7 Conclusion

In this study, we presented a meta-learning process applied to the filters in DCF-based visual object trackers. In particular, we applied an RL framework to obtain a policy that controls the online learning rate for the sequential update of the filters, whose training was performed outside of the tracking task. We also presented deep Q-learning to better acquire image features in a data-driven manner. We evaluated our proposal, called DQCF, on several open benchmark video datasets and observed that it outperforms state-of-the-art DCF-based trackers, including an alternative method that dynamically adjusts the learning rate. The results show the effectiveness of the use of dynamically control policy for the learning parameter with DCF-based trackers, making them robust to various situations in a generalized manner.

Chapter 6

Conclusion, Limitations, and Future Directions

In this dissertation, we investigated approaches to enhance sample efficiency and adaptability in two distinct domains: robot control (Chapters 2–4) and visual tracking tasks (Chapter 5). While these methods yielded promising results in each domain, several common challenges and limitations remain, as outlined below.

6.1 Limitations

6.1.1 Limited Real-World Validation and the Simulation Gap

The CPG-based methods (Chapters 2 and 3) and the model-based RL approach (Chapter 4) were evaluated in simulated environments, leaving large-scale tests on actual robot hardware or under long-term operation unexplored. In real hardware, issues such as sensor noise, component wear, and communication delays can arise, many of which are difficult to simulate accurately. Likewise, the visual tracking method (Chapter 5) has not yet been assessed on large-scale and diverse camera types over long periods. A key open question, therefore, is whether these approaches can maintain comparable performance and sample efficiency in real-world conditions. Assessing and addressing the gap between simulation and reality remains an essential task for future research.

6.1.2 Overfitting and Limited Exploration

Techniques aimed at improving sample efficiency often rely on relatively small sets of experiences, which can increase the risk of over-adaptation, or over-fitting [Auer, 2002, Fujimoto et al., 2018, Osband et al., 2019]. For example, in the model-based

RL method (Chapter 4), the learning process might repeatedly refine policies that resemble suboptimal behaviors accidentally performed early on, potentially trapping the policy in undesirable solutions. Furthermore, inadequate exploration may prevent the agent from discovering superior policies that exist outside its limited experience. In the visual tracking task (Chapter 5), a restricted dataset can also hinder the tracker’s adaptability to unknown changing patterns, potentially making fluctuating adjustment, while setting fixed learning rate for the pattern is perhaps not bad. Strengthening mechanisms that balance exploration with the avoidance of overfitting is an important area for improvement.

6.1.3 Inductive Bias in Model Classes

The hierarchical policies based on CPGs (Chapters 2 and 3) incorporate biological insights to generate periodic motions, effectively reducing the learning space and contributing to sample efficiency. However, if the choice of base waveforms or parameter settings is too restrictive, the diversity of possible behaviors may be limited, making it difficult to adapt to substantially different environments [Bellegarda and Ijspeert, 2022].

Additionally, the learning-rate adjustment method proposed in Chapter 5 uses a discretely defined set of candidate learning rates, and its effectiveness relative to a continuous action space remains unverified. A continuous space would naturally introduce an ordering of learning rates, which might help suppress extreme oscillations, but this possibility has not been fully explored in this research. Moreover, the method assumes a sequential filter-update process, which complicates, for example, scenarios involving long-term occlusions: once the target disappears and then reenters the sight, there is no mechanism, within the sequential update strategy, to “recall” previous information and restart tracking [Danelljan et al., 2016]. This is a broader challenge for DCF-based approaches, but it equally applies to the learning-rate adjustment method introduced here, indicating the need for further investigation in scenes with significant background changes or long-term target absence.

Such inductive biases can dramatically improve sample efficiency, yet whether they can be applied generally to a wider range of tasks and environments depends on how parameters are designed and how the models are involved.

6.1.4 Lack of Integrated Systems

In this dissertation, robot motion control and visual tracking tasks were treated independently, and specialized methods were developed for each. However, many real-world

applications demand integrated systems that perform actions while using vision [Gross et al., 2009, Gad et al., 2022, Agarwal et al., 2023]. A typical example is a mobile robot that dynamically adapts its movements based on camera inputs from its surroundings. Although the methods proposed here offer effective ways to enhance sample efficiency and adaptability for individual tasks, there has been limited investigation into unifying these approaches in a single framework, such as a world-model-based [Hafner et al., 2019, Hansen et al., 2024] or parallel-learning architecture [Bellegarda et al., 2024]. Pursuing an “integrated system” that jointly optimizes visual control and motion control is an important direction for improving the practical utility of these methods.

Summary of Limitations. Overall, this dissertation leaves open several issues and research opportunities:

- Bridging the gap between simulation and reality through comprehensive experiments on real hardware,
- Balancing overfitting risks and the need for sufficient exploration,
- Designing and extending model classes that leverage their inductive biases while maintaining broad applicability, and
- Developing integrated systems that handle multiple components simultaneously.

While these constraints pose challenges, they also imply important steps in adapting the presented methods to large-scale, diverse real-world environments.

6.2 Future Directions

As discussed in Section 6.1, there remain some challenges regarding real-world validation, leveraging inductive bias, and integrating multiple modules. This section outlines three potential research directions to address these challenges and advance the work presented in this dissertation.

6.2.1 Applying HRL-CPG Methods to a Real Quadruped Robot and Identifying Challenges

The first direction involves combining a GPU-accelerated simulation environment (e.g., Isaac Gym [Makoviychuk et al., 2021] or Genesis [Zhou et al., 2024]) with a medium-to-large-sized quadruped robot (e.g., Aliengo [Robotics]) for physical experiments, using the HRL-CPG methods developed in this dissertation. A practical starting point would

be to transfer policies learned in simulation to the real robot and evaluate how well they can learn to move in a commanded direction. By examining to what extent the simulation-based training can serve as a “hot start” for real-world performance, one can assess the severity of the simulation gap and the robot’s noise tolerance. If significant performance drops or instability arise during on-hardware tests, it may be necessary to incorporate additional mechanisms for exploration or noise-robust policy adaptation, and analyze how these improvements affect both of sample efficiency and stability. Through these iterative steps, we can clarify how issues such as overfitting or inadequate exploration arise in real environments and get a guidance for the solutions.

6.2.2 Integrating CPG Frameworks and Model-Based RL for Enhanced Control

Next, to further improve the efficiency of optimizing a quadruped robot controller, one promising approach is to integrate the CPG’s periodicity with MBRL to balance long-term predictive accuracy and stability. For instance, in Chapter 4, the method introduced distribution rollout and training rollout to mitigate model bias and reliance on outdated samples. By combining that with motion generation via CPG, it may be possible to suppress extreme or unnatural trajectories while realizing more stable, longer-horizon action planning.

Additionally, one could consider adopting a world-model approach [Hafner et al., 2019, Hansen et al., 2024], in which a latent transition model is learned instead of a direct state-transition model. Under this framework, transitions in the latent space are optimized with respect to predicted (cumulative) rewards or Q -values, aiming for an efficient policy optimization that focuses on reward maximization. By incorporating the CPG mechanism into part of this transition model, there is a potential to optimize the base waveforms through this reward-driven learning, rather than manually tuning them. If this type of data-driven training works, we can reduce the tuning cost—which lowers the effective sample efficiency—then learning could become more adaptive and more capable of addressing complex control tasks.

6.2.3 Pursuing Autonomous Systems through Specialized Recognition Modules

Ultimately, one can imagine robots that autonomously determine policies by integrating various sensor modalities, including visual feedback, to perform tasks in dynamic environments. For example, a system could incorporate not only the vision-based tracking method presented in this dissertation, but also object detection or scene recognition

modules, switching or coordinating among these modules depending on the situation. Concretely, a robot might need to search for a specific target while avoiding obstacles, or use a different module to identify a lost target again. In such scenarios, deciding *when* and *how* to transition between different modules becomes a core issue.

Furthermore, under long-term operation, we must account for conditions like complete occlusion of a target, or substantial changes in the surrounding environment over time. The system then needs a robust mechanism for learning when and how to re-identification or re-exploration, to maintain continuous task execution. By integrating multiple specialized recognition modules and optimizing them through reward-driven learning, a more general-purpose robotic intelligence becomes feasible. The foundational techniques for sample efficiency and adaptability developed in this dissertation offer a stepping stone toward realizing that goal.

To summarize, we propose three major avenues:

- Applying the methods to a real quadruped robot to identify practical challenges and refine solutions,
- Integrating CPG-based frameworks with MBRL to achieve stable, long-horizon planning, and
- Pursuing autonomous systems by combining multiple specialized recognition modules under a reward-driven approach.

Advancing these research directions in stages will facilitate a comprehensive examination of learning efficiency, and extensibility, thus promoting the development of algorithms capable of handling increasingly complex and dynamic real-world conditions. By iterating between simulation-based tests and on-hardware experiments, the methods proposed here may demonstrate effectiveness in higher-dimensional tasks and drive further progress in reinforcement learning for robotics.

Appendix A

Appendix of Chapter 2

A.1 Soft Actor-Critic Algorithm

For a particular policy π , we define the following soft Bellman equation:

$$Q^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}' \sim p(\cdot | \mathbf{s}, \mathbf{a})} [V^\pi(\mathbf{s}')], \quad (\text{A.1})$$

$$V^\pi(\mathbf{s}) = \mathbb{E}_{\mathbf{a} \sim \pi(\cdot | \mathbf{s})} [Q^\pi(\mathbf{s}, \mathbf{a}) - \alpha \log \pi(\cdot | \mathbf{s})], \quad (\text{A.2})$$

where Q^π and V^π are the soft Q-function and soft V-function, respectively, and α is the same as in eq. (2.4).

By iteratively applying the soft Bellman equation above, we observe that the soft V-function marginalized over the initial state distribution p_0 becomes the objective of the entropy-based regularized RL in eq. (2.4):

$$J(\pi) := \mathbb{E}_{\mathbf{s} \sim p_0(\cdot), \mathbf{a} \sim \pi(\cdot | \mathbf{s})} [Q^\pi(\mathbf{s}, \mathbf{a}) - \alpha \log \pi(\mathbf{a} | \mathbf{s})].$$

To solve the simultaneous equations, eqs. (A.1) and (A.2), corresponds to the policy evaluation. Hereafter, for the sake of simplicity, we will assume that $\alpha = 1$.

A variational optimization of the objective function, eq. (A.1), with respect to policy π becomes the soft-max of the soft Q-function Q^π . Subsequently, we introduce a policy improvement method such that the current policy approaches the soft-max policy in terms of the Kullback-Leibler (KL) divergence:

$$\pi' := \arg \min_{\tilde{\pi} \in \Pi} D_{KL} \left(\tilde{\pi}(\cdot | \mathbf{s}) \parallel \frac{\exp Q^\pi(\mathbf{s}, \cdot)}{Z(\mathbf{s})} \right),$$

where Π is a desired set of policies, e.g., a parameterized family of distributions such as Gaussian distributions, and $Z(\mathbf{s})$ is a normalization term that validates the soft-max

policy as a probability distribution. The use of the KL divergence allows this policy improvement to proceed in a moderate manner, and enables a policy implementation with a neural network.

The alternative iterations of the above policy evaluation and policy improvement referred to as soft policy iteration, would lead to an optimal policy with respect to the objective function, eq. (2.4).

Because the system dynamics $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ is unknown in a model-free RL setting, we need to solve the simultaneous equations, eqs. (A.1) and (A.2), without the system dynamics. According to the SAC algorithm Haarnoja et al. [2018], we use a least square temporal difference (LSTD) method Bradtke and Barto [1996] to obtain an approximated soft Q-function:

$$J_Q(\theta) := \mathbb{E}_{(\mathbf{s}, \mathbf{a}, \mathbf{s}') \sim \mathcal{D}} \left[\left(Q_\theta(\mathbf{s}, \mathbf{a}) - \left(r(\mathbf{s}, \mathbf{a}) + \gamma \hat{V}(\mathbf{s}') \right) \right)^2 \right],$$

where $\mathcal{D}$ is the dataset of previously sampled tuples of the state, action and next state, and Q_θ is a parameterized function of s and a with parameter θ to approximate the soft Q-function. Similar to the one presented in Haarnoja et al. [2018], we define an approximated soft V-function $\hat{V}$ as

$$\hat{V}(\mathbf{s}) := \mathbb{E}_{\mathbf{a} \sim \pi_\phi(\cdot|\mathbf{s})} [Q_{\bar{\theta}}(\mathbf{s}, \mathbf{a}) - \log \pi_\phi(\mathbf{a} | \mathbf{s})],$$

where $\bar{\theta}$ is the parameter of another function approximator, a target soft Q-function, which is smoothed with an exponentially forgetting weight. That is, we do not define a function approximator for the V-function but approximate it with Monte-Carlo samples using π_ϕ and $Q_{\bar{\theta}}$. This *critic* learning is performed using one-step backups, i.e., the current state $\mathbf{s}$, the current action $\mathbf{a}$, the associated reward r , and the next state $\mathbf{s}'$; the actual algorithm can be done online or offline with minibatches.

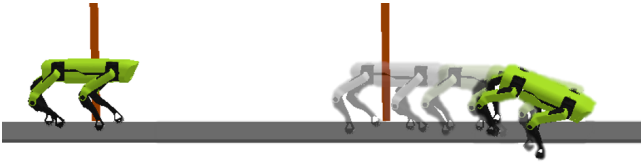
The policy improvement, i.e., the *actor* learning, in the SAC algorithm is performed as

$$\arg \min_{\phi} D_{KL} \left(\pi_\phi(\cdot|\mathbf{s}) \parallel \frac{\exp Q^\pi(\mathbf{s}, \cdot)}{Z(\mathbf{s})} \right) = \arg \min_{\phi} \mathbb{E}_{\mathbf{a} \sim \pi_\phi(\cdot|\mathbf{s})} [\log \pi_\phi(\mathbf{a}|\mathbf{s}) - Q^\pi(\mathbf{s}, \mathbf{a})],$$

where ϕ is the policy parameter. This optimization is conducted in a stochastic gradient decent manner by considering the gradient of the right-hand side of the equation above with respect to the policy parameter.



(a) Example of successful case



(b) Example of failure case due to fall



(c) Front view of a failure case due to fall.

Figure A.1: Stroboscopic photographs: two side-view images of successful and failed cases (a and b), and a front-view image of a failure case (c). The vertical red bars are displayed at 1-meter intervals.



Figure A.2: A screen shot of a preliminary experiment on surfaces with zebra-patterned bands. The white bands have a lower friction coefficient than the black bands.

A.2 Stroboscopic Photography while Walking

Figure A.1 presents stroboscopic photographs: two side-view images of successful and failed cases, and a front-view image of a failure case.

A “failure” is defined as a situation where the agent loses its posture due to events such as falls, as shown in Figure A.1b, before the termination of an episode. “Success” refers to cases where no such failure occurs. The detailed definition is provided at the end of the description of the reward function in Section 2.4.1.

Additionally, Figure A.2 is a screenshot of a preliminary experiment described in Section 2.5 (the end of the 4-th paragraph), where walking occurs on a surface with alternating friction levels at regular intervals, resembling a crosswalk pattern.

A.3 Evaluation of Learning Performance: Failure Rate and Progression Distance

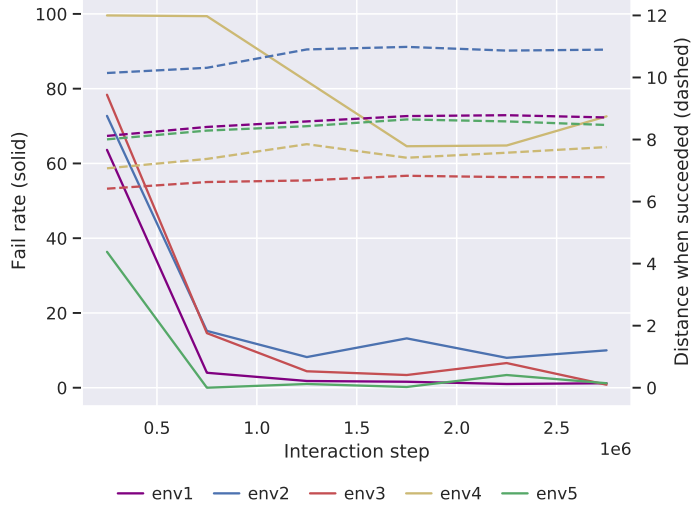


Figure A.3: Failure rate and the progression distance when succeeded.

We investigated the components of reward maximization performance. Figure A.3 shows the learning curves for the failure rate and progression distance during successful episodes. The failure rate is plotted on the left axis, while the progression distance is shown on the right axis. The failure rate was calculated based on 500 test episodes conducted every 500,000 steps (evaluations were performed every 5,000 steps with five random seeds). The progression distance was defined as the mean x-coordinate at the 1,000th step of successful test episodes. Since the episode length was fixed, the progression distance is proportional to the average forward speed along the x-axis.

In the early stages of reinforcement learning (RL), the robot rarely completed episodes in any environment. As training progressed, the failure rate decreased, while the progression distance monotonically increased from the early stages. Table A.4 summarizes the failure rates and progression distances during the final training interval.

The higher progression distance observed in environment2 (downhill) compared to the lower progression distance in environment3 (uphill) reflects differences in the robot's movement speed, which correspond to the maximum return differences shown in Figure 2.6.

These findings indicate that the robot learned to avoid failures while simultaneously acquiring gaits that enabled faster forward progression.

In Figure A.3, the slippery environment4 exhibited relatively frequent failures. However, as training progressed, the robot partially learned to avoid failures. The

Env	Fail rate	Distance [m]
1	1.20 %	8.71
2	10.00 %	10.90
3	0.80 %	6.79
4	72.60 %	7.76
5	1.20 %	8.47

Figure A.4: Fail rate and locomotion distance when succeeded at final interaction steps.

Table A.1: Hyperparameters commonly used in experiments in Section 2.4.

Parameter	Value
Discount (γ)	0.99
Replay buffer size	10^6
Environment step at which HRL-CPG starts (T_{start})	10^4
Stochastic gradient descent optimizer	Adam [Kingma and Ba, 2014]
Number of samples per minibatch (B)	256
Learning rate for critic networks	3×10^{-4}
Learning rate for the phase modulation policy and base frequency modulation policy	3×10^{-4}
Learning rate for μ_r (mean of robot control signal)	1×10^{-8}
Learning rate for σ_r (variance of robot control signal)	3×10^{-4}
The number of hidden units (for all neural networks except the waveform policy)	(256, 256)
Degree of Fourier expansion for robot controller (K)	5
Smoothing coefficient for two target networks (τ)	5×10^{-3}
Activation function used for all layers	ReLU

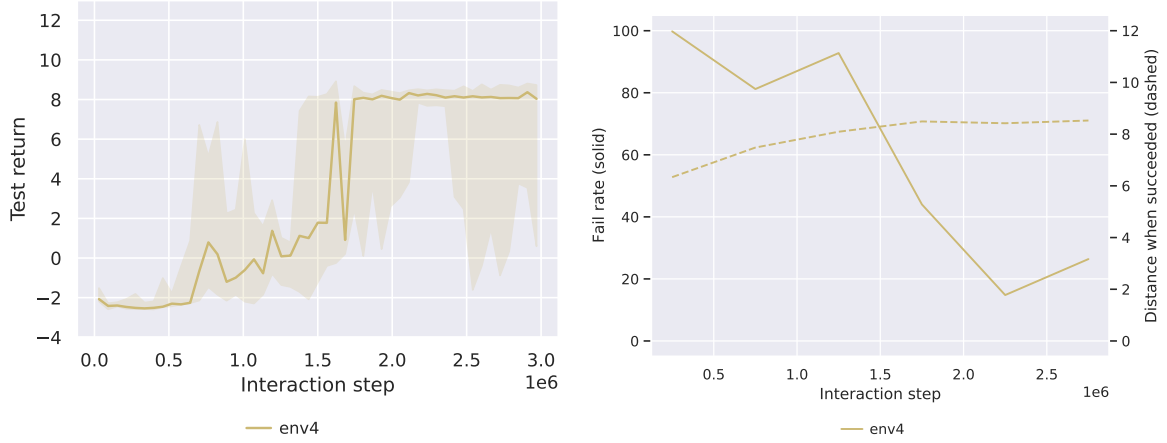
lack of improvement in the average return for environment4 was not due to the robot’s inability to learn sufficiently fast gaits but rather its failure to reduce the frequency of failures. As described in Appendix A.4, the same plots for experiments with hyperparameter adjustments specific to environment4 are shown in Figure A.5b. These results confirm that, with appropriate hyperparameter tuning, the robot successfully learned to avoid failures and achieve faster forward progression.

A.4 Supplementary Discussion on Hyperparameters

Table A.1 presents the hyperparameter settings used in the experiments described in Section 2.4. The implementation generally followed the original SAC algorithm [Haarnoja et al., 2018], with certain modifications, such as separate learning rates for the mean and variance.

While these hyperparameters provided satisfactory results in most environments, their performance in environment 4—a slippery setting with a friction coefficient of 0.5—was suboptimal. As shown in Figure 2.6a, the variation in test profiles for environment 4 was substantial, and the robot failed to achieve stable gaits even after one million training steps (Fig. A.3). Although the test return increased gradually, it plateaued at a middle level after approximately two million steps.

Further investigation into this middle-level performance revealed distinct trends. In one out of the five training runs, the robot achieved relatively effective control, but occasional failures were observed, likely due to the initial state configuration. In these cases, the robot would deviate from the aisle once per approximately 50 test episodes, resulting in penalties. The remaining four training runs displayed oscillatory policy learning, fluctuating between successful control—where the robot completed 1,000 steps without falling—and suboptimal control, where the robot frequently fell, often at the aisle’s edges.



(a) Sample efficiency curve in environment 4. (b) Failure rate and the progression distance when succeeded in environment 4.

Figure A.5: Learning curves in environment 4 with tuned hyperparameters.

To address this instability, hyperparameters were specifically tuned for environment 4. As illustrated in Figure A.5a, this adjustment allowed the robot to acquire stable gait patterns in approximately 2.5 million steps. Figure A.5b further demonstrates the learning progress, showing significant reductions in the failure rate when hyperparameters were optimized for this environment. Furthermore, under the tuned hyperparameters for environment 4, the same metric as in Table 2.3 improved to 6.14, compared to the result with the common hyperparameters.

While the default hyperparameter settings performed well across diverse environments, the unique challenges of environment 4 required targeted adjustments to ensure successful policy learning.

A.5 Deep Transition

As a baseline method In this study, we selected the approach presented by [Bellegarda and Ijspeert, 2022], utilizing the open-source implementation of Deep Transition [Shafiee et al., 2024]. The original Deep Transition paper employed a quadruped robot model of a similar size to the Rex model, adapting a single policy network model across multiple environments with varying physical parameters. For the comparison in the experiments, we adapted the CPG and PPO implementations from the Deep Transition codebase and applied them to the control task of the Rex model under the same problem setting as our proposed method. Specifically, we retained the original architecture of the policy and critic networks, with the neural network parameters numbering 183,692 and 182,273, respectively. This implementation is referred to as Deep Transition (PPO) in our experiments. When conducting experiments using the

default hyperparameters, the learning process was found to be slow. To address this issue, a hyperparameter search was performed on the mini-batch size, learning rate, and target KL divergence of the PPO algorithm.

On the other hand, for Deep Transition (SAC), we utilized the SAC algorithm implementation and network architecture described in Section 2.4.3. The numbers of the actor and critic model parameters were 80,920 and 156,162, respectively.

In both PPO and SAC implementations, the state inputs to the actor and critic included 18 quantities representing the robot’s state, consistent with our proposed method, augmented with 16-dimensional quantities summarizing the CPG state.

Appendix B

Appendix of Chapter 4

B.1 Toy Problem: Policy Evaluation on Random Tabular MDPs

This section provides details of the experiments conducted on a Random MDP as described in Section 4.1 to evaluate the importance of distribution rollouts.

For the experiment, we defined an MDP with $S = |\mathcal{S}|$ states and $A = |\mathcal{A}|$ actions. For each MDP, a random pair of transition matrix $T := \{T_{ijk}\} \in [0, 1]^{(S \times A \times S)}$, where $\sum_k T_{ijk} = 1$, and reward function $R := \{R_{ij}\} \in [0, 1]^{(S \times A)}$ was generated. The initial state was set as $s_0 = 0$ for all T , and the discount factor was set to $\gamma = 1 - \frac{1}{S}$. This defined the MDP task (T, R) , on which we applied the PSRL algorithm [Strens, 2000] for online reinforcement learning over E episodes.

From the observed trajectories up to episode E , we randomly selected a subsequence and defined it as ρ_b . Using the value function $V := \{V_i\} \in \mathbb{R}^S$ estimated by the PSRL algorithm at episode E , we computed a noisy value function estimate $\hat{V}_\eta$. The noise $n \sim \mathcal{N}(0, 1)$ was scaled by the weight vector w , defined to reduce the added noise in proportion to how frequently state i was visited in ρ_b . The noisy value function estimate was calculated as:

$$\hat{V}_\eta = V + \eta n \sum_i |V_i| w,$$

where η represents the noise intensity.

Using the dynamics model T_ψ and the policy π obtained at episode E , we computed the state distribution $(T_\psi^\pi)^{H_d} \rho_b$ after H_d rollout steps. The H_t -step MVE estimate was defined as:

$$\hat{V}_{\hat{V}_\eta, \pi, T_\psi}(s; H_t) := \sum_{t=0}^{H_t-1} \gamma^t \hat{r}(s_t, a_t) + \gamma^{H_t} \hat{V}_\eta(\hat{s}_{H_t}),$$

where $\hat{V}_\eta$ corresponds to the $H_t = 0$ case ($\hat{V}_{\hat{V}_\eta, \pi, T_\psi}(s; 0) = \hat{V}_\eta$).

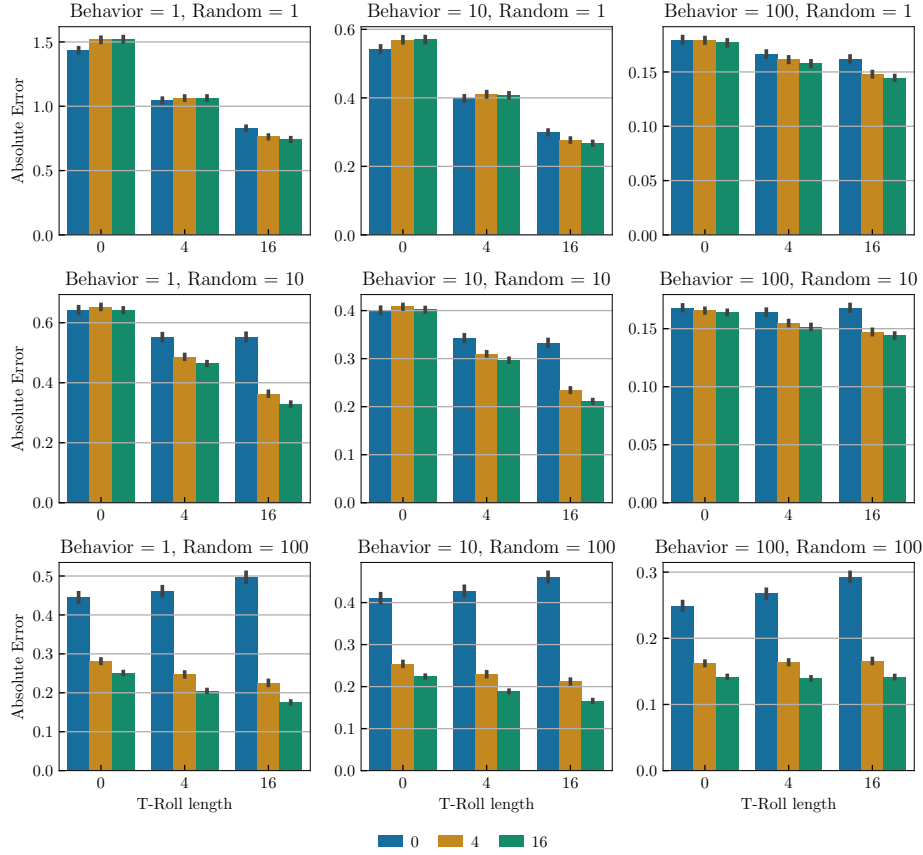


Figure B.1: Ratio of root mean squared error for the MVE estimator compared to the target critic, evaluated for varying size of MDP (the first value in the title) and levels of bias (the second value in the title), training rollout lengths (X-axis), and distribution rollout lengths (hue). This is full results shown in Figure 4.1.

For each experiment, we calculated the average values of $\hat{V}_{\hat{V}_{\eta, \pi, T_{\psi}}}$ and $\hat{V}_{\eta}$ over the state distribution $(T_{\psi}^{\pi})^{H_d} \rho_b$, and then evaluated the error.

The ratio of these errors was computed as the main metric.

This procedure was repeated 32 times for each MDP and for 64 randomly generated MDPs. Experiments were conducted for MDPs with state sizes $S = 4, 16, 64$, and multiple settings of η , H_d , and H_t . The resulting box plots for the error ratios are presented in Figure B.1. These results illustrate the impact of distribution rollouts across different configurations.

B.2 Implementation Detail

B.2.1 Neural network architecture

We adopt multi layer perceptron (MLP) to predict next state and reward by ensembles of stochastic generative models $\{T_{\psi}^{(1)}, \dots, T_{\psi}^{(M)}\}$. We use 8 members of the Gaus-

Algorithm 5 Double Horizon Model-Based Policy Optimization

Input: Models T_ψ , actor network π_θ and critic network Q_ϕ , target critic network $Q_{\bar{\phi}}$, replay buffer $\mathcal{D}_e$, rollout buffer $\mathcal{D}_r$

for $e = 0$ to $E - 1$ **do**

Interact with the environment with π_θ by T step

Add observed transitions to $\mathcal{D}_e$

Fit model T_ψ on uniformly sampled transitions from $\mathcal{D}_e$ until stopped by early stopping

Distribution rollout

Sample states from $\mathcal{D}_e$ and add sampled states from H_d -step distribution rollout to $\mathcal{D}_r$

Policy optimization

for $i = 0$ to $T - 1$ **do**

for $j = 0$ to $B - 1$ **do**

Training rollout

Sample states s_j from $\mathcal{D}_r$ and imagine transitions $\tau_j = (s_0 = s_j, a_0, r_0, s_1, a_1, r_1, \dots, s_{H_t}, a_{H_t})$

$\hat{Q}_{\bar{\phi}, \theta, \psi}(\tau_j) = r_0 + \sum_{t=1}^{H_t-1} \gamma^t \{r_t - \alpha \log \pi_\theta(a_t | s_t)\} + \gamma^{H_t} \{Q_{\bar{\phi}}(s_{H_t}, a_{H_t}) - \alpha \log \pi(a_{H_t} | s_{H_t})\}$

$\hat{V}_{\phi, \theta, \psi}(\tau_j) = \sum_{t=0}^{H_t-1} \gamma^t \{r_t - \alpha \log \pi_\theta(a_t | s_t)\} + \gamma^{H_t} \{Q_\phi(s_{H_t}, a_{H_t}) - \alpha \log \pi(a_{H_t} | s_{H_t})\}$

$g_\phi = \frac{1}{B} \sum_{j=0}^{B-1} \nabla_\phi (Q_\phi(s_j, a_j) - \hat{Q}_{\bar{\phi}, \theta, \psi}(\tau_j))^2$

$g_\theta = \frac{1}{B} \sum_{j=0}^{B-1} \nabla_\theta \hat{V}_{\phi, \theta, \psi}(\tau_j)$

Update ϕ with g_ϕ and θ with g_θ by SGD optimizers, respectively

Update target critic's parameter: $\bar{\phi} \leftarrow \phi\lambda + \bar{\phi}(1 - \lambda)$

sian model, across all environment and tasks in experiments. Each member m of the ensemble outputs mean and diagonal covariance as parameters of Gaussian distribution: $T_\psi(s' | s, a) := \mathcal{N}(\mu_\psi^{(j)}(s, a), \sigma_\psi^{(j)2}(s, a))$. Individual probabilistic models capture aleatoric noise. When predicting the next state and reward, TS-1 algorithm, a bootstrapping procedure by randomly selecting the member out of ensemble for each inputs and time steps, is used as uncertainty propagation proposed in [Chua et al., 2018]. Additionally we adopt Layer normalization [Ba et al., 2016] and Dropout [Srivastava et al., 2014] for regularization and SiLU [Elfwing et al., 2018] as activation function. We also use ensemble of MLP as critic networks used for randomized ensemble double Q-learning, or RED-Q [Chen et al., 2020]. We summarize the neural network architecture using Pytorch-like notation:

```

(model): Sequential(
  (0): EnsembleLinearLayer(
    8, in_size=S+A, out_size=256, decay=0.00025)
  (1): Dropout(p=0.0075)
  (2): SiLU()
  (3): LayerNorm((256,))
  (4): EnsembleLinearLayer(
    8, in_size=256, out_size=256, decay=0.0005)
  (5): Dropout(p=0.005)
  (6): SiLU()
  (7): LayerNorm((256,))
  (8): EnsembleLinearLayer(
    8, in_size=256, out_size=256, decay=0.00075)
  (9): Dropout(p=0.0025)
  (10): SiLU()
  (11): LayerNorm((256,))
  (12): EnsembleLinearLayer(
    8, in_size=256, out_size=S+1, decay=0.001)
)
)
for model,
(critics): Sequential(
  (0): EnsembleLinearLayer(
    5, in_size=S+A, out_size=512, decay=0.0)
  (1) Dropout(p=0.0001)
  (2): SiLU()
  (3) LayerNorm((256,))
  (4): EnsembleLinearLayer(
    5, in_size=512, out_size=512, decay=0.0)
  (5) Dropout(p=0.0001)
  (6): SiLU()
  (7) LayerNorm((256,))
  (8): EnsembleLinearLayer(
    5, in_size=512, out_size=512, decay=0.0)
  (9) Dropout(p=0.0001)
  (10): SiLU()
  (11) LayerNorm((256,))
  (12): EnsembleLinearLayer(
    5, in_size=512, out_size=1, decay=0.0)
)
)
for critic networks, and
(actor): Sequential(
  (0): Linear(in_features=S, out_features=512)
  (1): SiLU()
  (2): Linear(in_features=512, out_features=512)
  (3): SiLU()
  (4): Linear(in_features=512, out_features=512)
  (5): SiLU()
  (6): Linear(in_features=512, out_features=2A)
)
)

```

for actor networks, respectively, where S is the dimension of state space, and A is the dimension of action space. “EnsembleLinearLayer” denotes the layer which calculates the feed forward pass in parallel. The total number of weight parameters, for example on Humanoid task from GYM suite is about 2M, where $S=45$ and $A=17$.

B.2.2 Model Learning

In this study, we adopt ensemble of Gaussian distribution model with a diagonal covariance matrix. That is, the m th member model outputs $\mu_m, \sigma_m \in \mathcal{R}^{S+1}$, mean parameter for each dimension and the diagonal components of the covariance matrix, respectively. μ_m is a MLP which takes an input, while σ_m is a constant vector to model homoscedastic and aleatric noise. Thus, the objective of the model learning is minimizing the sum of the negative log-likelihood:

$$L(\psi) := \frac{1}{MND} \sum_{m=1}^M \sum_{n=1}^N w_m L_{mn}$$

where

$$L_{mn} := \sum_{i=1}^D \left\{ \log \sigma_{mi} + \frac{1}{2} \left(\frac{\bar{t}_{ni} - \mu_{mi}(x_n)}{\sigma_{mi}} \right)^2 \right\}$$

and $\mu_{mi}(x_n)$ represents the i th output of the neural network μ_m for the n th input $x_n := (s_n, a_n)$, and σ_{mi} denotes the noise regarding the i th dimension output of the m th ensemble member σ_m .

B.2.3 Early Stopping

When x_n is the input, let μ_{mn} and σ_m represent the mean and scale parameter, of the m th model's predicted Gaussian distribution. We aggregate the predictions across all ensemble members, and define the prediction of the DE-model as a Gaussian distribution with mean parameters for the d th dimension when the input is x_n as $\tilde{\mu}_{nd} := \frac{1}{M} \sum_{m=1}^M \mu_{mnd}$ and the scale parameter as $\tilde{\sigma}_{nd}^2 := s_{nd}^2 + \frac{1}{M} \sum_{m=1}^M \sigma_{md}^2$, where s_{nd}^2 is the sample variance among the ensemble of predictions $\{\mu_{mnd}\}_{m=1}^M$. This approach is often also called moment matching.

Then, we define the evaluation score S_n for the n th data as:

$$S_n = \frac{1}{2} \sum_{d=1}^D \left[\frac{(\tilde{\mu}_{nd} - t_{nd})^2}{\tilde{\sigma}_{nd}^2} + \log \tilde{\sigma}_{nd}^2 \right]$$

under the assumption that the predicted distribution follows a Gaussian distribution with a diagonal covariance matrix. After each epoch of model learning, we evaluate the score for each validation data point and compare the distribution of these scores based on the new parameters of the model with those from the previous parameters. If the distribution of the new scores shows significant improvement according to a Z-test with a significance level of 0.1, we continue training the model.

Our stopping strategy is the same as in the MBPO implementation: if the new

Table B.1: Hyper-parameters for DHMBPO used across all experiments in this study

Hyper-parameter	Value
Discount factor	0.995
Seed steps	5000
Action repeat	1 (Gym) 2 (DMControl)
Batch size	256
Update-to-data ratio	1
Replay buffer size	1M
Learning rate for the actor, critics and α	$3 \cdot 10^{-4}$
Initial value of α	0.1
Momentum coefficient c for target critic	0.995
Length of distribution rollout H_d	20
Length of training rollout H_t	5
Iteration per distribution rollout	20
Ensemble size of model	8
Optimizer for training model	AdamW [Loshchilov and Hutter, 2019]
Learning rate training model	$1 \cdot 10^{-3}$

model does not show improvement over a fixed number of epochs, the training is stopped. We set the threshold as $b \log d_s$, where b is a scalar-valued hyper-parameter, representing the base number of epochs, and d_s is the dimension of the state space.

B.3 Experiment Detail

We summarize in Table B.1 hyper-parameters shared for all experiments in Section 4.4.

Additionally we describe configuration for comparison in Section 4.4.3. In this study, when experimental results under identical settings were available, we used those results as part of the results (MBPO and Dreamer v3). For the rest of the results, we ran the publicly available source code with its default hyperparameter settings. For each GYM tasks, except for DHMBPO, each algorithms' hyper-parameters were tuned per-task.

For DMC tasks, Each of three algorithms' hyper-parameters were set commonly over all DMC tasks. We reported TD-MPC2's results obtained by executing the author's open-sourced code without TD-MPC2's MPC option for fair comparison to other algorithms, which does not use on-line planning.

B.4 Result Detail

In Figure B.2 we show plots for runtime of each algorithms on GYM tasks.

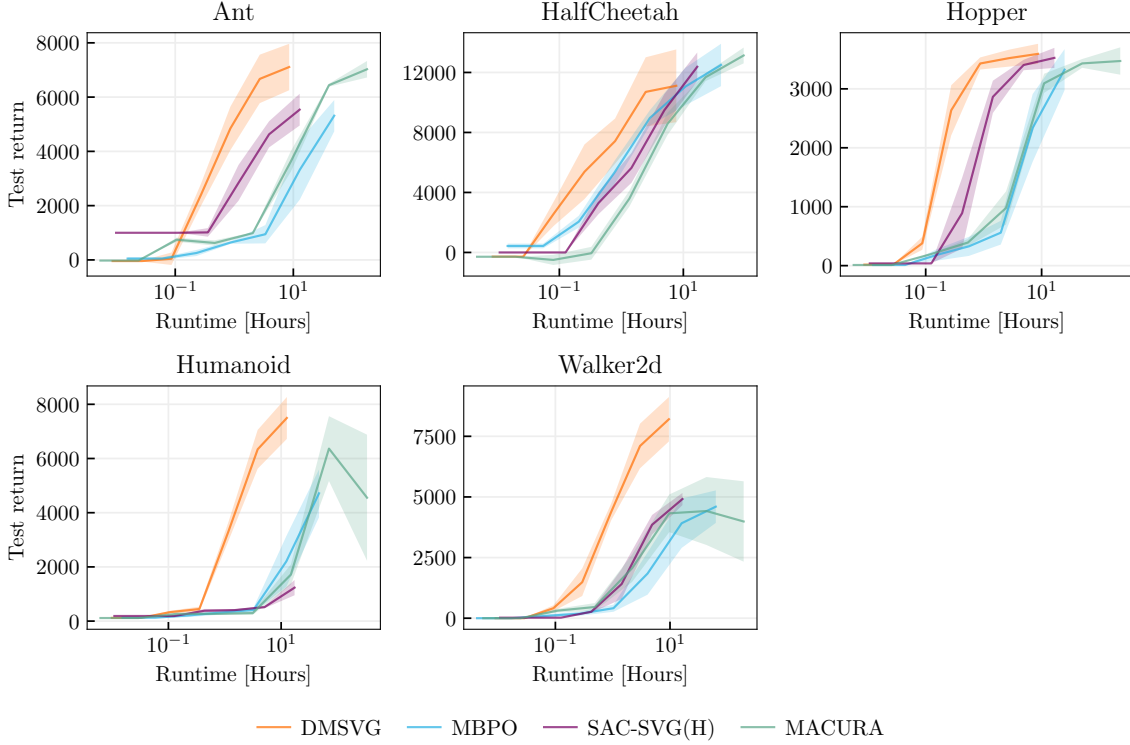


Figure B.2: Runtime comparison on GYM tasks.

we show individual sample efficiency curve for DMC tasks in Figure 4.5, and plots for runtime on DMC tasks in Figure B.4.

B.5 Additional Experiments for Discussion

B.5.1 Detailed Results of Hyper-parameter Sensitivity Analysis

In Figure B.5, we show per-task results of experiments in Section 4.4.2 with respect to sample efficiency (a) and to an Euclidean norm of policy gradient $\theta \in \mathcal{R}^D$: $\sqrt{\sum_i^D \theta_i^2}$.

While the variant with short TR (e.g., TR=1) was less sample-efficient than the baseline on almost all tasks, the variant with long TR (e.g., TR=7) showed slightly better sample efficiency on all but for the “finger-turn_hard” task (Figure B.5a). This is because taking gradient along with long prediction of transitions can cause large variance in estimating the gradient [Parmas et al., 2018], confirmed in Figure B.5b that the runs on the “finger-turn_hard” accompanied with explosion of policy gradient.

B.5.2 Measurements during Training Rollout

We conducted additional experiments to investigate the relative relationship and effectiveness of the lengths of distribution rollouts and training rollouts. For this analysis, we utilized a pre-trained DHMBPO agent that had been trained for 100K steps on each GYM task.

Using the replay buffer, we randomly selected initial states as inputs for training rollouts of various lengths $H_t = 1, \dots, 20$. For each H_t -step training rollout, we measured two evaluation metrics to assess its performance. This process was repeated $R = 1024$ times for statistical robustness.

These experiments provide insights into the interplay between rollout lengths and their impact on performance, shedding light on how the choice of H_t affects the overall behavior of the agent.

Figure B.6 shows the plots for two metrics at each training rollout metrics. One metric is uncertainty score, particularly geometric Jensen-Shannon divergence which are adopted in [Frauenknecht et al., 2024] to evaluate whether the predicted state is probably in-distribution of training data. Second metric is the signal-to-noise ratio (SNR) of the gradients of actor loss with regard to actor’s weights:

$$SNR = \sqrt{\frac{\sum_{r=1}^R \sum_{d=1}^D g_{rd}^2 / R}{\sum_{d=1}^D \mathbb{V}_r [g_{rd}]}} ,$$

where, g_{rd} is d th dimension of the gradient of the MVE estimator about r th batch inputs, and D is the dimension of the actor’s network weights and “ $\mathbb{V}_r$ ” denotes sample variance over R -times repeated values. If the ratio is relatively large, it is indicated that the calculated gradient is more reliable estimation, while we want to avoid to use gradients with small SNR.

From the plots we can see that, as the rollout horizon proceeds, predicted states are extended to unseen region in real the environment, while the gradient estimation becomes unreliable. For all tasks the SNR is about 0.1 at $H_t = 5$ while it fallen to almost 0 at H_t .

On the other hand, the uncertainty score seems to be relatively stable so we may use long horizon such as $H_t = 20$ just for input states, but not for calculation of the gradient due to aforementioned reason.

B.5.3 Application of Distribution Rollout to SAC-SVG(H) Implementation

We applied distribution matching to the SAC-SVG(H) algorithm [Amos et al., 2021], which employs the SVG method with a single GRU [Cho, 2014] as the recurrent neural network and one-step TD errors for training the critic networks. We refer to this variant as DMSAC-SVG(H) and evaluated it on five GYM tasks with varying distribution rollout lengths. As shown in Fig. B.7, the variants with longer distribution rollouts exhibited either poor learning performance or a drop in performance after a certain point. We investigated the training metrics and found that the critic losses started to explode as the distribution rollout length increased, as shown in Fig. B.8.

One potential cause of these results is the deterministic nature of the model used in the SAC-SVG(H) algorithm. Predictions generated by a fixed deterministic model are likely to introduce consistent biases. In contrast, the Deep Ensemble (DE) approach adopted by DHMBPO and MBPO uses multiple ensemble members, each potentially having similar biases. However, the manner in which these biases accumulate is random across ensemble members. During predictions, an ensemble member is randomly selected at each timestep, making the bias accumulation in the predicted states also random. As a result, within a single trajectory, these biases may cancel out to some extent, leading to an overall smaller average bias, albeit with larger variance.

Policy evaluation and improvement based on the higher-variance predictions of a DE model may slow down learning, but they avoid the risk of extreme predictions caused by deterministic models with large biases. On the other hand, policy evaluation and optimization using predictions with significant bias from deterministic models carry the risk of focusing on extreme values, potentially destabilizing the training process.

B.5.4 Comparison of Model Learning Performance

We also conducted a quantitative analysis to compare the predictive performance of models used in DHMBPO, SAC-SVG(H), and MBPO [Pineda et al., 2021], using common training and evaluation datasets.

Data Generation Procedure For each GYM task, we ran the DHMBPO algorithm for up to 500K steps. From the replay buffer, we created three input-output datasets containing data collected up to 5K, 20K, and 50K steps, respectively. Additionally, we created another dataset from all data collected between 495K and 500K steps. Each dataset was split into training (80%) and validation (20%) subsets, while the 495K–500K dataset was exclusively used for evaluation.

Evaluation Methodology For each of the three models, we trained them using the respective training datasets and measured RMSE loss on the training, validation, and evaluation datasets at each epoch. This procedure was repeated for three random seeds.

Results The results of the experiments are plotted separately for each dataset size, comparing training data versus validation data and training data versus evaluation data, as shown in Figs. B.9– B.11. Additionally, the results after 100K epochs are summarized in Table B.2. The shaded areas in the plots and the $\pm$ values in the table both represent the standard error over three seeds.

Table B.2: Summary of RMSE values after 100K epochs.

Metric	Task	Method Size	DHMBPO	MBPO	SAC-SVG(H)
Training RMSE	Ant	5000	0.29 ± 0.0	1.86 ± 0.02	1.1 ± 0.01
		20000	0.48 ± 0.0	1.21 ± 0.01	0.67 ± 0.0
		50000	0.5 ± 0.0	0.79 ± 0.02	0.59 ± 0.0
	HalfCheetah	5000	0.18 ± 0.0	1.46 ± 0.01	0.7 ± 0.01
		20000	0.35 ± 0.0	0.73 ± 0.01	0.57 ± 0.0
		50000	0.49 ± 0.0	1.58 ± 0.08	0.76 ± 0.0
	Hopper	5000	0.01 ± 0.0	0.06 ± 0.0	0.11 ± 0.0
		20000	0.02 ± 0.0	0.09 ± 0.0	0.18 ± 0.0
		50000	0.02 ± 0.0	0.04 ± 0.0	0.21 ± 0.0
	Humanoid	5000	0.44 ± 0.0	2.02 ± 0.0	2.99 ± 0.02
		20000	1.18 ± 0.0	1.99 ± 0.0	1.65 ± 0.0
		50000	1.55 ± 0.0	1.77 ± 0.01	1.69 ± 0.0
	Walker2d	5000	0.16 ± 0.0	1.74 ± 0.08	2.88 ± 0.01
		20000	0.36 ± 0.0	1.1 ± 0.01	1.48 ± 0.03
		50000	0.43 ± 0.0	0.72 ± 0.02	0.92 ± 0.0
Validation RMSE	Ant	5000	6.44 ± 0.03	4.29 ± 0.12	8.59 ± 0.07
		20000	1.51 ± 0.02	1.62 ± 0.02	3.64 ± 0.03
		50000	0.93 ± 0.01	1.07 ± 0.01	1.43 ± 0.01
	HalfCheetah	5000	3.72 ± 0.01	3.82 ± 0.03	8.78 ± 0.01
		20000	1.99 ± 0.02	1.87 ± 0.03	3.16 ± 0.05
		50000	1.61 ± 0.03	1.97 ± 0.11	3.9 ± 0.14
	Hopper	5000	0.19 ± 0.01	0.16 ± 0.01	0.25 ± 0.0
		20000	0.25 ± 0.02	0.2 ± 0.0	0.95 ± 0.03
		50000	0.13 ± 0.01	0.1 ± 0.01	1.15 ± 0.02
	Humanoid	5000	5.14 ± 0.06	5.09 ± 0.02	7.9 ± 0.1
		20000	3.25 ± 0.04	3.49 ± 0.02	9.25 ± 0.08
		50000	2.66 ± 0.01	2.52 ± 0.02	6.51 ± 0.0
	Walker2d	5000	4.08 ± 0.06	4.54 ± 0.16	6.04 ± 0.04
		20000	2.69 ± 0.02	2.79 ± 0.08	7.22 ± 0.06
		50000	1.46 ± 0.02	1.54 ± 0.03	2.49 ± 0.02
Evaluation RMSE	Ant	5000	8.61 ± 0.21	6.45 ± 0.18	10.9 ± 0.22
		20000	2.49 ± 0.06	2.05 ± 0.05	6.29 ± 0.11
		50000	1.13 ± 0.01	1.25 ± 0.01	2.61 ± 0.02
	HalfCheetah	5000	21.11 ± 0.24	274.94 ± 51.26	25.31 ± 1.15
		20000	12.76 ± 0.19	18.12 ± 1.23	16.56 ± 0.35
		50000	5.89 ± 0.06	27.91 ± 4.62	9.51 ± 0.23
	Hopper	5000	1.97 ± 0.01	20.67 ± 1.34	2.05 ± 0.03
		20000	0.99 ± 0.04	1.47 ± 0.19	1.61 ± 0.13
		50000	0.4 ± 0.0	0.62 ± 0.16	0.88 ± 0.01
	Humanoid	5000	12.91 ± 0.18	14.25 ± 0.54	17.22 ± 0.69
		20000	12.64 ± 0.21	14.63 ± 0.19	13.24 ± 0.46
		50000	11.47 ± 0.18	11.46 ± 0.17	12.86 ± 0.14
	Walker2d	5000	11.27 ± 0.28	251.58 ± 54.54	13.91 ± 0.59
		20000	11.37 ± 0.49	16.42 ± 2.58	12.38 ± 0.76
		50000	8.89 ± 0.18	22.69 ± 10.32	9.72 ± 0.3

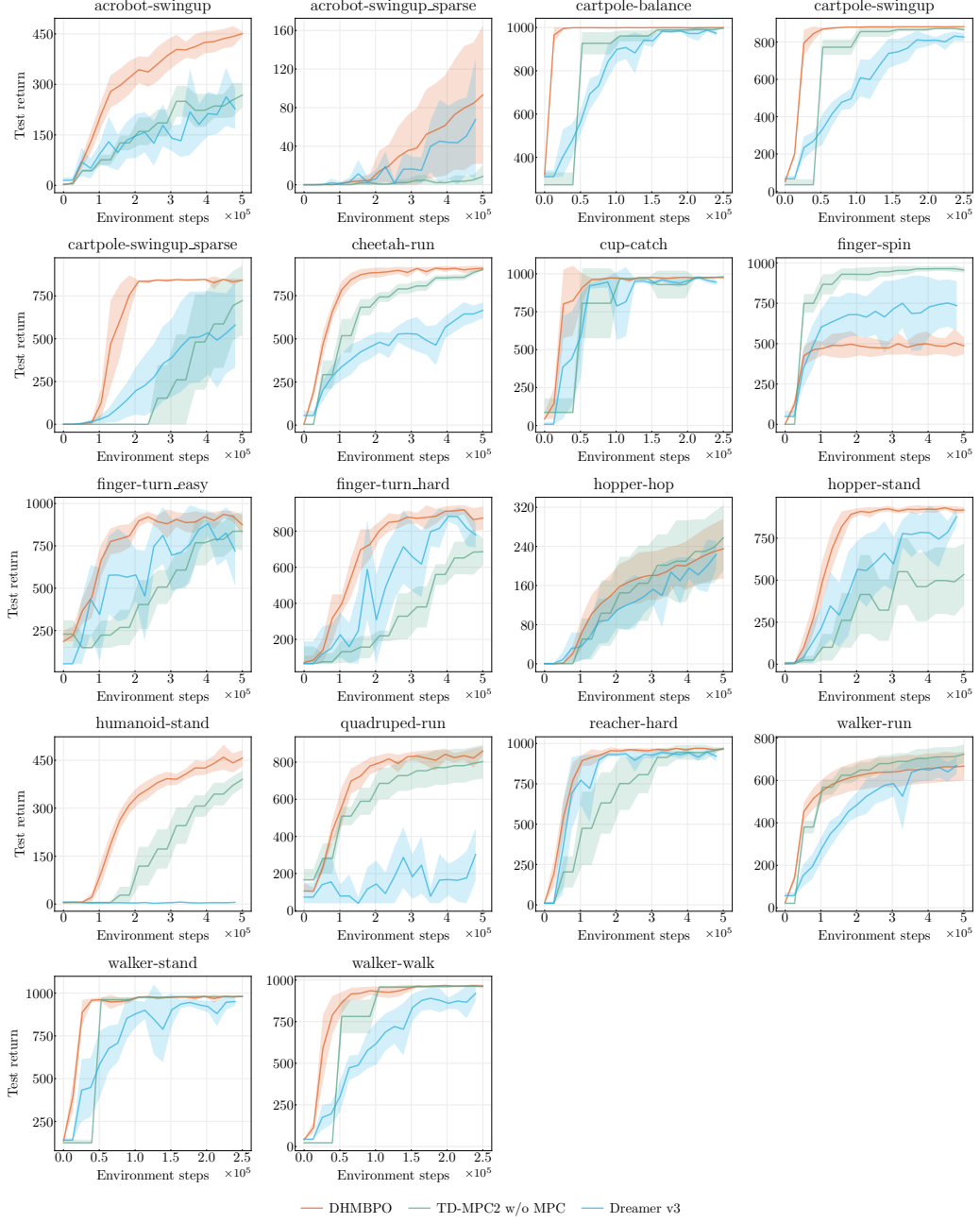


Figure B.3: Comparison of sample efficiency with latent model-based methods on DMC tasks. The solid line represents the sample mean over eight random seeds, and the shaded area indicates the 95% confidence intervals.

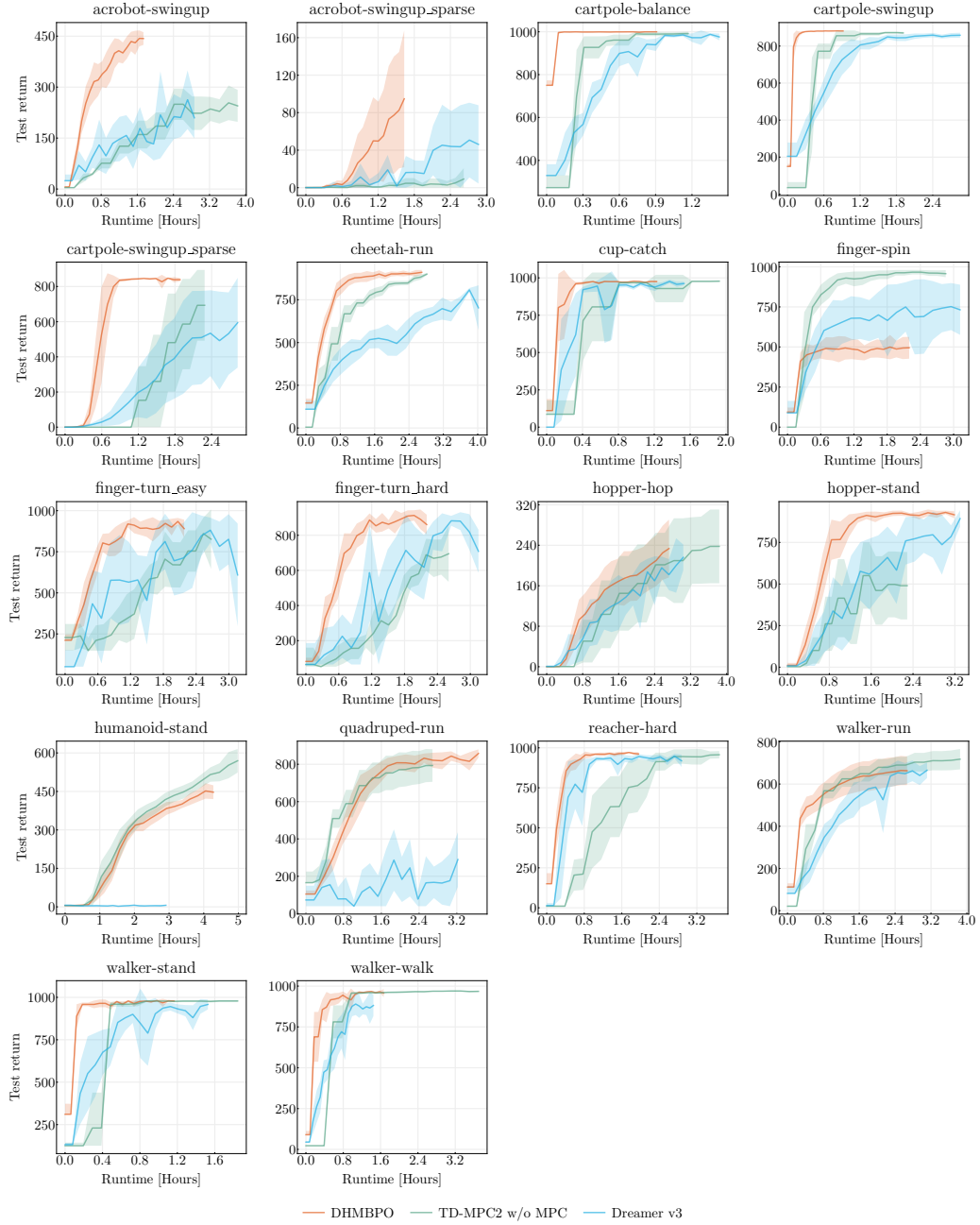
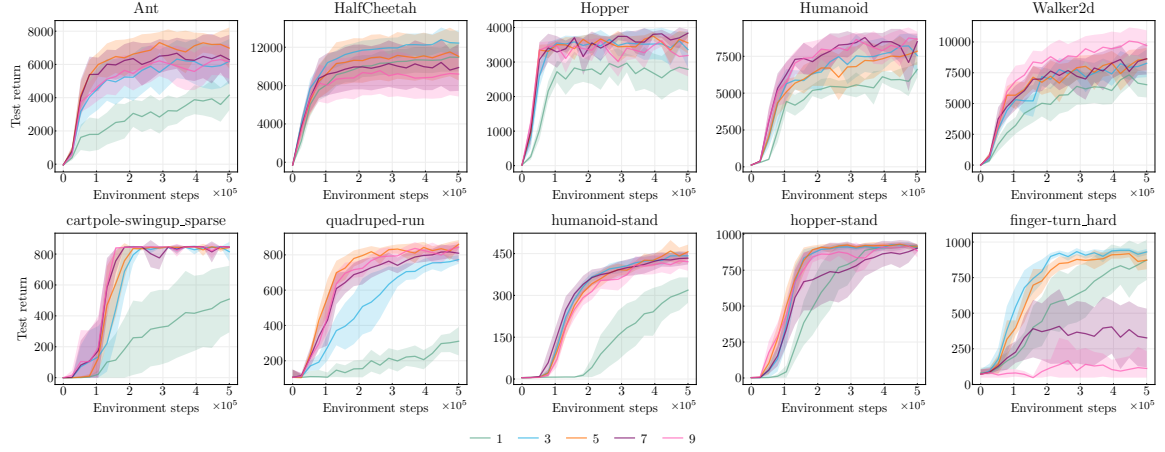
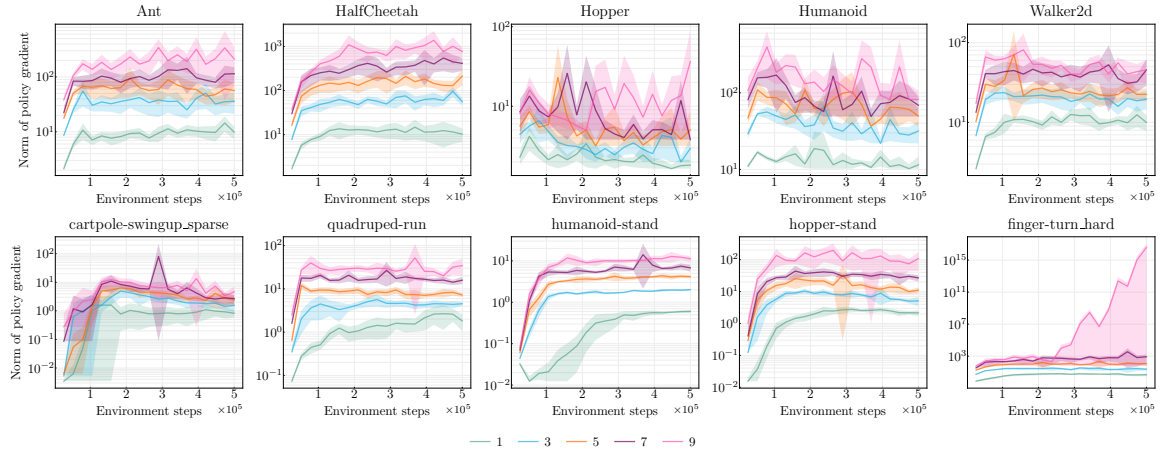


Figure B.4: Runtime comparison with latent model-based methods on DMC tasks.



(a) Sample efficiency curves.



(b) Norm of policy gradient.

Figure B.5: Development of test return (a) and the norm of policy gradient (b), over environment steps.

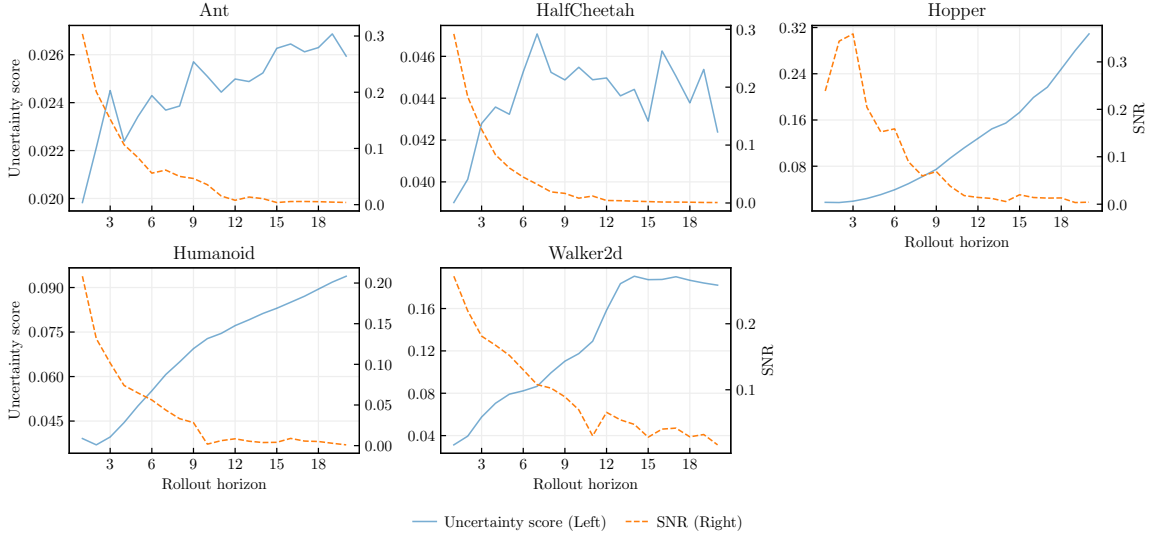


Figure B.6: Measurements at each step during training rollout. Uncertainty score proposed in Frauenknecht et al. [2024] (Left, solid) and Signal-to-noise ratio (SNR) of the gradients of actor loss with regard to actor’s weights (Right, dashed).

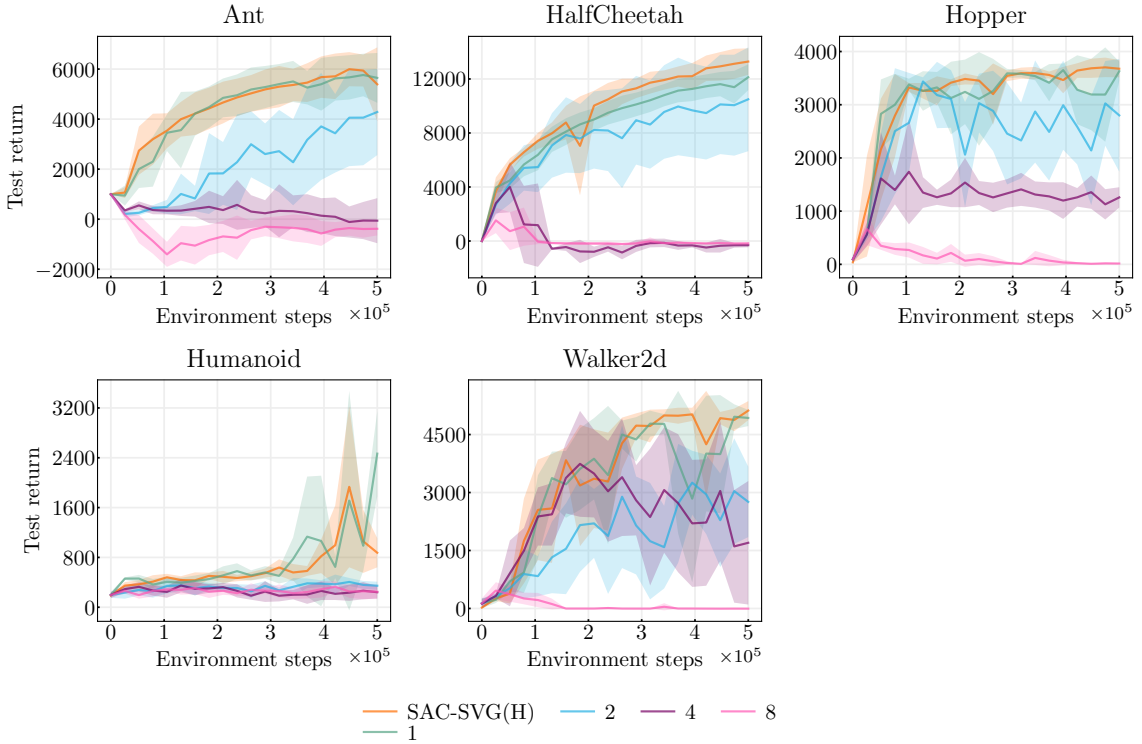


Figure B.7: Sample efficiency curve of DMSAC-SVG(H).

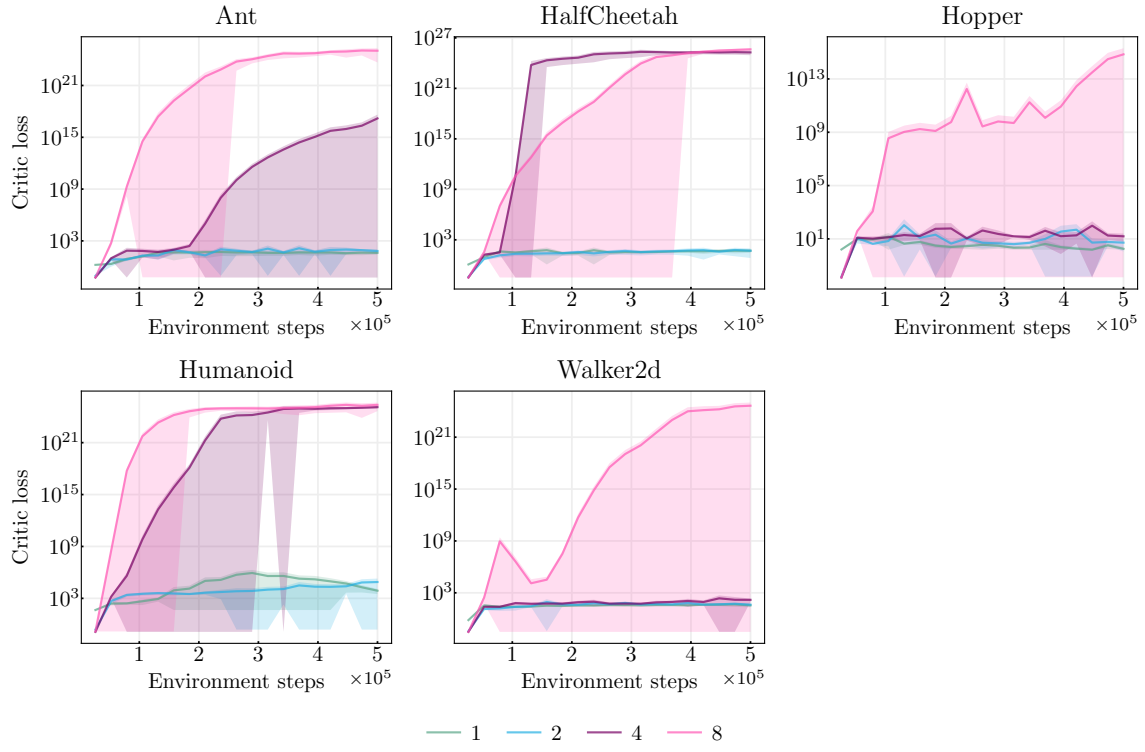
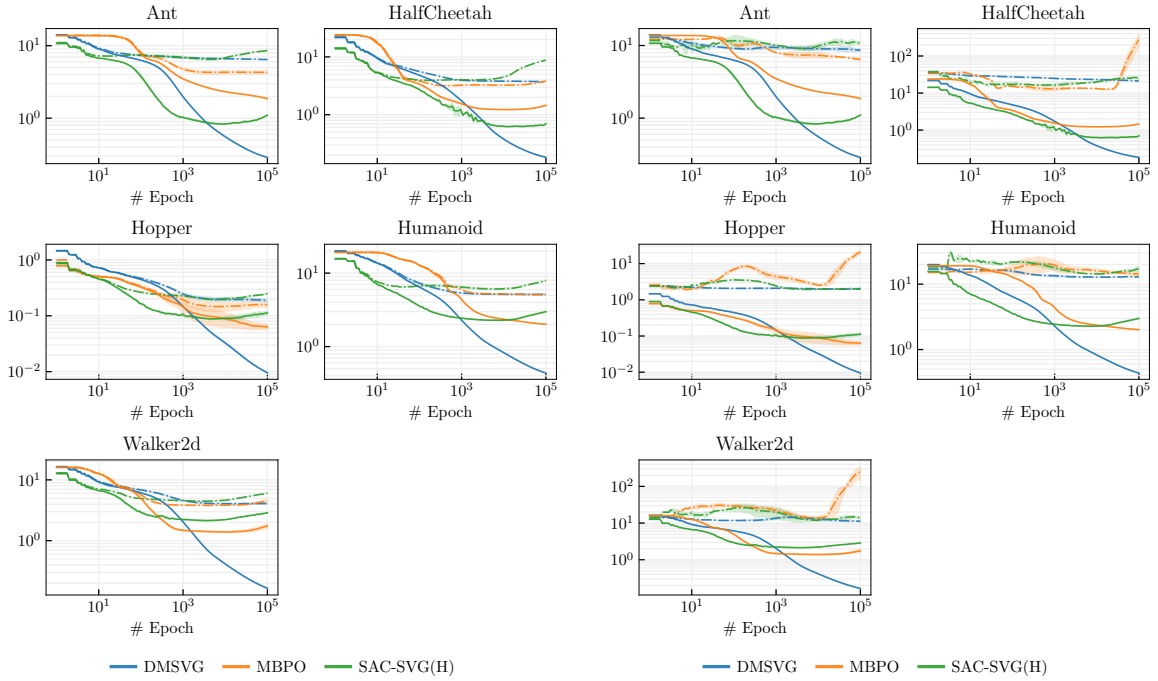


Figure B.8: Development of critic loss over the number of environment steps of DA+SAC-SVG(H).



(a) Training-Validation RMSE

(b) Training-Evaluation

Figure B.9: RMSE of model trained on 5K environment steps dataset. Solid line is for training RMSE and dashdot line is for validation RMSE or evaluation RMSE

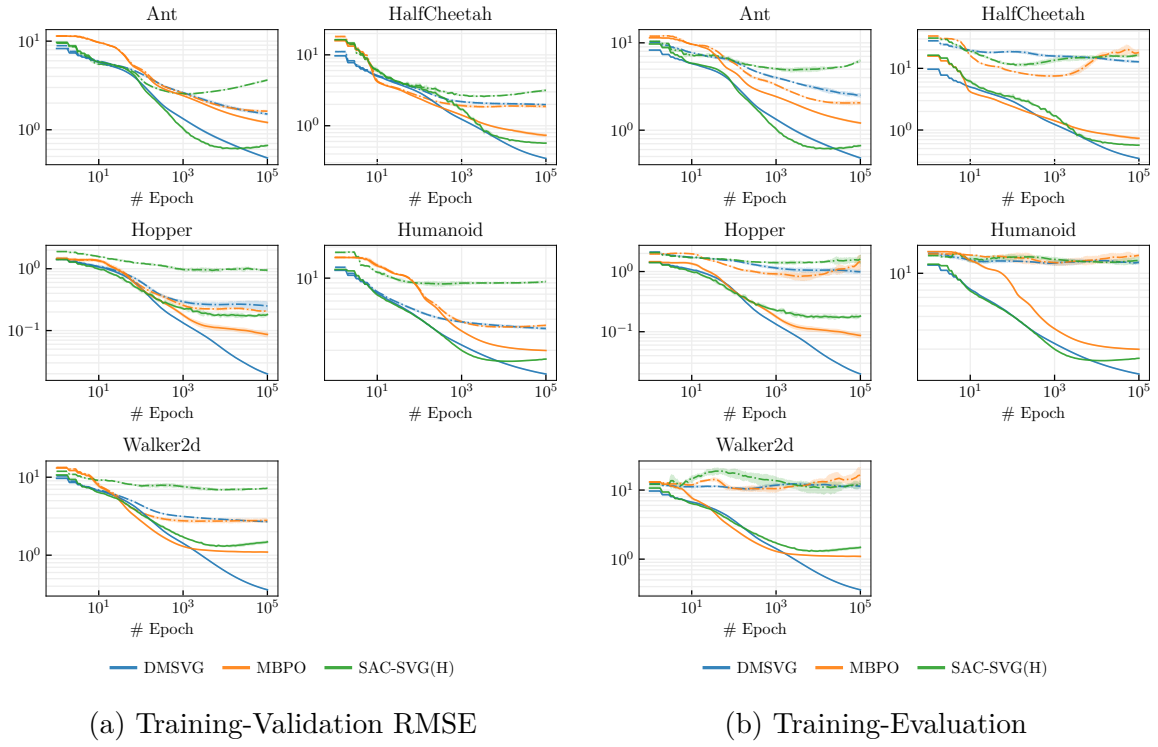


Figure B.10: RMSE of model trained on 20K environment steps dataset. Solid line is for training RMSE and dashdot line is for validation RMSE or evaluation RMSE

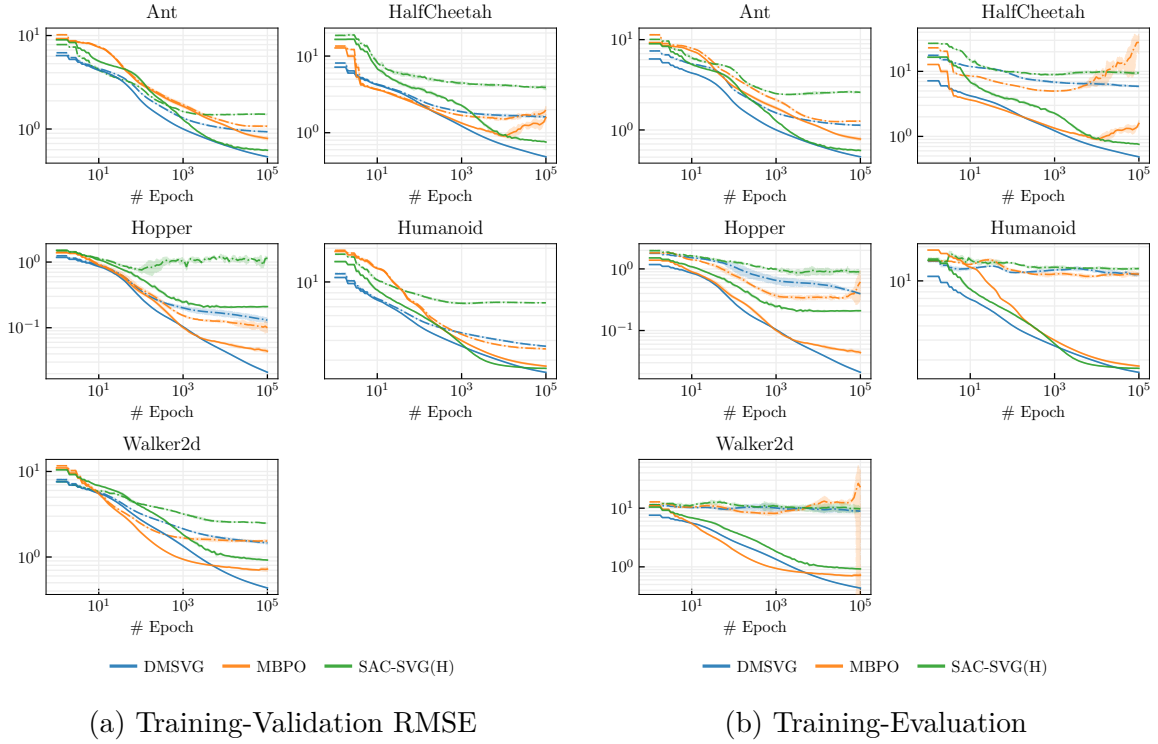


Figure B.11: RMSE of model trained on 50K environment steps dataset. Solid line is for training RMSE and dashdot line is for validation RMSE or evaluation RMSE

Bibliography

In this bibliography, the names of journals and conference proceedings will be abbreviated as follows:

Full Name	Abbreviation
AAAI Conference on Artificial Intelligence	AAAI
Advances in Neural Information Processing Systems	NIPS/NeurIPS
Computer Vision and Pattern Recognition	CVPR
European Conference on Computer Vision	ECCV
International Conference on Artificial Intelligence and Statistics	AISTATS
International Conference on Learning Representations	ICLR
International Conference on Machine Learning	ICML
International Conference on Robotics and Automation	ICRA
International Joint Conference on Artificial Intelligence	IJCAI
Journal of Machine Learning Research	JMLR
Proceedings of Machine Learning Research	PMLR

Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a Posteriori Policy Optimisation. 2018.

Ananye Agarwal, Ashish Kumar, Jitendra Malik, and Deepak Pathak. Legged locomotion in challenging terrains using egocentric vision. In *Conference on robot learning*, pages 403–415. PMLR, 2023.

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 34, 2021.

Brandon Amos, Samuel Stanton, Denis Yarats, and Andrew Gordon Wilson. On the model-based stochastic value gradient for continuous reinforcement learning. In *Learning for Dynamics and Control*, pages 6–20. PMLR, 2021.

Karol Arndt, Murtaza Hazara, Ali Ghadirzadeh, and Ville Kyrki. Meta reinforcement learning for sim-to-real domain adaptation. In *2020 IEEE international conference on robotics and automation (ICRA)*, pages 2725–2731. IEEE, 2020.

- Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.
- Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.
- Guillaume Bellegarda and Auke Ijspeert. CPG-RL: Learning Central Pattern Generators for Quadruped Locomotion. *IEEE Robotics and Automation Letters*, 7(4):12547–12554, 2022.
- Guillaume Bellegarda, Milad Shafiee, and Auke Ijspeert. Visual cpg-rl: Learning central pattern generators for visually-guided quadruped locomotion. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1420–1427. IEEE, 2024.
- Luca Bertinetto, Jack Valmadre, Stuart Golodetz, Ondrej Miksik, and Philip HS Torr. Staple: Complementary learners for real-time tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1401–1409, 2016.
- Adel Bibi, Matthias Mueller, and Bernard Ghanem. Target response adaptation for correlation filter tracking. In *European conference on computer vision*, pages 419–433. Springer, 2016.
- Gerardo Bledt and Sangbae Kim. Extracting Legged Locomotion Heuristics with Regularized Predictive Control. In *2020 IEEE International Conference on Robotics and Automation*, pages 406–412, 2020.
- David S Bolme, J Ross Beveridge, Bruce A Draper, and Yui Man Lui. Visual object tracking using adaptive correlation filters. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2544–2550. IEEE, 2010.
- Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 4243–4250. IEEE, 2018.
- Steven J Bradtke and Andrew G Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1):33–57, 1996.
- Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. 2016.

- Luigi Campanaro, Siddhant Gangapurwala, Daniele De Martini, Wolfgang Merkt, and Ioannis Havoutis. CPG-Actor: Reinforcement Learning for Central Pattern Generators. In *Towards Autonomous Robotic Systems*, pages 25–35. Springer International Publishing, 2021.
- Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8973–8979. IEEE, 2019.
- Xinyue Chen, Che Wang, Zijian Zhou, and Keith W Ross. Randomized Ensembled Double Q-Learning: Learning Fast Without a Model. In *International Conference on Learning Representations*, 2020.
- Kyunghyun Cho. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.
- Janghoon Choi, Junseok Kwon, and Kyoung Mu Lee. Visual Tracking by Reinforced Decision Making. *arXiv preprint arXiv:1702.06291*, 2017.
- Yinlam Chow, Brandon Cui, Moonkyung Ryu, and Mohammad Ghavamzadeh. Variational Model-based Policy Optimization. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence*, pages 2292–2299, 2021.
- Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep Reinforcement Learning in a Handful of Trials Using Probabilistic Dynamics Models. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 4759–4770, 2018.
- Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. Model-Based Reinforcement Learning via Meta-Policy Optimization. In *Proceedings of The 2nd Conference on Robot Learning*, volume 87, pages 617–629, 2018.
- Erwin Coumans and Yunfei Bai. PyBullet, a Python module for physics simulation for games, robotics and machine learning. <http://pybullet.org>, 2016–2019.
- Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

- Martin Danelljan, Gustav Häger, Fahad Khan, and Michael Felsberg. Accurate scale estimation for robust visual tracking. In *British Machine Vision Conference, Nottingham, September 1-5, 2014*. BMVA Press, 2014.
- Martin Danelljan, Gustav Hager, Fahad Shahbaz Khan, and Michael Felsberg. Learning spatially regularized correlation filters for visual tracking. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4310–4318, 2015.
- Martin Danelljan, Gustav Hager, Fahad Shahbaz Khan, and Michael Felsberg. Adaptive Decontamination of the Training Set: A Unified Formulation for Discriminative Visual Tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1430–1438, 2016.
- Martin Danelljan, Gustav Häger, Fahad Shahbaz Khan, and Michael Felsberg. Discriminative scale space tracking. *IEEE transactions on pattern analysis and machine intelligence*, 39(8):1561–1575, 2017.
- Marc Deisenroth and Carl E Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.
- Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.
- Aditya M Deshpande, Eric Hurd, Ali A Minai, and Manish Kumar. DeepCPG Policies for Robot Locomotion. *IEEE Transactions on Cognitive and Developmental Systems*, 15(4):2108–2121, 2023.
- Xingping Dong, Jianbing Shen, Wenguan Wang, Yu Liu, Ling Shao, and Fatih Porikli. Hyperparameter Optimization for Tracking With Continuous Deep Q-Learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 518–527, 2018.
- Pierluca D’Oro, Max Schwarzer, Evgenii Nikishin, Pierre-Luc Bacon, Marc G Bellemare, and Aaron Courville. Sample-Efficient Reinforcement Learning by Breaking the Replay Ratio Barrier. In *The Eleventh International Conference on Learning Representations*, 2023. URL <https://openreview.net/forum?id=OpC-9aBBVJe>.
- Kenji. Doya. Bifurcations in the learning of recurrent neural networks. In *IEEE International Symposium on Circuits and Systems*, volume 6, pages 2777–2780, 1992. doi: 10.1109/ISCAS.1992.230622.

- Markus Eisenbach, Alexander Kolarow, Konrad Schenk, Klaus Debes, and Horst-Michael Gross. View invariant appearance-based person reidentification using fast online feature selection and score level fusion. In *2012 IEEE Ninth International Conference on Advanced Video and Signal-Based Surveillance*, pages 184–190, 2012. doi: 10.1109/AVSS.2012.81.
- Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107: 3–11, 2018.
- Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep policy gradients: A case study on PPO and TRPO. 2020.
- Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I Jordan, Joseph E Gonzalez, and Sergey Levine. Model-based value expansion for efficient model-free reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, 2018.
- Bernd Frauenknecht, Artur Eisele, Devdutt Subhasish, Friedrich Solowjow, and Sebastian Trimpe. Trust the model Where it trusts itself – model-Based Actor-Critic with uncertainty-Aware Rollout Adaption. *arXiv [cs.LG]*, 2024.
- Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax - A Differentiable Physics Engine for Large Scale Rigid Body Simulation. In *NeurIPS Track on Datasets and Benchmarks*, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/hash/d1f491a404d6854880943e5c3cd9ca25-Abstract-round1.html.
- Scott Fujimoto, Herke van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 1587–1596, 2018.
- Yasuhiro Fukuoka and Hiroshi Kimura. Realization of Dynamic Turning Motion on Irregular Terrain of a Quadruped Robot “TEKKEN4” with Neural Oscillators. *Transactions of the Japan Society of Mechanical Engineers Series C*, 72(724):3848–3854, 2006.
- Abdalla Gad, Tasnim Basmaji, Maha Yaghi, Huda Alheeh, Mohammad Alkhedher, and Mohammed Ghazal. Multiple object tracking in robotic applications: Trends and challenges. *Applied Sciences*, 12(19), 2022. ISSN 2076-3417. URL <https://www.mdpi.com/2076-3417/12/19/9408>.

- Hamed Kiani Galoogahi, Terence Sim, and Simon Lucey. Correlation filters with limited boundaries. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 4630–4638. IEEE, 2015.
- Hamed Kiani Galoogahi, Ashton Fagg, and Simon Lucey. Learning Background-Aware Correlation Filters for Visual Tracking. *arXiv preprint arXiv:1703.04590*, 2017.
- Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. In *International Conference on Learning Representations*, 2015. URL <http://arxiv.org/abs/1412.6572>.
- Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems*, 33:21271–21284, 2020.
- H-M Gross, H Boehme, Ch Schroeter, and Ch Martin. Toomas: interactive shopping guide robots in everyday use-final implementation and experiences from long-term field trials. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2005–2012. IEEE, 2009.
- David Ha and Jürgen Schmidhuber. Recurrent World Models Facilitate Policy Evolution. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.
- Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 1352–1361, 2017.
- Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic Algorithms and Applications. 2018.
- Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to Control: Learning Behaviors by Latent Imagination. 2019.
- Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.
- Assaf Hallak and Shie Mannor. Consistent on-line off-policy evaluation. In *International Conference on Machine Learning*, pages 1372–1383. PMLR, 2017.

- Nicklas Hansen, Hao Su, and Xiaolong Wang. TD-MPC2: Scalable, Robust World Models for Continuous Control. In *The Twelfth International Conference on Learning Representations*, 2024.
- Nicklas A Hansen, Hao Su, and Xiaolong Wang. Temporal Difference Learning for Model Predictive Control. In *International Conference on Machine Learning*, pages 8387–8406. PMLR, 2022.
- Tomoki Harada, Yutaka Nakamura, Yoshio Matsumoto, and Hiroshi Ishiguro. Vibration suppression control using a pattern generator for a robot driven by air actuators. In *Proceedings of the 2008 IEEE International Conference on Robotics and Biomimetics*, pages 756–761, 2009.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. pages 1026–1034. IEEE, 2015. ISBN 978-1-4673-8390-5.
- Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. *Advances in neural information processing systems*, 28, 2015.
- João F Henriques, Rui Caseiro, Pedro Martins, and Jorge Batista. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(3):583–596, 2015.
- Donald F Hoyt and C Richard Taylor. Gait and the energetics of locomotion in horses. *Nature*, 292:239–240, 1981.
- Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.
- M Hutter, C Gehring, A Lauber, F Gunther, C D Bellicoso, V Tsounis, P Fankhauser, R Diethelm, S Bachmann, M Bloesch, H Kolvenbach, M Bjelonic, L Isler, and K Meyer. ANYmal - toward legged robots for harsh environments. *Adv. Robot.*, 31(17):918–931, 2017.
- Marco Hutter, Christian Gehring, Dominic Jud, Andreas Lauber, C. Dario Bellicoso, Vassilios Tsounis, Jemin Hwangbo, Karen Bodie, Peter Fankhauser, Michael Bloesch, Remo Diethelm, Samuel Bachmann, Amir Melzer, and Mark Hoepflinger. ANYmal - a highly mobile and dynamic quadrupedal robot. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 38–44, 2016.

- Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019.
- Auke Jan Ijspeert, Alessandro Crespi, Dimitri Ryczko, and Jean-Marie Cabelguen. From swimming to walking with a salamander robot driven by a spinal cord model. *Science*, 315(5817):1416–1420, 2007.
- Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, Dmitry Kalashnikov, Alex Irpan, Julian Ibarz, Sergey Levine, Raia Hadsell, and Konstantinos Bousmalis. Sim-to-Real via Sim-to-Sim: Data-efficient Robotic Grasping via Randomized-to-Canonical Adaptation Networks. 2018.
- Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: model-based policy optimization. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 12519–12530, 2019.
- Junqi Jin, Chengru Song, Han Li, Kun Gai, Jun Wang, and Weinan Zhang. Real-time bidding with multi-agent reinforcement learning in display advertising. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 2193–2201, 2018.
- Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model Based Reinforcement Learning for Atari. 2019.
- Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation. In *Proceedings of The 2nd Conference on Robot Learning*, volume 87, pages 651–673, 2018.
- Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2014.
- Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
- Haruka Kita, Sotetsu Koyamada, Yotaro Yamaguchi, and Shin Ishii. A Simple, Solid, and Reproducible Baseline for Bridge Bidding AI. *IEEE CoG*, 2024. URL <https://doi.org/10.1109/CoG60054.2024.10645547>.

- Jens Kober and Jan Peters. Imitation and reinforcement learning. *IEEE Robotics & Automation Magazine*, 17(2):55–62, 2010.
- Matej Kristan, Jiri Matas, Ales Leonardis, Michael Felsberg, Luka Cehovin, Gustavo Fernandez, Tomas Vojir, Gustav Hager, Georg Nebehay, and Roman Pflugfelder. The visual object tracking vot2015 challenge results. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 1–23, 2015.
- Yoshiki Kuramoto. Self-entrainment of a population of coupled non-linear oscillators. In *International Symposium on Mathematical Problems in Theoretical Physics*, pages 420–422, 1975.
- Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.
- Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.
- Yang Li and Jianke Zhu. A scale adaptive kernel correlation filter tracker with feature integration. In *European Conference on Computer Vision*, pages 254–265. Springer, 2014.
- Pengpeng Liang, Erik Blasch, and Haibin Ling. Encoding color information for visual tracking: Algorithms and benchmark. *IEEE Transactions on Image Processing*, 24(12):5630–5644, 2015.
- Ting Liu, Gang Wang, and Qingxiong Yang. Real-time part-based visual tracking via adaptive correlation filters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4902–4912, 2015.
- Yao Liu, Adith Swaminathan, Alekh Agarwal, and Emma Brunskill. Off-policy policy gradient with state distribution correction. *arXiv preprint arXiv:1904.08473*, 2019.
- Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*, 2019.

- Issei Maeda, Shun Maeoka, Kenichi Tokuda, and Shuro Nakajima. Gait emergence by discrete leg motion period control using contact signals and CPGs. *Transactions of the Japan Society of Mechanical Engineers*, 87(897):20–00340, 2021.
- Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning. In *NeurIPS Track on Datasets and Benchmarks*, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/hash/28dd2c7955ce926456240b2ff0100bde-Abstract-round2.html.
- Takamitsu Matsubara, Jun Morimoto, Jun Nakanishi, Masa-aki Sato, and Kenji Doya. Learning cpg-based biped locomotion with a policy gradient method. *Robotics and Autonomous Systems*, 54(11):911–920, 2006.
- K Matsuoka. Sustained oscillations generated by mutually inhibiting neurons with adaptation. *Biological Cybernetics*, 52(6):367–376, 1985.
- Kiyotoshi Matsuoka. Mechanisms of frequency and pattern control in the neural rhythm generators. *Biological Cybernetics*, 56(5):345–353, 1987.
- Kiyotoshi Matsuoka. Analysis of a neural oscillator. *Biological Cybernetics*, 104(4-5):297–304, 2011.
- Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):1979–1993, 2018.
- Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.
- Thomas M Moerland, Joost Broekens, Aske Plaat, and Catholijn M Jonker. Model-based Reinforcement Learning: A Survey. 2020.
- Takeshi Mori, Yutaka Nakamura, Masa-Aki Sato, and Shin Ishii. Reinforcement learning for a cpg-driven biped robot. In *Proceedings of the 19th National Conference on Artificial Intelligence*, AAAI’04, page 623–630. AAAI Press, 2004. ISBN 0262511835.
- Yutaka Nakamura, Takeshi Mori, Masaaki Sato, and Shin Ishii. Reinforcement learning for a biped robot based on a CPG-actor-critic method. *Neural Networks*, 20(6):723–735, 2007.

- Kohei Ohashi, Kosuke Nakanishi, Nao Goto, Yuji Yasui, and Shin Ishii. Orthogonal adversarial deep reinforcement learning for discrete-and continuous-action problems. *IEEE Access*, 2024.
- Ian Osband, Benjamin Van Roy, Daniel J Russo, and Zheng Wen. Deep exploration via randomized value functions. *Journal of Machine Learning Research*, 20(124):1–62, 2019.
- Dai Owaki, Masashi Goda, Sakiko Miyazawa, and Akio Ishiguro. A Minimal Model Describing Hexapedal Interlimb Coordination: The Tegotae-Based Approach. *Frontiers in Neurorobotics*, 11, 2017.
- Paavo Parmas, Carl Edward Rasmussen, Jan Peters, and Kenji Doya. PIPPS: Flexible model-based policy search robust to the curse of chaos. In *International Conference on Machine Learning*, pages 4065–4074. PMLR, 2018.
- Shubham Pateria, Budhitama Subagdja, Ah-hwee Tan, and Chai Quek. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 54(5):1–35, 2021.
- Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. 2017.
- Luis Pineda, Brandon Amos, Amy Zhang, Nathan O. Lambert, and Roberto Calandra. MBRL-Lib: A Modular Library for Model-based Reinforcement Learning. *Arxiv*, 2021.
- Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *International conference on machine learning*, pages 2817–2826. PMLR, 2017.
- Aske Plaat, Walter Kusters, and Mike Preuss. High-accuracy model-based reinforcement learning, a survey. *Artificial Intelligence Review*, 56(9):9541–9573, 2023.
- Ronald Poppe. A survey on vision-based human action recognition. *Image and vision computing*, 28(6):976–990, 2010.
- Horst Possegger, Thomas Mauthner, and Horst Bischof. In defense of color-based model-free tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2113–2120, 2015.

- Marc Raibert, Kevin Blankespoor, Gabriel Nelson, and Rob Playter. BigDog, the Rough-Terrain Quadruped Robot. *IFAC Proceedings Volumes*, 41(2):10822–10825, 2008.
- Esteban Real, Jonathon Shlens, Stefano Mazzocchi, Xin Pan, and Vincent Vanhoucke. Youtube-boundingboxes: A large high-precision human-annotated data set for object detection in video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5296–5305, 2017.
- Unitree Robotics. <https://www.unitree.com/products/a1/>.
- Gavin A Rummery and Mahesan Niranjana. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering, 1994.
- Nicola Russo. Rex: an open-source quadruped robot. <https://github.com/nicrusso7/rex-gym>, 2021.
- Manzoor Sajjad and Choi Youngjin. A unified neural oscillator model for various rhythmic locomotions of snake-like robot. *Neurocomputing*, 173:1112–1123, 2016.
- Stefan Schaal, Jan Peters, Jun Nakanishi, and Auke Ijspeert. Learning Movement Primitives. In *Robotics Research. The Eleventh International Symposium*, pages 561–572, 2005.
- Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.
- John Schulman, Sergey Levine, Philipp Moritz, Michael I Jordan, and Pieter Abbeel. Trust Region Policy Optimization. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning*, volume 37, pages 1889–1897, 2015.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. 2017.
- Max Schwarzer, Ankesh Anand, Rishab Goel, R Devon Hjelm, Aaron Courville, and Philip Bachman. Data-efficient reinforcement learning with self-predictive representations. In *International Conference on Learning Representations*, 2021.

- Milad Shafiee, Guillaume Bellegarda, and Auke Ijspeert. Viability leads to the emergence of gait transitions in learning agile quadrupedal locomotion on challenging terrains. *Nature Communications*, 15(1):3073, 2024.
- David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dhharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018. URL <https://www.science.org/doi/10.1126/science.aar6404>.
- Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.
- Malcolm Strens. A Bayesian framework for reinforcement learning. In *ICML*, volume 2000, pages 943–950, 2000.
- Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.
- Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.
- Gentaro Taga, Yoko Yamaguchi, and Hiroshi Shimizu. Self-organized control of bipedal locomotion by neural oscillators in unpredictable environment. *Biological Cybernetics*, 65(3):147–159, 1991.
- Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-Real: Learning Agile Locomotion For Quadruped Robots. 2018.
- Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.
- Joshua P Tobin. *Real-world robotic perception and control using synthetic data*. University of California, Berkeley, 2019.
- Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

- Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, 2023. URL <https://zenodo.org/record/8127025>.
- Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqu Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020.
- Ikechukwu Uchendu, Ted Xiao, Yao Lu, Banghua Zhu, Mengyuan Yan, Joséphine Simon, Matthew Bennice, Chuyuan Fu, Cong Ma, Jiantao Jiao, Sergey Levine, and Karol Hausman. Jump-Start Reinforcement Learning. 2022.
- Jessica Van Brummelen, Marie O ’Brien, Dominique Gruyer, and Homayoun Najjaran. Autonomous vehicle perception: The technology of today and tomorrow. *Transportation Research Part C: Emerging Technologies*, 89:384–406, 2018. ISSN 0968-090X. doi: <https://doi.org/10.1016/j.trc.2018.02.012>. URL <https://www.sciencedirect.com/science/article/pii/S0968090X18302134>.
- Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, pages 2094–2100, 2016.
- Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dhharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.
- Jiayu Wang, Chuxiong Hu, and Yu Zhu. CPG-Based Hierarchical Locomotion Control for Modular Quadrupedal Robots Using Deep Reinforcement Learning. *IEEE Robotics and Automation Letters*, 6(4):7193–7200, 2021.
- P J Whelan. Control of locomotion in the decerebrate cat. *Progress in Neurobiology*, 49(5):481–515, 1996.
- Pierre-Brice Wieber. Model Predictive Control for Biped Walking Motion Generation. *Journal of the Robotics Society of Japan*, 32(6):503–507, 2014.
- Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Online object tracking: A benchmark. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2411–2418, 2013.

- Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Object tracking benchmark. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9):1834–1848, 2015.
- Nikolai Yakovenko, Liangliang Cao, Colin Raffel, and James Fan. Poker-cnn: A pattern learning strategy for making draws and bets in poker games using convolutional networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), Feb. 2016. doi: 10.1609/aaai.v30i1.10013. URL <https://ojs.aaai.org/index.php/AAAI/article/view/10013>.
- De Jong Yeong, Gustavo Velasco-Hernandez, John Barry, and Joseph Walsh. Sensor and sensor fusion technology in autonomous vehicles: A review. *Sensors*, 21(6):2140, 2021.
- Sangdoo Yun, Jongwon Choi, Youngjoon Yoo, Kimin Yun, and Jin Young Choi. Action-Decision Networks for Visual Tracking with Deep Reinforcement Learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1349–1358, 2017.
- Da Zhang, Hamid Maei, Xin Wang, and Yuan-Fang Wang. Deep Reinforcement Learning for Visual Object Tracking in Videos. *arXiv preprint arXiv:1701.08936*, 2017.
- Shangdong Zhang, Bo Liu, Hengshuai Yao, and Shimon Whiteson. Provably convergent two-timescale off-policy actor-critic with function approximation. In *International Conference on Machine Learning*, pages 11204–11213. PMLR, 2020.
- Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey. In *2020 IEEE Symposium Series on Computational Intelligence*, pages 737–744, 2020.
- S. Kevin Zhou, Hoang Ngan Le, Khoa Luu, Hien V Nguyen, and Nicholas Ayache. Deep reinforcement learning in medical imaging: A literature review. *Medical Image Analysis*, 73:102193, 2021.
- Xian Zhou, Qiao Yiling, Xu Zhenjia, Wang Tsun-Hsuan, Chen Zhehuan, Zheng Juntian, Xiong Ziyang, Wang Yian, Zhang Mingrui, Ma Pingchuan, Wang Yufei, Dou Zhiyang, Kim Byungchul, Tian Yunsheng, Chen Yipu, Qiu Xiaowen, Lin Chunru, He Tairan, Si Zilin, Zhang Yunchu, Yang Zhanlue, Liu Tiantian, Li Tianyu, Yamazaki Kashu, Zhang Hongxin, Ha Huy, Zhang Yu, Liu Michael, Zheng Shaokun, Fu Zipeng, Wu Qi, Geng Yiran, Chen Feng, Hu Yuanming, Shi Guanya, Liu Lingjie, Komura Taku, Erickson Zackory, Held David, Li Minchen, Jim Fan Linxi, Zhu Yuke, Matusik Wojciech, Gutfreund Dan, Song Shuran, Rus Daniela, Lin Ming, Zhu Bo, Fragkiadaki

Katerina, and Gan Chuang. Genesis: A universal and generative physics engine for robotics and beyond. <https://github.com/Genesis-Embodied-AI/Genesis>, December 2024.