Smooth Exploration for Robotic Reinforcement Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract:
Reinforcement learning (RL) enables robots to learn skills from interactions with the real world. In practice, the unstructured step-based exploration used in Deep RL – often very successful in simulation – leads to jerky motion patterns on real robots. Consequences of the resulting shaky behavior are poor exploration, or even damage to the robot. We address these issues by adapting state-dependent exploration (SDE) [1] to current Deep RL algorithms. To enable this adaptation, we propose two extensions to the original SDE, using more general features and resampling the noise periodically, which leads to a new exploration method *generalized state-dependent exploration* (gSDE). We evaluate gSDE both in simulation, on PyBullet continuous control tasks, and directly on three different real robots: a tendon-driven elastic robot, a quadruped and an RC car. The noise sampling interval of gSDE enables a compromise between performance and smoothness, which allows training directly on the real robots without loss of performance.

1 Introduction

One of the first robots that used artificial intelligence methods was called "Shakey", because it would shake a lot during operation [2]. Shaking has now again become quite prevalent in robotics, but for a different reason. When learning robotic skills with deep reinforcement learning (DeepRL), the de facto standard for exploration is to sample a noise vector ϵ_t from a Gaussian distribution independently at each time step t, and then adding it to the policy output. This approach leads to the type of noise illustrated to the left in Fig. 1, and it can be very effective in simulation [3, 4, 5, 6, 7].

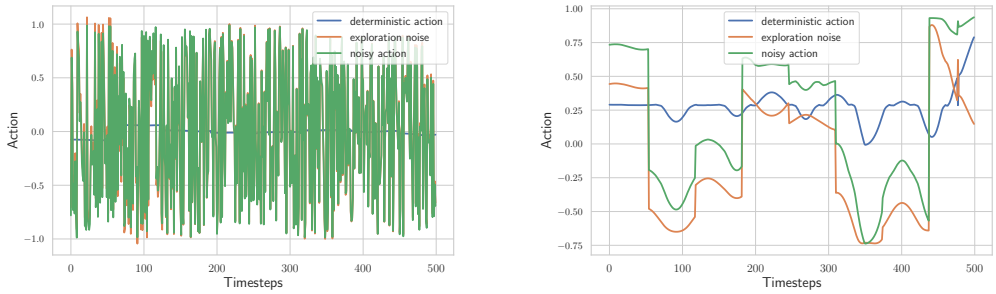


Figure 1: Left: unstructured exploration, as typically used in simulated RL. Right: gSDE provides smooth and consistent exploration.

Unstructured exploration has also been applied to robotics [8, 9]. But for experiments on real robots, it has many drawbacks, which have been repeatedly pointed out [1, 10, 11, 12, 13]: 1) Sampling independently at each step leads to shaky behavior [14], and noisy, jittery trajectories. 2) The jerky motion patterns can damage the motors on a real robot, and lead to increased wear-and-tear. 3) In the real world, the system acts as a low pass filter. Thus, consecutive perturbations may cancel each

Submitted to the 5th Conference on Robot Learning (CoRL 2021). Do not distribute.

other, leading to poor exploration. This is particularly true for high control frequency [15]. 4) It causes a large variance which grows with the number of time-steps [10, 11, 12]

In practice, we have observed all of these drawbacks on three real robots, including the tendon-driven robot David, depicted in Fig. 4a, which is the main experimental platform used in this work. For all practical purposes, Deep RL with unstructured noise cannot be applied to David.

In robotics, multiple solutions have been proposed to counteract the inefficiency of unstructured noise. These include correlated noise [8, 15], low-pass filters [16, 17], action repeat [18] or lower level controllers [16, 9]. A more principled solution is to perform exploration in parameter space, rather than in action space [19, 20]. This approach usually requires fundamental changes in the algorithm, and is harder to tune when the number of parameters is high.

State-Dependent Exploration (SDE) [1, 11] was proposed as a compromise between exploring in parameter and action space. SDE replaces the sampled noise with a state-dependent exploration function, which during an episode returns the same action for a given state. This results in smoother exploration and less variance per episode.

To the best of our knowledge, no Deep RL algorithm has yet been successfully combined with SDE. We surmise that this is because the problem that it solves – shaky, jerky movement – is not as noticeable in simulation, which is the current focus of the community.

In this paper, we aim at reviving interest in SDE as an effective method for addressing exploration issues that arise from using independently sampled Gaussian noise on real robots. Our concrete contributions, which also determine the structure of the paper, are:

1. Highlighting the issues with unstructured Gaussian exploration (Sect. 1).
2. Adapting SDE to recent Deep RL algorithms, and addressing some issues of the original formulation (Sects. 2.2 and 3).
3. Evaluate the different approaches with respect to the compromise between smoothness and performance, and show the impact of the noise sampling interval (Sects. 4.1 and 4.2).
4. Successfully applying RL directly on three real robots: a tendon-driven robot, a quadruped and an RC car, without need of a simulator or filters (Sect. 4.3).

2 Background

In reinforcement learning, an agent interacts with its environment, usually modeled as a Markov Decision Process (MDP) $(\mathcal{S}, \mathcal{A}, p, r)$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space and $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ the transition function. At every step t, the agent performs an action $\mathbf{a}$ in state $\mathbf{s}$ following its policy $\pi : \mathcal{S} \mapsto \mathcal{A}$. It then receives a feedback signal in the next state $\mathbf{s}'$: the reward $r(\mathbf{s}, \mathbf{a})$. The objective of the agent is to maximize the long-term reward. More formally, the goal is to maximize the expectation of the sum of discounted reward, over the trajectories ρ_π generated using its policy π:

$$\sum_t \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} \left[\gamma^t r(\mathbf{s}_t, \mathbf{a}_t) \right] \tag{1}$$

where $\gamma \in [0, 1)$ is the discount factor and represents a trade-off between maximizing short-term and long-term rewards. The agent-environment interactions are often broken down into sequences called *episodes*, that end when the agent reaches a terminal state.

2.1 Exploration in Action or Policy Parameter Space

In the case of continuous actions, the exploration is commonly done in the *action space* [21, 22, 23, 24, 25, 5]. At each time-step, a noise vector ϵ_t is independently sampled from a Gaussian distribution and then added to the controller output:

$$\mathbf{a}_t = \mu(\mathbf{s}_t; \theta_\mu) + \epsilon_t, \qquad \epsilon_t \sim \mathcal{N}(0, \sigma^2) \tag{2}$$

where $\mu(\mathbf{s}_t)$ is the deterministic policy and $\pi(\mathbf{a}_t|\mathbf{s}_t) \sim \mathcal{N}(\mu(\mathbf{s}_t), \sigma^2)$ is the resulting stochastic policy, used for exploration. θ_μ denotes the parameters of the deterministic policy.
For simplicity, throughout the paper, we will only consider Gaussian distributions with diagonal covariance matrices. Hence, here, σ is a vector with the same dimension as the action space $\mathcal{A}$.

Alternatively, the exploration can also be done in the *parameter space* [11, 19, 20]:

$$\mathbf{a}_t = \mu(\mathbf{s}_t; \theta_\mu + \epsilon), \qquad \epsilon \sim \mathcal{N}(0, \sigma^2) \tag{3}$$

at the beginning of an episode, the perturbation ϵ is sampled and added to the policy parameters θ_μ. This usually results in more consistent exploration but becomes challenging with an increasing number of parameters [19].

2.2 State-Dependent Exploration

*State-Dependent Exploration (*SDE*)* [1, 11] is an intermediate solution that consists in adding noise as a function of the state $\mathbf{s}_t$, to the deterministic action $\mu(\mathbf{s}_t)$. At the beginning of an episode, the parameters θ_ϵ of that exploration function are drawn from a Gaussian distribution. The resulting action $\mathbf{a}_t$ is as follows:

$$\mathbf{a}_t = \mu(\mathbf{s}_t; \theta_\mu) + \epsilon(\mathbf{s}_t; \theta_\epsilon), \qquad \theta_\epsilon \sim \mathcal{N}(0, \sigma^2) \tag{4}$$

This episode-based exploration is smoother and more consistent than the unstructured step-based exploration. Thus, during one episode, instead of oscillating around a mean value, the action $\mathbf{a}$ for a given state $\mathbf{s}$ will be the same.

SDE should not be confused with unstructured noise where the variance can be state-dependent but the noise is still sampled at every step, as it is the case for SAC.

In the remainder of this paper, to avoid overloading notation, we drop the time subscript t, i. e. we now write $\mathbf{s}$ instead of $\mathbf{s}_t$. $\mathbf{s}_j$ or $\mathbf{a}_j$ now refer to an element of the state or action vector.

In the case of a linear exploration function $\epsilon(\mathbf{s}; \theta_\epsilon) = \theta_\epsilon \mathbf{s}$, by operation on Gaussian distributions, Rückstieß et al. [1] show that the action element $\mathbf{a}_j$ is normally distributed:

$$\pi_j(\mathbf{a}_j|\mathbf{s}) \sim \mathcal{N}(\mu_j(\mathbf{s}), \hat{\sigma_j}^2) \tag{5}$$

where $\hat{\sigma}$ is a diagonal matrix with elements $\hat{\sigma_j} = \sqrt{\sum_i (\sigma_{ij}\mathbf{s}_i)^2}$.

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:

$$\frac{\partial \log \pi(\mathbf{a}|\mathbf{s})}{\partial \sigma_{ij}} = \frac{(\mathbf{a}_j - \mu_j)^2 - \hat{\sigma_j}^2}{\hat{\sigma_j}^3} \frac{\mathbf{s}_i^2 \sigma_{ij}}{\hat{\sigma_j}} \tag{6}$$

This can be easily plugged into the likelihood ratio gradient estimator [26], which enables adaptation of σ during training. SDE is therefore compatible with standard policy gradient methods, while addressing most shortcomings of the unstructured exploration.

For a non-linear exploration function, the resulting distribution $\pi(\mathbf{a}|\mathbf{s})$ is most of the time unknown. Thus, computing the exact derivative w.r.t. the variance is not trivial and may require approximate inference. As we focus on simplicity, we leave this extension for future work.

3 Generalized State-Dependent Exploration

Considering Eqs. (5) and (6), some limitations of the original formulation are apparent:

- i The noise does not change during one episode, which is problematic if the episode length is long, because the exploration will be limited [27].
- ii The variance of the policy $\hat{\sigma_j} = \sqrt{\sum_i (\sigma_{ij}\mathbf{s}_i)^2}$ depends on the state space dimension (it grows with it), which means that the initial σ must be tuned for each problem.
- iii There is only a linear dependency between the state and the exploration noise, which limits the possibilities.
- iv The state must be normalized, as the gradient and the noise magnitude depend on the state magnitude.

To mitigate the mentioned issues and adapt it to Deep RL algorithms, we propose two improvements:

1. We sample the parameters θ_ϵ of the exploration function every n steps instead of every episode.
2. Instead of the state $\mathbf{s}$, we can in fact use any features. We chose policy features $\mathbf{z}_\mu(\mathbf{s}; \theta_{\mathbf{z}_\mu})$ (last layer before the deterministic output $\mu(\mathbf{s}) = \theta_\mu \mathbf{z}_\mu(\mathbf{s}; \theta_{\mathbf{z}_\mu})$) as input to the noise function $\epsilon(\mathbf{s}; \theta_\epsilon) = \theta_\epsilon \mathbf{z}_\mu(\mathbf{s})$.

Sampling the parameters θ_ϵ every n steps tackles the issue i. and yields a unifying framework [27] which encompasses both unstructured exploration ($n = 1$) and original SDE ($n = \text{episode_length}$). Although this formulation follows the description of Deep RL algorithms that update their parameters every m steps, the influence of this crucial parameter on smoothness and performance was until now overlooked.

Using *policy features* allows mitigating issues ii, iii and iv: the relationship between the state $\mathbf{s}$ and the noise ϵ is non-linear and the variance of the policy only depends on the network architecture. This makes gSDE more task-independent (as the network architecture is usually kept constant) and saves a lot of parameters and computation when working with large state space such as images: the number of parameters and operations is only a function of the last layer size and action dimension and no more of the state space size. This formulation is therefore more general and includes the original SDE description, when using state as input to the noise function or when the policy is linear.

We call the resulting approach *generalized* State-Dependent Exploration (gSDE).

Deep RL algorithms Integrating this updated version of SDE into recent Deep RL algorithms, such as those listed in the appendix, is straightforward. For those that rely on a probability distribution, such as SAC or PPO, we can replace the original Gaussian distribution by the one from Eq. (5), where the analytical form of the log-likelihood is known (cf. Eq. (6)).

4 Experiments

In this section, we study gSDE to answer the following questions:

- How does gSDE compares to the original SDE? What is the impact of each proposed modification?
- How does gSDE compares to other type of exploration noise in terms of compromise between smoothness and performance?
- How does gSDE performs on a real system?

4.1 Compromise Between Smoothness and Performance

Experiment setup In order to compare gSDE to other type of exploration in terms of compromise between performance and smoothness, we chose 4 locomotion tasks from the PyBullet [28] environments: HALFCHEETAH, ANT, HOPPER and WALKER2D. They are similar to the one found in OpenAI Gym [29] but the simulator is open source and they are harder to solve[1]. In this section, we focus on the SAC algorithm as it will be the one used on the real robot, although we report results for additional algorithms such as PPO in the appendix.

To evaluate smoothness, we define a continuity cost $\mathcal{C} = 100 \times \mathbb{E}_t \left[(\frac{\mathbf{a}_{t+1} - \mathbf{a}_t}{\Delta^{\mathbf{a}}_{\max}})^2 \right]$ which yields values between 0 (constant output) and 100 (action jumping from one limit to another at every step). The continuity cost of the training $\mathcal{C}_{\text{train}}$ is a proxy for the wear-and-tear of the robot.

We compare the performance of the following configurations: (a) no exploration noise, (b) unstructured Gaussian noise (original SAC implementation), (c) correlated noise (Ornstein–Uhlenbeck process [30] with σ=0.2, OU noise in the figure), (d) adaptive parameter noise [19] (σ=0.2), (e) gSDE. To decorrelate the exploration noise from the one due to parameter update, and to be closer to a real robot setting, we apply the gradient updates only at the end of each episode.

[1] https://frama.link/PyBullet-harder-than-MuJoCo

We fix the budget to 1 million steps and report the average score over 10 runs together with the average continuity cost during training and their standard error. For each run, we test the learned policy on 20 evaluation episodes every 10000 steps, using the deterministic controller $\mu(\mathbf{s}_t)$. Regarding the implementation, we use a modified version of Stable-Baselines3 [31] together with the RL Zoo training framework [32]. The methodology we follow to tune the hyperparameters and their details can be found in the appendix. The code we used to run the experiments and tune the hyperparameters can be found in the supplementary material.

Algorithm	HALFCHEETAH		ANT		HOPPER		WALKER2D	
SAC	Return ↑	$\mathcal{C}_{\text{train}}$ ↓	Return ↑	$\mathcal{C}_{\text{train}}$ ↓	Return ↑	$\mathcal{C}_{\text{train}}$ ↓	Return ↑	$\mathcal{C}_{\text{train}}$ ↓
w/o noise	2562 +/- 102	**2.6** +/- 0.1	2600 +/- 364	**2.0** +/- 0.2	1661 +/- 270	**1.8** +/- 0.1	2216 +/- 40	**1.8** +/- 0.1
w/ unstructured	**2994** +/- 89	4.8 +/- 0.2	**3394** +/- 64	5.1 +/- 0.1	**2434** +/- 190	3.6 +/- 0.1	2225 +/- 35	3.6 +/- 0.1
w/ OU noise	2692 +/- 68	2.9 +/- 0.1	2849 +/- 267	2.3 +/- 0.0	2200 +/- 53	2.1 +/- 0.1	2089 +/- 25	2.0 +/- 0.0
w/ param noise	2834 +/- 54	2.9 +/- 0.1	3294 +/- 55	**2.1** +/- 0.1	1685 +/- 279	2.2 +/- 0.1	**2294** +/- 40	**1.8** +/- 0.1
w/ gSDE-2	**2987** +/- 85	4.1 +/- 0.2	**3366** +/- 50	4.7 +/- 0.1	**2532** +/- 70	2.8 +/- 0.1	2237 +/- 55	2.8 +/- 0.1
w/ gSDE-4	2798 +/- 41	4.1 +/- 0.2	**3227** +/- 182	3.8 +/- 0.2	2541 +/- 49	2.6 +/- 0.1	**2322** +/- 69	2.6 +/- 0.1
w/ gSDE-8	**2850** +/- 73	4.1 +/- 0.2	**3459** +/- 52	3.9 +/- 0.2	**2646** +/- 45	2.4 +/- 0.1	**2341** +/- 45	2.5 +/- 0.1
w/ gSDE-64	**2970** +/- 132	3.5 +/- 0.1	3160 +/- 184	3.5 +/- 0.1	2476 +/- 99	**2.0** +/- 0.1	**2324** +/- 39	2.3 +/- 0.1
w/ gSDE-Episodic	2741 +/- 115	3.1 +/- 0.2	3044 +/- 106	2.6 +/- 0.1	2503 +/- 80	**1.8** +/- 0.1	**2267** +/- 34	2.2 +/- 0.1

Table 1: Detailed return and continuity cost results for SAC with different type of exploration on PyBullet environments. We report the mean and standard error over 10 runs of 1 million steps. For each benchmark, we highlight the results of the method(s) with the best mean when the difference is statistically significant.

Results Table 1 and Fig. 2 shows the results on the PyBullet tasks and the compromise between continuity and performance. Without any noise ("No Noise" in the figure), SAC is still able to solve partially those tasks thanks to a shaped reward, but it has the highest variance in the results. Although the correlated and parameter noise yield lower continuity cost during training, it comes at a cost of performance. gSDE is able to achieve a good compromise between unstructured exploration and correlated noise by making use of the noise repeat parameter. gSDE-8 (sampling the noise every 8 steps) even achieves better performance with a lower continuity cost at train time. Such behavior is what is desirable for training on a real robot: we must minimize wear-and-tear at training while still obtaining good performance at test time.

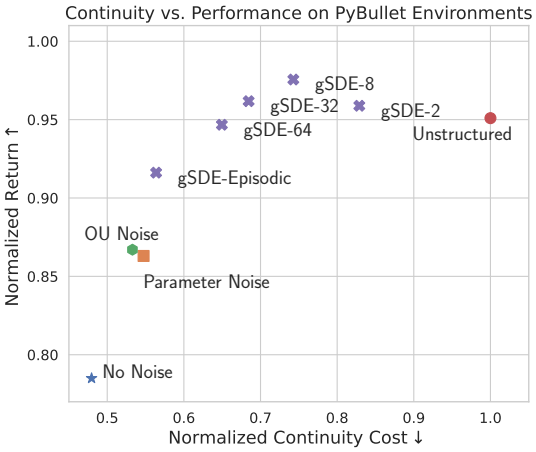


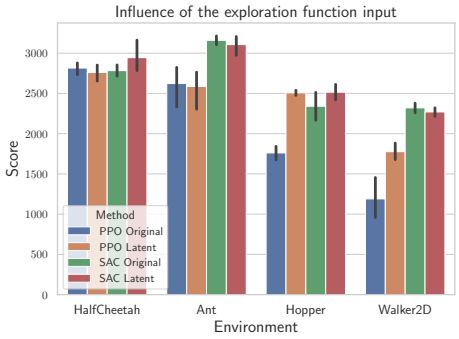
Figure 2: Normalized return and continuity cost of SAC on 4 PyBullet tasks with different type of exploration. gSDE provides a compromise between performance and smoothness.

4.2 Comparison to the Original SDE

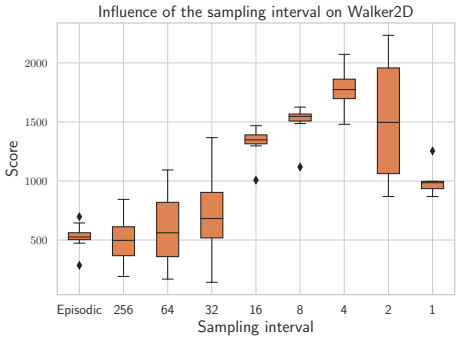
In this section, we investigate the contribution of the proposed modifications to the original SDE: sampling the exploration function parameters every n steps and using policy features as input to the noise function.

Sampling Interval gSDE is a n-step version of SDE, where n allows interpolating between the unstructured exploration $n = 1$ and the original SDE per-episode formulation. This interpolation permits a compromise between performance and smoothness at train time (cf. Table 1 and Fig. 2). Fig. 3b shows the importance of that parameter for PPO on the WALKER2D task. If the sampling interval is too large, the agent would not explore enough during long episodes. On the other hand, with a high sampling frequency $n \approx 1$, the issues mentioned in Sect. 1 arise.

Policy features as input Fig. 3a shows the effect of changing the exploration function input for SAC and PPO. Although it varies from task to task, using policy features ("latent" in the figure) is usually beneficial, especially for PPO. It also requires less tuning and no normalization as it depends only on the policy network architecture. Here, the PyBullet tasks are low dimensional and the state



(a) Exploration function input



(b) Sampling interval (PPO on WALKER2D)

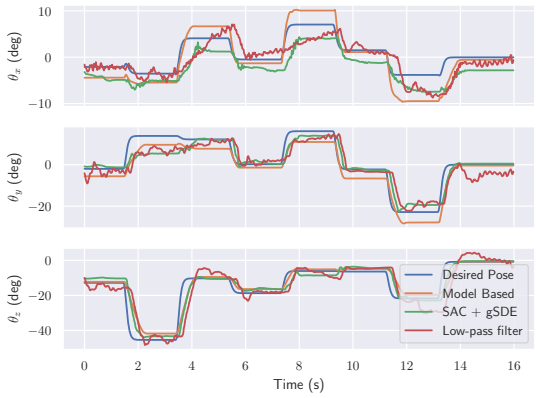
Figure 3: Impact of gSDE modifications over the original SDE on PyBullet tasks. (a) Influence of the input to the exploration function $\epsilon(\mathbf{s}; \theta_\epsilon)$ for SAC and PPO: using latent features from the policy $\mathbf{z}_\mu$ (Latent) is usually better than using the state $\mathbf{s}$ (Original). (b) The frequency of sampling the noise function parameters is crucial for PPO with gSDE.

space size is of the same order, so no careful per-task tuning is needed. Relying on features also allows learning directly from pixels, which is not possible in the original formulation.

Compared to the original SDE, the two proposed modifications are beneficial to the performance, with the noise sampling interval n having the most impact. Fortunately, as shown in Table 1 and Fig. 2, it can be chosen quite freely for SAC. In the appendix, we provide an additional ablation study that shows gSDE is robust to the choice of the initial exploration variance.

4.3 Learning to Control a Tendon-Driven Elastic Robot

(a) Tendon-driven elastic continuum neck in a humanoid robot



(b) Model-Based controller vs RL controller on the real robot

Figure 4: (a) The tendon-driven robot [33] used for the experiment. The tendons are highlighted in orange. (b) The model-based controller and the RL agent performs similarly on an evaluation trajectory.

Experiment setup To assess the usefulness of gSDE, we apply it on a real system. The task is to control a tendon-driven elastic continuum neck [33] (see Fig. 4a) to a given target pose. Controlling such a soft robot is challenging because of the nonlinear tendon coupling, together with the deformation of the structure that needs to be modeled accurately. This modeling is computationally expensive [34, 35] and requires assumptions that may not hold in the physical system.

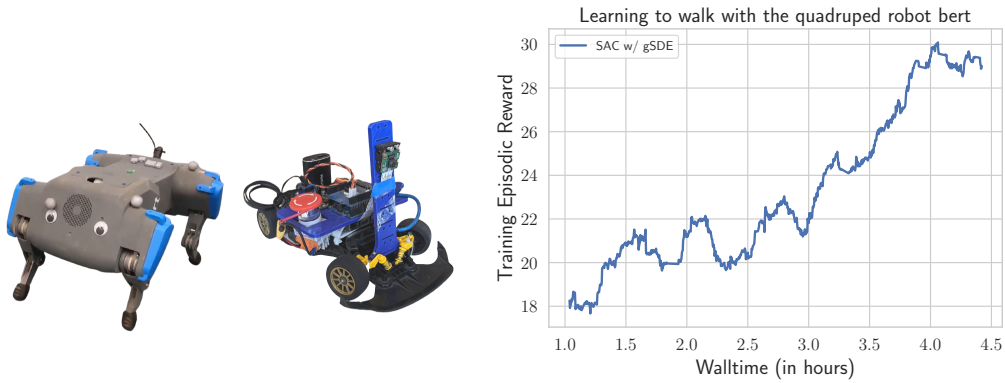
The system is under-actuated (there are only 4 tendons), hence, the desired pose is a 4D vector: 3 angles for the rotation θ_x, θ_y, θ_z and one for the position x. The input is a 16D vector composed of: the measured tendon lengths (4D), the current tendon forces (4D), the current pose (4D) and the target pose (4D). The reward is a weighted sum between the negative geodesic distance to the desired orientation and the negative Euclidean distance to the desired position. The weights are chosen such

that the two components have the same magnitude. We also add a small continuity cost to reduce the oscillations in the final policy. The action space consists in desired delta in tendon forces, limited to $5\,\mathrm{N}$. For safety reasons, the tendon forces are clipped below $10\,\mathrm{N}$ and above $40\,\mathrm{N}$. An episode terminates either when the agent reaches the desired pose or after a timeout of $5\,\mathrm{s}$. The episode is considered successful if the desired pose is reached within a threshold of $10\,\mathrm{mm}$ for the position and $5°$ for the orientation. The agent controls the tendons forces at $30\,\mathrm{Hz}$, while a PD controller monitors the motor current at $3\,\mathrm{kHz}$ on the robot. Gradient updates are directly done on a 4-core laptop, after each episode.

Results We first ran the unstructured exploration on the robot, but had to stop the experiment early: the high-frequency noise in the command was damaging the tendons and would have broken them due to their friction on the bearings. Therefore, as a baseline, we trained a policy using SAC with a hand-crafted action smoothing ($2\,\mathrm{Hz}$ cutoff Butterworth low-pass filter) for two hours.
Then, we trained a controller using SAC with gSDE for the same duration. We compare both learned controllers to an existing model-based controller (passivity-based approach) presented in [34, 35] using a pre-defined trajectory (cf. Fig. 4b). On the evaluation trajectory, the controllers are equally precise (cf. Table 2): the mean error in orientation is below $3°$ and the one in position below $3\,\mathrm{mm}$. However, the policy trained with the low-pass filter is much more jittery than the two others. We quantify this jitter as the mean absolute difference between two timesteps, denoted as *continuity cost* in Table 2.

	Unstructured noise	gSDE	Low-pass filter	Model-Based
Position error (mm)	N/A	2.65 +/- 1.6	1.98 +/- 1.7	1.32 +/- 1.2
Orientation error (deg)	N/A	2.85 +/- 2.9	3.53 +/- 4.0	2.90 +/- 2.8
Continuity cost (deg)	N/A	**0.20 +/- 0.04**	0.38 +/- 0.07	**0.16 +/- 0.04**

Table 2: Comparison of the mean error in position, orientation and mean continuity cost on the evaluation trajectory. We highlight best approaches when the difference is significant. The model-based and learned controllers yield comparable results but the policy trained with a low-pass filter has a much higher continuity cost.



(a) Quadruped and RC car robots

(b) Learning curve on the walking task for the quadruped robot

Figure 5: Additional robots successfully trained using SAC with gSDE directly in the real world.

Additional Real Robot Experiments We also successfully applied SAC with gSDE on two additional real robotics tasks (see Fig. 5a): training an elastic quadruped robot to walk (learning curve in Fig. 5b) and learning to drive around a track using an RC car. Both experiments are fully done on the real robot, without the use of simulation nor filter. We provide in the supplementary material the videos of the trained controllers.

5 Related Work

Exploration is a key topic in reinforcement learning [36]. It has been extensively studied in the discrete case and most recent papers still focus on discrete actions [37, 38].

Several works tackle the issues of unstructured exploration for continuous control by replacing it with correlated noise. Korenkevych et al. [15] use an autoregressive process and introduce two variables that allows to control the smoothness of the exploration. In the same vein, van Hoof et al. [27] rely on a temporal coherence parameter to interpolate between the step- or episode-based exploration, making use of a Markov chain to correlate the noise. This smoothed noise comes at a cost: it requires an history, which changes the problem definition.

Exploring in parameter space [10, 39, 11, 12, 40] is an orthogonal approach that also solves some issues of the unstructured exploration. It was successfully applied to real robot but relied on motor primitives [41, 12], which requires expert knowledge. Plappert et al. [19] adapt parameter exploration to Deep RL by defining a distance in the action space and applying layer normalization to handle high-dimensional space.

Population based algorithms, such as Evolution strategies (ES) or Genetic Algorithms (GA), also explore in parameter space. Thanks to massive parallelization, they were shown to be competitive [42] with RL in terms of training time, at the cost of being sample inefficient. To address this problem, recent works [20] proposed to combine ES exploration with RL gradient update. This combination, although powerful, unfortunately adds numerous hyperparameters and a non-negligible computational overhead.

Obtaining smooth control is essential for real robot, but it is usually overlooked by the DeepRL community. Recently, Mysore et al. [14] integrated a continuity and smoothing loss inside RL algorithms. Their approach is effective to obtain a smooth controller that reduces the energy used at test time on the real robot. However, it does not solve the issue of smooth exploration at train time, limiting their training to simulation only.

6 Conclusion

In this work, we highlighted several issues that arise from the unstructured exploration in Deep RL algorithms for continuous control. Due to those issues, these algorithms cannot be directly applied to learning on real-world robots.

To address these issues, we adapt State-Dependent Exploration to Deep RL algorithms by extending the original formulation: we sample the noise every n steps and replace the exploration function input by learned features. This generalized version (gSDE), provides a simple and efficient alternative to unstructured Gaussian exploration.

gSDE achieves very competitive results on several continuous control benchmarks, while reducing wear-and-tear at train time. We also investigate the contribution of each modification by performing an ablation study: the noise sampling interval has the most impact and allows a compromise between performance and smoothness. Our proposed exploration strategy, combined with SAC, is robust to hyperparameter choice, which makes it suitable for robotics applications. To demonstrate it, we successfully apply SAC with gSDE directly on three different robots.

Although much progress is being made in *sim2real* approaches, we believe more effort should be invested in learning directly on real systems, even if this poses challenges in terms of safety and duration of learning. This paper is meant as a step towards this goal, and we hope that it will revive interest in developing exploration methods that can be directly applied to real robots.

References

[1] 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.

[2] N. J. Nilsson. Shakey the robot. Technical Report 323, Artificial Intelligence Center, SRI International, Menlo Park, CA, USA, 1984. URL http://www.ai.sri.com/shakey/.

[3] 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.

[4] M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018.

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

[6] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)*, 37(4):143, 2018.

[7] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter. Learning agile and dynamic motor skills for legged robots. *arXiv preprint arXiv:1901.08652*, 2019.

[8] 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.

[9] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah. Learning to drive in a day. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8248–8254. IEEE, 2019.

[10] J. Kober and J. R. Peters. Policy search for motor primitives in robotics. In *Advances in neural information processing systems*, pages 849–856, 2009.

[11] T. Rückstiess, F. Sehnke, T. Schaul, D. Wierstra, Y. Sun, and J. Schmidhuber. Exploring parameter space in reinforcement learning. *Paladyn, Journal of Behavioral Robotics*, 1(1): 14–24, 2010.

[12] F. Stulp and O. Sigaud. Robot skill learning: From reinforcement learning to evolution strategies. *Paladyn, Journal of Behavioral Robotics*, 4(1):49–61, 2013.

[13] M. P. Deisenroth, G. Neumann, J. Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

[14] S. Mysore, B. Mabsout, R. Mancuso, and K. Saenko. Regularizing action policies for smooth control with reinforcement learning. 2021.

[15] D. Korenkevych, A. R. Mahmood, G. Vasan, and J. Bergstra. Autoregressive policies for continuous control deep reinforcement learning. *arXiv preprint arXiv:1903.11524*, 2019.

[16] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine. Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103*, 2018.

[17] S. Ha, P. Xu, Z. Tan, S. Levine, and J. Tan. Learning to walk in the real world with minimal human effort. *arXiv preprint arXiv:2002.08550*, 02 2020.

[18] M. Neunert, A. Abdolmaleki, M. Wulfmeier, T. Lampe, J. T. Springenberg, R. Hafner, F. Romano, J. Buchli, N. Heess, and M. Riedmiller. Continuous-discrete reinforcement learning for hybrid control in robotics. *arXiv preprint arXiv:2001.00449*, 2020.

[19] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*, 2017.

[20] A. Pourchot and O. Sigaud. Cem-rl: Combining evolutionary and gradient-based methods for policy search. *arXiv preprint arXiv:1810.01222*, 2018.

[21] 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.

[22] 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.

[23] 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.

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

[25] 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.

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

[27] H. van Hoof, D. Tanneberg, and J. Peters. Generalized exploration in policy search. *Machine Learning*, 106(9-10):1705–1724, oct 2017. Special Issue of the ECML PKDD 2017 Journal Track.

[28] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2019.

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

[30] G. E. Uhlenbeck and L. S. Ornstein. On the theory of the brownian motion. *Physical review*, 36(5):823, 1930.

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

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

[33] J. Reinecke, B. Deutschmann, and D. Fehrenbach. A structurally flexible humanoid spine based on a tendon-driven elastic continuum. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4714–4721. IEEE, 2016.

[34] 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.

[35] B. Deutschmann, M. Chalon, J. Reinecke, M. Maier, and C. Ott. Six-dof pose estimation for a tendon-driven continuum mechanism without a deformation model. *IEEE Robotics and Automation Letters*, 4(4):3425–3432, 2019.

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

[37] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy. Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*, pages 4026–4034, 2016.

[38] I. Osband, J. Aslanides, and A. Cassirer. Randomized prior functions for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 8617–8629, 2018.

[39] F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, and J. Schmidhuber. Parameter-exploring policy gradients. *Neural Networks*, 23(4):551–559, 2010.

[40] O. Sigaud and F. Stulp. Policy search in continuous action domains: an overview. *Neural Networks*, 2019.

[41] J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.

[42] F. Such, V. Madhavan, E. Conti, J. Lehman, K. Stanley, and J. Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 12 2017.