Supplementary Material

PINK NOISE IS ALL YOU NEED

A COLORED NOISE AND ORNSTEIN-UHLENBECK NOISE

Colored noise has an interesting property that was not mentioned in the main text: integrating a colored noise signal with parameter β again yields a colored noise signal, only with parameter $\beta + 2$. This stems from the property of the Fourier transform that an integration in the time domain corresponds to a multiplication with $(i2\pi f)^{-1}$ in the frequency domain. Let $v(t)$ be the original colored noise signal with $|\hat{v}(f)|^2 \propto f^{-\beta}$. Then the PSD of $x(t) = \int_0^t v(\tau)\,\mathrm{d}\tau$ is

$$|\hat{x}(f)|^2 = \left|\mathcal{F}\left[\int_0^t v(\tau)\,\mathrm{d}\tau\right](f)\right|^2 = \left|\frac{1}{i2\pi f}\hat{v}(f)\right|^2 \propto f^{-2}|\hat{v}(f)|^2 \propto f^{-(\beta+2)}. \tag{1}$$

From this, and the definition of white noise as colored noise with $\beta = 0$, it follows that Brownian motion (integrated white noise) is also colored noise with parameter $\beta = 2$. In Figure A.1, sampled signals of most of the noise types we use in this paper are shown, and in Figure A.2, we plot the power spectral densities of some of these.

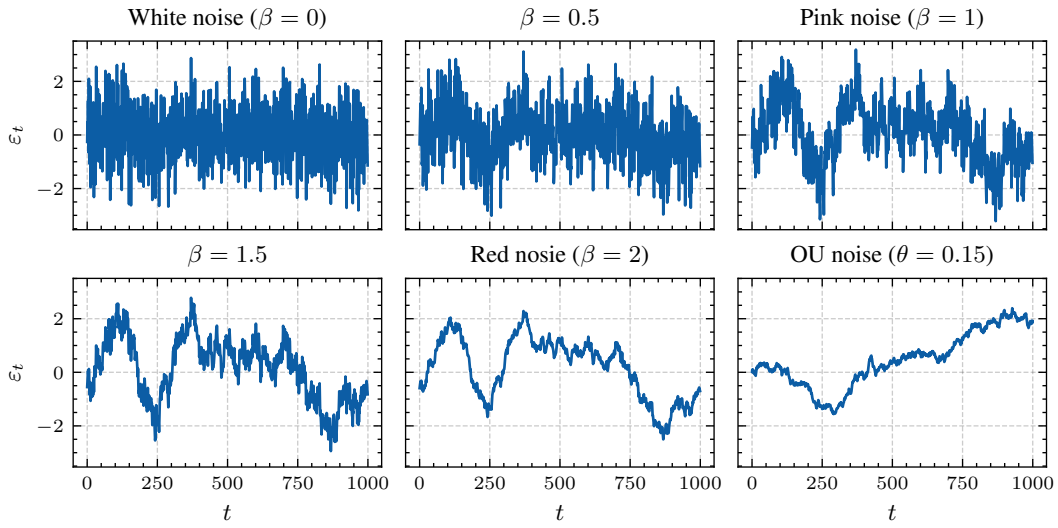

Figure A.1: Sampled signals from various action noise processes with noise scale $\sigma = 1$. The exception is OU noise, whose noise scale is adjusted such that $\mathrm{var}[\varepsilon_t] = 1$ (see discussion below).

We generate colored noise using the procedure described by Timmer & Koenig (1995), based on the Fast Fourier Transform (FFT) (Cooley & Tukey, 1965). This method is very efficient, as it only requires sampling a Gaussian signal in the frequency space (where the PSD is shaped), and then transforming it to the time domain via the FFT. In particular, this procedure is faster than sampling an Ornstein-Uhlenbeck signal (using the most common procedure, which we describe below). We use the `colorednoise` Python package (https://github.com/felixpatzelt/colorednoise) to sample colored noise signals, and always sample signals of the complete episode length (which we denote by $\varepsilon_{1:T} \sim \mathrm{CN}_T(\beta)$). The Python implementation contained a bug, which among other things made it so the generated "white noise" was correlated, and our fix of this bug is included as of version 2.1.0 of the package. Colored noise sampled according to this procedure is stationary and Gaussian: the signals are marginally identical to standard Gaussian distributions, i.e. $p(\varepsilon_t) = \mathcal{N}(\varepsilon_t \mid 0, 1)$. The only difference to white noise (independent Gaussian samples at every time step) is that they are temporally correlated: $p(\varepsilon_t, \varepsilon_{t'}) \neq p(\varepsilon_t)p(\varepsilon_{t'})$. This is shown empirically on the example of pink noise in Figure A.3.

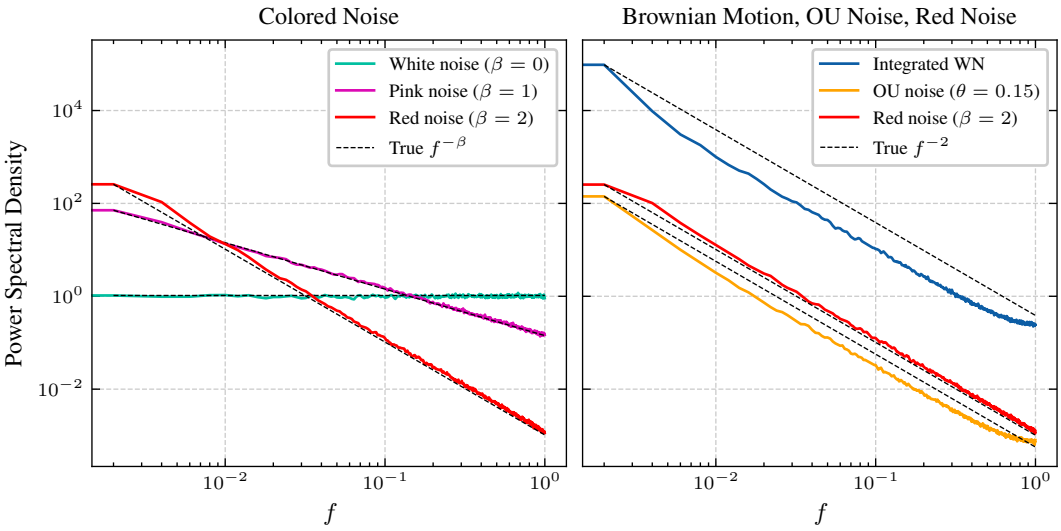

Figure A.2: Left: The power law trends can be seen in the PSDs of sampled colored noise signals. Right: Brownian motion, here generated by integrating white noise sampled from $\mathcal{N}(0, 1)$, is compared to two related stationary noises: Ornstein-Uhlenbeck noise ($\theta = 0.15$), and red noise. The similarity between OU and red noise is visible. All signals are of length $T = 1000$.

A.1 Ornstein-Uhlenbeck noise generation and variance correction

Also included in Figure A.3 is Ornstein-Uhlenbeck (OU) noise. It can be seen that OU noise starts out as non-stationary but quickly converges to the same marginal distribution $p(\varepsilon_t) = \mathcal{N}(\varepsilon_t \mid 0, 1)$ as the other noise types. Important to note is that all these noise types are suitable for use as action noise only because they are (or quickly become) stationary, and hence their variance does not grow without bounds (contrary to that of Brownian motion). The property that all noise types have the same marginal distribution shows that our results are only due to a change in the *temporal correlation* of the action noise, not in the scale or shape of the distribution, as this is the same as of regular Gaussian white noise. To make sure that OU noise converges to a standard Gaussian marginal distribution we cannot use a noise scale of $\sigma = 1$, but have to correct it. Ornstein-Uhlenbeck noise can be defined by the stochastic differential equation

$$\mathrm{d}x_t = -\theta x_t \,\mathrm{d}t + \sigma \,\mathrm{d}w_t \,, \tag{2}$$

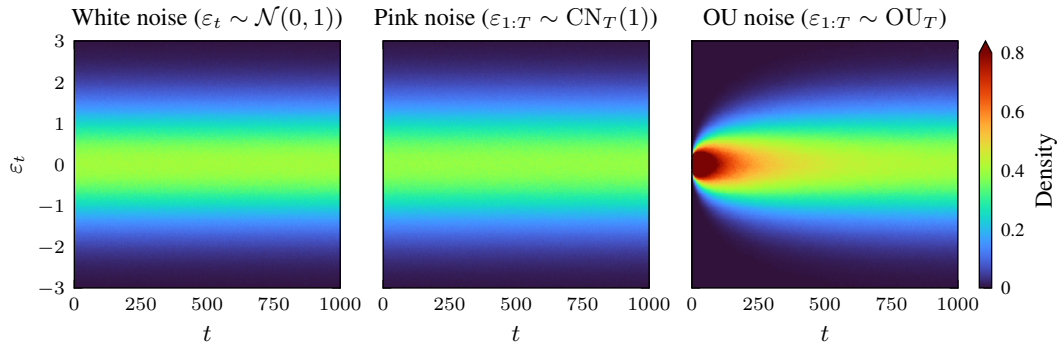

Figure A.3: The colored noise we use as action noise has the same marginal distribution as independent Gaussian samples. We sampled 3×10^5 action noise signals of length $T = 1000$ from each of the following random processes: independent Gaussian samples (white noise, left), pink noise (center), Ornstein-Uhlenbeck noise (right). At every time step t we show a histogram density estimate over action noise values ε_t. This shows that our results are only due to the increased temporal correlation of the action noise signals, as the marginal distributions remain unchanged from white noise.

where w_t is a Wiener process (integrated white noise with the property that $w(t)-w(t') \sim \mathcal{N}(0, t-t')$ for any $0 \leq t' < t$). This definition of Ornstein-Uhlenbeck noise is equivalent to the Langevin equation (3) in the main text, but is nicer to work with, as the white noise process η_t is ill-defined as the derivative of the Wiener process. We sample OU noise signals by discretizing the equation above:

$$x[t + \Delta t] = x[t] - \theta x[t]\Delta t + \sigma\sqrt{\Delta t}\varepsilon, \tag{3}$$

where $\varepsilon \sim \mathcal{N}(0, 1)$. Denoting $x_t \coloneqq x[t\Delta t]$ (with $x_{-1} = 0$) and $\varepsilon_t \sim \mathcal{N}(0, 1)$ for all $t \in \mathbb{N}_0$, it can be seen that

$$\begin{aligned}
x_0 &= \sigma\sqrt{\Delta t}\varepsilon_0 \\
x_1 &= x_0 - \theta x_0 \Delta t + \sigma\sqrt{\Delta t}\varepsilon_1 \\
&= \sigma\sqrt{\Delta t}(1 - \theta\Delta t)\varepsilon_0 + \sigma\sqrt{\Delta t}\varepsilon_1 \\
x_2 &= \sigma\sqrt{\Delta t}(1 - \theta\Delta t)^2\varepsilon_0 + \sigma\sqrt{\Delta t}(1 - \theta\Delta t)\varepsilon_1 + \sigma\sqrt{\Delta t}\varepsilon_2 \\
&\vdots \\
x_t &= \sigma\sqrt{\Delta t}\sum_{\tau=0}^{t}(1 - \theta\Delta t)^{t-\tau}\varepsilon_\tau.
\end{aligned}$$

Thus, as a sum of zero-mean Gaussian distributions, the marginal distribution is:

$$\begin{aligned}
p(x_t) &= \sigma\sqrt{\Delta t}\mathcal{N}\left(0, \sum_{\tau=0}^{t}((1 - \theta\Delta t)^{t-\tau})^2\right) \\
&= \mathcal{N}\left(0, \sigma^2\Delta t\sum_{\tau=0}^{t}(1 - \theta\Delta t)^{2\tau}\right).
\end{aligned}$$

The variance of this distribution is a geometric series which converges as $t \to \infty$ if $(1 - \theta\Delta t)^2 < 1$, which holds if $0 < \theta\Delta t < 2$. It is interesting to note that if $\theta\Delta t = 1$, then Eq. (3) yields white noise, as it reduces to $x_t = \sigma\sqrt{\Delta t}\varepsilon$. On the other hand, if $\theta\Delta t = 0$, the equation describes integrated white noise (Brownian motion), which is known to have unbounded variance. If $1 < \theta\Delta t < 2$, then the signal exhibits negative temporal correlation, which follows from Eq. (3). If the geometric series converges, then the limiting variance is given by

$$\frac{\sigma^2\Delta t}{1 - (1 - \theta\Delta t)^2}.$$

We can thus ensure a standard Gaussian marginal distribution (in the limit) by setting the noise scale to a "corrected" value of

$$\sigma = \sqrt{\frac{1 - (1 - \theta\Delta t)^2}{\Delta t}}, \tag{4}$$

which is how we set the OU noise scale throughout the paper to make the comparison with white and colored noise fair. In Figure A.3, it can be seen that this limiting marginal distribution is reached fairly quickly. In Section B, we also report Ornstein-Uhlenbeck results with the more common choice of $\sigma = 1$, which we find to generally perform slightly worse (cf. Figure B.1).

If the variance is corrected, then $\theta\Delta t$ is the only parameter of OU noise, such that e.g. ($\theta = 0.3, \Delta t = 1$) is equivalent to ($\theta = 30, \Delta t = 0.01$). This immediately follows by plugging Eq. (4) into Eq. (3), yielding

$$x_{t+1} = (1 - \theta\Delta t)x_t + \sqrt{1 - (1 - \theta\Delta t)^2}\varepsilon_t,$$

which only contains the product $\theta\Delta t$ as a parameter. In this paper we thus set $\Delta t = 0.01$ without loss of generality. In the main text we also only consider OU noise as a replacement for strongly correlated Brownian motion and always set $\theta = 0.15$, as this is the most common default setting used in practice.[1] However, as noted in the discussion above, Ornstein-Uhlenbeck noise can also exhibit intermediate temporal correlation between white noise and Brownian motion, by setting $0 < \theta < 100$ (i.e. $0 < \theta\Delta t < 1$). This raises the question of whether there is a certain parameterization of OU noise which is as general as pink noise.

[1]We chose these values for Δt and θ because these are the default choices the RL libraries we consider (Raffin et al., 2021; Pardo, 2020). Lillicrap et al. (2016) also recommend $\theta = 0.15$. If the variance is not corrected (we report these experiments in Section B), then the choice of Δt does make a difference.

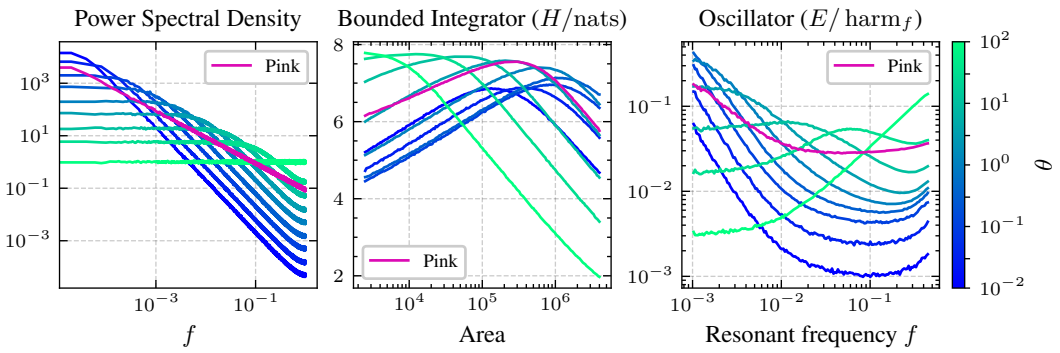

Figure A.4: Left: Power spectral densities of OU noise. OU noise interpolates between white noise and Brownian motion by changing the cutoff frequency of a low-pass filter which filters white noise. Center: Entropy achieved by OU noise of different θ on the bounded integrator environment. No θ achieves a higher worst-case entropy than pink noise. Right: Energy achieved by OU noise of different θ on the harmonic oscillator environment. No θ achieves a worst-case energy that comes close to the one of pink noise.

A.2 Generality of Ornstein-Uhlenbeck noise

The way in which OU noise interpolates between white noise and Brownian motion by choosing $\theta \in (0, 100)$ is very different to colored noise with $\beta \in (0, 2)$. We have shown (e.g. in Figure A.2) that colored noise with intermediate temporal correlation has a power-law power spectral density with intermediate exponent (or slope in the log-log plot). On the other hand, Ornstein-Uhlenbeck noise can be interpreted as a "leaky integration" of white noise, i.e. white noise passed through a low-pass filter. How "leaky" this integrator is, is controlled by the parameter θ: if $\theta = 0$ then the integrator is ideal, resulting in integrated white noise (Brownian motion with diverging variance). If $\theta = 100$ (with $\Delta t = 0.01$), then the integrator is "completely leaky" (an all-pass filter) and the white noise passes through without being integrated. In terms of the power spectral density this change in θ corresponds to shifting the cutoff frequency of the low-pass filter. This is shown on the left in Figure A.4 for $\theta \in \{0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30, 100\}$.

For an action noise type to be *general* (cf. Section 6), we want it to work well on all environments. In the power spectral density plots, it can already be seen that pink noise distributes power over the frequencies much more "generally" than Ornstein-Uhlenbeck noise of any θ: At any given frequency f, pink noise exhibits higher power than most values of θ, and all values of θ have lower power than pink noise at most frequencies. Why this makes pink noise a more general action noise can be made more concrete by revisiting the bounded integrator and harmonic oscillator environments introduced in Section 6. The *generality* of a noise measures how robust it is to the choice or parameterization of the environment: The most general noise type is the one which performs best on the most adversarial environment parameterization. Thus, the most general θ for an environment parameterized by a parameter α solves the following optimization problem:

$$\max_{\theta} \min_{\alpha} \operatorname{perf}(\alpha, \theta),$$

where the performance metric $\operatorname{perf}(\alpha, \theta)$ should be normalized appropriately such that the maximum performance attainable for different values of α is identical. This can be ensured by simply dividing by the performance attained by the best θ for each value of α:

$$\max_{\theta} \underbrace{\min_{\alpha, \theta'} \frac{\operatorname{perf}(\alpha, \theta)}{\operatorname{perf}(\alpha, \theta')}}_{\operatorname{generality}(\theta)}. \tag{5}$$

This gives the worst-case performance of the most general noise in terms of the best possible performance achievable by changing the noise type on this worst-case environment. By replacing the expression $\operatorname{perf}(\alpha, \theta)$ by $\operatorname{perf}(\alpha, \text{pink})$ and removing the maximization over θ, we can also calculate the generality of pink noise.

As discussed in Section 6, the performance of a noise type on the bounded integrator and oscillator environments is given by the achieved entropy and energy, respectively. This is shown for all values θ (as well as for pink noise) in Figure A.4, where the parameterization parameter α is the environment size for the bounded integrator and the resonant frequency for the harmonic oscillator. It can already be seen that on both environments, for each choice of θ there exists a parameter α where the performance of θ is worse than the worst-case performance of pink noise. This can be quantified by calculating the generality of each θ and pink noise on these environments according to Eq. (5). On the bounded integrator, the maximum generality of OU noise is 77%, and on the oscillator environment the maximum generality is 9.1%. On both environments, the maximum is attained by $\theta = 3$. Pink noise achieves generalities of 79% and 22% on the bounded integrator and oscillator environments, respectively. This gives further evidence that pink noise is good default.

B Additional Results

B.1 TD3

In addition to MPO and SAC, we also performed all experiments from the main text on TD3. MPO and SAC parameterize a stochastic policy, meaning they *learn* the action noise scale as a function $\sigma(s)$ of the state. TD3, on the other hand, uses a deterministic policy, and the action noise is added independently of the state. Usually, the noise scale σ is kept fixed over the course of training, and this how we handle it in our experiments as well. However, σ is an important hyperparameter, and there is no single value that works well on all environments. Thus, we repeat our experiments with all of the values $\sigma \in \{0.05, 0.1, 0.3, 0.5, 1\}$, and 10 different random seeds.

In Figure B.1, the results of the TD3 experiments with constant noise type are shown in the form of bootstrap distributions for the expected average performance, and compared to the same experiments on MPO and SAC, as well as to a Figure 3-like plot where the influence of the agent has been normalized out. As we have an additional hyperparameter (σ), we first average the TD3 performance over all σ values, before computing the average performance across tasks. The beneficial effect of

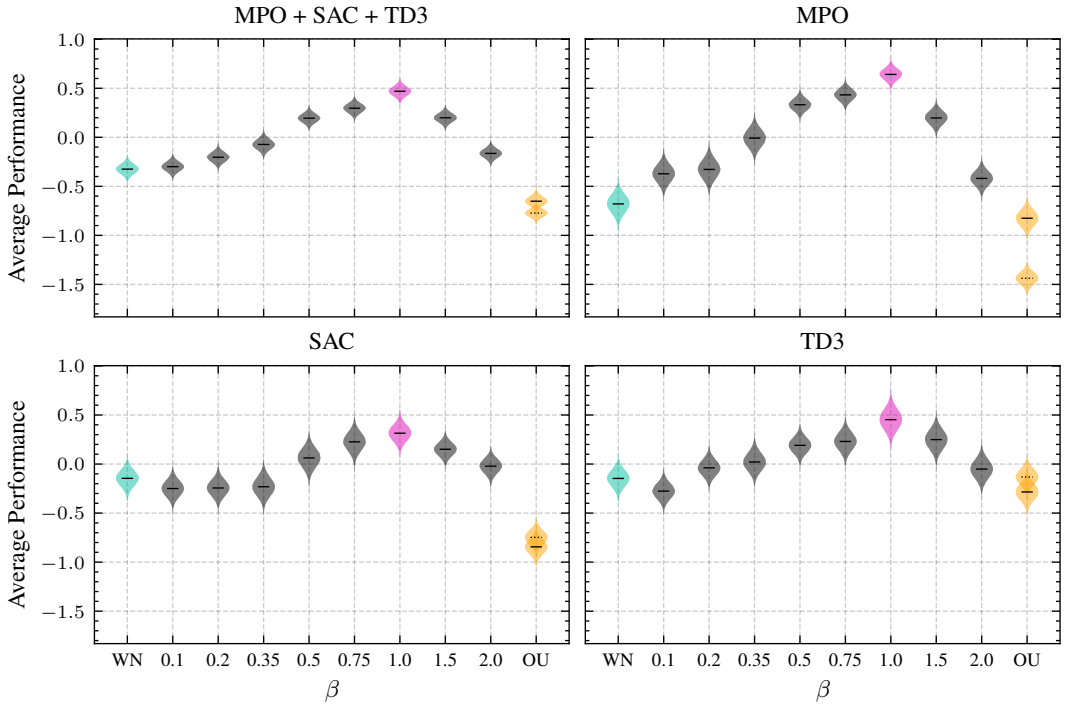

Figure B.1: All three algorithms (MPO, SAC, TD3) show a clear preference for pink action noise as measured by the average performance over the environments of Figure 2. The results of the OU experiments with the uncorrected noise scale of $\sigma = 1$ are marked with a dotted median.

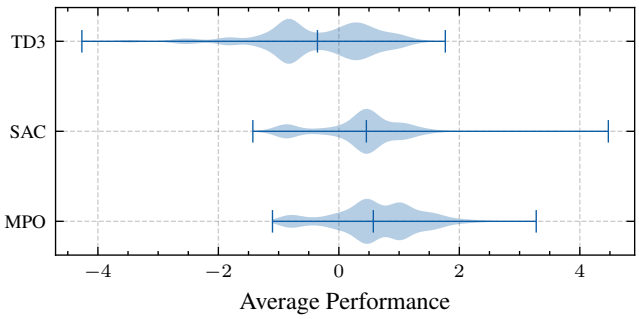

Figure B.2: Average performances across environments are combined from all β values (incl. WN). It can be seen that TD3 is consistently outperformed by both MPO and SAC. A closer look at the mean performance over all β values on each individual environment reveals that TD3 is outperformed on *all* environments by *both* MPO and SAC.

pink noise can be clearly seen on TD3 as well. In this figure we also show the results of Ornstein-Uhlenbeck noise with a noise scale of $\sigma = 1$ rather than the corrected noise scale of Eq. (4). Incidentally, these results also confirm Fujimoto et al. (2018)'s finding that, on TD3, white noise and OU noise (with $\theta = 0.15$) perform similarly.

The reason why we did not include TD3 into the analysis of the main text, is that we found TD3 to be consistently outperformed by both MPO and SAC. In Figure B.2, the average performances across environments are combined from all β values (incl. white noise), and shown for MPO, SAC and TD3. It can be seen that TD3 generally performs much worse than MPO and SAC. Looking at the mean performance over all β values on each individual environment, TD3 is outperformed on *all* environments by *both* MPO and SAC. We thus decided to exclude TD3 from our main analysis.

B.2 MPO & SAC

In the majority of this work, we measure performance in terms of the mean evaluation return over a training process. We use this method, because it implicitly measures both the final policy performance, and the sample efficiency (how quickly does the algorithm reach high performance). Most of the data we present is additionally normalized, which is necessary to aggregate performances over different environments, and thus it is often not very clear how exactly to interpret the results (other than recognizing statistical significance). In this section, we want to present some of our results in more familiar terms, namely learning curves and final policy performance.

To validate the approach of using the (mean) performance instead of the performance of the final policy, we have reproduced the results in Figure 3 using the final policy performance (mean evaluation return in the last 5% of the training process), shown in Figure B.3. In Figure B.4, we show learning curves of white noise, pink noise, and OU noise on all environments for MPO and SAC. Both visualizations confirm our takeaway that pink noise is a better default action noise than white noise or OU noise. More detailed results can be found in Section G.

The bootstrap distributions for the expected average performance (such as in Figures 3, 4, B.1, and B.3) are constructed by randomly choosing one seed for each environment, yielding one scalar

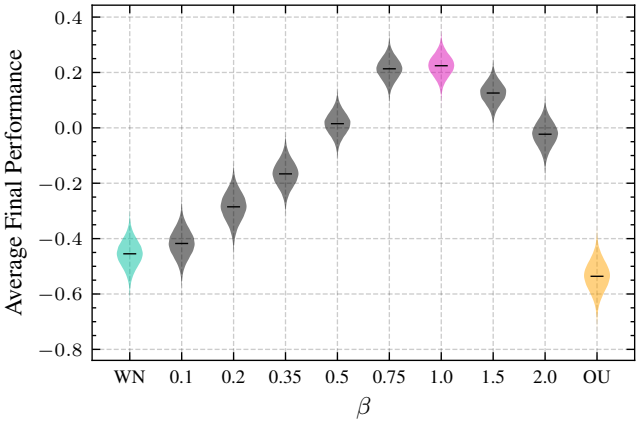

Figure B.3: The average *final* performance is like the average performance (see Section 4), but only uses the evaluation returns of the last 5% of training, thereby measuring the quality of the final learned policy. This figure shows the same analysis on MPO and SAC as Figure 3, and demonstrates that pink noise is preferable also in terms of final policy performance.

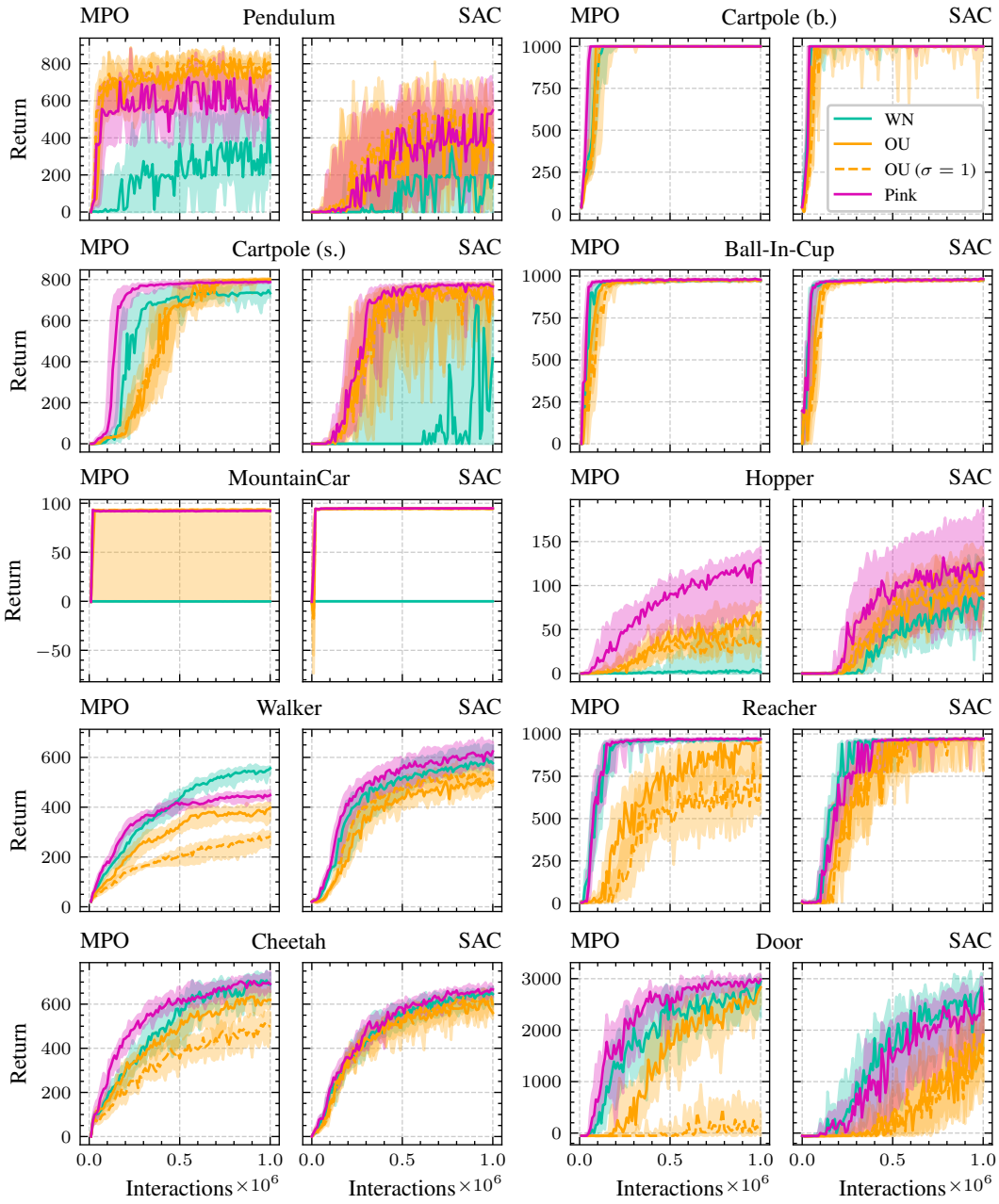

Figure B.4: Learning curves (median and interquartile range of evaluation returns) of the two baseline action noise types white noise (WN) and Ornstein-Uhlenbeck (OU) noise, as well as our suggestion of pink noise. It can be seen that pink noise, while not being better than both on all environments, is the best default choice. It is never outperformed by both white noise and OU noise, and routinely outperforms white noise (e.g. MountainCar), OU noise (e.g. Door), or both (e.g. Hopper).

(normalized) performance per environment, assuming all other variables like algorithm and noise type are fixed. Averaging these normalized performances (the reason that performances are normalized on each environment is so that this averaging is reasonable) gives an estimate for the *average performance* across environments of the given variables (e.g. noise type and algorithm). As there are S different random seeds (typically $S = 20$), we can repeat this procedure S times (with resampling) and take the mean of all S average performance estimates, giving us an estimate for the expected average performance of the given variables. Doing this N times (we use $N = 10^5$), the N estimates for the expected average performance can be collected into a bootstrap distribution, as shown in these figures.

C Environments & Algorithms

We evaluate our method on 10 different tasks (see Figure 2). Most of these are from the DeepMind Control Suite (DMC, Tassa et al., 2018), but we also use OpenAI Gym (Brockman et al., 2016) and the Adroit hand suite (Rajeswaran et al., 2018). The respective sources and exact IDs of all environments are compiled in Table C.1.

Environment	Source	ID
Pendulum	DMC	`pendulum (swingup)`
Cartpole (b.)	DMC	`cartpole (balance_sparse)`
Cartpole (s.)	DMC	`cartpole (swingup_sparse)`
Ball-In-Cup	DMC	`ball_in_cup (catch)`
MountainCar	Gym	`MountainCarContinuous-v0`
Hopper	DMC	`hopper (hop)`
Walker	DMC	`walker (run)`
Reacher	DMC	`reacher (hard)`
Cheetah	DMC	`cheetah (run)`
Door	Adroit	`door-v0`

Table C.1: Environments used in this work (see also Figure 2).

For our experiments, we relied on the TD3 and SAC implementations in Stable-Baselines3 (Raffin et al., 2021), as well as the MPO implementation in the Tonic RL library (Pardo, 2020). We only used the default hyperparameters of these algorithms, as provided by the libraries. Our own code for using colored noise with these libraries is made available online at `https://github.com/martius-lab/pink-noise-rl`.

D Bandit method details

To use a bandit algorithm to select the action noise color β for a rollout, it is necessary to define the bandit reward, which should score a rollout in terms of the β that was chosen. In our case, we use the rollout return (sum of rewards) as the score, as explained in Section 5.2. Additionally, we have to select a list of colors ("bandit arms") to search over: $B = (\beta_1, \beta_2, \ldots, \beta_K)$ (with $\beta_k \in [0, 2], \forall k$ in our case). If we assume that the bandit rewards (= rollout scores) are Gaussian distributed with a known standard deviation σ, we can use Bayesian inference to estimate the means ($\boldsymbol{\mu} \in \mathbb{R}^K$) of the reward distributions. A simple bandit algorithm we can use in this context is Thompson sampling, shown in Algorithm D.1 ($\mathbb{S}_+^K$ denotes the set of positive semi-definite $K \times K$ matrices). The relationships between the random variables are shown in the Bayesian network in Figure D.1a.

Algorithm D.1: Thompson Sampling

Input: Arms $B = (\beta_1, \ldots, \beta_K)$,
Reward distributions std σ
Initialize $\boldsymbol{m} \in \mathbb{R}^K, \Sigma \in \mathbb{S}_+^K$
for $i \in \mathbb{N}$ **do**
  Sample $\boldsymbol{q} \sim \mathcal{N}(\boldsymbol{m}, \Sigma)$
  $a_i \leftarrow \arg\max_{k \in \{1,\ldots,K\}} q_k$
  $\tau_i \leftarrow$ Run rollout with β_{a_i}
  $r_i \leftarrow$ score of rollout τ_i
  Do Bayesian update of $\boldsymbol{m}, \Sigma$ using $\{a_j, r_j\}_{j=1}^i, \sigma$
end

There is a second strong assumption in the Thompson sampling algorithm shown in Algorithm D.1 (similarly for other algorithms like UCB): it assumes that the reward distributions are stationary, i.e. that they don't change over time. This is not the case in the context of reinforcement learning: if the rollout score r_i is defined as the return, then, if the reinforcement learning algorithm works, it should naturally be the case that the policy improves over time, and thus, on average, $r_i > r_j$ for $i \gg j$. This setting of non-stationary bandit distributions can be addressed by using a sliding-window approach (e.g. Garivier & Moulines, 2008): instead of updating the belief parameters $\boldsymbol{m}, \Sigma$ with respect to the whole history of observations, only keep a window of the last N rollouts.

There remains one other problem: how do we choose the prior parameters $\boldsymbol{m}$ and Σ and the variance σ^2 of the reward distributions? For Σ, the easiest solution is to assume independent arms, i.e. make

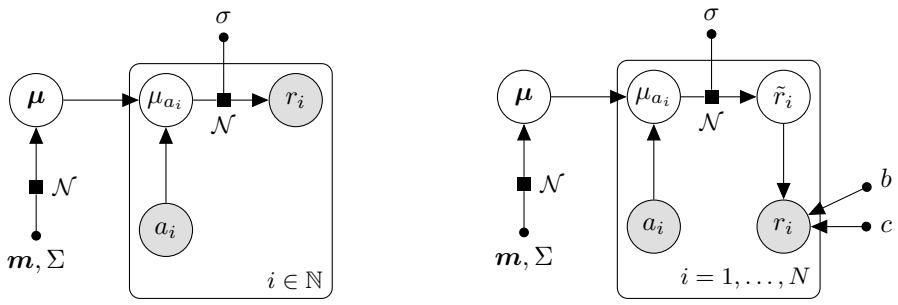

Figure D.1: (a) A Bayesian bandit with Gaussian reward distributions. The rewards from arm k are sampled from $\mathcal{N}(\mu_k, \sigma)$. Thompson sampling (Alg. D.1) can infer $\boldsymbol{\mu}$ while trading off exploration and exploitation. (b) By introducing the constants b and c, the algorithm can be made scale invariant by performing Thompson sampling with respect to the normalized reward $\tilde{r}_i = (r_i - b)/c$.

Σ diagonal. This is not necessarily the most efficient solution, as one can imagine that two similar β values will also perform similarly in their rollouts.[2] For $\boldsymbol{m}$, the non-stationarity becomes a problem: again assuming we use the rollout return as a score, these scores will probably be much lower at the beginning of training than at the end. Additionally, we might not even know the scale of returns in a task. To account for this, it would be necessary to make the prior variances Σ_{kk} very large/uninformed. Similarly, σ needs to be large, to account for the unknown scale of the bandit reward spread. However, this would mean that many more samples (rollouts) are necessary to tighten the belief distributions. This is a problem, especially because we only have a small set of N rollouts when using the sliding-window method.

The ideal would be a bandit method which is invariant with respect to affine transformations of the rewards, in the sense that it would make no difference if all rewards r were transformed to be $br + c$ for some constants $b > 0$ and $c \in \mathbb{R}$ for all arms. In Figure D.1b, this situation is shown in a Bayesian network. Here, the generative process is almost the same as before (see Figure D.1a), except that the reward $\tilde{r}_i$ is scaled and translated by $r_i = b\tilde{r}_i + c$ before observation. If, as shown, the constants b and c are independent of the chosen arm and stay constant within the window, it is possible to optimize them via maximum marginal likelihood, given the window of past observations of r_i.

The bandit inference task is to infer the distributional means $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_k)$ from the actions (color indices) $\boldsymbol{a} = (a_i)_{i=1}^N$ and rewards (rollout scores) $\boldsymbol{r} = (r_i)_{i=1}^N$. We set the prior means of the belief distributions to 0 ($\boldsymbol{m} = \mathbf{0}$), because we want the normalized reward distributions to be centered around 0. For now, we don't fix Σ, but let it be any positive semi-definite $K \times K$ matrix. The generative model for $\boldsymbol{r}$ is defined via the following prior and likelihood function:

$$p(\boldsymbol{\mu} \mid \Sigma) = \mathcal{N}(\boldsymbol{\mu} \mid \mathbf{0}, \Sigma) \tag{6}$$

$$p(\boldsymbol{r} \mid \boldsymbol{\mu}, \boldsymbol{a}, b, c, \sigma) = \prod_i \mathcal{N}(r_i \mid b\mu_{a_i} + c, (b\sigma)^2) \tag{7}$$

These lead us to the following evidence/marginal likelihood function:

$$p(\boldsymbol{r} \mid \boldsymbol{a}, b, c, \sigma, \Sigma) = \prod_i p(r_i \mid a_i, b, c, \sigma, \Sigma) \tag{8}$$

$$= \prod_i \int p(r_i \mid \boldsymbol{\mu}, a_i, b, c, \sigma)\, p(\boldsymbol{\mu} \mid \Sigma)\, \mathrm{d}\boldsymbol{\mu} \tag{9}$$

$$= \prod_i \int \mathcal{N}(r_i \mid b\mathbf{e}_{a_i}^\top \boldsymbol{\mu} + c, (b\sigma)^2)\, \mathcal{N}(\boldsymbol{\mu} \mid \mathbf{0}, \Sigma)\, \mathrm{d}\boldsymbol{\mu} \tag{10}$$

$$= \prod_i \mathcal{N}(r_i \mid b\mathbf{e}_{a_i}^\top \mathbf{0} + c, (b\sigma)^2 + b\mathbf{e}_{a_i}^\top \Sigma b \mathbf{e}_{a_i}) \tag{11}$$

[2] We also tried a different approach by using a modified RBF kernel matrix to account for covariance between the arms, but the results were essentially the same as with independent arms.

$$= \prod_i \mathcal{N}(r_i \mid c, b^2(\sigma^2 + \Sigma_{a_i a_i})), \tag{12}$$

where we used canonical basis vectors to represent $\mu_{a_i} = \mathbf{e}_{a_i}^\top \boldsymbol{\mu}$. For maximization, it is convenient to work with the log-evidence:

$$\log p(\boldsymbol{r} \mid \boldsymbol{a}, b, c, \sigma, \Sigma) = \log \prod_i \mathcal{N}(r_i \mid c, b^2(\sigma^2 + \Sigma_{a_i a_i})) \tag{13}$$

$$= \sum_i -\frac{1}{2} \log\big(2\pi b^2(\sigma^2 + \Sigma_{a_i a_i})\big) - \frac{(c - r_i)^2}{2b^2(\sigma^2 + \Sigma_{a_i a_i})} \tag{14}$$

$$=: L(b, c) \tag{15}$$

We can now maximize the evidence by setting the partial derivatives to 0:

$$\partial_c L(b, c) \propto \sum_i (c - r_i) = 0 \tag{16}$$

$$\partial_b L(b, c) = \sum_i \frac{-1}{b} + \frac{(c - r_i)^2}{b^3(\sigma^2 + \Sigma_{a_i a_i})} = 0 \tag{17}$$

Solving these equations gives us

$$c = \frac{1}{N} \sum_i r_i \tag{18}$$

$$b^2 = \frac{1}{N} \sum_i \frac{(c - r_i)^2}{\sigma^2 + \Sigma_{a_i a_i}}. \tag{19}$$

Using these values, we can "reconstruct" the unscaled/normalized reward

$$\tilde{r}_i = \frac{r_i - c}{b} \tag{20}$$

and perform Thompson sampling with respect to $\tilde{r}_i$. This *normalized Thompson sampling* algorithm, including the sliding window modification, is presented in Algorithm D.2.

Algorithm D.2: Normalized TS

Input: Arms $B = (\beta_1, \ldots, \beta_K)$,
Window size N

Initialize
$\boldsymbol{m} \leftarrow \mathbf{0} \in \mathbb{R}^K, \Sigma \in \mathbb{S}_+^K, \sigma \leftarrow 1$

for $l \in \mathbb{N}$ **do**
- $i \leftarrow l \mod N$
- $M \leftarrow \min\{l, N\}$
- Sample $\boldsymbol{q} \sim \mathcal{N}(\boldsymbol{m}, \Sigma)$
- $a_i \leftarrow \arg\max_{k \in \{1,\ldots,K\}} q_k$
- $\tau_i \leftarrow$ Run rollout with β_{a_i}
- $r_i \leftarrow$ score of rollout τ_i
- $c \leftarrow \frac{1}{M} \sum_{j=1}^M r_j$
- $b \leftarrow \sqrt{\frac{1}{M} \sum_{j=1}^M \frac{(c - r_j)^2}{\sigma^2 + \Sigma_{a_j a_j}}}$
- $\tilde{r}_i \leftarrow \frac{r_i - c}{b}$
- Do Bayesian update of $\boldsymbol{m}, \Sigma$ using $\{a_j, \tilde{r}_j\}_{j=1}^M, \sigma$

end

Next, we want to show that this method is indeed invariant to affine transformations of the bandit reward.

Proposition 1. *The posterior distribution over $\boldsymbol{\mu}$ in the normalized bandit algorithm (Alg. D.2) is identical for the observations $\boldsymbol{r} = (r_1, \ldots, r_N)$ and $\boldsymbol{r}' = b'\boldsymbol{r} + c'$, for all $b' > 0$ and $c' \in \mathbb{R}$. In other words, the algorithm is invariant to a scaling and translation of the rewards.*

Proof. In this setting, the observed rewards r_i are normalized to

$$\tilde{r}_i = \frac{r_i - c(\boldsymbol{r})}{b(\boldsymbol{r})} \tag{21}$$

with

$$c(\boldsymbol{r}) = \frac{1}{N} \sum_{i=1}^N r_i \tag{22}$$

$$b(\boldsymbol{r}) = \sqrt{\frac{1}{N} \sum_{i=1}^N \frac{(c(\boldsymbol{r}) - r_i)^2}{\sigma^2 + \Sigma_{a_i a_i}}}. \tag{23}$$

To prove the invariance of the algorithm, we will simply show that this normalized reward is the same for both sets of observations, i.e. that $\tilde{\boldsymbol{r}} = \tilde{\boldsymbol{r}}'$. Then, clearly, the posteriors $p(\boldsymbol{\mu} \mid \tilde{\boldsymbol{r}})$ and $p(\boldsymbol{\mu} \mid \tilde{\boldsymbol{r}}')$ will also be the same. Expanding $\tilde{\boldsymbol{r}}'$, we get:

$$\tilde{\boldsymbol{r}}' = \frac{\boldsymbol{r}' - c(\boldsymbol{r}')}{b(\boldsymbol{r}')} \tag{24}$$

$$= \frac{b'\boldsymbol{r} + c' - c(b'\boldsymbol{r} + c')}{b(b'\boldsymbol{r} + c')} \tag{25}$$

$$= \frac{b'\boldsymbol{r} + c' - \frac{1}{N}\sum_{i=1}^{N}(b'r_i + c')}{\sqrt{\frac{1}{N}\sum_{i=1}^{N}\frac{(\frac{1}{N}\sum_{j=1}^{N}(b'r_j + c') - (b'r_i + c'))^2}{\sigma^2 + \Sigma_{a_i a_i}}}} \tag{26}$$

$$= \frac{b'\boldsymbol{r} + c' - b'\frac{1}{N}\sum_{i=1}^{N} r_i - c'}{\sqrt{\frac{1}{N}\sum_{i=1}^{N}\frac{(b'\frac{1}{N}\sum_{j=1}^{N} r_j + c' - b'r_i - c')^2}{\sigma^2 + \Sigma_{a_i a_i}}}} \tag{27}$$

$$= \frac{b'(\boldsymbol{r} - c(\boldsymbol{r}))}{\sqrt{\frac{1}{N}\sum_{i=1}^{N}\frac{b'^2(c(\boldsymbol{r}) - r_i)^2}{\sigma^2 + \Sigma_{a_i a_i}}}} \tag{28}$$

$$= \frac{\boldsymbol{r} - c(\boldsymbol{r})}{b(\boldsymbol{r})} \tag{29}$$

$$= \tilde{\boldsymbol{r}} \tag{30}$$

Thus, we can conclude that the reward normalization indeed guarantees invariance to affine reward transformations in algorithms such as Thompson sampling. □

With this reward normalization, the prior parameters m (of $\boldsymbol{m} = m\mathbf{1}$) and s (of $\Sigma = s^2 I$) become redundant. We have already set $\boldsymbol{m} = \mathbf{0}$, and we now also set the prior variances Σ_{kk} to 1. This encourages the algorithm to keep the normalized mean estimates μ_k approximately $\mathcal{N}(0, 1)$-distributed. The "likelihood" parameter σ remains to be tuned, but it is now not necessary to account for the large uncertainty in the reward scale, as σ is only concerned with the normalized reward. In our experiments we always set $\sigma = 1$.

D.1 Bandit vs. Random

Although we found the normalized bandit algorithm (Alg. D.2) to work well on simple non-stationary tasks, in the RL setting (for choosing β) the performance was just as that of a random β selection for every rollout. In Table D.1, we list the results of a Welch t-test, testing for inequality of the performance distributions achieved by the bandit algorithm and random β selection on every environment. It can be seen that the two methods are statistically indistinguishable. This shows that the bandit method does not work as intended, as "random arm selection" should be an easy baseline to outperform. The reason for this is probably due to the rollout return not being informative enough as a bandit reward signal.

Environment	Bandit $\neq$ Random	p
Pendulum	✗	0.98
Cartpole (b.)	✗	0.09
Cartpole (s.)	✗	0.67
Ball-In-Cup	✗	0.87
MountainCar	✗	0.54
Hopper	✗	0.09
Walker	✗	0.15
Reacher	✗	0.70
Cheetah	✗	0.20
Door	✗	0.59

Table D.1: Bandit vs. Random (Welch t-test)

E SOLVING MOUNTAINCAR BY FFT

MountainCar is a very simple environment. Although its dynamics are almost those of a harmonic oscillator, there is a difference to the oscillator environment from Section 6: MountainCar's oscillation dynamics are non-linear. At the bottom of MountainCar's valley (see Figure 2), the small-angle approximation of a non-linear oscillator may be used, but for the motion to go up to the top, the behavior is different from simple harmonic motion. Nevertheless, we can use this insight to develop a very simple open-loop control algorithm to solve this environment, by running one rollout without applying any action (just letting the mountain make the car go back and forth a bit), then analyzing the resulting trajectory and inferring the hill's (small-angle) resonant frequency (via the Fast Fourier Transform algorithm). Finally, we can control the car by simply swinging it back and forth at the resonant frequency. This algorithm, which works very well on this task, is shown below.

```python
import gym
import numpy as np
from scipy.fft import rfft

# Initialize environment
env = gym.make('MountainCarContinuous-v0')
T = env._max_episode_steps

# Run a single rollout with no force. Save x-coordinate to `x`.
obs = env.reset()
x = [obs[0]]
for t in range(T):
    obs, *_ = env.step([0])
    x.append(obs[0])

# Find resonant frequency = highest peak of FFT (excluding DC)
f = (np.argmax(abs(rfft(x))[1:]) + 1) / (T + 1)

# Action plan (harmonic excitation)
a = np.sin(2*np.pi*f * np.arange(T))

# Test on 1000 rollouts
N = 1000
solved = 0
for i in range(N):
    env.reset()
    for t in range(T):
        _, r, _, _ = env.step([a[t]])
        if r > 0:
            solved += 1
            break

print(f"Solved: {solved/N * 100:.0f}%.")  # prints "Solved: 100%."
```

F OSCILLATOR ENVIRONMENT

The oscillator environment, which we make available online as a gym environment (<anonymous>), models the 1-dimensional motion of a particle of mass m, attached to the origin by an ideal spring of stiffness k, damped with friction coefficient b, and driven by a force (the action) F. This motion is described by the ordinary differential equation

$$m\ddot{x} = F - b\dot{x} - kx, \tag{31}$$

where x is the particle's position. In our experiments we set the friction coefficient b to zero, i.e. the system is undamped. This setup is then called a *simple harmonic oscillator*. The energy of the oscillator is the sum of kinetic and potential energy:

$$E = \frac{1}{2}m\dot{x}^2 + \frac{1}{2}kx^2. \tag{32}$$

The resonant frequency is:

$$f = \frac{1}{2\pi}\sqrt{\frac{k}{m}}. \tag{33}$$

As we want to configure the oscillator to have a given resonant frequency f, we need to find m and k accordingly. To get a unique solution, we impose a second constraint: the energy at $x = 1$ and $\dot{x} = 0$ should be $E = 2\pi^2$. If we now solve the two equations (32) and (33) for m and k, imposing the constraint on E, we get the solution

$$k = 4\pi^2 \tag{34}$$

$$m = \frac{1}{f^2} \tag{35}$$

to set the resonant frequency. In Figure F.1, a few pure-noise trajectories (akin to Figure 1) are shown on the oscillator environment.

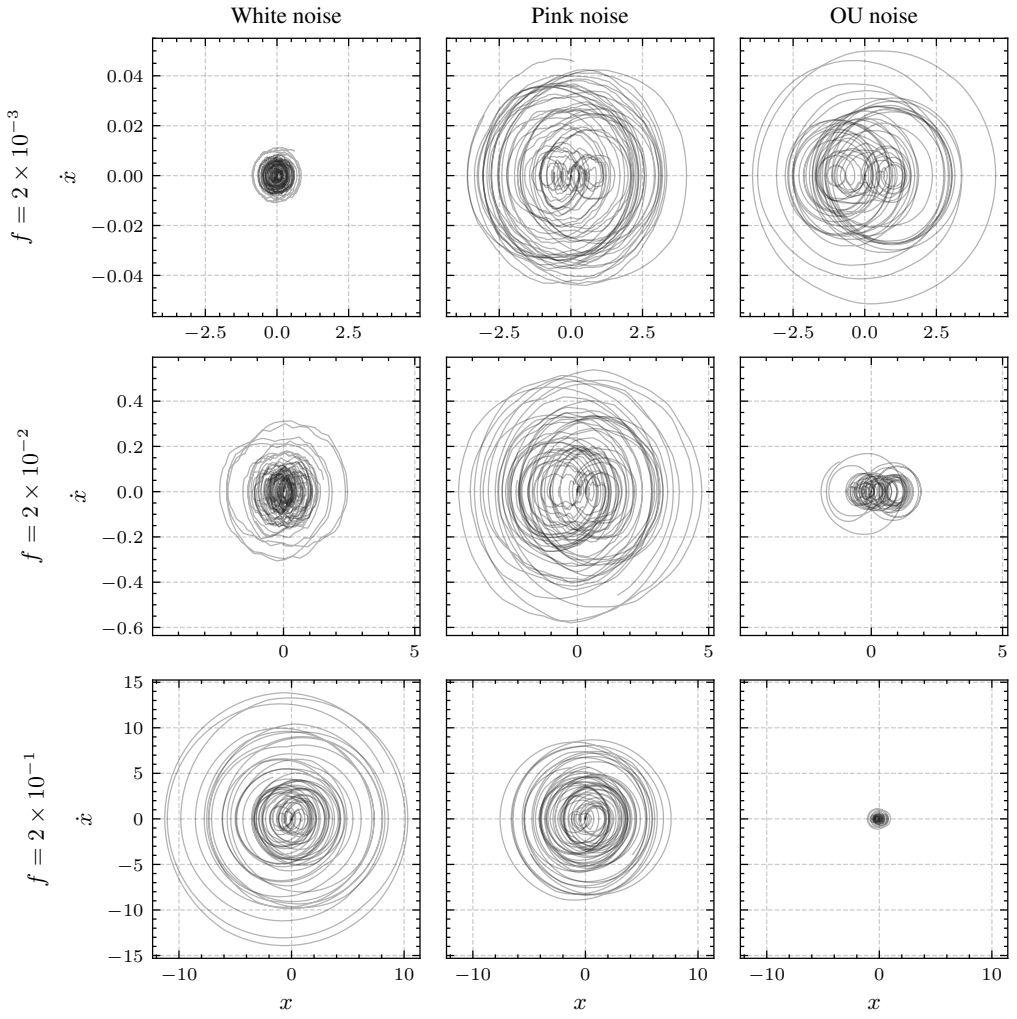

Figure F.1: Trajectories on the oscillator environment. For each of the 3 resonance frequencies $f \in \{0.002, 0.02, 0.2\}$, we sample 5 action noise signals of length $\frac{10}{f}$ of white noise, pink noise and OU noise. We can see what was already shown in Figure 5: pink noise is much less sensitive to the parameterization than white noise and OU noise, and always manages to excite the oscillator up to a certain amplitude. White noise and OU noise only work well in the high- and low-frequency regime, respectively.

G Detailed Results

Environment	Agent	Performance	WN	OU	Pink	Oracle	Anti-Oracle	Gain
Pendulum	MPO	Mean	247	651	558	670	239	430
		Final Policy	311	702	574			
	SAC	Mean	158	283	294	361	158	202
		Final Policy	224	350	446			
Cartpole (b.)	MPO	Mean	928	940	967	967	928	39
		Final Policy	999	1000	1000			
	SAC	Mean	939	890	941	950	890	59
		Final Policy	960	908	958			
Cartpole (s.)	MPO	Mean	535	499	666	666	489	177
		Final Policy	703	784	788			
	SAC	Mean	226	459	532	533	159	374
		Final Policy	377	608	730			
Ball-In-Cup	MPO	Mean	926	909	948	948	909	39
		Final Policy	974	973	978			
	SAC	Mean	930	901	933	941	901	39
		Final Policy	976	975	979			
MountainCar	MPO	Mean	13	52	91	92	13	78
		Final Policy	13	56	92			
	SAC	Mean	0	89	93	93	0	93
		Final Policy	0	90	94			
Hopper	MPO	Mean	14	34	69	69	14	54
		Final Policy	25	62	108			
	SAC	Mean	43	53	77	80	43	36
		Final Policy	89	94	119			
Walker	MPO	Mean	384	284	363	390	284	106
		Final Policy	530	377	448			
	SAC	Mean	437	363	471	472	363	108
		Final Policy	593	506	602			
Reacher	MPO	Mean	864	600	871	888	581	306
		Final Policy	956	856	966			
	SAC	Mean	776	653	745	776	653	122
		Final Policy	955	914	940			
Cheetah	MPO	Mean	481	440	543	543	440	103
		Final Policy	666	612	678			
	SAC	Mean	469	439	483	502	439	63
		Final Policy	631	577	640			
Door	MPO	Mean	1830	1376	2207	2207	1376	830
		Final Policy	2586	2492	2909			
	SAC	Mean	1332	546	1183	1332	546	785
		Final Policy	2192	1535	2195			

Table G.1: Comparison of final policy performance (see Section B.2) and mean performance over the training process (Section 4) on all environments. Results are averaged across seeds, and shown for white noise (WN), Ornstein-Uhlenbeck noise (OU), and pink noise (Pink) as action noise on MPO and SAC. Additionally, the Oracle and Anti-Oracle performances are shown. The gain between these (rightmost column) represents the difference achievable by changing the noise type, and is the basis for the "performance gain" measure used in Section 4.2.

REFERENCES

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016. URL http://arxiv.org/abs/1606.01540. 8

James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19:297–301, 1965. 1

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1582–1591. PMLR, 2018. URL http://proceedings.mlr.press/v80/fujimoto18a.html. 6

Aurélien Garivier and Eric Moulines. On upper-confidence bound policies for non-stationary bandit problems. arXiv: 0805.3415, 2008. URL https://arxiv.org/abs/0805.3415. 8

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In Yoshua Bengio and Yann LeCun (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1509.02971. 3

Fabio Pardo. Tonic: A deep reinforcement learning library for fast prototyping and benchmarking. *CoRR*, abs/2011.07537, 2020. URL https://arxiv.org/abs/2011.07537. 3, 8

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *J. Mach. Learn. Res.*, 22:268:1–268:8, 2021. URL http://jmlr.org/papers/v22/20-1364.html. 3, 8

Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In Hadas Kress-Gazit, Siddhartha S. Srinivasa, Tom Howard, and Nikolay Atanasov (eds.), *Robotics: Science and Systems XIV, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, June 26-30, 2018*, 2018. doi: 10.15607/RSS.2018.XIV.049. URL http://www.roboticsproceedings.org/rss14/p49.html. 8

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy P. Lillicrap, and Martin A. Riedmiller. Deepmind control suite. *CoRR*, abs/1801.00690, 2018. URL http://arxiv.org/abs/1801.00690. 8

Jens Timmer and Michel Koenig. On generating power law noise. *Astronomy and Astrophysics*, 300: 707, 1995. 1