# How does the type of exploration-noise affect returns and exploration on Reinforcement Learning benchmarks?

Jakob Hollenstein[1] Matteo Saveriano[1] Sayantan Auddy[1] Erwan Renaudo[1] Justus Piater[1]

***Abstract*— Deep Reinforcement Learning has shown promising results but remains sensitive to many parameters. Depending on the environment some algorithms work better than others. Each algorithm defaults to a specific type of exploration noise. We investigate whether continuous-control environments react differently to different types of noise. We show that for rollouts of purely random policies, sampling actions from different types of noise, Gaussian and Ornstein-Uhlenbeck, the type of noise has a strong impact on both the achieved episode returns and the exploration, measured in terms of state space coverage. Our experiments show that there is no uniquely preferable exploration noise and that in one type of environments exploration and returns are positively correlated, while in the other type more exploration reduces the achieved returns. We also find that similar exploration can be achieved by different types of noise, but that the cost can be highly different. Our results also support claims that the choice of exploration measure matters.**

## I. INTRODUCTION

In recent years Deep-Reinforcement Learning (D-RL) has seen astonishing success as well as startling failures: results are noisy and often sensitive to many parameters [11]. Often the benefit of one algorithm over another varies, amongst other things, depending on the task — the *environment*.

One important aspect influencing the performance of a D-RL algorithm on a specific environment is its ability to choose actions that help gather the necessary information. That is, to choose actions that allow for sufficient *exploration*. The way actions are chosen are particularly important for continuous-control, continuous-action space, tasks because enumeration of actions is not possible. Since many robotics tasks also feature continuous-action spaces, this is important for D-RL in the context of robotics.

Many D-RL algorithms employ undirected exploration-noise to discover useful actions—for example by adding exploration-noise on top of the actions chosen by the policy [6], [10] or by employing stochastic policies and sampling from those policies [3], [8], [13].

The most common types of noise are *Gaussian* noise or noise from a temporally-correlated *Ornstein-Uhlenbeck* process: some algorithms typically use Gaussian noise (TD3 [10]) or samples from a parameterized-Gaussian (SAC [13]), while other algorithms default to Ornstein-Uhlenbeck noise (DDPG [6]). If one of these algorithms performs better than another on a specific environment, could the choice of noise and the parameters of the noise generating process – the *noise configuration* – have a relevant impact on the performance? Do different environments require the same type of noise or do we need different noise configurations for different environments?

At the beginning of a D-RL learning process data is usually collected in a purely random fashion and until more rewarding action sequences are collected, the algorithm has to rely on purely random exploration.

One possible way noise could affect the learning process is by increasing or decreasing the *exploration*. In this work, we measure exploration by the diversity of the interaction data an agent collects.

Since purely random policies are similar to the initial stages of a D-RL learning process, we investigate the impact of such purely random policies driven by different types of noise, on different environments and compare their achieved returns and exploration. This allows us to focus on the exploration noise without having to control for the learning aspect of each algorithm. We measure the exploration as the diversity of collected interaction and show that on all environments the noise configuration *does have* an important impact. We found that on some environments higher exploration is positively correlated with higher returns of the random policies, while on other environments it is negatively correlated. Furthermore, while different types of noise can lead to similar exploration, we found that the return can be quite different if the return is related to the expended effort.

## II. METHOD

We use two types of noise, Gaussian and Ornstein-Uhlenbeck, described below and measure the exploration by the diversity of the collected interactions or more precisely by the coverage of the state space.

An intuitive way to measure state space coverage would be to use histogram based approaches, such as the Bin-Count [2], [16] which divides the state-space into equally many bins along each dimension and measures the ratio of non-empty bins to the total number of bins. Since the number of bins, as the product of divisions along each dimension, grows exponentially with the dimensionality, the required number of data points required to fill each bin with at least one point, becomes prohibitively large very quickly. Therefore, we do not report results using a histogram based

[1]Department of Computer Science, University of Innsbruck, Innsbruck, Austria
{name.surname}@uibk.ac.at

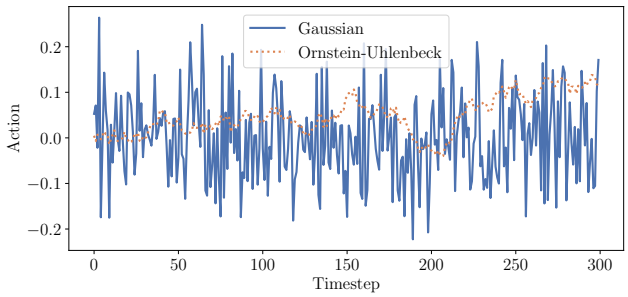



Fig. 1: Default action noises for algorithms include Gaussian noise (TD3) and Ornstein-Uhlenbeck noise (DDPG). This figure shows an example rollout for each of these noise types.

measure, but focus on measures that work with fewer points and higher dimensional spaces.

### A. Measures

Following the naming in [16] we use three different measures $\mathrm{X}_{\mathrm{BBM}}$, $\mathrm{X}_{\mathrm{NN}}$ and $\mathrm{X}_{\mathrm{Urel}}$ to assess the exploration:

*(i)* $\mathrm{X}_{\mathrm{BBM}}$ measures the spread of the data. The bounding box mean [14] creates a $d$ dimensional bounding box around the collected data $D = \{\ldots, \boldsymbol{s}^{(j)}, \ldots\}$ and measures the mean of the side-lengths of this bounding box:

$$\mathrm{X}_{\mathrm{BBM}} = \frac{1}{d}\sum_{i}^{d}\left[\max_j s_i^{(j)} - \min_j s_i^{(j)}\right]$$

*(ii)* $\mathrm{X}_{\mathrm{NN}}$, the nuclear norm [14] estimates the covariance matrix $C$ of the data and measures data spread by the trace, the sum of the eigenvalues of the estimated covariance:

$$\mathrm{X}_{\mathrm{NN}}(D) := \mathrm{trace}\left(C(D)\right)$$

*(iii)* $\mathrm{X}_{\mathrm{Urel}}(D)$, the Uniform-relative-entropy [16], assesses the uniformity of the collected data, by measuring the exploration as the symmetric divergence between a uniform prior over the state space $U$ and the data distribution $Q_D$:

$$\mathrm{X}_{\mathrm{Urel}}(D) = -D_{\mathrm{KL}}\left(U||Q_D\right) - D_{\mathrm{KL}}\left(Q_D||U\right) \qquad (1)$$

since $Q_D$ is only available through estimation, $D_{\mathrm{KL}}$ is estimated using the nearest-neighbor-ratio-estimator [7].

We investigate the exploration using the $\mathrm{X}_{\mathrm{Urel}}$, $\mathrm{X}_{\mathrm{BBM}}$ and $\mathrm{X}_{\mathrm{NN}}$ measures. In some environments, the state space limits are not properly provided, i.e. $s \in (-\infty, \infty)$; in these cases we use the empirical limits observed across all experiments on the respective environment.

### B. Noise types

Ornstein-Uhlenbeck action noise is created by the following temporal process, with each action dimension calculated independently of the other dimensions:

$$a_t = a_{t-1} + \theta(\mu - a_{t-1}) \cdot dt + \sigma\sqrt{(dt)} \cdot \varepsilon_t$$
$$a_0 = \mathbf{0} \qquad \varepsilon_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$$

In widely used RL-algorithm implementations [12] the default parameters are $\theta = 0.15$, $dt = 0.01$, $\mu = 0 \cdot \mathbf{1}$, $\sigma = 0.1 \cdot \boldsymbol{I}$.

The Gaussian noise is temporally uncorrelated and is typically applied on symmetric action spaces with commonly-used values [12], [15] of $\mu = 0$ and $\sigma = 0.1$ with $\Sigma = \boldsymbol{I} \cdot \sigma$. Actions $a_t$ are sampled according to

$$a_t \sim \mathcal{N}(\mu, \Sigma)$$

The noise generation assumes valid actions in the interval $[-1, 1]$. These actions are then scaled and clipped according to the actual action limits of the environment.

The difference between these two types of noise is illustrated in Fig. 1. A temporal correlation of the Ornstein-Uhlenbeck noise and the high amplitudes and wide variation between time steps of the Gaussian noise are clearly visible.

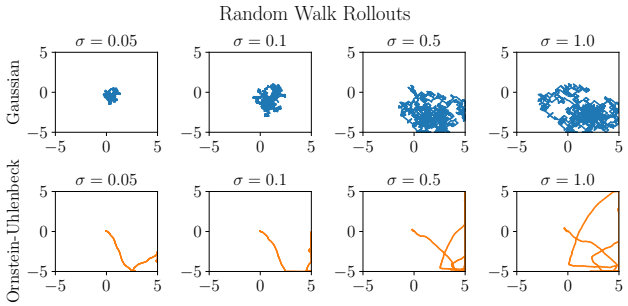



Fig. 2: Each plot shows one rollout trace using Gaussian noise (top) or Ornstein-Uhlenbeck noise (bottom) on our RandomWalk2d-v1 environment. The environment consists of a 2D state space within the limits $-5$ to 5. The agent can move in x and y direction. The line plots indicate the sequence of visited states—positions—starting from an initial position close to the center. (left–right) different noise parameters $\sigma$.

We evaluate the impact of these two types of noise on several common reinforcement learning environments: InvertedPendulumSwingupPyBulletEnv$-$v0 [5], [9], MountainCarContinuous$-$v0 [4], HopperPyBulletEnv$-$v0 [5], [9] and our own RandomWalk2d$-$v1 (see Appendix V-D). The latter is a simple environment to showcase the behavior of these random policies on 2D navigation in a limited area. Example rollouts on this 2D navigation environment are shown in Fig. 2. The $\sigma$ parameters are used as described above.

## III. RESULTS

Fig. 3 shows the results for environment interaction data collected by random policies consisting either of Gaussian noise (G) or Ornstein-Uhlenbeck noise (OU) in terms of the mean returns and exploration measured by $\mathrm{X}_{\mathrm{Urel}}$, $\mathrm{X}_{\mathrm{BBM}}$ and $\mathrm{X}_{\mathrm{NN}}$. For each environment, noise type and $\sigma$ setting, 50 independent experiments with respectively different seeds were conducted. In each experiment data were collected for 100000 steps, resulting in multiple episodes. The returns were calculated as the sum of rewards per episode. Then

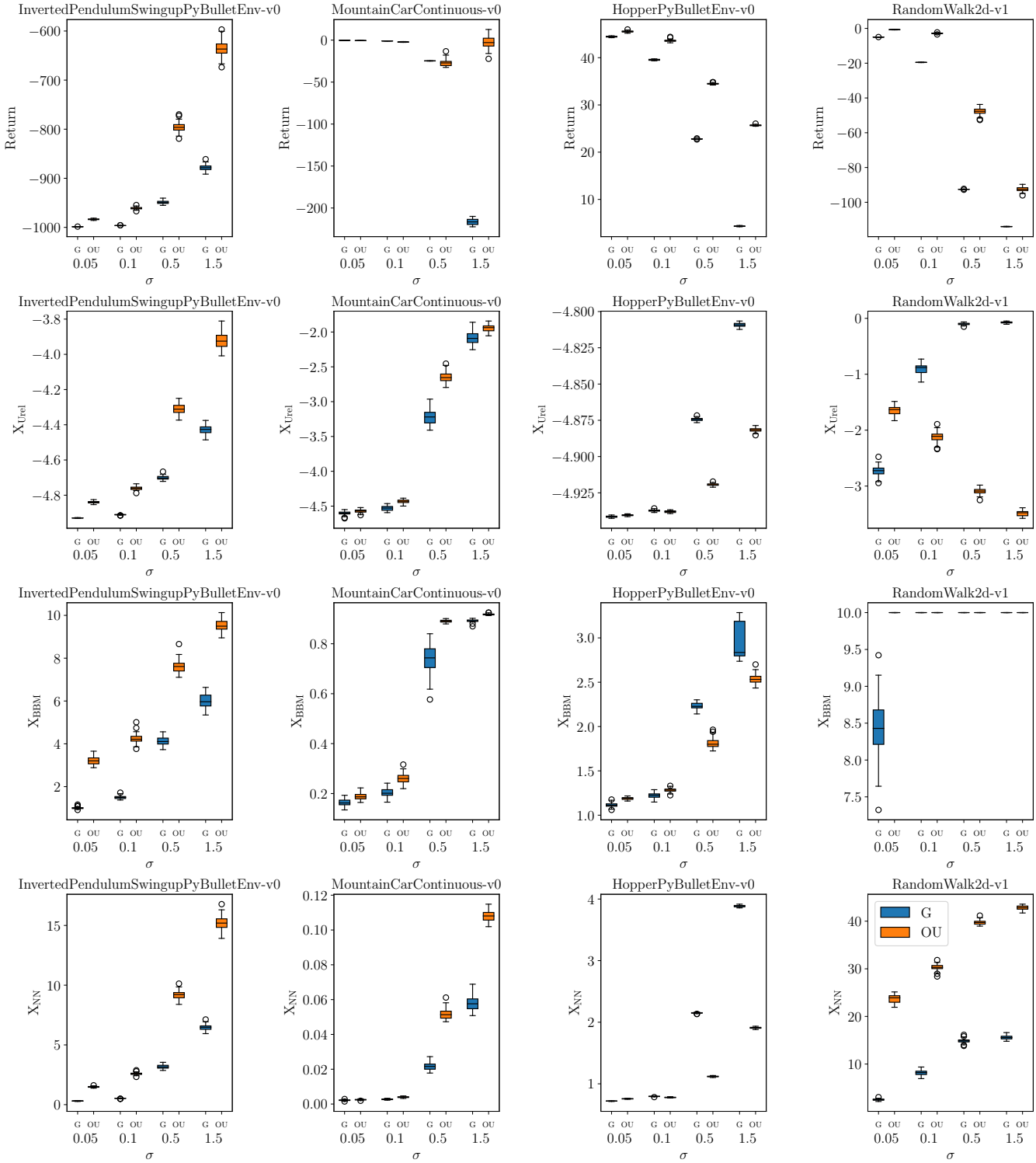



Fig. 3: Each plot shows evaluations for random policies consisting either of Gaussian noise (G) and Ornstein-Uhlenbeck noise (OU). The x-axis indicates the types and the used $\sigma$ parameters. The same $\sigma$ parameter is used for a pair of (G) and (OU) evaluations. (Left to Right) different environments used for evaluation. Top to bottom: (first row) mean returns on the y-axis, (second row) exploration in terms of $X_{Urel}$ on the y-axis, measuring the divergence between a uniform prior over the state space and the collected data, (third row) exploration measured by $X_{BBM}$ on the y-axis, measuring the bounding-box-side-length of the collected data. This measure has a tendency to saturate when data points reach the state space limits. (fourth row) exploration measured by $X_{NN}$ on the y-axis, measuring the spread of the data. $X_{NN}$ overestimates exploration when data points are concentrated around the limits.

the mean of these episode returns over all episodes in the 100000 steps was returned. The 50 independent runs yield 50 mean-return values which are summarized by one box-plot.

### A. Discussion

1) (Fig. 3, left) InvertedPendulumSwingupPyBulletEnv-v0 [9] shows a positive correlation between the exploration (second–fourth row) and the returns (top). Both the returns and the exploration are higher for the Ornstein-Uhlenbeck noise than for the Gaussian noise and both the exploration and return increase with larger $\sigma$ parameters. The better performance of the Ornstein-Uhlenbeck noise in this setting is expected, since the system needs to build up energy. Subsequent actions sampled from uncorrelated noise are more likely to undo energy buildup. In this environment the exploration measures $X_{BBM}$, $X_{NN}$, and $X_{Urel}$ agree with each other.
2) (Fig. 3, 2nd from left) MountainCarContinuous-v0 [4] shows a very slight preference for the Ornstein-Uhlenbeck noise in terms of exploration (third–fourth row). Similar to the InvertedPendulumSwingup-task, the system has to build up energy. Interestingly the $\sigma$ parameter appears to be even more important in terms of exploration than the type of noise. In this environment, the expended energy (action amplitude) is penalized and only reaching the high point of the mountain yields positive reward. While changes in the $\sigma$ parameter account for the exploration, the returns (first row) are quite different: the Gaussian noise expends a lot more effort to achieve similar exploration when compared to the Ornstein-Uhlenbeck noise and subsequently yields more negative returns. In this environment the exploration measures $X_{BBM}$, $X_{NN}$, and $X_{Urel}$ (second–fourth row) also roughly agree with each other, although we can see a saturation for $X_{BBM}$ (third row) in high exploration cases, and presumably a preference of $X_{NN}$ (fourth row) for extreme points as noted by [16].
3) (Fig. 3, 2nd from right) HopperPyBulletEnv-v0 [9] shows a negative correlation between the exploration (second–fourth row) and the achieved return (first row). Presumably this indicates an easier to explore state space— when compared to the under actuated dynamics of the InvertedPendulumSwingup and the MountainCar—in such a case, the noise function could easily reach a large part of the state space and collect interactions far away from reward-yielding regions. This interpretation is supported by the reward structure of the Hopper environment which also contains an energy-cost-term.
4) (Fig. 3, right) RandomWalk2d-v1 (see Appendix V-D) shows a large difference in exploration $X_{Urel}$ (second row) between the Gaussian and Ornstein-Uhlenbeck noise, in line with the rollouts illustrated in Fig. 2. Here the Gaussian noise is preferable, as the Ornstein-Uhlenbeck noise is likely to get the agent stuck in the state space limits. In line with [16] the $X_{BBM}$ (third row) saturates when the state space limits are reached. $X_{NN}$ (fourth row) prefers extreme points and shows a preference for Ornstein-Uhlenbeck noise, even though from Fig. 2 the problematic behavior of Ornstein-Uhlenbeck noise is clearly visible. In this environment we penalize actions by $a^T a$, as such the Gaussian noise incurs more negative reward (first row) than the Ornstein-Uhlenbeck noise.

## IV. Conclusion / Summary and Outlook

In this work we evaluated the impact of the type of exploration noise on the exploration, as state space coverage, with respect to different environments and the relation to returns achievable with the respective noise. We found that the type of noise in fact has a strong impact in terms of exploration and return. This could be part of the reason why some D-RL algorithms work better on some environments and worse on others (e.g. [11]).

We found that the relation between exploration and returns depends on the environment: depending on the environment the correlation between exploration and return can be positive, or negative and as such a unique selection of the noise to maximize exploration would not yield the highest returns.

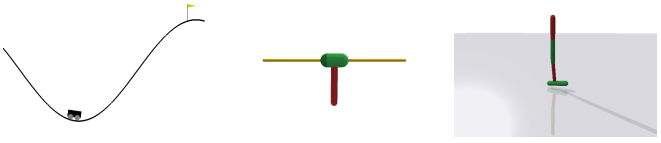
We further found that also the amplitude ($\sigma$) of the noise generating function has a very important impact that can be even larger than the impact of the type of noise.

Lastly, we found that, depending on the environment similar exploration can yield very different returns. For example, in MountainCar experiment, Ornstein-Uhlenbeck and Gaussian noise yield similar exploration, but the returns are far better for Ornstein-Uhlenbeck policies.

Our results provide some indication that in underactuated, hard-to-explore environments selecting the exploration noise to maximize exploration is beneficial. While in environments that are more easily explored, less diverse exploration is required. We believe that this should be helpful in finding good starting points for hyperparameter optimization in D-RL.

While we can see a strong impact of the noise configuration—type and $\sigma$—on both exploration and return of random policies, it is less clear whether this immediately also applies to the D-RL learning process. This question is subject of our ongoing research.

## V. Appendix



(a) (b) (c)

Fig. 4: Illustrations of the used environments: (a) MountainCarContinuous-v0, (b) InvertedPendulumSwingupPyBulletEnv-v0, (c) HopperPyBulletEnv-v0

### A. InvertedPendulumSwingupPyBulletEnv-v0

In the inverted pendulum swing up a pendulum mounted on a cart or linear actuator has to be swung up as illustrated in Fig. 4b. A commonly used implementation is provided by [4], requiring a license for the employed physics engine. The PyBullet physics engine [5] provides a free and open source alternative. PyBullet also includes alternative implementations for common environments from [4]. In particular we use the implementation by [9]. State space: $5d$, action space: $1d$.

### B. MountainCarContinuous-v0

In the Mountain Car environment [1] an under powered car has to be driven up a hill, reaching a resting position on top. We use the implementation by [4]. State space: $2d$, action space: $1d$.

### C. HopperPyBulletEnv-v0

The Hopper is a simple locomotion task, illustrated in Fig. 4c. Similar to above the implementation by [5], [9] is used, attempting to replicate the environments defined by [4]. State space: $15d$, action space: $3d$.

### D. RandomWalk2d-v1

For RandomWalk2d-v1 illustrations of rollout traces are shown in Fig. 2. The environment can be viewed as a two-dimensional navigation task in a spatially constrained environment. Movement is velocity controlled in $x$ and $y$ direction. State space: $2d$, action space: $2d$.

Since RandomWalk2d-v1 is a custom environment we defined for this paper, we briefly describe its mechanics here:

The states are $s \in \mathbb{R}^2$ and have the limits $s_i \in [-5,5]$, the actions $a \in \mathbb{R}^2$ and $a_i \in [-0.25,0.25]$, the integration delta time $dt = 1.0$. The states are integrated over time $t$ as:

$$s^{(t+1)} = s^{(t)} + a^{(t)} \cdot dt \quad \text{subject to the limits } s_i \in [-5,5]$$

The initial states $s^{(0)}$ are sampled

$$s^{(0)} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I} \cdot 1.0)$$

and rewards $r^{(t)}$ are calculated as

$$r^{(t)} = -\sum_i {a_i^{(t)}}^2$$

## References


[1] A. W. Moore, "Efficient memory-based learning for robot control," 1990.

[2] E. Glassman and R. Tedrake, "A quadratic regulator-based heuristic for rapidly exploring state space," in *2010 IEEE Int. Conf. Robotics and Automation (ICRA)*, IEEE, 2010.

[3] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust Region Policy Optimization," in *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15, JMLR.org, 2015, pp. 1889–1897.

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

[5] E. Coumans and Y. Bai, "PyBullet, a Python module for physics simulation for games, robotics and machine learning," 2016.

[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," in *Proc. 4th Int. Conf. Learning Representations, (ICLR)*, 2016.

[7] M. Noshad, K. R. Moon, S. Y. Sekeh, and A. O. Hero, "Direct estimation of information divergence using nearest neighbor ratios," in *2017 IEEE International Symposium on Information Theory (ISIT)*, 2017, pp. 903–907.

[8] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[9] B. Ellenberger, "PyBullet gymperium," *GitHub repository*, 2018.

[10] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," in *International Conference on Machine Learning*, PMLR, 2018, pp. 1587–1596.

[11] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th Innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, S. A. McIlraith and K. Q. Weinberger, Eds., AAAI Press, 2018, pp. 3207–3214.

[12] A. Hill, A. Raffin, M. Ernestus, A. Gleave, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable Baselines," *GitHub repository*, 2018.

[13] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft Actor-Critic Algorithms and Applications," *arXiv:1812.05905 [cs, stat]*, 2019.

[14] Z. Zhan, B. Aytemiz, and A. M. Smith, "Taking the scenic route: Automatic exploration for videogames," in *KEG@ AAAI*, 2019.

[15] A. Raffin, "RL baselines3 zoo," *GitHub repository*, 2020.

[16] J. Hollenstein, A. Sayantan, M. Saveriano, E. Renaudo, and J. Piater, "How do offline measures for exploration in reinforcement learning behave?" English, in *Knowledge Based Reinforcement Learning Workshop at IJCAI-PRICAI 2020, Yokohama, Japan*, 2021.