Smooth Exploration for Robotic Reinforcement Learning

A Supplementary Material

A.1 State Dependent Exploration

In the linear case, i. e. with a linear policy and a noise matrix, parameter space exploration and SDE are equivalent:

$$\begin{aligned} \mathbf{a}_t &= \mu(\mathbf{s}_t; \theta_\mu) + \epsilon(\mathbf{s}_t; \theta_\epsilon), && \theta_\epsilon \sim \mathcal{N}(0, \sigma^2) \\ &= \theta_\mu \mathbf{s}_t + \theta_\epsilon \mathbf{s}_t \\ &= (\theta_\mu + \theta_\epsilon)\mathbf{s}_t \end{aligned}$$

Because we know the policy distribution, we can obtain the derivative of the log-likelihood $\log \pi(\mathbf{a}|\mathbf{s})$ with respect to the variance σ:

$$\frac{\partial \log \pi(\mathbf{a}|\mathbf{s})}{\partial \sigma_{ij}} = \sum_k \frac{\partial \log \pi_k(\mathbf{a}_k|\mathbf{s})}{\partial \hat{\sigma}_j} \frac{\partial \hat{\sigma}_j}{\partial \sigma_{ij}} \tag{1}$$

$$= \frac{\partial \log \pi_j(\mathbf{a}_j|\mathbf{s})}{\partial \hat{\sigma}_j} \frac{\partial \hat{\sigma}_j}{\partial \sigma_{ij}} \tag{2}$$

$$= \frac{(\mathbf{a}_j - \mu_j)^2 - \hat{\sigma_j}^2}{\hat{\sigma_j}^3} \frac{\mathbf{s}_i^2 \sigma_{ij}}{\hat{\sigma}_j} \tag{3}$$

This can be easily plugged into the likelihood ratio gradient estimator [1], which allows adapting σ during training. SDE is therefore compatible with standard policy gradient methods, while addressing most shortcomings of the unstructured exploration.

A.2 Algorithms

In this section, we shortly present the algorithms used in this paper. They correspond to state-of-the-art methods in model-free RL for continuous control, either in terms of sample efficiency or wall-clock time.

A2C A2C is the synchronous version of Asynchronous Advantage Actor-Critic (A3C) [2]. It is an actor-critic method that uses parallel rollouts of n-steps to update the policy. It relies on the REINFORCE [1] estimator to compute the gradient. A2C is fast but not sample efficient.

PPO A2C gradient update does not prevent large changes that lead to huge drop in performance. To tackle this issue, Trust Region Policy Optimization (TRPO) [3] introduces a trust-region in the policy parameter space, formulated as a constrained optimization problem: it updates the policy while being close in terms of KL divergence to the old policy. Its successor, Proximal Policy Optimization (PPO) [4] relaxes the constraints (which requires costly conjugate gradient step) by clipping the objective using importance ratio. PPO makes also use of workers (as in A2C) and Generalized Advantage Estimation (GAE) [5] for computing the advantage.

5th Conference on Robot Learning (CoRL 2021), London, UK.

TD3 Deep Deterministic Policy Gradient (DDPG) [6] combines the deterministic policy gradient algorithm [7] with the improvements from Deep Q-Network (DQN) [8]: using a replay buffer and target networks to stabilize training. Its direct successor, Twin Delayed DDPG (TD3) [9] brings three major tricks to tackle issues coming from function approximation: clipped double Q-Learning (to reduce overestimation of the Q-value function), delayed policy update (so the value function converges first) and target policy smoothing (to prevent overfitting). Because the policy is deterministic, DDPG and TD3 rely on external noise for exploration.

SAC Soft Actor-Critic [10], successor of Soft Q-Learning (SQL) [11] optimizes the maximum-entropy objective, that is slightly different compared to the classic RL objective:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} \left[r(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathcal{H}(\pi(\cdot|\mathbf{s}_t)) \right]. \tag{4}$$

where $\mathcal{H}$ is the policy entropy and α is the entropy temperature and allows to have a trade-off between the two objectives.

SAC learns a stochastic policy, using a squashed Gaussian distribution, and incorporates the clipped double Q-learning trick from TD3. In its latest iteration [12], SAC automatically adjusts the entropy coefficient α, removing the need to tune this crucial hyperparameter.

Which algorithm for robotics? A2C and PPO are both on-policy algorithms and can be easily parallelized, resulting in relatively small training time. On the other hand, SAC and TD3 are off-policy and run on a single worker, but are much more sample efficient than the two previous methods, achieving equivalent performances with a fraction of the samples.

Because we are focusing on robotics applications, having multiple robots is usually not possible, which makes TD3 and SAC the methods of choice. Although TD3 and SAC are very similar, SAC embeds the exploration directly in its objective function, making it easier to tune. We also found, during our experiments in simulation, that SAC works for a wide range of hyperparameters. As a result, we adopt that algorithm for the experiment on a real robot and for the ablation study.

A.3 Real Robot Experiments

Common Setup For each real robot experiment, to improve smoothness of the final controller and tackle communication delays (which would break Markov assumption), we augment the input with the previous observation and the last action taken and add a small continuity cost to the reward. For each task, we decompose the reward function into a primary (what we want to achieve) and secondary component (soft constraints such as continuity cost). Each reward term is normalized, which allows to easily weight each component depending on their importance. Compared to previous work, we use the exact same algorithm as the one used for simulated tasks and therefore avoid the use of filter.

Learning to control an elastic neck. An episode terminates either when the agent reaches the desired pose or after a timeout of 5s, i. e. each episode has a maximum length of 200 steps. The episode is considered successful if the desired pose is reached within a threshold of $10mm$ for the position and $5deg$ for the orientation.

Learning to walk with the elastic quadruped robot bert. The agent receives joint angles, velocities, torques and IMU data as input (over Wi-Fi) and commands the desired absolute motor angles. The primary reward is the distance traveled and the secondary reward is a weighted sum of different costs: heading cost, distance to the center line and continuity cost. Thanks to a treadmill, the reset of the robot was semi-automated. Early stopping and monitoring of the robot was done using external tracking, but the observation is computed from on-board sensors only. An episode terminates if the robot falls, goes out of bounds or after a timeout of 5s. Training is done directly with the real robot over several days, totalizing around 8 hours of interaction.

Learning to drive with a RC car. The agent receives an image from the on-board camera as input and commands desired throttle and steering angle. Features are computed using a pre-trained auto-encoder as in Raffin and Sokolkov [13]. The primary reward is a weighted sum between a survival

bonus (no intervention by the safety driver) and the commanded throttle. There is only the continuity cost as secondary reward. One episode terminates when the safety driver intervenes (crash) or after a timeout of 1 minute. Training is done directly on the robot and requires less than 30 minutes of interaction.

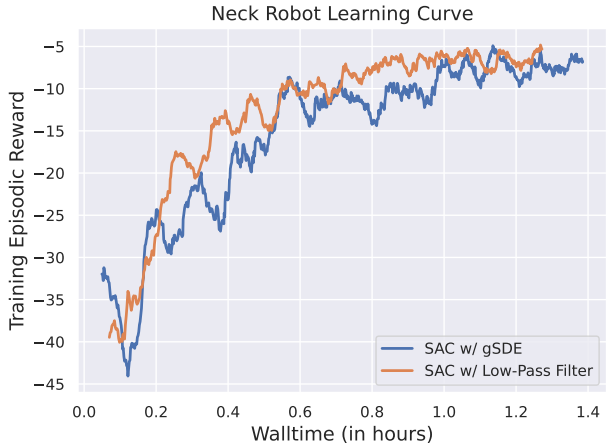


Figure 1: Learning curve for the tendon-driven robot. SAC with *g*SDE performs similarly to SAC with a low-pass filter for the primary reward (reaching target).

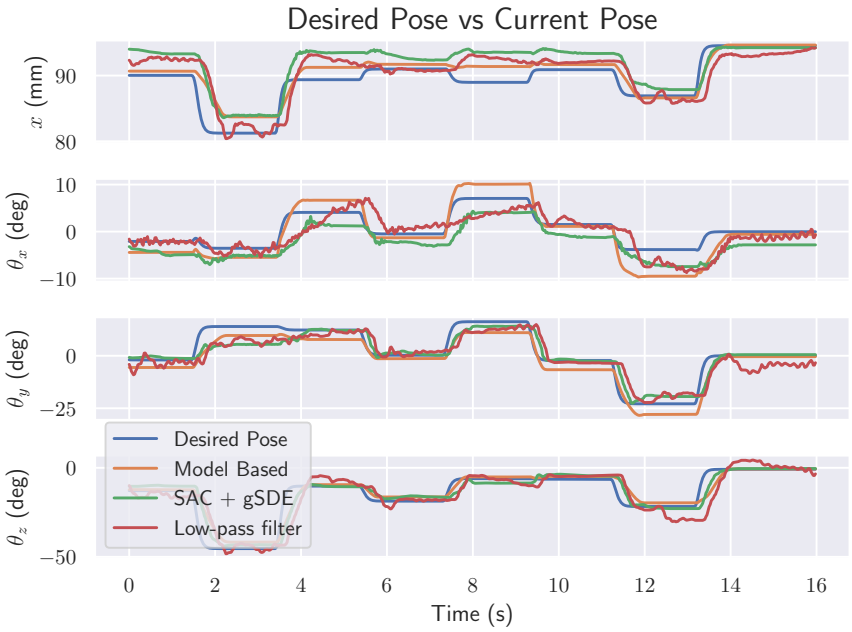


Figure 2: Comparison of the model-based controller with the learned RL agent on an evaluation trajectory: the two performs similarly.

	SAC + *g*SDE	Model-Based [14]
Error in position (mm)	2.65 +/- 1.6	1.32 +/- 1.2
Error in orientation (deg)	2.85 +/- 2.9	2.90 +/- 2.8

Table 1: Comparison of the mean error in position and orientation on the evaluation trajectory. The model-based and learned controller yield comparable results.

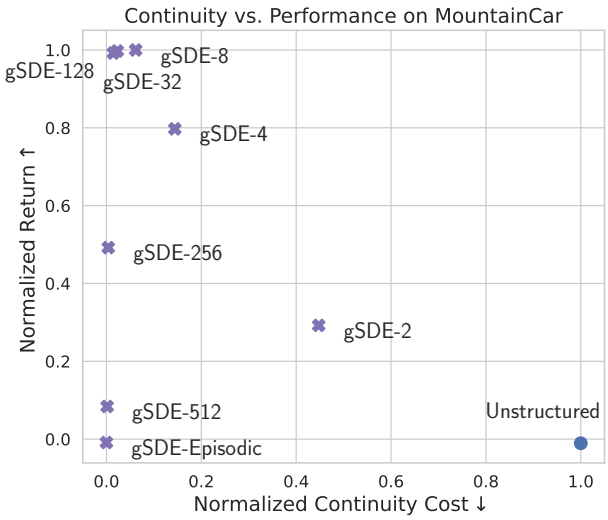


Figure 3: Normalized return and continuity cost of SAC on MountainCar task with different type of exploration. gSDE provides a compromise between performance and smoothness.

A.4 Addtional Experiments: MountainCar

To assess the applicability of gSDE on a non-periodic task, we train SAC with on the continuous version of the mountain car problem [15, 16]. In this environment, the agent must drive an underpowered car up a steep mountain road. The reward is deceptive: the agent must reach the goal using as few energy as possible and gets rewarded only when up the hill. The agent commands the power which indirectly influences its velocity.

Although low-dimensional (2-dimensional state and 1-dimensional action), this environment was shown to be challenging for DDPG [17].

Algorithm	MountainCar	
SAC	Return ↑	$\mathcal{C}_{\text{train}}$ ↓
w/ unstructured	-1.0 +/- 0.4	12.3 +/- 0.0
w/ gSDE-2	27.7 +/- 14	5.5 +/- 0.4
w/ gSDE-4	75.6 +/- 12.8	1.8 +/- 0.2
w/ gSDE-8	**94.8** +/- 0.1	0.8 +/- 0.0
w/ gSDE-16	**94.8** +/- 0.1	0.4 +/- 0.0
w/ gSDE-32	94.5 +/- 0.1	0.3 +/- 0.0
w/ gSDE-64	93.9 +/- 0.3	0.2 +/- 0.0
w/ gSDE-128	94.0 +/- 0.3	0.2 +/- 0.0
w/ gSDE-256	46.6 +/- 15.8	0.2 +/- 0.0
w/ gSDE-512	7.9 +/- 9.2	**0.0** +/- 0.0
w/ gSDE-Episodic	-0.8 +/- 0.3	**0.0** +/- 0.0

Table 2: Detailed return and continuity cost results for SAC with different type of exploration on the MountainCar environment. We report the mean and standard error over 10 runs of 60000 steps. For each benchmark, we highlight the results of the method(s) with the best mean when the difference is statistically significant.

Table 2 and Fig. 3 shows the results on MountainCar task and the compromise between continuity and performance.

Despite hyperparameter optimization, the problem cannot be solved by the original SAC (with unstructured noise) without additional external noise[1].

Because of the unstructured exploration, the commanded power oscillates at high frequency, making the velocity stay around the initial value of zero. The policy thus converges to a local minimum of doing nothing, which minimizes the consumed energy.

On the other hand, SAC with gSDE works for a wide range of the noise sampling interval n (gSDE-4 to gSDE-128), while also improving a lot on the continuity cost at train time. If the sampling interval is too large (for instance with gSDE-Episodic), the agent would not explore enough during long episodes and then converge to the local minimum.

A.5 Implementation Details

We used a PyTorch [18] version of Stable-Baselines [19] library, together with the RL Zoo training framework [20]. It uses the common implementations tricks for PPO [21] for the version using independent Gaussian noise.

For SAC, to ensure numerical stability, we clip the mean to be in range $[-2, 2]$, as it was causing infinite values. In the original implementation, a regularization $\mathcal{L}_2$ loss on the mean and standard deviation was used instead. The algorithm for SAC with gSDE is described in Algorithm 1.

Compared to the original SDE paper, we did not have to use the *expln* trick [22] to avoid exploding variance for PyBullet tasks. However, we found it useful on specific environment like *BipedalWalkerHardcore-v2*. The original SAC implementation clips this variance.

Algorithm 1 Soft Actor-Critic with gSDE

Initialize parameters θ_μ, θ_Q, σ, α
Initialize replay buffer $\mathcal{D}$
for each iteration **do**
  $\theta_\epsilon \sim \mathcal{N}(0, \sigma^2)$ ▷ Sample noise function parameters
  for each environment step **do**
    $\mathbf{a}_t = \pi(\mathbf{s}_t) = \mu(\mathbf{s}_t; \theta_\mu) + \epsilon(\mathbf{s}_t; \theta_\epsilon)$ ▷ Get the noisy action
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ ▷ Step in the environment
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$ ▷ Update the replay buffer
  end for
  for each gradient step **do**
    $\theta_\epsilon \sim \mathcal{N}(0, \sigma^2)$ ▷ Sample noise function parameters
    Sample a minibatch from the replay buffer $\mathcal{D}$
    Update the entropy temperature α
    Update parameters using ∇J_Q and ∇J_π ▷ Update actor μ, critic Q and noise variance σ
    Update target networks
  end for
end for

A.6 Learning Curves and Additional Results

Figure 5 shows the learning curves for SAC with different types of exploration noise.

Figure 6 and Figure 7 show the learning curves for off-policy and on-policy algorithms on the four PyBullet tasks, using gSDE or unstructured Gaussian exploration.

A.7 Ablation Study: Additional Plots

Figure 8 displays the ablation study on remaining PyBullet tasks. It shows that SAC is robust against initial exploration variance, and PPO results highly depend on the noise sampling interval.

Parallel Sampling The effect of sampling a set of noise parameters per worker is shown for PPO in Figure 9a. This modification improves the performance for each task, as it allows a more diverse

[1] See issue on the original SAC repository `https://frama.link/original-sac-mountaincar`

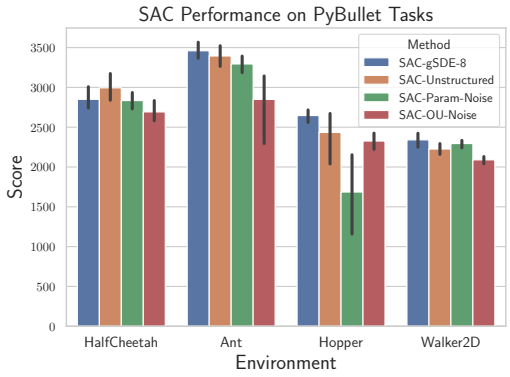


Figure 4: Performance results for SAC with different type of exploration on PyBullet tasks

	A2C		**PPO**	
Environments	*g*SDE	Gaussian	*g*SDE	Gaussian
HALFCHEETAH	**2028** +/- **107**	1652 +/- 94	**2760** +/- **52**	2254 +/- 66
ANT	**2560** +/- **45**	1967 +/- 104	**2587** +/- **133**	2160 +/- 63
HOPPER	1448 +/- 163	1559 +/- 129	**2508** +/- **16**	1622 +/- 220
WALKER2D	**694** +/- **73**	443 +/- 59	**1776** +/- **53**	1238 +/- 75

Table 3: Final performance (higher is better) of A2C and PPO on 4 environments with *g*SDE and unstructured Gaussian exploration. We report the mean over 10 runs of 2 million steps. For each benchmark, we highlight the results of the method with the best mean, when the difference is statistically significant.

	SAC		**TD3**	
Environments	*g*SDE	Gaussian	*g*SDE	Gaussian
HALFCHEETAH	2850 +/- 73	2994 +/- 89	2578 +/- 44	2687 +/- 67
ANT	3459 +/- 52	3394 +/- 64	**3267** +/- **34**	2865 +/- 278
HOPPER	2646 +/- 45	2434 +/- 190	2353 +/- 78	2470 +/- 111
WALKER2D	**2341** +/- **45**	2225 +/- 35	1989 +/- 153	2106 +/- 67

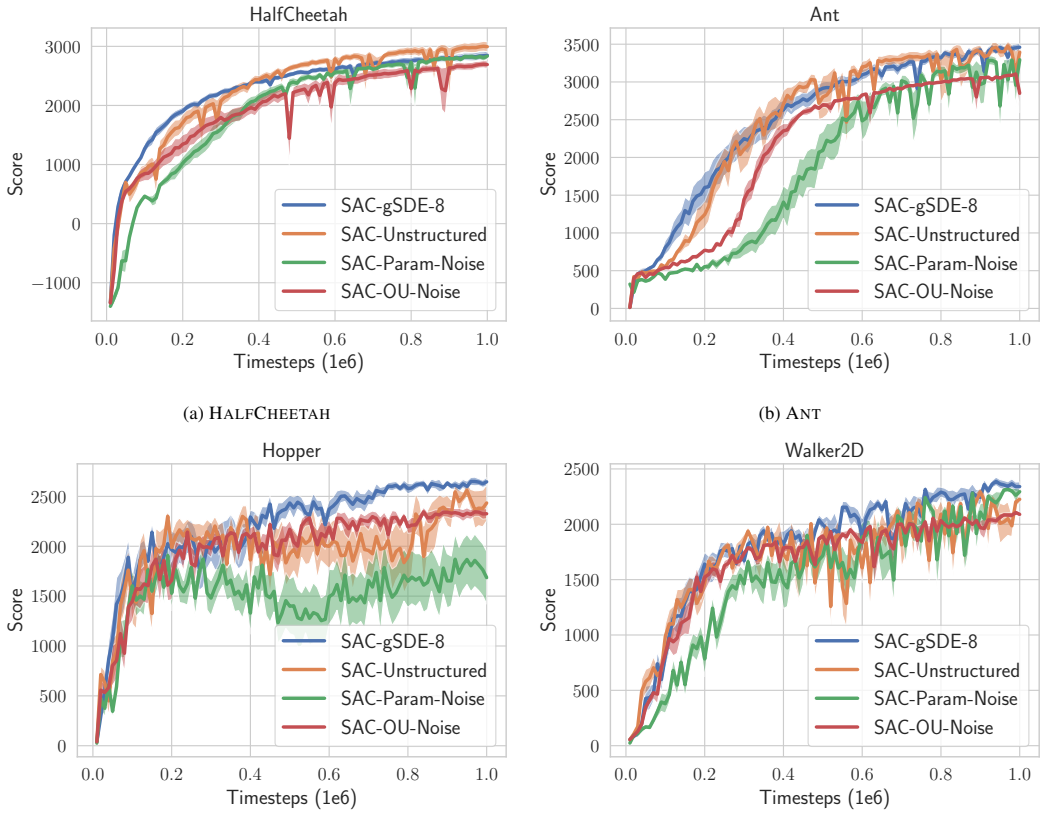
Table 4: Final performance (higher is better) of SAC and TD3 on 4 environments with *g*SDE and unstructured Gaussian exploration. We report the mean over 10 runs of 1 million steps. For each benchmark, we highlight the results of the method with the best mean, when the difference is statistically significant.

exploration. Although less significant, we observe the same outcome for A2C on PyBullet environments (cf. Figure 9b). Thus, making use of parallel workers improves both exploration and the final performance.

A.8 Hyperparameter Optimization

PPO and TD3 hyperparameters for unstructured exploration are reused from the original papers [4, 9]. For SAC, the optimized hyperparameters for *g*SDE are performing better than the ones from Haarnoja et al. [11], so we keep them for the other types of exploration to have a fair comparison. No hyperparameters are available for A2C in Mnih et al. [2] so we use the tuned one from Raffin [23].

To tune the hyperparameters, we use a TPE sampler and a median pruner from Optuna [24] library. We give a budget of 500 candidates with a maximum of $3 \cdot 10^5$ time-steps on the HALFCHEETAH environment. Some hyperparameters are then manually adjusted (e. g. increasing the replay buffer size) to improve the stability of the algorithms.



(a) HALFCHEETAH (b) ANT

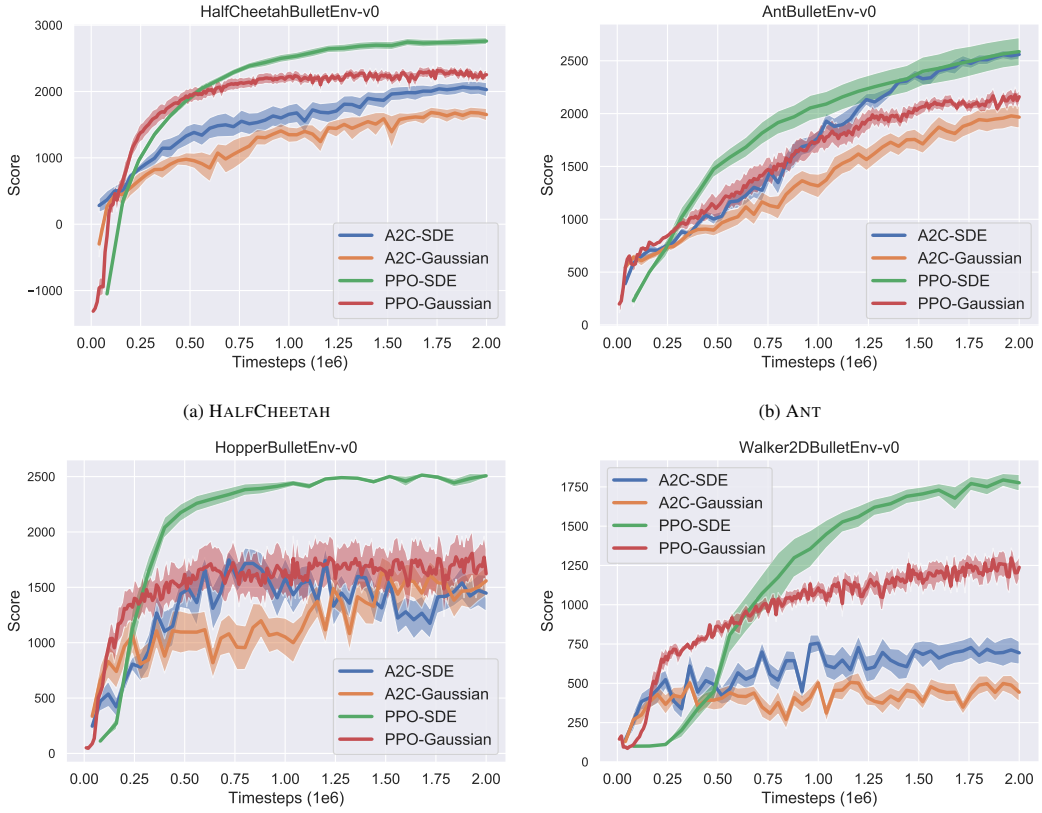
(c) HOPPER (d) WALKER2D

Figure 5: Learning curves for SAC with different type of exploration on PyBullet tasks. The line denotes the mean over 10 runs of 1 million steps.

A.9 Hyperparameters

For all experiments with a time limit, as done in [25, 26, 27, 19], we augment the observation with a time feature (remaining time before the end of an episode) to avoid breaking Markov assumption. This feature has a great impact on performance, as shown in Figure 10b.

Figure 10a displays the influence of the network architecture for SAC on PyBullet tasks. A bigger network usually yields better results but the gain is minimal passed a certain complexity (here, a two layers neural network with 256 unit per layer).



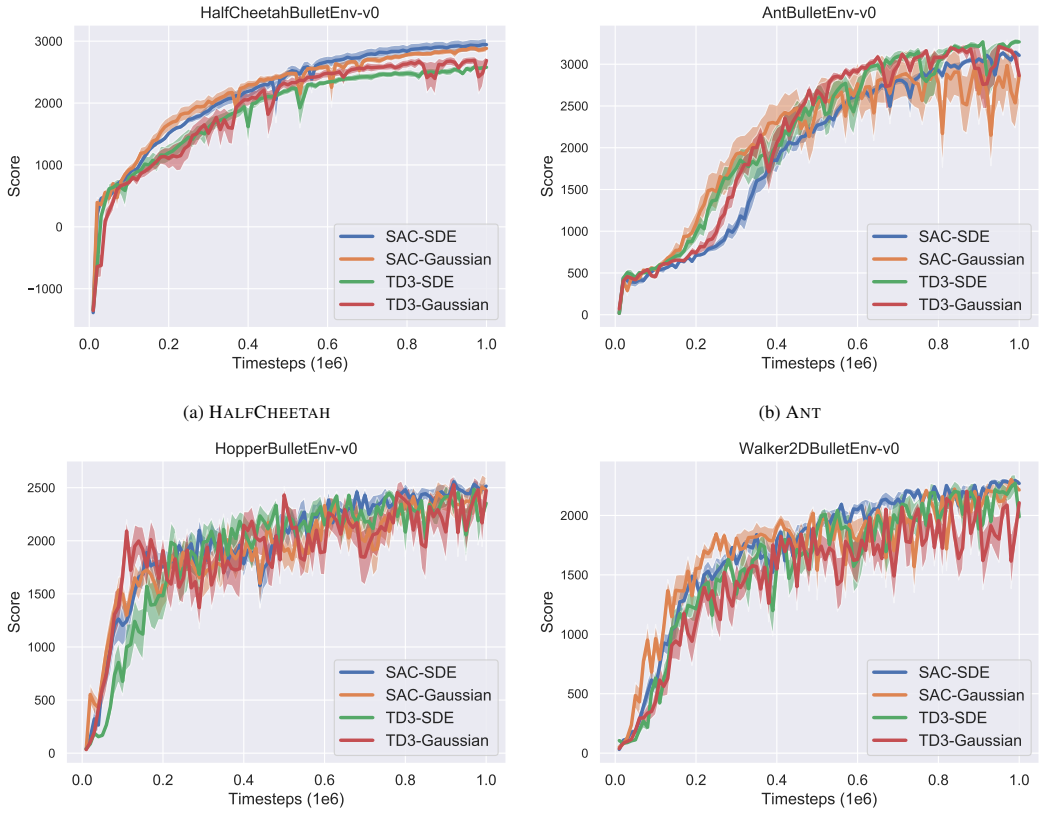
(a) HALFCHEETAH (b) ANT

(c) HOPPER (d) WALKER2D

Figure 6: Learning curves for on-policy algorithms on PyBullet tasks. The line denotes the mean over 10 runs of 2 million steps.

Table 5: SAC Hyperparameters

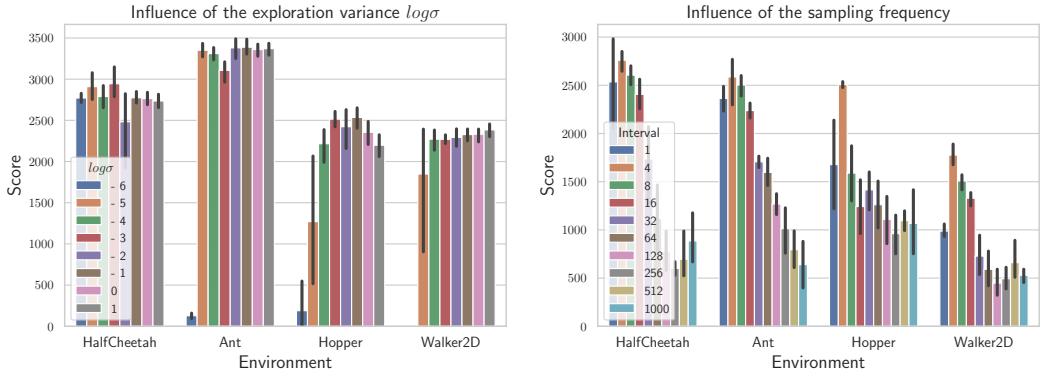
Parameter	Value
Shared	
optimizer	Adam [28]
learning rate	$7.3 \cdot 10^{-4}$
learning rate schedule	constant
discount (γ)	0.98
replay buffer size	$3 \cdot 10^5$
number of hidden layers (all networks)	2
number of hidden units per layer	[400, 300]
number of samples per minibatch	256
non-linearity	$ReLU$
entropy coefficient (α)	auto
target entropy	$-dim(\mathcal{A})$
target smoothing coefficient (τ)	0.02
train frequency	episodic
warm-up steps	10 000
normalization	None
gSDE	
initial $\log \sigma$	-3
gSDE sample frequency	8



(a) HALFCHEETAH (b) ANT

(c) HOPPER (d) WALKER2D

Figure 7: Learning curves for off-policy algorithms on PyBullet tasks. The line denotes the mean over 10 runs of 1 million steps.

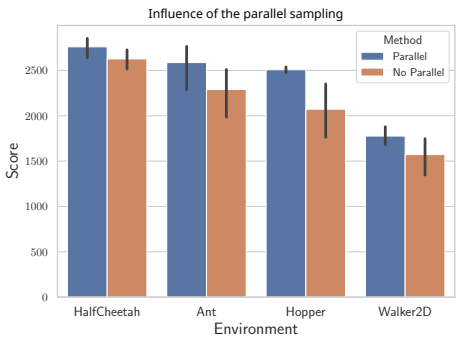


(a) Initial exploration variance $\log \sigma$ (SAC) (b) Noise sampling interval (PPO)

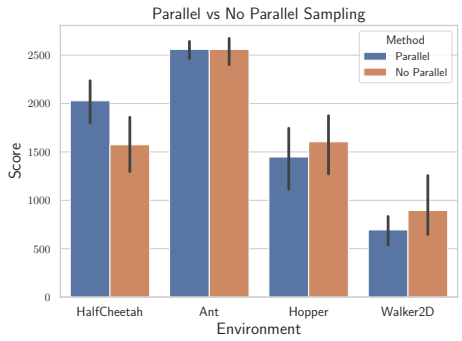
Figure 8: Sensitivity of SAC and PPO to selected hyperparameters on PyBullet tasks

Table 6: SAC Environment Specific Parameters

Environment	Learning rate schedule
HopperBulletEnv-v0	linear
Walker2dBulletEnv-v0	linear

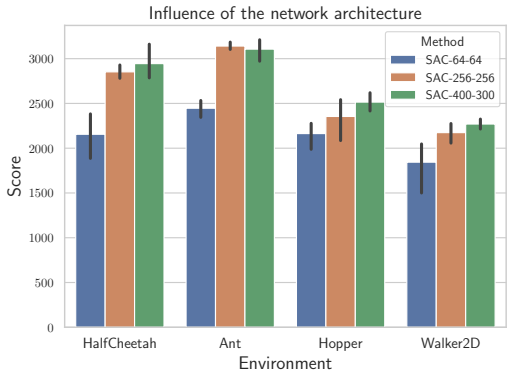


(a) Effect of parallel sampling for PPO

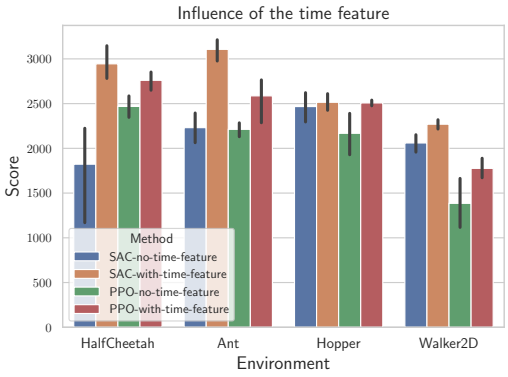


(b) Effect of parallel sampling for A2C

Figure 9: Parallel sampling of the noise matrix has a positive impact for PPO (a) and A2C (b) on PyBullet tasks.



(a) Influence of the network architecture



(b) Influence of the time feature

Figure 10: (a) Influence of the network architecture (same for actor and critic) for SAC on PyBullet environments. The labels displays the number of units per layer. (b) Influence of including the time or not in the observation for PPO and SAC.

Table 7: TD3 Hyperparameters

Parameter	Value
Shared	
optimizer	Adam [28]
discount (γ)	0.98
replay buffer size	$2 \cdot 10^5$
number of hidden layers (all networks)	2
number of hidden units per layer	[400, 300]
number of samples per minibatch	100
non-linearity	$ReLU$
target smoothing coefficient (τ)	0.005
target policy noise	0.2
target noise clip	0.5
policy delay	2
warm-up steps	10 000
normalization	None
*g*SDE	
initial $\log \sigma$	-3.62
learning rate for TD3	$6 \cdot 10^{-4}$
target update interval	64
train frequency	64
gradient steps	64
learning rate for *g*SDE	$1.5 \cdot 10^{-3}$
Unstructured Exploration	
learning rate	$1 \cdot 10^{-3}$
action noise type	Gaussian
action noise std	0.1
train frequency	every episode
gradient steps	every episode

Table 8: A2C Hyperparameters

Parameter	Value
Shared	
number of workers	4
optimizer	RMSprop with $\epsilon = 1 \cdot 10^{-5}$
discount (γ)	0.99
number of hidden layers (all networks)	2
number of hidden units per layer	[64, 64]
shared network between actor and critic	False
non-linearity	$Tanh$
value function coefficient	0.4
entropy coefficient	0.0
max gradient norm	0.5
learning rate schedule	linear
normalization	observation and reward [19]
gSDE	
number of steps per rollout	8
initial $\log \sigma$	-3.62
learning rate	$9 \cdot 10^{-4}$
GAE coefficient [5] (λ)	0.9
orthogonal initialization [21]	no
Unstructured Exploration	
number of steps per rollout	32
initial $\log \sigma$	0.0
learning rate	$2 \cdot 10^{-3}$
GAE coefficient [5] (λ)	1.0
orthogonal initialization [21]	yes

Table 9: PPO Hyperparameters

Parameter	Value
Shared	
optimizer	Adam [28]
discount (γ)	0.99
value function coefficient	0.5
entropy coefficient	0.0
number of hidden layers (all networks)	2
shared network between actor and critic	False
max gradient norm	0.5
learning rate schedule	constant
advantage normalization [19]	True
clip range value function [21]	no
normalization	observation and reward [19]
gSDE	
number of workers	16
number of steps per rollout	512
initial $\log \sigma$	-2
gSDE sample frequency	4
learning rate	$3 \cdot 10^{-5}$
number of epochs	20
number of samples per minibatch	128
number of hidden units per layer	[256, 256]
non-linearity	$ReLU$
GAE coefficient [5] (λ)	0.9
clip range	0.4
orthogonal initialization [21]	no
Unstructured Exploration	
number of workers	1
number of steps per rollout	2048
initial $\log \sigma$	0.0
learning rate	$2 \cdot 10^{-4}$
number of epochs	10
number of samples per minibatch	64
number of hidden units per layer	[64, 64]
non-linearity	$Tanh$
GAE coefficient [5] (λ)	0.95
clip range	0.2
orthogonal initialization [21]	yes

Table 10: PPO Environment Specific Parameters

Environment	Learning rate schedule	Clip range schedule	initial $\log \sigma$
gSDE			
AntBulletEnv-v0	default	default	-1
HopperBulletEnv-v0	default	linear	-1
Walker2dBulletEnv-v0	default	linear	default
Unstructured Exploration			
Walker2dBulletEnv-v0	linear	default	default

References

[1] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[2] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[3] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

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

[5] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[6] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[7] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, page I–387–I–395. JMLR.org, 2014.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[9] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

[10] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine. Reinforcement learning with deep energy-based policies. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1352–1361. JMLR. org, 2017.

[11] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[12] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[13] A. Raffin and R. Sokolkov. Learning to drive smoothly in minutes. https://github.com/araffin/learning-to-drive-in-5-minutes/, 2019.

[14] B. Deutschmann, A. Dietrich, and C. Ott. Position control of an underactuated continuum mechanism using a reduced nonlinear model. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 5223–5230. IEEE, 2017.

[15] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[16] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[17] C. Colas, O. Sigaud, and P.-Y. Oudeyer. Gep-pg: Decoupling exploration and exploitation in deep reinforcement learning algorithms. *arXiv preprint arXiv:1802.05054*, 2018.

[18] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann. Stable baselines3. https://github.com/DLR-RM/stable-baselines3, 2019.

[19] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

[20] A. Raffin. Rl baselines3 zoo. https://github.com/DLR-RM/rl-baselines3-zoo, 2020.

[21] 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. URL https://openreview.net/forum?id=r1etN1rtPB.

[22] T. Rückstieß, M. Felder, and J. Schmidhuber. State-dependent exploration for policy gradient methods. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 234–249. Springer, 2008.

[23] A. Raffin. Rl baselines zoo. https://github.com/araffin/rl-baselines-zoo, 2018.

[24] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[25] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[26] F. Pardo, A. Tavakoli, V. Levdik, and P. Kormushev. Time limits in reinforcement learning. *arXiv preprint arXiv:1712.00378*, 2017.

[27] A. Rajeswaran, K. Lowrey, E. V. Todorov, and S. M. Kakade. Towards generalization and simplicity in continuous control. In *Advances in Neural Information Processing Systems*, pages 6550–6561, 2017.

[28] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.