# MP3: Movement Primitive-Based (Re-)Planning Policy

**Fabian Otto** [*] [1,2], **Hongyi Zhou**[*] [3], **Onur Celik**[4], **Ge Li**[4], **Rudolf Lioutikov**[3] and **Gerhard Neumann**[4]

[*] These authors contributed equally to this work
[1] University of Tübingen, Germany
[2] Bosch Center for Artificial Intelligence, Germany
[3] Intuitive Robots Lab, Karlsruhe Institute of Technology, Germany
[4] Autonomous Learning Robots Lab, Karlsruhe Institute of Technology, Germany
fabian.otto@bosch.com, hongyi.zhou@kit.edu

**Abstract:** We introduce a novel deep reinforcement learning (RL) approach called Movement Primitive-based Planning Policy (MP3). By integrating movement primitives (MPs) into the deep RL framework, MP3 enables the generation of smooth trajectories throughout the whole learning process while effectively learning from sparse and non-Markovian rewards. Additionally, MP3 maintains the capability to adapt to changes in the environment during execution. Although many early successes in robot RL have been achieved by combining RL with MPs, these approaches are often limited to learning single stroke-based motions, lacking the ability to adapt to task variations or adjust motions during execution. Building upon our previous work, which introduced an episode-based RL method for the non-linear adaptation of MP parameters to different task variations, this paper extends the approach to incorporating replanning strategies. This allows adaptation of the MP parameters throughout motion execution, addressing the lack of online motion adaptation in stochastic domains requiring feedback. The project website can be accessed at https://intuitive-robots.github.io/mp3_website/.

**Keywords:** Movement primitives, reinforcement learning, robot learning

## 1 Introduction

Traditional deep RL methods use a step-based policy, where at each time step the policy explores in the atomic action space. During interaction with the environment, the agent collects state, action, and reward data points at each time step, which are used to update the policy. Although using every atomic action generates a vast amount of data-points for the policy update, it also complicates exploration due to the typical random walk behavior and introduces a lot of noise in the policy evaluation process (see Figure 2). Therefore, these methods often rely on informative reward signals throughout the interaction sequence, making them less effective in sparse or non-Markovian settings where feedback from the environment is delayed. Moreover, step-based exploration can result in slower convergence and jerky, potentially dangerous behavior for robots.

In contrast, RL with MPs (MPRL) is typically based on episode-based RL (ERL) [1, 2, 3, 4]. ERL methods learn to parameterize a desired trajectory used for a controller based on a task description known as the context, which remains fixed throughout the entire episode. These methods explore the trajectory space, meaning that a parameter is sampled given the context only once at the beginning of the episode and executed without resampling. This exploration strategy results in time-correlated exploration, smooth behaviors, and improved performance in sparse or non-Markovian reward settings [4]. In our recent work [4], we integrated ERL with MPs into a deep policy gradient algorithm that is based on trust region projection layer (TRPL) [5]. While this algorithm can non-linearly adapt the parameters of the MP to the given context and achieve high-quality policies for complex robotic tasks, it is inherently constrained to generating open-loop trajectories that cannot be adapted or adjusted during execution.

This paper is an extension of Otto et al. [4], where we add learning non-linear replanning policies instead of just the initial adaptation of the MP to the context, combining the benefits of ERL with MPs and step-based RL (SRL) methods. MP3 still explores in the trajectory space, yet the agent is now able to change the desired trajectory within an episode, enabling it to adapt its behavior to unpredictable changes in the environment. We demonstrate the effectiveness of our method by presenting various complex simulated robotic tasks, such as robot table tennis, beer-pong, a complex box-pushing task, and large-scale manipulation tasks on Meta-World [6]. We compare MP3 to state-of-the-art SRL and ERL methods and illustrate improved performance in sophisticated, sparse reward settings and settings that require replanning.

## 2 Related Works

**Episode-based Reinforcement Learning.** In the framework of contextual episode-based policy search [1, 3], RL is usually treated as a black-box optimization problem. The goal is to maximize the expected return $R(\boldsymbol{w}, \boldsymbol{c})$ by optimizing a context $\boldsymbol{c}$ dependent searching distribution $\pi(\boldsymbol{w}|\boldsymbol{c})$ over the controller parameters $\boldsymbol{w}$. The return function $R(\boldsymbol{w}, \boldsymbol{c})$ is not subject to any structural assumptions, and it can be any non-Markovian function of the resulting trajectory due to the black-box nature of the problem. Most ERL algorithms are focused on the non-contextual setting, where different optimization techniques have been used, such as policy gradients [7], natural gradients [8], stochastic search strategies [9, 10, 11], or trust-region optimization techniques [2, 3, 12]. Early methods that incorporate context adaptation [12, 11] only consider a linear mapping from context to parameter space, which is a major limitation on the performance of these approaches. In contrast, we consider highly nonlinear context-parameter relationships using neural networks.

**Reinforcement Learning with Movement Primitives.** While most works of MPRL concentrate on learning a single MP parameter vector for a single task configuration [2, 13, 14, 15], some methods allow linear adaptation of the MP's parameter vector to the context [3, 16, 17]. In addition, a few RL approaches leverage non-linear policies combined with predefined action primitives, such as pushing or grasping motions [18, 19]. One approach that directly uses MPs and deep networks in a SRL setting is neural dynamic policies (NDP) [20]. However, the main exploration of NDP still takes place at the action level rather than at the trajectory level, similar to standard step-based approaches, which neglects the main benefit of using MPs in an RL context.

## 3 Deep Reinforcement Learning with Movement Primitives

In this work, we present a framework to effectively combine MPs with deep RL methods. This framework consists of three major components (see Figure 1):

- One RL policy which takes the environment observation as input and outputs an MP weight vector that is used for multiple time steps.
- One MP model which uses the weight vector as input to generate a desired trajectory.
- One low-level controller which converts the desired trajectory into raw actions and interacts with the environment.

This approach is simple but highly versatile. The planning horizon (length of the generated trajectory before a new weight vector is chosen) can vary from a single step to the entire episode. Two special cases correspond to two common RL paradigms: (i) When the planning horizon is equal to one, our framework is similar to an SRL algorithm (although with a higher dimensional action space). (ii) When it is equal to the episode length, the framework corresponds to an ERL algorithm.

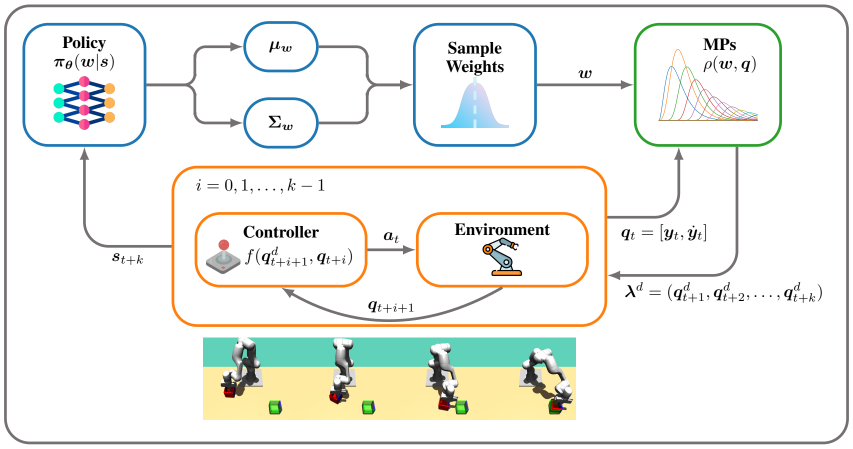

Policy
$\pi_\theta(\boldsymbol{w}|\boldsymbol{s})$
$\mu_{\boldsymbol{w}}$
$\Sigma_{\boldsymbol{w}}$
Sample Weights
$\boldsymbol{w}$
MPs
$\rho(\boldsymbol{w}, \boldsymbol{q})$
$i = 0, 1, \ldots, k-1$
Controller
$f(\boldsymbol{q}^d_{t+i+1}, \boldsymbol{q}_{t+i})$
$\boldsymbol{a}_t$
Environment
$\boldsymbol{q}_t = [\boldsymbol{y}_t, \dot{\boldsymbol{y}}_t]$
$\boldsymbol{\lambda}^d = (\boldsymbol{q}^d_{t+1}, \boldsymbol{q}^d_{t+2}, \ldots, \boldsymbol{q}^d_{t+k})$
$\boldsymbol{s}_{t+k}$
$\boldsymbol{q}_{t+i+1}$

Figure 1: This figure provides an overview of the proposed framework that combines deep RL with MPs. Instead of generating a raw action directly, the policy generates a set of weights that parameterizes a MP. The MP predicts a desired trajectory given the weights and initial conditions, which is then converted to raw actions using a tracking controller.

### 3.1 Movement Primitives

MPs are a widely used tool for motion representation and generation in robotics. They are used as building blocks for movements, allowing for the modulation of motion behavior and the creation of more complex movements through combination or concatenation.

**Probabilistic Movement Primitives (ProMPs).** probabilistic movement primitive (ProMP) [21] generate the trajectory $y(t)$ by a linear basis function model

$$y(t) = \boldsymbol{\Phi}^\intercal(t)\boldsymbol{w},$$

where $\boldsymbol{w}$ is the time-independent weight vector, $y(t)$ is the trajectory position at time step $t$ and $\boldsymbol{\Phi}$ are pre-defined time-dependent basis functions. One of the key limitations of ProMPs as a representation method is their lack of smoothness in trajectory replanning and concatenation, limiting the usage of ProMPs in situations where the weight vector needs to be updated throughout motion execution due to unpredictable changes in the environment.

**Dynamic Movement Primitives (DMPs).** Dynamic movement primitives (DMPs) [22, 23] form a trajectory by integrating a dynamic system, providing smooth replanning of both position and velocity [24, 25, 26]. However, this smoothness comes at a computational cost, as DMPs require online numerical integration to compute a trajectory.

**Probabilistic Dynamic Movement Primitives (ProDMPs).** To combine the advantages and address the limitations of ProMPs and DMPs, Li et al. [27] recently proposed probabilistic dynamic movement primitives (ProDMPs). ProDMPs formulate trajectory similar to that of ProMPs:

$$y(t) = c_1 y_1(t) + c_2 y_2(t) + \boldsymbol{\Phi}(t)^\intercal \boldsymbol{w},$$

where the added terms $c_1 y_1(t) + c_2 y_2(t)$ are included to ensure accurate trajectory initialization. This formulation combines the distributional modeling benefits of ProMP with the precision in trajectory initiation offered by DMP. In this work, we specifically use the ProDMPs model as our trajectory generator because it ensures smooth replanning with low computational cost.

### 3.2 Reinforcement Learning Objective with Movement Primitives

While traditional SRL methods rely on single raw actions $\boldsymbol{a}_t \in \mathcal{A}$ per time step, we train a policy to select a weights vector $\boldsymbol{w}_t \in \mathcal{W}$ in MP's parameter space $\mathcal{W}$. The weights vector is then translated to a desired trajectory of the proprioceptive states $\boldsymbol{\lambda}^d = (\boldsymbol{q}^d_{t+1}, \boldsymbol{q}^d_{t+2}, \ldots, \boldsymbol{q}^d_{t+k})$, where $q^d_t = [\boldsymbol{y}^d_t, \dot{\boldsymbol{y}}^d_t]$ consists of desired position $\boldsymbol{y}^d_t$ and desired velocity $\dot{\boldsymbol{y}}^d_t$ at time step $t$, and $k$ denotes the planning horizon. Given the desired trajectory and the measured proprioceptive state, a tracking controller $f(q^d_t, q_t)$ decides the action at each step, resulting in a trajectory in the raw action space $(\boldsymbol{a}_{t+1}, \boldsymbol{a}_{t+2}, \ldots, \boldsymbol{a}_{t+k}) \in \mathcal{A}$. In contrast to the step-wise sample $(\boldsymbol{s}_t, \boldsymbol{a}_t, R_t)$ used in SRL, we use temporarily-abstracted samples of the form $(\boldsymbol{s}_t, \boldsymbol{w}_t, R^k_t)$. The reward $R^k_t = R_{t:t+k-1}$ of each trajectory segment is defined as the cumulative reward over all the segment's time steps $t$ to $t+k-1$

$$R^k_t(\boldsymbol{s}_t, \boldsymbol{w}_t) = \sum_{i=0}^{k-1} \gamma^i r(\boldsymbol{s}_{t+i}, \boldsymbol{a}_{t+i}), \tag{1}$$

where $a_t$ and $s_t$ are the executed actions and observed states following the desired trajectory and tracked by the controller. While our approach supports different $k$ for each segment, we only consider planning segments with equal length in this work. We can compute the episode return by taking the cumulative discounted sum of the segment rewards. Using the notation from above, we can express this as

$$G^k_t = \sum_{i=0}^{\lceil T/k-1 \rceil} \gamma^{ik} R^k_{t+ki}(\boldsymbol{w}_{t+ki}, \boldsymbol{s}_{t+ki}), \tag{2}$$

where $\gamma \in (0, 1]$ is the discount factor. It is worth noting that there are two special cases to consider. In the black-box setting, in other words, when the MP parameters are chosen only at the beginning of the episode, then $k = T$ and the segment reward equals the episode return

$$R^{T-1}_0 = \sum_{t=0}^{T-1} \gamma^t r(a_t, s_t). \tag{3}$$

The second special case is step-based RL. That is, we choose a new parameter vector at every time step and $k = 1$. In this case, segment reward is equivalent to step reward

$$R^1_t = r(a_t, s_t). \tag{4}$$

This gives the insight that we can alter between SRL and ERL by choosing different planning horizons $k$.

## 3.3 Policy-gradients for MP weight-selection policies

With these rewards, we can now also define matching value and advantage functions

$$V^\pi(\boldsymbol{s}) = \mathbb{E}\left[G^k_t | \boldsymbol{s}_t = \boldsymbol{s}; \pi_\theta\right] \qquad A^\pi(\boldsymbol{s}, \boldsymbol{w}) = \mathbb{E}\left[G^k_t | \boldsymbol{s}_t = \boldsymbol{s}, \boldsymbol{w}_t = \boldsymbol{w}; \pi_\theta\right] - V^\pi(\boldsymbol{s}). \tag{5}$$

Following the step-based policy gradient [28, 29], we optimize the advantage function using the likelihood ratio gradient and an importance sampling estimator. The resulting objective

$$\hat{J}(\pi_{\boldsymbol{\theta}}, \pi_{\boldsymbol{\theta}_\text{old}}) = \mathbb{E}_{(\boldsymbol{s},\boldsymbol{w})\sim p(\boldsymbol{s}), \pi_{\boldsymbol{\theta}_\text{old}}}\left[\frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{w}|\boldsymbol{s})}{\pi_{\boldsymbol{\theta}_\text{old}}(\boldsymbol{w}|\boldsymbol{s})} A^{\pi_{\boldsymbol{\theta}_\text{old}}}(\boldsymbol{s}, \boldsymbol{w})\right], \tag{6}$$

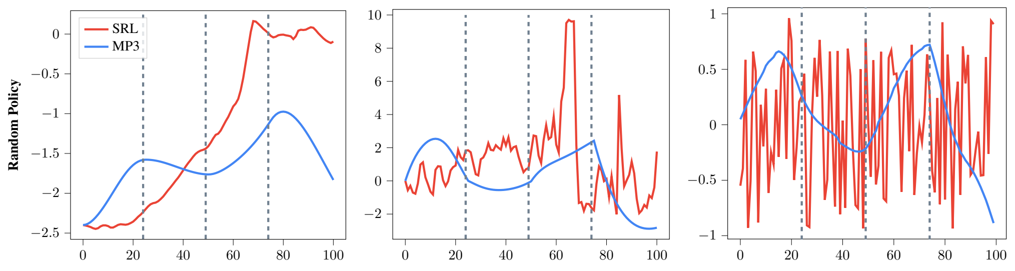
is maximized w.r.t $\boldsymbol{\theta}$, with $\pi_{\boldsymbol{\theta}_\text{old}}$ being the old behavior policy used for sampling. We can further make use of a learned state-value function $V_\phi(\boldsymbol{s}) \approx V^\pi(\boldsymbol{s})$ for the advantage estimator, which is approximated by optimizing

$$\arg\min_\phi \mathbb{E}_{(\boldsymbol{s},\boldsymbol{w})\sim p(\boldsymbol{s}), \pi_{\boldsymbol{\theta}_\text{old}}}\left[\left(V_\phi(\boldsymbol{s}) - G^k_t\right)^2\right]. \tag{7}$$

This formulation also enables the use of advantage estimation methods, such as general advantage estimation [30]. During the update of the policy, neither the MP $\rho(\boldsymbol{w}, \boldsymbol{q})$ nor the controller $f(\boldsymbol{q}^d_{t+1}, \boldsymbol{q}_t)$ are needed, i.e., our approach would work with any form of parametrizable controller.

## 3.4 Choice of the Planning Horizon

Our method harnesses the merits of two common RL paradigms: step-based RL (SRL) and episode-based RL (ERL). The agent's behavior can seamlessly switch between the two paradigms according to the planning horizon $k$. There are three cases when selecting the planning horizon.



(a) Position (rad) (b) Velocity (rad/s) (c) Torque (Nm)

Figure 2: This figure presents a comparison between step-level exploration (red) and the proposed trajectory-level exploration that is used in MP3 (blue).

**Black-Box Setting.** The first special case arises when the planning horizon is equal to the episode length, that is, $k = T$. In this case, the agent generates only one desired trajectory for the entire episode, similar to an open-loop motion planner. We refer to this setting as MP3-Black Box (MP3-BB) in the following discussion, since it treats reinforcement learning as a black-box optimization problem. The black-box nature facilitates dealing with sparse and non-Markovian rewards, leading to a more intuitive reward design. However, the black-box nature also limits its applicability in dynamic environments, where the agent must adapt to environmental changes during execution.

**Step-Based Setting.** At the opposite end of the planning horizon spectrum is the case where $k = 1$. Here, the agent only executes the desired trajectory for one step, after which it generates a new plan, repeating this loop throughout the episode similar to the SRL setting. However, the use of MPs guarantees second-order smoothness (position and velocity), resulting in a more consistent and smooth behavior during exploration (see Figure 2). Nonetheless, we did not observe significant improvements using this setting over the standard SRL setting, and our discussion only aims to highlight the flexibility of the proposed method.

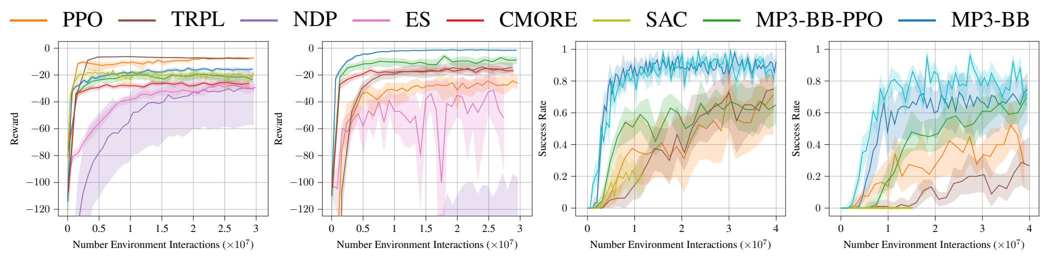
**Re-planning with MPs.** The more general case falls somewhere between the two extremes of SRL and ERL. In this approach, the agent generates a new desired trajectory after executing the current trajectory for a predefined number of steps ($1 < k < T$). This method leverages the strengths of both SRL and ERL while addressing some of their shortcomings:

1. In SRL, stochastic raw action selection often results in jerky random walk behavior that does not fully explore the trajectory space of the agent. In contrast, our approach explores the weight space of MPs, leveraging the MP's smoothness guarantees for more consistent and effective exploration (see Figure 2).

2. Trajectory-level exploration encapsulates the temporal abstraction within each trajectory segment, reducing the number of decisions to make for each episode and improving the agent's ability to handle the sparsity in the reward function.

3. The ERL agent only makes decisions at the beginning of each episode and treats each episode as a black-box, limiting their ability to address observation noises and dynamics in the environment. Our approach addresses this shortcoming by incorporating periodic re-planning during online execution.

Many SRL algorithms use a similar design called *frame-skipping*, which can help with the partial observability of some Atari games [31]. However, frame-skipping just repeats the same action for the "skipped" frames, limiting the trajectory's expressive capacity. In contrast, planning with MP can "skip" more frames without compromising expressiveness.



(a) 5D Reacher - Dense (b) 5D Reacher - Sparse (c) Box Push - Dense (d) Box Push - Sparse

Figure 3: The figures (a) and (b) show the learning curve for the 5D reacher task with dense and sparse reward signals. The success rate for box pushing with dense and sparse rewards are presented in figures (c) and (d), respectively.

## 4 Experimental Results

For our evaluation, we begin by demonstrating the effectiveness of our method in handling sparse and non-Markovian rewards, improving precision in the black-box setting with $k = T$. Next, we conduct a large-scale study on all 50 Meta-World tasks [6] to showcase our competitive performance on various robot manipulation tasks that come with highly shaped dense rewards. Finally, we evaluate our method with replanning for several tasks with dynamics in the environment.

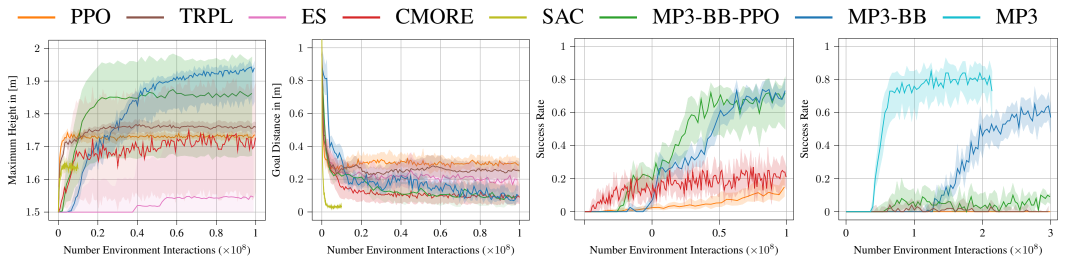
We compare our methods, which will be noted as MP3 and MP3-BB for the replanning and black-box cases respectively, against several other step-based methods, including proximal policy optimization (PPO) [32], TRPL [5], soft actor critic (SAC) [33], and NDP [20], as well as a deep evolution strategies (ES) [34], the linear adaption method contextual model-based relative entropy stochastic search (CMORE) with ProMPs [12] as well as MP3-PPO (MP3-PPO) and MP3-BB-PPO (MP3-BB-PPO), which are equivalent to MP3 and MP3-BB but trained with PPO instead of TRPL.

We evaluate our method on 20 different seeds and compute ten evaluation runs after each iteration. To report our results, we use the interquartile mean (IQM) with a 95% stratified bootstrap confidence interval and performance profiles where feasible Agarwal et al. [35]. For a detailed description of the hyperparameters used in the evaluation, please refer to Appendix F.

### 4.1 Black-Box Reinforcement Learning

**Dealing with Sparse Rewards** As introductory tasks, we use an extended reacher task from OpenAI gym [36] by using five actuated joints and a complex robot box pushing task. For a detailed environment description, please see Appendix D. While SRL algorithms demonstrate competitive performance in dense rewards, they all encounter substantial performance degradation in sparse reward settings. On the other hand, our method maintains good performance in both settings.

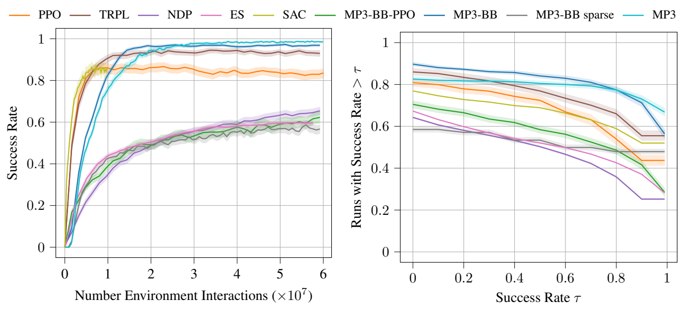
**Dealing with non-Markovian Rewards** To assess the effectiveness of our method in complex reward settings, we test it with non-Markovian rewards, which are particularly useful for robot learning tasks that require the agent to use feedback from the full trajectory history. We consider three environments under non-Markovian reward settings. Firstly, we use a modified version of the OpenAI Gym hopper [36], which aims to jump as high as possible and land at a target location (see Appendix D.3). Secondly, we conduct experiments in a Beer pong environment [17]. In this task, the target is to throw a ball into a cup at various locations on a table (see Appendix D.4). Finally, we showcase the performance of our method in a robot table tennis task, which targets to return ball with different initial states to randomized desired landing positions. The results are presented at Figure 4. While SRL methods are in general struggling with these tasks, MP3-BB provides an effective solution for handling non-Markovian reward structures, which are often more natural and easier to define than engineered dense rewards.



(a) Hopper Jump - Max Height (b) Hopper Jump - Dist. Error (c) Beer Pong (d) Table Tennis

Figure 4: The figures (a) and (b) show the maximum jumping height of the hopper's center of mass and the target distance, respectively. With the non-Markovian reward, the hopper can jump approximately 20cm higher with increased goal precision. Figure (c) shows the beer pong task. The success rate of the table tennis task is shown in (d).

## 4.2 Large Scale Robot Manipulation



(a) Success Rate (b) Performance Profile

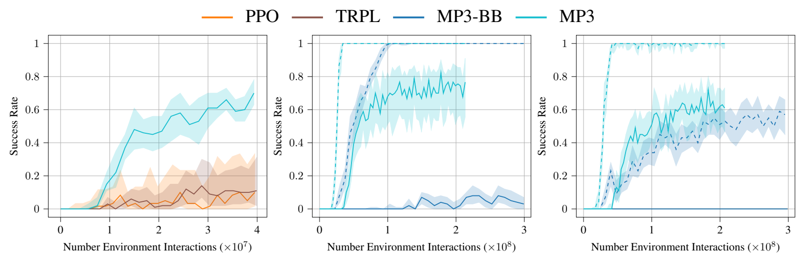
Figure 5: Metaworld Evaluation. (a) Overall Success Rate across all 50 tasks, reported using Interquartile Mean (IQM). (b) Performance profile, illustrating the fraction of runs that exceed the threshold specified on the x-axis.

We also showcase our ability to learn high quality policies on the Meta-World benchmark suite [6]. We train individual policies for each environment but use the same hyperparameters. Our results (Figure 5a) show that PPO and TRPL achieve the best sample complexity, but MP3-BB performs competitively in terms of asymptotic performance and even outperforms PPO slightly in terms of asymptotic performance. Although the gap between PPO and MP3-BB in the aggregated view is relatively small, the corresponding performance profiles (Figure 5b) reveal that MP3-BB performs better above the 80% threshold. This means that MP3-BB finds more consistent solutions than PPO with higher precision and solves these tasks without failures. SAC performs similar to PPO, whereas NDP, ES, and MP3-BB-PPO are not achieving a competitive performance.

## 4.3 Replanning with Movement Primitives

We evaluate our approach in the online replanning case by decreasing the planning horizon, such that $1 \leq k < T$, positioning it between SRL ($k = 1$) and ERL ($k = T$). This approach, which we refer to as MP3, offers two significant benefits. Firstly, it leads to a more precise policy due to the closed-loop nature of the method. Secondly, it enables the handling of environmental dynamics through online replanning. We conduct a thorough ablation about the impact of planning horizon on performance, for the results and discussion, please see Appendix A.

**Quality of the Learned Policy.** We evaluated the performance of MP3 agents by conducting experiments in three challenging environments: the Meta-World benchmark suite [6] for large-scale robot manipulation (Figure 5), Box Pushing with dense (Figure 3c) and sparse (Figure 3d) rewards, and Table Tennis with non-Markovian reward (Figure 4d). We observed that the use of replanning yields better asymptotic performance in all cases while it can harm slightly the sample efficiency (observed in Meta-World experiments). We attribute this to the higher dimensional state space that must be considered in the replanning case compared to the black-box case.



(a) Box Push - Switch (b) Table Tennis - Switch (c) Table Tennis - Wind

Figure 6: This figure displays the success rate of perturbed tasks with and without replanning. The success rate of box pushing with goal switching is shown at (a), as well as the success rate of table tennis with goal switching (b) and with wind (c). In the box pushing tasks (a), the solid lines represent learning curves of dense reward, while the dashed lines are learning curves from temporal-sparse reward. In table tennis tasks (b) and (c), the solid lines represent the success rates, and the dashed lines are hit rate.

**Dealing with Uncertainties in the Environments.** To demonstrate the robustness of MP3 in handling unforeseen events in the environment, we modified the box pushing and the table tennis tasks to include uncertainties that require incorporating feedback throughout the execution of the episode.

In the box-pushing experiments, we randomly switch to a new target position and orientation during execution after 20% of the max episode length. We compared the performance of our method against step-based PPO and TRPL in the dense reward setting. The results in Figure 6a show that MP3 achieved the best performance under this setting.

For the table tennis environment, we test two kinds of uncertainties. We compare MP3 only with the MP3-BB as the SRL algorithms have shown to be incapable of solving the table tennis task even in a static environment. Firstly, we modify the desired landing position of the ball, similar to the goal change for the box pushing task. Specifically, we initialize the desired landing position at a random location and randomly switch it during the episode. Our results in Figure 6b suggest that the MP3 agent is able to adapt its behavior and return the ball to the new target point with high precision. In contrast, the MP3-BB agent, which only receives the initial observation containing the initial target position, can only hit the ball but cannot solve this task. Secondly, we add wind to the environment by applying a random force to the ball, which is unknown to the agent and constant for an entire episode. However, the agent can still infer the underlying applied force according to the velocity of the ball, but only after observing the ball for a certain number of time steps. Due to the wind, the MP3-BB agent is not able to hit the ball consistently, while the MP3 agent slightly drops in performance but can still achieve reasonably good results (Figure 6c).

## 5 Conclusion and Limitations

Our work presents a new approach for combining SRL and ERL by integrating recent advancements in trust-region-based policy search [5] and MPs [27]. This approach is a promising way to handle tasks with sparse and non-Markovian rewards, enabling a more intuitive reward design. Furthermore, our method showed competitive performance against state-of-the-art SRL algorithms in large-scale robot manipulation tasks, as confirmed by thorough empirical evaluations.

Although our proposed method shows promise, there remain two significant limitations that require addressing in future work. Firstly, our current approach only considers fixed-length planning horizons and relies solely on time-based replanning triggers. Yet, it is considered more natural only replanning when a certain event happens. Secondly, our method, and ERL approaches in general, typically require more interaction time than SRL in dense reward settings. This is mainly due to the encapsulation of temporal-correlated information in highly abstracted samples.

**Acknowledgments**

If a paper is accepted, the final camera-ready version will (and probably should) include acknowledgments. All acknowledgments go at the end of the paper, including thanks to reviewers who gave useful comments, to colleagues who contributed to the ideas, and to funding agencies and corporate sponsors that provided financial support.

## References

[1] M. P. Deisenroth, G. Neumann, J. Peters, et al. A survey on policy search for robotics. *Foundations and trends in Robotics*, 2(1-2):388–403, 2013.

[2] A. Abdolmaleki, R. Lioutikov, J. R. Peters, N. Lau, L. Pualo Reis, and G. Neumann. Model-based relative entropy stochastic search. *Advances in Neural Information Processing Systems*, 28, 2015.

[3] C. Daniel, G. Neumann, and J. Peters. Hierarchical relative entropy policy search. In *Artificial Intelligence and Statistics*, pages 273–281. PMLR, 2012.

[4] F. Otto, O. Celik, H. Zhou, H. Ziesche, N. A. Vien, and G. Neumann. Deep black-box reinforcement learning with movement primitives. *arXiv preprint arXiv:2210.09622*, 2022.

[5] F. Otto, P. Becker, N. Anh Vien, H. C. Ziesche, and G. Neumann. Differentiable trust region layers for deep reinforcement learning. In *International Conference on Learning Representations*, 2021.

[6] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2019. URL https://arxiv.org/abs/1910.10897.

[7] F. Sehnke, C. Osendorfer, T. Rückstiess, A. Graves, J. Peters, and J. Schmidhuber. Parameter-exploring policy gradients. *Neural Networks*, 21(4):551–559, May 2010. doi:10.1016/j.neunet.2009.12.004.

[8] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber. Natural evolution strategies. *The Journal of Machine Learning Research*, 15(1):949–980, 2014.

[9] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[10] S. Mannor, R. Y. Rubinstein, and Y. Gat. The cross entropy method for fast policy search. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 512–519, 2003.

[11] A. Abdolmaleki, D. Simões, N. Lau, L. P. Reis, and G. Neumann. Contextual direct policy search. *Journal of Intelligent & Robotic Systems*, 96(2):141–157, 2019.

[12] V. Tangkaratt, H. van Hoof, S. Parisi, G. Neumann, J. Peters, and M. Sugiyama. Policy search with high-dimensional context variables. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[13] J. Kober and J. Peters. Policy search for motor primitives in robotics. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 21. Curran Associates, Inc., 2008. URL https://proceedings.neurips.cc/paper/2008/file/7647966b7343c29048673252e490f736-Paper.pdf.

[14] F. Stulp and O. Sigaud. Path integral policy improvement with covariance matrix adaptation. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, page 1547–1554, Madison, WI, USA, 2012. Omnipress. ISBN 9781450312851.

[15] F. Stulp and O. Sigaud. Policy improvement methods: Between black-box optimization and episodic reinforcement learning. 2012.

[16] A. Kupcsik, M. P. Deisenroth, J. Peters, L. A. Poha, P. Vadakkepata, and G. Neumann. Model-based contextual policy search for data-efficient generalization of robot skills. *Artificial Intelligence*, 247:415–439, 2017. doi:10.1016/j.artint.2014.11.005. URL http://eprints.lincoln.ac.uk/25774/1/Kupcsik_AIJ_2015.pdf. Impact Factor: 3.333.

[17] O. Celik, D. Zhou, G. Li, P. Becker, and G. Neumann. Specializing versatile skill libraries using local mixture of experts. In *Conference on Robot Learning*, pages 1423–1433. PMLR, 2022.

[18] M. Dalal, D. Pathak, and R. R. Salakhutdinov. Accelerating robotic reinforcement learning via parameterized action primitives. *Advances in Neural Information Processing Systems*, 34: 21847–21859, 2021.

[19] O. Zenkri, N. A. Vien, and G. Neumann. Hierarchical policy learning for mechanical search. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 1954–1960. IEEE, 2022.

[20] S. Bahl, M. Mukadam, A. Gupta, and D. Pathak. Neural dynamic policies for end-to-end sensorimotor learning. *Advances in Neural Information Processing Systems*, 12 2020. URL https://arxiv.org/abs/2012.02788v1.

[21] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann. Probabilistic movement primitives. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper/2013/file/e53a0a2978c28872a4505bdb51db06dc-Paper.pdf.

[22] S. Schaal. *Dynamic Movement Primitives -A Framework for Motor Control in Humans and Humanoid Robotics*, pages 261–280. Springer Tokyo, Tokyo, 2006. ISBN 978-4-431-31381-6. doi:10.1007/4-431-31381-8_23. URL https://doi.org/10.1007/4-431-31381-8_23.

[23] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal. Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors. *Neural Computation*, 25(2), 2013.

[24] F. Brandherm, J. Peters, G. Neumann, and R. Akrour. Learning replanning policies with direct policy search. *IEEE Robotics and Automation Letters*, 4(2):2196–2203, 2019.

[25] M. Ginesi, D. Meli, H. Nakawala, A. Roberti, and P. Fiorini. A knowledge-based framework for task automation in surgery. In *2019 19th International Conference on Advanced Robotics (ICAR)*, pages 37–42. IEEE, 2019.

[26] H. Lee, H. Seo, and H.-G. Kim. Trajectory optimization and replanning framework for a micro air vehicle in cluttered environments. *Ieee Access*, 8:135406–135415, 2020.

[27] G. Li, Z. Jin, M. Volpp, F. Otto, R. Lioutikov, and G. Neumann. Prodmps: A unified perspective on dynamic and probabilistic movement primitives. *arXiv preprint arXiv:2210.01531*, 2022.

[28] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[29] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust Region Policy Optimization. In *Proceedings of Machine Learning Research*, pages 1889–1897, 2015. URL http://proceedings.mlr.press/v37/schulman15.html.

[30] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[31] A. Braylan, M. Hollenbeck, E. Meyerson, and R. Miikkulainen. Frame skip is a powerful parameter for learning to play atari. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[32] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms. In *arXiv preprint*, 2017. URL `http://arxiv.org/abs/1707.06347`.

[33] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[34] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[35] R. Agarwal, M. Schwarzer, P. S. Castro, A. Courville, and M. G. Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 2021.

[36] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

## A Ablation Studies

We conduct ablation studies to evaluate each component's influence on the proposed method that aim to answer the following questions:

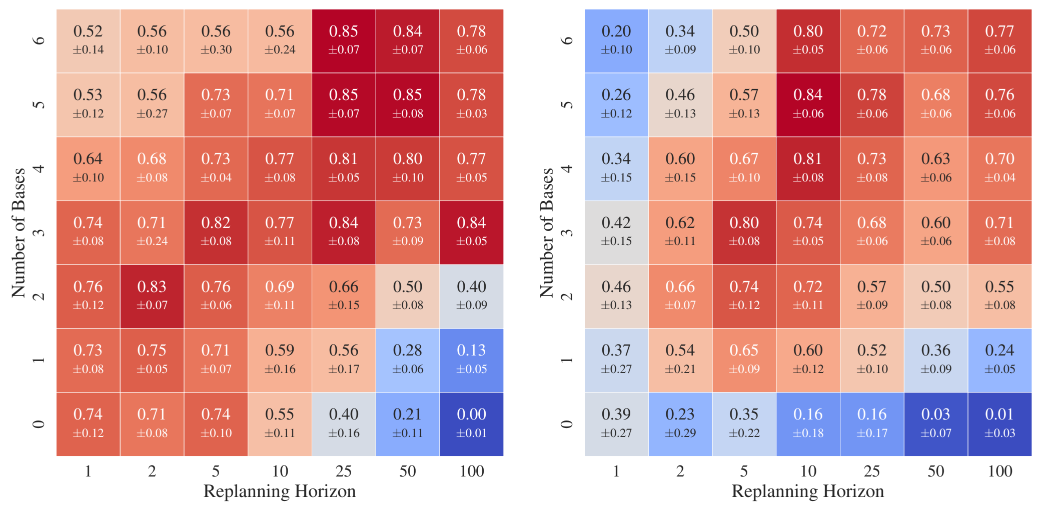
- **Q1** What is the impact of varying the number of bases and the length of the replanning horizon on the performance of MP3?
- **Q2** How does the performance of the non dynamic-based ProMP with replanning compare to MP3?
- **Q3** Can the policy be effectively learned in the parameter space without incorporating proper trust regions?
- **Q4** How does the performance of dynamic-based ProDMPs compare to non dynamic-based ProMPs in MP3-BB setting?

Firstly, we study the correlations between the number of bases of MPs and the length of the planning horizon (replanning steps) in Figure 7. We train agents in box pushing environments with both dense and sparse rewards, using different combinations of planning horizons $k \in \{1, 2, 5, 10, 25, 50, 100\}$ and the number of bases $N \in \{0, 1, 2, 3, 4, 5, 6\}$. A value of 0 for the number of bases indicates that the agent only uses the goal basis of the ProDMPs, leading to the same action space dimension as SRL algorithms. When the planning horizon is equal to 1, the learning objective of replanning reduces to a SRL objective. In this case, the only difference between replanning and SRL is that the agent explores the parameters space of the MP, which usually has a higher dimensionality ((N + 1) $\times$ DoF) compared to the action space that a step-based agent explores. Another special case is when the replanning horizon equals the episode length ($k = 100 = T$), which corresponds to the MP3-BB setting.
**Q1** can be answered according to results in Figure 7. First, planning with a longer horizon requires a greater number of bases to achieve optimal performance. A longer planning horizon means less chance for the agent to adapt trajectories by adjusting the weights, limiting the ability to generate complex trajectories. This limitation, in turn, reduces performance in tasks that require fine manipulation, such as box pushing. Second, longer planning horizons contribute to improved performance in the (temporal) sparse reward setting. This is attributed to the usage of high temporal abstracted samples in the policy updates. However, it does not necessarily mean MP3-BB will always perform better in the sparse reward setting, as the black-box setting lacks the ability to correct its behavior due to the absence of the feedback signals from inter-execution observations.

For **Q2**, we compare replanning with dynamic-based (ProDMPs) and non dynamic-based MPs (ProMPs) in Figures 8a and 8b. The results demonstrate that the policy with dynamic-based MPs yields a policy with a higher success rate and lower control cost. This is largely due to the fact that non-dynamic MPs can result in abrupt transitions between different planning segments, leading to discontinuities in the motion.

To address **Q3**, we evaluate policy search algorithms without trust regions in the replanning setting, and present the results in Figure 8c. In both dense and sparse reward settings of box pushing, MP3 outperforms MP3-PPO in terms of sample efficiency and success rate. The need for a more stable optimization and the higher dimensional nature of learning in parameter spaces could account for this observed improvement.

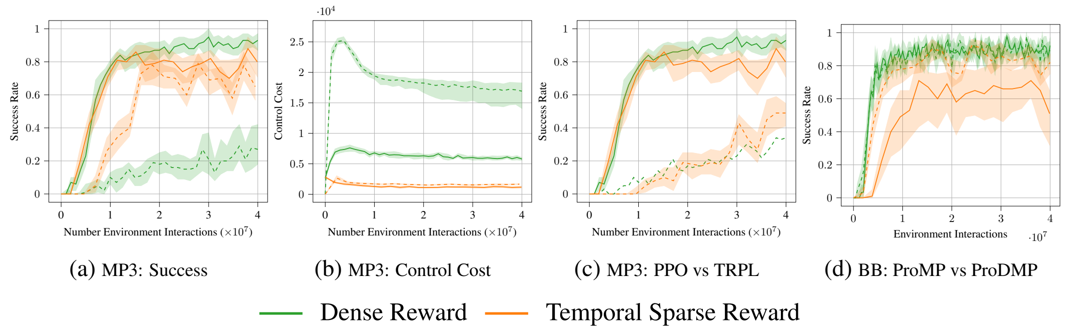
Finally, to answer **Q4**, we compare the performance between the black-box agent with ProDMPs and with ProMPs. The results in Figure 8d show that in dense and sparse reward settings, both algorithms' sample efficiency and success rate are similar. In the sparse reward setting, MP3-BB with ProMPs shows slightly higher asymptotic performance. This difference is due to the different shapes of bases, and we believe the minor performance gap can be mitigated by selecting MP's parameters that minimize the differences in bases. The overall results suggest that the types of MP make no significant difference in the black-box setting.



(a) Median Success Rate for Dense Reward

(b) Median Success Rate for Sparse Reward

Figure 7: This figure shows the median success rate and standard deviation for different configurations of box pushing environments using dense (a) and sparse reward (b), with varying numbers of bases $N$ and replanning horizons $k$. When $N = 0$, the weight bases are disabled, and only the goal basis of the ProDMP is used, with the action space dimension equal to that of SRL. $k = 1$ and $k = 100 = T$ correspond to SRL with MPs and MP3-BB, respectively. We evaluate each combination using ten random seeds and 20 contexts per seed. In general, the results suggest that: 1) longer planning horizons require more bases for optimal performance, 2) long planning horizons can help improve performance in sparse reward settings.



(a) MP3: Success

(b) MP3: Control Cost

(c) MP3: PPO vs TRPL

(d) BB: ProMP vs ProDMP

Figure 8: This figure presents an ablation study for MP3 and MP3-BB with different MPs and learning algorithms. The figures (a) and (b) show the success rate and episode control cost for the box-pushing task, respectively. Here, we compare MP3 with ProDMPs (solid) to MP3 with ProMPs (dashed) to show the need for ProDMPs when replanning. Figure (c) demonstrates the need for using TRPL in MP3 (solid). MP3-PPO with ProDMPs using replanning (dashed) on box pushing tasks is not able to achieve the same performance. Lastly, the figure (d) shows that MP3-BB with ProDMPs (dashed) is performing similarly to the MP3-BB with ProMPs (solid) in dense reward setting, and slightly better in the sparse reward setting.

## B Derivations of probabilistic dynamic movement primitive.

In this section, we will briefly present the main derivations of ProDMPs. We start with the fundamental aspects of DMPs and then derive ProDMPs from the analytical solution of the DMPs' ODE. Finally, we present the solution to a initial value problem of the ODE, which allows us to perform smooth replanning during trajectory execution in a computationally efficient manner. For the sake of simplicity, we introduce the approach by means of a 1-DoF dynamical system. For higher DoF systems, we refer to the original paper by [27].

**dynamic movement primitive** Schaal [22], Ijspeert et al. [23] model a single movement execution as a trajectory $\boldsymbol{\lambda} = [y_t]_{t=0:T}$ using a second-order linear dynamical system with a non-linear forcing function $f$,

$$\tau^2 \ddot{y} = \alpha(\beta(g-y) - \tau \dot{y}) + f(x), \quad f(x) = x\frac{\sum \varphi_i(x) w_i}{\sum \varphi_i(x)} = x\boldsymbol{\varphi}_x^\intercal \boldsymbol{w}, \tag{8}$$

where $y = y(t)$, $\dot{y} = \mathrm{d}y/\mathrm{d}t$, $\ddot{y} = \mathrm{d}^2y/\mathrm{d}t^2$ represent the position, velocity, and acceleration of the system at time step $t$, respectively. $\alpha$ and $\beta$ are spring-damper constants, $g$ is a goal attractor, and $\tau$ is a time constant that can be used to adjust the execution speed of the resulting trajectory. To achieve goal convergence, DMPs define the forcing function based on an exponentially decaying phase variable $x(t) = \exp(-\alpha_x/\tau\ t)$, where $\varphi_i(x)$ represents the (unnormalized) basis functions. The shape of the trajectory as it converges to the goal is controlled by the weights $w_i \in \boldsymbol{w}$, $i = 1...N$. The trajectory of the motion $\boldsymbol{\lambda}$ is obtained by integrating the system numerically from the starting time to the target time point. However, this process is often computationally expensive.

**Solving the dynamic movement primitives' underlying ODE.** Li et al. recognize that the governing equation of DMPs, given in Eq.(8), has an analytical solution, as it is a second-order linear non-homogeneous ODE with constant coefficients. To better convey this method, the ODE and its homogeneous counterpart can be rewritten in a standard form as:

$$\textbf{Non-homo. ODE:} \quad \ddot{y} + \frac{\alpha}{\tau}\dot{y} + \frac{\alpha\beta}{\tau^2}y = \frac{f(x)}{\tau^2} + \frac{\alpha\beta}{\tau^2}g \equiv F(x,g), \tag{9}$$

$$\textbf{Homo. ODE:} \quad \ddot{y} + \frac{\alpha}{\tau}\dot{y} + \frac{\alpha\beta}{\tau^2}y = 0. \tag{10}$$

With appropriate configuration of the spring-damper coefficients, i. e., $\beta = \alpha/4$ ([22, 23]), the system is critically damped and the motion generated by the DMPs will settle to the target position smoothly and efficiently. The analytical solution of Eq. (9) in this case takes the form

$$y = c_1y_1 + c_2y_2 - y_1\int \frac{y_2F}{Y}\mathrm{d}t + y_2\int \frac{y_1F}{Y}\mathrm{d}t, \quad Y = y_1\dot{y}_2 - \dot{y}_1y_2, \tag{11}$$

$$y_1 = y_1(t) = \exp\left(-\frac{\alpha}{2\tau}t\right), \qquad y_2 = y_2(t) = t\exp\left(-\frac{\alpha}{2\tau}t\right), \tag{12}$$

where $y_1$ and $y_2$ are the complementary functions of the homogeneous function in Eq. (10) and $\dot{y}_1$, $\dot{y}_2$ their corresponding derivatives w.r.t. time. By utilizing the fundamental of calculus, which states that $\int h(t)\mathrm{d}t = \int_0^t h(t')\mathrm{d}t' + c$, where $c \in \mathbb{R}$ is a constant, the two indefinite integrals in Eq. (11) can be transformed into two definite integrals. During this transformation, the learnable parameters $\boldsymbol{w}$ and $g$ which control the shape of the trajectory, can be extracted from the resulting definite integrals. Finally, the trajectory position and velocity can be expressed in a compact matrix form as

$$y = c_1y_1 + c_2y_2 + [y_2\boldsymbol{p_2} - y_1\boldsymbol{p_1} \quad y_2q_2 - y_1q_1]\begin{bmatrix}\boldsymbol{w}\\ g\end{bmatrix} \tag{13}$$

$$\dot{y} = c_1\dot{y}_1 + c_2\dot{y}_2 + [\dot{y}_2\boldsymbol{p_2} - \dot{y}_1\boldsymbol{p_1} \quad \dot{y}_2q_2 - \dot{y}_1q_1]\begin{bmatrix}\boldsymbol{w}\\ g\end{bmatrix}, \tag{14}$$

where $\boldsymbol{p}_1$, $\boldsymbol{p}_1$, $q_1$, $q_2$ represent the elements used to formulate the definite integrals in the matrix form, as

$$\boldsymbol{p}_1(t) = \frac{1}{\tau^2}\int_0^t t'\exp\left(\frac{\alpha}{2\tau}t'\right)x(t')\boldsymbol{\varphi}_x^\intercal\mathrm{d}t', \quad \boldsymbol{p}_2(t) = \frac{1}{\tau^2}\int_0^t \exp\left(\frac{\alpha}{2\tau}t'\right)x(t')\boldsymbol{\varphi}_x^\intercal\mathrm{d}t', \tag{15}$$

$$q_1(t) = \left(\frac{\alpha}{2\tau}t - 1\right)\exp\left(\frac{\alpha}{2\tau}t\right) + 1, \quad q_2(t) = \frac{\alpha}{2\tau}\left[\exp\left(\frac{\alpha}{2\tau}t\right) - 1\right]. \tag{16}$$

It is worth noting that, despite the closed form solution for $q_1$ and $q_2$, $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ cannot be obtained analytically because of the complex nature of the $\boldsymbol{\varphi}_x$. As a result, they must be computed numerically. However, the extraction of the learnable parameters $\boldsymbol{w}$ and $g$ from the integrals in Eq. (13) and (14) enables the sharing of the remaining integrals among all trajectories to be generated. In other words, these integrals can be pre-computed offline and used as constant functions during online trajectory computation, which significantly simplifies the trajectory generation procedure and speeds it up. These remaining integrals are referred to as the position basis $\boldsymbol{\Phi}(t)$ and velocity basis $\dot{\boldsymbol{\Phi}}(t)$, and the ProDMPs represent the position and velocity in a similar manner of ProMPs as:

$$y(t) = c_1 y_1(t) + c_2 y_2(t) + \boldsymbol{\Phi}(t)^{\intercal}\boldsymbol{w}_g, \quad \dot{y}(t) = c_1 \dot{y}_1(t) + c_2 \dot{y}_2(t) + \dot{\boldsymbol{\Phi}}(t)^{\intercal}\boldsymbol{w}_g, \tag{3.1}$$

where $\boldsymbol{w}_g$ is a concatenation vector containing $\boldsymbol{w}$ and $g$.

**Solve the initial value problem.** To compute the coefficients $c_1$ and $c_2$, a solution to the initial value problem represented by the Eq.(3.1) must be found. Li et al. suggest using the current robot state, which consists of the robot's position and velocity $(y_b, \dot{y}_b)$ at the replanning time step $t_b$, as the natural condition for ensuring a smooth transition between the previous and newly generated trajectory. We denote the values of the complementary functions and their derivatives at time $t_b$ as $y_{1_b}, y_{2_b}, \dot{y}_{1_b} \dot{y}_{2_b}$, and the values of the position and velocity basis functions as $\boldsymbol{\Phi}_b$, $\dot{\boldsymbol{\Phi}}_b$. By substituting these values into Eq.(3.1), $c_1$ and $c_2$ can be calculated as:

$$\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} \frac{\dot{y}_{2_b} y_b - y_{2_b} \dot{y}_b}{y_{1_b} \dot{y}_{2_b} - y_{2_b} \dot{y}_{1_b}} + \frac{y_{2_b} \dot{\boldsymbol{\Phi}}_b^{\intercal} - \dot{y}_{2_b} \boldsymbol{\Phi}_b^{\intercal}}{y_{1_b} \dot{y}_{2_b} - y_{2_b} \dot{y}_{1_b}} \boldsymbol{w}_g \\ \frac{y_{1_b} \dot{y}_b - \dot{y}_{1_b} y_b}{y_{1_b} \dot{y}_{2_b} - y_{2_b} \dot{y}_{1_b}} + \frac{\dot{y}_{1_b} \boldsymbol{\Phi}_b^{\intercal} - y_{1_b} \dot{\boldsymbol{\Phi}}_b^{\intercal}}{y_{1_b} \dot{y}_{2_b} - y_{2_b} \dot{y}_{1_b}} \boldsymbol{w}_g \end{bmatrix}. \tag{17}$$

## C Trust Region Projection Layers with KL-Divergence

As already mentioned in the main text, TRPLs [5] present a scalable and mathematically sound approach for enforcing trust regions in step-based deep RL. The layer takes the output of a standard Gaussian policy as input in terms of mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\Sigma}$ and projects it into the trust region if the given mean and variance violate their respective bounds. This projection is done for each input state individually. Subsequently, the projected Gaussian policy distribution with parameters $\tilde{\boldsymbol{\mu}}$, $\tilde{\boldsymbol{\Sigma}}$ is used for any further steps, e. g. for sampling and/or loss computation. Formally, the layer solves the following two optimization problems for each state $\boldsymbol{s}$

$$\arg\min_{\tilde{\boldsymbol{\mu}}_s} d_{\text{mean}}\left(\tilde{\boldsymbol{\mu}}_s, \boldsymbol{\mu}(s)\right), \quad \text{s.t.} \quad d_{\text{mean}}\left(\tilde{\boldsymbol{\mu}}_s, \boldsymbol{\mu}_{\text{old}}(s)\right) \leq \epsilon_{\boldsymbol{\mu}}, \quad \text{and} \tag{18}$$

$$\arg\min_{\tilde{\boldsymbol{\Sigma}}_s} d_{\text{cov}}\left(\tilde{\boldsymbol{\Sigma}}_s, \boldsymbol{\Sigma}(\boldsymbol{s})\right), \quad \text{s.t.} \quad d_{\text{cov}}\left(\tilde{\boldsymbol{\Sigma}}_s, \boldsymbol{\Sigma}_{\text{old}}(\boldsymbol{s})\right) \leq \epsilon_{\Sigma}, ` \tag{19}$$

where $\tilde{\boldsymbol{\mu}}_s$ and $\tilde{\boldsymbol{\Sigma}}_s$ are the optimization variables for input state $\boldsymbol{s}$ and $\epsilon_\mu$ and $\epsilon_\Sigma$ are the trust region bounds for mean and covariance, respectively. Finally, $\mu_{\text{old}}$ and $\Sigma_{\text{old}}$ are the reference mean and covariance for the trust region and $d_{\text{mean}}$ as well as $d_{\text{cov}}$ are the similarity metrics for the mean and covariance of a decomposable distance or divergence measure. As we only leverage the KL-divergence projection, we will provide only details for this particular projection below. For the other two projections we refer the reader to Otto et al. [5].

Inserting the mean part of the Gaussian KL divergence into Equation 18 yields

$$\arg\min_{\tilde{\mu}} \left(\mu - \tilde{\mu}\right)^{\mathrm{T}} \Sigma_{\text{old}}^{-1} \left(\mu - \tilde{\mu}\right) \quad \text{s.t.} \quad \left(\mu_{\text{old}} - \tilde{\mu}\right)^{\mathrm{T}} \Sigma_{\text{old}}^{-1} \left(\mu_{\text{old}} - \tilde{\mu}\right) \leq \epsilon_\mu .$$

After differentiating the dual w.r.t. $\tilde{\mu}$, we can solve for the projected mean

$$\tilde{\mu} = \frac{\mu + \omega \mu_{\text{old}}}{1 + \omega} \quad \text{with} \quad \omega = \sqrt{\frac{\left(\mu_{\text{old}} - \mu\right)^{\mathrm{T}} \Sigma_{\text{old}}^{-1} \left(\mu_{\text{old}} - \mu\right)}{\epsilon_\mu}} - 1,$$

leveraging the optimal Lagrange multiplier $\omega$. Similarly, we can insert the covariance part of the Gaussian KL divergence into Equation 19, which results in

$$\arg\min_{\tilde{\Sigma}} \operatorname{tr}\left(\Sigma^{-1}\tilde{\Sigma}\right) + \log\frac{|\Sigma|}{|\tilde{\Sigma}|}, \quad \text{s.t.} \quad \operatorname{tr}\left(\Sigma_{\text{old}}^{-1}\tilde{\Sigma}\right) - d + \log\frac{|\Sigma_{\text{old}}|}{|\tilde{\Sigma}|} \leq \epsilon_\Sigma,$$

where $d$ is the number of degrees of freedom (DoF). Once again, differentiating and solving the dual $g(\eta)$ for the projected covariance yields

$$\tilde{\Sigma} = \left(\frac{\eta^* \Sigma_{\text{old}}^{-1} + \Sigma^{-1}}{\eta^* + 1}\right)^{-1} \quad \text{with} \quad \eta^* = \arg\min_\eta g(\eta), \text{ s.t. } \eta \geq 0.$$

Here, the the optimal Lagrange multiplier $\eta^*$ cannot be computed in closed form, however, a standard numerical optimizer, such as BFGS, is able to efficiently find it. This can be made differentiable by taking the differentials of the KKT conditions of the dual. For more details, we refer to the original work [5].

## D Environment Details

### D.1 Reacher5d

For the Reacher task we modify the original OpenAI gym Reacher-v2 by adding three additional joints, resulting in a total of five joints. The task goal is still to minimize the distance between the goal point $\mathbf{p}_{goal}$ and the end-effector $\mathbf{p}$. We, however, only sample the goal point for $y \geq 0$, i. e. in the first two quadrants, to slightly reduce task complexity while maintaining the increased control complexity. The observation space remains unchanged, unless for the sparse reward where

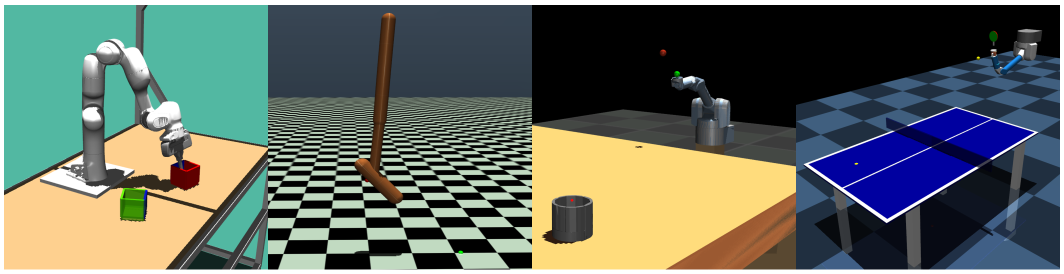


Figure 9: Visualization of the four control tasks box pushing, hopper jumping, beer pong, and table tennis.

we additionally add the current step value to make learning possible for step-based methods. The context space only contains the coordinates of the goal position. The action space is the 5d equivalent to the original version.

For the reward the original setting leverages the goal distance

$$R_{\text{goal}} = \|\mathbf{p} - \mathbf{p}_{goal}\|_2$$

and the action cost

$$\tau_t = \sum_i^K (a_t^i)^2,$$

**Dense Reward.** The dense reward in the 5d setting, hence, stays the same and the agent receives in each time step $t$

$$R_{\text{tot}} = -\tau_t - R_{\text{goal}}$$

**Sparse Reward.** The sparse reward only returns the task reward in the last time step $T$ and additionally adds a velocity penalty $R_{\text{vel}} = \sum_i^K (\dot{q}_T^i)^2$, where $\dot{\mathbf{q}}$ are the joint velocities, to avoid overshooting

$$R_{\text{tot}} = \begin{cases} -\tau_t & t < T, \\ -\tau_t - 200R_{\text{goal}} - 10R_{\text{vel}} & t = T. \end{cases}$$

## D.2 Box Pushing

The goal of the box-pushing task is to move a box to a specified goal location and orientation using the seven DoF Franka Emika Panda. Hence, the context space for this task is the goal position $x \in [0.3, 0.6]$, $y \in [-0.45, 0.45]$ and the goal orientation $\theta \in [0, 2\pi]$. In addition to the contexts, the observation space for the step-based algorithms contains the positions and velocities of the joint angles, as well as position and orientation quaternions for the actual box and the target. For the action space we use the torques per joint and additionally add gravity compensation in each time step, that does not have to be learnt. The task is considered successfully solved if the position distance $\leq$ 0.05m and the orientation error $\leq$ 0.5rad. For the total reward we consider different sub-rewards. First, the distance to the goal

$$R_{\text{goal}} = \|\mathbf{p} - \mathbf{p}_{goal}\|,$$

where $\mathbf{p}$ is the box position and $\mathbf{p}_{goal}$ the goal position itself. Second, the rotation distance

$$R_{\text{rotation}} = \frac{1}{\pi} \arccos |\mathbf{r} \cdot \mathbf{r}_{goal}|,$$

where $\mathbf{r}$ and $\mathbf{r}_{goal}$ are the box orientation and goal orientation in quaternion, respectively. Third, an incentive to keep the rod within the box

$$R_{\text{rod}} = \text{clip}(||\mathbf{p} - \mathbf{h}_{pos}||, 0.05, 10)$$

where $\mathbf{h}_{pos}$ is the position of the rod tip. Fourth, a similar incentive that encourages to maintain the rod in a desired rotation

$$R_{\text{rod\_rotation}} = \text{clip}(\frac{2}{\pi} \arccos |\mathbf{h}_{rot} \cdot \mathbf{h}_0|, 0.25, 2),$$

where $\mathbf{h}_{rot}$ and $\mathbf{h}_0 = (0.0, 1.0, 0.0, 0.0)$ are the current and desired rod orientation in quaternion, respectively. And lastly, we utilize the following error

$$\text{err}(\mathbf{q}, \dot{\mathbf{q}}) = \sum_{i \in \{i | |q_i| > |q_i^b|\}} (|q_i| - |q_i^b|) + \sum_{j \in \{j | |\dot{q}_j| > |\dot{q}_j^b|\}} (|\dot{q}_j| - |\dot{q}_j^b|).$$

Here, $\mathbf{q}$, $\dot{\mathbf{q}}$, $\mathbf{q}^b$, and $\dot{\mathbf{q}}^b$ are the robot joint's position and velocity as well as their respective bounds. Additionally, we consider an action cost in each time step $t$

$$\tau_t = \sum_i^K (a_t^i)^2,$$

where $K = 7$ is the number of DoF. Similar to the aforementioned reacher task, we consider both dense and sparse reward setups.

**Dense Reward.** The dense reward provides information about the goal and rotation distance in each time step $t$ on top of the utility rewards

$$R_{\text{tot}} = -R_{\text{rod}} - R_{\text{rod\_rotation}} - 5e^{-4}\tau_t - \text{err}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - 3.5R_{\text{goal}} - 2R_{\text{rotation}}.$$

**Temporal Sparse Reward.** The time-dependent sparse reward is similar to the dense reward, but only returns the goal and rotation distance in the last time step $T$

$$R_{\text{tot}} = \begin{cases} -R_{\text{rod}} - R_{\text{rod\_rotation}} - 0.02\tau_t - \text{err}(\mathbf{q}, \dot{\mathbf{q}}), & t < T, \\ -R_{\text{rod}} - R_{\text{rod\_rotation}} - 0.02\tau_t - \text{err}(\mathbf{q}, \dot{\mathbf{q}}) - 350R_{\text{goal}} - 200R_{\text{rotation}}, & t = T. \end{cases}$$

**Goal Switching.** To demonstrate the ability of our algorithm to handle the changing goal. We randomly switch to a new target at $20\%$ of the max episode length. To ensure that the new target is solvable within the given episode length, we sample its position near to the old target position. The new target position, denoted as $[x_{new}, y_{new}]$, is computed as follows:

$$[x_{new}, y_{new}] = [x_{old}, y_{old}] + [\Delta x, \Delta y],$$

where $\Delta x, \Delta y$ are randomly sampled within the range $[-0.25, 0.2]$. Additionally, the new target orientation is determined by uniformly sampling a value from the range of $[0, 2\pi]$.

### D.3 Hopper Jump

In the Hopper jump task the agent has to learn to jump as high as possible and land on a certain goal position at the same time. We consider five basis functions per joint resulting in an 15 dimensional weight space. The context is four-dimensional consisting of the initial joint angles $\theta \in [-0.5, 0]$, $\gamma \in [-0.2, 0]$, $\phi \in [0, 0.785]$ and the goal landing position $x \in [0.3, 1.35]$. The full observation space extends the original observation space from the OpenAI gym Hopper by adding the x-value of the goal position and the x-y-z difference between the goal point and the reference point at the Hopper's foot. The action space is the same as for the original Hopper task. We consider a non-Markovian reward function for the episode-based algorithms and a step-based reward for PPO, which we have extensively designed to obtain the highest possible jump.

**Non-Markovian Reward.** In each time-step $t$ we provide an action cost

$$\tau_t = 10^{-3} \sum_i^K (a_t^i)^2,$$

where $K = 3$ is the number of DoF. In the last time-step $T$ of the episode we provide a reward which contains information about the whole episode as

$$\begin{aligned} R_{height} &= 10h_{max}, \\ R_{gdist} &= ||p_{foot,T} - p_{goal}||_2, \\ R_{cdist} &= ||p_{foot,contact} - p_{goal}||_2, \\ R_{healthy} &= \begin{cases} 2 & \text{if } z_T \in [0.5, \infty] \text{and } \theta, \gamma, \phi \in [-\infty, \infty] \\ 0 & \text{else} \end{cases}, \end{aligned}$$

where $h_{max}$ is the maximum jump height in z-direction of the center of mass reached during the whole episode, $p_{foot,t}$ is the x-y-z position of the foot's heel at time step $t$, $p_{foot,contact}$ is the foot's heel position when having a contact with the ground after the first jump, $p_{goal}$ is the goal landing position of the heel. $R_{healthy}$ is a slightly modified reward of the healthy reward defined in the original hopper task. The hopper is considered as 'healthy' if the z position of the center of mass is within the range $[0.5m, \infty]$. This encourages the hopper to stand at the end of the episode. Note that all states need to be within the range $[-100, 100]$ for $R_{healthy}$. Since this is defined in the hopper task from OpenAI already, we haven't mentioned it here. The total reward at the end of an episode is given as

$$R_{tot} = -\sum_{t=0}^{T} \tau_t + R_{height} + R_{gdist} + R_{cdist} + R_{healthy}.$$

**Step-Based Reward.** We consider a step-based alternative reward such that PPO is also able to learn a meaningful behavior on this task. We have tuned the reward such that we can obtain the best performance. The observation space is the same as in the original hopper task from OpenAI extended with the goal landing position and the current distance of the foot's heel and the goal landing postion. We again consider the action cost in each time-step $t$

$$\tau_t = 10^{-3} \sum_{i}^{K} (a_t^i)^2,$$

and additionally consider the rewards

$$\begin{aligned} R_{height,t} &= 3h_t \\ R_{gdist,t} &= 3||p_{foot,t} - p_{goal}||_2 \\ R_{healthy,t} &= \begin{cases} 1 & \text{if } z_t \in [0.5, \infty] \text{and } \theta, \gamma, \phi \in [-\infty, \infty] \\ 0 & \text{else} \end{cases}, \end{aligned}$$

where these rewards are now returned to the agent in each time-step $t$, resulting in the reward per time-step

$$r_t(s_t, a_t) = -\tau_t + R_{height,t} + R_{gdist,t} + R_{healthy,t}.$$

## D.4 Beer Pong

In the Beer Pong task the $K = 7$ Degrees of Freedom (DoF) robot has to throw a ball into a cup on a big table. The context is defined by the cup's two dimensional position on the table which lies in the range $x \in [-1.42, 1.42]$, $y \in [-4.05, -1.25]$. For the step-based algorithms we consider cosine and sine of the robot's joint angles, the angle velocities, the ball's distance to the bottom of the cup, the ball's distance to the top of the cup, the cup position and the current time step. The action space for the step-based algorithms is defined as the torques for each joint, the parameter space for the episode-based methods is 15 dimensional which consists of the two weights for the basis functions per joint and the duration of the throwing trajectory, i.e. the ball release time.

We generally consider action penalties

$$\tau_t = \frac{1}{K} \sum_{i}^{K} (a_t^i)^2,$$

consisting of the sum of squared torques per joint. For $t < T$ we consider the reward

$$r_t(s_t, a_t) = -\alpha_t \tau_t,$$

with $\alpha_t = 10^{-2}$. For $t = T$ we consider the non-Markovian reward

$$R_{task} = \begin{cases} -4 - min(||p_{c,top} - p_{b,1:T}||_2^2) - 0.5||p_{c,bottom} - p_{b,T}||_2^2 \cdots & \\ \cdots - 2||p_{c,bottom} - p_{b,k}||_2^2 - \alpha_T \tau, & \text{if cond. 1} \\ -4 - min(||p_{c,top} - p_{b,1:T}||_2^2) - 0.5||p_{c,bottom} - p_{b,T}||_2^2 - \alpha_T \tau, & \text{if cond. 2} \\ -2 - min(||p_{c,top} - p_{b,1:T}||_2^2) - 0.5||p_{c,bottom} - p_{b,T}||_2^2 - \alpha_T \tau, & \text{if cond. 3} \\ -||p_{c,bottom} - p_{b,T}||_2^2 - \alpha_T \tau, & \text{if cond. 4} \end{cases}$$

$$R_{task} = \left\{ \begin{array}{ll} -4 - min(||p_{c,top} - p_{b,1:T}||_2^2) - 0.5||p_{c,bottom} - p_{b,T}||_2^2 \cdots & \\ \cdots - 2||p_{c,bottom} - p_{b,k}||_2^2 - \alpha_T \tau, & \text{if cond. 1} \\ -4 - min(||p_{c,top} - p_{b,1:T}||_2^2) - 0.5||p_{c,bottom} - p_{b,T}||_2^2 - \alpha_T \tau, & \text{if cond. 2} \\ -2 - min(||p_{c,top} - p_{b,1:T}||_2^2) - 0.5||p_{c,bottom} - p_{b,T}||_2^2 - \alpha_T \tau, & \text{if cond. 3} \\ -||p_{c,bottom} - p_{b,T}||_2^2 - \alpha_T \tau, & \text{if cond. 4} \end{array} \right. ,$$

where $p_{c,top}$ is the position of the top edge of the cup, $p_{c,bottom}$ is the ground position of the cup, $p_{b,t}$ is the position of the ball at time point $t$, and $\tau$ is the squared mean torque over all joints during one rollout and $\alpha_T = 10^{-4}$. The different conditions are:

- cond. 1: The ball had a contact with the ground before having a contact with the table.
- cond. 2: The ball is not in the cup and had no table contact
- cond. 3: The ball is not in the cup and had table contact
- cond. 4: The ball is in the cup.

Note that $p_{b,k}$ is the ball's and the ground's contact position and is only given, if the ball had a contact with the ground first.

At time step $t = T$ we also give information whether the agent's chosen ball release time $B$ was reasonable

$$R_{release} = \left\{ \begin{array}{ll} -30 - 10(B - B_{min})^2, & \text{if } B < B_{min} \\ -30 - 10(B - B_{max})^2, & \text{if } B < B_{max} \end{array} \right. ,$$

where we define $B_{min} = 0.1s$ and $B_{max} = 1s$, such that the agent is encouraged to throw the ball within the time range $[B_{min}, B_{max}]$.

The total return over the whole episode is therefore given as

$$R_{tot} = \sum_{t=1}^{T-1} r_t(s_t, a_t) + R_{task} + R_{release}$$

A throw is considered as successful if the ball is in the cup at the end of an episode.

## D.5 Table Tennis

We consider table tennis for the entire table, i. e. incoming balls are anywhere on the side of the robot and goal locations anywhere on the opponents side. The goal is to use the 7 degree of freedoms (DoFs) robotic arm to hit the incoming ball based on its landing position and return it as close as possible to the specified goal location. As context space we consider the initial ball position $x \in [-1, -0.2]$, $y \in [-0.65, 0.65]$ and the goal position $x \in [-1.2, -0.2]$, $y \in [-0.6, 0.6]$. The full observation space again contains the positions and velocities of the joints on top of the above context information. The torques of the joints make up the action space. For this experiment, we do not use any gravity compensation and allow in the episode-based setting to learn the start time

$t_0$ and the trajectory duration $T$. The task is considered successful if the returned ball lands on the opponent's side of the table and within $\leq 0.2$m to the goal location. The max episode length of the table tennis environment is 350 steps. However, to accelerate the simulation, the episode will end immediately if any of the following terminated conditions are met:

- terminated cond. 1: A contact between the ball and the floor is detected,
- terminated cond. 2: The agent has hit the ball and then a contact between the ball and the table is detected.

The reward signal in the table tennis environment is defined as

$$r_{task} = \begin{cases} 0, & \text{if cond. 1} \\ 0.2 - 0.2\tanh\left(\min ||\mathbf{p}_r - \mathbf{p}_b||^2\right), & \text{if cond. 2} \\ 3 - 2\tanh\left(\min ||\mathbf{p}_r - \mathbf{p}_b||^2\right) - \tanh\left(||\mathbf{p}_l - \mathbf{p}_{goal}||^2\right), & \text{if cond. 3} \\ 6 - 2\tanh\left(\min ||\mathbf{p}_r - \mathbf{p}_b||^2\right) - 4\tanh\left(||\mathbf{p}_l - \mathbf{p}_{goal}||^2\right), & \text{if cond. 4} \\ 7 - 2\tanh\left(\min ||\mathbf{p}_r - \mathbf{p}_b||^2\right) - 4\tanh\left(||\mathbf{p}_l - \mathbf{p}_{goal}||^2\right), & \text{if cond. 5} \end{cases}$$

where $\mathbf{p}_r$ is the position of racket center, $\mathbf{p}_b$ is the position of the ball, $\mathbf{p}_l$ is the ball landing position, $\mathbf{p}_{goal}$ is the target position. The different conditions are

- cond. 1: the end of episode is not reached,
- cond. 2: the end of episode is reached,
- cond. 3: cond.2 is satisfied and robot did hit the ball,
- cond. 4: cond.3 is satisfied and the returned ball landed on the table,
- cond. 5: cond.4 is satisfied and the landing position is at the opponent's side.

The episode ends when any of the following conditions are met

- the maximum horizon length is reached
- ball did land on the floor without hitting
- ball did land on the floor or table after hitting

For BBRL-PPO and BBRL-TRPL, the whole desired trajectory is obtained ahead of environment interaction, making use of this property we can collect some samples without physical simulation. The reward function based on this desired trajectory is defined as

$$r_{traj} = -\sum_{(i,j)} |\tau^d_{ij}| - |q^b_j|, \quad (i,j) \in \{(i,j) \mid |\tau^d_{ij}| > |q^b_j|\}$$

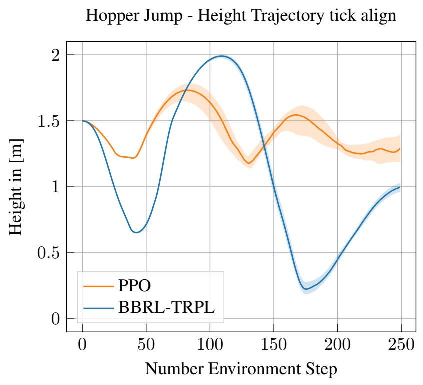
where $\tau^d$ is the desired trajectory, $i$ is the time index, $j$ is the joint index, $q^b$ is the joint position upperbound. The desired trajectory is considered as invalid if $r_{traj} < 0$, an invalid trajectory will not be executed by robot. The overall reward for BBRL is defined as:

$$r = \begin{cases} r_{traj}, & r_{traj} < 0 \\ r_{task}, & \text{otherwise} \end{cases}$$

**Goal Switching.** To evaluate the capability of our approach in handling goal changes in the presence of non-Markovian reward, we designed a goal-switching task based on the table tennis environment. Given that the episode lengths are not fixed in this environment, we fixed the target changing time at the 99-th step after the episode begins. To simplify the task and make it easier to visualize, we restricted the range of the randomly sampled initial target to the left half of the table, specifically $y \in [-0.65, 0], x \in [-1.2, 0.2]$. At the 99-th step, there is 50% of chance that the goal is switched to another random position from the right side of the table, namely $y \in [0, 0.65], x \in [-1.2, 0.2]$.

**Wind as External Perturbation.** To further investigate the performance of our approach in handling environmental perturbations, we introduced artificial wind to the environment. At the beginning of

(a) Hopper Jump - Height Trajectory



(b) 5D Reacher - Sparse

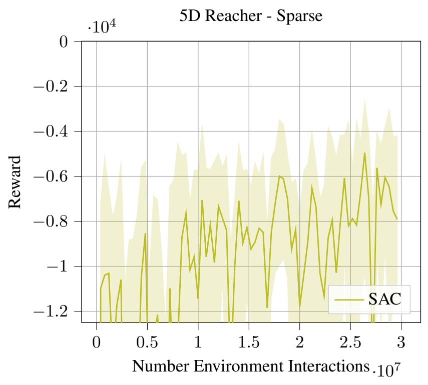


Figure 10: a The improved performance on the Hopper Jump task is also demonstrated on the jumping profile for a fixed context. While MP3-BB jumps once as high as possible, PPO constantly tries to maximize the height at each time step which leads to several jumps throughout the episode and consequently to a lower maximum height. b Learning curve of SAC for the sparse reward of the 5D Reacher task.

each episode, we randomly sample a value $f \in [-0.1, 0.1]$ to represent the constant wind force. This force was then applied as an external force to the ball at each simulation step. It's important to note that, in this specific task, we also augmented the observation space of the agent to include the velocity of the ball. By incorporating this information, the agent was able to infer the underlying "wind speed" and adjust its behavior accordingly. Since this information is not directly observable at the beginning of the episode, episode-based policies, struggled to solve the task.

## E Additional Evaluations

## F Hyperparameters

For all methods, where applicable, we optimized the learning rate, sample size, batch size, number of layers, and the number of epochs. For all MP based methods, we additionally optimized the number of basis functions. Moreover, we found that NDP requires tuning of the scale of the predicted DMP weights, which was hard-coded to 100 in the original code base. However, this value only worked for the Meta-World tasks, but not for the other tasks, hence we adjusted it to allow for a fair comparison. The population size of ES is always half the number of samples because two function evaluations are used per parameter vector.

Table 1: Hyperparameters for the 5D Reacher experiments.

| | PPO | NDP | TRPL | SAC | CMORE | ES | MP3-BB-PPO | MP3-BB |
|---|---|---|---|---|---|---|---|---|
| number samples | 16000 | 16000 | 16000 | 1000 | 120 | 200 | 64 | 64 |
| GAE $\lambda$ | 0.95 | 0.95 | 0.95 | 0.95 | n.a. | n.a. | n.a. | n.a. |
| discount factor | 0.99 | 0.99 | 0.99 | 0.99 | n.a. | n.a. | n.a. | n.a. |
| $\epsilon_\mu/\epsilon$ | n.a. | n.a. | 0.005 | n.a. | 0.1 | n.a. | n.a. | 0.05 |
| $\epsilon_\Sigma$ | n.a. | n.a. | 0.0005 | n.a. | n.a. | n.a. | n.a. | 0.0005 |
| optimizer | adam | adam | adam | adam | n.a. | adam | adam | adam |
| epochs | 10 | 10 | 20 | 1000 | n.a. | n.a. | 100 | 100 |
| learning rate | 3e-4 | 3e-4 | 5e-5 | 3e-4 | n.a. | 1e-2 | 3e-4 | 3e-4 |
| use critic | True | True | True | True | False | False | False | False |
| epochs critic | 10 | 10 | 10 | 1000 | n.a. | n.a. | n.a. | n.a. |
| learning rate critic (and alpha) | 3e-4 | 3e-4 | 3e-4 | 3e-4 | n.a. | n.a. | n.a. | n.a. |
| number minibatches | 32 | 32 | 64 | n.a. | n.a. | n.a. | 1 | 1 |
| batch size | n.a. | n.a. | n.a. | 256 | n.a. | n.a. | n.a. | n.a. |
| buffer size | n.a. | n.a. | n.a. | 1e6 | n.a. | n.a. | n.a. | n.a. |
| learning starts | 0 | 0 | 0 | 10000 | 0 | 0 | 0 | 0 |
| polyak_weight | n.a. | n.a. | n.a. | 5e-3 | n.a. | n.a. | n.a. | n.a. |
| trust region loss weight | n.a. | n.a. | 10 | n.a. | n.a. | n.a. | n.a. | 10 |
| normalized observations | True | True | True | False | False | False | False | False |
| normalized rewards | True | True | False | False | False | False | False | False |
| observation clip | 10.0 | 10.0 | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. |
| reward clip | 10.0 | 10.0 | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. |
| critic clip | 0.2 | 0.2 | n.a. | n.a. | n.a. | n.a. | 0.2 | n.a. |
| importance ratio clip | 0.2 | 0.2 | n.a. | n.a. | n.a. | n.a. | 0.2 | n.a. |
| hidden layers | [32, 32] | [32, 32] | [32, 32] | [128,128] | n.a. | [32, 32] | [32, 32] | [32, 32] |
| hidden layers critic | [32, 32] | [32, 32] | [32, 32] | [128,128] | n.a. | n.a. | n.a. | n.a. |
| hidden activation | tanh | tanh | tanh | relu | n.a. | tanh | tanh | tanh |
| initial std | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| number basis functions | n.a. | 5 | n.a. | n.a. | 5 | n.a. | 5 | 5 |
| number zero basis | n.a. | n.a. | n.a. | n.a. | 1 | n.a. | 1 | 1 |
| weight scale | n.a. | 20 | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. |

Table 2: Hyperparameters for the box pushing experiments.

| | PPO | NDP | TRPL | SAC | ES | MP3 | MP3-BB-PPO | MP3-BB |
|---|---|---|---|---|---|---|---|---|
| number samples | 16000 | 16000 | 16000 | 1000 | 250 | 160 | 160 | 40 |
| GAE $\lambda$ | 0.95 | 0.95 | 0.95 | 0.95 | n.a. | n.a. | n.a. | n.a. |
| discount factor | 1.0 | 0.99 | 1.0 | 0.99 | n.a. | 1.0 | n.a. | n.a. |
| $\epsilon_\mu$ | n.a. | n.a. | 0.005 | n.a. | n.a. | 0.05 | n.a. | 0.05 |
| $\epsilon_\Sigma$ | n.a. | n.a. | 0.00005 | n.a. | n.a. | 0.0005 | n.a. | 0.0005 |
| optimizer | adam | adam | adam | adam | adam | adam | adam | adam |
| epochs | 10 | 10 | 20 | 1000 | n.a. | 20 | 100 | 20 |
| learning rate | 1e-4 | 1e-4 | 5e-5 | 1e-4 | 1e-2 | 3e-4 | 1e-4 | 3e-4 |
| use critic | True | True | True | True | False | True | True | True |
| epochs critic | 10 | 10 | 10 | 1000 | n.a. | 10 | 100 | 10 |
| learning rate critic (and alpha) | 1e-4 | 1e-4 | 2e-4 | 1e-4 | n.a. | 3e-4 | 1e-4 | 3e-4 |
| number minibatches | 40 | 32 | 40 | n.a. | n.a. | 1 | 1 | 1 |
| batch size | n.a. | n.a. | n.a. | 256 | n.a. | n.a. | n.a. | n.a. |
| buffer size | n.a. | n.a. | n.a. | 1e6 | n.a. | n.a. | n.a. | n.a. |
| learning starts | 0 | 0 | 0 | 10000 | 0 | 0 | 0 | 0 |
| polyak_weight | n.a. | n.a. | n.a. | 5e-3 | n.a. | n.a. | n.a. | n.a. |
| trust region loss weight | n.a. | n.a. | 10 | n.a. | n.a. | 10 | n.a. | 10 |
| normalized observations | True | True | True | False | False | False | False | False |
| normalized rewards | True | True | False | False | False | False | False | False |
| observation clip | 10.0 | 10.0 | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. |
| reward clip | 10.0 | 10.0 | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. |
| critic clip | 0.2 | 0.2 | n.a. | n.a. | n.a. | n.a. | 0.2 | n.a. |
| importance ratio clip | 0.2 | 0.2 | n.a. | n.a. | n.a. | n.a. | 0.2 | n.a. |
| hidden layers | [256, 256] | [256, 256] | [256, 256] | [256, 256] | [256, 256] | [128, 128] | [128, 128] | [128, 128] |
| hidden layers critic | [256, 256] | [256, 256] | [256, 256] | [256, 256] | n.a. | [32, 32] | [32, 32] | [32, 32] |
| hidden activation | tanh | tanh | tanh | relu | tanh | relu | tanh | relu |
| initial std | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| MP type | n.a. | DMP | n.a. | n.a. | n.a. | ProDMP | ProMP | ProMP |
| number basis functions | n.a. | 5 | n.a. | n.a. | n.a. | 4 | 5 | 5 |
| number zero basis | n.a. | n.a. | n.a. | n.a. | n.a. | 0 | 1 | 1 |
| $k$ | n.a. | 5 | n.a. | n.a. | n.a. | 25 | 100 | 100 |
| weight scale | n.a. | 10 | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. |

Table 3: Hyperparameters for the Meta-World experiments.

| | PPO | NDP | TRPL | SAC | ES | MP3 | MP3-BB-PPO | MP3-BB |
|---|---|---|---|---|---|---|---|---|
| number samples | 16000 | 16000 | 16000 | 1000 | 200 | 64 | 16 | 16 |
| GAE $\lambda$ | 0.95 | 0.95 | 0.95 | 0.95 | n.a. | 1 | n.a. | n.a. |
| discount factor | 0.99 | 0.99 | 0.99 | 0.99 | n.a. | 1 | n.a. | n.a. |
| $\epsilon_\mu$ | n.a. | n.a. | 0.005 | n.a. | n.a. | 0.075 | n.a. | 0.005 |
| $\epsilon_\Sigma$ | n.a. | n.a. | 0.0005 | n.a. | n.a. | 0.0005 | n.a. | 0.0005 |
| optimizer | adam | adam | adam | adam | adam | adam | adam | adam |
| epochs | 10 | 10 | 20 | 1000 | n.a. | 10 | 100 | 100 |
| learning rate | 3e-4 | 3e-4 | 5e-5 | 3e-4 | 1e-2 | 5e-5 | 3e-4 | 3e-4 |
| use critic | True | True | True | True | False | True | False | False |
| epochs critic | 10 | 10 | 10 | 1000 | n.a. | 10 | n.a. | n.a. |
| learning rate critic (and alpha) | 3e-4 | 3e-4 | 3e-4 | 3e-4 | n.a. | 3e-4 | n.a. | n.a. |
| number minibatches | 32 | 32 | 64 | n.a. | n.a. | 32 | 1 | 1 |
| batch size | n.a. | n.a. | n.a. | 256 | n.a. | n.a. | n.a. | n.a. |
| buffer size | n.a. | n.a. | n.a. | 1e6 | n.a. | n.a. | n.a. | n.a. |
| learning starts | 0 | 0 | 0 | 10000 | 0 | 0 | 0 | 0 |
| polyak_weight | n.a. | n.a. | n.a. | 5e-3 | n.a. | n.a. | n.a. | n.a. |
| trust region loss weight | n.a. | n.a. | 10 | n.a. | n.a. | 10 | n.a. | 10 |
| normalized observations | True | True | True | False | False | True | False | False |
| normalized rewards | True | True | False | False | False | False | False | False |
| observation clip | 10.0 | 10.0 | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. |
| reward clip | 10.0 | 10.0 | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. |
| critic clip | 0.2 | 0.2 | n.a. | n.a. | n.a. | n.a. | 0.2 | n.a. |
| importance ratio clip | 0.2 | 0.2 | n.a. | n.a. | n.a. | n.a. | 0.2 | n.a. |
| hidden layers | [128, 128] | [128, 128] | [128, 128] | [256, 256] | [128, 128] | [256, 256] | [32, 32] | [32, 32] |
| hidden layers critic | [128, 128] | [128, 128] | [128, 128] | [256, 256] | n.a. | [256, 256] | n.a. | n.a. |
| hidden activation | tanh | tanh | tanh | relu | tanh | tanh | tanh | relu |
| initial std | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| MP type | n.a. | DMP | n.a. | n.a. | n.a. | ProDMP | ProMP | ProMP |
| number basis functions | n.a. | 5 | n.a. | n.a. | n.a. | 3 | 5 | 5 |
| number zero basis | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. | 1 | 1 |
| $k$ | n.a. | n.a. | n.a. | n.a. | n.a. | 100 | n.a. | n.a. |
| weight scale | n.a. | 100 | n.a. | n.a. | n.a. | 10 | 10 | 10 |

Table 4: Hyperparameters for the Hopper Jump experiments.

| | PPO | TRPL | SAC | CMORE | ES | MP3-BB-PPO | MP3-BB |
|---|---|---|---|---|---|---|---|
| number samples | 16384 | 16384 | 1000 | 60 | 200 | 64 | 64 |
| GAE $\lambda$ | 0.95 | 0.95 | 0.95 | n.a. | n.a. | n.a. | n.a. |
| discount factor | 0.99 | 0.99 | 0.99 | n.a. | n.a. | n.a. | n.a. |
| $\epsilon_\mu/\epsilon$ | n.a. | 0.005 | n.a. | 0.1 | n.a. | n.a. | 0.005 |
| $\epsilon_\Sigma$ | n.a. | 0.00005 | n.a. | n.a. | n.a. | n.a. | 0.0005 |
| optimizer | adam | adam | adam | n.a. | adam | adam | adam |
| epochs | 10 | 20 | 1000 | n.a. | n.a. | 100 | 100 |
| learning rate | 3e-4 | 5e-5 | 1e-4 | n.a. | 0.01 | 1e-4 | 5e-5 |
| use critic | True | True | True | False | False | False | False |
| epochs critic | 10 | 10 | 1000 | n.a. | n.a. | n.a. | n.a. |
| learning rate critic (and alpha) | 3e-4 | 3e-4 | 1e-4 | n.a. | n.a. | n.a. | n.a. |
| number minibatches | 32 | 64 | n.a. | n.a. | n.a. | 1 | 1 |
| batch size | n.a. | n.a. | 256 | n.a. | n.a. | n.a. | n.a. |
| buffer size | n.a. | n.a. | 1e6 | n.a. | n.a. | n.a. | n.a. |
| learning starts | 0 | 0 | 10000 | 0 | 0 | 0 | 0 |
| polyak_weight | n.a. | n.a. | 5e-3 | n.a. | n.a. | n.a. | n.a. |
| trust region loss weight | n.a. | 10 | n.a. | n.a. | n.a. | n.a. | 25 |
| normalized observations | True | True | False | False | False | False | False |
| normalized rewards | True | False | False | False | False | False | False |
| observation clip | 10.0 | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. |
| reward clip | 10.0 | n.a. | n.a. | n.a. | n.a. | n.a. | n.a. |
| critic clip | 0.2 | n.a. | n.a. | n.a. | n.a. | 0.2 | n.a. |
| importance ratio clip | 0.2 | n.a. | n.a. | n.a. | n.a. | 0.2 | n.a. |
| hidden layers | [128, 128] | [128, 128] | [128, 128] | n.a | [128, 128] | [32, 32] | [32, 32] |
| hidden layers critic | [128, 128] | [128, 128] | [128, 128] | n.a | n.a | n.a | n.a |
| hidden activation | tanh | tanh | relu | n.a. | tanh | tanh | tanh |
| initial std | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| number basis functions | n.a. | n.a. | n.a. | 5 | n.a. | 5 | 5 |
| number zero basis | n.a. | n.a. | n.a. | 1 | n.a. | 1 | 1 |

Table 5: Hyperparameters for the Beer Pong experiments.

| | PPO | CMORE | BBRL-PPO | BBRL-TRPL |
|---|---|---|---|---|
| number samples | 16384 | 60 | 160 | 160 |
| GAE $\lambda$ | 0.95 | n.a. | n.a. | n.a. |
| discount factor | 0.99 | n.a. | n.a. | n.a. |
| $\epsilon_\mu/\epsilon$ | n.a. | 0.1 | n.a. | 0.005 |
| $\epsilon_\Sigma$ | n.a. | n.a. | n.a. | 0.0005 |
| optimizer | adam | n.a. | adam | adam |
| epochs | 10 | n.a. | 100 | 100 |
| learning rate | 3e-4 | n.a. | 1e-4 | 5e-5 |
| use critic | True | False | False | False |
| epochs critic | 10 | n.a. | n.a. | n.a. |
| learning rate critic (and alpha) | 3e-4 | n.a. | n.a. | n.a. |
| number minibatches | 32 | n.a. | 1 | 1 |
| trust region loss weight | n.a. | n.a. | n.a. | 25 |
| normalized observations | True | False | False | False |
| normalized rewards | True | False | False | False |
| observation clip | 10.0 | n.a. | n.a. | n.a. |
| reward clip | 10.0 | n.a. | n.a. | n.a. |
| critic clip | 0.2 | n.a. | 0.2 | n.a. |
| importance ratio clip | 0.2 | n.a. | 0.2 | n.a. |
| hidden layers | [128, 128] | n.a. | [32, 32] | [32, 32] |
| hidden layers critic | [128, 128] | n.a. | n.a. | n.a. |
| hidden activation | tanh | n.a. | tanh | tanh |
| initial std | 1.0 | 1.0 | 1.0 | 1.0 |
| number basis functions | n.a. | 2 | 2 | 2 |
| number zero basis | n.a. | 2 | 2 | 2 |

Table 6: Hyperparameters for the Table Tennis experiments.

| | PPO | TRPL | MP3 | BBRL-PPO | BBRL-TRPL |
|---|---|---|---|---|---|
| number samples | 16000 | 16000 | 360 | 200 | 200 |
| GAE $\lambda$ | 0.95 | 0.95 | n.a. | n.a. | n.a. |
| discount factor | 0.99 | 0.99 | 1.0 | n.a. | n.a. |
| $\epsilon_\mu$ | n.a. | 0.0005 | 0.005 | n.a. | 0.0005 |
| $\epsilon_\Sigma$ | n.a. | 0.00005 | 0.0005 | n.a. | 0.00005 |
| optimizer | adam | adam | adam | adam | adam |
| epochs | 10 | 20 | 20 | 100 | 100 |
| learning rate | 1e-4 | 5e-5 | 2e-4 | 1e-4 | 3e-4 |
| use critic | True | True | True | True | True |
| epochs critic | 10 | 10 | 10 | 100 | 100 |
| learning rate critic (and alpha) | 1e-4 | 1e-4 | 2e-4 | 1e-4 | 3e-4 |
| number minibatches | 40 | 40 | 1 | 1 | 1 |
| trust region loss weight | n.a. | 10.0 | 10 | n.a. | 25 |
| normalized observations | True | True | False | False | False |
| normalized rewards | True | False | False | False | False |
| observation clip | 10.0 | n.a. | n.a. | n.a. | n.a. |
| reward clip | 10.0 | n.a. | n.a. | n.a. | n.a. |
| critic clip | 0.2 | n.a. | n.a. | 0.2 | n.a. |
| importance ratio clip | 0.2 | n.a. | n.a. | 0.2 | n.a. |
| hidden layers | [256, 256] | [256, 256] | [256] | [256] | [256] |
| hidden layers critic | [256, 256] | [256, 256] | [256] | [256] | [256] |
| hidden activation | tanh | tanh | relu | tanh | relu |
| initial std | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| MP type | n.a. | n.a. | ProDMP | ProMP | ProMP |
| number basis functions | n.a. | n.a. | 3 | 3 | 3 |
| number zero basis | n.a. | n.a. | 0 | 1 | 1 |
| $k$ | n.a. | n.a. | 50 | n.a. | n.a. |
| weight scale | n.a. | n.a. | n.a. | n.a. | n.a. |