Sample-efficient Cross-Entropy Method for Real-time Planning

Cristina Pinneri[1,3] **Shambhuraj Sawant**[1] **Sebastian Blaes**[1] **Jan Achterhold**[2]
Jörg Stückler[2] **Michal Rolínek**[1] **Georg Martius**[1]

[1]Autonomous Learning Group, Max Planck Institute for Intelligent Systems, Germany, Tübingen
[2]Embodied Vision Group, Max Planck Institute for Intelligent Systems, Germany, Tübingen,
`name.surname@tuebingen.mpg.de`
[3]Max Planck ETH Center for Learning Systems

Abstract: Trajectory optimizers for model-based reinforcement learning, such as the Cross-Entropy Method (CEM), can yield compelling results even in high-dimensional control tasks and sparse-reward environments. However, their sampling inefficiency prevents them from being used for real-time planning and control. We propose an improved version of the CEM algorithm for fast planning, with novel additions including temporally-correlated actions and memory, requiring 2.7-22× less samples and yielding a performance increase of 1.2-10× in high-dimensional control problems.

Keywords: cross-entropy-method, model-predictive-control, planning, trajectory-optimization, model-based reinforcement learning

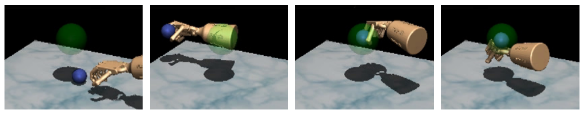
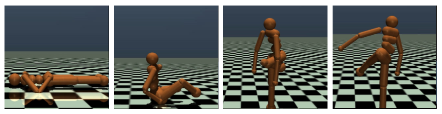

Figure 1: Found behaviors for RELOCATE environment (left) and HUMANOID STANDUP (right)

1 Introduction

Recent work in model-based reinforcement learning (MBRL) for high-dimensional systems employs population-based algorithms as trajectory optimizers [1, 2, 3, 4, 5]. Sampling-based methods have also been used in the control community in scenarios when the cost function is not differentiable [6]. The particular appeal of these methods lies in a few but important factors: the possibility of optimizing black-box functions; lower sensitivity to hyperparameter tuning and thus higher robustness; no requirement of gradient information; lower susceptibility to local optima. The Cross-Entropy Method (CEM) [7] was introduced for the first time in the 1990s as a stochastic, derivative-free, global optimization technique, but it is just in recent years that it gained traction in the model-based RL community. CEM for trajectory optimization is indeed a promising metaheuristics which has been shown to work well even with learned models, producing comparable or higher performance than model-free reinforcement learning methods, as shown in [1, 2, 3].

There is a problem, however, intrinsic to the nature of population-based optimizers, which makes these methods so far unsuitable for **real-time planning and control**, even in conjunction with a learned model: the high computational price. Heuristics like CEM require a large number of samples to minimize the objective function. This creates severe limitations for its deployment in real-time control for robotics, requiring a dramatic speed-up.

Our approach originates exactly from this question: is it possible to do real-time planning with a zeroth-order optimizer like CEM? Our method proposes an enhancement of the original CEM for the purpose of trajectory optimization in model-predictive control and comprises various ways to address the inefficiency of sampling in high-dimensional systems, including equipping CEM with memory and generating time-correlated action sequences. Our upgrades are unified under the name **iCEM**.

Contributions We present iCEM, a faster, more sample-efficient and higher performing version of the CEM algorithm that could potentially bridge the gap between MBRL in simulation and real-time robotics. We present a detailed examination of the key improvements over CEM with an extensive ablation study. Finally, we test the results on several hard continuous-control robotic tasks in the MuJoCo simulator [8] such as HUMANOID STANDUP, and manipulation environments with sparse rewards like FETCH PICK&PLACE or other manipulation environments with many degrees of freedom like DOOR and RELOCATE. In the latter, we solve the task with 90% success rate while using 13.7$\times$ less samples and get an average performance improvement of 400% over the state-of-the-art CEM.

In order to study the algorithmic improvements without being biased by model errors, we perform all our ablations with the ground truth dynamics. In addition to this, we report the performance when used in combination with learned models from a reimplementation of the PlaNet framework [3] (without requiring additional fine-tuning), showing a speed-up that potentially allows online planning with iCEM, without a substantial loss on the overall performance.

To the best of our knowledge, this is the first work that aims at making CEM itself fast enough to be used for real-time robot planning and control. It can be integrated into any existing method that uses the standard CEM or other zeroth-order optimizers. The source code can be found at `https://github.com/martius-lab/iCEM.git`.

Related Work Many works on MBRL and motion planning show that it is possible to control systems without making use of gradient descent. Indeed, Evolution Strategies (ES) [9] regained popularity for their successful use in RL [10, 11, 12], making population-based methods an attractive alternative to policy gradient or as a supportive guidance [13, 14]. Sampling-based techniques have also been used for model-predictive control (MPC), like model predictive path integral (MPPI) control [4], with applications to aggressive driving by using a GPU [15].

In particular, the Cross-Entropy Method (CEM) [7, 16, 17], thoroughly analyzed in [18], has been used both for direct policy optimization [19] and planning with learned models [1, 2], to improve the performance of rapidly exploring random trees [20], with successful applications in many fields of science. Examples include visual tracking [21], bioinformatics [22], and network reliability [23].

In Duan et al. [24] it is reported that CEM has better performance also over the more sophisticated Covariance Matrix Adaptation ES (CMA-ES) [25], the latter being computationally more expensive since it computes the full covariance matrix, while actions in CEM are sampled independently along the planning horizon, requiring only a diagonal covariance matrix.

The core focus of our work is to make CEM functional for real-time decision making. Recent works in this direction propose a differentiable version of CEM [26] or to jointly use model gradients together with the CEM search [14]. Wang and Ba [2] use CEM on the policy parameters rather than in the action space. Nevertheless, the whole procedure still depends on the speed of the CEM optimization, making it the bottleneck for fast planning.

2 The Cross-Entropy Method

The cross-entropy method (CEM) is a derivative-free optimization technique that was originally introduced in Rubinstein and Davidson [7] as an adaptive importance sampling procedure for the estimation of rare-event probabilities that makes use of the *cross-entropy* measure.

CEM can be seen as an Evolution Strategy which minimizes a cost function $f(x)$ with $f : \mathbb{R}^n \to \mathbb{R}$ by finding a suitable "individual" x. The individuals are sampled from a population/distribution and evaluated according to $f(x)$. Then, they are sorted based on this cost function and a fixed number of "elite" candidates is selected.

This *elite-set* is going to determine the parameters of the population for the next iteration. In the standard case, the population is modeled with a Gaussian distribution with mean μ and diagonal covariance matrix diag(σ^2), where $\mu, \sigma \in \mathbb{R}^n$. By fitting μ and σ to the elite-set, the sampling distribution concentrates around the x with low cost. After several iterations of this selection procedure, an x close to a local optimum, or even the global optimum, is found. Due to this iterative procedure, the total number of evaluated samples becomes extensive which can lead to a slow run time depending on the computational cost of f.

2.1 Standard modifications of CEM for model-predictive control: CEM$_{\text{MPC}}$

In the MPC setting, CEM is used every timestep to optimize an h-step planning problem on the action sequences, see Alg. S1. For a d dimensionality of the action space we have now: $\mu_t, \sigma_t \in \mathbb{R}^{d\times h}$. For the terminology, we call *CEM-iteration* one step of the inner loop of CEM that optimizes the sampling distribution (line 8–12 in Alg. S1). The outer loop *step* marks the progression in the environment by executing one action. Naturally, the next step considers the planning problem one timestep later. As a typical modification [1, 2], the initial mean μ_t of the CEM distribution is shift-initialized from the optimized μ_{t-1}, see Suppl. E.1.

Another standard modification is to use a momentum term [27] in the refitting of the distributions between the CEM-iterations (line 12 in Alg. S1). The reason is that only a small elite-set is used to estimate many parameters of the sampling distribution. A simple choice is $\mu_t^{i+1} = \alpha\mu_t^i + (1 - \alpha)\mu^{elite-set_i}$ where $\alpha \in [0, 1]$ and i is the index of *CEM-iterations*. Actions are always limited such that the standard method uses truncated normal distributions with suitably adapted bounds instead of unbounded Gaussian distributions. We call this variance **CEM$_{\text{MPC}}$**.

In Chua et al. [1] (PETS method), in addition to the standard improvements above, the sampling distribution was modified (only documented in the source code). Instead of setting the truncation bounds to match the action-range, the truncation is always set to 2σ, and σ is adapted to be not larger than $\frac{1}{2}b$ where b is the minimum distance to the action bounds. We refer to this method as **CEM$_{\text{PETS}}$**.

3 Improved CEM – iCEM

In this section we thoroughly discuss several improvements to CEM for the purpose of model-predictive control (MPC) and trajectory optimization, with the goal to achieve strong performance already with a low number of samples. This section is complemented by the ablations in Sec. 4.3 and the sensitivity analysis in Sec. C.2.

3.1 Colored noise and correlations

The CEM action samples should ideally produce trajectories which maximally explore the state space, especially if the rewards are sparse. Let us consider a simple stochastic differential equation in which the trajectory x is a direct integration of the stochastic actions a:

$$\frac{\mathrm{d}}{\mathrm{d}t}x(t) = a(t) \tag{1}$$

In the case of Gaussian inputs, $x(t)$ is a Brownian random walk, which is commonly used to describe the trajectory of particles under random perturbations. It comes as no surprise that coherent trajectories (temporally correlated) cannot be generated by uncorrelated inputs, like the ones sampled in CEM.

It was witnessed many times in nature that animals revert to different strategies, rather than plain Brownian exploration, when they need to efficiently explore the space in search for food. In fact, when prey is scarce, animals like sharks or other predatory species produce trajectories which can be described by the so-called Lévy walks [28]. Classically, Lévy walks exhibit velocities with long-term correlations (being sampled from a power-law distribution), and consequentially produce trajectories with higher variance than a Brownian motion [29].

If we look at the action sequence as a time series, its correlation structure is directly connected to the power spectral density (PSD) as detailed in Sec. F in the Supplementary. The PSD is the squared norm of the value of Fourier transform and intuitively quantifies how much each frequency is occurring in the time series. The CEM actions, being sampled independently along the planning horizon, have a constant power spectral density (PSD), more commonly referred to as *white-noise*. How does the PSD of a time series with non-zero correlations look like? For this purpose, we introduce generalized colored-noise for the actions a as the following PSD:

$$\text{PSD}_a(f) \propto \frac{1}{f^\beta} \tag{2}$$

where f is the frequency and β is the colored-noise scaling exponent. $\beta = 0$ corresponds to white noise, a value of $\beta > 0$ means that high frequencies are less prominent than low ones. In signal

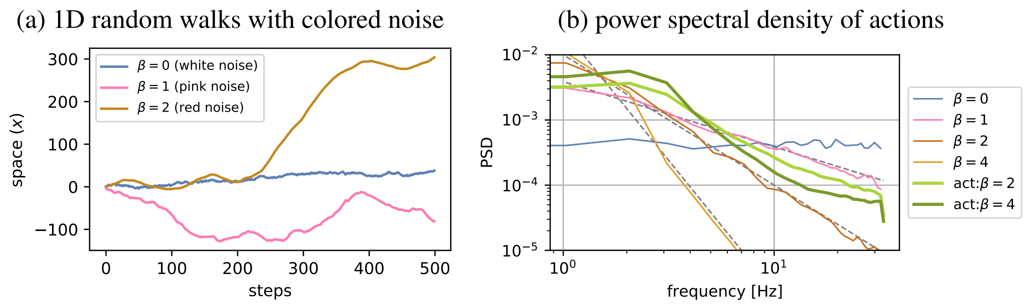

Figure 2: Colored random noise. (a) random walks with colored noise of different temporal structure. (b) power spectrum of colored-random action sequences for different β and of the chosen (and successful) action-sequences of iCEM generated by differently colored search noise (act:$\beta = 2$ and 4) for the HUMANOID STANDUP task. Successful action sequences are far from white-noise ($\beta = 0$).

processing they are called *colored noise* with pink noise for $\beta = 1$, and Brownian or red noise for $\beta = 2$, but any other exponent is possible.

How does the trajectory $x(t)$ of Eq. 1 look when we use colored-noise actions? Figure 2(a) shows three examples with the same action variance – the larger the β, the larger the coherence and the larger distances can be reached. This can be formalized by computing the PSD$_x$ of the state-space trajectory (x). Using Eq. 1 and Eq. 2 we find:

$$\mathrm{PSD}_x(f) = \|\mathcal{F}[x(t)](f)\|^2 \overset{(*)}{=} \frac{\|\mathcal{F}[a(t)](f)\|^2}{4\pi^2 f^2} = \frac{\mathrm{PSD}_a(f)}{4\pi^2 f^2} \propto \frac{1}{f^{\beta+2}}. \tag{3}$$

where the equality $(*)$ results from the integration property of Fourier transforms, which is $\mathcal{F}[\frac{\mathrm{d}}{\mathrm{d}t}x(t)] = i2\pi f \mathcal{F}[x(t)]$. As a result, the PSD of $x(t)$ is directly controlled by the choice of β – higher β results in stronger low frequency components, as evident in Fig. 2(a).

Let us consider now the effect of colored-noise in a robotic setting: the HUMANOID STANDUP task, see Fig. 1, included in the OpenAI Gym [30] environments. Figure 2(b) displays the PSD$_a$ of different action-noise processes together with the PSD of successful action-sequences. Notice the log-log scale – a straight line corresponds to a power-law decay as in Eq. 2. When using such a colored-noise to sample action-sequences inside CEM (more details below), we obtain a dramatically improved speed and performance. Considering the spectrum of the successful action sequences found by our proposed iCEM method (green lines in Fig. 2(b)) we see a clear preference of low frequencies as well as a sharp drop for the highest frequency (corresponding to alternating actions at every step). Regardless of whether we use $\beta = 2$ or $\beta = 4$, the action sequence follows roughly $\beta = 1.5$ with an additional bump at 2-3 Hz. More information on the choice of β can be found in Suppl. C.1.

We introduce the colored-noise in CEM as a function of β which creates correlated action sequences with a PSD as in (2). For sampling, we use the efficient implementation of [31] based on Fast Fourier Transform [32]. It relies on the fact that PSD of a time-series can be directly modified in the frequency space. Indeed, if we want to sample actions with a PSD as in (2), we have to apply the following transformation to the original white noise actions $a(t)$:

$$\bar{a}(t) = \mathcal{F}^{-1}\left[\frac{1}{f^{\beta/2}}\mathcal{F}[a(t)]\right] \quad \text{gives} \quad \mathrm{PSD}_{\bar{a}}(f) = \left\|\frac{1}{f^{\beta/2}}\mathcal{F}[a(t)](f)\right\|^2 = \frac{1}{f^{\beta}}\mathrm{PSD}_a(f) \propto \frac{1}{f^{\beta}} \tag{4}$$

The resulting sampling function, which we will call $\mathcal{C}^{\beta}(d, h)$, returns d (one for each action dimension) sequences of length h (horizon) sampled from colored noise distribution with exponent β and with zero mean and unit variance.

3.2 CEM with memory

In the standard CEM, once the inner loop is completed, the optimized Gaussian distribution and the entirety of all the elite-sets generated at each iteration get discarded. According to the parameters used in Chua et al. [1], this amounts to an average of ~ 55000 discarded actions **per step**. To increase efficiency, the following improvements reuse some of this information:

1. *Keep* elites: Storing the elite-set generated at each inner CEM-iteration and adding a small fraction of them to the pool of the next iteration, instead of discarding the elite-set in each CEM-iteration.

2. *Shift* elites: Storing a small fraction of the elite-set of the last CEM-iteration and add each a random action at the end to use it in the next environment step.

The reason for not shifting the entire elite-set in both cases is that it would shrink the variance of CEM drastically in the first CEM-iteration because the last elites are quite likely dominating the new samples and have small variance. We use a fraction of 0.3 in all experiments.

3.3 Smaller Improvements

Executing the best action (*best-a*) The purpose of the original CEM algorithm is to estimate an unknown probability distribution. Using CEM as a trajectory optimizer detaches it from its original purpose. In the MPC context we are interested in the best possible action to be executed. For this reason, we choose the first action of the best seen action sequence, rather than executing the first mean action, which was actually never evaluated. Consequently, we add the mean to the samples of the last CEM-iteration to allow the algorithm to still execute the mean action. For more details, see Sec. E.3.

Clipping at the action boundaries (*clip*) Instead of sampling from a truncated normal distribution, we sample from the unmodified normal distribution (or colored-noise distribution) and clip the results to lie inside the permitted action interval. This allows to sample maximal actions more frequently.

Decay of population size (*decay*) One of the advantages of CEM over the simplest Evolution Strategies is that the standard deviation is not fixed during the optimization procedure, but adapts according to the elite-set statistics. When we are close to an optimum, the standard deviation will automatically decrease, narrowing down the search and fine-tuning the solution. For this reason, it is sufficient to sample fewer action sequences as the CEM-iterations proceed. We introduce then an exponential decrease in population size of a fixed factor γ. The population size of iteration i is $N_i = \max(N\gamma^{-i}, 2K)$, where the $\max$ ensures that the population size is at least double the size of the *elite-set*.

The final version of the algorithm is showed in Alg. 1. Hyper-parameters are given in Sec. C in the supplementary. Except β we use the same parameters for all settings. The planning horizon is 30.

4 Experiments

The aim of the experiment section is to benchmark CEM-based methods on hard high-dimensional robotic tasks that need long horizon planning and study their behavior in the low-sampling budget regime. The control tasks range from locomotion to manipulation with observation-dimension ranging from 18 to 376, and action-spaces up to 30 dimensions. We use the ground truth dynamics model given by the Mujoco simulator as well as learned latent-dynamics models in the PlaNet [3] framework. Details and videos can be found in the Supplementary.

4.1 Environments

First, we consider the following three challenging the environments contained in OpenAI Gym [30]:

HALFCHEETAH RUNNING: (Gym v3) A half-cheetah agent should maximize its velocity in the positive x-direction. In contrast to the standard setting, we prohibit a rolling motion of the cheetah, commonly found by strong optimization schemes, by heavily penalizing large angles of the root joint.

HUMANOID STANDUP: (Gym v2) A humanoid robot is initialized in a laying position, see Fig. 1. The goal is to stand-up without falling, i.e. reaching as high as possible with the head.

FETCH PICK&PLACE (sparse reward): (Gym v1) A robotic manipulator has to move a box, randomly placed on a table, to a randomly selected target location. The agent is a Cartesian coordinate robotic arm with a two finger gripper attached to its end effector. The reward is only the negative Euclidean distance between box and target location, so without moving the box there is no reward.

Algorithm 1: Proposed iCEM algorithm. Color brown is iCEM and blue is CEM$_{\text{MPC}}$ and iCEM.

```
Parameters:
   $N$: number of samples; $h$: planning horizon; $d$: action dimension; $K$: size of elite-set; $\beta$: colored-noise exponent
   CEM-iterations: number of iterations; $\gamma$: reduction factor of samples; $\sigma_{init}$: noise strength
for $t$ = 0 to $T-1$                                                    // loop over episode length
do
    if $t$ == 0 then
        $\mu_0 \leftarrow$ constant vector in $\mathbb{R}^{d \times h}$
    else
        $\mu_t \leftarrow$ shifted $\mu_{t-1}$ (and repeat last time-step)              // see Suppl. E.1
    $\sigma_t \leftarrow$ constant vector in $\mathbb{R}^{d \times h}$ with values $\sigma_{init}$
    for $i$ = 0 to CEM-iterations$-1$ do
        $N_i \leftarrow \max(N \cdot \gamma^{-i}, 2 \cdot K)$
        samples $\leftarrow$ $N$ samples from $\mathcal{N}(\mu_t, \text{diag}(\sigma_t^2))$          // only CEM & CEM$_{\text{MPC}}$
        samples $\leftarrow$ $N_i$ samples from clip$(\mu_t + \mathcal{C}^\beta(d,h) \odot \sigma_t^2)$      // only iCEM, see Eq. 4
        if $i$ == 0 then
            add fraction of **shifted** elite-set$_{t-1}$ to samples
        else
            add fraction of elite-set$_t$ to samples
        if $i$ == last-iter then
            add mean to samples
        costs $\leftarrow$ cost function $f(x)$ for $x$ in samples
        elite-set$_t$ $\leftarrow$ best $K$ samples according to costs
        $\mu_t, \sigma_t \leftarrow$ fit Gaussian distribution to elite-set$_t$ with momentum
    execute action in first $\mu_t$                                      // only CEM and CEM$_{\text{MPC}}$
    execute first action of best elite sequence                          // only iCEM
```

Furthermore, we test iCEM on three environments from the DAPG project [33]. The basis of these environments is a simulated 24 degrees of freedom ShadowHand. Each environment requires the agent to solve a single task:

DOOR: The task is to open a door by first pushing down the door handle which releases the latch, enabling the agent to open the door by pulling at the handle. The reward (as in [33]) is the sum of the negative distance between palm and door handle, the openness of the door and a quadratic penalty on the velocities. Additional bounties are given for opening the door. The state space contains the relative joint positions of the hand, the latch position, the absolute door, palm and handle position, the relative position between palm and handle and a flag indicating whether the door is open or not.

DOOR (sparse reward): The same as DOOR except the reward does not contain the distance of the palm to the handle, so without opening the door there is no reward.

RELOCATE: In the relocate environment the task, see Fig. 1, is to move a ball to a target location. To achieve the goal, the ball needs to be lifted into the air. The reward signal is the negative distance between palm and ball, ball and target and bounties for lifting up the object and for when the object is close to the target. The state space contains the relative joint positions of the hand and the pairwise relative positions of the palm, the ball, and the target.

4.2 Main results

We want to obtain a sample efficient CEM that can potentially be used in real-time given a moderate model runtime. For this reason, we study how the performance degrades when decreasing the number of samples per time-step, in order to find a good compromise between execution speed and desired outcome.

Figure 3 presents the performance of iCEM, CEM$_{\text{MPC}}$, CEM$_{\text{PETS}}$ and vanilla CEM for different budgets, where a budget is the total number of trajectories per *step*. Budget is defined in a local sense

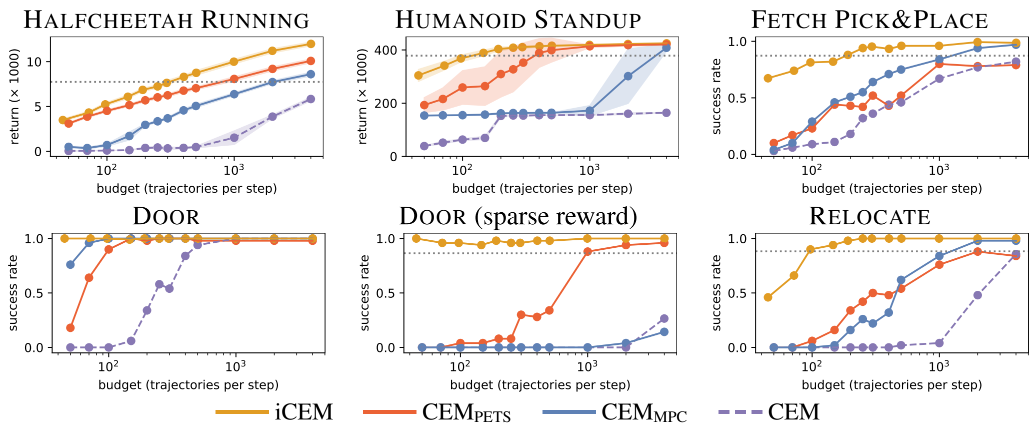

Figure 3: Performance dependence on the planning budget. Notice the log-scale on the x-axis.

for maintaining consistency across different episode lengths. It clearly demonstrates that iCEM is the only method to perform well even with extremely low budgets. In addition, iCEM has consistently higher performance than the baselines for all considered budgets, see also Table S3.

To quantify the improvements, Table 1 compares iCEM with the respective best baseline in each environment. We report the sample efficiency factor based on the approximate budget needed to reach 90% of the best baseline performance (at budget 4000) and see that iCEM is 2.7-21.9× more sample efficient. Similarly, we consider how much higher performance iCEM has w.r.t. the best baseline for a given budget (averaged over budgets$<$ 1000) and find 120-1030% of the best baseline performance.

Planning using learned dynamics models: In addition to planning in environments with given ground-truth dynamics, we investigate the behavior of iCEM for planning using learned dynamics models. For this, we train dynamics models from pixel input on several DeepMind control suite tasks using PlaNet [3]. The dynamics model is learned from pixels with training data repeatedly collected with the respective planner. We compare the performance of the entire training and planning process, see Fig. 4, with (a) the CEM planner with budget 10000 and 10 CEM-iterations, (b) iCEM with small budget (366) and 3 CEM-iterations, and (c) the CEM planner with small budget (366) and 3 CEM-iterations. For all planners but iCEM, we execute the mean action of the distribution. We observe that iCEM with a budget of only 366 is not far behind the extensive CEM (a). Moreover, iCEM is clearly better than the baseline (c) with the same low budget for the CHEETAH RUN and WALKER WALK environments. CUP CATCH is a challenging learning task due to its sparse reward. Presumably, training progress largely depends on observing successful rollouts early in training. On this task, iCEM reaches similar performance to the other CEM methods. We provide further details and results in the supplementary material.

Towards real-time control: With ground-truth models and CPU-parallelization, we reach close to real-time performance for simple environments (HALFCHEETAH). However, the most important scenario is the one with learned models: in the PlaNet approach we reach indeed real-time planning with iCEM using our own PyTorch implementation, see Table 2.

Table 1: Sample efficiency and performance increase of iCEM w.r.t. the best baseline. The first 4 columns consider the budget needed to reach 90% of the best baseline (dashed lines in Fig. 3). The last column is the average improvement over the best baseline in the given budget interval.

	90% base-line@4000	∼ budget iCEM	∼ budget baseline	efficiency factor	iCEM w.r.t. baseline budgets	%
HALFCHEETAH RUNNING	7744	312	840	2.7	50–1000	120%
HUMANOID STANDUP	378577	121	372	3.06	50–1000	128%
FETCH PICK&PLACE	0.87	185	1330	7.2	50–1000	243%
DOOR (sparse reward)	0.86	45	985	21.9	100–1000	1030%
RELOCATE	0.88	95	1300	13.7	100–1000	413%

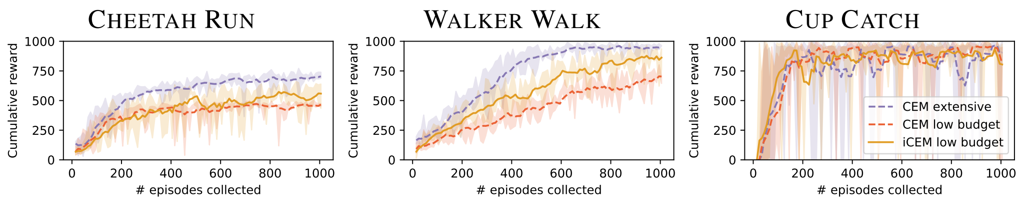

Figure 4: PlaNet performance using an extensive CEM variant (budget 10000) and two low-budget variants of CEM and iCEM (budget 366). Shown is the mean and min/max-band cumulative reward (three independent restarts) with average-smoothing over 50 episodes. iCEM outperforms the low-budget baseline on CHEETAH RUN and WALKER WALK, and performs similarly on CUP CATCH.

Table 2: Runtimes for iCEM with different compute budgets using Mujoco simulator and the PlaNet models. Times are given in seconds per env-step (total wall-clock time = time/step $\times$ episode length). *: Xeon® Gold 6154 CPU @ 3.00GHz, and **: Xeon® Gold 5220, NVidia® Quadro RTX 6000.

Envs	Threads	Budget (trajectories per step) 100	300	500	2000	dt
HALFCHEETAH RUNNING*	1	0.326	0.884	1.520	5.851	0.05
	32	0.027	0.066	0.109	0.399	
HUMANOID STANDUP*	1	2.745	8.811	13.259	47.469	0.015
	32	0.163	0.456	0.719	2.79	
FETCH PICK&PLACE*	1	8.391	26.988	40.630	166.223	0.04
	32	0.368	1.068	1.573	6.010	

	iCEM (366)	CEM (10000)	dt
PlaNet (PyTorch)**	0.044±0.003	0.18±0.031	0.04–0.08

4.3 Ablation study

To study the impact of each of our improvement individually, we conducted ablations of iCEM (orange bars in Fig. 5) and additions to CEM$_{\text{MPC}}$ (blue bars in Fig. 5) for some environments and budgets, see Sec. D for all combinations and more details.

Some components have bigger individual impact than others, e.g. using *color*ed noise consistently has a huge impact on the final result followed by *keep* and *shift* elites and *best-a*ction execution. However, the addition of all components together is necessary to reach top performance. As expected, the impact of the different additions become more relevant in the low-budget regime.

5 Conclusions

In this work, we introduced iCEM: a sample-efficient improvement of CEM intended for real-time planning. Most notably, we introduce temporally correlated action sampling and memory for previous trajectories. These additions were crucial for solving, for the first time, complicated tasks in MBRL with *very few* samples, e.g., humanoid stand-up or door opening (with sparse rewards) with only 45 trajectories per step. With this budget, we manage to enter in the real-time regime, as shown in the experiments with learned models. We hope this encourages future work with zero-order optimizers for real-time robot control.

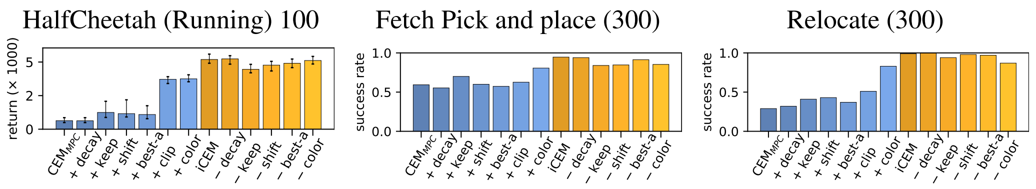

Figure 5: Ablation studies. Blue bars show CEM$_{\text{MPC}}$ with each improvement added separately. Yellow bars show iCEM with each features removed separately. Feature names are listed in Sec. 3.

Acknowledgments

This work was supported by the German Federal Ministry of Education and Research (BMBF): Tübingen AI Center, FKZ: 01IS18039B. The authors thank the Max Planck ETH Center for Learning Systems for supporting Cristina Pinneri, and the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting Jan Achterhold and Sebastian Blaes.

References

[1] K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 4754–4765, 2018.

[2] T. Wang and J. Ba. Exploring model-based planning with policy networks. In *International Conference on Learning Representations (ICLR)*, 2020.

[3] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 2555–2565, 2019.

[4] G. Williams, A. Aldrich, and E. Theodorou. Model predictive path integral control using covariance variable importance sampling. *arXiv preprint arXiv:1509.01149*, 2015.

[5] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566, May 2018.

[6] A. Richards and J. P. How. Robust variable horizon model predictive control for vehicle maneuvering. *International Journal of Robust and Nonlinear Control*, 16(7):333–351, 2006.

[7] R. Rubinstein and W. Davidson. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, 1999.

[8] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033, 2012.

[9] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis, TU Berlin, 1971.

[10] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[11] K. Choromanski, M. Rowland, V. Sindhwani, R. Turner, and A. Weller. Structured evolution with compact architectures for scalable policy optimization. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 970–978, 2018.

[12] H. Mania, A. Guy, and B. Recht. Simple random search of static linear policies is competitive for reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1800–1809, 2018.

[13] S. Khadka and K. Tumer. Evolution-guided policy gradient in reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1188–1200, 2018.

[14] H. Bharadhwaj, K. Xie, and F. Shkurti. Model-predictive control via cross-entropy and gradient-based optimization. In A. M. Bayen, A. Jadbabaie, G. Pappas, P. A. Parrilo, B. Recht, C. Tomlin, and M. Zeilinger, editors, *Proceedings of Conference on Learning for Dynamics and Control Learning (L4DC)*, volume 120, pages 277–286. PMLR, 2020.

[15] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. Theodorou. Aggressive driving with model predictive path integral control. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1433–1440, 2016.

[16] R. Y. Rubinstein and D. P. Kroese. *The Cross Entropy Method: A Unified Approach To Combinatorial Optimization, Monte-Carlo Simulation (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2004.

[17] Z. Botev, D. Kroese, R. Rubinstein, and P. L'Ecuyer. *Chapter 3. The Cross-Entropy Method for Optimization*, volume 31, pages 35–59. Elsevier, 2013.

[18] L. Margolin. On the convergence of the cross-entropy method. *Annals of Operations Research*, 134:201–214, 2005.

[19] A. Pourchot and O. Sigaud. CEM-RL: Combining evolutionary and gradient-based methods for policy search. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2018.

[20] M. Kobilarov. Cross-entropy motion planning. *International Journal of Robotic Research (IJRR)*, 31:855–871, 2012.

[21] L. Čehovin, M. Kristan, and A. Leonardis. An adaptive coupled-layer visual model for robust visual tracking. In *International Conference on Computer Vision (ICCV)*, pages 1363–1370, 2011.

[22] S. Lin and J. Ding. Integration of ranked lists via cross entropy monte carlo with applications to mRNA and microRNA studies. *Biometrics*, 65:9–18, 2008.

[23] K.-P. Hui, N. Bean, M. Kraetzl, and D. P. Kroese. The cross-entropy method for network reliability estimation. *Annals of Operations Research*, 134:101–118, 2005.

[24] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2016.

[25] N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In *Proceedings of IEEE International Conference on Evolutionary Computation (ECTA)*, pages 312–317, 1996.

[26] B. Amos and D. Yarats. The differentiable cross-entropy method. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.

[27] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

[28] N. Humphries, N. Queiroz, J. Dyer, N. Pade, M. Musyl, K. Schaefer, D. Fuller, J. Brunnschweiler, T. Doyle, J. Houghton, G. Hays, C. Jones, L. Noble, V. Wearmouth, E. Southall, and D. Sims. Environmental context explains Lévy and Brownian movement patterns of marine predators. *Nature*, 465:1066–9, 2010.

[29] M. F. Shlesinger, J. Klafter, and Y. M. Wong. Random walks with infinite spatial and temporal moments. *Journal of Statistical Physics*, 27:499–512, 1982. doi:10.1007/BF01011089.

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

[31] J. Timmer and M. Koenig. On generating power law noise. *Astronomy and Astrophysics*, 300: 707–710, 1995.

[32] W. T. Cochran, J. W. Cooley, D. L. Favin, H. D. Helms, R. A. Kaenel, W. W. Lang, G. C. Maling, D. E. Nelson, C. M. Rader, and P. D. Welch. What is the fast fourier transform? *Proceedings of the IEEE*, 55(10):1664–1674, 1967.

[33] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine. Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations. In *Proceedings of Robotics: Science and Systems (RSS)*, 2018. `https://github.com/aravindr93/hand_dapg`.

[34] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems (NeurIPS)*, page 2951–2959, 2012.

[35] R. Fossion, E. Landa, P. Stránský, V. Velazquez, J. C. Lopez Vieyra, I. Garduño, D. García, and A. Frank. Scale invariance as a symmetry in physical and biological systems: Listening to photons, bubbles and heartbeats. *AIP Conference Proceedings*, 1323:74–90, 12 2010.

Supplementary Material

In this supplementary material we detail the performances of iCEM with both ground truth and learned models, and discuss the hyperparameter selection with a sensitivity analysis. We present the ablation figures for all the environments and 3 fixed budgets. We conclude with an analysis stressing the relation between time-correlated action sequences and their power spectrum. Some videos of iCEM in action can be found at `https://martius-lab.github.io/iCEM/`.

A Pseudocode of the vanilla Cross Entropy Method (CEM) in the MPC setting.

Algorithm S1: Cross-Entropy Method (CEM) for Trajectory Optimization

```
Parameters:
  $N$: number of samples; $K$: size of elite-set; $h$: horizon;
  $\sigma_{init}$: initial standard deviation; CEM-iterations: number of iterations
for $t$ = 0 to $T-1$                                    // loop over episode length
do
    $\mu_0 \leftarrow$ zeros in $\mathbb{R}^{d\times h}$
    $\sigma_0 \leftarrow$ constant vector in $\mathbb{R}^{d\times h}$ with values $\sigma_{init}$
    for $i$ = 0 to CEM-iterations$-1$ do
        samples $\leftarrow N$ samples from $\mathcal{N}(\mu_t, \mathrm{diag}(\sigma_t^2))$
        costs $\leftarrow$ cost function $f(x)$ for $x$ in samples
        elite-set $\leftarrow$ best $K$ samples according to costs
        $\mu_t$, $\sigma_t \leftarrow$ fit Gaussian distribution to elite-set
    execute first action of mean sequence $\mu_t$
```

B Performance results

Table S3 shows the performance values for a selection of budgets in all environments. The values are reported for 50 independent runs (and 100 for FETCH PICK&PLACE) in the case of the ground-truth environments. For the PlaNet experiments we report the statistics for 3 independent training runs with 10 evaluation rollouts each. Note that for the success rate the variance is defined by the rate itself (Bernoulli distribution). Table S3 is complemented by Fig. S6, which shows the additional PlaNet experiments with REACHER EASY, FINGER SPIN and CARTPOLE SWINGUP.

B.1 Budget selection

Table S4 gives different budgets used for evaluating iCEM performance and the corresponding internal optimizer settings. Note that the number of *CEM-iterations* and the number of initial trajectories N depends on the overall budget and, due to the decay $\gamma = 1.25$, there are more *CEM-iterations* possible for iCEM while keeping the same budget.

Table S3: Performances for all environments for a selection of budgets. We report the cumulative reward (marked with [1]) and the success rate (marked with [2]).

Envs	Budget 100		Budget 300		Budget 500	
	iCEM	CEM$_{\text{MPC}}$	iCEM	CEM$_{\text{MPC}}$	iCEM	CEM$_{\text{MPC}}$
HALFCHEETAH RUNNING[1]	**5236±167**	699±120	**7633±250**	3682±119	**8756±255**	5059±179
HUMANOID STANDUP[1]	**368 k±12 k**	155 k±488	**411 k±5 k**	163 k±495	**416 k±1.8 k**	164 k±158
FETCH PICK&PLACE[2]	**0.81**	0.29	**0.95**	0.64	**0.96**	0.75
DOOR[2]	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**
DOOR (sparse reward)[2]	**0.96**	0.0	**0.96**	0.0	**0.98**	0.0
RELOCATE[2]	**0.9**	0.0	**1.0**	0.22	**1.0**	0.62
	iCEM (366)		plain CEM (366)		plain CEM (10000)	
PlaNet CHEETAH RUN[1]	589.49±49.45		419.04±11.04		**685.17±18.89**	
PlaNet CUP CATCH[1]	938.3±37.79		667.83±445.3		**963.33±24.42**	
PlaNet WALKER WALK[1]	846.37±71.46		711.17±119.36		**954.21±31.91**	
PlaNet REACHER EASY[1]	**926.07±194.17**		783.07±352.69		693.6±423.51	
PlaNet FINGER SPIN[1]	523.43±35.07		523.43±29.46		**667.37±190.71**	
PlaNet CARTPOLE SWINGUP[1]	772.32±52.33		761.51±41.36		**800.05±47.9**	

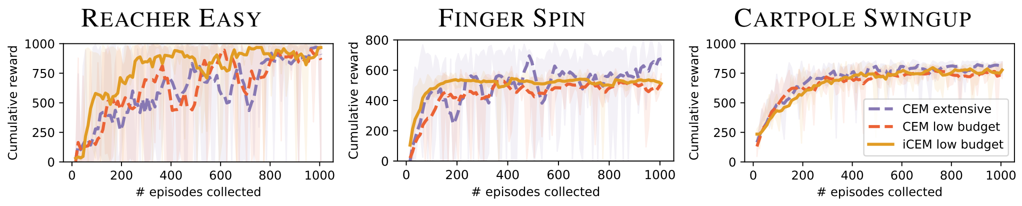

Figure S6: Additional PlaNet experiments. For details, see Fig. 4.

C Hyper-parameters

Zeroth order optimization requires minimal hyperparameter tuning in comparison to gradient descent methods, to the extent that it is used itself to tune the hyperparameters of deep networks [34].

The main parameters in CEM, aside from the length of the planning horizon h and the number of trajectories (determined by population size N and number of *CEM-iterations*), are: the size of the *elite-set* K, the initial standard deviation σ_{init} and the α-momentum.

To these, iCEM adds the colored-noise exponent β, the decay factor γ, and the fraction of reused elites ξ. We unified the values of $\alpha, K, \sigma_{init}, \gamma$, and ξ for all the presented tasks, see Table S5.

For experiments with the ground truth model we use an horizon of 30 and for the PlaNet experiments we use the horizon of 12 to be consistent with the original PlaNet paper. All the other parameters are fixed to the same values for all the environments, with the exception of the noise-exponent β, as reported in Table S6.

The environment episode length (standard for these environments) are given in Table S7.

C.1 Choice of colored-noise exponent β

The choice of the β is intuitive and directly related to the nature of each task. For some tasks, the robotic system requires high-frequency control: in HALFCHEETAH RUNNING, for example, it is important to switch actions at a very fast rate, suggesting to select a very low β value. On the other hand, there are many environments where the preferred action sequences are smoother, indicating a more dominant presence of lower-frequencies: for example, manipulation environments as FETCH PICK&PLACE and RELOCATE will require a higher β.

The same holds for the HUMANOID STANDUP task, as we saw in Fig. 2(b), which prefers a non-flat spectral density with a predominance of lower frequencies, reason why feeding actions drawn from a Gaussian distribution would inevitably translate in a "waste" of energy. By picking a β in the right range, we avoid this and consequently make the whole optimization procedure more efficient.

Table S4: Budget-dependent internal optimizer settings (notation: *CEM-iterations* / N).

	Budgets											
	50	70	100	150	200	250	300	400	500	1000	2000	4000
iCEM	2 / 25	2 / 40	3 / 40	3 / 60	4 / 65	4 / 85	4 / 100	5 / 120	5 / 150	6 / 270	8 / 480	10 / 900
CEM	2 / 25	2 / 35	2 / 50	2 / 75	3 / 66	3 / 83	3 / 100	4 / 100	4 / 125	4 / 250	6 / 333	8 / 500

Table S5: Fixed Hyperparameters used for all experiments.

	# elites K	initial std. σ_{init}	momentum α	decay γ	fraction reused elites ξ
iCEM	10	0.5	0.1	1.25	0.3
CEM	10	0.5	–	1.0	0

Aside from this, providing a precise value for β is not critical. Environments that require a high-frequency control need a low β otherwise a value around 2–4 seems adequate, as shown in the sensitivity plot in Fig. S8.

C.2 Sensitivity

If the number of trajectories is high enough, there is little sensitivity to the other parameters, as shown in Fig. S7 and Fig. S8. This means that for very low budgets – the ones relevant for real-time planning – selecting the right parameters becomes more important. We can the measure the impact of every parameter by comparing the first (low budget) and last column (higher budget) of Fig. S9. As the number of samples increases, adding features does not have significant consequences on the final performance.

However, selecting the colored-noise exponent β in the right range, can still have a significant effect depending on the specific task, even for higher budgets. For example, it is important to not use high values of β for high-frequency control tasks like HALFCHEETAH RUNNING. On the other hand, using higher β on the HUMANOID STANDUP is fundamental when the provided budget is low (100). In fact, as illustrated in Fig. S8 (b), it is crucial to increase β to any value above 2, in order to not sample uncorrelated action sequences.

Besides that, iCEM shows lower sensitivity with respect to the initial standard deviation of the sampling distribution. As an example, we report the effect of σ_{init} on the success rate of the FETCH PICK&PLACE task in Fig. S7 (c): CEM$_{\text{MPC}}$ prefers a narrower range for σ_{init} between 0.4 and 0.6, in contrast to iCEM, for which any value above 0.2 yields similar results.

Table S6: Env-dependent Hyperparameter choices.

	iCEM/CEM with ground truth		iCEM with PlaNet	
horizon h	30		12	
colored-noise exponent β	0.25	HALFCHEETAH RUNNING	0.25	CHEETAH RUN
	2.0	HUMANOID STANDUP	0.25	CARTPOLE SWINGUP
	2.5	DOOR	2.5	WALKER WALK
	2.5	DOOR (sparse reward)	2.5	CUP CATCH
	3.0	FETCH PICK&PLACE	2.5	REACHER EASY
	3.5	RELOCATE	2.5	FINGER SPIN
initial std. σ_{init}			0.5	CHEETAH RUN
			0.5	WALKER WALK
			0.5	CUP CATCH
			0.5	REACHER EASY
			1.0	FINGER SPIN
			1.0	CARTPOLE SWINGUP

Table S7: Environment settings. These are the standard settings for the environemnts. For PlaNet the numbers come from the custom action repeat used in [3].

	iCEM/CEM with ground truth		iCEM with PlaNet	
Episode length	1000	HALFCHEETAH RUNNING	250	CHEETAH RUN
	1000	HUMANOID STANDUP	125	CARTPOLE SWINGUP
	200	DOOR	500	WALKER WALK
	200	DOOR (sparse reward)	250	CUP CATCH
	50	FETCH PICK&PLACE	250	REACHER EASY
	200	RELOCATE	500	FINGER SPIN

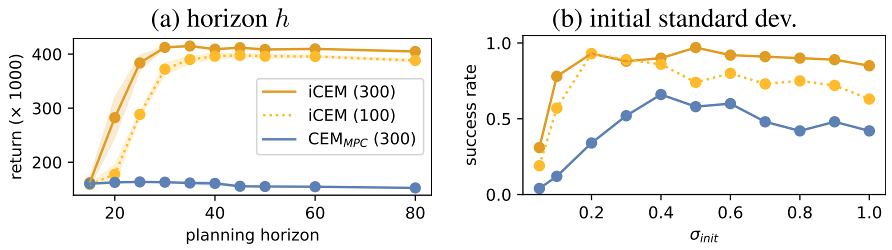

Figure S7: Sensitivity to hyper-parameters of iCEM. (a) horizon h in HUMANOID STANDUP and in (b) the initial standard deviation for FETCH PICK&PLACE. See Fig. S8 for the sensitivity to β.

Another relevant observation is the effect of the planning horizon length h for the HUMANOID STANDUP in Fig. S7 (a): even in the low-budget case, iCEM can better exploit longer action sequences by generating samples with higher correlations in time.

C.3 Hyperparameters for PlaNet

We reimplemented PlaNet [3] in PyTorch to conduct our experiments and borrow all algorithmic details and hyperparameter settings from the original paper. As in [3], for every training run, we collect 5 initial rollouts from the respective environment by randomly sampling from its action space. After every 100th training step, we extend the training set by collecting an additional rollout using the planner, but add a Gaussian distributed exploration noise $\epsilon \sim \mathcal{N}(0, \mathbb{I} \cdot 0.3^2)$ to each action. After every 1000th training step, we additionally collect a test rollout for evaluation (which is not added to the training set) using the planner without exploration noise, yielding the results in Fig. 4. Results

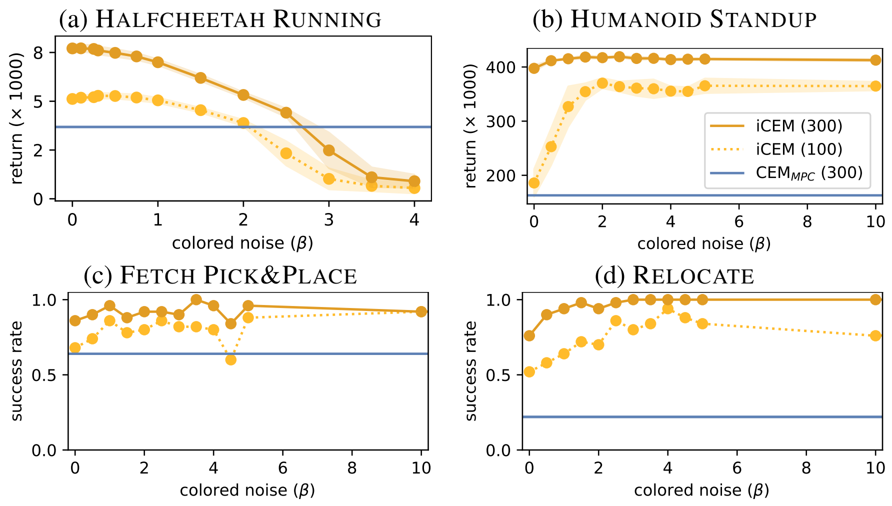

Figure S8: Sensitivity to the colored noise exponent β of iCEM.

for additional environments are depicted in Fig. S6. For both train and test collections the planner is identical within each experiment, being either "CEM extensive", "CEM low budget", or "iCEM low budget" (see Table S8 for details). For each experiment configuration we report results on 3 independent training runs. After training each model for 100k steps, we collect 10 evaluation rollouts (without exploration noise) per training run (i.e., 30 in total) and report the results in table S3.

Table S8: PlaNet CEM details

	CEM-iterations	initial candidates	elites	clip action	best action	decay	reuse elites	shift means	shift elites
CEM extensive	10	1000	100	yes	no	1.0	no	no	no
CEM low budget	3	122	10	yes	no	1.0	no	no	no
iCEM low budget	3	150	10	yes	yes	1.25	yes	yes	yes

	budget	mean as sample	momentum α	initial std. σ_{init}	colored-noise exponent β
CEM extensive	10000	no	0	1.0	0
CEM low budget	366	no	0	1.0	0
iCEM low budget	366	yes	0.1	see table S6	see table S6

D Ablation results

In Fig. S9 the ablations and additions are shown for all environments and a selection of budgets. As we use the same hyperparameters for all experiments, see Sec. C, in some environments a few of the ablated versions perform slightly better but overall our final version has the best performance. As seen in Fig. S9, not all components are equally helpful in the different environments as each environment poses different challenges. For instance, in HUMANOID STANDUP the optimizer can get easily stuck in a local optimum corresponding to a sitting posture. Keeping balance in a standing position is also not trivial since small errors can lead to unrecoverable states. In the FETCH PICK&PLACE environment, on the other hand, the initial exploration is critical since the agent receives a meaningful reward only if it is moving the box. Then colored noise and keep elites and shifting elites is most important.

E Details on the iCEM improvements

E.1 Shift Initialization

The shift-initialization of the mean $\mu_{t-1}(\cdot, j+1)$ of the sampling distribution, as mentioned in Sec. 2.1 and used in Alg. 1 line 9 is as follows:

$$\mu_t(\cdot, j) = \mu_{t-1}(\cdot, j+1) \qquad \text{for } 1 \leq j \leq h-1 \tag{5}$$
$$\mu_t(\cdot, h) = \mu_{t-1}(\cdot, h) \tag{6}$$

where the parenthesis denote index-access: (action dimension, horizon timestep). Note, that in the CEM$_{\text{PETS}}$ method Eq. 6 is $\mu_t(\cdot, h) = \vec{0}$.

E.2 Sampling Colored Noise

To sample action sequences with a specific power spectrum we use the efficient implementation of [31], which can be found as a python package at https://github.com/felixpatzelt/colorednoise.

E.3 Adding the mean actions

As the dimensionality of the action space increases, it gets more and more difficult to sample an action sequence closer to the mean of the distribution. Nevertheless, executing the mean might be beneficial for many tasks which require "clean" action sequences like, for example, manipulation,

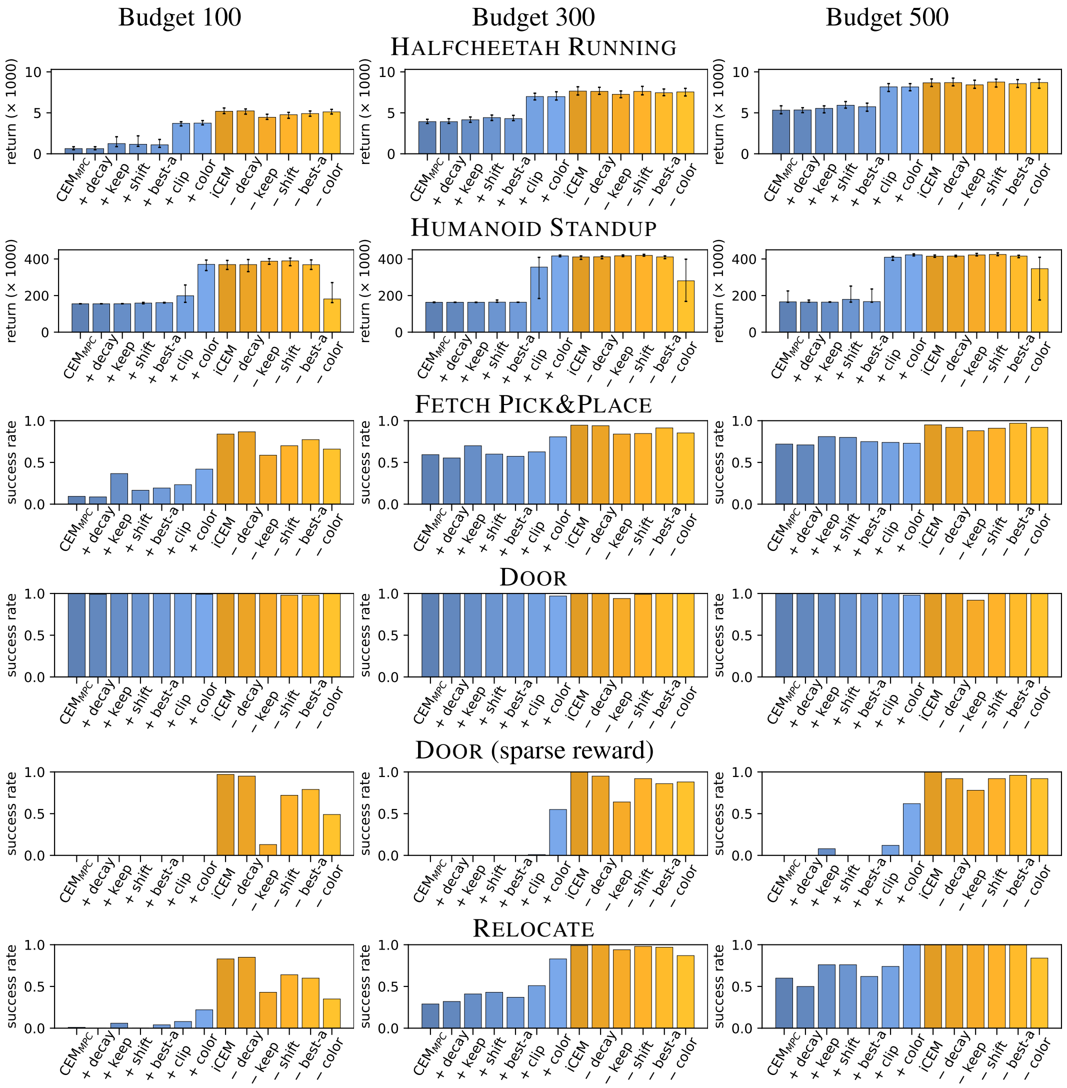

Figure S9: Ablation studies

object-reaching, or any linear trajectory in the state-space. Adding the mean to the samples fixes this problem and closes the gap with the original CEM, allowing the algorithm to pick either the mean or the best sampled action.

However, we noticed an unexpected performance degradation when adding the mean in every CEM-iteration, presumably due to the effect of quicker narrowing down the variance along CEM-iterations. Adding the mean just at the last iteration prevents this bias and has advantageous effects. If the mean survives the last iteration and becomes part of the elite-set, it will be automatically shifted to the successive time step.

F Spectral characteristics of noise

We can achieve more efficient exploration by choosing different kinds of action noise, which in turn affects the type of correlations between actions at different time steps. We can notice this by writing down the auto-correlation function which, according to the Wiener-Khinchin theorem, can be expressed as the inverse Fourier transform of the power spectral density of the control input: $C(\tau) = \mathcal{F}^{-1}[\mathrm{PSD}_a(f)]$. If the power spectral density follows the inverse power law of Eq. (2), and we apply a scale transformation in the time domain $\tau \to \tau' = s\tau$, then, from the frequency scaling property of the Fourier transforms:

$$\begin{aligned}
C(s\tau) &= \mathcal{F}^{-1}\left[\frac{1}{s}\mathrm{PSD}_a\left(\frac{f}{s}\right)\right] \\
&= \mathcal{F}^{-1}\left[\frac{1}{s}s^{\beta}\mathrm{PSD}_a(f)\right] \qquad \text{using Eq. (2)} \\
&= s^{\beta-1}\mathcal{F}^{-1}[\mathrm{PSD}_a(f)] \\
&= s^{\beta-1}C(\tau)
\end{aligned}$$

From this self-referential formula we can understand to which degree the actions lose similarity with a copy of themselves at a different point in time, as detailed in [35].

In particular, white noise is a memory-less process and does not produce any correlations at different times.