Investigating the Impact of Action Representations in Policy Gradient Algorithms

Jan Schneider*, Pierre Schumacher*†, Daniel Häufle†‡, Bernhard Schölkopf*, Dieter Büchler*

* Max Planck Institute for Intelligent Systems, Tübingen, Germany
† Hertie-Institute for Clinical Brain Research, Tübingen, Germany
‡ Institute for Modelling and Simulation of Biomechanical Systems, University of Stuttgart, Germany

Abstract—**Reinforcement learning (RL) is a versatile framework for learning to solve complex real-world tasks. However, influences on the learning performance of RL algorithms are often poorly understood in practice. We discuss different analysis techniques and assess their effectiveness for investigating the impact of action representations in RL. Our experiments demonstrate that the action representation can significantly influence the learning performance on popular RL benchmark tasks. The analysis results indicate that some of the performance differences can be attributed to changes in the complexity of the optimization landscape. Finally, we discuss open challenges of analysis techniques for RL algorithms.**

I. INTRODUCTION

Reinforcement learning (RL) holds great potential for learning complex behaviors for solving real-world robotics tasks [1]. However, the performance of RL methods often hinges on seemingly small implementation details [2] and design decisions in the task formulation [3]. These findings indicate that the inner workings of popular RL algorithms are often poorly understood. Recent works tackle this issue by introducing analysis techniques for investigating different aspects of the learning process [4], [5]. In this work, we evaluate and discuss the effectiveness of these analysis methods for investigating the impact of action representations in RL.

Recent results suggest that action representations can have a significant influence on the learning performance of RL algorithms [6]–[9]. However, these works consider only a narrow set of custom-designed tasks. Furthermore, the reasons for the performance differences are still unclear. Understanding the causes of these results would enable designing action spaces that are particularly suited for RL, potentially improving learning efficiency and robustness. Therefore, we first evaluate the performance of RL with different action representations on a variety of well-known benchmark tasks. Subsequently, we apply two analysis techniques from the literature to investigate the causes of performance differences resulting from the choice of action space. Finally, we discuss the benefits and shortcomings of each analysis method in this context.

II. INVESTIGATING THE LEARNING PERFORMANCE FOR DIFFERENT ACTION REPRESENTATIONS

Our analysis focuses on the well-known RL benchmark tasks from OpenAI Gym [10] and the DeepMind Control Suite [11]. These tasks are commonly used for evaluating learning performance in RL, which allows putting our results into context.

We choose Proximal Policy Optimization (PPO) [12] as our backbone RL algorithm since it is conceptually simple and yet one of the most popular RL algorithms.

To avoid the computational expense of tuning hyperparameters for all combinations of action space and task, we use the default hyperparameters from Stable-Baselines3 [13]. While tuned hyperparameters are available for some tasks in [14], these were optimized for the original torque-controlled versions of the tasks. Since changing the action space might also influence which hyperparameters work well for learning, using the tuned hyperparameters would impede a fair comparison.

A. Action spaces

We compare and analyze three action spaces: *torque control*, *velocity control*, and *position control*. These action spaces are widely used in robotics applications and were recently analyzed in the context of RL [6]. Torque control is the default actuation method for the benchmark tasks and requires no further modifications. We implement the velocity- and position-controlled task variants by adding an additional linear controller on top of the original simulated agent. This linear controller takes target joint positions / velocities from the RL agent and computes torques to progress toward this target. Although it is common practice in robotics to operate such low-level controllers at a higher frequency, we execute the RL agent and the low-level controller at the same frequency to ensure a fair comparison with the torque control configuration. In the following, $\mathbf{q}$ and $\dot{\mathbf{q}}$ denote the current joint positions and velocities, respectively. The velocity controller interprets the actions $\mathbf{a}$ of the policy as joint target velocities and computes torques according to the following control law

$$\tau = K_d^{VC}(\mathbf{a} - \dot{\mathbf{q}}), \tag{1}$$

where the controller gain K_d^{VC} is a hyperparameter.

The following proportional-derivative (PD) controller implements the position control

$$\tau = K_p^{PC}(\mathbf{a} - \mathbf{q}) - K_d^{PC}\dot{\mathbf{q}}, \tag{2}$$

where the actions $\mathbf{a}$ are interpreted as joint target positions, and K_p^{PC} and K_d^{PC} are controller gains that determine the influence of the position and velocity terms, respectively. Similar to Peng and van de Panne [6], we set the target velocity of the PD controller to zero to make the controller stabilize the system if it reaches the target position.
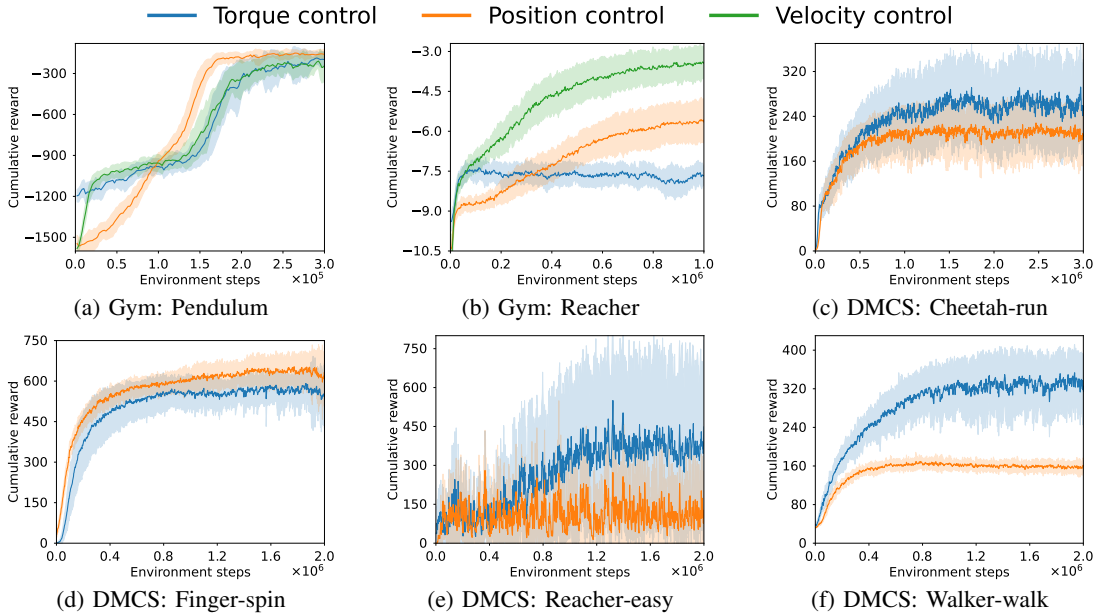
Fig. 1: Learning curves of PPO on benchmark tasks from OpenAI Gym [10] and the DeepMind Control Suite (DMCS) [11] with different action spaces. All results are averaged over 10 random seeds with the shaded area representing the standard deviation over the seeds. For the tasks Pendulum, Reacher, and Walker-walk, the choice of the action space has a significant impact on the learning performance. The performance difference is smaller but still noticeable for the remaining tasks.

When using these action representations for RL, we observe significant differences in learning performance. The learning curves depicted in Figure 1 illustrate this observation. However, there is no action space that works best across all tasks. To understand the causes for the performance differences, we take a closer look at different aspects of the policy optimization and show analysis results for two tasks: Reacher from OpenAI Gym and Walker-walk from the DeepMind Control Suite. The tasks require distinct skills and the learning performance is significantly affected by the choice of action space. We refer to Appendix A for results on the remaining tasks.

B. Visualizing the optimization landscape does not always yield insights into learning performance

Generally, the shape of the optimization surface determines the difficulty of a learning problem. A rugged optimization landscape with many shallow local optima is usually more challenging to navigate than a smooth, convex one. Gradient-based methods, such as policy gradients, are particularly affected by the shape of the optimization surface since they are prone to converge to sub-optimal solutions.

Visual inspection of loss landscapes can provide intuitive insights into the difficulty of an optimization problem. However, due to the large number of parameters in typical neural networks, it is impossible to visualize the loss directly. Li *et al.* [15] propose a dimensionality reduction technique to create two-dimensional visualizations of the loss surface of neural networks. Starting from the current parameters, the technique plots the value of the loss function along two random directions in parameter space. Recently, Sullivan *et al.* [5] and Bekci and Gümüs [16] applied the method to analyze RL algorithms.

Similar to Sullivan *et al.* [5], we discretize the parameter space on a 31×31 grid along the two random directions. For each point on the grid, we create an instance of the agent with parameters corresponding to the grid location. We evaluate the discounted cumulative reward and the PPO loss of each agent with 200,000 samples to obtain good approximations of the true values of these objectives. Even though the visualization method includes a significant amount of stochasticity, we find that plots generated with different random seeds are visually similar, see Appendix B.

PPO does not directly optimize the cumulative reward and operates on a surrogate loss instead. Some works analyze the reward surface [5], while others focus on the loss surface [4], [16]. To get the complete picture, we visualize both the reward and loss surfaces resulting from different action spaces in Figure 2. Figure 2(a4) shows that the loss surface for the Reacher torque control configuration is rugged and has numerous local optima. The difficulty of optimizing such a loss might explain the poor learning performance of the agent in this configuration. For the remaining action representations, the optimization surface appears simpler to optimize. Figure 2(b) does not yield a clear intuition for the performance difference between the two Walker-walk configurations. The loss surfaces are visually very similar, yet the agent performs significantly better in the torque control configuration.

Another interesting observation is the striking discrepancy between the visualizations of the reward surfaces and the loss surfaces. This observation supports the finding in [4] that the loss that PPO optimizes is often poorly correlated with the cumulative reward, which is the true objective in RL.

We showed that optimization surface visualizations can yield intuitive insights into the optimization difficulty for certain

(a1) TC, reward   (a2) PC, reward   (a3) VC, reward   (b1) TC, reward   (b2) PC, reward

(a4) TC, neg. loss   (a5) PC, neg. loss   (a6) VC, neg. loss   (b3) TC, neg. loss   (b4) PC, neg. loss

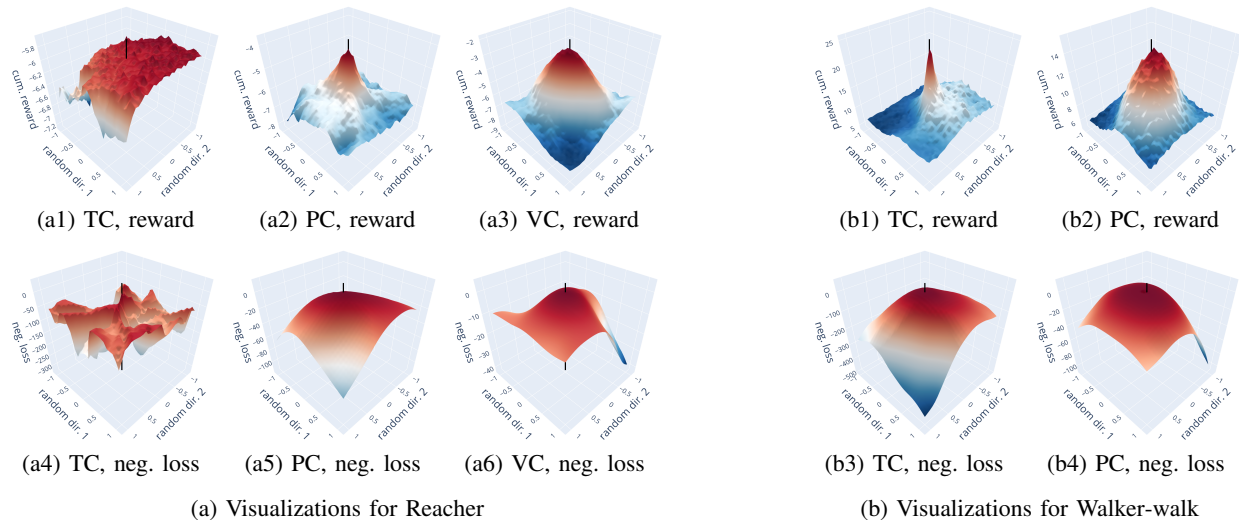(a) Visualizations for Reacher   (b) Visualizations for Walker-walk

Fig. 2: Visualizations of the optimization surface for the Reacher (left) and Walker-walk (right) tasks. All plots are generated by plotting the objective value along two random directions in parameter space, starting from the parameters found after training for 1 million environment steps (marked by the black line). The top row shows the discounted cumulative reward, while the bottom row displays the surrogate loss optimized by PPO (negated for visual clarity). For Reacher, the optimization surface of the torque control (TC) configuration (plots (a1) and (a4)) appears significantly more complex than that of the position control (PC) and velocity control (VC) configurations (plots (a2), (a5) and plots (a3), (a6), respectively). For Walker-walk, the complexity of the optimization surfaces for the two action representations appears similar.

tasks. However, in other cases, the interpretation of the results is not as clear since the method does not produce a clear metric for comparison. Furthermore, this approach is constrained to analyzing a single snapshot of the training at a time. Due to the continual update of the policy, the distribution of collected data changes constantly, which affects the loss landscape. Moreover, the method uses a large number of samples to approximate the true objective. However, during RL training, the optimizer does not directly operate on the true objective but rather uses a coarse approximation generated with a low number of samples. Given these limitations, we describe an alternative analysis method in the following section. This method results in a numerical criterion, which facilitates comparison across different configurations and allows visualizing results for the entire training as a simple line plot.

C. More accurate gradient estimates do not imply better learning performance

In this section, we explore the effects that the low-sample approximation of the loss has on the gradient estimation. The accuracy of the gradient estimates is crucial since it determines the direction of the policy update. Precise gradient estimates allow reaching an optimum of the objective quickly, while noisy estimates can result in the optimization oscillating through the loss landscape without improving the agent's performance. We utilize the analysis method by Ilyas *et al.* [4] to measure and compare the quality of the gradient estimates in tasks with different action spaces. To evaluate the gradient quality throughout the entire learning process, we execute the analysis at 20 checkpoints during the agent's training. For each checkpoint, we approximate the true gradient with 10^7 samples. In contrast, PPO with default hyperparameters uses 64

samples to estimate the gradient during training. We compute the average pairwise cosine similarity between the gradients estimated during training and our approximation of the true gradient. This quantity provides insights into how closely the gradients used during training match the true gradient. Since the PPO loss is the sum of a policy loss and a value function loss, we also execute the analysis on each term individually to measure which of the terms is more problematic for the gradient estimation.

Figure 3 displays the results of the gradient analysis for the Reacher and Walker-walk tasks. The results appear counter-intuitive since the better-performing configurations for Reacher use worse gradients. For Walker-walk, this is also the case for the first 700,000 steps. Afterward, the trend reverses, and the better-performing configuration also uses better gradient estimates. We hypothesize that these unclear results are caused by a more complex relationship between the gradient quality and learning performance. Not only does the gradient estimation influence the learning performance, but the learning performance also affects the gradient estimation. If the optimizer gets close to an optimum, the loss landscape locally becomes relatively flat, and small estimation errors can change the gradient direction drastically. Furthermore, noisy gradient estimates can even be beneficial for escaping shallow local optima [17]. In Appendix C, we demonstrate that training with more accurate gradient estimates does not always yield performance gains.

Another interesting observation in Figure 3 is that the gradient accuracy is significantly worse for the policy loss than for the value function loss. A plausible explanation is the use of importance sampling in the policy loss, which increases the influence of individual samples on the loss term.
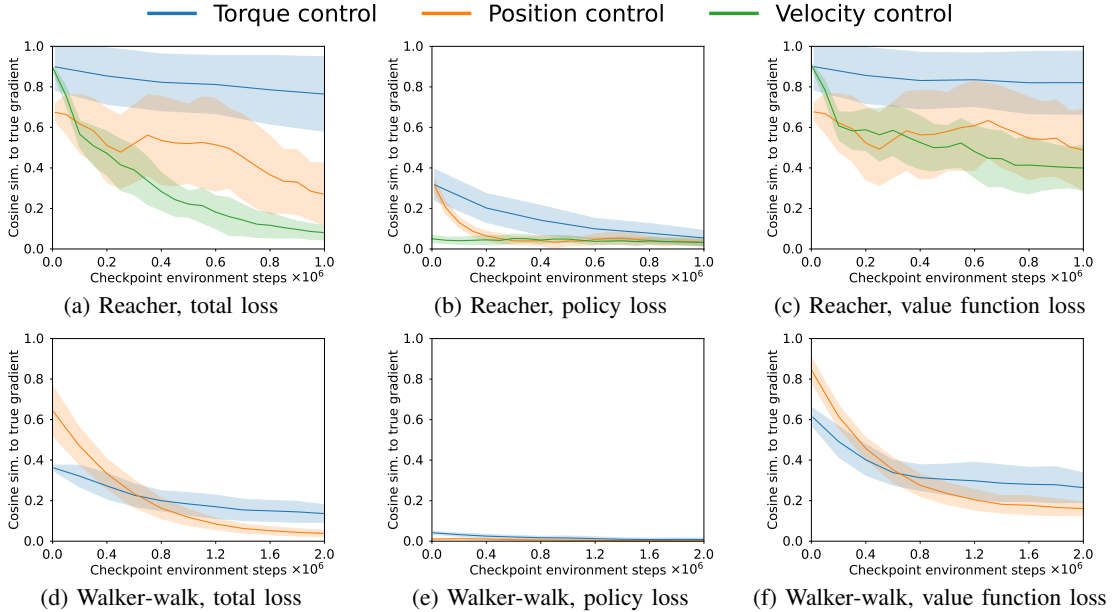
Fig. 3: The quality of the gradient estimates measured as the average cosine similarity to a good approximation of the true gradient, computed with 10^7 samples. All plots show the analysis results at 20 checkpoints during training, averaged over 10 training runs with different random seeds. The top row displays the results on the Reacher task, while the bottom row focuses on Walker-walk. Plots (a) and (d) display the results for the gradients estimated for the entire PPO loss and plots (b), (e) and (c), (f) respectively regard the policy and value function loss individually. For Reacher, the better-performing configurations use worse gradient estimates. For Walker-walk, this trend reverses at around 700,000 steps. Furthermore, the gradient quality is significantly worse for the policy loss than for the value function loss.

III. CONCLUSION

In this work, we demonstrated that action representations can have a significant impact on the learning performance for RL benchmark tasks. We described two analysis methods and evaluated their effectiveness for analyzing the impact of the action representation. Our results suggest that the performance gains for the Reacher task can be attributed to a smoother optimization landscape but this explanation does not seem to transfer to Walker-walk, demonstrating that visualizing the optimization landscape is not always sufficient to gather insights about the learning performance. Furthermore, our evaluation indicates that the accuracy of the gradient estimates does not correlate well with learning performance, which suggests a more complex relationship between these two factors.

Our analysis results are not yet sufficient to explain all observed performance differences. To achieve this goal, the following challenges need to be addressed.

1) Normalizing the analysis results with respect to the learning progress: Properties like the complexity of the optimization landscape and the accuracy of gradient estimates influence how quickly the agent progresses on the objective. However, agents on different levels of performance generally operate in different parts of the state space, which in turn can have an effect on these properties. Proficient agents might operate in regions of the state space that require fine-grained control, which in turn could result in optimization surfaces that are challenging to optimize. To fully understand influences on learning performance, it is crucial to control for these effects.

2) Disentangling different effects on the RL algorithm: Changing the action representation typically affects multiple aspects of the learning process simultaneously. For example, we have seen the influence on both the optimization landscape complexity and the gradient estimation accuracy in Sections II-B and II-C, respectively. Furthermore, such changes could also affect other algorithm components, like the agent's exploration. Since these effects typically occur together, it remains unclear which influences are responsible for the performance differences.

3) Assessing the effects of other task design choices and hyperparameters: Other choices in the task design, like the state representation, also have a significant impact on the learning performance and analysis results. Specifically, there might be choices that are particularly suitable for certain action spaces. In Appendix D, we investigate the effect of defining both the states and actions in the same space. Similarly, the hyperparameters of the learning algorithm and the controller gains influence the results. We tuned the controller gains to minimize the average error over a fixed number of time steps. However, the reinforcement learning algorithm might yield better results with a more aggressive controller since this choice potentially allows to utilize higher torques.

Further understanding the impact of action representations on learning performance enables tuning action representations to make learning more efficient and robust. Beyond the use for optimizing action representations, sophisticated analysis tools could also provide vital insights into obstacles that impede learning progress in practice. These insights could pave the way for designing more sample-efficient and robust RL algorithms.

REFERENCES

[1] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[2] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, "Implementation matters in deep RL: A case study on PPO and TRPO," in *International Conference on Learning Representations*, 2020.

[3] D. Reda, T. Tao, and M. van de Panne, "Learning to locomote: Understanding how environment design matters for deep reinforcement learning," in *Proceedings of the 13th ACM SIGGRAPH Conference on Motion, Interaction and Games*, 2020, pp. 1–10.

[4] A. Ilyas, L. Engstrom, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, "A closer look at deep policy gradients," in *International Conference on Learning Representations*, 2020.

[5] R. Sullivan, J. K. Terry, B. Black, and J. P. Dickerson, "Cliff diving: Exploring reward surfaces in reinforcement learning environments," in *International Conference on Machine Learning*, PMLR, 2022, pp. 20744–20776.

[6] X. B. Peng and M. van de Panne, "Learning locomotion skills using DeepRL: Does the choice of action space matter?" In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2017, pp. 1–13.

[7] P. Varin, L. Grossman, and S. Kuindersma, "A comparison of action spaces for learning manipulation tasks," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2019, pp. 6015–6021.

[8] M. Bogdanovic, M. Khadiv, and L. Righetti, "Learning variable impedance control for contact sensitive tasks," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6129–6136, 2020.

[9] I. Wochner, P. Schumacher, G. Martius, D. Büchler, S. Schmitt, and D. Häufle, "Learning with muscles: Benefits for data-efficiency and robustness in anthropomorphic tasks," in *6th Annual Conference on Robot Learning*, 2022.

[10] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv preprint arXiv:1606.01540*, 2016.

[11] S. Tunyasuvunakool, A. Muldal, Y. Doron, S. Liu, S. Bohez, J. Merel, T. Erez, T. Lillicrap, N. Heess, and Y. Tassa, "dm_control: Software and tasks for continuous control," *Software Impacts*, vol. 6, p. 100022, 2020, ISSN: 2665-9638. DOI: https://doi.org/10.1016/j.simpa.2020.100022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2665963820300099.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[13] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-Baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html.

[14] A. Raffin, *RL Baselines3 Zoo*, https://github.com/DLR-RM/rl-baselines3-zoo, 2020.

[15] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," *Advances in neural information processing systems*, vol. 31, 2018.

[16] R. Y. Bekci and M. Gümüs, "Visualizing the loss landscape of actor critic methods with applications in inventory optimization," *arXiv preprint arXiv:2009.02391*, 2020.

[17] A. Neelakantan, L. Vilnis, Q. V. Le, I. Sutskever, L. Kaiser, K. Kurach, and J. Martens, "Adding gradient noise improves learning for very deep networks," *arXiv preprint arXiv:1511.06807*, 2015.

APPENDIX

A. Analysis results on remaining tasks
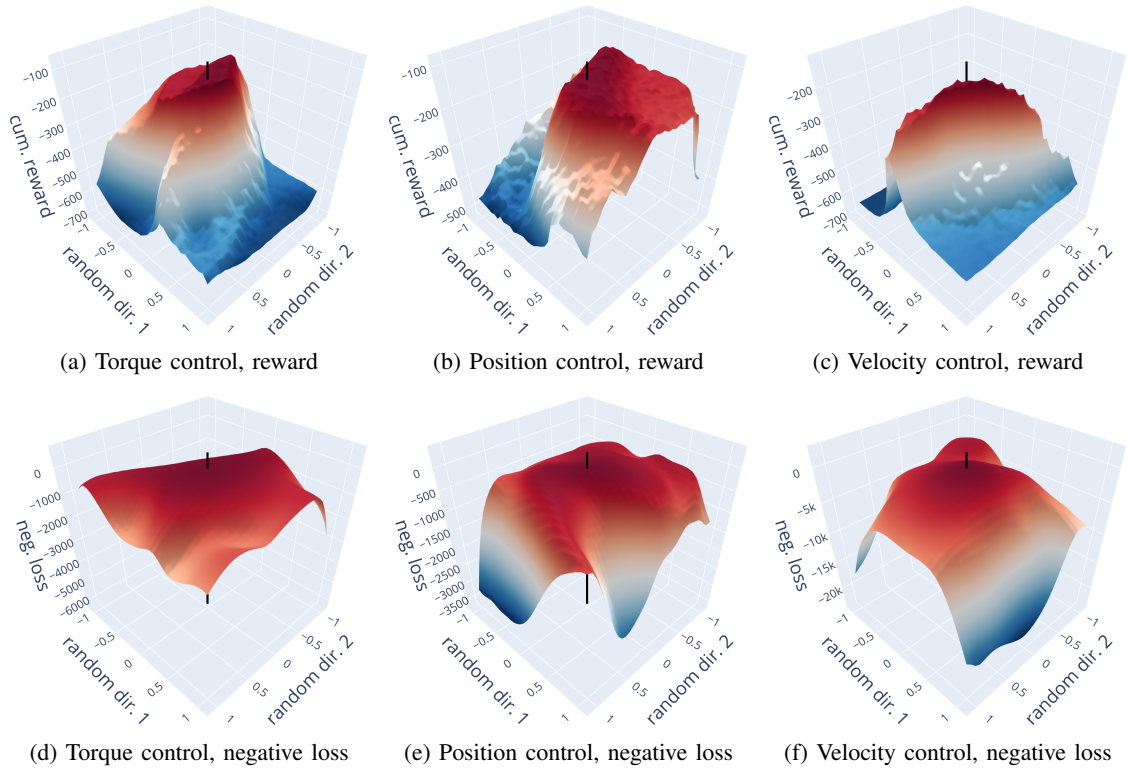
   1) Optimization surface visualization:

(a) Torque control, reward   (b) Position control, reward   (c) Velocity control, reward

(d) Torque control, negative loss   (e) Position control, negative loss   (f) Velocity control, negative loss

Fig. 4: Visualizations of the optimization surface for the Pendulum task.

(a) Torque control, reward   (b) Position control, reward

(c) Torque control, negative loss   (d) Position control, negative loss
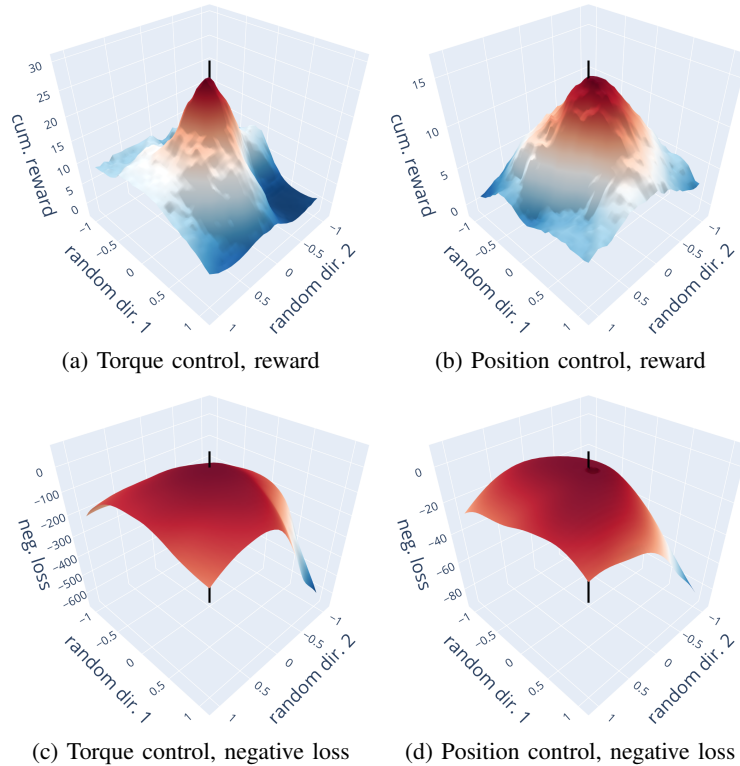
Fig. 5: Visualizations of the optimization surface for the Cheetah-run task.

(a) Torque control, reward

(b) Position control, reward

(c) Torque control, negative loss

(d) Position control, negative loss
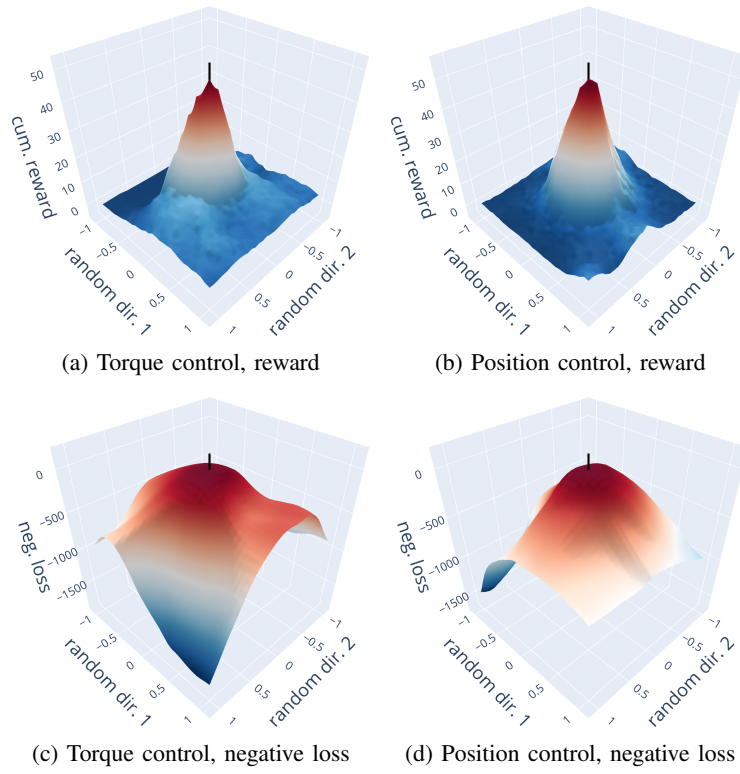
Fig. 6: Visualizations of the optimization surface for the Finger-spin task.

(a) Torque control, reward

(b) Position control, reward
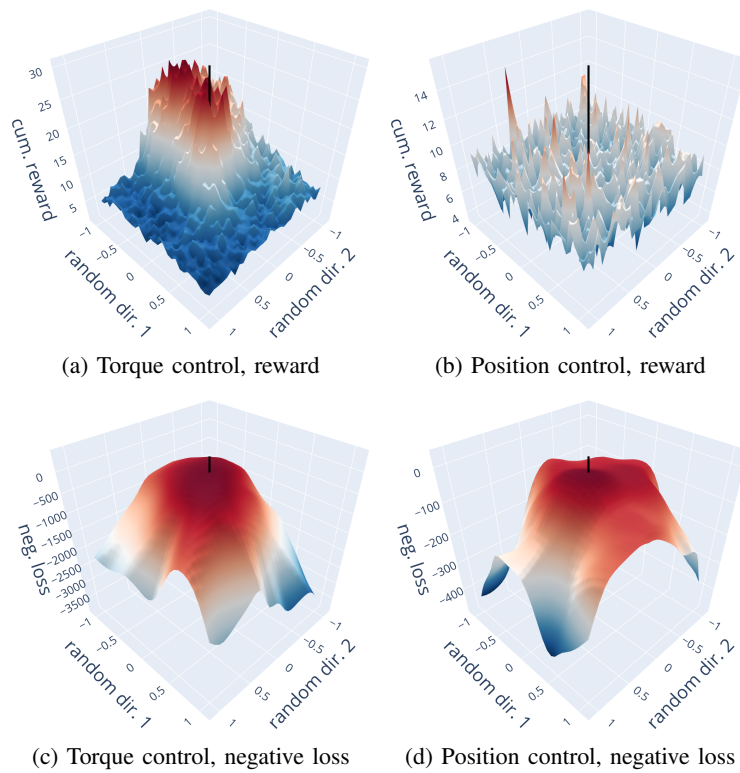
(c) Torque control, negative loss

(d) Position control, negative loss

Fig. 7: Visualizations of the optimization surface for the Reacher-easy task.
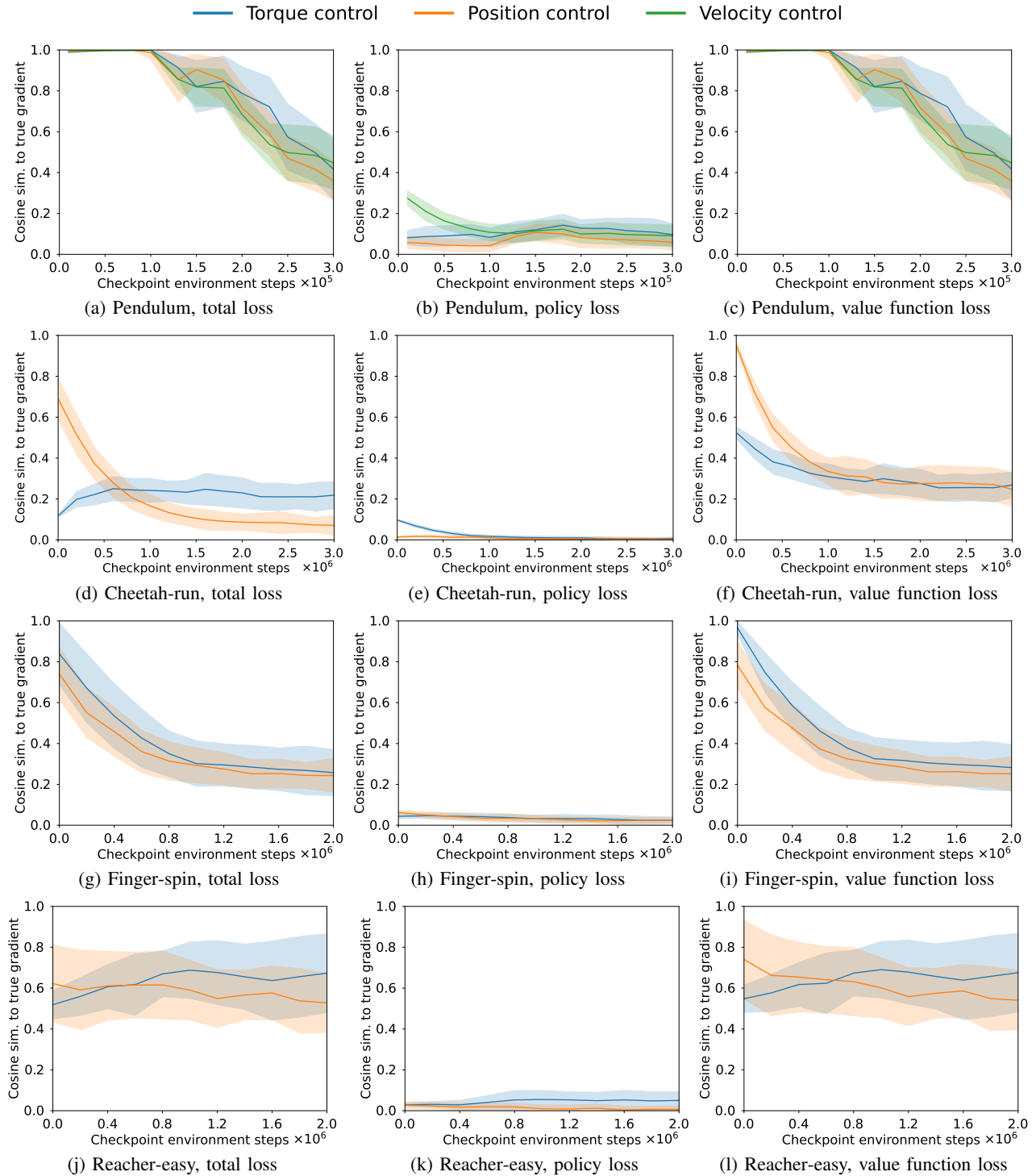
2) Accuracy of the gradient estimates:

Fig. 8: The quality of the gradient estimates measured as the average cosine similarity to a good approximation of the true gradient, computed with 10^7 samples. All plots are averaged over 10 training runs with different random seeds with the shaded region depicting the standard deviation across the training runs.

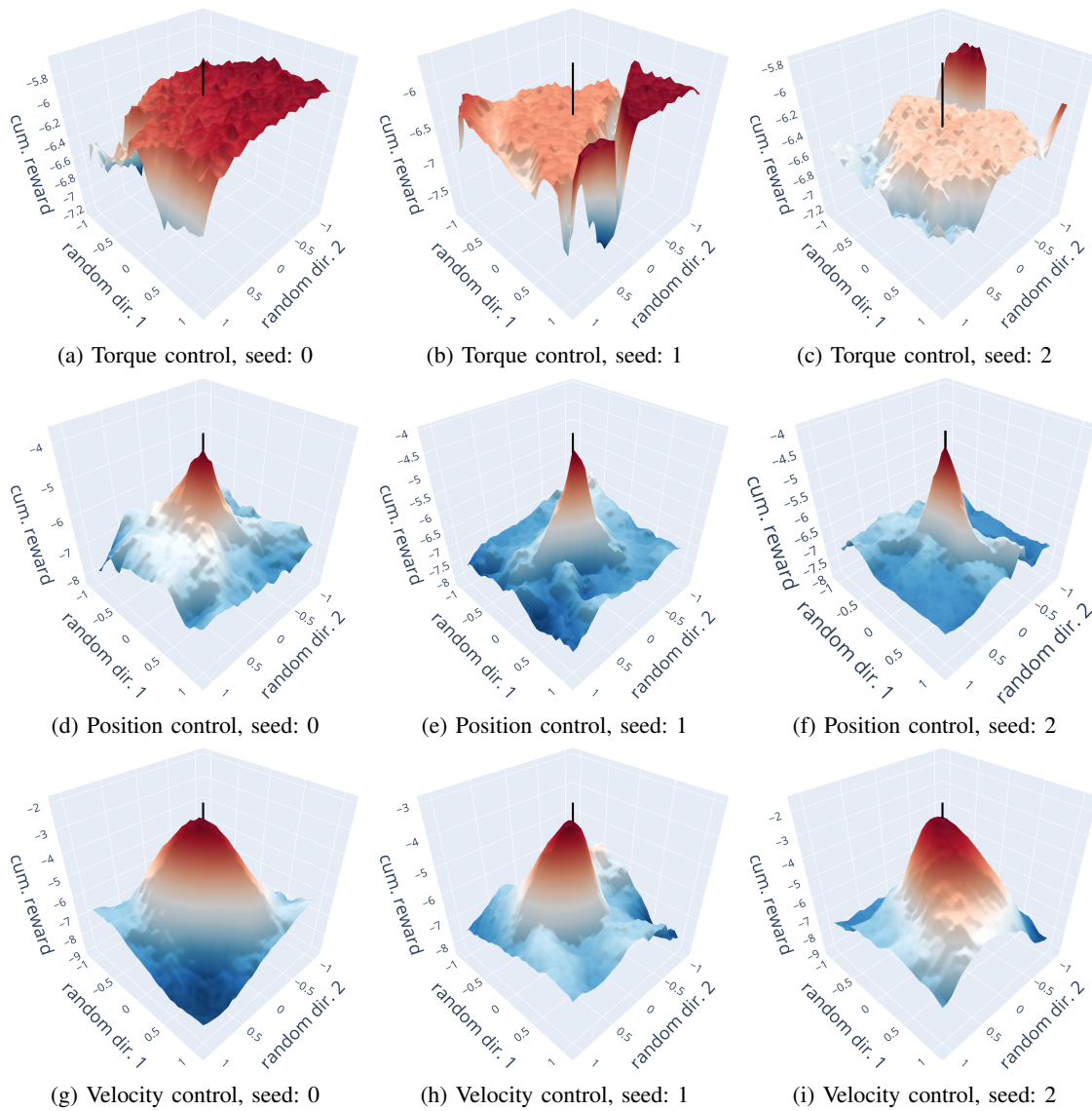*B. Reproducibility of the optimization surface visualizations*

(a) Torque control, seed: 0   (b) Torque control, seed: 1   (c) Torque control, seed: 2

(d) Position control, seed: 0   (e) Position control, seed: 1   (f) Position control, seed: 2

(g) Velocity control, seed: 0   (h) Velocity control, seed: 1   (i) Velocity control, seed: 2

Fig. 9: Reproducibility of the reward surface plots on the Reacher task across training runs with different random seeds. All plots are generated after training for 1 million environment steps. The rows display results for different action representations. Across the columns, the representation is the same but the random seed used for training the agent and generating the random directions varies. Even though the visualization process includes a significant amount of stochasticity, we find that the plots for each action space configuration display similar characteristics.

C. Training with more accurate gradient estimates

In this section, we explore the effect of training with more accurate gradient estimates. Instead of the batches of 64 samples that are commonly used in PPO to predict the policy gradient, we utilize 100,000 samples to get significantly more accurate gradient estimates. The learning curves in Figure 10 indicate that more accurate gradient estimates can significantly accelerate learning and improve the final performance in certain cases. However, the torque control configuration still performs poorly, indicating that the action representation influences factors of the learning process beyond the quality of the gradient estimates.
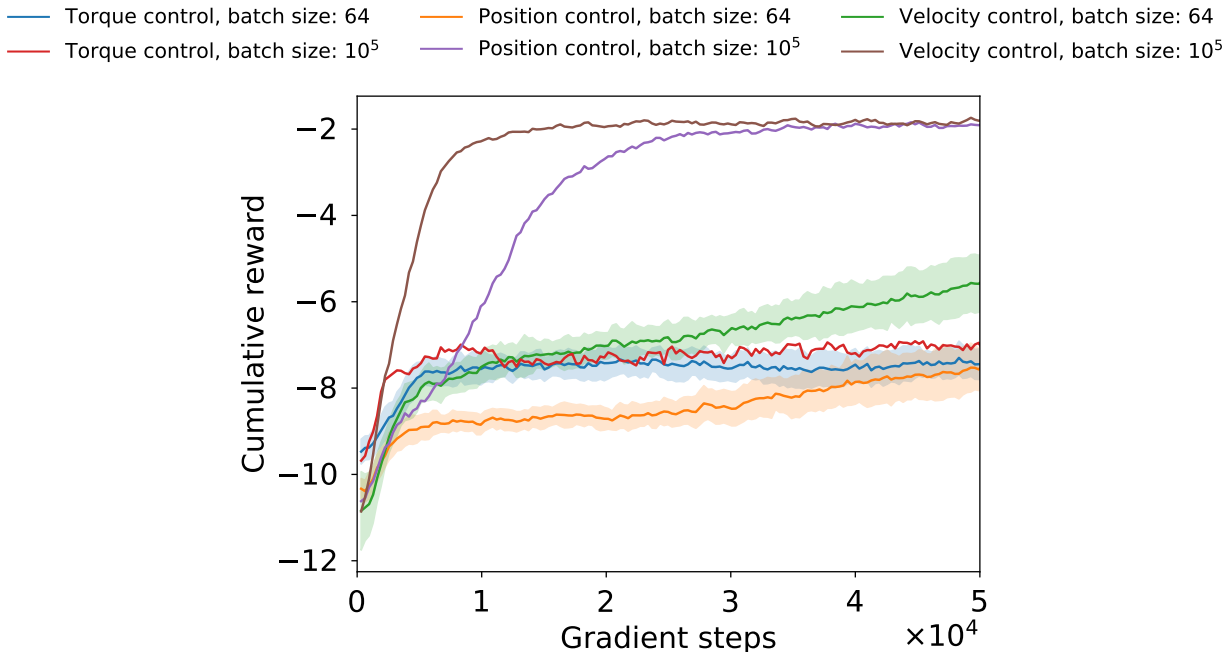

Fig. 10: Learning curves for the Reacher task with gradient estimates of different accuracies. For each action space configuration, we compare the learning performance of an agent that uses accurate estimates of the policy gradient (estimated on batches of 10^5 samples) to the learning performance of the default agent that uses rough estimates of the policy gradient (estimated on batches of 64 samples). Note that the x-axis uses gradient steps instead of environment steps to account for the fact that more data is collected for the accurate gradient estimates. More accurate gradients improve the convergence rate and final performance significantly for the position and velocity control configurations but not for the torque control configuration.

D. Defining states and actions in the same space

In this section, we explore the interplay between state and action representations. Particularly, we investigate whether a task becomes easier to learn if both the states and actions are defined in the same space. Intuitively, such a setting should make it easier for the agent to understand the relationship between actions and resulting states.

We test this hypothesis on the Reacher task. To that end, we define a modified version of the task that defines the target position in joint space instead of end-effector space. Modifying the state space of the original Reacher task, we arrive at the following definition of the state $\mathbf{s}$

$$\mathbf{s} = (\mathbf{q}, \mathbf{q}_{\text{target}}, \dot{\mathbf{q}}, \Delta\mathbf{q}_{\text{target}}), \tag{3}$$

where $\mathbf{q}$ denotes the current joint angles of the manipulator, $\mathbf{q}_{\text{target}}$ the target position expressed in joint angles, $\dot{\mathbf{q}}$ the current joint velocities, and $\Delta\mathbf{q}_{\text{target}}$ the joint angle distance to the target.

To ensure consistency, we also define the reward in joint space[1]. Specifically, we use the norm of the joint differences as the reward function.

$$r(\mathbf{s}, \mathbf{a}) = \left\| \frac{2}{\pi} \Delta\mathbf{q}_{\text{target}} \right\| \qquad \text{with } \mathbf{s} = (\mathbf{q}, \mathbf{q}_{\text{target}}, \dot{\mathbf{q}}, \Delta\mathbf{q}_{\text{target}}) \tag{4}$$

The factor $\frac{2}{\pi}$ ensures that the reward is in the same range as the end-effector-space reward of the original task.

On this task, we execute PPO with the three action spaces defined in Section II-A. To assess the effects of the simple linear controller not always reaching the target position perfectly, we introduce the *ideal position control* configuration in addition. In this configuration, we remove the linear controller and simply set the simulation state to the target joint angles at each time step.

---

[1]With the reward from the original task, which is defined in end-effector space, two configurations solve the task. For clarity, we would need to include both configurations in the observations.

This scheme emulates a position controller that always reaches the given target positions perfectly in a single step. Note that the task is quite simple in this configuration since the optimal policy is to simply take the target position from the state and apply it as action at every time step. The learning curves of PPO with the different action spaces on the joint space Reacher task are given in Figure 11.
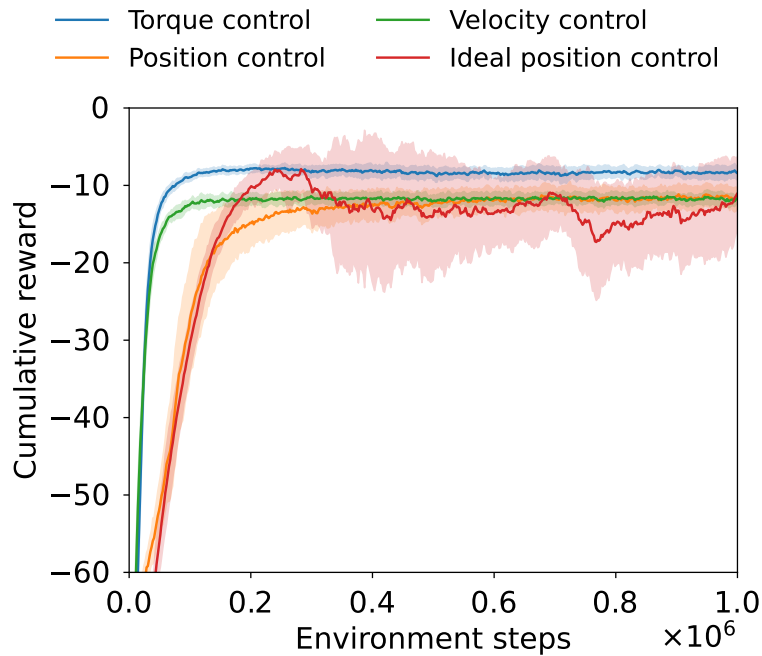

Fig. 11: Learning curves for the modified Reacher task, where the target states are expressed in joint space and the reward is defined as the difference to the target joint angles. Although the task is intuitively very simple in the position control configurations, the torque control configuration achieves superior performance and has lower variance across random seeds.

Surprisingly, torque control outperforms both position control configurations, even though the task is intuitively quite easy to solve in these configurations. This result invalidates the hypothesis that defining the state and action in the same space is always superior to using different representations. Moreover, this finding further illustrates that it is often challenging to get an intuitive understanding of the reasons for task difficulty in reinforcement learning.