CRISTINA PINNERI

SAMPLE-EFFICIENT MODEL-BASED REINFORCEMENT LEARNING

DISS. ETH NO. 29436

SAMPLE-EFFICIENT MODEL-BASED REINFORCEMENT LEARNING

A dissertation submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH
(Dr. Sc. ETH Zurich)

presented by

CRISTINA PINNERI
MSc., Polytechnic University of Turin

born on 23 January 1993
citizen of the Italian Republic

accepted on the recommendation of

Prof. Dr. Andreas Krause, ETH Zurich
Prof. Dr. Georg Martius, MPI-IS
Prof. Dr. Marc Toussaint, TU Berlin

2023

ABSTRACT

Reinforcement Learning (RL) is a powerful framework for decision making and adaptive learning by interaction. Even though at its core it consists of trial-and-error learning, it has become a critical tool for research on Artificial Intelligence (AI). In the last decade, RL algorithms have been able to master strategic games like Chess and Go, and control a variety of robotic and industrial platforms, from locomotion and manipulation to power plants, and even nuclear fusion reactors. By incorporating deep neural networks (NN) as function approximators, "*deep* RL" reached the ability to handle high-dimensional state and action spaces, and have in principle better generalization across tasks, making RL solutions versatile and promising. However, using deep neural networks comes with certain caveats. RL algorithms often face issues such as brittleness due to overfitting and sensitivity to hyperparameters, which come on top of the typical RL challenges, such as low sample efficiency, difficulty in handling sparse rewards, delayed credit assignment for long-horizon tasks, sensitivity to reward function design. In this dissertation, we present a series of novel contributions that address some of the problems faced by RL, with the ultimate goal of improving its efficiency, robustness, and generalization for continuous control tasks. Specifically, we will present more robust approaches to trajectory optimization, coupled with NN function approximation for policy learning, model learning, and reward learning. In particular, the majority of this work is centered around zero-order optimization for model-predictive control, which we demonstrate to be more performing, robust, and reproducible than gradient-based trajectory optimizers. Throughout this dissertation, we will show how zero-order optimization can be used to efficiently solve tasks with sparse rewards, how it can be used in the context of imitation learn-

ing, and how it can be exploited in conjunction with model learning for uncertainty propagation. Finally, we will present a method to learn reward functions from scratch, in a purely self-supervised fashion. Through extensive experiments in simulated environments, our methods demonstrate significant improvements in learning efficiency and performance, reducing the required number of interactions with the environment while still achieving near-optimal solutions. This work aims to provide a viable approach to tackle part of the challenges of deep RL, addressing the efficiency and robustness of the learning process without relying on predefined expert knowledge.

SOMMARIO

L'apprendimento per rinforzo, o reinforcement learning (RL), è un insieme di tecniche per i processi decisionali e l'apprendimento adattivo per mezzo di interazioni. Sebbene, nel suo nucleo, consista nell'apprendimento per tentativi ed errori, RL è diventato uno strumento fondamentale per la ricerca sull'intelligenza artificiale (AI). Nell'ultimo decennio, gli algoritmi RL sono stati in grado di padroneggiare giochi strategici come gli scacchi e il Go, e di controllare una varietà di piattaforme robotiche e industriali, dalla locomozione e manipolazione alle centrali elettriche e reattori nucleari Tokamak. Utilizzando reti neurali profonde come approssimatori universali di funzioni, l'appredimento rinforzato profondo (*deep* RL) ha raggiunto la capacità di gestire spazi di stato e di azione ad alta dimensione e, in linea di principio, di avere una migliore generalizzazione tra i vari obiettivi, rendendo le soluzioni RL versatili e promettenti. Tuttavia, l'utilizzo di reti neurali profonde comporta alcune complicazioni. Gli algoritmi di RL devono spesso affrontare problemi come la fragilità dovuta all'overfitting e l'estrema sensibilità agli iperparametri, che si aggiungono alle sfide tipiche del RL, come la bassa efficienza rispetto al numero di dati, la difficoltà nel gestire segnali di rinforzo poco frequenti, il problema del *delayed credit assignment*, la sensibilità al design della funzione di rinforzo (*reward function)*. In questa tesi presentiamo una serie di contributi originali che affrontano alcuni dei problemi di RL, con l'obiettivo finale di migliorarne l'efficienza, la robustezza e l'adattabilità per problemi di controllo continuo. La nostra ricerca esplora vari aspetti di RL, come l'uso di reti neurali per l'apprendimento di *policy*, di modelli dinamici e di reward functions, insieme ad approcci più robusti per l'ottimizzazione delle traiettorie. In particolare, la maggior parte di questo lavoro è incentrata sull'ottimizzazione di ordine zero per il

controllo predittivo con modelli, che abbiamo dimostrato essere più performante, robusta e riproducibile degli ottimizzatori di traiettoria basati sui gradienti. Nel corso di questa tesi, mostreremo come l'ottimizzazione di ordine zero possa essere utilizzata per risolvere in modo efficiente compiti con segnali di rinforzo sparsi, come possa essere utilizzata nel contesto dell'apprendimento per imitazione e come possa essere sfruttata insieme all'apprendimento del modello per la propagazione dell'incertezza. Infine, presenteremo un metodo per imparare le reward functions da zero, in modo puramente auto-supervisionato. Attraverso esperimenti in simulazione, i nostri metodi dimostrano miglioramenti significativi nell'efficienza e nelle prestazioni, riducendo il numero di interazioni necessarie con l'ambiente e ottenendo comunque soluzioni quasi ottimali. Questo lavoro mira a fornire un valido approccio per affrontare parte delle sfide del deep RL, migliorando l'efficienza e l'adattabilità del processo di apprendimento senza fare affidamento su nozioni predefinite.

CONTENTS

1

INTRODUCTION

> Mankind will possess incalculable advantages and extraordinary control over human behavior when the scientific investigator will be able to subject his fellow men to the same external analysis he would employ for any natural object, and when the human mind will contemplate itself not from within but from without.
>
> Ivan Pavlov

While we are still far from understanding what originates thought and critical reasoning in humans, there is evidence that human behavior is guided by *learning through reinforcement*. The history of Reinforcement Learning (RL) dates back to the 20th century, when B.F. Skinner and I. Pavlov conducted a series of experiments on conditioning in animals [1, 2] to understand the mechanisms underlying learning. The idea that, in animals, responses were reinforced by their consequences was already formalized by psychologist E. Thorndike as the "Law of Effect" [3]:

> Responses that produce a satisfying effect in a particular situation become more likely to occur again in that situation, and responses that produce a discomforting effect become less likely to occur again in that situation.
>
> — Edward Thorndike

Thorndike's law of effect postulates that animals do not reason, but learn in a trial-and-error fashion by interacting with a physical environment until a successful outcome is obtained. This type of decision-making strategy links rewards with actions in a *retrospective* way, and is opposed to the *prospective* view according to which animals form field maps of their environment as a guidance mechanism for learning, also called "cognitive maps" [4, 5].

There is a wealth of experimental evidence on the human brain suggesting that there are multiple mechanisms for behavioral choice [6], and that decision-making processes are both reflective (prospective) and reflexive (retrospective), as we would expect. Reinforcement Learning theory also evolved in a similar way, now including two main branches: model-free (reflexive) and model-based (reflective) RL [7]. Model-free RL learns a reactive mapping between states and actions that increase the agent's reward, without any knowledge of the underlying physical principles of the environment. Model-based RL instead learns an internal representation of the world, which is used to learn the consequence of each action. Similarly to model-based RL, another computational account of prospective reasoning can be found in model-predictive control (MPC), efficiently used in robotics and many industrial processes [8, 9]. In MPC, a known model is used to iteratively optimize the given reward over a certain *planning horizon* while incorporating constraints and feedback from the environment. Given its planning ahead component, MPC can also be seen as a form of reflective decision-making.

In the course of this thesis, we will see how model-based and model-free RL can be integrated with MPC-style iterative planning to create control strategies that are more robust than their single components. Model-free RL algorithms, for example, are able to learn quickly from large amounts of data but are sensitive to the training hyperparameters, do not generalize well, and lack a planning component, which can lead to suboptimal decision-making in complex environments. On the other hand, model-based RL methods have the potential to be more flexible but often suffer from biases related to model estimation, which can quickly compound when used in an autoregressive fashion. Integrating model-based RL with MPC can offer the best of both worlds, as MPC provides a planning framework that can incorporate the learned model of the world and optimize the agent's actions over the planning horizon. This can lead to more sample-efficient decision-making strategies that can learn from experience, adapt to changing environments,

and optimize their behavior over time while being reactive as well as risk-averse.

1.1 OUTLINE AND CONTRIBUTIONS

This dissertation is centered around sample-efficient model-based strategies for control tasks. In particular:

- CHAPTER 3 presents an improved zero-order optimization method used in the context of model-predictive control that we named **iCEM**. Our approach originates from an intuitive consideration on the distribution generating the agent's actions vs the optimized distribution. We show one order-of-magnitude higher sample efficiency in a variety of control tasks, ranging from a simulated humanoid stand-up to a hand opening a door. This chapter is based on the paper Pinneri et al. [10]:

  "Sample-efficient zero-order trajectory optimization for real-time planning"
  Cristina Pinneri, Shambhuraj Sawant, Sebastian Blaes, Jan Achterhold, Jörg Stückler, Michal Rolínek, Georg Martius
  CoRL 2020: *Conference on Robot Learning*

- CHAPTER 4 explores supervised policy learning to extract a fast global policy imitating the powerful, yet local, iCEM optimizer. Since previous imitation learning methods are shown to fail, we add an *adaptive weighting* term to learn robust policies. We call this adaptive policy extraction method **APEX**. We compare our results with model-free RL policies and show a considerable improvement over all the considered control tasks. This chapter is based on Pinneri et al. [11]:

"Extracting strong policies for robotics tasks from zero-order trajectory optimizers"
Cristina Pinneri*, Shambhuraj Sawant*, Sebastian Blaes, Georg Martius
ICLR 2021: *International Conference on Learning Representations*
**equal contribution*

- CHAPTER 5 examines in detail the integration of the iCEM optimizer with learned models. Our contribution highlights that the trajectory sampling technique used to estimate the final cost is not fully exploiting the information coming from the learned model. We propose a way to separate the *aleatoric* and *epistemic* uncertainties and propagate them along the planning horizon in a way that keeps the RL agent curious yet risk-averse. This chapter is based on Vlastelica et al. [12]:

"Risk-averse zero-order trajectory optimization"
Marin Vlastelica*, Sebastian Blaes*, Cristina Pinneri, Georg Martius
CoRL 2021: *Conference on Robot Learning*

- CHAPTER 6 brings learning into another aspect of planning: the reward (or cost) function. We take a self-supervised approach to automatically learn a notion of distance without any domain knowledge, but only geometric considerations. The learned cost can be used for either model-free or model-based approaches. This chapter is based on Pinneri, Martius, and Krause [13]:

"Neural all-pairs shortest path for reinforcement learning"
Cristina Pinneri, Georg Martius, Andreas Krause
NeurIPS 2022: *Deep Reinforcement Learning Workshop*

1.2 ADDITIONAL PUBLICATIONS

This dissertation also includes a couple of works that have been done in collaboration with other researchers, and during research internships. These works do not directly align with the main body of the dissertation. As such, they are presented separately in APPENDIX A, where a brief summary is provided, explaining their connection to the broader context of this work. The mentioned papers are:

"Equivariant Data Augmentation from State Inputs for Generalization in Offline RL"
Cristina Pinneri, Sarah Bechtle, Markus Wulfmeier, Arunkumar Byravan, Will Whitney, Jingwei Zhang, Martin Riedmiller
under submission

"Pink noise is all you need: colored noise exploration in deep RL"
Onno Eberhard, Jakob Hollenstein, Cristina Pinneri, Georg Martius
ICLR 2023: *International Conference on Learning Represenations*

1.3 COLLABORATORS

This PhD thesis benefited from many fruitful collaborations. My supervisors Georg Martius and Andreas Krause helped define the main research directions of this work. The ideas presented in Chapter 3 were developed with the help of Michal Rolínek. Sebastian Blaes and Shambhuraj Sawant helped me with experimental results with ground truth dynamics. Jörg Stückler and Jan Achterhold added the section with the PlaNet experiments. Chapter 4 is the outcome of the joint effort with Shambhuraj Sawant. Together with him and Sebastian Blaes, we developed the final heuristics for adaptive

policy learning. In Chapter 5, I proposed to use more information from the ensemble's distribution and integrate the model's uncertainty to estimate the final cost. This idea was further shaped by Marin Vlastelica and Sebastian Blaes to extract the epistemic term and use it as optimism in the face of uncertainty.

2

BACKGROUND

This Chapter presents the basic tools and frameworks that will be used in this dissertation. It starts with a section on the fundamentals of reinforcement learning and model-predictive control, and then it will lay the basis for the main backbone of this work, which is zero-order trajectory optimization.

2.1 REINFORCEMENT LEARNING

The fundamental idea of RL is to learn how to select actions that maximize a reward signal over time. In order to do so, the decision maker, or *agent*, interacts with an *environment* and collects *rewards*, which it will use to learn a *policy*, i.e. a reactive mapping from perceived states to future actions. The outcome of this process is a *trajectory* that will lead the agent to its desired goal. The framework used to describe this idea in mathematical terms is a Markov Decision Process, MDP in short. MDPs are a simplified abstraction of the learning cycle, but have been proven to be widely applicable [15–19]. MDPs are formally defined by a tuple $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma \rangle$,

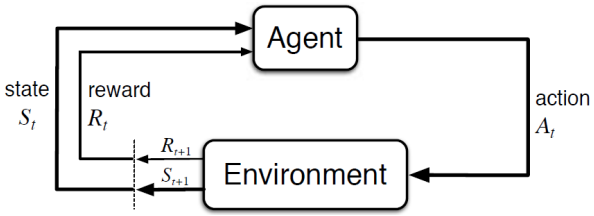


Figure 1: The learning cycle represented as a Markov decision process [14].

where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{R}$ is the reward function, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability, and γ is the discount rate, representing the agent's preference for immediate rewards over delayed rewards. MDPs can be either finite or infinite (depending on the number of states and actions), and episodic or continuous, depending on whether the agent's interaction with the environment is divided into episodes with terminal states, or if it continues indefinitely.

The basic property of an MPD is that the future state of the system only depends on the information available at the current state, which is also called *Markov property*. More specifically, the state and the reward at the next time step $t + 1$, indicated by the random variables $S_{t+1} \in \mathcal{S}$ and $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ respectively, are determined by the state and the action taken at time t, namely the random variables S_t and $A_t \in \mathcal{A}$. Several of these interactions between agent and environment generate a trajectory τ:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, \dots$$

The probability of transitioning from one state to another can be either discrete or continuous depending on the nature of the MDP, and it depends on the value of the random variables describing the state and the action, namely S_t and A_t. From the Markov property, we can derive the transition probability from the probability that the random variables R_{t+1} and S_{t+1} take specific values s' and r, expressed as:

$$p(s', r|s, a) \coloneqq \Pr\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\}$$

The transition probability $\mathcal{P}$ is then its marginalization over r:

$$p(s'|s, a) = \sum_{r \in \mathcal{R}} p(s', r|s, a)$$

If we discard the transition model and only rely on observed transitions, we talk about *model-free* RL; otherwise, if we learn the model and use it to generate imaginary trajectories, we are in the realm of *model-based* RL.

If we define the policy mapping as $\pi : \mathcal{S} \to \mathcal{A}$, we can finally express the posterior distribution of the trajectory τ induced by the Markov property as:

$$p_\theta(\tau) = p(s_0, a_0, r_1, s_1, a_1, \dots) = \delta(s_0) \prod_{t=0}^{\infty} p(s_{t+1}, r_{t+1} | s_t, a_t) \pi_\theta(a_t | s_t)$$

where θ parameterizes the policy and consequently the trajectory distribution. The initial state distribution is instead expressed by $\delta : \mathcal{S} \to [0, 1]$.

The optimal parameter $\theta^\star$ is the one that maximizes the expected cumulative reward over time, also called *expected return*. Mathematically:

$$\theta^\star = \arg\max_\theta \mathbb{E}_{\tau \sim p(\tau)} \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \tag{1}$$

In this case, we expressed the reward as a two-argument function depending on both state and action, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, which gets discounted with a *discount rate* $\gamma \in [0, 1]$.

The many different ways to optimize the parameter θ, so that the expected return is maximized, are at the core of RL research. The majority of RL algorithms revolve around expressing the expected return in different ways, so that the optimization process gets simplified. In particular, depending on which variables we condition the expected return, we obtain two important quantities in RL: the state-value function $v_\pi(s)$ (or simply *V function*) if we condition on the state, and the action-value function $q_\pi(s, a)$ (or *Q function*) if we condition on both state and action. The state-value function expresses the expected return when starting from state s and then following a policy π, and it can be written as:

$$v_\pi(s) \coloneqq \mathbb{E}_{\tau \sim p(\tau)} [\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0 = s] \tag{2}$$

where the reward function is rewritten as r_{t+1}, since we follow the convention that the reward at time step $t + 1$ is determined by s_t and

a_t. The action-value function represents the expected return when starting from state s, taking action a, and then following π:

$$q_\pi(s,a) := \mathbb{E}_{\tau\sim p(\tau)}[\sum_{t=0}^{\infty}\gamma^t r_{t+1}|s_0 = s, a_0 = a] \tag{3}$$

Naturally, this can be used to express $v_\pi(s)$ as $\sum_a \pi(a|s)q_\pi(s,a)$. One of the most important properties of the value functions in eq. 2 and 3 is that they can be rewritten in a recursive fashion, where the value at state s depends on the average of the value for all possible next states and actions, weighted by the probability of them occurring, respectively $p(s',r|s,a)$ and $\pi(a|s)$. This recursive formulation that uses bootstrapped estimates of future values is commonly known as *Bellman equation*. In particular, the Bellman equation for $v_\pi(s)$ is:

$$\begin{aligned} v_\pi(s) &:= \mathbb{E}_{\tau\sim p(\tau)}[\sum_{t=0}^{\infty}\gamma^t r_{t+1}|s_0 = s] \\ &= \sum_a \pi(a|s)\sum_{s',r} p(s',r|s,a)(r+\gamma v_\pi(s')) \end{aligned} \tag{4}$$

the full derivation can be found in [14]. Similarly, we can derive the Bellman equation for the Q function knowing that $v_\pi(s) = \sum_a \pi(a|s)q_\pi(s,a)$ and the recursion in eq. 4:

$$\begin{aligned} q_\pi(s,a) &= \sum_{s',r} p(s',r|s,a)(r+\gamma v_\pi(s')) \\ &= \sum_{s',r} p(s',r|s,a)(r+\gamma \sum_{a'} \pi(a'|s')q_\pi(s',a')) \end{aligned} \tag{5}$$

The purpose of a reinforcement learning agent acting under a policy π is to maximize the expected return or, as we have seen, the value $v_\pi(s)$, for all $s \in \mathcal{S}$. Therefore, value functions induce a partial ordering over policies, for which a policy π' is better over a policy π if the associated value is higher:

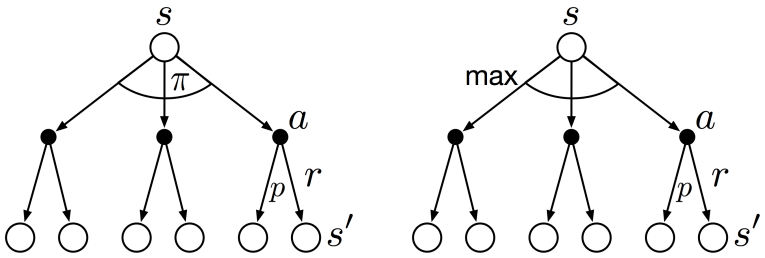
$$\pi' \geq \pi \iff v_{\pi'}(s) \geq v_\pi(s) \quad \forall s \in \mathcal{S}$$

The policy (or policies) which is better or equal to all the others is called the *optimal policy*. The optimal value functions also respect the recursive relationships (eq. 4 & 5) which are now called *Bellman optimality equations*:

$$v^\star(s) = \max_a \sum_{s',r} p(s', r|s, a)(r + \gamma v^\star(s')) \tag{6}$$

$$q^\star(s, a) = \sum_{s',r} p(s', r|s, a)(r + \gamma \max_{a'} q^\star(s', a')) \tag{7}$$

In fig. 2 we show the different backup operations for the state-value function done in eq. 4 and eq. 6 respectively.



(a) Backup diagram for v_π (b) Backup diagram for $v^\star$

Figure 2: Backup diagrams for the Bellman equations: expectation (a) vs optimality (b) [14]

The optimal policy $\pi^\star$ can be derived acting greedily with respect to $v^\star$, as the value function already contains all the future information. In other words, the actions that appear best after a *one-step search* on $v^\star$ are the optimal ones, and a policy that assigns a non-zero probability to only these actions is the optimal policy. With the optimal action-value function $q^\star$ this is even easier, as we do not need to perform an exhaustive one-step search in the environment, since all the information is already cached in $q^\star(s, a)$.

2.1.1 *Dynamic Programming*

Dynamic Programming (DP) provides an efficient method for computing the optimal values and policies for finite MDPs given complete knowledge of the MDP. The most popular DP algorithms are *policy iteration* and *value iteration*. In particular, *generalized policy iteration* (GPI) is an iterative combination of both, which allows to converge towards the optimal policy by iteratively refining the value function estimates and improving the policy based on the updated values.

Naturally, computing the optimal values can be challenging, because the computational complexity scales with the number of states and actions, also known as *curse of dimensionality*. Moreover, the transition model is not always known. To address the latter challenge, alternative model-free solutions are necessary, which do not require complete knowledge of the environment and force the learner to rely on visited transitions (samples from the transition model).

2.1.2 *Model-free RL*

Model-free methods can approach decision-making problems with or without *bootstrapping* value function estimates. The first class of solutions is called *temporal difference* (TD) learning, while the latter falls under the umbrella of *Monte Carlo* (MC) methods. MC methods estimate value functions and derive optimal policies by averaging the returns of sampled episodes (sample estimate of eq. 4 or 5), bypassing the need for a complete model of the environment. MC methods also introduce concepts such as on-policy and off-policy learning, which allow for greater flexibility and learning efficiency in reinforcement learning applications. On-policy learning focuses on evaluating and improving the policy currently being followed, while off-policy learning allows the agent to converge to the optimal policy using data collected from another policy.

In contrast to MC methods, TD learning can update the value functions before the episode ends. This key distinction enables TD learning to be more efficient in situations where episodes can be long or even continuous. Similarly to MC methods, TD learning also follows the idea of generalized policy iteration, updating the approximate policy and the value function after every step in the environment until they reach optimality. One of the most famous TD algorithms is *Q-learning* [20], which updates the action-value function in the following way:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

As mentioned before, Q learning is a type of *one-step* TD or TD(0), which updates the value function for the current state using the reward received and the value function estimate of the next state. However, TD learning can be extended to multi-step methods, which combine the advantages of both MC and TD approaches. One of the most popular multi-step TD methods is the *n-step* TD or TD(n), which uses n-step returns to update the value function. By varying the parameter n, the agent can control the degree of bootstrapping and choose between a more MC-like or TD-like learning process. Both MC and TD methods offer unique advantages in tackling reinforcement learning problems, with MC methods providing unbiased estimates of value functions and TD methods allowing for faster learning via bootstrapping. In particular, the combination of these methods with expressive function approximators such as deep neural networks, can lead to powerful *deep RL* algorithms that can tackle complex and high-dimensional problems. Deep RL algorithms can be used to learn policies directly from raw sensor inputs, such as images, and have been applied successfully in various domains, including game playing, robotics, and natural language processing [15, 17]. However, the use of function approximators also introduces new challenges, such as instability, non-convergence, and non-trivial generalization properties, which require careful algorithm design and training techniques, such as experience replay, target networks, and regularization.

2.1.2.1 *Policy Search*

Another approach to RL that is well-suited for high-dimensional or continuous action spaces is policy search. Policy search algorithms directly search for the optimal policy by optimizing a parameterized policy function $\pi(a|s;\theta) = \pi_\theta(a|s)$. This can be done using gradient-based optimization techniques or derivative-free methods such as evolutionary strategies or bayesian optimization. Policy search can be more sample-efficient than value-based methods, as it only needs to evaluate the policy function at each iteration, rather than computing a value function over the entire state space. However, policy search can also be computationally expensive and can suffer from local optima, especially in high-dimensional search spaces. Deep neural networks have also been applied to represent the policy function and have achieved state-of-the-art performance in various domains [21–26]. The quantity that policy search methods aim to optimize is the return induced by the policy:

$$J(\theta) = \mathbb{E}_{s_0 \sim \delta(s_0)} v_{\pi_\theta}(s_0)$$

All the methods that optimize this objective via gradient ascent are called *policy gradient* (PG) methods. The parameter updates take this form:

$$\theta_{t+1} = \theta_t + \alpha \widehat{\nabla J(\theta_t)}$$

where $\widehat{\nabla J}$ is an estimate of the gradient. The earliest description of a PG method is provided by Williams with the REINFORCE algorithm [27].

As vanilla PG methods often suffer from high variance in gradient estimates, which can lead to unstable and slow learning, several techniques have been developed to address these issues. One approach is to use a value function, or a critic, to reduce the variance in the policy gradient updates, often in the form of a *baseline* for the expected return. Famous policy search algorithms of this kind include Trust Region Policy Optimization (TRPO) [25], which uses trust regions to ensure that policy updates do not deviate too far from the current policy, and Proximal Policy Optimization (PPO)

[26], which uses a clipped surrogate objective to prevent the policy from changing too quickly. Both methods make use of value function estimates as part of the objective function.

On the other side, algorithms that use both policy gradients and value function estimates through bootstrapping, are at the basis of *actor-critic* (AC) methods. The main difference with PG methods is that the V and Q estimates are used as critics – i.e. to assess the value of an action – and not as baselines [14, 28]. Prominent examples of AC algorithms include the Deep Deterministic Policy Gradient (DDPG) [22] and Soft Actor-Critic (SAC) [23].

2.1.3 *Model-based RL*

In the previous section, we discussed model-free RL approaches that do not rely on explicit knowledge of the environment's dynamics. Model-based RL, on the other hand, focuses on learning an explicit model of the environment and using it for planning and decision-making. By utilizing a learned model, model-based RL methods can often achieve better sample efficiency than their model-free counterparts.

There are two main components to model-based RL (MBRL): learning the model and using the model for planning. The learning phase involves building a representation of the environment's dynamics, often referred to as the transition model. This model can be learned using supervised learning techniques, such as neural networks, Gaussian processes, or decision trees, to predict the next state (and often reward) given the current state and action. The quality of the learned model is crucial, as inaccuracies in the model can lead to suboptimal policies and poor performance.

Once the model is learned, the planning phase involves using the model to find the best course of action for the agent. There are several planning methods that can be used in conjunction with the learned model, such as tree search algorithms (e.g., Monte Carlo Tree Search [17]), optimization-based methods (e.g., model predictive control or RL-based [29, 30]), and sampling-based methods

(e.g., rapidly-exploring random trees [31], zero-order trajectory sampling [32]). These planning methods leverage the learned model to explore possible future trajectories and select actions that maximize the expected return.

A key challenge in MBRL is balancing the exploration and exploitation trade-off. The agent needs to explore the environment to improve its model, but it also needs to exploit its current knowledge to maximize rewards. Various exploration strategies have been proposed to address this issue, such as optimistic exploration, where the agent assumes that unexplored states hold higher rewards, or information-theoretic approaches, where the agent seeks to maximize the information gained from each interaction with the environment. In Chapter 5 we will consider this problem more in detail.

2.2 CLOSING THE LOOP WITH MODEL PREDICTIVE CONTROL

Model Predictive Control (MPC) is a well-established optimization-based control technique that involves using an explicit model of the system dynamics to predict the system's behavior over a finite horizon, in order to determine the control inputs that optimize a given objective function [9, 33], possibly subjected to hard constraints on state and control inputs. MPC relies on an accurate model of the system dynamics and iteratively solves an optimization problem at each time step to obtain an optimal control sequence. The first control input of this sequence is applied to the system, and the process is repeated at the next time step, taking into account the updated state of the system. Formally, the MPC problem can be defined as:

$$\begin{aligned} &\min_{a_0,\dots,a_{H-1}} \sum_{t=0}^{H-1} c(s_t, a_t) + c(s_H) \\ &\text{s. t. } s_{t+1} = f(s_t, a_t), \quad t = 0, \dots, H-1 \\ &a_{\min} \leq a_t \leq a_{\max}, \quad t = 0, \dots, H-1 \end{aligned} \tag{8}$$

where (s_t, a_t) are the state and control inputs[1] at time t, H is the prediction horizon, $c(s_t, a_t)$ is the cost function, and $f(s_t, a_t)$ represents the system dynamics. In this formulation, $a_{\min}$ and $a_{\max}$ represent the lower and upper bounds on the control inputs, respectively. These constraints ensure that the control inputs are within feasible limits, which is particularly important for real-world systems with actuator limitations or safety requirements. This *receding horizon* approach enables the controller to adapt to changing conditions and disturbances in real-time. MPC has been successfully applied in various fields, including robotics [34, 35], automotive [36, 37], and process control [8, 38]. However, standard MPC approaches often rely on having differentiable or linear models and quadratic cost functions, which can affect performance in highly uncertain or fast-changing environments [39]. By integrating these approaches with model-based learning algorithms, we can improve exploration, handle model uncertainty, and adapt to diverse environments. Some notable works in this area include PILCO [29], the information-theoretic exploration method by Williams [40], and neural network guided MPC [32, 41].

2.2.1 *Probabilistic Ensemble with Trajectory Sampling*

One notable approach that combines the strengths of MPC and MBRL is the Probabilistic Ensembles with Trajectory Sampling (PETS) algorithm [32]. PETS employs an ensemble of neural networks to model the dynamics of the environment, capturing both the mean and uncertainty of the model's predictions. The learned environment model can be used to generate a set of trajectories by applying various control sequences to the current state. The agent then evaluates these trajectories according to the expected cumulative reward and selects the one that maximizes it. This approach is conceptually similar to the optimization problem solved in MPC.

1 To maintain consistency with the rest of the text, we use the variables s and a to represent the system state and control action, respectively, instead of the traditional variables x and u commonly used in control theory.

These probabilistic models enable PETS to perform uncertainty-aware trajectory optimization, as the algorithm explicitly considers the uncertainty associated with model predictions when sampling and evaluating trajectories. By doing so, PETS can effectively balance exploration and exploitation, leading to improved performance and sample efficiency in challenging environments.

While PETS provides a principled approach for incorporating model uncertainty at the learning stage, in Chapter 5 we will present a novel strategy that takes into account all the sources of uncertainty also at the planning stage. In the next section, we will discuss another method for optimizing control tasks in MBRL that does not require explicit gradient information. This approach can be particularly useful for systems with non-differentiable dynamics or when the gradient computation is prohibitively expensive.

2.3 ZERO-ORDER OPTIMIZATION

With a solid understanding of reinforcement learning, model-based RL, and their connection to model predictive control and trajectory sampling, we can now delve into zero-order trajectory optimization. In the following section, we will present the fundamentals of this optimization technique and its application in the context of trajectory optimization.

Zero-order optimizers are a powerful class of optimization methods that solely rely on function evaluations, without the need for higher-order information from the gradient or the Hessian. These approaches are particularly suited for *black-box* optimization problems, as they only require iterative evaluations of the specified objective function. Moreover, they are not reliant on strong assumptions like continuity or differentiability of the objective function.

Well-known examples of this type of black-box optimizers are evolution strategies (ES), which belong to the family of evolutionary algorithms (EA). These methods are population-based metaheuristics that imitate some aspects of natural evolution mechanisms such as mutation, recombination, and selection. In particular, evolution

strategies are designed to be used for continuous black-box optimization and do not make use of crossover operators, which are mainly adopted in the context of genetic algorithms (also a subset of EA) which deal with sequences of discrete inputs [42]. Practical implementations of ES look like a "trial and error" process and follow a simple optimization procedure. Specifically, there is a *fitness* function $f : \mathbb{R}^n \to \mathbb{R}$ to be optimized, which can be evaluated for any input x. The goal is to find the best parameter $\theta^\star$ through iterative sampling from $p_\theta(x)$. This can be achieved in three main steps:

1. generating a population of individuals, or candidate solutions, $\{x_i\}$ sampled from $p_\theta(x)$;
2. evaluating each of them according to their fitness $f(x_i)$;
3. selecting the fittest individuals and evolving them into the population for the next iteration by updating the value of θ.

The advantage of ES is that the convergence is not affected by the properties of the black-box function $f(x)$, they are easily parallelizable, they have higher chances of escaping from local minima, and most of them grow linearly with the dimensionality of the search space. Furthermore, these algorithms are well-suited for problems with highly non-linear functions or discrete optimization problems – where gradients are unavailable, unreliable, or difficult to compute. Different types of ES can be often distinguished based on the parameterization of the distribution $p_\theta(x)$ (step 1) and on how the parameters are updated (step 3). Examples of renowned ES are:

RANDOM SEARCH The population is generated by sampling from a hypersphere surrounding the current position. Then, it iteratively moves to better positions in the search space. It was originally introduced by Anderson [44] and further analyzed by L. A. Rastrigin [43, 45]. The latter is homonymous with the Rastrigin function shown in Fig. 3, which is a non-convex highly multimodal function easily optimizable by ES.

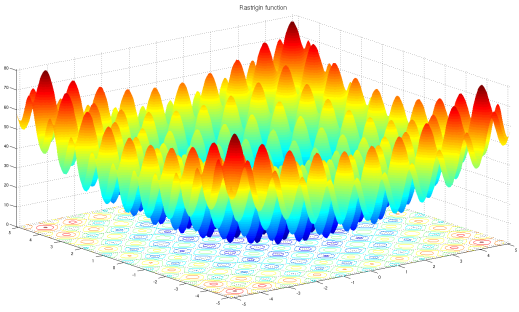


Figure 3: Visualization of the Rastrigin function [43]. It is used as a traditional benchmark for optimization algorithms such as Evolutionary Algorithms or higher-order methods.

CROSS-ENTROPY METHOD (CEM) The population distribution $p_\theta(x)$ is modeled as an n-dimensional isotropic Gaussian distribution [46], where $\theta = (\mu, \sigma)$ and $p_\theta(x) \sim \mathcal{N}(\mu, \sigma^2 \mathbb{I}) = \mu + \sigma\mathcal{N}(0, \mathbb{I})$, with μ and $\sigma \in \mathbb{R}^n$, respectively representing the mean and the standard deviation. The optimization cost grows as $O(n)$ since we are only estimating a diagonal covariance matrix.

COVARIANCE MATRIX ADAPTATION (CMA-ES) The population distribution is still a multivariate Gaussian, but a low-rank approximation of the full covariance matrix is estimated [47].

NATURAL EVOLUTION STRATEGIES (NES) In this case, the selection process does not consider only a subset of the candidate solutions but the expected value over the entire population. Moreover, the parameters are updated according to an estimation of the natural gradient that uses the Fisher Information matrix [48, 49].

In the next section, we will focus the discussion on the Cross-Entropy Method and how it has been successfully used in the context of model-predictive control.

2.3.1 *The Cross-Entropy Method*

The cross-entropy method (CEM) is a derivative-free optimization technique that was originally introduced in Rubinstein [46] 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/-fitness 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 $\text{diag}(\sigma^2)$, 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.

In the MPC setting, CEM is used at every timestep to optimize an h-step planning problem on the action sequences. In Alg. 1 we show the pseudocode of vanilla CEM as a trajectory optimizer for model-predictive control. The parameters μ_t, σ_t now represent matrices in $\mathbb{R}^{d \times h}$, with d dimensionality of the action space. In addition to the standard iterations that are part of the CEM algorithm - here called *CEM-iterations* (line 8–12 in Alg. 1) - we will also have an outer loop which marks the temporal progression in the environment by executing one action. Naturally, the next step considers the planning problem one timestep later.

It is important to notice that the action sequences proposed by the $n \times h$ Gaussian distributions are sampled independently along n and h, and that the covariance matrix is still diagonal. In other words, each action vector inside the planning sequence has the same probability of being sampled, whether we are at the beginning or the end of the sequence. The corresponding random process is

called *white noise*. This type of noise is generally used to describe random disturbances with a very small correlation period, e.g. thermal motion of electrons.

Algorithm 1: Cross-Entropy Method (CEM) for Trajectory Optimization

Parameters:
  N: number of samples; K: size of elite-set; h: horizon;
  σ_{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 σ_{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 μ_t

Limitations

In the last section, we have seen that CEM can be used to generate action sequences for model-predictive control. However, the random process behind action-plan sampling is similar to the one generating thermal noise with zero correlations in time. Since action plans do not generally behave like the elementary vibrations of electrons, using i.i.d. Gaussians is not optimal for continuous control tasks.

Additionally, due to the iterative nature of population-based optimizers, the total number of evaluated samples becomes extensive which can lead to a slow run time depending on the computational cost of f.

Moreover, CEM is a stateless optimizer, meaning that it has to be queried again whenever we want the optimal action sequence from a different input state, and it does not keep memory of previously found solutions. Therefore, even if we have the benefits of a global search process that uses population-based metaheuristics, the final outcome is very local. The opposite happens with global policies parameterized, e.g., by neural networks and trained following local descent directions provided by the stochastic gradient.

In Chapter 3 and 4 we propose two possible solutions to these problems, which can be combined and used for general control tasks. In particular, Chapter 3 focuses on how to improve the CEM optimizer itself, while in Chapter 4 we propose to combine zero order optimization and policy learning to obtain a global solution. To this purpose, the next sections will be devoted to explaining the challenges of learning global policies.

2.4 SUPERVISED ACTORS

In Section 2.1 we have seen how policies can be learned through value and policy iteration, where the main learning signal is determined by the Bellman equations (standard model-free RL), or also by direct policy search, where no value function is strictly needed and the learner only takes gradient steps in the policy space.

The most successful RL algorithms are variations of the aforementioned methods, and they have been able to solve many complex continuous control tasks from states (e.g. joint positions and velocities) as well as pixels (e.g. camera images) [15, 17, 22, 23, 50]

However, both off-policy and on-policy model-free RL algorithms suffer from high sensitivity to their hyperparameters, need a lot of finetuning, have poor sample efficiency, have very slow convergence, and need strong exploration methods to tackle sparse rewards and find global optima. Moreover, they still have to deal with the typical challenges associated with RL such as the exploitation-exploration trade-off, delayed credit assignment, and generalization out-of-distribution.

Nevertheless, other options are possible outside the realm of reinforcement learning, if we assume that the exploration problem has been solved and we can already access high-quality training data. For example, learning a policy can be also done through supervised learning. In this case, the policy is the *student* while the labeled data comes from a *teacher* or expert. This approach is also more common in the context of offline RL, which can be reframed as a supervised learning problem [51–53], where the end result is a policy, usually conditioned on goal states [54, 55] or the reward function [51, 56].

One popular form of supervised learning in the context of sequential decision making is Imitation Learning (IL). In IL, the student policy has access to a set of expert demonstrations

$$\tau_i = (s_0, a_0, s_1, a_1, \dots)_i,$$

and it can directly learn how to imitate the teacher's actions e.g. by maximum likelihood estimation, also called Behavioral Cloning (BC):

$$\arg\max_\theta \mathbb{E}_{(s,a,g)\sim\mathcal{D}}[\log \pi_\theta(a|s)]$$

where $\mathcal{D} = \{\tau_1, \tau_2, \dots\}$, and $(s, a, g) \in \mathcal{S}, \mathcal{A}$ respectively.

Alternatively, it can indirectly learn a policy by first extracting the reward function, which is called Inverse Reinforcement Learning (IRL) [57, 58].

In some cases, the policy has access to the expert at training time and it can query its output actions for hindsight correction for example [59, 60]. This is especially useful since a policy learned via BC can often lead the agent outside of the state distribution of the training data. This problem is called *covariate shift* or distribution shift [61], which we will discuss more extensively in Chapter 4.

3

THE IMPROVED CROSS-ENTROPY METHOD

Summary

In this chapter we will present our improved version of the Cross-Entropy Method. As explained previously, zero-order optimizers 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. Therefore, we propose an improved variant 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.

3.1 INTRODUCTION

Recent work in model-based reinforcement learning (MBRL) for high-dimensional systems employs population-based algorithms as trajectory optimizers [32, 41, 62–64]. Sampling-based methods have also been used in the control community in scenarios when the cost function is not differentiable [65]. 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) [46] was introduced for the first time in the 1990s as a stochastic, derivative-free, global optimization technique, but

This chapter is based on the paper *"Sample-efficient zero-order trajectory optimization for real-time planning"*, Cristina Pinneri, Shambhuraj Sawant, Sebastian Blaes, Jan Achterhold, Jörg Stückler, Michal Rolínek, Georg Martius [10].

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 [32, 62, 63].

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 zero 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.

In this chapter 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 [66] 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× 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 [63] (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 zero order optimizers. The source code can be found at `https://github.com/martius-lab/iCEM.git`.

3.1.1 *CEM for model-predictive control: CEM$_{MPC}$*

In Chapter 2 we discussed how zero order optimizers can be used in the context of model predictive control. In particular, there are some common adaptations to employ when using the Cross-Entropy Method. A typical modification [32, 62] shift-initializes the current mean μ_t of the CEM distribution from the previously optimized μ_{t-1} (see B.4.1 in the Appendix). Another standard modification is to use a momentum term [67] in the refitting of the distributions between the CEM-iterations (line 12 in Alg. 1). 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. In the rest of this chapter, we will refer to this variant as CEM$_{\text{MPC}}$.

In Chua et al. [32] (PETS method), in addition to the standard improvements discussed above, the sampling distribution was truncated. 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$_{\text{PETS}}$.

3.2 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. 3.3.3 and the sensitivity analysis in Sec. B.2.2.

3.2.1 *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:

$$\frac{\mathrm{d}}{\mathrm{d}t}x(t) = a(t) \tag{9}$$

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 [68]. 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 [69].

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. B.5 in the Appendix. 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:

$$\text{PSD}_a(f) \propto \frac{1}{f^\beta} \tag{10}$$

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 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.

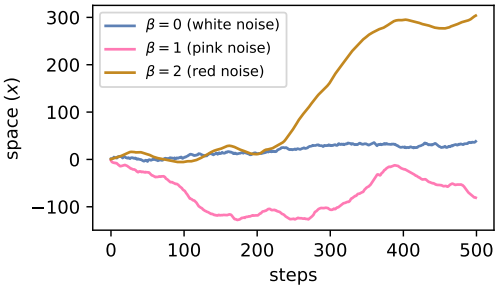
How does the trajectory $x(t)$ of Eq. 9 look when we use colored-noise actions? Figure 4a shows three examples with the same action variance – the larger the β, the larger the coherence and the larger distances can be reached. This can be formalized by computing the PSD_x of the state-space trajectory (x). Using Eq. 9 and Eq. 10 we find:

$$\text{PSD}_x(f) = \|\mathcal{F}[x(t)](f)\|^2 \overset{(*)}{=} \frac{\|\mathcal{F}[a(t)](f)\|^2}{4\pi^2 f^2} = \frac{\text{PSD}_a(f)}{4\pi^2 f^2} \propto \frac{1}{f^{\beta+2}} \tag{11}$$

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. 4a.

Let us consider now the effect of colored-noise in a robotic setting: the Humanoid Standup task, see Fig. 8, included in the OpenAI Gym [70] environments. Figure 4b 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. 10. 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

(a) 1D random walks with colored noise



(b) power spectral density of action

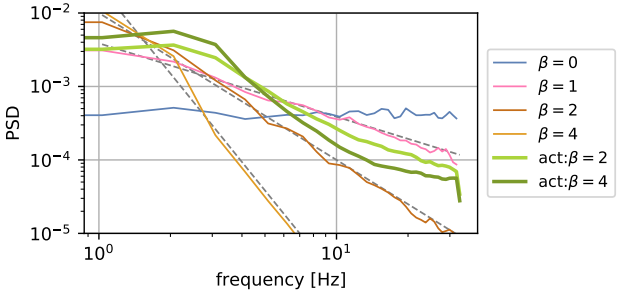


Figure 4: Colored random noise. (a) random walks with colored noise of different temporal structure. (b) power spectrum of colored-random action sequences for different β and of the chosen (and successful) action-sequences of iCEM generated by differently colored search noise (act:$\beta = 2$ and 4)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

proposed iCEM method (green lines in Fig. 4b) 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 Appendix B.2.1.

We introduce the colored-noise in CEM as a function of β which creates correlated action sequences with a PSD as in (10). For sampling, we use the efficient implementation of [71] based on Fast Fourier Transform [72]. 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 (10), we have to apply the following transformation to the original white noise actions $a(t)$:

$$\bar{a}(t) = \mathcal{F}^{-1}\left[\frac{1}{f^{\beta/2}}\mathcal{F}[a(t)]\right] \quad \text{gives} \tag{12}$$

$$\text{PSD}_{\bar{a}}(f) = \left\| \frac{1}{f^{\beta/2}}\mathcal{F}[a(t)](f) \right\|^2 = \frac{1}{f^{\beta}}\text{PSD}_a(f) \propto \frac{1}{f^{\beta}} \tag{13}$$

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.

3.2.2 *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. [32], this amounts to an average of $\sim$ 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.

3.2.3 *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. B.4.3.

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. 4. Hyper-parameters are given in Sec. B.2 in the appendix. Except β we use the same parameters for all settings. The planning horizon is 30.

3.3 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 [63] framework (details in Appendix B.2.3).

3.3.1 *Environments*

First, we consider the following three challenging the environments contained in OpenAI Gym [70]:

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. 8. 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 ran-

Algorithm 1: Proposed iCEM algorithm. Color brown is iCEM and blue is CEM_{MPC} and iCEM.

Parameters:
  N: number of samples; h: planning horizon; d: action dimension; K: size of elite-set; β: colored-noise exponent
  CEM-iterations: number of iterations; γ: reduction factor of samples; σ_{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 μ_{t-1} (and repeat last time-step) // see App. B.4.1
  $\sigma_t \leftarrow$ constant vector in $\mathbb{R}^{d \times h}$ with values σ_{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_{MPC}
    samples $\leftarrow N_i$ samples from $\text{clip}(\mu_t + \mathcal{C}^\beta(d,h) \odot \sigma_t^2)$ // only iCEM, see Eq. 12
    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 μ_t // only CEM and CEM_{MPC}
  execute first action of best elite sequence // only iCEM

domly 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.

Furthermore, we test iCEM on three environments from the DAPG project [73]. 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 the handle. The reward (as in [73]) 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 rewards 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 from the palm to the handle, so without opening the door there is no reward.

RELOCATE In the relocate environment the task, see Fig. 8, 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.

3.3.2 *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,

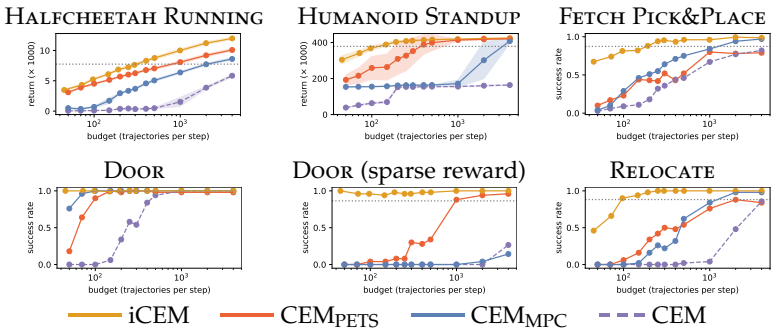


Figure 5: Performance dependence on the planning budget. Notice the log-scale on the x-axis.

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 5 presents the performance of iCEM, CEM_{MPC}, CEM_{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 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 9.

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.

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. 5). The last column is the average improvement over the best baseline in the given budget interval.

	90% baseline@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%

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 [63]. 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. 6, 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 Appendix B.

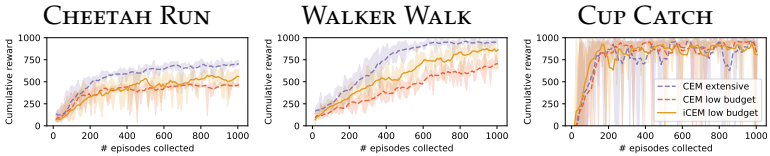


Figure 6: PlaNet performance using an extensive CEM variant (budget 10000) and two low-budget variants of CEM and iCEM (budget 366). Shown is the mean and min/max-band cumulative reward (three independent restarts) with average-smoothing over 50 episodes. iCEM outperforms the low-budget baseline on Cheetah Run and Walker Walk, and performs similarly on Cup Catch.

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.

3.3.3 *Ablation study*

To study the impact of each of our improvement individually, we conducted ablations of iCEM (orange bars in Fig. 7) and additions to CEM_{MPC} (blue bars in Fig. 7) for some environments and budgets, see Sec. B.3 for all combinations and more details.

Some components have bigger individual impact than others, e.g. using *colored* 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 becomes more relevant in the low-budget regime.

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 × episode length). *: Xeon® Gold 6154 CPU @ 3.00GHz, and **: Xeon® Gold 5220, NVidia® Quadro RTX 6000.

		Budget (trajectories per step)				
Envs	Threads	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

3.4 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) [74] regained popularity for their successful use in RL [49, 75, 76], making population-based methods an attractive alternative to policy gradient or as a supportive guidance [77, 78]. Sampling-based techniques have also been used for model-predictive control (MPC), like model predictive path integral (MPPI) control [64], with applications to aggressive driving by using a GPU [79].

In particular, the Cross-Entropy Method (CEM) [46, 80, 81], thoroughly analyzed in [82], has been used both for direct policy optimization [83] and planning with learned models [32, 62], to improve the performance of rapidly exploring random trees [84], with

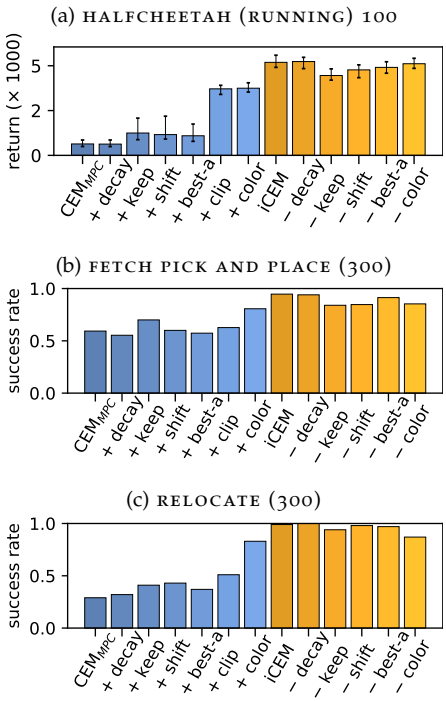


Figure 7: Ablation studies. Blue bars show CEM_{MPC} with each improvement added separately. Yellow bars show iCEM with each features removed separately. Feature names are listed in Sec. 3.2.

successful applications in many fields of science. Examples include visual tracking [85], bioinformatics [86], and network reliability [87].

In Duan et al. [88] it is reported that CEM has similar performance to standard trajectory optimizers like iLQG, as well as better performance over the more sophisticated Covariance Matrix Adaptation ES (CMA-ES) [47], 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 this chapter was to show that CEM can be functional for real-time decision making. Recent works in this direction propose a differentiable version of CEM [89] or to jointly use model gradients together with the CEM search [78]. Wang and Ba [62] 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.

3.5 CONCLUSION

In this chapter, we introduced iCEM: a sample-efficient improvement of CEM intended for real-time planning. Most notably, we introduced temporally correlated action sampling and memory for previous trajectories. Sampling actions from a colored-noise distribution introduces long-range dependencies that we would not have with "thin-tailed" distributions, such as the Gaussian distribution. The proposed 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 managed to enter the real-time regime, as shown in the experiments with learned models. In the next chapter, we will see how to extract a global reactive policy from the iCEM optimizer.

4

ADAPTIVE POLICY EXTRACTION

Summary

Zero-order trajectory optimizers like iCEM have so far shown strong performance for solving high-dimensional and continuous robotics tasks, however, their output is inherently local and they have zero generalization performance. In this chapter, we propose a technique to jointly optimize the trajectory with iCEM and distill a policy, which is essential for fast execution in real robotic systems. Our method builds upon standard approaches, like guidance cost and dataset aggregation, and introduces a novel adaptive factor which prevents the optimizer from collapsing to the learner's behavior at the beginning of the training. The extracted policies reach unprecedented performance on challenging tasks like making a humanoid stand up and opening a door without reward shaping.

4.1 INTRODUCTION

The general purpose of model-based and model-free reinforcement learning (RL) is to optimize a trajectory or find a policy that is fast and accurate enough to be deployed on real robotic systems.

Policies optimized by model-free RL algorithms achieve outstanding results for many challenging domains [90, 91], however, in order to converge to the final performance, they require a large number of interactions with the environment and can hardly be used on real robots, which have a limited lifespan. Moreover, real robotic systems are high-dimensional and have a highly non-convex opti-

This chapter is based on the paper *"Extracting strong policies for robotics tasks from zero-order trajectory optimizers"*, Cristina Pinneri*, Shambhuraj Sawant*, Sebastian Blaes, Georg Martius [11].

mization landscape, which makes policy gradient methods prone to converge to locally optimal solutions. In addition, model-free RL methods only gather task-specific information, which inherently limits their generalization performance to new situations.

On the other hand, recent advances in model-based RL show that it is possible to match model-free performance by learning uncertainty-aware system dynamics [29, 32, 92]. The learned model can then be used within a model-predictive control framework for trajectory optimization. Zero-order optimizers are gaining a lot of traction in the model-based RL community [32, 62, 64] since they can be used with any choice of model and cost function, and can be surprisingly effective in finding high-performance solutions [93] (close to a global optimum) in contrast to their gradient-based counterparts, which are often highly dependent on hyperparameter tuning [94]. One of the most popular optimizers is the Cross-Entropy Method (CEM), originally introduced in the 90s by Rubinstein [46].

Despite their achievements, using zero-order methods for generating action sequences is time consuming in complex high-dimensional environments, due to the extensive sampling, making it hard to deploy them for real-time applications.

Extracting a policy from powerful zero-order optimizers like CEM would bridge the gap between model-based RL in simulation and real-time robotics. As of today, this is still an open challenge [62].

We analyze this issue and showcase several approaches for policy extraction from CEM. In particular, we will use the sample-efficient modification of CEM (iCEM) presented in the previous chapter [93]. Throughout this chapter, we will call these optimizers "experts" as they provide demonstration trajectories. To isolate the problem of bringing the policy's performance close to the expert's one, we consider the true simulation dynamics as our forward model.

Our contributions can be summarized as follows:

- pinpointing the issues that arise when trying to distill a policy from a multimodal, stochastic teacher;

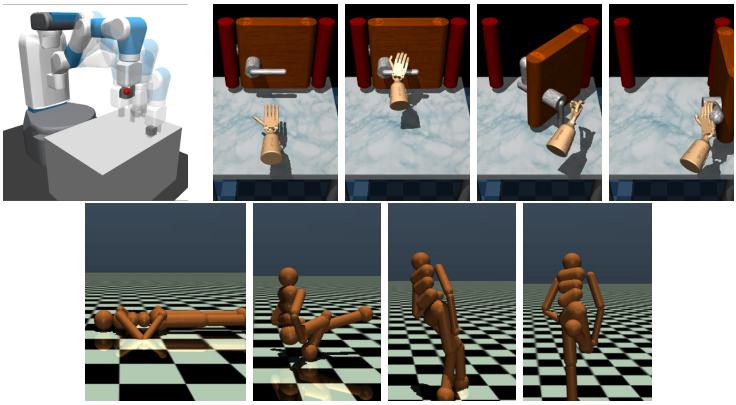


Figure 8: Environments and exemplary behaviors of the learned policy using APEX. From left to right: FETCH PICK&PLACE (sparse reward), DOOR (sparse reward), and HUMANOID STANDUP.

- introducing APEX, an Adaptive Policy EXtraction procedure that integrates iCEM with DAgger and a novel adaptive variant of Guided Policy Search;
- our specific integration of methods produces an improving adaptive teacher, with higher performance than the original iCEM optimizer;
- obtaining strong policies for hard robotic tasks in simulation (HUMANOID STANDUP, FETCH PICK&PLACE, DOOR), where model-free policies would usually just converge to local optima.

Videos showing the performance of the extracted policies and other information can be found at https://martius-lab.github.io/APEX.

4.2 RELATED WORK

Our objective is to extract high-performing policies from CEM experts that can operate with a few planning samples to make iterative learning fast. Other kinds of zero-order optimizers have been used to generate control sequences [64, 95] but they still have to evaluate thousands of trajectories for each time step. Even simple random shooting has been used as a trajectory optimizer to bootstrap a model-free policy [41].

To train policies from optimal control solutions, it was shown that the expert optimizers need to be guided towards the learning policy – known as guided policy search (GPS) [96, 97]. In our work, the expert does not come from optimal control but is the stochastic iCEM optimizer, which we will also refer to as *teacher*. We apply GPS in a model model-predictive control setting, as done in Levine and Koltun [96], Mordatch and Todorov [98], Mordatch et al. [99], Zhang et al. [100], Kahn et al. [101], and Sun et al. [102] using local trajectory optimization to generate a dataset for training a global policy through imitation learning. These approaches alternate between training a policy and creating new data with a model-based supervisor guided towards the learner, which was formalized in Sun et al. [102]. Stochastic experts require particular guidance strategies, such as an **adaptive cost** formulation that we propose here, together with expert warm-starting via distribution initialization and additional samples from the policy. A simple form of warm-starting was already done in Wang and Ba [62].

Recently, approaches like simple point-to-point supervised training such as Behavioral Cloning (BC), or Generative Adversarial Network training (GAN) have been explored [62] for policy distillation from CEM, but only largely sub-optimal policies could be extracted. When the policy is used alone at test time and not in combination with the MPC-CEM optimizer, its performance drops significantly, becoming almost random for some environments. We argue that the reason behind the difficulty in distilling a policy from CEM expert

data is the multimodality of the CEM solution space and its inherent stochasticity due to the sampling, which we address below.

Another possibility to train higher-performance policies from experts is DAgger [59], which is an on-policy method, asking the expert to relabel/correct policy actions. Nevertheless, as we will show in the following sections, DAgger alone is not sufficient to extract high-performing policies in our setting. To solve this problem, we use a guiding cost in combination with DAgger. A combination of GPS and DAgger-style relabeling was proposed in PLATO [101] to create, however, unbiased training data from iLQG experts. Since unguided DAgger is not appropriate with CEM, PLATO is not successful in our setting either. The components of our algorithm will be explained in the following sections.

4.3 METHODS

Trajectory optimization aims to find a suitable action sequence $\vec{a}_t = (a_t, a_{t+1}, \ldots, a_{t+h})$ of horizon h that minimizes a cost function $f(\vec{a}_t, s_t)$, where s_t is the current state of the system. i. e.

$$\vec{a}_t^{\star} \leftarrow \arg\min_{\vec{a}_t} f(\vec{a}_t, s_t)\,. \tag{14}$$

The cost function f encodes the task. Optimal control is obtained if f is the trajectory cost until step h plus the cost-to-go under the optimal policy for an infinite-horizon problem, evaluated at the last state in the finite horizon trajectory. Typically, the cost-to-go is replaced by a proxy cost such as the distance to a target. In our case, the cost f is given by the sum of negative environment rewards up to the planning horizon.

To optimize Eq. 14, we use the improved CEM algorithm (iCEM) [10] introduced in Chapter 3. We remind that this method makes use of *colored-noise* and *memory*, and it generates correlated action sequences with a non-flat frequency spectrum, differently from the flat Gaussian noise of CEM. This, together with the memory addition, results in one order-of-magnitude sample reduction.

4.3.1 *Using a policy to inform the optimization: iCEM$_\pi$*

Unguided search for action sequences can be hard, in particular, if the cost function has large flat regions (sparse rewards). Since we aim at extracting a policy $\pi(s)$, we can expect that after some examples the policy can be used to guide the search into the right region. Thus, we warm-start the mean μ of the iCEM Gaussian distribution with the policy actions. In particular, at time $t = 0$ where no prior information exists, the mean is initialized from rolling out the trajectory with the policy until the planning horizon: $\mu \leftarrow (\pi(s_0), \pi(s_1), \dots \pi(s_h))$. Whenever the action for a new step is computed, iCEM uses shift initialization of the mean, and only the mean-action at the end of the horizon is initialized from the policy. Since iCEM is sample-based we also provide it with samples from the policy directly. More concretely, the actions performed by the policy are added in the last iteration if iCEM. The policy-informed iCEM algorithm is called iCEM$_\pi$ and is shown in Alg. 4.

4.3.2 *Off- and On-Policy Imitation Learning*

How difficult is it to clone a policy from iCEM$_\pi$? Using a basic off-policy method like Behavioral Cloning (BC), which simply does point-to-point L2 loss minimization, does not work out-of-the-box. This is because during test-time, the policy visits state-space regions for which it never received any expert feedback, and it consequently fails. This problem is also known as "covariate shift". The performance gap (difference in cost to go) between the expert and the policy scales quadratically in time due to the covariate shift [59].

A classical way to address this problem is with on-policy imitation learning. DAgger (Dataset Aggregation) by Ross, Gordon, and Bagnell [60] is the most popular example. DAgger works by iteratively rolling out the current policy, querying the expert for the correct actions (also called relabeling) on the states visited by the policy, and training the policy on the aggregate dataset across all

iterations. The covariate shift is alleviated by populating the data with states visited by previous iterations of the policy.

Issues of stochastic teachers

Nevertheless, some problems arise when considering population-based optimizers as experts, both for BC alone or in combination with DAgger. The optimized action sequence of iCEM might converge to different solutions every time it is asked to relabel the policy actions. Naturally, many actions can result in an equally high-performing solution. As an example: in the Fetch Pick & Place task, well before grasping the box, the action of opening or closing the gripper can be arbitrary, as long as the action of opening the gripper is executed on time. To illustrate this problem, we analyze the variance of the relabeled actions produced by the same expert on a fixed state sequence. The variance is high, spanning over a third of the action-space, as can be seen in Fig. 9a.

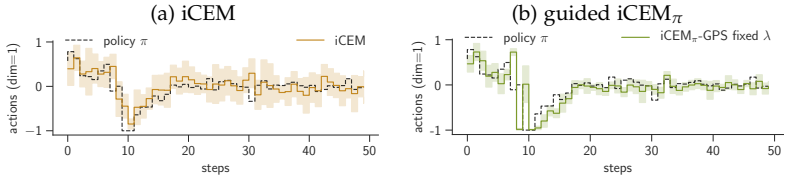


Figure 9: Variance of DAgger actions when relabeling 10 times the same trajectory in case of unguided iCEM (a) vs. guided iCEM$_\pi$ (b), for the Fetch Pick&Place task. The action variance of iCEM is considerably higher than the one of iCEM$_\pi$-GPS guided by the shown policy (std-dev=0.30 vs 0.13). The policy π is trained from a single expert rollout.

4.3.3 *Guided Policy Search*

To lift some of the aforementioned problems, the expert can be guided by the current policy. For high-dimensional control tasks, in

addition to the high variance (Fig. 9a), zero-order optimizers may also find multiple different solutions, leading to multi-modal training data. This prohibits successful policy extraction through supervised learning. To address these issues, the cost function f in Eq. 14 becomes the main cost J (task) plus a guidance cost to bias the optimizer's solution towards the policy:

$$\vec{a}_t^{\star} \leftarrow \arg\min_{\vec{a}_t} J_t(\vec{a}_t, s_t) + \lambda D_{\mathrm{KL}}(\pi_\theta || \vec{a}_t), \tag{15}$$

which is also known as Guided Policy Search (GPS) [96]. When competing solutions arise, the one closer to the policy is preferred, tackling in this way the problem of multimodality and creating much more consistent training data. As a remark, in our experiments, we use deterministic policies which reduces the KL-divergence to a squared difference between the actions.

In addition to Eq. 15, we further guide iCEM by warm-starting it from the policy, as implemented in iCEM$_\pi$ (Alg. 4). As a consequence, both together lead to a reduced action variance, as seen in Fig. 9b. For an illustrative state sequence, the standard deviation of expert actions drops from $\sigma = 0.30$ without guidance to $\sigma = 0.13$.

The data for training the policy comes from the expert interacting with the environment, that we call *expert rollouts*. When combining GPS with DAgger we use the same cost-function (Eq. 15) for the normal expert rollout and for asking the expert to perform action relabeling. This contrasts with PLATO [101], designed for iLQG experts, where the guidance cost is only used for the expert rollouts. Also, in PLATO the policy is not executed to collect new data, but relabeling of guided expert data is done using the unmodified cost to collect optimal training data. For zero-order optimizers, unguided solutions are not helpful as shown above. Without relabeling the actual policy rollouts, we found PLATO not to work in our setting.

The hyperparameter λ in Eq. 15 is difficult to choose. It might be adapted according to learning progress or other heuristics. However, when using the guided cost in iCEM other considerations have to be made which lead to a simple adaptation scheme introduced in the next section.

4.3.4 *Adaptive auxiliary cost weighting*

The purpose of λ is to trade-off the potential loss in cost by staying close to the policy. Generalizing Eq. 15, we can phrase the cost function f in Eq. 14 as the sum of the main cost J and a set of auxiliary costs C_j^{aux}:

$$\vec{a}_t^{\star} \leftarrow \arg\min_{\vec{a}_t} J_t(\vec{a}_t, s_t) + \sum_j \lambda_j C_j^{\text{aux}}(\vec{a}_t, s_t) \tag{16}$$

Examples of auxiliary costs are the guidance cost as in Eq. 15 and the action norm $\|\vec{a}\|$ to prefer small action magnitudes. We follow the philosophy that the auxiliary costs are subordinate and should only bias the solutions without causing a large performance drop. This leads to the idea that the auxiliary costs should never dominate the optimization. Since the main costs might vary by orders of magnitude, for instance, in sparse reward settings, we opt for a formulation where the auxiliary costs can maximally lead to a fixed fractional loss in performance.

As a motivating example, let us consider hard tasks that present flat regions in the cost function: in this case, a non-zero λ can lead to a failure of guided iCEM to find any good solution. How is that possible? Let us consider the Fetch Pick & Place task as shown in Fig. 8. When none of the sampled action-sequences moves the box (which is quite likely in the first iCEM iteration) then all sequences are equivalent under the main cost. Elites (Alg. 4 line 17) are only selected based on the auxiliary costs resulting in a quick reduction in sampling variance and preventing a good solution to be ever found. To illustrate this phenomenon, Fig. 12 displays the sampling distribution of iCEM$_\pi$ throughout the optimization iterations (line 9 in Alg. 4). Using a fixed λ (iCEM$_\pi$-fixed λ) typically converges to the policy action which is often unsuccessful in a new situation or when the policy is still weak.

We propose the following adaptation of λ_j:

$$\lambda_j = c_j \frac{\mathcal{R}(J)}{\mathcal{R}(C_j^{\text{aux}}) + \epsilon} \tag{17}$$

where c_j is the new hyperparameter, $\epsilon > 0$ is a small regularization to avoid amplifying tiny auxiliary costs, and $\mathcal{R}(C)$ represents the cost-range of C in the elite set:

$$\mathcal{R}(C) = \max_{\text{elite-set}} C - \min_{\text{elite-set}} C\,. \tag{18}$$

This formulation ensures that λ_i is zero if the main cost J cannot be improved by the preliminary elite-set (as in the example above). When an actual improvement is possible and the cost landscape is not flat ($R(J) > 0$), the auxiliary cost influences up to a fixed fraction (c_j) of the cost-range. The plot of the adaptive lambda is shown in Fig. 10. Thus, the sampling distribution for the adaptive

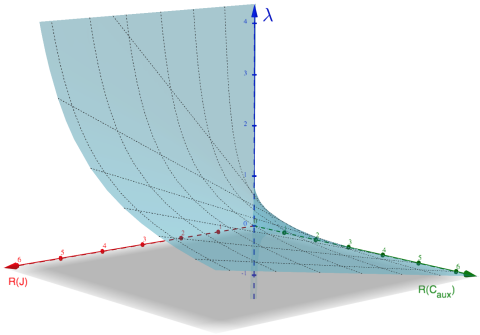


Figure 10: Illustration of the adaptive λ_j parameter as a function of the main cost range $R(J)$ and auxiliary cost range $R(C_j^{\text{aux}})$.

case will converge only if a good solution is found, as visualized in Fig. 12a (iCEM$_\pi$-GPS adaptive λ) for one auxiliary cost being the KL-term as in Eq. 15. The new hyperparameter c is easier to tune than λ. We use the same c for all experiments, although the main costs are 3 orders of magnitude apart.

Another aspect of the ratio in Eq. 17 is that, when none of the elite sequences is close to the policy network, then the denominator of Eq. 17 is small, which heavily steers the sampling towards the policy, however, *only* if the numerator is not zero ($R(J) \neq 0$), i. e.a reward signal from the environment is detected. In Fig. 11, we report

how the adaptive λ parameter changes over time for a weak policy (untrained neural network, low performance) and a medium policy (partially trained, medium performance), in the FETCH PICK&PLACE environment. In both runs, we fixed the goal and target locations. In the case of a weak policy, no successful solution is found in the first few time steps, meaning the min cost is the same as the max cost among elites. Hence, λ is zero, allowing the optimizer to freely explore without being hampered by the weak policy. In the case of a better policy, a solution is found early on, possibly because of the optimizer being guided by the policy. Thus λ becomes non-zero in the second time step already.

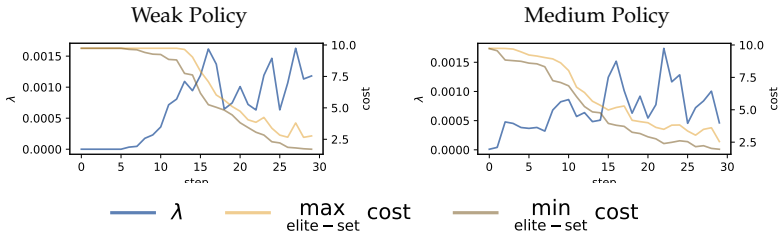


Figure 11: Evolution of the adaptive λ parameter during planning. Left for a weak, right for a medium policy. The light and dark orange curves show the original min/max cost (J) among the elites. The blue curves show how lambda changed due to the differences in the original costs.

Teacher improvement

Another effect of adaptive guidance in combination with iCEM$_\pi$ is the improvement of the expert/teacher, which is not common nor expected in standard imitation learning. The increased success-rate is shown in Fig. 12b. As we can see from the plot, the guidance alone is not sufficient to reach the original iCEM expert performance: when the auxiliary cost becomes adaptive, the iCEM benefits from the policy.

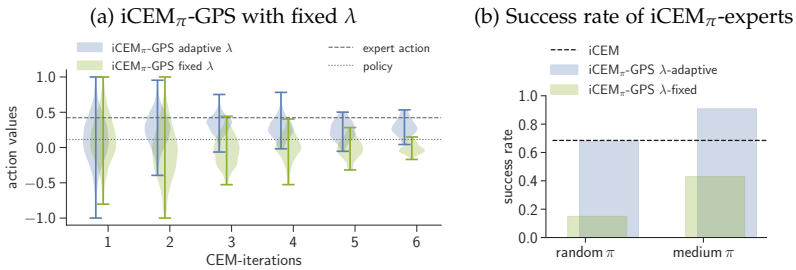


Figure 12: Effect of adaptive λ throughout iCEM$_\pi$ iterations and success rate on the FETCH PICK&PLACE task. **(a)** The action sampling distribution is shown over the iCEM-iterations (at a predefined time-step) and one of the 4 action-dimensions when guiding with a weak policy. Dashed lines indicate the action of the policy and of a high compute-budget iCEM expert. Fixed λ shifts the distribution too early, resulting in a collapse to the policy behavior and failure to find a good solution. **(b)** Average success rate of iCEM$_\pi$ expert (low compute-budget with 45 samples) over 800 episodes.

4.3.5 *Putting the pieces together: APEX*

Our method, named Adaptive Policy EXtraction (APEX), uses adaptive policy-guided iCEM$_\pi$ and DAgger using the same expert to create data for successful training of policies in an iterative fashion.

The main steps are: create data from the guided expert, update the policy, rollout the policy and relabel the actions using the same expert (DAgger), add the relabeled data to the dataset, and update the policy. This iterates until the desired performance is reached. More details are found in the pseudo-code in Alg. 3.

4.4 RESULTS

Can strong policies be obtained from model-based planning and imitation learning? By considering high-dimensional challenging robotics environments, some of them with sparse rewards, we test

APEX and find that the answer is yes. In some cases, we considerably improve the state-of-the-art currently held by model-free methods.

We perform our experiments on a selection of 4 environments, listed below, which use the MuJoCo [66] physics engine:

HUMANOID STANDUP (OpenAI Gym [70] v2) A humanoid robot is initialized in a laying position. The goal is to stand-up without falling, i. e.reaching as high as possible with the head. We use a task horizon of 500.

FETCH PICK&PLACE (SPARSE REWARD) (OpenAI Gym v1) A robotic manipulator has to move a box, randomly placed on a table, to a randomly selected target location. The reward is only the negative Euclidean distance between box and target location, so without moving the box there is no reward.

DOOR (SPARSE REWARD) (DAPG project [73]) A simulated 24 degrees of freedom ShadowHand has to open a door with a handle. The reward is the sum of door opening, a quadratic penalty on the velocities, and a bounty for opening the door (in contrast to [73] where a guidance of the hand to the handle was used).

HALFCHEETAH RUNNING (OpenAI 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 by heavily penalizing large absolute angles of the root joint. In the standard setting, numeric instabilities in the simulator are exploited by iCEM.

In Fig. 13, we compare APEX against several imitation learning baselines: Behavioral Cloning (BC) (iCEM BC), DAgger (iCEM DAgger), and BC from iCEM$_\pi$ with guidance cost (fixed λ) and warm-starting (iCEM$_\pi$-GPS). For reference, we also provide the performance of SAC[1] [23] as a model-free RL baseline, to get an idea of the difficulty of the learning tasks. Note that we use partially sparse re-

1 We took the implementation from https://github.com/vitchyr/rlkit

wards settings in FETCH PICK&PLACE (only box to goal reward) and DOOR (sparse reward) (no hand-to-handle reward) which makes them particularly challenging. However, for FETCH PICK&PLACE, a specialized method for goal-reaching tasks using hindsight relabeling (DDPG+HER) [103] can solve the task. In HUMANOID STANDUP the performance of SAC marks the behavior of just sitting. This is a local optima and is hard to escape for gradient-based methods. APEX manages to stand up but cannot balance for long, presumably because there are many ways the robot can fall. Notably, the DOOR (sparse reward) environment with its 24 DoF Shadow hand, is solved by our method with a high success-rate.

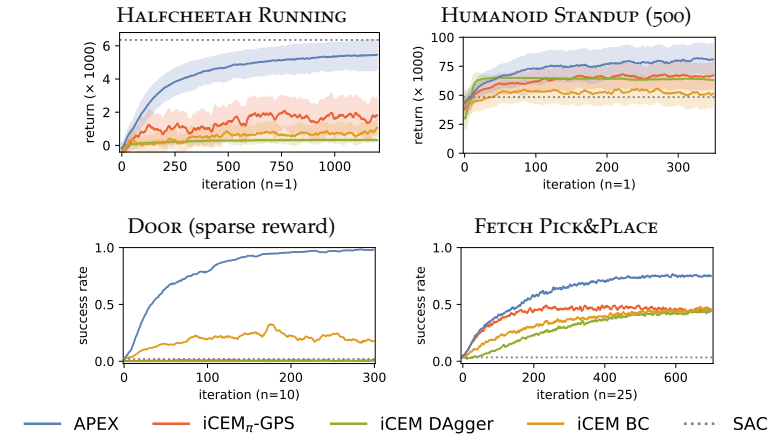


Figure 13: Policy performance on the test environments for APEX and baselines. SAC performance is provided for reference.

A very interesting effect of our approach is that the iCEM$_\pi$ expert working inside of APEX improves with the policy, as seen in Fig. 14. When guided by a policy, the performance raises (dashed blue lines) with the policy performance, even if the policies themselves are not very strong yet. Together, APEX is able to shrink the gap between expert and policy performance and yields strong results (see also the illustrative behaviors in Fig. 8). The policy can achieve signifi-

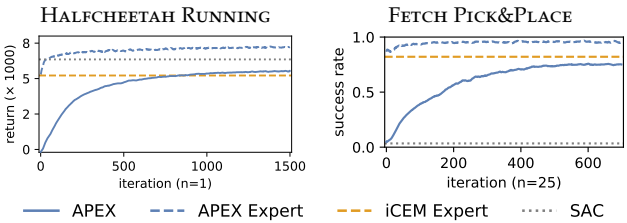


Figure 14: Interplay between policy and expert. Policy performance (solid line) and expert performances (dashed lines) on selected test environments for APEX. Due to warm-starting and adding policy samples, experts improve with the policy. For low budgets this effect is stronger, see Fig. 34.

cantly higher performance than iCEM especially in the low-budget case in HALFCHEETAH RUNNING, presented in Fig. 34 in Appendix C.

4.4.1 *Ablations*

Are all components of APEX required? We perform several ablations to investigate this question. Figure 15 shows the performance when removing each individual component from APEX, as detailed in Appendix C.2. Using the adaptive guidance cost instead of a *fixed* λ is only important in the sparse reward environments, as expected. In case of dense rewards, the λ adaptation does perform identically. Removing warm-starting has a drastic effect in all environments. APEX without DAgger is also much worse. Not adding policy samples to the optimization has an interesting effect: At first glance the policy performance is higher, however, asymptotically the full APEX is better or on par. The reason is shown in Fig. 16 where we report the expert performance of APEX and APEX without policy samples. The expert in the latter case only marginally improves upon the standard iCEM. Thus, the policy is limited to the iCEM performance, whereas in APEX it can go beyond, see Fig. 14.

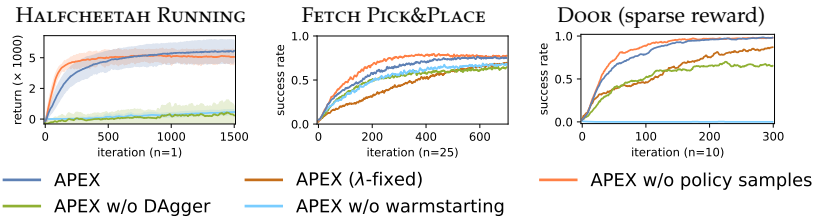


Figure 15: Ablation experiments. We remove different components of the APEX algorithm, see legend. In case of HALFCHEETAH RUNNING, the performance for APEX with λ-fixed is not reported as it matches that of APEX.

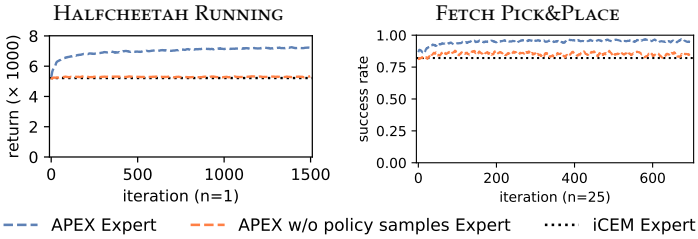


Figure 16: The expert performance for APEX and APEX without adding policy sample. As seen, the expert performance improves with learnt policy as the added policy sample directs expert distribution towards a better solution space.

4.5 CONCLUSIONS

We considered the problem of policy extraction from model-based trajectory optimizers based on sampling like iCEM. Our method (APEX) is able to extract strong policies for hard robotic tasks which are especially challenging for model-free RL methods. This is achieved by imitation learning from a guided iCEM expert, where both policy and expert mutually propel themselves to higher performances.

With this work, we want to propose a promising stepping stone towards learning high-performing policies for real robots, where speed and minimal interaction with the system are more important than asymptotic performance. Model-based RL methods are very sample efficient [32, 62] in terms of actual environment interactions, but are too slow to run in real-time yet. With the recently proposed iCEM, the possibility for real-time planning became feasible but still needs huge computational resources. Extracting high performing policies is a promising route to success, where we provide here an important ingredient.

Another route to real robot applications is to pretrain models and policies in simulation and then adapt to the real system and extract policies with APEX. Similarly, the sim-to-real transfer can be done without learned models but using the ground-truth simulations as done here to get a strong initialization for fine-tuning on the real hardware. One might ask: why adopting a model-based planning method if we are using only a simulator? Because planning with population-based optimizers can produce nearly-optimal solutions and the extracted policy (if able to match expert's performance) can beat model-free baselines like SAC, which can get stuck in local minima in high-dimensional systems, as happened in the complicated Humanoid Standup task.

This chapter, together with the first one, was focused on finding more general and faster solutions for continuous control problems. In the next chapter, we will analyze a different but very relevant aspect of model-based optimization: uncertainty propagation for risk-averse planning.

Algorithm 2: Improved Cross-Entropy Method with warm-starting and samples from policy π, denoted as iCEM$_\pi$. Blue marks policy *warm-starting* and red is *adding policy samples*.

Input: $f(\vec{a}, s)$: cost function; π: policy; N: # of samples; h: planning horizon; k: elite-set size; β: colored-noise exponent; σ_{init}: noise strength; *iter*: # of iterations; γ: reduction factor.

$\tau \leftarrow \emptyset$
for $t = 0$ ***to*** $T-1$ **do**
  $s \leftarrow$ get the current state from the environment
  if $t == 0$ **then**
    $\mu_0 \leftarrow h$-steps rollout of model with π starting from s
  else
    $\mu_t \leftarrow$ shifted μ_{t-1} with last time-step action given by π
  $\sigma_t \leftarrow$ constant vector in $\mathbb{R}^{d \times h}$ with values σ_{init}
  for $i = 0$ ***to*** $iter-1$ **do**
    $N_i \leftarrow \max(N \cdot \gamma^{-i}, 2 \cdot k)$
    samples $\leftarrow N_i$ samples from $\text{clip}(\mu_t + \mathcal{C}^\beta(d, h) \odot \sigma_t^2)$
      // with $\mathcal{C}^\beta(d, h)$ colored-noise Normal distribution with noise-exponent β and dimension (d, h)
    add a fraction of the shifted/reused elite-set to samples
    if $i ==$ *last-iter* **then**
      add μ_i to samples
      add policy actions to samples // h-step rollout with policy from current state s
    costs $\leftarrow$ cost function $f(\vec{a}, s)$ for $\vec{a}$ 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
  $a \leftarrow$ first action of best elite sequence
  add (s, a) to τ
  execute a
return τ // return the trajectory

Algorithm 3: Adaptive Policy EXtraction procedure (APEX)

Input: iCEM$_\pi$; π_θ: policy network; n: # rollouts per iteration

init θ randomly;
$\mathcal{D} \leftarrow \emptyset$;
$i \leftarrow 1$
while *not converged* **do**
  $f(\vec{a}, s) \leftarrow J_t(\vec{a}, s_t) + \lambda_1 D_{\text{KL}}(\pi_\theta || \vec{a}) + \lambda_2 \|\vec{a}\|$ // see Eqns. 15, 16, and 17
  $\tau_{\text{CEM}} \leftarrow n$ Rollout with iCEM$_\pi(f(\vec{a}, s), \pi_\theta)$ // τ is the resulting trajectory
  add τ_{CEM} to $\mathcal{D}$
  $\theta \leftarrow$ train policy π_θ on $\mathcal{D}$
  $\tau_\pi \leftarrow$ Rollout with π_θ
  $\tau_{\text{DAgger}} \leftarrow$ relabel actions in τ_π with iCEM$_\pi(f(\vec{a}, s), \pi_\theta)$ // DAgger
  add τ_{DAgger} to $\mathcal{D}$
  $\theta \leftarrow$ train policy π_θ on $\mathcal{D}$
  $i \leftarrow i + 1$ // one APEX iteration
return π_θ

5

RISK-AVERSE ZERO-ORDER TRAJECTORY OPTIMIZATION

Summary

At the beginning of this dissertation, we discussed how zero-order trajectory optimizers are well-suited for MPC and MBRL. One of the most popular applications in MBRL combines CEM with an ensemble of stochastic neural networks incorporating epistemic and aleatoric uncertainty (PETS). However, the receding horizon problem does not explicitly take into account the learned uncertainties, throwing away a lot of information. In this chapter, we introduce a simple but effective method for managing risk in zero-order trajectory optimization, balancing optimism in the face of epistemic uncertainty and pessimism in the face of aleatoric uncertainty. Various experiments indicate that the separation of uncertainties is essential to perform well with data-driven MPC approaches in uncertain and safety-critical control environments.

5.1 INTRODUCTION

Data-driven approaches to sequential decision-making are becoming increasingly popular [24, 104–106]. They hold the promise of reducing the number of prior assumptions about the system that are imposed by traditional approaches that are based on nominal models. Such approaches come in several different flavors [16]. Model-free approaches attempt to extract closed-loop control policies directly from data, while model-based approaches rely on a learned

This chapter is based on the paper *"Risk-averse zero-order trajectory optimization"*, Marin Vlastelica, Sebastian Blaes, Cristina Pinneri, Georg Martius [12].

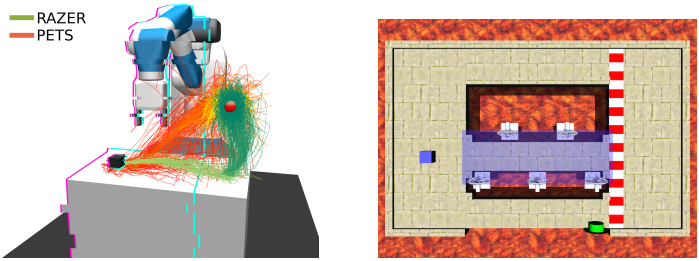
model of the dynamics to either generate novel data to extract a policy or to be used in a model-predictive control fashion (MPC). This work belongs to the latter line of work.

Model-based methods have several advantages over pure model-free approaches. Firstly, humans tend to have a better intuition on how to incorporate prior knowledge into a model rather than into a policy or value function. Secondly, most model-free policies are bounded to a specific task, while models are task-agnostic and can be applied for optimizing arbitrary cost functions, given sufficient exploration.

Nevertheless, learning models for control comes with certain caveats. Traditional MPC methods require the model and cost function to permit a closed-form solution which restricts the function class prohibitively. Alternatively, gradient-based iterative optimization can be employed, which allows for a larger class of functions but typically fails to yield satisfactory solutions for complicated function approximators such as deep neural network models. In addition, calculating first-order or even second-order information for trajectory optimization tends to be computationally costly, which makes it hard to meet the time constraints of real-world settings. This motivates the usage of zero-order methods, i.e gradient-free or sample-based, such as the improved Cross-entropy Method (iCEM) that do not rely on gradient information but are efficiently parallelizable.

Many methods relying on a learned model and zero-order trajectory optimizers have been proposed [32, 62, 64], but all share the same problem: compounding of errors through auto-regressive model prediction. This naturally brings us to the question of how can we effectively manage model errors and uncertainty to be more data-efficient and safe. Arguably, this is one of the main obstacles to applying data-driven model-based methods to the real world, e.g. to robotics settings.

In this chapter, we introduce a risk-averse zero-order trajectory optimization method (RAZER) for managing errors and uncertainty in zero-order MPC and test it on challenging scenarios (Fig. 17).



(a) Noisy-FetchPickAndPlace (b) BridgeMaze

Figure 17: Environments considered for uncertainty-aware planning. Code and videos are available at https://martius-lab.github.io/RAZER/

We argue that it is essential to differentiate between the two types of uncertainty in the model-predictive setting: the aleatoric uncertainty arising from inherent noise in the system and epistemic uncertainty arising from the lack of knowledge [107, 108]. We measure these uncertainties by making use of probabilistic ensembles with trajectory sampling similar to PETS [32]. Our contributions can be summarized as follows: (i) a method for separation of uncertainties in probabilistic ensembles (termed PETSUS); (ii) efficient use of aleatoric and epistemic uncertainty in model-based zero-order trajectory optimizers.

5.2 RELATED WORK

UNCERTAINTY ESTIMATION In the typical model-based reinforcement learning (MBRL) setting, the true transition dynamics function is modeled through an approximator. Impressive results have been achieved by both parametric models [63, 109–111], such as neural networks, and nonparametric models [112–115], such as Gaussian Processes (GP). The latter inspired seminal work on the incorporation of the dynamics model's uncertainty for long-term planning [115, 116]. However, their usability is limited to low-data, low-dimensional regimes with smooth dynamics [117, 118], which

is not ideal for robotics applications. Alternative parametric approaches include ensembling of deep neural networks, used both in the MBRL community [32, 119], and outside [120, 121]. In particular, ensembles of *probabilistic* neural networks established state-of-the-art results [32], but focus mainly on estimating the expected cost and disregard the underlying uncertainties. In comparison, we propose a treatment of the resulting uncertainties of the ensemble model.

ZERO-ORDER MPC The learned model can be used for policy search like in PILCO [29, 115, 116, 122] or for online model-predictive control (MPC) [32, 40, 123]. In this work, we do planning in an MPC fashion and employ a zero-order method as a trajectory optimizer, since they have shown to be less likely to get stuck in local minima and make an explicit treatment of the uncertainty in the cost possible. Specifically, we consider a sample-efficient implementation of the Cross-Entropy method [46, 81] introduced in [93] and explained in detail in Chapter 3.

SAFE MPC Separating the sources of uncertainty is of particular importance for applications directly affecting humans' safety, as self-driving cars, elderly care systems, or in general any application that involves a physical interaction between agents and humans. Disentangling epistemic from aleatoric uncertainty allows for separate optimization of the two, as they represent semantically different objectives as per definition. Extensive research on uncertainty decomposition has been done in the Bayesian setting and the context of safe policy search [124–126], MPC planning [127–129], and distributional RL [130, 131]. On the other hand, a state-of-the-art baseline for ensemble learning like PETS [32], despite estimating uncertainty, only optimizes for the *expected* cost during action evaluation. Our work aims at filling this gap by explicitly integrating the propagated uncertainty information in the zero-order MPC planner.

5.3 METHOD

Our approach concerns itself with the efficient usage of uncertainties in zero-order trajectory optimization and is therefore generally applicable to such optimizers. We are interested in modeling noisy system dynamics $s_{t+1} = f(s_t, a_t, w(s_t, a_t))$ where f is a nonlinear function, s_t the observation vector, a_t applied control input and $w(s_t, a_t)$ a noise term sampled from an arbitrary distribution.

Consequently, in the absence of prior knowledge about the function f, the system needs to be modeled by a complex function approximator such as a neural network. Furthermore, we are interested in managing uncertainties based on our fitted model, which is erroneous. To this end, we use stochastic ensembles of size K, where the output of each model $\boldsymbol{\vartheta}^k(s_t, a_t)$ are parameters of a normal distribution depending on input observation s_t and control a_t. As a by-product, our auto-regressive model prediction based on sequence of control inputs, indicated by the vector $\vec{a}$, becomes a predictive distribution over trajectories $\tau = (s_0, a_0, s_1, a_1 \dots)$; $\psi^\tau(s_t, \vec{a}) := p(\tau | s_t, \vec{a}; \theta)$ where θ denotes the parameters of the ensemble. For convenience, from this point onward we will differentiate between multiple usages of ψ^τ. We denote with:

- $\psi^s_{\Delta t}$ the distribution $p(s_{t+\Delta t} | s_t, \vec{a}_{t:t+\Delta t-1}; \theta)$ over states at time step $t + \Delta t$,
- $\psi^{\boldsymbol{\vartheta}}_{\Delta t}$ the distribution over the Gaussian parameter outputs $p(\boldsymbol{\vartheta}_{t+\Delta t} | s_t, \vec{a}_{t:t+\Delta t-1}; \theta)$ at time step $t + \Delta t$ of the planner.

5.3.1 *Planning and Control*

To validate our hypothesis that accounting for uncertainty in the environment and model prediction is essential to develop risk-averse policies, we use the Cross-Entropy method with the improvements presented in Chapter 3 (iCEM). Accordingly, at each time step t we sample a finite number of control sequences $\vec{a}$ for a finite horizon H, which we evaluate from the state s_t using an auto-regressive forward model and the cost function. The sampling distribution is

refitted in multiple rounds based on the best performing trajectories. After this optimization step, the first action of the mean of the fitted Gaussian distribution is executed. Since this approach utilizes a predictive model for a finite horizon at each time step, it naturally falls into the category of MPC methods.

Although we use iCEM, our approach of managing uncertainty can generically be applied to other zero-order trajectory optimizers such as MPPI [40] by a modification of the trajectory cost function.

5.3.2 *The Problem of Uncertainty Estimation*

Since we have a stochastic model of the dynamics, at the model prediction time step t we observe a distribution over potential outcomes. Indeed, since our model outputs are parameters of a Gaussian distribution, with auto-regressive predictions we end up with a distribution over possible Gaussians for a certain time step t.

Given a sampled action sequence $\boldsymbol{a}$ and the initial state s_t we observe a distribution over trajectories ψ_τ. To efficiently sample from the trajectory distribution ψ_τ we use the technique introduced by Chua et al. [32] (PETS) which involves prediction particles that are sampled from the probabilistic models and randomly mixed between ensemble members at each prediction step. In this way, the sampled trajectories are used to perform a Monte Carlo estimate of the expected trajectory cost $\mathbb{E}_{\tau\sim\psi^\tau}[c(\tau)]$. However, this does not take the properties of ψ^τ into account, which might be a high-entropy distribution and may lead to very risky and unsafe behavior. In this work, we alleviate this by looking at the properties of ψ^τ, i.e. different kinds of uncertainties arising from the predictive distribution.

5.3.3 *Learned Dynamics Model*

We learn a dynamics model f_θ that approximates the true system dynamics $s_{t+1} = f(s_t, a_t, w(s_t, a_t))$. As a model class, we use an ensemble of neural networks with stochastic outputs as in Chua

et al. [32]. Each model k, parameterizes a multivariate Gaussian distribution with diagonal covariance,

$$f_\theta^k(s_t, a_t) = \mathcal{N}(s_{t+1}; s_t + \mu_\theta^k(s_t, a_t), \Sigma_\theta^k(s_t, a_t))$$

where $\mu_\theta^k(\cdot, \cdot)$ and $\Sigma_\theta^k(\cdot, \cdot)$ are model functions outputting the respective parameters.

Iteratively, while interacting with the environment, we collect a dataset of transitions $\mathcal{D}$ and train each model k in the ensemble by the following negative log-likelihood loss on the Gaussian outputs:

$$\mathcal{L}(\theta, k) = \mathbb{E}_{s_t, a_t, s_{t+1} \sim \mathcal{D}} \Big[-\log \mathcal{N}(s_{t+1}; s_t + \mu_\theta^k(s_t, a_t), \Sigma_\theta^k(s_t, a_t)) \Big] \tag{19}$$

In addition, we use several regularization terms to make the model training more stable. We provide more details on this in App. D.2.

5.3.4 *Separation of Uncertainties*

In the realm of parametric estimators, two uncertainties are of particular interest. *Aleatoric* uncertainty is the kind that is irreducible and results from inherent noise of the system, e.g. sensor noises in robots. On the other hand, we have *epistemic* uncertainty resulting from lack of data or knowledge, which is reducible. This begs the question, how can we separate these uncertainties given an autoregressive dynamics model f_θ? The way that we efficiently sample from ψ^τ is by mixing sampled prediction particles, according to the sampling procedure $TS1$ from Chua et al. [32], and by additionally propagating the Gaussian mean along the planning horizon H. This process is illustrated by the red and yellow lines in Fig. 18. We call this propagation method PETSUS, standing for Probabilistic Ensembles with Trajectory Sampling and Uncertainty Separation.

Aleatoric uncertainty

Simple model prediction disagreement is not a good measure for aleatoric uncertainty since it can be entangled with epistemic un-

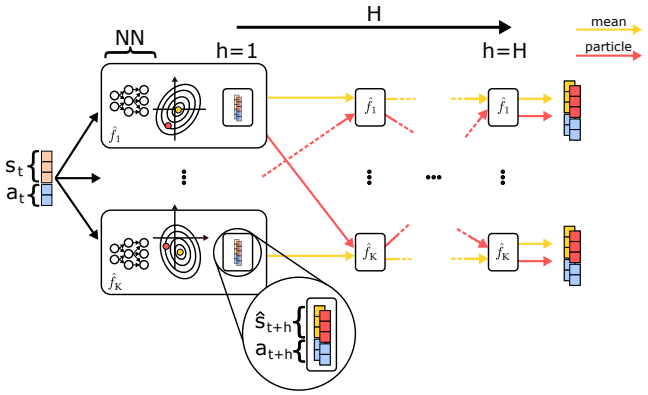


Figure 18: Probabilistic Ensembles with Trajectory Sampling and Uncertainty Separation (PETSUS)

certainty. Given how we model the system dynamics, we measure aleatoric uncertainty as entropy of the predicted normal distributions across ensemble members. More concretely, given a sampled particle state at planning step $t + \Delta t$, $\tilde{s}^k_{t+\Delta t}$, we define the estimated aleatoric uncertainty for an ensemble member associated to particle k at time step $t + \Delta t$:

$$\mathfrak{A}^k_{\Delta t}(s|\tilde{s}^k_{t+\Delta t}, a_{t+\Delta t}) = \mathcal{H}_{s\sim\psi^s_{\Delta t,k}}(s) \tag{20}$$

Where $\psi^s_{\Delta t,k}$ is the output distribution of ensemble model k after applying the action sequence $\vec{a}$ up to time step $t + \Delta t$. The quantity of interest for us is the *expected* aleatoric uncertainty for time slice $t + \Delta t$:

$$\mathfrak{A}_{\Delta t} = \sum_{k=0}^{\mathcal{K}} \mathfrak{A}^k_{\Delta t}(s|\tilde{s}^k_{t+\Delta t}, a_{t+\Delta t}) \tag{21}$$

Intuitively, since we only have access to the ensemble for sampling, we take a time-slice in the sampled trajectories from ψ^τ and compute the output entropies. Moreover, since we assume a Gaussian 1-step predictive distribution, this is an expectation over differential Gaussian entropy, which can be computed in closed form.

In this work, we use the variance Σ in place of the entropy of the Gaussian distribution, which scales linearly with $\log|\Sigma|$. Therefore, the expected particle variance for time slice $t+1$ of the prediction horizon is:

$$\mathrm{Var}^{\mathfrak{A}}_{t+1} = \frac{1}{K}\sum_{k=1}^{K}\Sigma^k_\theta(\tilde{s}_t, a_t) \tag{22}$$

Note that $\Sigma^k_\theta(\tilde{s}_t, a_t)$ outputs the covariance of the prediction for the following time step $t+1$. In the following, we use entropy of Gaussian and variance interchangeably as uncertainty estimates.

Epistemic uncertainty

For estimating the epistemic uncertainty, one would be tempted to look at the disagreement between ensemble models in parameter space $\mathrm{Var}[\theta]$, but this is not completely satisfying, since neural networks tend to be over-parametrized and variance within the ensemble still may exist albeit the optimum has been reached by all ensemble models. An alternative would be to calculate the Fisher information metric $\mathcal{I} := \mathrm{Var}[\nabla_\theta \log \mathcal{L}(s_{t+1}|s_t, a_t)]$ where $\mathcal{L}$ denotes the likelihood function, but this tends to be expensive to compute.

Therefore, we consider the predictive entropy over the Gaussian parameters $\boldsymbol{\vartheta}$ at time step $t+\Delta t$ as our measure for the epistemic uncertainty:

$$\mathfrak{E}(s_t, \vec{a}_{t:t+\Delta t-1}) = \mathcal{H}_{\psi^{\boldsymbol{\vartheta}}_{\Delta t}}(\boldsymbol{\vartheta} \mid s_t, \vec{a}_{t:t+\Delta t-1}) \tag{23}$$

This quantity is 0 given perfect predictions of the model. Note that, because of auto-regressive predictions of a nonlinear model, this is a very difficult object to handle. Nevertheless, since our predictive distribution $p(s|s_t, a_t; \boldsymbol{\vartheta})$ is parametrized by model output, we may utilize disagreement in $\boldsymbol{\vartheta}_t$ to approximate $\mathfrak{E}$. To get correct estimations, we need to propagate mean predictions $\bar{s}$ in addition to the particles as illustrated as the yellow lines in Fig. 18. We quantify epistemic uncertainty as ensemble disagreement at time step t:

$$\mathrm{Var}^{\mathfrak{E}}(s_{t+1}) = \mathrm{Var}^e[\mu^k_\theta(\bar{s}_t, a_t)] + \mathrm{Var}^e[\Sigma^k_\theta(\bar{s}_t, a_t)] \tag{24}$$

where Var^e is the empirical variance over the $k = 1 \dots K$ ensembles.

5.3.5 *Implementing Risk-Averse ZERo-Order Trajectory Optimization (RAZER)*

We assume the task definition is provided by the cost $c(s_t, \vec{a})$. For trajectory optimization, we start from a state s_t and predict with an action sequence $\vec{a}$ the future development of the trajectory τ. Along this trajectory, we want to compute a single cost term which is conveniently defined as the expected cost of all particles $\tilde{s}$ summed over the planning horizon H:

$$c(s_t, \vec{a}) = \sum_{\Delta t=1}^{H} \frac{1}{K} \sum_{k=1}^{K} c(\tilde{s}^k_{t+\Delta t}, a_{t+\Delta t}). \tag{25}$$

The optimizer, in our case iCEM, will optimize the action sequence $\vec{a}$ to minimize the cost in a probabilistic sense, i.e. $p(\vec{a} \mid s) \propto \exp(-\beta\, c(s, \vec{a}))$ where β reflects the strength of the optimizer (the higher the more likely it finds the global optimum).

To make the planner uncertainty-aware, we need to make sure it avoids unpredictable parts of the state space by making them less likely. Using the aleatoric uncertainty provided by PETSUS (Eq. 22), we define the aleatoric penalty as

$$c_{\mathfrak{A}}(s_t, \vec{a}) = w_{\mathfrak{A}} \cdot \sum_{\Delta t=1}^{H} \sqrt{\mathrm{Var}^{\mathfrak{A}}_{t+\Delta t}}, \tag{26}$$

where $w_{\mathfrak{A}} > 0$ is a weighting constant. The larger the aleatoric uncertainty, the higher the cost.

To guide the exploration to states where the model has epistemic uncertainty (Eq. 24), we use an epistemic bonus:

$$c_{\mathfrak{E}}(s_t, \vec{a}) = -w_{\mathfrak{E}} \cdot \sum_{\Delta t=1}^{H} \sqrt{\mathrm{Var}^{\mathfrak{E}}_{t+\Delta t}}, \tag{27}$$

where $w_{\mathfrak{E}} > 0$ is a weighting constant.

5.4 EXPERIMENTS

We study our uncertainty-aware planner in 2 continuous state and action space environments and compare to naively optimizing the

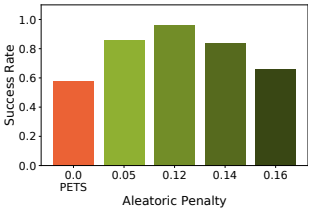
particle-based estimate of the expected cost similarly to Chua et al. [32]. Two additional environments are considered in the Appendix D. We start by giving a description of the environments.

BRIDGEMAZE This toy environment (see Fig. 17b) was specifically designed to study the different aspects of uncertainty independently. The agent (blue cube) starts on the left platform and has to reach the goal platform on the right. To reach the goal platform, the agent has to move over one of three bridges without falling into the lava. The upper bridge is safeguarded by walls; hence, it is the safest path to the goal but also the longest. The lower bridge has no walls and therefore is more dangerous for an unskilled agent to cross but the path is shorter. The middle bridge is the shortest path to the goal. However, randomly appearing strong winds perpendicular to the bridge might cause the agent to fall off the bridge with some probability, making this bridge dangerous.

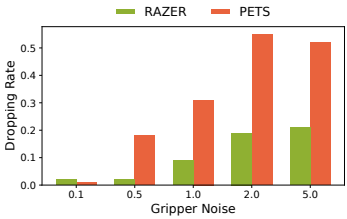
NOISY-FETCHPICKANDPLACE Based on the *FetchPickAndPlace-v1* gym environment. Additive action noise is applied to the gripper so that its grip on the box might become tighter or looser. The noise is applied for x-positions < 0.8 which is illustrated in Fig. 17a by a blue line causing the agent to drop the box with high probability if it tries to lift the box too early.

5.4.1 *Algorithmic Choices and Training Details*

For model-predictive planning we use the iCEM implementation described in Chapter 3 [93]. Further details about hyperparameters can be found in Appendix B.2. For planning, we use the same architecture for the ensemble of probabilistic models, both in RAZER and in PETS. The only difference is that in RAZER we also forward propagate the mean state predictions in addition to the sampled state predictions. Further details can be found in Appendix D.2.1.



(a) *BridgeMaze* success depending on $w_{\mathfrak{A}}$ for 50 runs.



(b) Dropping rate in *NoisyFetchPickAndPlace* for 100 runs.

Figure 19: Risk-averse planning in the face of aleatoric uncertainty yields higher success rates in noisy environments. For (b) we use ground truth models and a fixed aleatoric penalty weight $w_{\mathfrak{A}}$.

5.4.2 *Risk-Averse Planning*

Once a good model is learned, it can be used for safe planning. What differentiates RAZER from PETS is that it makes explicit use of uncertainty estimates while in the latter uncertainties only enter planning by taking the mean over the particle costs and not differentiating between different sources of uncertainty.

BRIDGEMAZE Figure 19a shows the success rate of PETS and RAZER in the *BridgeMaze*. In both cases, we use the same model that was trained from data collected during a training run with $w_{\mathfrak{E}} = 0.05$. Hence, the model saw enough training data from all parts of the state space. The noise in the environment is tuned such that there is a chance to cross the bridge without falling. While in Fig. 37b PETS avoided this path because of an overestimation of the state's value due to a lack of training data and sometimes sees a chance to cross the bridge. However, these attempts are very likely to fail because of stronger winds that occur randomly, resulting in a success rate of only 58%. RAZER does not rely on sampling for the aleatoric part and can thus avoid risk. With a higher penalty constant the success rate increases up to 96% but only as long as the agent is willing to take a risk at all. For large values of $w_{\mathfrak{A}}$ the

agent becomes so conservative that it only moves slowly (decreasing reward in Fig. 19a).

FETCHPICKANDPLACE In this environment, a 7-DoF robot arm should bring the box to a target position – starting and target positions are at the opposite sides of the table. The shortest path is to lift the box and move in a straight line to the target. However, with noise applied to the gripper action, there is a certain probability to drop the box along the way. When penalizing aleatoric uncertainty, this is avoided and also fewer trajectory samples are "wasted" in high-entropic regions, as presented in Fig. 17a. Figure 19b shows the number of times the box is dropped on the table depending on the aleatoric penalty. RAZER adopts a cautious behavior, preferring to slide the box on the table and lifting it only in the area without action noise, maintaining a dropping rate lower than 20%, even when considerable noise is applied.

5.5 CONCLUSION

In this chapter, we have provided a methodology to separate uncertainties in stochastic ensemble models (PETSUS) which can be used as a tool to build risk-averse model-based planners (RAZER). This type of risk-averseness can be achieved by a simple modification of the cost function in form of uncertainty penalties in zero-order trajectory optimizers. Furthermore, the separation of uncertainties allows us to do active exploration via the epistemic bonus which favours better generalization of the learned model. Further details on this part can be found in App. D.

While in this chapter we analyzed possible ways to deal with long-horizon predictions and uncertainty propagation for auto-regressive models, in the next one we will present a method to learn informative cost functions, in order to shorten the planning horizon and not rely on inaccurate predictions.

6

NEURAL ALL-PAIRS SHORTEST PATH FOR REINFORCEMENT LEARNING

Summary

In the course of this dissertation, we have used the improved CEM as a trajectory optimizer for receding-horizon control problems. Thanks to the long-horizon sampling with colored noise, we were able to find plausible paths to the goal even with sparse cost functions. However, if the planning horizon needs to be reduced for computational constraints, it becomes essential to have a more informative cost function. This trade-off between the accuracy given by planning over long horizons and the information coming from the cost function has led us to propose a self-supervised method for learning distances from scratch, which can be used as a cost function for model-based RL, or reshaped as a reward for model-free RL. In this chapter, we propose a method that learns *neural all-pairs shortest paths*, which we use to learn goal-conditioned policies, requiring zero domain-specific knowledge. In particular, our approach includes both a self-supervised signal from the temporal distance between state pairs of an episode, and a metric-based regularizer that leverages the triangle inequality for additional connectivity information between state triples. This dynamical distance is fully self-supervised, compatible with off-policy learning, and robust to local minima.

This chapter is based on the paper *"Neural all-pairs shortest path for reinforcement learning"*, Cristina Pinneri, Georg Martius, Andreas Krause [13].

6.1 INTRODUCTION

Teaching an AI agent to autonomously and efficiently reach goals is still an open problem in the reinforcement learning (RL) community. One of the bottlenecks is correctly specifying a learning signal: an intuitive choice would be a binary reward indicating whether the goal is reached or not, but this kind of signal does not provide enough feedback to the learner, as it would require extensive exploration before receiving any information. As an alternative, hand-crafted reward shaping is often employed on top of the "true" sparse reward - such as the Euclidean distance to the goal [132, 133] - but that might lead to reward hacking, a widely observed phenomenon in which the agent optimizes for the local optima introduced by the shaped reward instead of the real sparse objective, generating unintended behaviors [134].

Instead of imposing brittle heuristic shaping, recent research shows that learning *dynamical distances* is a valid alternative [135]. The purpose of these functions is to learn all-pairs shortest path distances (APSP). One of the first works in this direction was presented in the 90s by Kaelbling [136] and utilizes a Q-learning agent to construct a goal-conditioned action-value function $d(s, a, g)$, drawing *de facto* the first connection between RL and the APSP problem, based on the Floyd-Warshall algorithm [137].

However, Kaelbling [136] demonstrate the method in a low dimensional tabular setting, where all the goals were known in advance. Recent works [135, 138] propose extensions including function approximation, but are constrained to on-policy learning. We propose a method that learns a dynamical distance function with function approximation, **off-policy**, without any reinforcement signal, employing only a self-supervised signal from the actual number of time steps separating state pairs, and a second loss inspired by the triangle inequality, as in the all-pairs shortest path algorithm.

6.1.1 *Shortest Paths and RL*

The problem of learning dynamical distances is closely related to the one of learning value functions in RL. The Q-learning algorithm itself can be seen as a generalization of a single-source shortest path problem (SSSP) for directed graphs, in the context of decision making. The negative Q-function $-Q_\pi(s, \pi(s))$ represents the shortest path distance under the policy π. The dynamic programming (DP) equation underlying Q-learning is the Bellman update [139], while the one for single-source shortest path is provided by the Dijkstra's algorithm [140] or by other variations like the Bellman-Ford algorithm [141], depending on the input graph class. However, the nature of these recursive equations can still be traced to the Bellman's optimality principle [142].

Goal-conditioned Q-functions [143] generalize distance learning for a multi-goal setting, as they are defined over pairs of states and desired goals. This is similar to the all-pairs shortest path problem (APSP), which computes the shortest path between any two vertices in a graph. However, the DP equations behind these problems are different. Goal-conditioned Q-learning is typically based on the temporal difference update, where the only bootstrapped value is the estimate of the Q-function. Dedicated algorithms for the solution of the APSP problem are built on different DP principles.

The Floyd-Warshall algorithm [137] for APSP, for example, finds shortest paths by iteratively enforcing the triangle inequality between state triples: the shortest path between state s_i and s_k is cast in terms of *relaxations*, such that the distances are initialized by overestimation, and are updated as

$$d(s_i, s_k) := \min_j \left[d(s_i, s_j) + d(s_j, s_k) \right] .$$

$O(N^3)$ relaxations are needed for the algorithm to converge, where N is the number of vertices in the graph. Kaelbling [136] propose

a RL connection to the Floyd-Warshall algorithm that also includes actions, formalized as

$$d(s_i, a_i, s_k) := \min_j \left[d(s_i, a_i, s_j) + \min_{a'} d(s_j, a', s_k) \right].$$

Differently from a temporal difference update, this equation makes use of available estimates of $d(s_i, a_i, s_j)$ as well. Several other works [136, 138, 144] adapt the Floyd-Warshall algorithm to RL, with results mainly in the tabular setting or for on-policy learning. These methods rely on having accessibility to the ground truth cost between state pairs. Our approach lifts this impractical assumption and learns distances in a self-supervised fashion. Moreover, since we are in a reinforcement learning setting, we can let the agent decide *which* paths to increase depending on their reachability, in a bottom-up approach, rather than initializing all the distances to infinity as in the Floyd-Warshall algorithm.

To summarize, in this chapter we present a way to learn dynamical distances and goal-conditioned policies off-policy, and in a fully self-supervised way, employing:

- a novel off-policy correction term for distances learned via temporal regression, and a corrective bootstrapping loss inspired by the Floyd-Warshall algorithm on graphs, adapted to RL.
- a general method to learn dynamical distance functions from scratch, that can be used as part of a goal-conditioned RL algorithm, or as a cost for model-based RL, and that, differently from previous work, successfully deals with local minima (created e.g. by obstacles) without explicitly encoding any type of information in the distance function.

6.2 RELATED WORK

Dealing with environments with sparse reinforcement signals complicates long-term credit assignment, as it introduces higher variance in Monte Carlo (MC) learning and higher bias in Temporal

Difference (TD) updates [145]. Many techniques are focused on accelerating learning by "densifying" the reward signal with auxiliary functions while not affecting the optimal policy of the underlying sparse problem [132, 146, 147]. Alternatively, other kinds of auxiliary reward functions can be used to facilitate exploration according to different strategies as curiosity [148], prediction error [149], information gain [150], or predicting state reachability [151]. On the other side there are methods like Hindsight Experience Replay (HER) [103] that do not try to solve the exploration problem, but introduce auxiliary rewards based on the distance to the achieved goal rather than to the desired one. As done in HER, we also employ goal-conditioned value functions [143].

Our approach, however, is based on dynamical distances, which are functions expressing the distance between state pairs based on some notion of *functional similarity*. In the literature, dynamical distances have been learned via direct regression using temporal regression [135], in the form of goal-conditioned policies [152], via Q-learning by relabeling goals [153, 154], or goal-directed Q-learning with negative transition mining [155]. We extend the on-policy temporal regression presented in Hartikainen et al. [135] to be suited for off-policy learning, while simultaneously learning a goal-conditioned policy that uses the learned dynamical distance as a negative reward. Differently from other methods [151, 153], we do not explicitly perform graph search. We instead make use of a term that expresses whether two states were ever part of the same trajectory. At training time, we only check if the *edge* (s, a, s') has ever been visited or not. In practice, we use locality sensitive hashing [156], which allows for generalization based on angular distance similarity between states.

6.3 BACKGROUND

We consider the problem of an agent attempting to solve a goal-reaching task. The goal reaching Markov Decision Process (MDP) [103, 143] is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{G}, r, \gamma, \rho_0, \rho_g)$, where the

state, action and goal spaces are indicated by $\mathcal{S}, \mathcal{A}, \mathcal{G} \in \mathbb{R}$. In our case, the goal space $\mathcal{G}$ can be an arbitrary subset of the state space. The initial state s_0 is sampled from the distribution ρ_0, while the goal is sampled from ρ_g. Our aim is to learn a stochastic policy $\pi : \mathcal{S} \times \mathcal{G} \to \Lambda(\mathcal{A})$ that maximizes the expected discounted return

$$\mathbb{E}_{\tau \sim \rho_\tau, s_g \sim \rho_g} \Big[\sum_{t=0}^{T-1} \gamma^t r(s_t, a_t, s_g) \Big]$$

6.3.1 *Hindsight Experience Replay*

Let us now consider the particular case in which the reward function is sparse, i.e. the indicator function $r(s_t, a_t, g) = -\mathbb{I}\{s_t \neq g\}$. This assumption complicates the credit assignment problem and becomes a bottleneck for sample efficiency. To overcome this issue, Andrychowicz et al. [103] introduce Hindsight Experience Replay (HER). The main idea of HER is to relabel past failed trajectories with the indicator function mentioned above, where the goal g is the one that was actually achieved, rather than the desired one.

Although success signals are generally more frequent while using HER, they remain binary, and thus hold little information. In this work we build upon the relabeling framework introduced in HER, but we propose to compute rewards from a more informative learned dynamical distance. Our distance, however, is not tied to the goal sampling procedure presented in HER, and can be used with any relabeling scheme or goal-conditioned RL algorithm as a shaped reward.

6.3.2 *Dynamical Distances*

We learn a distance function that predicts the expected number of timesteps along the shortest path between two states. This is learned via a self-supervision signal given by the empirical distance between states of a rollout, as done in Hartikainen et al. [135], which

we extend to an off-policy setting. In the original paper, the distance d_θ^π is associated with a policy π, which gives rise to the training loss:

$$L_d^\pi(\theta) = \frac{1}{2}\mathbb{E}_{\substack{\tau\sim\rho_\pi \\ i\sim[0,T] \\ j\sim[i,T]}}\left[\left(d_\theta^\pi(s_i, s_j) - (j-i)\right)^2\right], \qquad \texttt{(temporal regression)}$$

where the empirical distance between the states $(j-i)$ is relative to the policy π. The complete dynamical distance learning (DDL) algorithm uses $d_\theta^\pi(s, s_g)$ as a negative reward to optimize the policy π and it is guaranteed to converge to the optimal policy π^*. However, it requires on-policy data collection, which bottlenecks the efficiency of the pipeline. Moreover, it also requires a small amount of goal proposals to be used during on-policy training, selected by a human operator from the last visited goals. In practice, the user is shown a batch of achieved goals and has to select the "best" one, which will be used to train the policy π.

6.4 METHOD

6.4.1 *Off-policy Temporal Regression*

To overcome the on-policy bottleneck, and achieve better sample efficiency with off-policy data, we propose to condition the dynamical distance on the action value and learn the function $d_\theta(s_i, a_i, s_j)$ from a replay buffer by minimizing the following loss:

$$L_d(\theta) = \mathbb{E}_{\substack{\tau\sim\rho_\pi \\ i\sim[0,T] \\ j\sim[i,T]}}\left[\left(d_\theta(s_i, a_i, s_j) - \min\left[(j-i), d_{\theta_{old}}(s_i, a_i, s_j) + \mathrm{U}[d_{\theta_{old}}(s_i, a_i, s_j)]\right]\right)^2\right], \tag{28}$$

where θ_{old} are previous distance estimates from a frozen network, updated at a slower frequency for stability reasons, and $\mathrm{U}[\cdot]$ can indicate any measure of uncertainty associated to the network given the input (s_i, a_i, s_j).

Like this, we can train d_θ from suboptimal off-policy trajectories, as the supervision signal is the minimum value between the upper

bound of the true distance, given by the empirical distance $(j - i)$, and a frozen estimate of the dynamical distance.

Intuitively, thanks to the generalization capabilities of powerful approximators such as neural networks, $d_{\theta_{old}}$ may provide reasonable lower distance estimates, which can be used as a target in the loss, rather than some temporal overestimate $(j - i)$ coming from suboptimal trajectories at the beginning of training. We explain in further detail our choice of uncertainty measure in Sec. 6.4.2.

6.4.2 *Uncertainty with Counts*

We introduce an uncertainty term in Eq. 28 so that the min operator discards distance estimates with high uncertainty, and it starts to consider proposals from the distance network otherwise. Our formulation of the uncertainty term is inspired by a recent work on pessimistic initialisation of Q-functions by [157] with count models. We adopt the same model based on locality-sensitive hashing (LSH) [156], but we are not bound to any particular counting scheme. We define the count model as $N(s_i, a_i)$ and use it to pessimistically initialise the distance in the following way:

$$\mathrm{U}[d_\theta(s_i, a_i, s_j)] \stackrel{\text{def}}{=} \frac{C}{(1 + N(s_i, a_i))^M} \qquad \texttt{for the temporal loss} \tag{29}$$

where $M, C > 0$ are hyperparameters. This penalty term polynomially decays with the number of counts, so that the distance gets updated with the off-policy estimate $d_\theta(s_i, a_i, s_j)$ only if that state-action combination has been seen enough times. This pessimistic term is also used to avoid the trivial zero solution of Eq. 28.

6.4.3 *Local Connectivity and Triangular Loss*

When the goal is not part of the replay buffer and we are in the presence of obstacles, the learned distance underestimates the shortest path to the goal. Let us consider the example in Fig. 20. In this case, the initial states distribution ρ_0 samples the starting points from

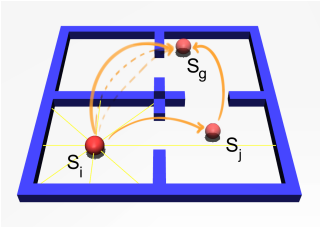


Figure 20: 2D point-mass example. The goal has never been visited. Only the intermediate state is present in the replay buffer. The length of the arrows indicates the distance value between state pairs.

the lower left room. On the other hand, the goals are always sampled from the upper right room. Unless the agent sees a sufficient amount of paths that lead to the goal, it will not be able to learn the correct shortest path distance.

Contrary to the Floyd-Warshall algorithm, where all the distances are initialized with infinity and then "relaxed" with the triangle inequality, we consider an alternative approach and start with underestimates caused by the off-policy temporal regression, which we artificially increase whenever we sample another point in the buffer that is connected to the goal. In order to express this connectivity information, we employ the count based term already introduced in the previous section to estimate uncertainty. The count term $N(s_i, a_i, s_g)$ then indicates how many times in total the goal s_g was reached from state s_i after taking the first action a_i.

For instance, let us consider the case in which the goal s_g was never reached from state s_i, but there is a path in the buffer $\mathcal{D}$ from an intermediate state s_j to the goal state s_g (Fig. 20). In this case, we increase the value of $d_\theta(s_i, a_i, s_g)$ as:

$$d^\star_\theta(s_i, a_i, s_g) = d_\theta(s_i, a_i, s_j) + d_\theta(s_j, a_j, s_g) \tag{30}$$

The distance $d^\star_\theta(s_i, a_i, s_g)$ is then a penalized hypotenuse, that will converge to the true shortest path distance and thus mitigate local minima.

6.5 ALGORITHM SUMMARY

The final formulation for the loss comprises the sum of these two terms (from Eq. 28 and 30):

$$L_{temporal}(\theta) = \mathbb{E}_{\substack{\tau\sim\mathcal{D} \\ i\sim[0,T] \\ j\sim[i,T]}} \left[\left(d_\theta(s_i, a_i, s_j) - \min[(j-i), d_{\theta_{old}}(s_i, a_i, s_j) + \mathrm{U}[d_{\theta_{old}}] \right)^2 \right] \tag{31a}$$

$$L_{triangular}(\theta) = \mathbb{E}_{\substack{s_g\sim\mathcal{D}_{last} \\ (s_i,a_i)\sim\mu(s_g) \\ (s_j,a_j)\sim\overline{\mu}(s_g)}} \left[\left(d_\theta(s_i, a_i, s_g) - (d_{\theta_{old}}(s_i, a_i, s_j) + d_{\theta_{old}}(s_j, a_j, s_g)) \right)^2 \right] \tag{31b}$$

where $\mu(s_g) = \{(s,a) | (s,a) \in \mathcal{D}_{last} \wedge N(s,a,s_g) = 0\}$ and $\overline{\mu}(s_g) = \mathcal{D} \setminus \mu(s_g)$

where $\mathcal{D}_{last}$ is the replay buffer containing the most recently collected rollouts. Notice that in the triangular loss we can either increase or decrease the value of the hypotenuse. In one case the hypotenuse is penalized, in the other one it is regularized (relaxed, as in Floyd-Warshall) because a better path was found. Moreover, in this loss, s_g always indicates the desired goal of the trajectory which s_i is sampled from. Thus, we are only correcting the paths that should have led to the goal. We underline that this procedure does not search for the (s_j, a_j) tuple minimizing the triangle inequality, it purely samples (s_j, a_j) from the points actually present in the replay buffer $\mathcal{D}$.

6.6 EXPERIMENTS

Our method is tested on 4 environments illustrated in Fig. 21:

2D NAVIGATION A point mass in two dimensions has to navigate a playground with internal walls to reach a goal. In the setting we present, using the Euclidean goal-distance as a heuristic cost would fail the task because of the presence of local optima created by the walls.

Algorithm 4: Proposed Neural-APSP algorithm.

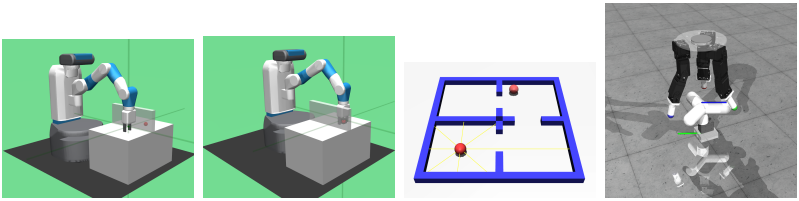
Input : $\mathcal{D}$: empty replay buffer; $\mathcal{D}_{last}$: replay buffer of most recently collected rollouts; θ: distance network parameter; θ_{old}: distance target network parameter; I: training iterations

Output: goal-conditioned policy π, distance network d_θ

for $i = 0$ ***to*** $I-1$ // loop over training iterations

do

- $\mathcal{D}_{last} \leftarrow$ COLLECT DATA WITH POLICY π
- UPDATE COUNT MODELS with data from $\mathcal{D}_{last}$
- **Train** `TemporalLoss`**:**
  - sample episode from buffer $\mathcal{D}$
  - sample two indices i and j, with $j > i$
  - fit distance network with input (s_i, a_i, s_j) and label $\min[j-i, d_{\theta_{old}} + \mathrm{U}]$ // Eq. 31a
- UPDATE TARGET NETWORK
- **Train** `TriangularLoss`**:**
  - sample s_i from trajectory τ in $\mathcal{D}_{last}$ and set s_g to desired goal state of τ
  - sample s_j from full replay buffer $\mathcal{D}$
  - **if** $N(s_i, a_i, s_g) == 0$ *and* $N(s_j, a_j, s_g) > 0$ **then** // path present from s_j but not s_i
    - compute new hypotenuse $d^\star(s_i, a_i, s_g)$ using the sampled s_j // Eq. 30
  - **else**
    - do nothing
  - fit distance network with input (s_i, a_i, s_g) and label $d^\star(s_i, a_i, s_g)$ // Eq. 31b
- **Train** `Policy` π**:**
  - sample episodes from the replay buffer $\mathcal{D}$
  - relabel with HER and reward given by negative distance
  - train π with SAC on relabeled data
- $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_{last}$



(a) Fetch Reach (with and without Wall)

(b) Fetch Pick and Place (with and without Wall)

(c) Point Maze

(d) D'Claw

Figure 21: MuJoCo Environments

3D REACHING TASK A 7DoF robot arm has to reach a goal. In the FETCH REACH WALL variation, the robot arm is initialized in one half of the table while the goal is placed on the other half, on the ground, with a wall separating the table in two halves.

3D MANIPULATION TASK A 7DoF robot arm has to fetch, pick and place a box to a target location (FPP in short). The goal can be sampled either above or on the table with 50% probability. In the FETCH PICK AND PLACE WITH WALL (FPP WALL), the box is always placed on one half of the table, and the goal in the other, so the the only solution is lifting it.

CLAW MANIPULATION [158] A 9-DOF "claw"-like robot is required to turn a valve to various positions. The state space includes the positions of each joint of each claw (3 joints on 3 claws) and embeds the current angle of the valve in Cartesian coordinate ($\theta \to (\sin\theta, \cos\theta)$). The robot is controlled via joint angle control. The goal space consists only of the claw angle, which is sampled uniformly from the unit circle.

For all the baselines and ablations we use HER [103] while learning a policy with Soft Actor Critic [159], while for the FETCH PICK AND PLACE environment we use HER combined with DDPG [22] due to better empirical performance. All the methods make use of the

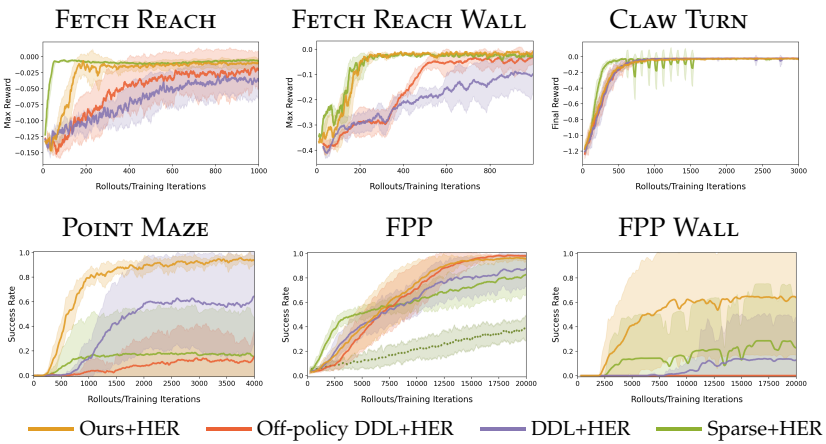


Figure 22: Performance degradation of Sparse+HER. When more exploration is needed or the environment presents non-trivial reachability properties, HER struggles to find a solution. Our method is either comparable to the best baseline/ablation, or better. The quantity presented in the plots is the rolling mean of the maximum reward or the success rate at evaluation time, averaged over 10 seeds. For the Fetch Pick and Place task we augment the goal space to include the end effector position as well. The dotted darker line is the original HER performance without augmentation.

future strategy presented in Andrychowicz et al. [103], each of them with the best k selected by grid search. The parameters used in the experiments are reported in the appendix. We consider 4 kinds of reward coupled with HER:

SPARSE+HER (BASELINE) Standard binary reward with hindsight relabeling of the buffer with the default SAC hyperparameters.

DDL+HER (BASELINE) We use the original HER relabeling scheme, and rewards given by the negative temporal loss (on-policy). This method is slightly different from [135] as it does

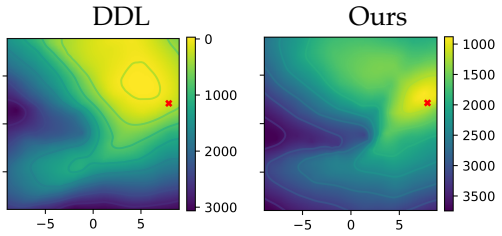


Figure 23: Heatmap of the Q function estimate (negative sign) for the POINT MAZE environment, learned by using DDL+HER (left) and Ours+HER (right).

not make use of the original "user preferences" to propose goals.

OFF-POLICY DDL+HER (ABLATION) We augment DDL+HER by introducing the off-policy temporal regression loss term of Sec. 6.4.1.

OURS+HER Our complete method, *neural APSP*, including the off-policy temporal regression loss (Eq. 31a) and the bootstrapped triangular loss (Eq. 31b), as described in Algorithm 4.

6.6.1 *Local Optima*

The results in Fig. 22 show how our method is robust even when we consider environments with obstacles or goals that are hard to reach. The off-policy temporal loss can find shortcuts as suggested by the function approximator, and introducing the triangular loss further improves performance In particular, the FETCH PICK AND PLACE task is an eloquent example. In this case we opt to augment the goal space to also consider the end effector position: the goal is to have both box and gripper at the target state. This simple addition already improves the sample efficiency of Sparse+HER by an order of magnitude. However, we can also notice how Sparse+HER quickly reaches a 50% success rate, equivalent to reaching all the

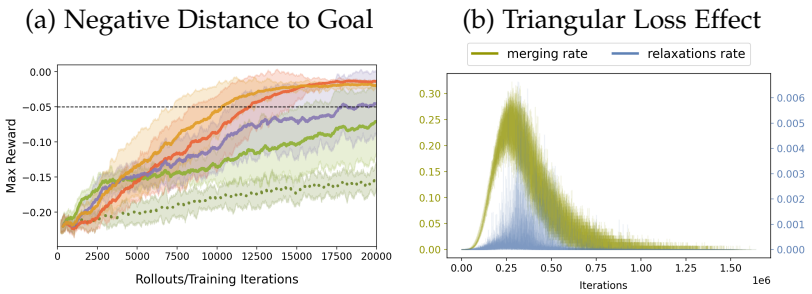


Figure 24: Fetch Pick and Place task. Our method is able to reach the goal at a faster rate as it avoids the local minima of placing the end effector in between the target in the air and the box on the ground. The dashed black line indicates the minimum negative distance required to solve the task. The dotted dark green line is the original HER performance [103] without goal augmentation.

goals placed on the table, but cannot generalize so easily to the goals in the air. In fact, given the augmented goal state, it often reaches a suboptimal solution by placing the end effector in between the current box position and its target position. While our method takes more time to catch up, it eventually overtakes both baselines. In Fig. 24b we show the effect of the triangular loss for the Fetch Pick and Place task. Specifically, the rates at which the distances to the goal get increased (merging rate) or regularized (relaxations rate) for every batch.

In the Fetch Pick and Place with Wall task, it is even more clear that the triangular loss consistently helps avoiding local minima. The robot arm is able to lift the box and bring it to the other side in most cases, without the need to specify an auxiliary reward to avoid the wall. The rates at which the hypotenuse gets penalized/merged or regularized/relaxed are presented in Fig. 26. We also show two illustrative frames from Sparse+HER and Ours+HER, that show the local minima where the baselines tend to get stuck (Fig. 25).

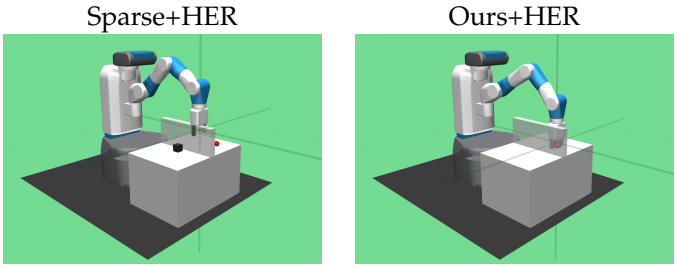


Figure 25: Frames of trajectories produced by a goal-conditioned policy learned from Sparse+HER vs Ours+HER. The additional wall introduces further exploration difficulties in the Fetch Pick and Place task.

6.6.2 *Sample Efficiency*

Being off-policy, our method can make better use of the transitions present in the replay buffer. In Fig. 27 we see how, even if all methods are trained with a fixed exploration noise and a max-entropy policy, only our approach is able to consistently explore the goal distribution (top right room). In the specific example of the Point Maze, Off-Policy DDL+HER fails to solve the task on its own as it underestimates the distance to the goal. In Fig. 23 we represent the Q-function learned by SAC for DDL+HER, and for Ours+HER. The maximum value of the Q-function (its negative value, in the plot) is not centered around the goal for DDL+HER. We presume this might be due to estimation errors induced by the (off-policy) exploration noise.

We also achieve better sample efficiency for the Fetch Pick and Place task. As shown in Fig. 24a, we manage to get a maximum reward above the threshold after circa 11k rollouts, employing 45% less samples than DDL+HER, and achieving the same performance of Sparse+HER with less than half of the rollouts. In the variant With Wall, the efficiency gain is 5x higher.

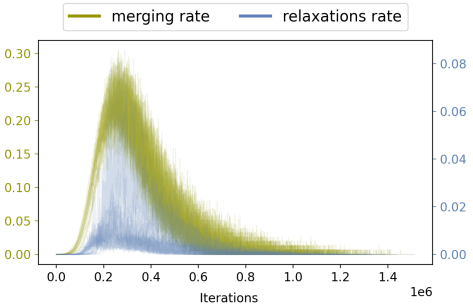


Figure 26: Triangular loss effect for the FETCH PICK AND PLACE WITH WALL task. Our learned distance helps the optimizer to avoid the local minima created by the wall. The *merging rate* of the hypotenuse into the catheti sum is the same as in the case without wall (Fig. 24b), namely, at most 30% of the training batch gets penalized. However, the rate at which the hypotenuse gets corrected – the *relaxations rate* – has non-zero values also later in training.

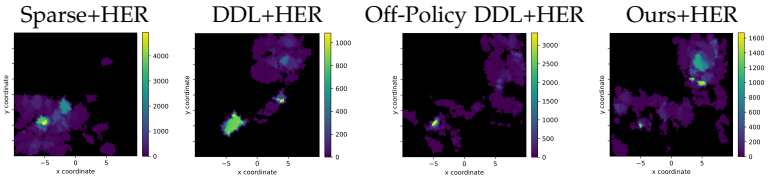


Figure 27: State coverage of POINT MAZE obtained with each method. Sparse+HER is only able to explore the lower room and parts of the adjacent one. All the methods are provided with either $\epsilon = 0.1$ or $\epsilon = 0.3$ exploration, depending on which performs better.

6.7 CONCLUSIONS

Learning distances or reward functions in a self-supervised fashion is a generally difficult problem, as it relies on bootstrapped estimates, which can produce strongly biased and high-variance solu-

tions. Solving it, leads to increased sample efficiency during learning which is highly relevant for reinforcement learning applied to real robotic tasks. In addition, learning the distance function does not require any domain knowledge nor expert demonstrations and is thus of general interest for goal-conditioned tasks. The proposed *neural APSP* algorithm shows promising empirical results which indicate that learning these distances, and the corresponding goal-conditioned policies, is not only possible, but also robust to local optima and sample efficient.

7

CONCLUSION

7.1 SUMMARY

Throughout this dissertation, we presented a series of methods aimed at overcoming some of the challenges of RL, with a focus on improving online sample efficiency, having more robust solutions, and learning distance functions that can reduce the planning horizon without sacrificing performance. Each chapter revolved around a different aspect of efficiency and robustness, specifically:

ICEM Chapter 3 presented iCEM and showed how zero-order optimizers can be better adapted for continuous control. The way in which the solution space is explored is more reliable and "far-sighted" than the myopic look-ahead given by the gradient vector. Our proposed algorithm introduces temporally correlated noise to account for the complexity of the action sequences along the planning horizon, and it reduces the amount of wasted samples by keeping memory of the elite set and by better propagating information inter– and intra–iterations. Adding all the improvements together, led to a performance increase of 120-1000% and a sample efficiency increase of 270-2200%. iCEM was also integrated into the model-based control library `mbrl-lib`[1] by Meta AI.

APEX Chapter 4 was motivated by the research question: can we distill the behavior of a powerful zero-order optimizer like iCEM into a single neural network? And if so, how does the performance compare with policies learned via model-free RL? Considering iCEM as the expert, we looked at the

1 https://github.com/facebookresearch/mbrl-lib

problem through the lens of imitation learning. Our results indicated that a simple combination of existing methods, like DAgger [59] or GPS [97], is not sufficient to correctly imitate a stochastic and multimodal expert like iCEM. With APEX, we proposed an adaptive method that modifies the expert to follow the student policy without the risk of prematurely collapsing to its behavior, achieving much better results than a strong model-free baseline like SAC [23].

RAZER Chapter 5 analyzed another important aspect of model-based RL with zero-order planners which was only partially investigated in previous work: using neural networks to model aleatoric and epistemic uncertainty and balancing them accordingly during planning. In particular, we argued that the planner should be pessimistic in the face of aleatoric uncertainty – for risk-averse behavior – and optimistic in the face of epistemic uncertainty – for active learning.

N-APSP In Chapter 6 we explored another aspect that improves the flexibility and efficiency of model-based approaches: NN function approximation for distance learning. Usually, in model-predictive control, the optimization problem considers the sum of the costs up to a fixed horizon. Attaching a terminal value function to the receding-horizon cost can give more feedback to the planner and eventually even drastically reduce the planning horizon [95, 160]. With our work on neural-APSP, we propose a way to learn this value in a fully self-supervised way. In our experimental evaluation, the distance is reshaped as a reward and used in a model-free fashion (no planning), proving that we can effectively learn domain-specific information in a distance function without the need for reward engineering.

7.2 LIMITATIONS AND FUTURE WORK

The results and contributions of this dissertation pave the way for many exciting avenues of future research.

For iCEM presented in Chapter 3, there are several possible improvements, intended to further increase the sample efficiency of such zero-order trajectory optimizers. In fact, while iCEM produces nearly-optimal action plans, it cannot be run in real-time for high-dimensional systems. One possibility could be to consider an *adaptive planning horizon* that varies depending on factors such as distance to the goal or uncertainty in the cost/model. This would allow the algorithm to focus the computational power on more relevant time steps of the optimization process. Additionally, one could investigate a *dynamic population size* based on the population variance, where a smaller variance would lead to fewer samples, and vice-versa. This adaptive sampling strategy may result in better performance and faster convergence.

The adaptive policy extraction proposed in Chapter 4 is still limited by the speed of the online sampling procedure of the iCEM expert. Assuming that this limitation is overcome, there is an additional bottleneck to consider: the efficiency of the relabeling procedure. Future work could address this issue by developing more advanced data aggregation techniques to prioritize informative samples for relabeling – e. g., considering uncertainty-aware queries as in [161] – thereby reducing the overall number of interactions required with the expert and the environment. Another direction is to integrate back reinforcement learning and consider methods from offline RL, which gained a lot of traction in recent years [162, 163]. Offline RL allows training the policy also from sub-optimal data, fully offline, without any need to query the agent that generated the data, therefore improving efficiency.

In Chapter 5, we introduced ensemble disagreement as an exploration strategy for active learning in model-based RL. It could be interesting to investigate other possibilities for structured exploration with dynamical models. For instance, incorporating graph neural

networks [164, 165] or other forms of structured models could enable agents to capture relational information and facilitate graph-based exploration strategies. Alternatively, other structural properties of the dynamical model can be leveraged for active learning. For example, equivariance-guided exploration, as briefly discussed in Appendix A, could help the RL agent identify patterns and symmetries, allowing it to prioritize exploration in areas of the state-action space that are expected to confirm or deny the presence of model equivariance.

Finally, extending the self-supervised distance learning approach in Chapter 6 to the offline setting could result in a more comprehensive and versatile framework for distance learning. Although various techniques for learning cost functions from scratch exist, most are limited to the online off-policy setting or require extensive domain-specific knowledge. Developing a robust and efficient framework for learning distances from logged data could open new possibilities for leveraging the vast amounts of available offline data for real-world applications. In addition to this, neural-APSP could be extended to consider pixel inputs as well, in order to be used for vision-based robotic applications. As our method consists of mainly two architectural components, a neural network for the distance function and a count model based on hashing, it could be in principle extended to image-based tasks as both of these elements have shown impressive results when learning from down-sampled images as well [135, 157].

ACKNOWLEDGEMENTS

I would like to begin by expressing my deepest gratitude to all those who have supported and encouraged me throughout the course of my doctoral research. The intellectual and professional growth I have experienced has been both challenging and rewarding in ways I could not have anticipated. My sincerest thanks go to my supervisors, Georg Martius and Andreas Krause, for their constant guidance, patience, and mentorship. Your expertise and commitment to my development as a researcher have been instrumental in the successful completion of this dissertation. I also express my gratitude to Marc Toussaint for being part of the defense committee.My appreciation also goes to the Max Planck ETH Center for Learning Systems (CLS) and the German Federal Ministry of Education and Research (BMBF) for the financial support that made this research possible.

To my fellow researchers and friends at the Autonomous Learning Group, thank you for the intellectual camaraderie, the stimulating debates, the barbecues in the forest, the pizza parties, and the shared laughter. You have made this journey enjoyable and memorable, and I could not have asked for a better group of colleagues. In particular, I wanna thank all the incredible women in the AL group, your presence has been a vital part in creating an encouraging and inspiring environment.

I would also like to thank the people that made my internships two remarkable experiences. To Roberto Calandra, for always being ready to discuss research despite the timezone and the parental leave. Heartfelt thanks also to Martin Riedmiller and the Control Team and other researchers at Deepmind. My experience in London was so great also thanks to your support and enthusiasm. In particular, thanks to Oliver, Vitaly, Abbas, Nicholas for the football matches, the sailing trip and the jam sessions. Moreover, I wanna thank Sarah for being an inspiring

friend and researcher, Mohak for being the coolest intern, and Federico for the great discussions on virtually any topic and the time together.

Doing a PhD also means going through the most defining years of our lives at a personal level, so I feel compelled to thank all those who have been there, whether through a kind word, a much-needed late-night discussion, or a shared moment of respite. In particular, I wanna thank Ahmad for the infinite kindness and the conversations on late-stage capitalism, Dominik for the unstoppable enthusiasm and the many Feierabende that helped us through the PhD, Michal for the deep conversations and the hike with my mom. I also wanna thank Giambattista for pushing me to think big and aim high, Sergey for the delightful discussions in art museums, Maggie for the friendship we developed painting after painting, and Marco for being an incredibly rare combination of wit and empathy.
Finally, a loving thought to my flatmates: Sabrina, Gigi, (Viola!) and Marie, my late twenties would have been dull and unmemorable without your friendship, you are simply the best.

Last but not least, I am deeply thankful to my family for their unwavering love and belief in me. To my parents, Antonella and Pasquale, for being there, always, and trying their best to support me. And to my sister, Aurora, I hope you know how much you and your support mean to me. Lastly, I wanna thank those who cannot hear my words: my *nonna* Caterina for having been like a second mother to me, and my dog Toby for being the most reliable and reproducible source of emotional support, which I learned to be two non trivial qualities in the world of deep learning.

With a touch of humor, I conclude with the words of Douglas Adams, "I may not have gone where I intended to go, but I think I have ended up where I needed to be". I dedicate this work to the unpredictable and serendipitous nature of academic research, and to all those who have joined me along the way.

A

ADDITIONAL PUBLICATIONS

A.1 EQUIVARIANT DATA AUGMENTATION FROM STATE INPUTS FOR GENERALIZATION IN OFFLINE RL

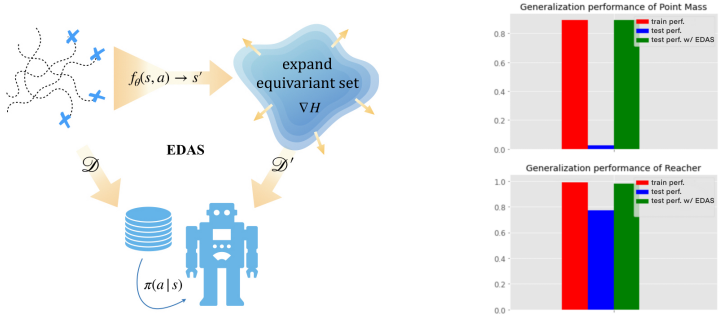


Figure 28: (a) EDAS framework. (b) Generalization performance on the Point Mass and Planar Reacher environments in the DeepMind Control Suite.

During my research internship at Deepmind, I investigated another possible way to improve data efficiency, this time in the context of offline Reinforcement Learning. Offline RL has the potential to enable learning policies from previously collected data, without the need for extensive online interactions. However, generalization to

This section is based on the paper *"Equivariant data Augmentation from state inputs for generalization in offline RL"*, Cristina Pinneri, Sarah Bechtle, Markus Wulfmeier, Arunkumar Byravan, Will Whitney, Jingwei Zhang, Martin Riedmiller.

unseen situations remains a major challenge. In this paper, we propose an Equivariant Data Augmentation approach from State inputs (EDAS) that leverages the structure of the underlying system to expand the dataset in a principled way. By first learning a dynamical model, our method identifies and exploits inherent invariances, leading to improved generalization performance. In particular, we learn the bounds of the equivariant transformation distribution (translation) using a combination of equivariance loss and entropy regularization, which allows us to handle partial symmetries and represent systems with invariances only in specific parts of their domains. Experiments on the DMControl Suite [166] demonstrate the effectiveness of our approach, with the learned policy showing comparable performance on both training and test goals. This framework is flexible, potentially adaptable to various group transformations and environments, and can be extended to handle imperfect models in noisy settings. Additionally, it opens up the possibility to use the equivariance prediction error to guide exploration during the model learning process and check whether the system is invariant to certain types of transformations or not.

A.2 PINK NOISE IS ALL YOU NEED: COLORED NOISE EXPLORATION IN DEEP RL

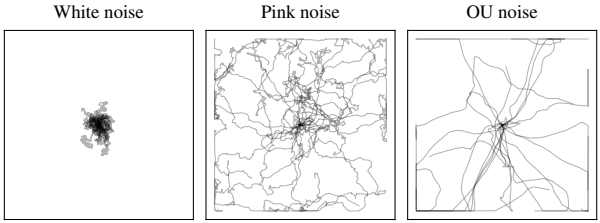


Figure 29: Trajectories of pure noise agents on a bounded integrator environment. Pink noise (center) provides a balance of local and global exploration, and covers the state space more uniformly than the other two.

This paper explores more in detail the benefits of using colored-noise exploration (Sec. 3.2.1) in deep reinforcement learning. In off-policy deep reinforcement learning with continuous action spaces, exploration is often implemented by injecting action noise into the action selection process. Popular algorithms based on stochastic policies, such as SAC or MPO, inject white noise by sampling actions from uncorrelated Gaussian distributions. In many tasks, however, white noise does not provide sufficient exploration, and temporally correlated noise is used instead. A common choice is Ornstein-Uhlenbeck (OU) noise, which is closely related to Brownian motion (red noise). Both red noise and white noise belong to the broad family of *colored noise*. In this work, we perform a comprehensive experimental evaluation on MPO and SAC to explore the effectiveness of other colors of noise as action noise. We find that pink noise, which is halfway between white and red noise, significantly outperforms white noise, OU noise, and other alternatives on a wide range of environments. Thus, we recommend it as the default choice for action noise in continuous control.

This section is based on the paper *"Pink noise is all you need: colored noise exploration in deep RL"*, Onno Eberhard, Jakob Hollenstein, Cristina Pinneri, Georg Martius [167].

B

APPENDIX TO CHAPTER 3: ICEM

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/.

Index

B.1 PERFORMANCE RESULTS

Table 3 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 3 is complemented by Fig. 30, which shows the additional PlaNet experiments with REACHER EASY, FINGER SPIN and CARTPOLE SWINGUP.

Table 3: 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_{MPC}	iCEM	CEM_{MPC}	iCEM	CEM_{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**	

B.1.1 *Budget selection*

Table 4 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 $\gamma = 1.25$,

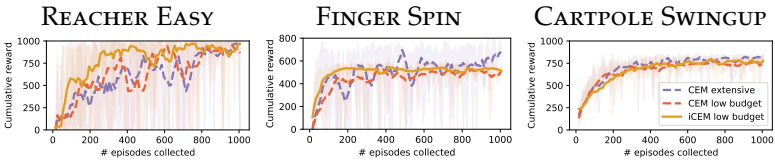


Figure 30: Additional PlaNet experiments. For details, see Fig. 6.

there are more *CEM-iterations* possible for iCEM while keeping the same budget.

Table 4: 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

B.2 HYPER-PARAMETERS

Zero 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 [168].

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 5.

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 parame-

ters are fixed to the same values for all the environments, with the exception of the noise-exponent β, as reported in Table 6.

The environment episode length (standard for these environments) are given in Table 7.

Table 5: 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

Table 6: 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

B.2.1 *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-

Table 7: Environment settings. These are the standard settings for the environments. For PlaNet the numbers come from the custom action repeat used in [63].

	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

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. 4b, 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.

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. 32.

B.2.2 *Sensitivity*

If the number of trajectories is high enough, there is little sensitivity to the other parameters, as shown in Fig. 31 and Fig. 32. This

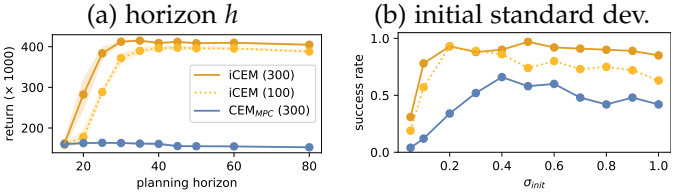


Figure 31: Sensitivity to hyper-parameters of iCEM. (a) horizon h in HUMANOID STANDUP and in (b) the initial standard deviation for FETCH PICK&PLACE. See Fig. 32 for the sensitivity to β.

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. 33. 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. 32b, 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. 31c: 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.

Another relevant observation is the effect of the planning horizon length h for the HUMANOID STANDUP in Fig. 31a: even in the low-budget case, iCEM can better exploit longer action sequences by generating samples with higher correlations in time.

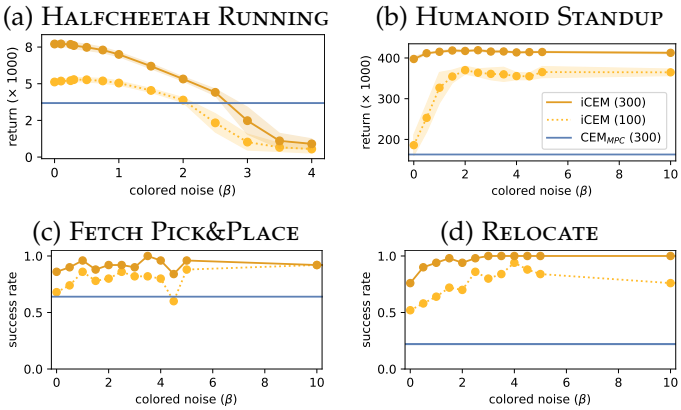


Figure 32: Sensitivity to the colored noise exponent β of iCEM.

B.2.3 *Hyperparameters for PlaNet*

We reimplemented PlaNet [63] in PyTorch to conduct our experiments and borrow all algorithmic details and hyperparameter settings from the original paper. As in [63], 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. 6. Results for additional environments are depicted in Fig. 30. 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 8 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 3.

Table 8: 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 6	see table 6

B.3 ABLATION RESULTS

In Fig. 33 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. B.2, 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. 33, 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.

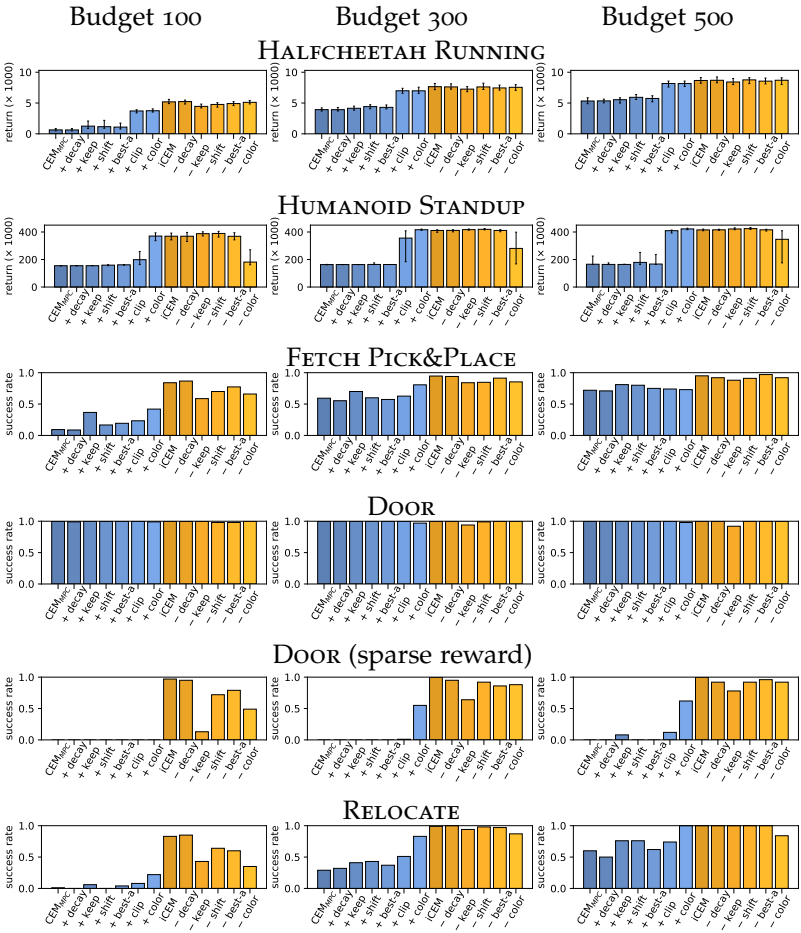


Figure 33: Ablation studies

B.4 DETAILS ON THE ICEM IMPROVEMENTS

B.4.1 *Shift Initialization*

The shift-initialization of the mean $\mu_{t-1}(\cdot, j+1)$ of the sampling distribution, as mentioned in Sec. 3.1.1 and used in Alg. 4 line 9 is as follows:

$$\mu_t(\cdot, j) = \mu_{t-1}(\cdot, j+1) \qquad \text{for } 1 \leq j \leq h-1 \tag{32}$$

$$\mu_t(\cdot, h) = \mu_{t-1}(\cdot, h) \tag{33}$$

where the parenthesis denote index-access: (action dimension, horizon timestep). Note, that in the CEM_{PETS} method Eq. 33 is $\mu_t(\cdot, h) = \vec{0}$.

B.4.2 *Sampling Colored Noise*

To sample action sequences with a specific power spectrum we use the efficient implementation of [71], which can be found as a python package at `https://github.com/felixpatzelt/colorednoise`.

B.4.3 *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, 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.

B.5 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: $C(\tau) = \mathcal{F}^{-1}[\text{PSD}_a(f)]$. If the power spectral density follows the inverse power law of Eq. (10), 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:

$$\begin{aligned} C(s\tau) &= \mathcal{F}^{-1}\left[\frac{1}{s}\text{PSD}_a\left(\frac{f}{s}\right)\right] \\ &= \mathcal{F}^{-1}\left[\frac{1}{s}s^{\beta}\text{PSD}_a(f)\right] \qquad \text{using Eq. (10)} \\ &= s^{\beta-1}\mathcal{F}^{-1}[\text{PSD}_a(f)] \\ &= s^{\beta-1}C(\tau) \end{aligned}$$

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 [169].

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

C

APPENDIX TO CHAPTER 4: APEX

The following material includes performance tables, wall-clock times for the proposed method per iteration, and extended ablation experiments to understand the performance of policy extraction and the expert. Lastly, we will also present the interplay between policy and expert as in Fig. 14 but for lower budgets.

Index

C.1 PERFORMANCE TABLES

We report the performance numbers for our experiments in Tab. 9, while the expert/teacher settings are indicated in Tab. 10, and policy settings are in Tab. 11. Wall-clock times are in Tab. 12.

Table 9: Performances for all environments for APEX and APEX policy. Enviroments are abbreviated for space reasons: HC-Run: HALFCHEETAH RUNNING, Hum-Up: HUMANOID STANDUP, and FPP: FETCH PICK&PLACE. We report the cumulative reward (marked with [1]) and the success rate (marked with [2]). SAC and iCEM baseline performances are provided for reference. For an explanation about the budget, see [93].

Envs	APEX (budget 45)		APEX (budget 100)		APEX (budget 300)		SAC
	Expert	Policy	Expert	Policy	Expert	Policy	
HC-Run[1]	5556±241	4847±665	7202±510	5596±779	9310±437	6218±1337	6352
Hum-Up[1]	149.0k±9.6k	73.5k±11.9k	173.7k±13.4k	80.5k±14.1k	207.3k±4.5k	88.7k±12.3k	48.4k
FPP[2]	0.893	0.0311	0.947	0.770	–	–	0.034
DOOR[2]	1.0	0.99	–	–	–	–	0.02

Envs	iCEM (budget 45)	iCEM (budget 100)	iCEM (budget 300)
HC-Run[1]	3488±119	5235.9±167	7632.54±250
Hum-Up[1]	146.3k±13.7k	182.2k±12.5k	202.4k±51.6k
FPP[2]	0.67	0.81	–
DOOR[2]	1.0	–	–

C.2 ABLATION EXPERIMENTS

In order to understand which components of APEX are affecting the performance of the policy extraction and the expert, several ablations were carried out. The results of this ablation studies are shown in Fig. 15 and Fig. 16. In the following, the implementation details of the different ablation studies are discussed.

APEX (λ-fixed): Instead of using the adaptive scheme for λ_j that were proposed in Eq. 17, its value is set to a constant value, with a value chosen to work well in the respective environment.

APEX w/o DAgger: Instead of using DAgger for policy extracting, plain behavioral cloning is used. In DAgger, states visited by the policy are relabeled with actions from the expert and added in addition to the expert data to the training dataset of the policy. In our ablation, only the data from the expert is added to the training

Table 10: Expert settings for the considered methods (The values for colored noise exponent for different environments are taken from [93])

	# elites K	Initial std. σ_{init}	Momentum α	Decay γ	Fraction reused elites ζ	Guidance scaling constant c	Horizon h
iCEM	10	0.5	0.1	1.25	0.3	–	30
iCEM$_\pi$	10	0.5	0.1	1.25	0.3	–	30
APEX	10	0.5	0.1	1.25	0.3	0.025	30
	Warm Start	Add Policy Sample	# rollouts per iteration n				
iCEM	False	False	–				
iCEM$_\pi$ / APEX	True	True	1 (Halfcheetah Running, Humanoid Standup) 10 (Door), 25 (Fetch Pick&Place)				

Table 11: Policy settings for iCEM$_\pi$ and APEX

# layers	Size	Activation fn	l1 reg.	l2 reg.	Optimizer	Learning rate
3	128	ReLu	1e-6	1e-5	Adam	5e-4
Batch size	Iterations	# latest rollouts used for training				
1024	1000	50 (Halfcheetah Running), 100 (Humanoid Standup) 150 · 25 (Fetch Pick&Place), 100 · 10 (Door)				

dataset of the policy. In case of DAgger, the policy gets twice the amount of data as in the case of behavioral cloning because in each iteration data from the expert and relabeled data from the policy is added to the dataset. In both cases, the same number of gradient-steps are performed during training of the policy.

APEX w/o warmstarting: Instead of initializing the means of all action dimensions along the planning horizon with the actions from the policy at the beginning of each rollout, means along action dimensions and the planning horizon are initialized with zero. After each planning step, the mean of action t is not initialized with the action from the policy but repeats the last action at time-step $t-1$.

Table 12: Wall-clock times for APEX per iteration. Reference machine for wall clock-times is an Intel Xeon Gold 6154 CPU @ 3.00GHz using 32 Cores for parallel simulation models. All times are in minutes.

		Wall-clock times per iteration (in min.)			
Env	Budget	Expert-only	DAgger-only	Training	Total
HALFCHEETAH RUNNING	45	1.8	2.0	1.4	5.6
	100	2.3	2.5	1.4	6.5
	300	4.0	4.2	1.5	10.1
HUMANOID STANDUP	45	1.7	2.5	0.9	5.3
	300	5.6	6.9	0.9	13.8
FETCH PICK&PLACE	45	7.8	8.7	3.2	20.0
(25 rollouts)	100	12.2	13.5	3.5	29.5
DOOR (10 rollouts)	45	9.7	11.2	3.9	25.1

APEX w/o policy samples: No additional sample trajectory with actions from the policy is added to the other samples during planning (see Alg. 4). To clarify in the full version with policy samples one of the random samples is overwritten to keep N samples.

C.3 EXPERT AND POLICY INTERPLAY

In Fig. 34 we report the performance of the policy and the experts, for different compute budgets. As discussed in the main paper, it is interesting to note that the expert inside APEX improves with the policy. The can lead to a higher policy performance than the original iCEM with the same compute budget.

Figure 34: Same as Fig. 14 but for different compute-budgets: 45, 100 (normal), 300. Notice, that in the case of low and normal budgets in HALFCHEETAH RUNNING the policy outperforms the iCEM expert. In FETCH PICK&PLACE the policies are able to match the iCEM performance.

D

APPENDIX TO CHAPTER 5: RAZER

In this supplementary material, we provide additional details for RAZER. For the sake of completeness, we will report some additional theory and experiments present in the published paper [12] and carried out mainly by my collaborators (Sec. 1.3). We also provide videos that showcase the risk-averse behavior of RAZER at https://sites.google.com/view/razer-traj-opt.

Index

D.1 ADDITIONAL THEORY AND EXPERIMENTS

D.1.1 *Extra environments*

We consider two additional environments:

Noisy-HalfCheetah This environment is based on *HalfCheetah-v3* from the OpenAI Gym toolkit. We introduce aleatoric uncertainty to the system by adding Gaussian noise $\xi \sim \mathcal{N}(\mu, \sigma^2)$ to the actions when the forward velocity is above 6. The action noise translates into a non-Gaussian and potentially very complicated state space noise distribution that makes the control problem very challenging.

Solo8-LeanOverObject In this robotic environment, the task of a quadrupedal robot [170] is to stand up and lean forward to reach a target position (purple markers need to reach green dots in Fig. 35) without hitting an object visualized by the red cube representing the unsafe zone. The robot starts in a laying position as shown in the inset of Fig. 35. As in the *Noisy-HalfCheetah* environment, Gaussian action noise is applied to mimic real-world perturbances.

D.1.2 *Risk-averse Planning*

NOISY-HALFCHEETAH How does RAZER perform on the *Noisy-HalfCheetah* environment when models are learned from scratch? Without aleatoric penalty, the planner is optimistic. Risky situations are only detected if a failing particle is sampled. Thus, the noise is mostly neglected and the robot increases its velocity, gets destabilized, and ends up slower than with the aleatoric penalty (Fig. 36a).

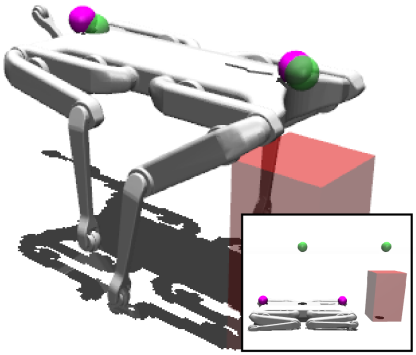


Figure 35: Solo8-LeanOverObject

D.1.3 *Probabilistic Safety Constraints*

When applying data-driven control algorithms to real systems, safety is of utmost importance. In the realm of zero-order optimization, safety constraints can be easily introduced by putting an infinite cost on constraint-violating trajectories. Nevertheless, we are dealing with erroneous stochastic nonlinear models which lead to nontrivial predictive distributions of future states, based on the control sequence $\vec{a}$. For this reason, we want to control the risk of violating the safety constraints that we, as practitioners, are willing to tolerate. If we denote the observation space as $\mathbb{S}$, given a violation set $\mathbb{C} \subset \mathbb{S}$, we define the probability of the control sequence $\vec{a}$ to enter the violation set at time $t + \Delta t$ as:

$$p(s \in \mathbb{C} \mid s_t, \vec{a}) = \int_{s \in \mathbb{C}} \psi^s_{\Delta t}(s \mid s_t, \vec{a}).$$

In practice, it is hard to compute this integral efficiently, since our distribution $\psi^s_{\Delta t}$ is nontrivial as a result of nonlinear propagation of uncertainty. Furthermore, the violation set $\mathbb{C}$ might not have the structure necessary to allow an efficient solution to the integral, in which case one needs to resort to Monte Carlo estimation.

To simplify computation and gain speed, we consider box violation sets resulting in each dimension of s being constrained to be outside of $[a, b] \in \{a, b \mid a, b \in \mathbb{R}^2, a < b\}$. By performing moment matching by a Gaussian in each time-slice $\psi^s_{\Delta t}$, the probability of ending up in state s at time step $t + \Delta t$ is given by integrating $\mathcal{N}(s; \mu_{t+\Delta t}, \Sigma_{t+\Delta t})$, where μ and Σ are estimated by Monte Carlo sampling. If we further assume a diagonal covariance Σ, this integral can be deconstructed into d univariate Gaussian integrals, which can be computed fast and in closed form. Hence, the probability of a constraint violation happening at time step t is defined by:

$$p(s \in \mathbb{C} \mid s_t, \vec{a}) = \prod_{i=0}^{d} \int_{s \in \mathbb{C}} \mathcal{N}(s^i; \mu^i_{t+\Delta t}, \sigma^i_{t+\Delta t}) \tag{34}$$

We integrate this into the planning method by adding:

$$c_{\mathfrak{S}}(s_t, \vec{a}) = w_{\mathfrak{S}} \cdot \sum_{\Delta t=1}^{H} [\![p(\hat{s}_{t+\Delta t} \in \mathbb{C}) > \delta]\!] \tag{35}$$

where $[\![\cdot]\!]$ is Iverson bracket. and $w_{\mathfrak{S}}$ is either a large penalty $c_{\max}$ or 0 to disable safety. An alternative for implementing safety constraints into CEM is by changing the ranking function [171]. The overall algorithm used in a model-predictive control fashion is outlined in sec. D.3.

D.1.4 *Active Learning for Model Improvement*

If model uncertainties are used for risk-averse planning, they are only meaningful if the model has the right training data. Only from good data can the parameters of the approximate noise model be learned correctly. In case of too little data, the agent might avoid parts of the state space due to an overestimation of the model uncertainties. On the other hand, the agent might enter unsafe regions for which the uncertainties are underestimated. By adding the epistemic bonus to our domain-specific cost, the planner can actively seek states with high epistemic uncertainty, i. e.for which no or only little training data exists.

Figure 19a shows this active data gathering process for the *Bridge-Maze* environment. PETS finds one particular solution to the problem of reaching the goal platform. It chooses the path over the safer, lower bridge rather than the dangerous middle path and the longer path via the upper bridge (Fig. 37b). Once, one solution is found, the model overfits to it without exploring any other parts of the state space. This is also reflected in the plateauing of the red curve in Fig. 37a.

In comparison, RAZER actively explores larger and larger parts of the state space with an increasing weight of the epistemic bonus (Fig. 37a). RAZER not only finds the easy solution found by PETS but also extensively explores other parts of the state space (Fig. 37c). To not get stuck at the middle bridge during exploration due to the inherent noise, it is important to separate between epistemic and aleatoric uncertainties. Only the former should be used for exploration. With enough data, our model can correctly capture the uncertainties of these states resulting in the epistemic uncertainty approaching zero.

D.1.5 *Planning with External Safety Constraints*

NOISY-HALFCHEETAH: We consider a safety constraint on the height of the body above ground simulating a narrow passage. Figure 36b shows the number of safety violations. Note that PETS has the same penalty cost for hard violations.

SOLO8-LEANOVEROBJECT: In this experiment, the robot has to move to two target points with its front and rear of the trunk while avoiding entering a specified rectangular area (fragile object). The front feet are fixed. To track the points, the robot has to lean forward, such that it can lose balance due to noisy actions. In contrast to PETS, RAZER successfully manages to satisfy the safety constraints almost always as shown in Fig. 38. However, satisfying the safety constraint comes with the cost of reduced tracking accuracy.

D.2 IMPLEMENTATION DETAILS

D.2.1 *Model Learning*

Parameters used for model learning in the *BridgeMaze* experiments:

Table 13: Model parameters for *BridgeMaze*

Ensemble parameters

Name	Value
num_layers	6
size	400
activation	silu
ensemble_size (n)	5
output_activation	None
l1_reg	0
weight_initializer	truncated_normal
bias_initializer	0
use_spectral_normalization	False

Stochastic NN parameters

Name	Value
var_clipping_low	-10.0
var_clipping_high	4
state_dependent_var	True
regularize_automatic_var_scaling	False

Remaining parameters

Name	Value
lr	0.002
grad_norm	2.0
batch_size	512
weight_decay	$1e^{-5}$
use_input_normalization	True
use_output_normalization	False
epochs	25
predict_deltas	True
train_epochs_only_with_latest_data	False
iterations	0
optimizer	Adam
propagation_method	TS1
sampling_method	sample

We bound the predicted log variance by applying (as in Chua et al. [32, see Appendix A.1])

```
logvar = max_logvar - softplus(max_logvar - logvar)

logvar = min_logvar + softplus(logvar - min_logvar)
```

to the output of the network that predicts the log variance, `logvar`. In principle, we could differentiate through this bound to automatically adjust the bounds `max_logvar` and `min_logvar`. However, we decided to not make these parameters learnable.

Parameters used for model learning in the *Noisy-HalfCheetah* environment (only differences to *BridgeMaze* environment):

Table 14: Model parameters for *Noisy-HalfCheetah*

Ensemble parameters

Name	Value
num_layers	4
size	200

Stochastic NN parameters

Name	Value
var_clipping_low	-6.0
state_dependent_var	True

Remaining parameters

Name	Value
lr	0.0002
grad_norm	None
batch_size	256
weight_decay	$3e^{-5}$
epochs	50

D.2.2 *Controller Parameters*

Parameters used in the iCEM controller. For an explanation of the different parameters, we refer the reader to Chapter 3 and Appendix B.2.

D.2.3 *Timings*

While our code is not tuned for speed specifically, in this section we provide some timings for a single step in the environment (hyperparameters are set as specified in App. D.2.1 and App. D.2.2, with num_simulated_trajectories = 128 and op_iterations = 3) in Table 18.

Table 15: Controller parameters, BridgeMaze environment.

Action sampler parameters

Name	Value
alpha	0.1
colored_noise	true
elite_size	10
execute_best_elite	true
finetune_first_action	false
fraction_elites_reused	0.3
init_std	0.5
keep_previous_elites	true
noise_beta	2.0
opt_iterations	3
relative_init	true
shift_elites_over_time	true
use_mean_actions	true

Remaining parameters

Name	Value
cost_along_trajectory	sum
delta	0.0
factor_decrease_num	1
horizon	30
num_simulated_trajectories	128

D.2.4 *Uncertainty Separation*

In our method, we separate the epistemic uncertainty, denoted as $\mathfrak{E}$ and aleatoric uncertainty, denoted as $\mathfrak{A}$, the details of which are explained in Sec. 5.3 with the resulting costs that arise. Since we are using a variant of the CEM algorithm that needs to sort the sampled action sequences $\vec{a}$ according to their cost, the cost of an action sequence is a single floating point number.

The stochastic NN ensemble that we are using samples trajectories from the predictive distribution ψ_τ for each action sequence $\vec{a}$. In addition, our variant (PETSUS), also propagates the mean pre-

Table 16: Controller parameters, Noisy-HalfCheetah environment (only difference to BridgeMaze environment).

Action sampler parameters

Name	Value
noise_beta	0.25
opt_iterations	4

Remaining parameters

Name	Value
num_simulated_trajectories	120

Table 17: Controller parameters, Solo8-LeanOverObject environment (only difference to BridgeMaze environment).

Action sampler parameters

Name	Value
init_std	0.3
noise_beta	3.0

diction $\bar{s}_t$ for each ensemble member for an action sequence $\vec{a}$. The auto-regressive prediction follows a recursive relation:

$$[\bar{s}_{t+1}, \Sigma_{t+1}] = \boldsymbol{\vartheta}(\bar{s}_t, a_t)$$

We make use of this in order to estimate the epistemic uncertainty $\mathfrak{E}$. At each time point of the predicted sequence of observations, we take the empirical variance of the outputted Gaussian parameters $\boldsymbol{\vartheta}(\bar{s}_t, a_t)$, predicted from the previous mean prediction $\bar{s}_t$ and control a_t, across the ensembles for that time slice in the predicted trajectories. This is then summed up across horizon H to obtain the epistemic bonus for action sequence $\vec{a}$.

Fig. 39 shows that scaling $w_{\mathfrak{E}}$ results in better state-coverage. This is of particular interest if we want to learn models that are able to generalize to different task settings, e. g.when changing the cost function. While the naive PETS algorithm overfits the model to the

Table 18: Timings per one environment step in ms. We measured the timings on a system with 1 GeForce GTX 1050 Ti, an Intel Core i7-6800K and 31GB of memory.

Environment	Timing [ms]
BridgeMaze	0.25
Noisy-HalfCheetah	0.14

task at hand, RAZER learns a truly task-agnostic model and is able to reap the benefits of model-based approaches to control.

For the aleatoric penalty we rely on the actual predictions of the covariance $\Sigma(s_t, a_t)$ and average them across the time slice, following with the sum across horizon H. Alternatively to this, we also use the entropy of the Gaussian as the $\mathfrak{A}$ uncertainty measurement. In Sec. D.2.5 we argue how these terms are interchangeable.

Note that, for the safety term ideally we want to use the full distribution ψ_τ and separation in aleatoric and epistemic uncertainty is neither required nor desirable.

D.2.5 *Entropy vs. Variance as Uncertainty Measurement*

We use entropy of Gaussian and variance interchangeably as uncertainty estimates. We have found that utilizing the variance directly causes RAZER to be much more risk-averse, which can be explained by the variance not being suppressed by the log term in the entropy. Moreover, using the variance directly is much more interpretable and easier to tune because it is of the same scale as the observation space.

D.2.6 *Observation Space vs. Cost Space Uncertainty*

A natural question to ask when attempting to make efficient use of uncertainties in MPC is where to measure these uncertainties. As

an alternative to observation space uncertainties, one could measure uncertainty in cost space. Here we argue why this is not a reasonable thing to do for each of the individual cost terms.

EPISTEMIC BONUS Since we operate under the desiderata that the benefit of model-based methods is in task-agnosticism, we should not measure epistemic uncertainty in the cost space, since this would decouple the task definition through the cost from the observation space and would lead to learning models that are not task-agnostic.

ALEATORIC PENALTY This is perhaps the most questionable case for using observation space uncertainty instead of cost space uncertainty. Nevertheless, we assume that high-aleatoric uncertainty translates to control difficulty, and we want to avoid parts of the observation space that are difficult to control. Moreover, the uncertainty measurements become completely invalidated in the case of a task switch, which plays against the task-agnosticism desiderata.

SAFETY PENALTY Safety is something that is enforced by infusing the algorithm with prior knowledge through a set of constraints which mostly manifest themselves as subsets of the observation space $\mathcal{S}$ or action space $\mathcal{A}$.

D.3 ALGORITHM

In Algo. 5 we provide an overview of the CEM algorithm that we utilize for implementing RAZER. Concretely, we use an improved sample efficient version of CEM as proposed by Pinneri et al. [93] that involves shift-initialization of the distribution mean, sampling time-correlated noise and further improvements.

Algorithm 5: RAZER: Risk-aware and safe CEM-MPC

Parameters:
  N: number of samples; P: Number of particles, H: planning horizon; $w_{\mathfrak{A}}$, $w_{\mathfrak{E}}$, $w_{\mathfrak{S}}$ CEM-iterations
for $t = 1$ ***to*** T // loop over episode length
do
  for $i = 1$ ***to*** *CEM-iterations* **do**
    $(\text{samples}_p)_{p=1}^{P} \leftarrow N$ samples from CEM(μ_t^i, Σ_t^i), with P particles per sample
    $c, c_{\mathfrak{A}}, c_{\mathfrak{E}}, c_{\mathfrak{S}} \leftarrow$ compute cost functions over particles
    $c_{\text{tot}} = c + c_{\mathfrak{A}} + c_{\mathfrak{E}} + c_{\mathfrak{S}}$ // compute total cost
    elite-set$_t \leftarrow$ best K samples according to total cost
    $\mu_t^{i+1}, \Sigma_t^{i+1} \leftarrow$ fit Gaussian distribution to elite-set$_t$
  execute first action of best elite sequence
  shift-initialize μ_{t+1}^1

D.4 ENVIRONMENTS DETAILS

All environments are based on the MuJoCo physics engine [66]. The **Noisy-Halfcheetah** and **Noisy-FetchPickAndPlace** environments are based on *HalfCheetah-v3* and *FetchPickAndPlace-v1*, respectively.

BRIDGEMAZE We designed the *BridgeMaze* environment to show the different aspects of uncertainty, namely the epistemic and aleatoric uncertainty, in isolation. The agent is a simple cube with only a free joint attached to it. The state-space

$$s = [s_0, s_1, s_2, a, b, c, d, v_{s_0}, v_{s_1}, v_{s_2}]$$

is 10-dimensional, consisting of 3 positional (s_0 to s_2), 4 rotational (a to d) and 3 velocity-based (v_{s_0} to v_{s_2}), agent-centric coordinates. The action-space $a = [\tau_{s_0}, \tau_{s_1}]$ is 2-dimensional. The torque τ applied to the agent in s_0- and s_1-direction.

The task in the environment is to reach a goal platform at $s_0^\star \geq 12$ by crossing one of three bridges that go over deadly lava.

The domain reward is defined as

$$r_t(s_t, a_t, s_{t+1}) = \begin{cases} |(s_0)_t - s_0^\star| - |(s_0)_{t+1} - s_0^\star|, & \text{if } (s_2)_{t+1} \geq -1.5 \\ 0, & \text{if } (s_0)_{t+1} \geq s_0^\star \text{ and } (s_2)_{t+1} \geq -1.5 \\ -1, & \text{otherwise} \end{cases} \tag{36}$$

where $s_0^\star$ is the x-coordinate of the goal state. Intuitively, the agent is guided by the negative distance to the target. The reward is zero once the agent goes beyond the imaginary finish line delimited at $s_0^\star = 12$. If the agent falls into the lava ($s_2 < -1.5$), it will get a reward of – 1 until the end of the episode. We define the cost for planning as $c_t(s_t, a_t, s_{t+1}) = -r_t(s_t, a_t, s_{t+1})$.

We designed the environments such that the agent is able to accelerate fast and also comes to a full stop relatively fast if no torque is applied. This makes the control problem and the task of learning the model relatively easy.

Noise is added in form of an external force in s_1-direction injected through the `xfrc_applied` attribute of the model. The sign of the force, as well as the force amplitude, sampled from $f_{\text{ext}} \in \mathcal{U}(0, f_{\text{ext}}^{\max})$, are randomly changing every 5 simulation steps. The external force is added only if $-8 \leq s_0 \leq 8$ and $-3.6 \leq s_1 \leq 3.6$ (area of the middle bridge). Otherwise the external force is zero.

NOISY-HALFCHEETAH We utilize a modified HalfCheetah environment where we apply a normally distributed noise term $\xi \sim \mathcal{N}(\mu, \Sigma)$ to the simulator state in the case when the velocity of the cheetah is greater than 6. More concretely, let s_t denote the simulator state at time step t, then the modified state is calculated as follows:

$$s_t' = s_t + \xi_t \tag{37}$$

In our case, Σ is a diagonal covariance matrix with the diagonal terms equal to 0.2. In addition, for the safety experiments with the Noisy-HalfCheetah we create a virtual ceiling at height $h = 0.3$. In the case that the body height crosses this threshold, the agent incurs

a large penalty. When the safety-constraint is violated, we don't end the episode.

NOISY-FETCHPICKANDPLACE We modified the original *FetchPickAndPlace-v1* environment to show the effect of the aleatoric penalty on the CEM action plan. Given the difficulty of the task, we performed the experiments without the learned model, using instead an ensemble of noisy ground truth dynamics. In this way, we could more easily understand the role of the aleatoric uncertainty during planning.

The noise term $\xi \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ is applied to the action controlling the gripper state: a positive additive noise forces the robot to open the grip with a force proportional to the noise magnitude. This noise is applied to all the ground truth models of the ensemble, and to the environment as well.

In particular, the box position is centered at y-coordinate -1.5 while the target is at $y = 2.0$. The gripper state is noisy until $y = 1.67$, right before the target.

SOLO8-LEANOVEROBJECT The state space of the this environment is 47-dimensional. It contains the absolute position, rotation, velocity and angular velocity of the robot as well as the positions and velocities of all the joints. In addition, the state contains the positions of the end-effectors and of the sites at the front and back of the robot. The actions space is 8-dimensional and controls the relative position of the joints. We fixed the two front legs of the robot with a soft-constraint to the ground to prevent the robot from uncontrollable jumping. We apply Gaussian noise to the action with a mean of 0 and a diagonal covariance matrix with the diagonal elements all being 0.3. The noise is uniformly applied over the entire state-action-space.

The experiments for the *Solo8-LeanOverObject* environment use the ground truth model during planning. The same noise were applied in the 'mental' as well as the 'real' environment.

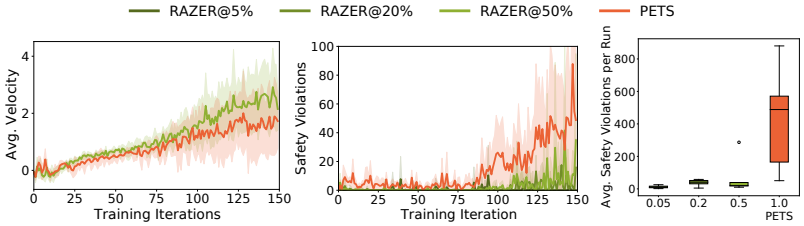
D.4.1 *Computing State-Space Coverage*

For computing the state coverage in Fig. 37a we divided the continues state-space in 50 equally spaced bins in the range $-20 \leq s_0 \leq 20$ and $-10 \leq s_1 \leq 15$. The state space-coverage is the fractions between states visited at least once and the total number of states.

D.5 APPLICATION TO TRANSFER LEARNING

In this work we have demonstrated that an approach such as PETS [32] to data-driven MPC that relies on zero-order trajectory optimization of the expected cost is not enough to manage uncertain environments and safety constraints. These problems need to be addressed when dealing with sim-to-real. The separation of uncertainties allows us to effectively manage epistemic uncertainty in the real system, which is important for improving the model once distribution shift to the real system happens. This can be done in a way of combining the epistemic bonus and probabilistic safety constraints, such that the policy explores parts of the state space where there is knowledge to be obtained while avoiding high-cost regions as a consequence of the incurred safety and aleatoric penalties.

In comparison to standard approaches for sim-to-real which involve domain randomization at training time, this approach incurs lower computational overhead and relies on learning on the real system.



(a) With aleatoric penalty (10 runs).

(b) With safety constraints.

(c) With safety constraints.

Figure 36: *Noisy-HalfCheetah* environment (task lengths 300 steps) with learned models. At 150 iterations we have seen only 45k points. (a) Performance under noisy actions. By applying the aleatoric penalty, RAZER can navigate the uncertainties better – leading to higher returns faster. (b) Safety violations above a certain body height (simulating a low ceiling) for different values of δ. In (c) the number of violations is averaged over the last 50 iterations (summed over 10 rollouts).

(a) State space exploration over time depending on epistemic bonus ($w_{\mathfrak{E}}$).

(b) State space coverage with $w_{\mathfrak{E}} = 0$.

(c) State space coverage with $w_{\mathfrak{E}} = 0.05$.

Figure 37: Active learning setting: The epistemic bonus allows RAZER to seek states for which no or only little training data exists (a,c). Means and standard deviations for (a) were computed over 5 runs. PETS overfits to a particular solution (b). In (b) and (c), the brightness of the dots is proportional to the time when they were first encountered.

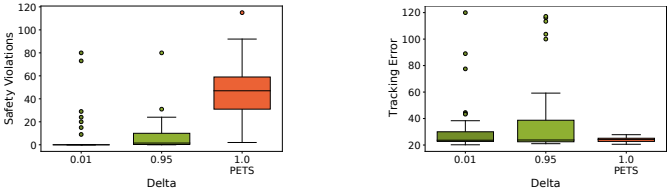


Figure 38: Safe planning vs. task-oriented planning in the *Solo8-LeanOverObject* environment with noisy actions. Left: number of safety violations for different values of δ (Eq. 35). Right: enforcing safety constraints causes slight reduction in tracking accuracy due to the fixed planning budget and the competing objectives of task and safety costs.

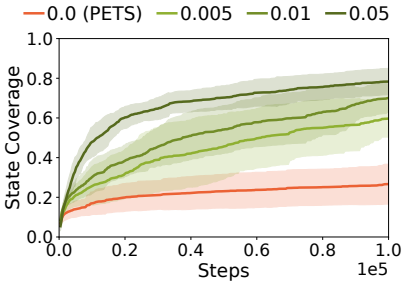


Figure 39: Exploration over time.

E

APPENDIX TO CHAPTER 6: N-APSP

In this supplementary material, we provide some insights regarding the count models used to discretize the space, and the hyperparameters used to train the goal-conditioned Q-functions, distances, and policies.

Index

E.1 COUNT MODELS

Our approach makes use of Locality Sensitive Hashing (LSH) to map high-dimensional, continuous inputs to discrete hash codes. The idea behind LHS is to hash states according to a certain similarity metrics. Our LHS algorithm of choice is a variation of SimHash [172], which uses angular distance between states as a similarity

metrics. The mapping function first projects the input (dim: n) to a lower dimensional space (dim: k):

$$\phi(s) = \left\lfloor \frac{Mg(s) + v}{\sigma} \right\rfloor$$

where $g(s)$ is an optional preprocessing function, which in our case is equal to the identity mapping, $M \in \mathbb{R}^{k,n}$ is a matrix with entries drawn from an i.i.d. standard Gaussian $\mathcal{N}(0,1)$ and $v \in \mathbb{R}^k$ is a random vector with uniform entries in the interval $[0, \sigma)$, where σ is a task-dependent constant. SimHash takes the sign of the random projection, while we round it up to the next integer as in [173]. This better reflects similarity under the Euclidean distance rather than angular distance and the probability of a collision (two inputs having the same hash) depends on their Euclidean distance.

Once we have computed the mapping $\phi(\cdot)$, we use it inside the Count-Min Sketch algorithm. Count-Min Sketch is designed to support memory-efficient counting without introducing too many overcounts. It maintains a separate count n_j for each hash function ϕ_j defined as $\phi_j(s) = \phi(s) \% p_j$, where p_j is a large prime number. Our implementation follows the one of Tang et al. [156] in the static hashing variant[1]. As in their paper, we consider $j = 6$ "buckets" equal to [999931, 999953, 999959, 999961, 999979, 999983], which we keep fixed for all the experiments. The number of counts is then $\min_{j=[1,6]} n_j(\phi_j(s))$.

E.1.1 *Granularity*

In Fig. 40 we show the effect of the dimension k on the granularity of the state space counting. The count model $N(s,a)$ was previously updated with the (s,a) transitions from a set of 2000 trajectories collected with *Ours+HER* policy, which amounts to 200k transitions.

1 https://github.com/openai/EPG/blob/master/epg/exploration.py

Algorithm 6: Count models update

Input : $\mathcal{D}_{\text{last}}$: replay buffer of most recently collected rollouts; n: input dimension, k: hashing dimension; σ: random vector interval range; I: training iterations

Output: single-state count model $N(s, a)$, state-pair count model $N(s, a, s')$

Initialize random matrix $M \in \mathbb{R}^{n,k}$ with entries from the standard Gaussian distribution $\mathcal{N}(0, 1)$
Initialize random vector $v \in \mathbb{R}^k$ with entries from uniform distribution on the interval $[0, \sigma)$
for $i = 0$ ***to*** $I-1$ // loop over training iterations
do
    for *(s, a)* ***in*** $\mathcal{D}_{\text{last}}$ // update $N(s, a)$
    do
        increase counts for (s, a)
    for τ ***in*** $\mathcal{D}_{\text{last}}$ // update $N(s, a, s')$
    do
        increase counts for all possible triples $(s_i, a_i, s_j) \in \tau$, with $j > i$ // $O(\text{len}(\tau)^2)$ triples

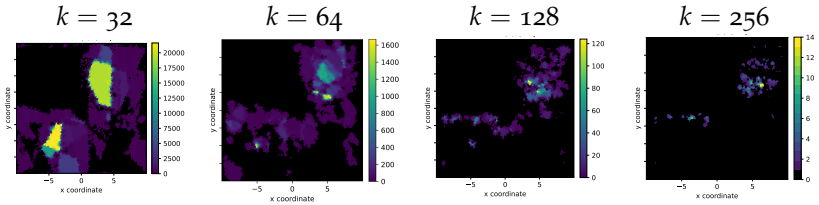
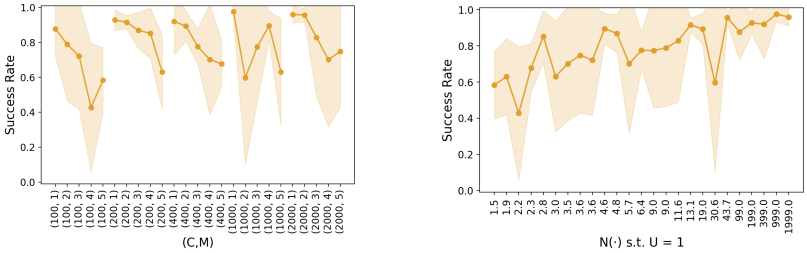


Figure 40: Granularity of the single-state count model $N(s, a)$ for the POINT MAZE environment depending on the dimension of the hash key k. Every plot represents the number of estimated counts of any input (s, a) where $s = (x, y, v_x = 0, v_y = 0)$ and $a = (0, 0)$, given that the count model previously visited a dataset of 200k points (trajectories obtained with *Ours + HER*).

E.1.2 *Sensitivity analysis*

As we can see from Fig. 41a, the performance on the Point Maze drops as we increase the scale at which the uncertainty decays (exponent M). At iteration 0, when no point has been visited, the count term is zero and the corresponding uncertainty is equal to its maximum value, which is the value of C. Then, for a fixed C, the higher the exponent M is, the faster the uncertainty decays with the counts and, consequently, the performance. This is because in the Point Maze environment we observe the "wormhole" phenomenon, also documented in [153], where the distance estimates are overly optimistic and do not take into account the presence of obstacles. As a result the agent thinks that it can go through them in order to reach a goal on the other side.

In this case, having a stronger uncertainty for the distance estimate is better, as confirmed in Fig. 41b, where we plot the success rate vs. the amount of counts necessary to have 1 unit of uncertainty. The plot shows how the performance increases with the number of counts, suggesting that a very pessimistic count model is a better choice for the Point Maze. The values we used for all of our experiments are C = 400 and M = 2.



(a) Grouped by the numerator C.

(b) Ordered by the count needed to have uncertainty = 1 timestep

Figure 41: Sensitivity analysis of the parameters regulating the decay of the count-based uncertainty $U[\cdot] = \frac{C}{(1+N(\cdot))^M}$.

E.2 GOAL AUGMENTATION FOR FETCH PICK AND PLACE

In Fig. 42b we report the FETCH PICK AND PLACE performance without the additional goal for the end effector which we used in the final experiments (Fig. 22). The reported value is in line with the results presented in the original HER paper [103]. Augmenting the goal space with an extra goal position for the end effector, equal to the one for the box, increases the sample efficiency of a $\sim 10x$ factor without introducing any explicit reward shaping.

E.3 IMPLEMENTATION DETAILS

E.3.1 *Hyperparameters*

All the networks (Q-functions, policies, distances) use the Adam optimizer. All the task horizons are equal to 50 time steps, apart from the 100 time steps used for the POINT MAZE task. Table 19 contains the parameters used for DDPG and SAC, together with the best k value for HER relabeling. Differently from the other methods, the best k for Ours+HER is lower; in fact, only 20% of the data ($k = 0.25$) gets relabeled with the achieved goal. We believe that this

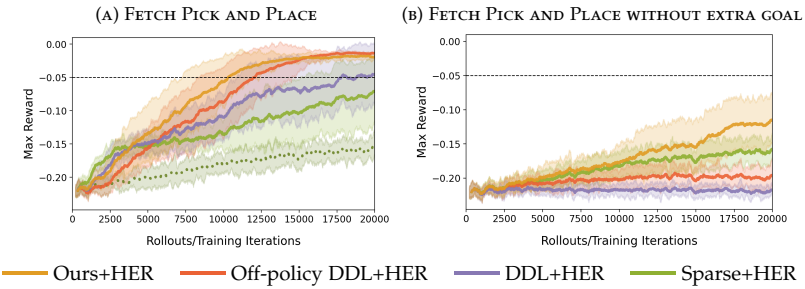


Figure 42: Performance comparison on the FETCH PICK AND PLACE task with and without additional goal for the end effector. The dark green dotted line of the left plot is the same as the solid green line of the right plot.

is due to the effect of the triangular loss: the distance to the desired goal better reflects the true shortest path and it is more informative than the distance to the achieved goal. In Table 20 we report the architecture and training parameters for the distance network, and the chosen hashing key k for every environment.

E.3.2 *Distance Learning*

The temporal loss in eq. 31a is trained on states belonging to the same trajectory. The rollouts are chosen randomly from the buffer, then we use a very basic procedure to sample the tuples (s_i, a_i, s_j). The index i is sampled uniformly from 0 to $T-1$, while the index j is sampled uniformly from $i+1$ to $T-1$.

Parameter	Value
Episode Length	50
Batch Size	256
Updates per Episode	100
Replay Buffer Size	$5e10^5$
Learning Rate	0.001
Discount Factor γ	0.98
Polyak Averaging	0.95
Action Noise (DDPG)	0.2
Action L_2 penalty (DDPG)	1
Random ϵ-Exploration	0.3
Q-Target Clipping (Sparse+HER)	[-50,0]
Q-Target Clipping (others)	[-1275,0]
Policy Network	3×256
Q-Function Network	3×256
Activation Function	ReLU
Weight Initialization	Xavier Uniform
Normalize Input	Yes
HER Replay Strategy	Future
HER Replay-k	see right table

Environment	Method	HER-k
Point Maze	Ours+HER	0.25
	DDL+HER	4
	Off-Policy DDL+HER	4
	Sparse+HER	None
Fetch Reach	Ours+HER	4
	DDL+HER	4
	Off-Policy DDL+HER	4
	Sparse+HER	4
Fetch Reach With Wall	Ours+HER	4
	DDL+HER	4
	Off-Policy DDL+HER	4
	Sparse+HER	4
Fetch Pick And Place	Ours+HER	4
	DDL+HER	4
	Off-Policy DDL+HER	4
	Sparse+HER	4
Fetch Pick And Place with Wall	Ours+HER	4
	DDL+HER	8
	Off-Policy DDL+HER	8
	Sparse+HER	4
Claw	Ours+HER	0
	DDL+HER	4
	Off-Policy DDL+HER	4
	Sparse+HER	4

Table 19: SAC & DDPG hyperparameters and best-k (HER) for each environment (grid search over {0, 0.25, 1, 4, 8}).

Parameter	Value
Distance Network	3×256
Distance Network (DDL)	2×256
Distance Network (DDL)	3×256 (Fetch Envs)
Batch Size	256
Updates per Episode	100
Replay Buffer Size	$1e10^6$
Replay Buffer Size (DDL)	$1e10^5$
Learning Rate	0.0003

Environment	k for for $N(s,a)$	k for for $N(s,a,s')$	σ
Point Maze	64	32	20
Fetch Reach (With Wall)	128	128	0.3
Fetch Pick And Place	32	64	0.3
Claw	32	64	0.3

Table 20: (left) Parameters for the distance network used in Ours+Her, DDL+HER, Off-Policy DDL + HER. (right) Chosen dimension of the hash state key for the presented environments. Left column is for the single state count model used in the temporal loss, while the right column is for the double state model used in the triangular loss.

LIST OF FIGURES

CHAPTER 2

CHAPTER 3

CHAPTER 5

CHAPTER 6

APPENDIX A

APPENDIX B

APPENDIX C

APPENDIX D

APPENDIX E

LIST OF TABLES

CHAPTER 3

APPENDIX B

APPENDIX C

APPENDIX D

APPENDIX E

BIBLIOGRAPHY

[1] Burrhus Frederic Skinner. *Science and human behavior.* New York: The Macmillan Company, 1953. URL: https://doi.org/10.1002/sce.37303805120.

[2] Ivan Pavlov. «Conditioned reflexes: An investigation of the physiological activity of the cerebral cortex.» In: *Oxford University Press* (1927). URL: https://doi.org/10.1002/sce.37303805120.

[3] Edward Lee Thorndike. *Animal intelligence; experimental studies.* New York: The Macmillan Company, 1911, p. 324. URL: https://doi.org/10.5962/bhl.title.55072.

[4] John O. Keefe and Lynn Nadel. *The hippocampus as a cognitive map*. Oxford University Press, 1978. URL: https://doi.org/10.1093/acprof:oso/9780199210862.003.0006.

[5] Edward C. Tolman. «Cognitive maps in rats and men.» In: *Psychological review* 55.4 (1948), p. 189. URL: https://doi.org/10.1037/h0061626.

[6] Ray J. Dolan and Peter Dayan. «Goals and Habits in the Brain.» In: *Neuron* 80.2 (2013), pp. 312–325. ISSN: 0896-6273. URL: https://doi.org/10.1016/j.neuron.2013.09.007.

[7] Nathaniel D. Daw, Yael Niv, and Peter Dayan. «Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control.» In: *Nature neuroscience* 8.12 (2005), pp. 1704–1711. URL: https://doi.org/10.1038/nn1560.

[8] S. Joe Qin and Thomas A. Badgwell. «A survey of industrial model predictive control technology.» In: *Control Engineering Practice* 11.7 (2003), pp. 733–764. ISSN: 0967-0661. URL: https://doi.org/10.1016/S0967-0661(02)00186-7.

[9] Eduardo F. Camacho and Carlos Bordons Alba. *Model predictive control*. Advanced Textbooks in Control and Signal Processing. Springer, 1999. DOI: https://doi.org/10.1007/978-1-4471-3398-8.

[10] Cristina Pinneri, Shambhuraj Sawant, Sebastian Blaes, Jan Achterhold, Joerg Stueckler, Michal Rolínek, and Georg Martius. «Sample-efficient Cross-Entropy Method for Real-time Planning.» In: *Conference on Robot Learning (CoRL)*. 2020. DOI: https://doi.org/10.48550/arXiv.2008.06389.

[11] Cristina Pinneri, Shambhuraj Sawant, Sebastian Blaes, and Georg Martius. «Extracting strong policies for robotics tasks from zero-order trajectory optimizers.» In: *International Conference on Learning Representations (ICLR)*. 2021. URL: https://openreview.net/pdf?id=Nc3TJqbcl3.

[12] Marin Vlastelica, Sebastian Blaes, Cristina Pinneri, and Georg Martius. «Risk-averse zero-order trajectory optimization.» In: *Conference on Robot Learning (CoRL)*. 2021. URL: https://openreview.net/pdf?id=WqUl7sNkDre.

[13] Cristina Pinneri, Georg Martius, and Andreas Krause. «Neural all-pairs shortest path for reinforcement learning.» In: *Deep Reinforcement Learning Workshop at the International Conference on Neural Information Processing Systems (NeurIPS)*. 2022. URL: https://openreview.net/pdf?id=w3jZFKGLrJ.

[14] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. Adaptive computation and machine learning. MIT press, 2018. URL: http://incompleteideas.net/book/RLbook2020.pdf.

[15] Volodymyr Mnih et al. «Human-level control through deep reinforcement learning.» In: *Nature* 518.7540 (2015), pp. 529–533. URL: https://doi.org/10.1038/nature14236.

[16] Jens Kober, J Andrew Bagnell, and Jan Peters. «Reinforcement learning in robotics: A survey.» In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1238–1274. URL: https://doi.org/10.1177/0278364913495721.

[17] David Silver et al. «Mastering the game of Go with deep neural networks and tree search.» In: *Nature* 529.7587 (2016), pp. 484–489. URL: https://doi.org/10.1038/nature16961.

[18] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. «Reinforcement learning for solving the vehicle routing problem.» In: *International Conference on Neural Information Processing Systems (NeurIPS)*. Vol. 31. 2018. URL: https://dl.acm.org/doi/10.5555/3327546.3327651.

[19] Ning Liu, Ying Liu, Brent Logan, Zhiyuan Xu, Jian Tang, and Yanzhi Wang. «Learning the dynamic treatment regimes from medical registry data through Deep Q-network.» In: *Scientific Reports* 9.1 (2019), p. 1495. URL: https://doi.org/10.1038/s41598-018-37142-0.

[20] Christopher Watkins and Peter Dayan. «Q-learning.» In: *Machine learning* 8 (1992), pp. 279–292. URL: https://doi.org/10.1007/BF00992698.

[21] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. «Asynchronous methods for deep reinforcement learning.» In: *International Conference on Machine Learning (ICML)*. 2016, pp. 1928–1937. URL: https://dl.acm.org/doi/10.5555/3045390.3045594.

[22] 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: *International Conference on Learning Representations (ICLR)*. 2016. URL: https://doi.org/10.48550/arXiv.1509.02971.

[23] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. «Soft Actor-Critic: Off-policy maximum entropy Deep Reinforcement Learning with a stochastic actor.» In: *International Conference on Machine Learning (ICML)*. 2018, pp. 1856–1865. URL: https://doi.org/10.48550/arXiv.1801.01290.

[24] Julian Schrittwieser et al. «Mastering Atari, Go, chess and shogi by planning with a learned model.» In: *Nature* 588.7839 (2020), pp. 604–609. URL: https://doi.org/10.1038/s41586-020-03051-4.

[25] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. «Trust region policy optimization.» In: *International Conference on Machine Learning (ICML)*. 2015, pp. 1889–1897. URL: https://dl.acm.org/doi/10.5555/3045118.3045319.

[26] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. «Proximal policy optimization algorithms.» In: 2017. URL: http://arxiv.org/abs/1707.06347.

[27] Ronald J. Williams. «Simple statistical gradient-following algorithms for connectionist reinforcement learning.» In: *Machine Learning* 8.3 (1992), pp. 229–256. URL: https://doi.org/10.1007/BF00992696.

[28] Yanwei Jia and Xun Yu Zhou. «Policy gradient and Actor-Critic learning in continuous time and space: Theory and algorithms.» In: *Journal of Machine Learning Research* 23.1 (2022). ISSN: 1532-4435. URL: https://www.jmlr.org/papers/volume23/21-1387/21-1387.pdf.

[29] Marc Peter Deisenroth and Carl Edward Rasmussen. «PILCO: A model-based and data-efficient approach to policy search.» In: *International Conference on Machine Learning (ICML)*. 2011, 465–472. URL: http://www.icml-2011.org/papers/323_icmlpaper.pdf.

[30] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. «When to trust your model: Model-based policy optimization.» In: vol. 32. 2019. URL: https://dl.acm.org/doi/10.5555/3454287.3455409.

[31] Steven M. LaValle. «Rapidly-exploring random trees : a new tool for path planning.» In: *The annual research report* (1998).

[32] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. «Deep reinforcement learning in a handful of trials using probabilistic dynamics models.» In: *International Conference on Neural Information Processing Systems (NeurIPS)*. 2018, pp. 4754–4765. URL: https://dl.acm.org/doi/10.5555/3327345.3327385.

[33] Carlos E. García, David M. Prett, and Manfred Morari. «Model predictive control: Theory and practice—A survey.» In: *Automatica* 25.3 (1989), pp. 335–348. ISSN: 0005-1098. URL: https://doi.org/10.1016/0005-1098(89)90002-2.

[34] Cunjia Liu, Wen-Hua Chen, and John Andrews. «Tracking control of small-scale helicopters using explicit nonlinear MPC augmented with disturbance observers.» In: *Control Engineering Practice* 20 (Mar. 2012), pp. 258–268. URL: https://doi.org/10.1016/j.conengprac.2011.10.015.

[35] Moses Bangura and Robert Mahony. «Real-time model predictive control for quadrotors.» In: *IFAC Proceedings Volumes* 47.3 (2014), pp. 11773–11780. URL: https://doi.org/10.3182/20140824-6-ZA-1003.00203.

[36] Paolo Falcone, Francesco Borrelli, Jahan Asgari, Hongtei Eric Tseng, and Davor Hrovat. «Predictive active steering control for autonomous vehicle systems.» In: *IEEE Transactions on Control Systems Technology* 15.3 (2007), pp. 566–580. URL: https://doi.org/10.1109/TCST.2007.894653.

[37] Francesco Borrelli, Paolo Falcone, Tamas Keviczky, Jahan Asgari, and Davor Hrovat. «MPC-based approach to active steering for autonomous vehicle systems.» In: *International Journal of Vehicle Autonomous Systems* 3.2-4 (2005), pp. 265–291. URL: https://dx.doi.org/10.1504/IJVAS.2005.008237.

[38] J.M. Maciejowski. *Predictive control with constraints*. Vol. 39. Harlow, UK: Prentice-Hall, Pearson Education Limited, Jan. 2003. URL: https://books.google.it/books?id=HV_Y58c7KiwC.

[39] David Q. Mayne, James B. Rawlings, Christopher V. Rao, and Pierre O.M. Scokaert. «Constrained model predictive control: Stability and optimality.» In: *Automatica* 36.6 (2000), pp. 789–814. ISSN: 0005-1098. URL: https://doi.org/10.1016/S0005-1098(99)00214-9.

[40] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou. «Information theoretic MPC for model-based reinforcement learning.» In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 1714–1721. URL: https://doi.org/10.1109/ICRA.2017.7989202.

[41] Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. «Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning.» In: *International Conference on Robotics and Automation (ICRA)*. Brisbane, Australia: IEEE Press, 2018, 7559–7566. URL: https://doi.org/10.1109/ICRA.2018.8463189.

[42] John H. Holland. *Adaptation in Natural and Artificial Systems*. Cambridge, MA, USA: MIT Press, 1975. URL: https://doi.org/10.1137/1018105.

[43] L.A. Rastrigin. «Random search in problems of optimization, identification and training of control systems.» In: *Journal of Cybernetics* 3.3 (1973), pp. 93–103. URL: https://doi.org/10.1080/01969727308546050.

[44] R. L. Anderson. «Recent Advances in Finding Best Operating Conditions.» In: *Journal of the American Statistical Association* 48.264 (1953), pp. 789–798. URL: https://doi.org/10.1080/01621459.1953.10501200.

[45] L.A. Rastrigin. «Extremal control by the method of random scanning.» In: *Automation and Remote Control*. Vol. 21. 1960, pp. 891–896.

[46] Reuven Rubinstein. «The Cross-Entropy Method for Combinatorial and Continuous Optimization.» In: *Methodology And Computing in Applied Probability* 1.2 (1999), pp. 127–190. URL: https://doi.org/10.1023/A:1010091220143.

[47] N. Hansen and A. Ostermeier. «Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation.» In: *International Conference on Evolutionary Computation (ECTA)*. 1996, pp. 312–317. URL: https://doi.org/10.1109/ICEC.1996.542381.

[48] Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. «Natural evolution strategies.» In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 949–980. URL: https://dl.acm.org/doi/10.5555/2627435.2638566.

[49] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. «Evolution strategies as a scalable alternative to reinforcement learning.» In: *ArXiv preprint* (2017). URL: https://doi.org/10.48550/arXiv.1703.03864.

[50] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. «Rainbow: Combining Improvements in Deep Reinforcement Learning.» In: *AAAI Conference on Artificial Intelligence*. 2017. URL: https://dl.acm.org/doi/10.5555/3504035.3504428.

[51] Aviral Kumar, Xue Bin Peng, and Sergey Levine. «Reward-Conditioned Policies.» In: *ArXiv preprint* (2019). URL: https://doi.org/10.48550/arXiv.1912.13465.

[52] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. «Decision Transformer: Reinforcement Learning via Sequence Modeling.» In: *International Conference on Neural Information Processing Systems (NeurIPS)*. Vol. 34. 2021, pp. 15084–15097. URL: https://

proceedings.neurips.cc/paper_files/paper/2021/file/7f489f642a0ddb10272b5c31057f0663-Paper.pdf.

[53] Yiming Ding, Carlos Florensa, Pieter Abbeel, and Mariano Phielipp. «Goal-conditioned Imitation Learning.» In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Vol. 32. 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/c8d3a760ebab631565f8509d84b3b3f1-Paper.pdf.

[54] Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. «End-to-end driving via conditional imitation learning.» In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 4693–4700. URL: https://doi.org/10.1109/ICRA.2018.8460487.

[55] Dibya Ghosh, Abhishek Gupta, Ashwin Reddy, Justin Fu, Coline Manon Devin, Benjamin Eysenbach, and Sergey Levine. «Learning to Reach Goals via Iterated Supervised Learning.» In: *International Conference on Learning Representations (ICLR)*. 2021. URL: https://openreview.net/forum?id=rALA0Xo6yNJ.

[56] Juergen Schmidhuber. «Reinforcement Learning Upside Down: Don't Predict Rewards–Just Map Them to Actions.» In: 2019. URL: https://doi.org/10.48550/arXiv.1912.02875.

[57] Andrew Y. Ng, Stuart Russell, et al. «Algorithms for inverse reinforcement learning.» In: *International Conference on Machine Learning (ICML)*. Vol. 1. 2000, p. 2. URL: https://ai.stanford.edu/~ang/papers/icml00-irl.pdf.

[58] Pieter Abbeel and Andrew Y Ng. «Apprenticeship learning via inverse reinforcement learning.» In: *International conference on Machine Learning (ICML)*. 2004, p. 1. URL: https://doi.org/10.1145/1015330.1015430.

[59] Stéphane Ross and Drew Bagnell. «Efficient reductions for imitation learning.» In: *International Conference on Artificial Intelligence and Statistics*. 2010, pp. 661–668. URL: https://proceedings.mlr.press/v9/ross10a.html.

[60] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. «A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning.» In: *International Conference on Artificial Intelligence and Statistics*. 2010. URL: https://proceedings.mlr.press/v15/ross11a.html.

[61] Dean A. Pomerleau. «ALVINN: An Autonomous Land Vehicle In a Neural Network.» In: *International Conference on Neural Information Processing Systems (NeurIPS)*. Vol. 1. 1988. URL: https://proceedings.neurips.cc/paper/1988/file/812b4ba287f5ee0bc9d43bbf5bbe87fb-Paper.pdf.

[62] Tingwu Wang and Jimmy Ba. «Exploring model-based planning with policy networks.» In: *International Conference on Learning Representations (ICLR)*. 2020. URL: https://openreview.net/forum?id=H1exf64KwH.

[63] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. «Learning Latent Dynamics for Planning from Pixels.» In: *International Conference on Machine Learning (ICML)*. 2019, pp. 2555–2565. URL: https://doi.org/10.48550/arXiv.1811.04551.

[64] Grady Williams, Andrew Aldrich, and Evangelos Theodorou. «Model predictive path integral control using covariance variable importance sampling.» In: *ArXiv preprint* abs/1509.01149 (2015). URL: https://doi.org/10.48550/arXiv.1509.01149.

[65] Arthur Richards and Jonathan P. How. «Robust variable horizon model predictive control for vehicle maneuvering.» In: *International Journal of Robust and Nonlinear Control* 16.7 (2006), pp. 333–351. URL: https://doi.org/10.1002/rnc.1059.

[66] Emanuel Todorov, Tom Erez, and Yuval Tassa. «Mujoco: A physics engine for model-based control.» In: *International Conference on Intelligent Robots and Systems (IROS)*. 2012, pp. 5026–5033. URL: https://doi.org/10.1109/IROS.2012.6386109.

[67] Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. «A tutorial on the cross-entropy method.» In: *Annals of operations research* 134.1 (2005), pp. 19–67. URL: https://doi.org/10.1007/s10479-005-5724-z.

[68] Nicolas Humphries et al. «Environmental context explains Lévy and Brownian movement patterns of marine predators.» In: *Nature* 465 (2010), pp. 1066–9. URL: https://doi.org/10.1038/nature09116.

[69] Michael F. Shlesinger, Joseph Klafter, and Y. M. Wong. «Random walks with infinite spatial and temporal moments.» In: *Journal of Statistical Physics* 27 (3 1982), pp. 499–512. URL: https://doi.org/10.1007/BF01011089.

[70] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. «OpenAI Gym.» In: *ArXiv preprint* (2016). URL: https://doi.org/10.48550/arXiv.1606.01540.

[71] J. Timmer and M. Koenig. «On generating power law noise.» In: *Astronomy and Astrophysics* 300 (1995), pp. 707–710. URL: https://ui.adsabs.harvard.edu/abs/1995A&A...300..707T.

[72] William T Cochran, James W Cooley, David L Favin, Howard D Helms, Reginald A Kaenel, William W Lang, George C Maling, David E Nelson, Charles M Rader, and Peter D Welch. «What is the fast Fourier transform?» In: *Proceedings of the IEEE* 55.10 (1967), pp. 1664–1674. URL: https://doi.org/10.1109/PROC.1967.5957.

[73] 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: *Robotics: Science and Systems (RSS)*. 2018. URL: https://doi.org/10.48550/arXiv.1709.10087.

[74] I. Rechenberg. «Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.» PhD thesis. TU Berlin, 1971. URL: https://doi.org/10.1002/fedr.19750860506.

[75] Krzysztof Choromanski, Mark Rowland, Vikas Sindhwani, Richard Turner, and Adrian Weller. «Structured evolution with compact architectures for scalable policy optimization.» In: *International Conference on Machine Learning (ICML)*. 2018, pp. 970–978. URL: https://doi.org/10.48550/arXiv.1804.02395.

[76] Horia Mania, Aurelia Guy, and Benjamin Recht. «Simple random search of static linear policies is competitive for reinforcement learning.» In: *International Conference on Neural Information Processing Systems (NeurIPS)*. 2018, pp. 1800–1809. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/7634ea65a4e6d9041cfd3f7de18e334a-Paper.pdf.

[77] Shauharda Khadka and Kagan Tumer. «Evolution-guided policy gradient in reinforcement learning.» In: *International Conference on Neural Information Processing Systems (NeurIPS)*. 2018, pp. 1188–1200. URL: https://dl.acm.org/doi/10.5555/3326943.3327053.

[78] Homanga Bharadhwaj, Kevin Xie, and Florian Shkurti. «Model-predictive control via cross-entropy and gradient-based optimization.» In: *Conference on Learning for Dynamics and Control Learning (L4DC)*. Ed. by Alexandre M. Bayen, Ali Jadbabaie, George Pappas, Pablo A. Parrilo, Benjamin Recht, Claire Tomlin, and Melanie Zeilinger. Vol. 120. PMLR, 2020,

pp. 277–286. URL: https://doi.org/10.48550/arXiv.2004.08763.

[79] Grady Williams, Paul Drews, Brian Goldfain, James M. Rehg, and Evangelos Theodorou. «Aggressive driving with model predictive path integral control.» In: *International Conference on Robotics and Automation (ICRA)* (2016), pp. 1433–1440. URL: https://doi.org/10.1109/ICRA.2016.7487277.

[80] Reuven Y. Rubinstein and Dirk P. Kroese. *The cross entropy method: A unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Berlin, Heidelberg: Springer, 2004. URL: https://doi.org/10.1007/978-1-4757-4321-0.

[81] Zdravko Botev, Dirk Kroese, Reuven Rubinstein, and Pierre L'Ecuyer. «The Cross-Entropy method for optimization (Chapter 3).» In: *Handbook of Statistics*. Vol. 31. Elsevier, 2013, pp. 35–59. URL: https://doi.org/10.1016/B978-0-444-53859-8.00003-5.

[82] L. Margolin. «On the convergence of the cross-entropy method.» In: *Annals of Operations Research* 134 (2005), pp. 201–214. URL: https://doi.org/10.1007/s10479-005-5731-0.

[83] Aloïs Pourchot and Olivier Sigaud. «CEM-RL: Combining evolutionary and gradient-based methods for policy search.» In: *International Conference on Learning Representations (ICLR)*. 2018. URL: https://doi.org/10.48550/arXiv.1810.01222.

[84] Marin Kobilarov. «Cross-entropy motion planning.» In: *International Journal of Robotic Research (IJRR)* 31 (2012), pp. 855–871. URL: https://doi.org/10.1177/0278364912444543.

[85] L. Čehovin, M. Kristan, and A. Leonardis. «An adaptive coupled-layer visual model for robust visual tracking.» In: *International Conference on Computer Vision (ICCV)*. 2011, pp. 1363–1370. URL: https://doi.org/10.1109/ICCV.2011.6126390.

[86] Shili Lin and Jie Ding. «Integration of ranked lists via cross-entropy Monte Carlo with applications to mRNA and microRNA studies.» In: *Biometrics* 65 (2008), pp. 9–18. URL: https://doi.org/10.1111/j.1541-0420.2008.01044.x.

[87] Kin-Ping Hui, Nigel Bean, Miro Kraetzl, and Dirk P. Kroese. «The cross-entropy method for network reliability estimation.» In: *Annals of Operations Research* 134 (2005), pp. 101–118. URL: https://doi.org/10.1007/s10479-005-5726-x.

[88] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. «Benchmarking deep reinforcement learning for continuous control.» In: *International Conference on Machine Learning (ICML)*. 2016. URL: https://doi.org/10.48550/arXiv.1604.06778.

[89] Brandon Amos and Denis Yarats. «The differentiable cross-entropy method.» In: *International Conference on Machine Learning (ICML)*. 2020. URL: https://doi.org/10.48550/arXiv.1909.12830.

[90] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, SM Eslami, et al. «Emergence of locomotion behaviours in rich environments.» In: *ArXiv preprint* (2017). URL: https://doi.org/10.48550/arXiv.1707.02286.

[91] Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. «Learning dexterous in-hand manipulation.» In: *The International Journal of Robotics Research* 39.1 (2020), pp. 3–20. URL: https://doi.org/10.1177/0278364919887447.

[92] Yilun Du, Toru Lin, and Igor Mordatch. «Model based planning with energy based models.» In: *Conference on Robot Learning (CoRL)*. 2019. URL: https://doi.org/10.48550/arXiv.1909.06878.

[93] Cristina Pinneri, Shambhuraj Sawant, Sebastian Blaes, Jan Achterhold, Joerg Stueckler, Michal Rolinek, and Georg Martius. «Sample-efficient cross-entropy method for real-time planning.» In: *Conference on Robot Learning (CoRL)*. 2020. DOI: https://doi.org/10.48550/arXiv.2008.06389.

[94] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. «Deep reinforcement learning that matters.» In: *Conference On Artificial Intelligence (AAAI)* (2018). URL: https://doi.org/10.1609/aaai.v32i1.11694.

[95] Kendall Lowrey, Aravind Rajeswaran, Sham Kakade, Emanuel Todorov, and Igor Mordatch. «Plan online, learn offline: Efficient learning and exploration via model-based control.» In: *International Conference on Learning Representations (ICLR)*. 2019. URL: https://openreview.net/forum?id=Byey7n05FQ.

[96] Sergey Levine and Vladlen Koltun. «Guided policy search.» In: *International Conference on Machine Learning (ICML)*. 2013. URL: https://proceedings.mlr.press/v28/levine13.html.

[97] Sergey Levine and Pieter Abbeel. «Learning neural network policies with guided policy search under unknown dynamics.» In: *International Conference on Neural Information Processing Systems (NeurIPS*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. 2014, pp. 1071–1079. URL: http://papers.nips.cc/paper/5444-learning-neural-network-policies-with-guided-policy-search-under-unknown-dynamics.pdf.

[98] Igor Mordatch and Emanuel Todorov. «Combining the benefits of function approximation and trajectory optimization.» In: *Robotics: Science and Systems*. 2014. URL: http://dx.doi.org/10.15607/RSS.2014.X.052.

[99] Igor Mordatch, Kendall Lowrey, Galen Andrew, Zoran Popovic, and Emanuel V. Todorov. «Interactive control of diverse complex characters with neural networks.» In: *In-*

ternational Conference on Neural Information Processing Systems (NeurIPS). 2015, pp. 3132–3140. URL: https://proceedings.neurips.cc/paper_files/paper/2015/file/2612aa892d962d6f8056b195ca6e550d-Paper.pdf.

[100] Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. «Learning deep control policies for autonomous aerial vehicles with MPC-guided policy search.» In: *International Conference on Robotics and Automation (ICRA)*. 2016, pp. 528–535. URL: https://doi.org/10.1109/ICRA.2016.7487175.

[101] Gregory Kahn, Tianhao Zhang, Sergey Levine, and Pieter Abbeel. «Plato: Policy learning using adaptive trajectory optimization.» In: *International Conference on Robotics and Automation (ICRA)*. 2017. URL: https://doi.org/10.1109/ICRA.2017.7989379.

[102] Wen Sun, Geoffrey J. Gordon, Byron Boots, and J. Andrew Bagnell. «Dual policy iteration.» In: *International Conference on Neural Information Processing Systems (NeurIPS)*. 2018. URL: https://doi.org/10.48550/arXiv.1805.10755.

[103] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. «Hindsight Experience Replay.» In: *International Conference on Neural Information Processing Systems (NeurIPS)*. Vol. 30. 2017, pp. 5048–5058. URL: https://proceedings.neurips.cc/paper/2017/file/453fadbd8a1a3af50a9df4df899537b5-Paper.pdf.

[104] Yuxiang Yang, Ken Caluwaerts, Atil Iscen, Tingnan Zhang, Jie Tan, and Vikas Sindhwani. «Data efficient reinforcement learning for legged robots.» In: *Conference on Robot Learning (CoRL)* (2019). URL: https://doi.org/10.48550/arXiv.1907.03613.

[105] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. «Imitation learning: A survey of learning

methods.» In: *ACM Computing Surveys (CSUR)* 50.2 (2017), pp. 1–35. URL: https://doi.org/10.1145/3054912.

[106] Athanasios S Polydoros and Lazaros Nalpantidis. «Survey of model-based reinforcement learning: Applications on robotics.» In: *Journal of Intelligent & Robotic Systems* 86.2 (2017), pp. 153–173. URL: https://doi.org/10.1007/s10846-017-0468-y.

[107] Stephen C. Hora. «Aleatory and epistemic uncertainty in probability elicitation with an example from hazardous waste management.» In: *Reliability Engineering & System Safety* 54.2 (1996), pp. 217–223. URL: https://doi.org/10.1016/S0951-8320(96)00077-4.

[108] Armen Der Kiureghian and Ove Ditlevsen. «Aleatory or epistemic? Does it matter?» In: *Structural Safety*. Risk Acceptance and Risk Communication 31.2 (2009), pp. 105–112. ISSN: 0167-4730. URL: https://doi.org/10.1016/j.strusafe.2008.06.020.

[109] Ian Lenz, Ross A Knepper, and Ashutosh Saxena. «DeepMPC: Learning deep latent features for model predictive control.» In: *Robotics: Science and Systems*. 2015. URL: https://doi.org/10.15607/RSS.2015.XI.012.

[110] Justin Fu, Sergey Levine, and Pieter Abbeel. «One-shot learning of manipulation skills with online dynamics adaptation and neural network priors.» In: *International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 4019–4026. URL: https://doi.org/10.1109/IROS.2016.7759592.

[111] Yarin Gal, Rowan McAllister, and Carl Edward Rasmussen. «Improving PILCO with Bayesian neural network dynamics models.» In: *Data-Efficient Machine Learning Workshop at the International Conference of Machine Learning (ICML)*. Vol. 4. 34. 2016, p. 25. URL: http://www.cs.ox.ac.uk/people/yarin.gal/website/PDFs/DeepPILCO.pdf.

[112] Juš Kocijan, Roderick Murray-Smith, Carl E. Rasmussen, and Agathe Girard. «Gaussian process model based predictive control.» In: *American control conference*. Vol. 3. 2004, pp. 2214–2219. URL: https://doi.org/10.23919/ACC.2004.1383790.

[113] Duy Nguyen-Tuong, Jan Peters, and Matthias Seeger. «Local gaussian process regression for real time online model learning and control.» In: *International Conference on Neural Information Processing Systems (NeurIPS)*. 2008, pp. 1193–1200. URL: https://proceedings.neurips.cc/paper_files/paper/2008/file/01161aaa0b6d1345dd8fe4e481144d84-Paper.pdf.

[114] Alexandra Grancharova, Juš Kocijan, and Tor A Johansen. «Explicit stochastic predictive control of combustion plants based on Gaussian process models.» In: *Automatica* 44.6 (2008), pp. 1621–1631. URL: https://doi.org/10.1016/j.automatica.2008.04.002.

[115] Marc Peter Deisenroth, Dieter Fox, and Carl E. Rasmussen. «Gaussian processes for data-efficient learning in robotics and control.» In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.2 (2013), pp. 408–423. URL: https://doi.org/10.1109/TPAMI.2013.218.

[116] Sanket Kamthe and Marc Deisenroth. «Data-efficient reinforcement learning with probabilistic model predictive control.» In: *International Conference on Artificial Intelligence and Statistics*. 2018, pp. 1701–1710. URL: https://doi.org/10.48550/arXiv.1706.06491.

[117] Carl E. Rasmussen and Malte Kuss. «Gaussian Processes in Reinforcement Learning.» In: *International Conference on Neural Information Processing Systems (NeurIPS)*. 2003. URL: https://proceedings.neurips.cc/paper_files/paper/2003/file/7993e11204b215b27694b6f139e34ce8-Paper.pdf.

[118] Carl E. Rasmussen and Christopher K.I. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. Cambridge, MA, USA: MIT Press, 2006. URL: https://doi.org/10.1007/978-3-540-28650-9_4.

[119] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. «Model-ensemble trust-region policy optimization.» In: *International Conference on Learning Representations (ICLR)*. 2018. URL: https://openreview.net/forum?id=SJJinbWRZ.

[120] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. «Deep exploration via bootstrapped DQN.» In: *International Conference on Neural Information Processing Systems (NeurIPS)*. 2016, 4033–4041. URL: https://arxiv.org/pdf/1602.04621.pdf.

[121] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. «Simple and scalable predictive uncertainty estimation using deep ensembles.» In: *International Conference on Neural Information Processing Systems (NeurIPS)*. 2017, 6405–6416. URL: https://proceedings.neurips.cc/paper/2017/file/9ef2ed4b7fd2c810847ffa5fa85bce38-Paper.pdf.

[122] Sebastian Curi, Felix Berkenkamp, and Andreas Krause. «Efficient model-based reinforcement learning through optimistic policy search and planning.» In: *International Conference on Neural Information Processing Systems (NeurIPS)*. 2020. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/a36b598abb934e4528412e5a2127b931-Paper.pdf.

[123] Manfred Morari and Jay H Lee. «Model predictive control: past, present and future.» In: *Computers & Chemical Engineering* 23.4-5 (1999), pp. 667–682. URL: https://doi.org/10.1016/S0098-1354(98)00301-9.

[124] Oliver Mihatsch and Ralph Neuneier. «Risk-sensitive reinforcement learning.» In: *Machine learning* 49.2 (2002), pp. 267–290. URL: https://doi.org/10.1023/A:1017940631555.

[125] Javier Garcıa and Fernando Fernández. «A comprehensive survey on safe reinforcement learning.» In: *Journal of Machine Learning Research* 16.1 (2015), pp. 1437–1480. URL: https://www.jmlr.org/papers/volume16/ailon15a/ailon15a.pdf.

[126] Stefan Depeweg, Jose-Miguel Hernandez-Lobato, Finale Doshi-Velez, and Steffen Udluft. «Decomposition of uncertainty in Bayesian deep learning for efficient and risk-sensitive learning.» In: *International Conference on Machine Learning (ICML)*. 2018, pp. 1184–1193. URL: https://doi.org/10.48550/arXiv.1710.07283.

[127] Ermano Arruda, Michael J. Mathew, Marek Kopicki, M. Mistry, M. Azad, and J. Wyatt. «Uncertainty averse pushing with model predictive path integral control.» In: *International Conference on Humanoid Robotics (Humanoids)* (2017), pp. 497–502. URL: https://doi.org/10.1109/HUMANOIDS.2017.8246918.

[128] Keuntaek Lee, Gabriel Nakajima An, Viacheslav Zakharov, and Evangelos A. Theodorou. «Perceptual attention-based predictive control.» In: *Conference on Robot Learning (CoRL)*. Vol. 100. 2020, pp. 220–232. URL: http://proceedings.mlr.press/v100/lee20b.html.

[129] Ian Abraham, Ankur Handa, Nathan Ratliff, Kendall Lowrey, Todd D. Murphey, and Dieter Fox. «Model-based generalization under parameter uncertainty using path integral control.» In: *Robotics and Automation Letters* 5.2 (2020), pp. 2864 –2871. URL: https://doi.org/10.1109/LRA.2020.2972836.

[130] William R. Clements, Benoit-Marie Robaglia, Bastien Van Delft, Reda Bahi Slaoui, and Sébastien Toth. «Estimating risk and uncertainty in deep reinforcement learning.» In: *Workshop on Uncertainty & Robustness in Deep Learning at the In-*

ternational Conference on Machine Learning (ICML) (2020). URL: https://doi.org/10.48550/arXiv.1905.09638.

[131] Jianyi Zhang and P. Weng. «Safe distributional reinforcement learning.» In: *nternational Conference on Distributed Artificial Intelligence (DAI)*. 2021. URL: https://doi.org/10.1007/978-3-030-94662-3_8.

[132] Andrew Y Ng, Daishi Harada, and Stuart Russell. «Policy invariance under reward transformations: Theory and application to reward shaping.» In: *International Conference on Machine Learning (ICML)*. Vol. 99. 1999, pp. 278–287. URL: https://people.eecs.berkeley.edu/~russell/papers/icml99-shaping.pdf.

[133] Ashique Rupam Mahmood, Dmytro Korenkevych, Brent Komer, and James Bergstra. «Setting up a reinforcement learning task with a real-world robot.» In: *nternational Conference on Intelligent Robots and Systems (IROS)* (2018), pp. 4635–4640. URL: https://doi.org/10.1109/IROS.2018.8593894.

[134] Alexander Pan, Kush Bhatia, and Jacob Steinhardt. «The effects of reward misspecification: Mapping and mitigating misaligned models.» In: *International Conference on Learning Representations (ICLR)*. 2022. URL: https://openreview.net/forum?id=JYtwGwIL7ye.

[135] Kristian Hartikainen, Xinyang Geng, Tuomas Haarnoja, and Sergey Levine. «Dynamical distance learning for semi-supervised and unsupervised skill discovery.» In: *International Conference on Learning Representations (ICLR)* (2020). URL: https://doi.org/10.48550/arXiv.1907.08225.

[136] Leslie Pack Kaelbling. «Learning to achieve goals.» In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 1993. URL: https://people.csail.mit.edu/lpk/papers/ijcai93.ps.

[137] Robert W. Floyd. «Algorithm 97: Shortest path.» In: *Communications of the Association for Computing Machinery* 5.6 (1962), p. 345. URL: https://doi.org/10.1145/367766.368168.

[138] Tom Jurgenson, Or Avner, Edward Groshev, and Aviv Tamar. «Sub-goal trees: A framework for goal-based reinforcement learning.» In: *International Conference on Machine Learning (ICML)*. Vol. 119. 2020, pp. 5020–5030. URL: https://proceedings.mlr.press/v119/jurgenson20a.html.

[139] Richard Bellman. «Dynamic Programming.» In: *Science* 153.3731 (1966), pp. 34–37. URL: https://doi.org/10.1126/science.153.3731.34.

[140] Edsger W. Dijkstra. «A note on two problems in connexion with graphs.» In: *Numerische Mathematik* 1 (1959), pp. 269–271. URL: https://doi.org/10.1145/3544585.3544600.

[141] Richard Bellman. «On a routing problem.» In: *Quarterly of applied mathematics* 16.1 (1958), pp. 87–90. URL: https://doi.org/10.1090/qam/102435.

[142] Moshe Sniedovich. «Dijkstra's algorithm revisited: the dynamic programming connexion.» In: *Control and Cybernetics* 35 (2006), pp. 599–620. URL: http://matwbn.icm.edu.pl/ksiazki/cc/cc35/cc3536.pdf.

[143] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. «Universal Value Function Approximators.» In: *International Conference on Machine Learning (ICML)*. Vol. 37. 2015, pp. 1312–1320. URL: https://proceedings.mlr.press/v37/schaul15.html.

[144] Vikas Dhiman, Shurjo Banerjee, Jeffrey Mark Siskind, and Jason J. Corso. «Floyd-Warshall reinforcement learning: Learning from past experiences to reach new goals.» In: *ArXiv preprint* (2018). URL: https://doi.org/10.48550/arXiv.1809.09318.

[145] Jose A Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. «Rudder: Return decomposition for delayed rewards.» In: *International Conference on Neural Information Processing Systems (NeurIPS)* 32 (2019). URL: https://

proceedings.neurips.cc/paper_files/paper/2019/file/16105fb9cc614fc29e1bda00dab60d41-Paper.pdf.

[146] Marek Grzes and Daniel Kudenko. «Plan-based reward shaping for reinforcement learning.» In: *International IEEE Conference Intelligent Systems*. Vol. 2. 2008, pp. 10–22. URL: https://doi.org/10.1109/IS.2008.4670492.

[147] Jette Randløv and Preben Alstrøm. «Learning to drive a bicycle using reinforcement learning and shaping.» In: *International Conference on Machine Learning (ICML)*. Vol. 98. 1998, pp. 463–471. URL: https://dl.acm.org/doi/10.5555/645527.757766.

[148] Jürgen Schmidhuber. «A possibility for implementing curiosity and boredom in model-building neural controllers.» In: *International Conference on Simulation of Adaptive Behavior*. 1991, pp. 222–227. URL: https://doi.org/10.7551/mitpress/3115.003.0030.

[149] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. «Curiosity-driven exploration by self-supervised prediction.» In: *International Conference on Machine Learning (ICML)*. 2017, pp. 2778–2787. URL: https://doi.org/10.1109/CVPRW.2017.70.

[150] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. «Vime: Variational information maximizing exploration.» In: *International Conference on Neural Information Processing Systems (NeurIPS)* 29 (2016). URL: https://proceedings.neurips.cc/paper/2016/file/abd815286ba1007abfbb8415b83ae2cf-Paper.pdf.

[151] Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. «Semi-parametric topological memory for navigation.» In: *International Conference on Learning Representations (ICLR)*. 2018. URL: https://openreview.net/forum?id=SygwwGbRW.

[152] Dibya Ghosh, Abhishek Gupta, and Sergey Levine. «Learning actionable representations with goal conditioned policies.» In: *International Conference on Learning Representations (ICLR)*. 2019. URL: https://openreview.net/forum?id=Hye9lnCct7.

[153] Ben Eysenbach, Russ R Salakhutdinov, and Sergey Levine. «Search on the Replay Buffer: Bridging Planning and Reinforcement Learning.» In: 32 (2019). Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/5c48ff18e0a47baaf81d8b8ea51eec92-Paper.pdf.

[154] Carlos Florensa, Jonas Degrave, Nicolas Heess, Jost Tobias Springenberg, and Martin Riedmiller. «Self-supervised learning of image embedding for continuous control.» In: *ArXiv preprint* (2019). URL: https://doi.org/10.48550/arXiv.1901.00943.

[155] Stephen Tian, Suraj Nair, Frederik Ebert, Sudeep Dasari, Benjamin Eysenbach, Chelsea Finn, and Sergey Levine. «Model-based visual planning with self-supervised functional distances.» In: *International Conference on Learning Representations (ICLR)*. 2021. URL: https://openreview.net/forum?id=UcoXdfrORC.

[156] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. «#Exploration: A study of count-based exploration for deep reinforcement learning.» In: *International Conference on Neural Information Processing Systems (NeurIPS)*. 2017, pp. 2753–2762. URL: https://proceedings.neurips.cc/paper/2017/hash/3a20f62a0af1aa152670bab3c602feed-Abstract.html.

[157] Tabish Rashid, Bei Peng, Wendelin Boehmer, and Shimon Whiteson. «Optimistic Exploration even with a Pessimistic Initialisation.» In: *International Conference on Learning Rep-*

resentations (ICLR). 2020. URL: https://openreview.net/forum?id=r1xGP6VYwH.

[158] Michael Ahn, Henry Zhu, Kristian Hartikainen, Hugo Ponte, Abhishek Gupta, Sergey Levine, and Vikash Kumar. «Robel: Robotics benchmarks for learning with low-cost robots.» In: *Conference on Robot learning (CoRL)*. 2020, pp. 1300–1313. URL: https://doi.org/10.48550/arXiv.1909.11639.

[159] Tuomas Haarnoja, Aurick Zhou, P. Abbeel, and Sergey Levine. «Soft Actor-Critic: Off-policy maximum entropy Deep Reinforcement Learning with a stochastic actor.» In: *International Conference on Machine Learning (ICML)*. 2018, pp. 1856–1865. URL: https://doi.org/10.48550/arXiv.1801.01290.

[160] Mingyuan Zhong, Mikala Johnson, Yuval Tassa, Tom Erez, and Emanuel Todorov. «Value function approximation and model predictive control.» In: *Symposium on adaptive dynamic programming and reinforcement learning (ADPRL)*. 2013, pp. 100–107. URL: https://homes.cs.washington.edu/~todorov/papers/ZhongADPRL13.pdf.

[161] Yuchen Cui, David Isele, Scott Niekum, and Kikuo Fujimura. «Uncertainty-aware data aggregation for deep imitation learning.» In: *International Conference on Robotics and Automation (ICRA)* (2019), pp. 761–767. URL: http://arxiv.org/abs/1905.02780.

[162] Ziyu Wang et al. «Critic regularized regression.» In: *International Conference on Neural Information Processing Systems (NeurIPS)*. Vol. 33. 2020, pp. 7768–7778. URL: https://proceedings.neurips.cc/paper/2020/file/588cb956d6bbe67078f29f8de420a13d-Paper.pdf.

[163] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. «Conservative q-learning for offline reinforcement learning.» In: *International Conference on Neural Information Processing Systems (NeurIPS)*. Vol. 33. 2020, pp. 1179–1191.

URL: https://proceedings.neurips.cc/paper/2020/file/0d2b2061826a5df3221116a5085a6052-Paper.pdf.

[164] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. «Graph neural networks: A review of methods and applications.» In: *AI Open* 1 (2020), pp. 57–81. ISSN: 2666-6510. URL: https://doi.org/10.1016/j.aiopen.2021.01.001.

[165] Cansu Sancaktar, Sebastian Blaes, and Georg Martius. «Curious exploration via structured world models yields zero-shot object manipulation.» In: *International Conference on Neural Information Processing Systems (NeurIPS)*. 2022. URL: https://openreview.net/forum?id=NnuYZ1el24C.

[166] Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. «dm_control: Software and tasks for continuous control.» In: *Software Impacts* 6 (2020), p. 100022. URL: https://www.sciencedirect.com/science/article/pii/S2665963820300099.

[167] Onno Eberhard, Jakob Hollenstein, Cristina Pinneri, and Georg Martius. «Pink noise is all you need: Colored noise exploration in Deep RL.» In: *International Conference on Learning Representations (ICLR)*. Vol. 10. 2023, p. 2. URL: https://openreview.net/pdf?id=hQ9V5QN27eS.

[168] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. «Practical Bayesian optimization of machine learning algorithms.» In: *International Conference on Neural Information Processing Systems (NeurIPS)*. 2012, 2951–2959. URL: https://papers.nips.cc/paper_files/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf.

[169] Ruben Fossion, E. Landa, P. Stránský, Victor Velazquez, Juan Carlos Lopez Vieyra, I. Garduño, D. García, and Alejandro Frank. «Scale invariance as a symmetry in physical and biological systems: Listening to photons, bubbles and

heartbeats.» In: *AIP Conference Proceedings* 1323 (Dec. 2010), pp. 74–90. URL: https://doi.org/10.1063/1.3537868.

[170] Felix Grimminger, Avadesh Meduri, Majid Khadiv, Julian Viereck, Manuel Wüthrich, Maximilien Naveau, Vincent Berenz, Steve Heim, Felix Widmaier, Thomas Flayols, et al. «An open torque-controlled modular robot architecture for legged locomotion research.» In: *Robotics and Automation Letters* 5.2 (2020), pp. 3650–3657. URL: http://dx.doi.org/10.1109/LRA.2020.2976639.

[171] Min Wen and Ufuk Topcu. «Constrained cross-entropy method for safe reinforcement learning.» In: *International Conference on Neural Information Processing Systems (NeurIPS)*. Vol. 31. 2018. URL: https://proceedings.neurips.cc/paper/2018/file/34ffeb359a192eb8174b6854643cc046-Paper.pdf.

[172] Moses Charikar. «Similarity estimation techniques from rounding algorithms.» In: *ACM Symposium on Theory of Computing (STOC)*. 2002. URL: https://doi.org/10.1145/509907.509965.

[173] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. «Locality-sensitive hashing scheme based on p-stable distributions.» In: *Symposium on Computational Geometry (SCG)*. 2004. URL: https://doi.org/10.1145/997817.997857.

CURRICULUM VITAE

PERSONAL DATA

Name	Cristina Pinneri
Date of Birth	January 23, 1993
Place of Birth	Reggio Calabria, Italy
Citizen of	Italy

EDUCATION

2018 – Today	**PhD Student at the Max Planck ETH Center for Learning Systems** Tübingen, Germany Zürich, Switzerland
2016 – 2017	**MSc. - Master Degree in Nanotechnologies for ICT's** Polytechnic University of Turin, Italy
2014 – 2016	**MSc. - International Master Degree in Physics of Complex Systems** Polytechnic University of Turin, Italy International School for Advanced Studies (SISSA), Italy International Center for Theoretical Physics (ICTP), Italy Université Pierre et Marie Curie (UPMC), France

2011 – 2014 **Diploma - Superior Graduate School in Physics**
Scuola Superiore di Catania[1] (SSC)
Summa Cum Laude

2005 – 2015 **MA.[2] - Master Degree in Piano**
Conservatory of Reggio Calabria, Italy

2011 – 2014 **BSc. - Bachelor in Physics**
University of Catania, Italy
Summa cum laude

EXPERIENCE

2022 **Research Intership at Google Deepmind**
London, UK

2021 **Research Intership at Meta AI**
San Francisco, US

2021 **Teaching assistant for Introduction to Machine Learning**
ETH, Zurich

2020 **Teaching assistant for Probabilistic Artificial Intelligence**
ETH, Zurich

1 The Scuola Superiore di Catania (SSC) is a special structure for higher education of the University of Catania, characterized by selective admission tests and strict performance requirements. It was instituted to develop the skills of young students, through additional classes and beforehand research initiation. Every student at SSC is a student at the University of Catania at the same time. Among other requirements, a SSC student has to have the highest marks in all disciplines and has to defend an additional final project thesis.
See also http://www.scuolasuperiorecatania.it

2 Qualification received by *"Higher Education Institutions for Fine Arts, Music and Dance - AFAM"*

PUBLICATIONS

2022 **Pinneri, C.** and Martius, G., and Krause, A., Neural all-pairs shortest path for Reinforcement Learning, In: NeurIPS 2022 Workshop on *DeepRL* (NeurIPS DeepRL 2022)

2022 Eberhard, O. and Hollenstein, J. and **Pinneri, C.** and Martius, G., Pink Noise Is All You Need: Colored Noise Exploration in Deep Reinforcement Learning, In: *Proc. of the International Conference on Learning Representations* (ICLR 2023)

2021 Vlastelica, M. and Blaes, S. and **Pinneri, C.** and Martius, G., Risk-Averse Zero-Order Trajectory Optimization, In: *Proc. of the Conference on Robot Learning* (CoRL 2021)

2021 **Pinneri, C.** et al., Extracting Strong Policies for Robotics Tasks from zero-order trajectory optimizers, In: *Proc. of International Conference on Learning Representations* (ICLR 2021)

2020 **Pinneri, C.** et al., Sample-efficient Cross-Entropy Method for Real-time Planning In: *Proc. of the Conference on Robot Learning* (CoRL 2020)

2018 **Pinneri, C.** and Martius, G., Systematic self-exploration of behaviors for robots in a dynamical systems framework In: *Proc. Artificial Life XI*, (ALife 2018)

AWARDS AND OTHER ACTIVITIES

2018 Participation in *Machine Learning Summer School*, UAM, Madrid

2017 Selected as one of the 100 Future Makers of 2017 by BCG company, Milan

2017 Participation in *Pre-Doc Summer School on Learning Systems*, ETH, Zurich

2016	Participation in *Spring School on Physics of Complex Systems*, ICTP, Trieste
2014	Participation in *Winter School on Quantitative Systems Biology*, ICTP, Trieste
2011	National Italian Prize "Scafati" for Poetry and Prose – *First Classificate* with the short story "I fiori di Mashid"
2010	Regional Qualification at Olympiads of Philosophy
2007, 2008	Double National Qualification at Olympiads of Astronomy
2006	Third Place at Regional Competition of Figure Skating, Calabria, Italy

COMPUTER SKILLS

PYTHON, PYTORCH, MATLAB, JAX, C, FORTRAN, NETLOGO, R, LaTeX, GnuPlot

LANGUAGES

Italian	Mothertongue
English	Proficient
German	Intermediate
French	Basic
Spanish	Basic