Nature-Inspired Inductive Biases in Learning Robots

DISSERTATION

der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Sebastian BLAES
aus Rüdesheim am Rhein

Tübingen
2022

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	24.11.2022
Dekan:	Prof. Dr. Thilo STEHLE
1. Berichterstatter/-in:	Dr. Georg MARTIUS
2. Berichterstatter/-in:	Prof. Dr. Martin V. BUTZ

Declaration of Authorship

I, Sebastian BLAES, declare that this thesis titled, "Nature-Inspired Inductive Biases in Learning Robots" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Abstract

Mathematisch-Naturwissenschaftliche Fakultät
Max Planck Institute for Intelligent Systems

Doktor der Naturwissenschaften

Nature-Inspired Inductive Biases in Learning Robots

by Sebastian BLAES

The work presented in this thesis studies various nature-inspired inductive biases in the domain of model-free and model-based reinforcement learning with the goal of designing AI agents that act more efficiently and autonomously in natural environments. The domain of robotic manipulation tasks is particularly interesting as it involves non-trivial system dynamics and requires abundant planning and reasoning. The inductive biases under investigation are primarily inspired by intelligent agents found in nature, such as humans and other animals. The primary sources of inspiration are as follows. (1) Hierarchically organized and specialized cortical structures facilitating efficient skills learning. (2) The self-organized playing of children to form intuitive theories and models about the world. (3) Structured exploration strategies based on various forms of intrinsic motivation and long-lasting temporal correlations in motor commands. (4) Imitation Learning. (5) Uncertainty-aware planning of motor commands in imagined models of a non-deterministic world.

Consequently, this work continues a long history of ideas and research efforts that take inspiration from nature to build more competent AI agents. These efforts culminated in research fields such as hierarchical reinforcement learning, developmental robotics, intrinsically motivated reinforcement learning, and representation learning. This work builds on the ideas that were advanced in these fields. It combines them with model-free and model-based reinforcement learning methods to solve challenging robotic manipulation tasks from scratch. Empirical studies are carried out to support the hypothesis that nature-inspired inductive biases might be an essential building block in designing more competent AI agents.

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Abstrakt

Mathematisch-Naturwissenschaftliche Fakultät
Max Planck Institute for Intelligent Systems

Doktor der Naturwissenschaften

Nature-Inspired Inductive Biases in Learning Robots

von Sebastian BLAES

Die in dieser Dissertation vorgestellten Arbeiten studieren verschiedene von der Natur inspirierte induktive Verzerrungen im Kontext von modellfreiem und modellbasiertem selbstverstärkenden Lernen, mit dem Ziel, KI Agenten zu entwerfen, die effizient und autonom in der realen Welt handeln. Dabei sind von Robotern zu bewältigende Objektmanipulationsaufgaben von besonderem Interesse, da die zeitliche Entwicklung dieser dynamischen Systeme nicht trivial ist und Manipulationsaufgaben schwierige Planungsprobleme darstellen. Die betrachteten induktiven Verzerrungen sind hauptsächlich von in der Natur zu findenden intelligenten Agenten, wie Tiere und Menschen, inspiriert. Die primären Inspirationsquellen sind wie folgt. (1) Hierarchisch organisierte und spezialisierte kortikale Strukturen, die die effektive Erlernung von Fähigkeiten unterstützen. (2) Das selbstorganisierte Spielen von Kindern zum Zwecke der Formung intuitiver Modelle und Theorien über die Welt. (3) Strukturierte Explorationsstrategien basierend auf unterschiedliche Formen von intrinsischer Motivation und lang anhaltender zeitlicher Korrelationen in motorischen Befehlen. (4) Imitationslernen. (5) Die Planung von Aktionssequenzen unter der Berücksichtigung von Unsicherheiten in mentalen Modellen der nicht-deterministischen Welt.

Diese Arbeit ist die Fortsetzung einer langen Historie von Ideen und Forschungsbemühungen, die Inspiration aus der Natur ziehen, um kompetentere KI Agenten zu entwickeln. Die Bemühungen in diesen Forschungsfeldern mündeten in der Ausbildung verschiedener Forschungsfelder wie hierarchisches selbstverstärkendes Lernen, Entwicklungsrobotik, intrinsisch motiviertes selbstverstärkendes Lernen und Repräsentationslernen. Diese Arbeit baut auf den in diesen Feldern entwickelten Ideen und Konzepten auf und kombiniert diese mit Methoden von modellfreiem und modellbasiertem selbstverstärkenden Lernen, um es Robotern zu ermöglichen, herausfordernde Objektmanipulationsaufgaben von Grund auf zu lösen. Die Hypothese, dass von der Natur inspirierte induktive Verzerrungen einen essenziellen Beitrag zur Erschaffung kompetenterer KI Agenten liefern könnten, wird dabei durch zahlreiche empirische Studien unterstützt.

Acknowledgements

I want to express my deepest gratitude to my supervisor, Dr. Georg Martius, for allowing me to pursue a Ph.D. in the Autonomous Learning group, for his invaluable feedback and guidance throughout my doctoral studies, and for the freedom he gave me to explore my ideas. I am also very thankful for the financial and academic support from the International Max Planck Research School (IMPRS) for Intelligent Systems (IS). Special thanks to Dr. Leila Masri and Sara Sorce for their personal support. I am also very grateful for the valuable feedback from the members of my thesis advisory committee, Prof. Dr. Martin V. Butz and Prof. Dr. Ludovic Righetti. I could not have undertaken this journey without the support of my parents, Johannes and Dagmar Bleas, who always encouraged me to follow my path. I also want to thank my sisters Franziska and Julia and my brother Maximilian.

I want to thank all my collaborators, which are acknowledged at the beginning of each section. Many thanks to Jakob Hollenstein, Dr. Pavel Kolev, Dr. Arash Tavakoli, and especially Cansu Sancaktar for proofreading my thesis and providing valuable feedback. Thanks to all the members of the Autonomous Learning group, everyone at the Max Planck Institute for Intelligent Systems, and all the people I had the pleasure to meet on this journey. They made it an unforgettable experience, and many of them influenced me in profound ways.

I want to thank my friends who endured this long process with me, always offering support and companionship.

Contents

List of Figures

List of Tables

Acronyms

A3C Asynchronous Advantage Actor Critic. 57

AI Artificial Intelligence. 4, 5

APEX Adaptive Policy EXtraction. xvi, xvii, 72, 73, 76, 77, 81–87, 89–91, 124, 125

BC Behavioral Cloning. xvi, 7, 71, 77, 78, 84, 86, 87, 91, 125

CEM Cross-Entropy Method. xiii, xvi, xvii, xix, 7, 30, 31, 72–76, 83–85, 88, 90, 123

CNN Convolutional Neural Network. 130

CWYC Control What You Can. xiii, xv, xvi, xviii, 6, 7, 39–42, 51, 52, 55–57, 59–62, 64–67, 71, 119–123

DAgger Dataset Aggregation. xvii, 77, 84, 86–88, 91, 125

DDPG Deep Deterministic Policy Gradient. 27, 51

DL Deep Learning. 26

DoF Degree of Freedom. xvi, 54, 71, 72, 81, 104–106

DP Dynamic Programming. xiii, 20–22

FRF Frequency Response Function. 74, 90, 123

GAN Generative Adversarial Network. 34

GNN Graph Neural Network. 42, 122–124

GP Gaussian Process. 32

GPI Generalized Policy Iteration. xiii, 22

GPS Guided Policy Search. xvi, xvii, 29, 77–80, 87–89, 91, 125

HER Hindsight Experience Replay. 51

HRL Hierarchical Reinforcement Learning. xiii, 6, 13, 32–34, 39, 57, 122, 130

iCEM improved Cross-Entropy Method. xvi, xvii, xix, 7, 72, 73, 75–77, 79–91, 102, 108, 123–125

IL Imitation Learning. 71, 73, 84, 86, 91, 124, 125

iLQR Iterative Linear Quadratic Regulator. 29

IM Intrinsic Motivation. xix, 5, 8, 13, 17, 34, 35, 55–57, 121, 122

IMRL Intrinsically Motivated Reinforcement Learning. 5, 6, 13, 34, 35

LQR Linear–Quadratic Regulator. 32

MBRL Model-Based Reinforcement Learning. 5–8, 30, 32, 71, 95, 103, 123

MC Monte-Carlo. xiii, 20–22, 24, 26, 95

MDP Markov Decision Process. xiii, 6, 13, 15–17, 19–21, 27–29, 31–33, 42, 52, 54, 71, 73, 91, 95, 97–99, 102, 125

ML Machine Learning. 4, 17, 27, 39, 126

MPC Model Predictive Control. xiii, xvi, 7, 30, 31, 71–73, 75, 77–79, 83, 84, 90, 91, 95, 102, 115, 123

NN Neural Network. xvi, xvii, 4, 5, 7, 8, 27, 32, 71–73, 76–80, 84, 86–91, 95, 97, 100, 101, 122, 124, 125, 130

PETS Probabilistic Ensembles with Trajectory Sampling. xvii, xviii, 108–115

PETSUS Probabilitic Ensembles with Trajectory Sampling and Uncertainty Separation. 100, 115, 125

PILCO Probabilistic Inference for Learning Control. 29

POMDP Partially Observable Markov Decision Process. 16

PPO Proximal Policy Optimization. 27

PSD Power Spectral Density. 73–75, 90, 123, 124

RAZER Risk-Averse Zero-Order Trajectory Optimization Method. xvii, xviii, 95, 97, 99–103, 108–115, 124–127

RL Reinforcement Learning. xiii, xix, 4, 6, 7, 13, 16, 17, 20–22, 26–30, 33, 34, 39, 41, 54, 55, 64, 67, 71–73, 77, 87, 91, 95, 97, 115, 119, 121, 122, 125

RNN Recurrent Neural Network. 123

RS Random Shooting. 30

SAC Soft Actor Critic. xvi, 27, 51, 55–57, 84, 87, 89, 121

SMDP Semi-Markov Decision Process. 33

TD Temporal Difference. xiii, 21, 22, 24

TD3 Twin Delayed DDPG. 57

TRPO Trust Region Policy Optimizatio. 27

UVFA Universal Value Function Approximator. 32

1

INTRODUCTION

1.1 Natural Intelligence: Autonomous Biological Agents

The autonomy and ability to quickly adapt to a dynamic environment are critical for the existence and survival of any intelligent creature. Throughout evolution, nature provided humans and other animals with various innate behaviors (Versace et al., 2015), specialized cortical structures (Kanwisher, 2010), the ability to adapt such skills and structures to a changing environment (Kuhl et al., 1992; Saffran et al., 1996), and learning mechanisms to acquire entirely new competencies.

In precocial species, newborns are relatively mature shortly after birth and show a range of complex behaviors that do not require any or large amounts of learning (Versace et al., 2015). For example, to survive, prey animals like horses can stand and run minutes after they are born (Miller et al., 2005). Altricial species like humans, on the other hand, go through a prolonged maturation period in which most of their behavior has to be learned through social interactions, imitation learning, and exploratory self-play (Nehaniv et al., 2007). Nonetheless, the human brain has a variety of hard-coded components that ensure the survival of the individual and the species as a whole. For instance, reflexes bypass the conscious part of the brain entirely (Johns, 2014) and are direct and hard-coded motor responses to specific stimuli. An example of such a reflex is the sucking reflex in newborns that ensures the baby's access to food while it is still not in complete control of its own body. Moreover, the sucking reflex is a stepping stone for learning voluntary breastfeeding behavior (Sherman Ross et al., 1957) and therefore guides learning in a world full of countless possibilities.

Imitating the behavior of others plays a vital role in the acquisition of new skills (Aitken, 2018) in humans and many other animals. For example, human newborns can already match facial expressions of other humans (Andrew N. Meltzoff et al., 1997) and infants imitate facial and manual gestures from observations (Andrew N Meltzoff et al., 1977). This mirroring behavior occurs even before the emergence of a reflective self-awareness that typically develops between 13 and 26 months of age (Lewis, 2012) together with the formation of self-reference and self-conscious emotions. These observations argue for the existence of innate cortical structures that support learning through imitation. One of these systems might be the mirror system (Rizzolatti et al., 2001) with mirror neurons (Acharya et al., 2012) showing a similar activation pattern independent of whether a person performs an action directly or observes someone else performing the same activity. Not only do children use imitation learning to acquire new skills, but humans throughout their entire life imitate the behavior of other humans to modulate already existing skills (Möttönen et al., 2005) or learn entirely new behavior.

An equally important role plays self- or intrinsically motivated learning in acquiring new skills and adapting preexisting structures. It is believed that children use self-play to develop and practice subroutines (Bruner, 1973) that later benefit more goal-directed behavior (Weisler et al., 1976) and serve children as a playground to conduct experiments and analyze the statistics in their observations to form intuitive theories and models about the world (Gopnik et al., 2004). There is growing evidence that during self-play, children explore surprising events (Stahl et al., 2015), observations that violate their prior beliefs (Legare et al., 2010) or situations that maximize information gain (Ruggeri et al., 2019). In other animals, more rudimentary but not utterly random exploration strategies are observed too. For example, if food is scarce, sharks show an exploratory behavior described by Lévy flights in

that it exhibits long-lasting temporal correlations compared to a purely random exploration strategy (Sims et al., 2008; Humphries et al., 2010).

Humans and other animals form intuitive theories and models about the world (Gopnik et al., 2004). However, they are constantly confronted with and have to act within an inherently uncertain environment. Uncertainties can originate either from a lack of information or inherently stochastic system dynamics. Therefore, it is critical for acting beings to constantly reevaluate and update their inner beliefs about the world by incorporating new or confounding information from the sensorimotor stream. In human cognition, this is known as metacognition (Metcalfe et al., 1994). In humans and other animals, it is observed that uncertainties about the world are taken into account during the planning of actions, either by taking actions that reduce the uncertainty about the environment (Belger et al., 2018) or by acting in a risk-averse fashion if the outcomes of certain actions are hard to predict or failure comes with a high cost. Research suggests that humans optimize motor plans for task performance while explicitly accounting for the known uncertainty in the environment (Alhussein et al., 2021).

All this research implies that the behavior observed in biological agents is not purely learned from the ground up. Instead, evolution equipped biological agents with innate skills and shaped the brain to facilitate the adaptation of these skills and the learning of new skills. Different species occupying different ecological niches developed similar structures to different extents and even developed entirely different systems depending on their needs. For instance, humans have developed very distinct brain areas for language processing (Friederici, 2011) because of the importance of social interactions in human lives.

1.2 From Natural to Artificial Intelligence: Autonomous Artificial Agents

The desire to build intelligent machines dates back to antiquity. Already one of the first known automata from Greek mythology, Talos (McCorduck et al., 2004), has been made by Hephaestus in the image of humans. It is not surprising that Artificial Intelligence (AI) systems are often inspired by the human brain, as it is the most sophisticated learning system known to humankind. The actual advent of AI systems began with the development of programmable computers. Alan Turing, one of the pioneers of theoretical computer science and artificial intelligence, stated that to build a general AI, one must create a machine that can learn like a child (Turing, 1990). In "The Need for Biases in Learning Generalizations" (Mitchell, 1980) it is argued that a program can never make the leap necessary to classify instances beyond those it has observed. Therefore, other sources of information in the form of biases have to be introduced to choose one generalization over the other. Jonschkowski et al. (2015) argues for numerous robotic priors for learning state representations, namely a temporal coherence prior, a proportionality prior, a causality prior, and repeatability prior. Neural Networks (NNs), one of the main drivers of modern Machine Learning (ML), are inspired by how the brain processes information with artificial neurons mimicking the behavior of individual cells in the brain (Rosenblatt, 1957). Reinforcement Learning (RL), a field of ML concerned with learning agents that act in an environment to maximize a performance metric, is primarily inspired by the trial-and-error-based learning and "reinforcement" theories in animal and human learning (Farley et al., 1954; M. L. Minsky, 1954). In recent years, RL combined

with powerful function approximators such as deep NN showed remarkable successes in solving Atari games on a super-human level (Mnih, Kavukcuoglu, Silver, Graves, et al., 2013), beating humans in two-player games like Chess and Go (Silver, Hubert, et al., 2017), and solving challenging robotic manipulation tasks (H. Kim et al., 2003; OpenAI et al., 2019) with high-dimensional continuous state spaces and action spaces.

Most of these works learn a monolithic policy from raw experience in an end-to-end fashion. However, this requires an immense amount of data that needs to be collected through the agent's interactions with the environment. For instance, learning the control policy that solves a Rubik's cube with a human-like Shadow Dexterous Hand requires a cumulative experience equivalent to roughly 13 thousand years of real-time experience (OpenAI et al., 2019). On the other hand, humans can often learn from just a few examples (Lake et al., 2019). One potential reason for the efficiency of humans in learning new skills is the extensive repertoire of prior knowledge they can make use of. Another reason, as argued in the previous section, is that evolution shaped cortical structures to facilitate learning relevant skills for survival. Typically, AI agents are trained and evaluated in the same highly confined environments with clearly defined and externally provided goals. In contrast, humans live in an open and dynamic environment that requires constant adaptation and quick reactions to unforeseeable events. It is very hard to make accurate predictions far into the future in the real world because of the many sources of uncertainty that exist in the natural world.

To make learning in artificial agents more efficient and self-organized, some researchers aim to simulate more aspects of human cognition with AI systems, for instance, by incorporating inductive biases into the learning process that are often inspired by how evolution shaped the brain to become the impressive learning machine that is known to us today or by how children explore their environment without any external guidance. The research field of embodied robotics views the agent's embodiment and the learning algorithm in unity. In embodied robotics, it is studied how the constraints emerging from this unity shape the learning of skills and influence the behavior of artificial agents in open-ended environments. See Lungarella et al. (2003), Asada et al. (2009), and Oudeyer (2010) for surveys of the field.

The goal of Intrinsically Motivated Reinforcement Learning (IMRL) is to get rid of the idea of specifying particular external goals altogether and instead define various intrinsic reward signals that encourage the agent to learn new skills and explore the environment in an autonomous and open-ended fashion. Numerous definitions and implementations of Intrinsic Motivation (IM) were developed in the past, including IM based on the disagreement between a mental model and real observations (Schmidhuber, 1991; Pathak et al., 2017), learning progress (Schmidhuber, 1991; Lopes et al., 2012; K. Kim et al., 2020) (used in Ch. 3), or information gain (Pfaffelhuber, 1972; Storck et al., 1995; Cover, 1999; Houthooft et al., 2016). Section 2.2.5 of Ch. 2 discusses other forms of IM in more detail.

In representation learning, priors in the form of information bottlenecks (Yingjun et al., 2019) or the composability of state abstractions (Burgess et al., 2019) are used to learn state abstractions useful for downstream control tasks.

Another field of research that gained traction recently is Model-Based Reinforcement Learning (MBRL). In MBRL, a mental model of the environment is learned to make predictions about the future. "Mental simulations" are used to learn control policies from "imagined data" (Richard S Sutton, 1990; Richard S Sutton, 1991a; Richard S Sutton, 1991b) or for planning (Richards, 2005; Chua et al., 2018). Section 2.2.3 of Ch. 2 discusses the different research directions within MBRL in more

detail. Chapter 4 and Ch. 5 use planning to solve challenging robotic manipulation tasks. One promise of MBRL is that models are more general than specialized control policies and can be transferred between tasks. An advantage of models is that uncertainties can be modeled explicitly and considered during planning. In optimal control, this is studied under robust optimization (Arruda et al., 2017; Abraham et al., 2020). However, in MBRL, uncertainty-aware planning is less well explored. Chapter 5 presents a method for uncertainty estimation and planning with learned dynamics models.

1.3 Scope of this Work

The works presented in this thesis are placed between the research fields discussed above by posing the following questions:

- How can specialized and hierarchically organized structured models aid the training of control policies?
- How do compartmentalized policies compare to monolithic policies in the open-ended learning setting?
- How can intrinsic motivation help structure and guide the developmental self-play of artificial agents?
- How can other forms of exploration like the temporally structured exploration described by Lévy flights help build more efficient agents?
- Which inductive biases are necessary to effectively learn neural network policies via imitation learning from a non-deterministic teacher?
- How can uncertainties be faithfully estimated and explicitly utilized in model-based reinforcement learning to maximize information gain?

Chapter 2 reviews the mathematical formalisms and methods that are used throughout this work. Markov Decision Processes (MDPs) are introduced as a popular and widely adopted formalism to describe problems of sequential decision-making. The remaining sections of this chapter discuss several learning methods for solving MDPs via RL. RL as a general framework for solving MDPs is introduced in Sec. 2.2. RL is used to solve MDPs from experience efficiently, that is, to learn control policies that produce action sequences that maximize a given performance metric. Model-free RL methods (Sec. 2.2.1) learn a policy directly from the raw experience of the agent in the environment. They do not have direct access to nor make any assumptions about the underlying model of the MDP. Model-based RL methods (Sec. 2.2.3), on the other hand, aim to learn a model of the MDP from data. Section 2.2.3 discusses various types of models that can be learned from data and different ways of utilizing models to solve MDPs. Section 2.2.4 introduces Hierarchical Reinforcement Learning (HRL) as a framework for solving complex problems through hierarchical abstraction. Section 2.2.5 surveys the research field of Intrinsically Motivated Reinforcement Learning (IMRL), an area that is largely inspired by behavioral psychology and developmental robotics.

Chapter 3 presents *Control What You Can (CWYC)*, a HRL agent that aims to maximize its controllability over environments through self-play. *CWYC* combines model-based planning in hierarchical abstracted spaces with model-free RL to learn low-level control policies. The design of the *CWYC* agent is primarily motivated

by how self-play in infants and small children is organized and takes place. In a developmental phase, the agent is given time to explore its environment in a completely self-organized fashion to learn useful subroutines. Later, the agent can compose these subroutines for goal-directed behavior. Learning low-level subroutines in the developmental phase is supported by several hierarchically organized structured models and driven by the agent's intrinsic motivation to maximize learning progress. The structured models support the agent's learning in two ways: (1) They significantly reduce the low-level controllers' computational complexity and planning burden when solving compositional long-horizon manipulation tasks. (2) They allow for structured exploration on the level of objects and goals instead of unstructured exploration on the low-level actions typically used in RL. The *CWYC* agent manages a self-guided learning curriculum distributing learning efforts between self-imposed tasks to maximize learning progress. An object- or task-level planning module keeps track of task dependencies in compositional object manipulation tasks, e.g., tasks requiring to go through one or multiple funnel states like picking up a particular tool. The planning module ensures that suitable subroutines are executed in the correct order. An object-centric relational learning module learns and provides meaningful sub-goals that allow the agent to "glue together" the individual subroutines in a way that leads to success in downstream tasks. Although the planning modules have a predefined internal structure that captures certain aspects of the planning problem, they are still learned from data and flexible enough to adapt to the peculiarities of the different environments.

CWYC uses specialized structured models to guide the exploration and to support a low-level NN control policy in challenging compositional tasks. Chapter 4 addresses two major challenges that arise if models are used explicitly for decision making down to the low-level control: (1) How can the space of possible future outcomes be explored efficiently. Especially if no prior knowledge or gradient information is available, one must resort to gradient-free or zero-order methods. (2) How can planning methods be made fast enough to be executed in real-time on physical robots. Specifically, zero-order planning methods combined with learned dynamics models are considered to optimize action sequences in mental simulations or an open-loop fashion without taking feedback from the real system into account. If the mental simulation is executed in an Model Predictive Control (MPC) fashion, the planning method can be cast as a closed-loop control policy. Section 2.2.3 of Ch. 2 introduces model learning in the context of MBRL, the Cross-Entropy Method (CEM) and MPC. Inspired by the efficient exploration of various animals like insects and predatory species, the *improved Cross-Entropy Method (iCEM)* replaces the temporally unstructured action sampling of CEM with a temporally structured action sampling that is akin to the so-called Lévy flights. In addition, memory inside the *iCEM* allows the reuse of information from the previous mental simulations to increase the sample efficiency of the planner even further. Empirical studies demonstrate a significant improvement of iCEM over CEM in sample complexity and performance, making it more apt for real-time robotic applications. To push the planning method even further toward real-time operation, it is combined with imitation learning techniques to extract an NN policy from expert trajectories generated by *iCEM*. Empirically it is shown that a simple Behavioral Cloning (BC) approach with a one-sided influence from teacher to student can result in sub-optimal NN policies because of the stochastic nature of the CEM optimizer. Implementing an adaptive mutual influence between teacher and student based on the student's performance shows that strong NN control policies can be learned from stochastic teachers even in challenging and high-dimensional robotic manipulation tasks. Moreover, the quality of the

solutions produced by the planning method even improves alongside the performance of the NN policy.

When dealing with physical robots in the real world, the system's failure can lead to the robot's destruction or dangerous situations if humans and robots interact with each other. As discussed earlier, animals naturally incorporate any uncertainties they might have about the world into their planning for future actions. This results in more cautious or reckless behavior depending on the situation and the animal's intentions. Moreover, animals are intrinsically motivated to maximize their information gain during exploration by explicitly seeking situations with reducible uncertainties. To build robots that act naturally in the real world, they have to have a sense of how much they can trust their internal beliefs about the world. To this end, Ch. 5 studies uncertainty estimation and uncertainty-aware planning in the context of MBRL. If models are learned from data, two types of uncertainty may occur. Aleatoric uncertainty reflects the uncertainty originating from noise intrinsic to the system. The aleatoric uncertainty is irreducible, and any uncertainty-aware planner needs to take this uncertainty into account to minimize the risk of failure. Epistemic uncertainty originates from insufficient training data and can be reduced by collecting more and better data. The epistemic uncertainty is directly related to the quality of the predictions produced by the model. To decrease the epistemic uncertainty and thereby maximize information gain as well as increase the quality of the projections, more data needs to be collected for which the model makes wrong predictions, i.e., for which the epistemic uncertainty is high. Consequently, the epistemic uncertainty can be phrased as a form of IM that should drive the planner towards situations that are difficult to predict. This work combines the estimation and disentanglement of aleatoric and epistemic uncertainties in learned models with model-based planning that makes explicit use of the two types of uncertainties in active learning, risk-averse planning, and planning under external safety constraints.

Chapter 6 provides a general discussion of the presented works in the context of biological agents and the closely related research fields discussed before. Finally, the thesis closes with an outlook and final remarks in Ch. 7.

2

Theoretical Background

Contents

Introduction

This chapter reviews the mathematical formalisms and concepts used throughout this work. Section 2.1 introduces Markov Decision Processes (MDPs) as a widely adopted mathematical formalism for modeling processes involving sequential decision making. In a nutshell, in an MDP a system is fully described in terms of states, actions, and the action conditioned transition dynamics between states. Each project discussed in this work is about solving an MDP by using methods from Reinforcement Learning (RL), planning, Hierarchical Reinforcement Learning (HRL), Intrinsically Motivated Reinforcement Learning (IMRL), and (spatial) reasoning.

Section 2.2 introduces RL as a collection of practical methods to efficiently solve MDPs. The goal of RL is to learn a policy, or behavioral recipe, that acts optimal w.r.t. the underlying MDP. In model-free RL (Sec. 2.2.1), the transition dynamics of the MDP is assumed to be unknown, and a policy is learned directly from data collected through interactions of the policy with the MDP. Model-based RL (Sec. 2.2.3) methods aim to learn an explicit model of the MDP transition dynamics. Once such a model is learned, it can be used to generate imagined data or for planning. Section 2.2.4 introduces HRL as a way of solving complex, long-horizon tasks through decomposition and hierarchical abstraction. This chapter closes by reviewing the research field of HRL (Sec. 2.2.5). In HRL, numerous formulations of Intrinsic Motivation (IM) are studied with the goal of building genuinely autonomous agents that act in open-ended environments without any predefined purposes.

Even though this chapter captures many of the key results of the topics mentioned above, a substantial amount of research has been done in these fields. The reader is encouraged to consult the seminal textbooks in these fields (Richard S. Sutton and Andrew G. Barto, 1998; D. P. Bertsekas, 2011; D. Bertsekas, 2012) or the survey articles referenced in the individual sections for further references.

Notation

The following table serves as a reference for the notation used throughout this work.

$(x_i)_{i=0}^{N-1}$	A sequence of length N
x_i	The i-th element of the sequence $(x_i)_{i=0}^{N-1}$
X	Capital letters denote random variables
$\mathcal{X}$	Calligraphic letters denote vector spaces
$x \in \mathcal{X}$	Small letters denote elements of a vector space
$x(t)$, x_t	Time related indices are denoted by the letter t and appear in brackets or as subscripts (for a compact notation)

TABLE 2.1: Table of notation used throughout the work.

2.1 Markov Decision Processes

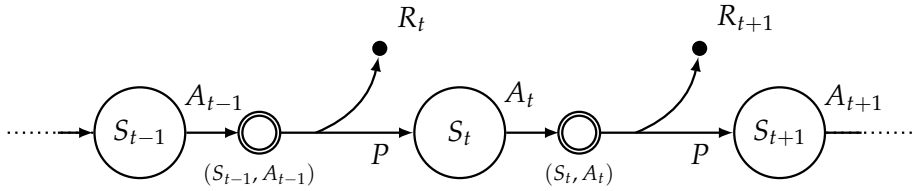


FIGURE 2.1: Graphical depiction of a discrete-time MDP. At time step t, the system is in state S_t. By taking an action A_t, the system transitions into state S_{t+1} with probability $P(S_{t+1} \mid S_t,\ A_t)$. While transitioning between states, the system provides feedback in the form of a reward signal R_{t+1}.

Formally, a Markov Decision Process (MDP) (Bellman, 1957; Howard, 1960) is defined as the 4-tuple $\mathcal{M} = \left(\mathcal{S},\ \mathcal{A},\ P^a_{ss'},\ R^a_{ss'}\right)$. In an MDP, the system is entirely defined by its current state $s \in \mathcal{S}$, with $\mathcal{S}$ being the space of all possible states the system can be in. In the context of MDPs, the intuitive meaning is that all the historical information about past states that might be needed for subsequent decision making is already contained in the current state. Such a memory-less system, or state, is said to have the Markov Property (see definition 2.1.1). In the following, it is assumed that all MDPs are discrete in time, i.e., every two consecutive states s_t and s_{t+1} have the same finite distance in time between each other. In an MDP, decisions are made in the form of actions $a \in \mathcal{A}$, with $\mathcal{A}$ being the space of all possible actions. Contrary to the temporal locality of states, actions can have long-lasting effects on future states or outcomes. For instance, the MDP might have two distinct, completely disconnected branches that are reached by taking different actions from the same root state.

Given a certain state s and action a, the transition kernel $P^a_{ss'}\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,\ 1]$ defines the probability of transitioning into a new state $s' \in \mathcal{S}$:

$$P^a_{ss'} = p(s' \mid s,\ a) = \mathrm{P}(S_{t+1} = s' \mid S_t = s,\ A_t = a), \tag{2.1}$$

with

$$\int_{s' \in \mathcal{S}} p(s' \mid s,\ a)\ \mathrm{d}s' = 1, \quad \forall s \in \mathcal{S},\ a \in \mathcal{A}.$$

The transition kernel fully describes the dynamics of an MDP.

The reward function $R^a_{ss'}\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ maps every state-action-state triplet to a real-valued number. In MDPs, the reward function is used to define goals or tasks by providing a transition-dependent feedback signal. The general convention is that transitions with higher rewards are higher valued w.r.t. the current task or lead to states with a smaller distance to a goal state.

Viewed over multiple time steps, states and actions give rise to a sequence called a trajectory:

$$(S_t,\ A_t)_{t=0}^{t=H-1} = (S_0,\ A_0,\ S_1,\ A_1,\ \ldots), \tag{2.2}$$

with H being the horizon or length of the trajectory. The state S_{t+1} is only conditioned on the previous state S_t and action A_t and is distributed according to Eq. 2.1 due to the Markov Property:

Definition 2.1.1 (Markov Property). *A system or state has the Markov Property if and only if the probability of transitioning into state S_{t+1} depends only on the previous state S_t and action A_t and not on the history of previous states and actions:*

$$\mathrm{P}(S_{t+1} \mid S_t,\ A_t) = \mathrm{P}(S_{t+1} \mid S_0,\ A_1,\ S_1,\ \ldots,\ S_t,\ A_t).$$

Every MDP defines an optimization problem of finding an action sequence that maximizes the cumulative rewards induced by the trajectory $(S_t,\ A_t)_{t=0}^{H-1}$. With the knowledge of the transition kernel, the optimization problem can be solved to optimality. Yet, for most interesting problems, the transition kernel is unknown apriori, and approximate solutions must be found. Figure 2.1 shows a graphical depiction of an MDP. Nodes with a normal border depict states. Nodes with double borders represent state-action pairs. Filled nodes are used for rewards. Transitions between states or state-action pairs are depicted as directed arrows. This nomenclature is used throughout this work.

What kind of problems can be formalized as MDPs? It turns out that many problems involving sequential decision-making can be cast as MDPs: From discrete state-space and action-space problems like two-player games such as Chess and the game of Go to continuous state-space and action-space problems like robotic control problems. Nonetheless, the real world is usually more complex than board games or controlled robotic experiments. Partially Observable Markov Decision Processes (POMDPs) extend MDPs to deal with problems in which only partial information about the system is available.

2.2 Reinforcement Learning

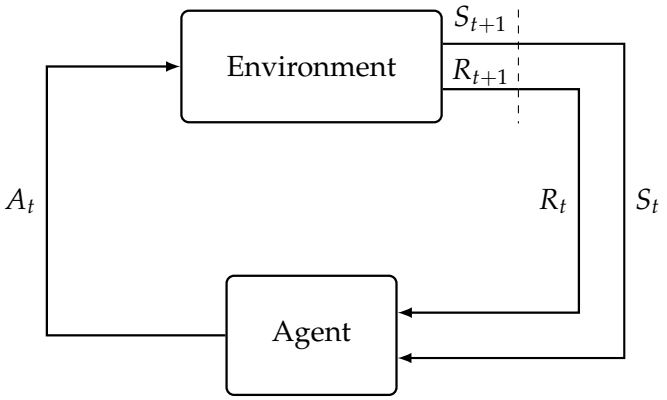


FIGURE 2.2: Graphical depiction of the RL problem. An agent interacts with an environment by receiving its current state S_t and the reward signal R_t. By sending actions A_t, the agent causes the environment to transition into a new state S_{t+1} according to the transition probability $\mathrm{P}(S_{t+1} \mid S_t,\ A_t)$.

The previous section introduced MDPs as a formal framework to model processes of sequential decision making. This section discusses the Reinforcement Learning (RL) problem. RL translates the abstract mathematical concepts introduced in the previous section into physical concepts and provides practical solutions, i.e., algorithms, to solve MDPs. In RL, an agent interacts with an environment by receiving the environment's current state and reward and taking consecutive actions.

Example: *In a robotic control setting, actions can be low-level commands like joint torques or end-effector positions but also high-level goals like washing the dishes or following a predefined path.*

By sending actions to the environment, the agent changes the state of the environment according to its internal dynamics, which is entirely defined by the MDP transition kernel $P^a_{ss'}$.

Example: *In a robotic control setting, the state typically includes information about the different objects in the environment or information about the robot's surroundings. In addition, it often contains sensor readings and joint information of the robot itself. This is in contrast to a human-centered view of what constitutes the "self". As humans, we associate such information with our perception and proprioception and not as part of the external world.*

After executing an action, the agent receives the new state and a feedback signal in the form of a reward from the environment. The reward encodes a specific task the agent is supposed to solve, e.g., picking up a cup from a table. By common convention, the reward is part of the environment and therefore external to the agent. There are instances in which the reward or parts of the reward is internal to the agent, e.g., as some form of Intrinsic Motivation (IM). Different types of IM are reviewed in Sec. 2.2.5 and are used in the works discussed in Ch. 3 and Ch. 5. Generally, the reward signal allows the agent to assess the quality of its action w.r.t. its current task. Throughout this work, the reward signal is assumed to be a scalar value, potentially in the form of a summation of multiple extrinsic and intrinsic reward components. It is important to note that ideally the reward does not tell the agent *how* to solve a problem but rather *what* kind of outcome is to be expected.

Example: *For instance, the reward should not tell a robot exactly how to pick up a cup: First, open the gripper. Then move the gripper to the cup. Pick up the cup. Finally, lift the cup from the table. Instead, the reward should tell the robot that eventually, the cup should not be on the table and in full control of the robot.*

Nevertheless, in practice this assumption is often violated by providing shaped and highly engineered rewards.

The existence of a reward signal and the fact that the agent can choose actions to change the distribution of future rewards and states is one of the most notable differences between RL and other Machine Learning (ML) fields such as (un-)supervised ML and renders the RL problem much harder than other problems in ML. Another related issue is the so-called credit assignment problem (M. Minsky, 1961). The credit assignment problem describes the challenge of distributing credit among the countless past decisions that lead to a particular reward at some later point in time. The credit assignment problem is closely intertwined with the sequential nature of RL problems and the fact that past actions can influence future outcomes.

Figure 2.2 depicts the interaction between the different components in the RL setting. Formally, the objective of RL is to learn a policy that maximizes the discounted return or discounted accumulated future rewards:

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{2.3}$$

In Eq. 2.3, $0 < \gamma \leq 1$ refers to the so called discount factor. The discount factor ensures that the infinite sum is bounded:

$$G_t \leq \sum_{k=0}^{\infty} \gamma^k r_{\max} = \frac{r_{\max}}{1-\gamma}, \quad |\gamma| < 1, \tag{2.4}$$

with $r_{\max}$ being the maximum achievable reward. Another commonly used interpretation of the discount factor is that it trades off recent against more distant rewards. In other words, for $\gamma \to 0$, the agent is more myopic, while for $\gamma \to 1$, the agent is more concerned about future outcomes. Immediate and future rewards in Eq. 2.3 are related by a recursive relationship:

$$G_t = R_{t+1} + \gamma G_{t+1}. \tag{2.5}$$

A stationary policy $\pi \in \Pi$, with $\pi \colon \mathcal{S} \times \mathcal{A} \to [0,\ 1]$ and Π being the space of all possible policies, is a function that maps states to a distribution over actions:

$$A \sim \pi(\cdot \mid S) = p(A \mid S). \tag{2.6}$$

To find the next optimal action A_t at time step t, an agent has to be able to predict the long-term outcome, i.e., the future return G_t, of following the policy starting in a state S_t. To this end, a state-value function $v_\pi \colon \mathcal{S} \to \mathbb{R}$ for policy π can be defined as:

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{r+k+1} \,\middle|\, S_t = s \right], \quad \forall s \in \mathcal{S}, \tag{2.7}$$

where the expectation $\mathbb{E}_\pi [\cdot]$ in Eq. 2.7 captures the entire agent-environment dynamics:

$$\mathbb{E}_\pi \left[f(s) \right] = \int_{\mathcal{A}} \pi(a \mid s) \int_{\mathcal{S}} \int_{\mathbb{R}} p(s' \mid s,\ a) p(r \mid s,\ a) f(s) \, \mathrm{d}a \, \mathrm{d}s' \, \mathrm{d}r. \tag{2.8}$$

To access the long-term outcome of a particular action, an action-value function $q_\pi \colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ for policy π can be defined as:

$$\begin{aligned} q_\pi(s,\ a) &= \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] \\ &= \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a \right], \quad \forall s \in \mathcal{S},\ a \in \mathcal{A}. \end{aligned} \tag{2.9}$$

Similarly to the recursive relationship between immediate and future rewards in Eq. 2.5, a recursive relationship between the value of the current state and the value of the next states can be derived:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi \left[G_t \mid S_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} \mid S_t = s \right] \\ &= \int_{\mathcal{A}} \pi(a \mid s) \int_{\mathcal{S}} \int_{\mathbb{R}} p(s',\ r \mid s,\ a) \left[r + \gamma \, \mathbb{E}_\pi \left[G_{t+1} \mid S_{t+1} = s' \right] \right] \mathrm{d}a \, \mathrm{d}s' \, \mathrm{d}r \\ &= \int_{\mathcal{A}} \pi(a \mid s) \int_{\mathcal{S}} \int_{\mathbb{R}} p(s',\ r \mid s,\ a) \left[r + \gamma v_\pi(s') \right] \mathrm{d}a \, \mathrm{d}s' \, \mathrm{d}r, \quad \forall s \in \mathcal{S}, \end{aligned} \tag{2.10}$$

with $p(s',\ r \mid s,\ a) = p(s' \mid s,\ a) p(r \mid s,\ a)$. Equation 2.10 reveals an important property of the value function: To determine the value of a state s, only a one-step look-ahead to the value of the next state s' is necessary. This allows to compute the value function very efficiently because all future states after the next state can be ignored in the computation.

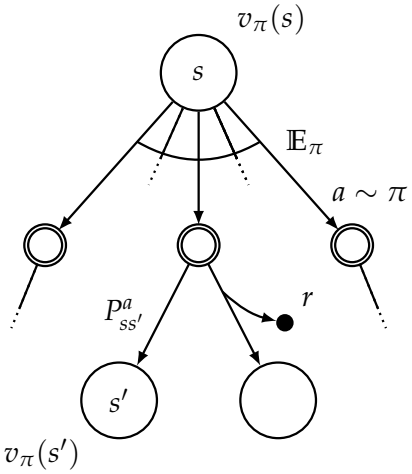


FIGURE 2.3: Backup diagram for the value function. To update the value of state S, a one-step look-ahead to the values of states S' is sufficient. The next state values are aggregated by taking the expectation over the action distribution.

Equation 2.10 is known as the Bellman equation (Richard E. Bellman, 1957) for v_π. The Bellman equation can be rewritten in the form of the Bellman operator $\mathcal{T}^\pi \colon \mathcal{S} \to \mathcal{S}$:

Definition 2.2.1. *Let π, v be an arbitrary policy and a value function, respectively. Then, the Bellman operator is defined by:*

$$\left[\mathcal{T}^\pi(v)\right](s) = \int_{\mathcal{A}} \pi(a|s) \int_{\mathcal{S}} \int_{\mathbb{R}} p(s', r|s, a) \left[r + \gamma v(s')\right] da\, ds'\, dr \qquad \forall s \in \mathcal{S}. \quad (2.11)$$

Given a policy π, the Bellman operator has a unique fixed point:

$$\mathcal{T}^\pi v_\pi = v_\pi, \quad (2.12)$$

which is a consequence of the contraction map theorem for the Bellman operator:

Theorem 2.2.1. *Given an MDP with $\gamma \in [0, 1)$. For any two value functions $v_1(s)$ and $v_2(s)$ the Bellman operator $\mathcal{T}^\pi$ is a γ-contraction mapping under the supremum norm $\|f(x)\|_\infty = \sup_{x \in \mathcal{X}} |f(x)|$:*

$$\|\mathcal{T}^\pi v_1 - \mathcal{T}^\pi v_2\|_\infty \leq \gamma \|v_1 - v_2\|_\infty. \quad (2.13)$$

For the proof see Appendix A.1. From the Banach fixed-point theorem (Banach, 1922)

Theorem 2.2.2 (Banach fixed-point Theorem)**.** *Given a complete metric space $(\mathcal{S}, d)$, with d being a metric on $\mathcal{S}$. The contraction mapping $\mathcal{T} \colon \mathcal{S} \to \mathcal{S}$ has a unique fixed-point $v^\star$ with:*

$$\mathcal{T} v^\star = v^\star. \quad (2.14)$$

it follows that $v_\pi(s)$ is a fixed-point of the Bellman operator $\mathcal{T}^\pi$ for the policy π. Intuitively, Eq. 2.14 states that if one starts from an arbitrary value function $v(s)$ and repeatedly applies the Bellman operator $\mathcal{T}^\pi$ for the policy π, v will converge to the value function $v_\pi(s)$ for the policy π:

$$\lim_{k \to \infty} (\mathcal{T}^\pi)^k v = v_\pi. \quad (2.15)$$

The Bellman equation and the iterative scheme in Eq. 2.15 are the basis for many popular RL algorithms. Equation 2.15 is also called policy evaluation because the quality of the current policy is evaluated in the MDP under consideration. The so-called backup diagram for the value function is depicted in Fig. 2.3. For the action-value function, an equivalent Bellman equation and Bellman operator exist.

Similar to the value function for an arbitrary policy π, we can define an optimal value function $v^\star$ for an optimal policy $\pi^\star$. This is because value functions define a partial ordering over policies with $\pi \geq \pi'$ if and only if $v_\pi(s) \geq v_{\pi'}(s),\ \forall s \in \mathcal{S}$. Hence, at least one policy $\pi^\star \geq \pi'$ exists, with $v_{\pi^\star} \geq v_{\pi'},\ \forall \pi' \in \Pi$, that is better or equal to any other policy. In fact, there might exist multiple optimal policies, all sharing the same optimal value function. The optimal value function is defined as:

$$v^\star(s) = \max_{\pi \in \Pi} v_\pi(s), \quad \forall s \in \mathcal{S}. \tag{2.16}$$

The optimal Bellman operator for the optimal value function is defined as:

$$\mathcal{T}^\star(v) = \max_\pi \mathcal{T}^\pi(v),$$

or equivalently (for any reward, there is an optimal deterministic policy (Puterman, 1994)):

$$\left[\mathcal{T}^\star(v)\right](s) = \max_{a \in \mathcal{A}} \int_{\mathcal{S}} \int_{\mathbb{R}} p(s', r|s, a) \left[r + \gamma v(s')\right] ds'\, dr \qquad \forall s \in \mathcal{S},$$

The optimal Bellman operator has the unique solution:

$$\mathcal{T}^\star v^\star = v^\star. \tag{2.17}$$

Repeatedly applying the optimal Bellman operator to any value function v yields the optimal value function:

$$\lim_{k \to \infty} (\mathcal{T}^\star)^k v = v^\star. \tag{2.18}$$

A similar optimal Bellman operator can be defined for the optimal action-value function $q^\star$.

Once the optimal (action-)value function is computed via the iterative scheme defined in Eq. 2.18, the optimal policy $\pi^\star$ can be retrieved by following the greedy policy, that is, by following the policy that maximizes the optimal (action-)value function. One advantage of the (action-)value is that for computing the optimal policy, the agent only has to act locally optimal or greedy w.r.t. the (action-)value function in the current state. The reason is that the (action-)value function for a given state already contains all the necessary information about all possible future outcomes. Therefore, a one-step look-ahead search is sufficient to get long-term optimal actions.

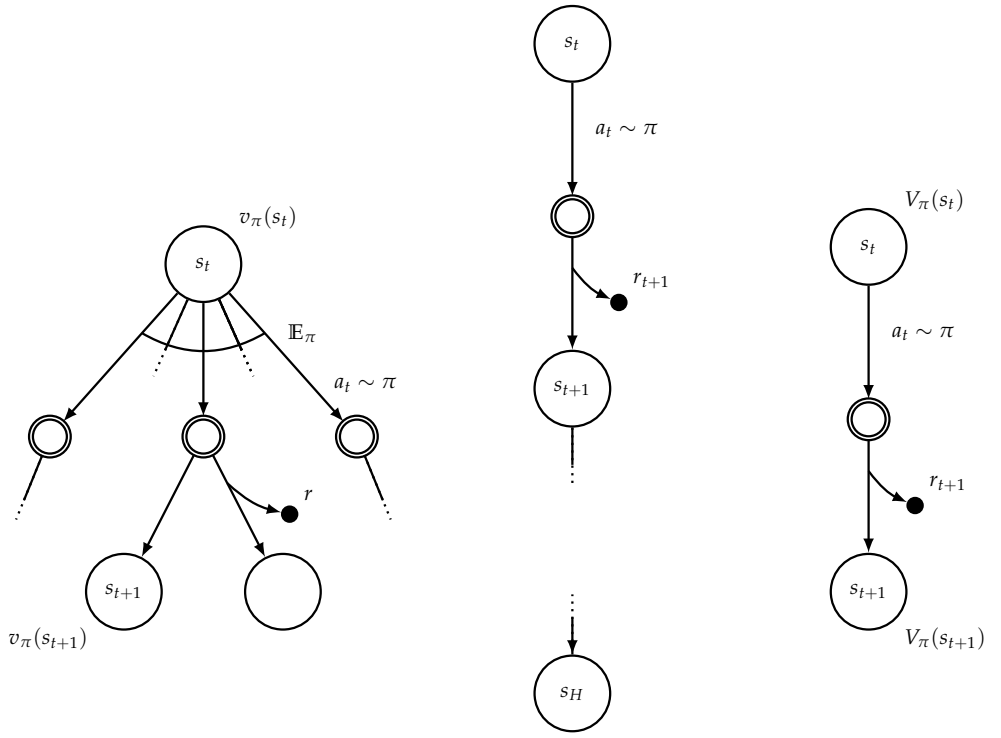
In the case of finite state spaces and action spaces, the Bellman equation defines a system of $|\mathcal{S}|$, with $|\mathcal{S}|$ being the number of states, non-linear equations with d unknowns. If the system's dynamics are known, this system of equations can be solved to optimality with Dynamic Programming (DP). Figure 2.4A shows the backup diagram for DP. In the case of unknown dynamics, Monte-Carlo (MC) methods can be used to approximate the stochastic variables from MC samples of agent-environment interactions. The backup diagram for MC methods is depicted in Fig. 2.4B. In most robotic control tasks, neither state spaces nor actions spaces are discrete, nor are the system dynamics known apriori. In that case, one has to

resort to powerful function approximators to model the (action-)value functions or the dynamics of the MDP.

2.2.1 Model-Free Reinforcement Learning

This section reviews some of the key algorithms and results from model-free RL. In model-free RL, the model of the MDP is assumed to be unknown. Value functions and policies are learned from data collected from interactions of the agent with the environment. No explicit model of the system dynamics or the reward function is learned. Section 2.2.1.1 and Sec. 2.2.1.2 discuss value-based RL methods. Section 2.2.1.3 presents policy gradient methods and Sec. 2.2.1.4 reviews the actor-critic framework that combines ideas from value-based and policy gradient methods.

2.2.1.1 Temporal Difference Learning and Bootstrapping



(A) Dynamic Programming (B) Monte-Carlo (C) Temporal Difference

FIGURE 2.4: Backup diagrams for (A) DP-, (B) MC- and (C) TD-learning.

Temporal Difference (TD) learning (Richard S. Sutton, 1988) aims to learn a value function by approximating the Bellman equation in various ways. Conceptually, Temporal Difference (TD) learning can be placed between Dynamic Programming (DP) and Monte-Carlo (MC) methods. With the former, it shares the idea of updating value estimates with estimates of the value of the next state. This is also called bootstrapping. MC methods, on the other hand, wait for the episode to terminate to compute a value estimate from the actual returns of the environment. Bootstrapping

happens in the transition from the second to the third line in the following equation:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi \left[G_t \mid S_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} \mid S_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma V_\pi(S_{t+1}) \mid S_t = s \right] \end{aligned} \tag{2.19}$$

where the true expected future return G_{t+1} is replaced with the value estimate $V_\pi(s_{t+1})$ of the next state. Similar to MC methods, the value function is learned directly from raw experience, that is, from transition tuples that are collected while the agent interacts with the environment:

$$\mathcal{D} = \{(s_i,\ a_i,\ r_i,\ s_{i+1})\}_{i=0}^{|\mathcal{D}|-1}. \tag{2.20}$$

TD learning is an iterative process and the update rule for the (approximate) value function V is defined as:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right],\ \forall s \in \mathcal{S}, \tag{2.21}$$

where $\alpha > 0$ is called the learning rate and the term in the square brackets is called the td-error δ_t:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t). \tag{2.22}$$

Intuitively, the value function converges if $\delta_t \to 0,\ \forall t$, that is, the value function converges if the value estimate of the current state is equal to the instantaneous reward r_{t+1} plus the discounted value of the next state $V(s_{t+1})$.

Figure 2.4c shows the backup diagram of TD-learning alongside the backup diagrams of DP and MC.

2.2.1.2 Q-Learning

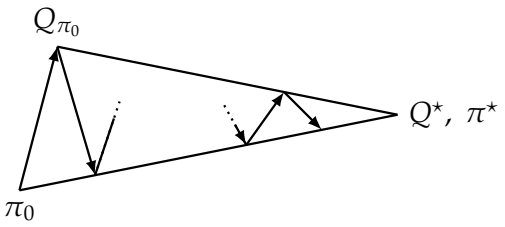


FIGURE 2.5: Update scheme of Generalized Policy Iteration (GPI): Alternating between policy evaluation and policy improvement steps. Over time, both the action-value function and the policy converge to the (approximate) optimal action-value function and policy.

Although the value function can be an interesting object in its own right, in RL, we are typically interested in learning a control policy π. TD-learning can be naturally extended from a pure estimation problem to a control problem by computing the action-value function and alternating between value iteration (or policy evaluation) and policy improvement steps. Eventually, this process will converge to the optimal action-value function and policy as shown in Fig. 2.5. The resulting off-policy TD control algorithm is called Q-learning (Watkins et al., 1992) and has the

following update rule:

$$Q(s,\ a) \leftarrow Q(s,\ a) + \alpha \left[r(s,\ a) + \gamma \max_{a'} Q(s',\ a') - Q(s,\ a) \right],\ \forall s \in \mathcal{S},\ a \in \mathcal{A}. \tag{2.23}$$

In Q-learning, the policy improvement step happens implicitly by locally maximizing the action-value function with respect to the action, i.e., by following the greedy policy. Watkins et al. (1992) provides the following theorem:

Theorem 2.2.3. *In the Q-learning algorithm with the update rule as defined in Eq. 2.23, bounded rewards* $|r_t| \leq r_{\max}$, $0 \leq \alpha < 1$ *and*

$$\sum_{t=0}^{\infty} \alpha_t(s,\ a) = \infty, \qquad \sum_{t=0}^{\infty} \alpha_t^2(s,\ a) < \infty, \qquad \forall s \in \mathcal{S},\ a \in \mathcal{A} \tag{2.24}$$

the action-value function Q *is guaranteed to converge to the optimal action-value function* $Q^\star$*:*

$$Q_n(s,\ a) \to Q^\star(s,\ a) \tag{2.25}$$

as $n \to \infty$ *with probability* 1.

The theorem guarantees the convergence of Q to $Q^\star$ even though Q-learning relies on several approximations.

2.2.1.3 Policy Gradient

In continuous action spaces $\mathcal{A} = \mathbb{R}^{n_a}$, the policy π_θ is typically modeled as a parametric function $\pi_\theta(A \mid S) = \pi(A \mid S;\ \theta)$ with model parameters $\theta \in \mathbb{R}^{n_\theta}$. Since computing the maximum over actions in the value function update can be challenging in continuous action spaces, policy gradient methods bypass the learning of a value function altogether. Instead, the policy is learned directly by optimizing an objective function, that is, by maximizing the expected future reward:

$$\begin{aligned} J(\theta) &= \int_{\mathcal{S}} \rho^\pi(s) v_\pi(s)\ \mathrm{d}s \\ &= \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} \pi_\theta(a|s) q_\pi(s,a)\ \mathrm{d}s\ \mathrm{d}a. \end{aligned} \tag{2.26}$$

In Eq. 2.26,

$$\rho^\pi(s) = \int_{\mathcal{S}_0} \sum_{t=0}^{\infty} \gamma^t \cdot p_0(s_0) \cdot p(s_0 \to s,\ t,\ \pi)\ \mathrm{d}s_0 \tag{2.27}$$

is the stationary state distribution induced by the policy-environment interaction, with the start-state probability $p_0(s_0)$, start-states $s_0 \in \mathcal{S}_0$, and $p(s_0 \to s,\ t,\ \pi)$ being the probability of ending up in state s after t steps of following policy π.

The advantage of policy gradient methods is that the gradient of Eq. 2.26 with respect to θ can be computed efficiently. This is in contrast to the max-operation in Q-learning methods that becomes intractable in the case of continuous action spaces.

The objective in Eq. 2.26 can be optimized with approximate gradient ascent:

$$\theta_{i+1} = \theta_i + \alpha \widehat{\nabla_\theta J(\theta_t)}, \tag{2.28}$$

with $\alpha > 0$ being the learning rate and $\widehat{\nabla J(\theta_t)}$ being the stochastic estimate of the gradient. To compute the gradient of Eq. 2.26, π_θ has to be differentiable with respect

to θ which can be easily achieved by choosing an appropriate model class. However, computing the gradient of the stationary state distribution is problematic because it depends on the policy and the environment to which we do not have access.

The policy gradient theorem (R. J. Williams, 1992) provides a solution to this problem:

Theorem 2.2.4 (Policy Gradient Theorem)**.** *For any differentiable policy π_θ and the objective function defined in Eq. 2.26, the policy gradient is proportional to:*

$$\begin{aligned}\nabla_\theta J(\theta) &\propto \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} q_\pi(s,\ a) \nabla_\theta \pi(a \mid s;\ \theta)\ \mathrm{d}s\ \mathrm{d}a \\ &= \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} q_\pi(s,\ a) \pi(a \mid s;\ \theta) \nabla_\theta \log \pi(a \mid s;\ \theta)\ \mathrm{d}s\ \mathrm{d}a \\ &= \mathbb{E}_{s\sim\rho,\, a\sim\pi}\left[q_\pi(s,\ a) \nabla_\theta \log \pi(a \mid s;\ \theta)\right] \end{aligned} \tag{2.29}$$

with $\log \pi(a \mid s;\ \theta)$ being the gradient of the log-likelihood or Score function.

For the proof see Appendix A.2. The key insight of the policy gradient theorem is that the computation of the derivative of the stationary state distribution can be avoided altogether by using the likelihood ratio trick (second to third line), which removes any model dependency from the calculation. An intuitive interpretation of the policy gradient is that it moves the probability mass of $\pi(a \mid s;\ \theta)$ onto actions that maximize $Q_\pi(s,\ a)$.

The expectation in Eq. 2.29 can be approximated from MC rollouts in the environment, that is, by approximating q_π with MC samples of the expected future return G_t. The resulting algorithm is known as the *REINFORCE* algorithm (R. J. Williams, 1992). One downside of MC-based approaches is that they typically have high variances in their gradient estimates. This variance can be reduced by adding a so-called baseline to the policy gradient. In the next section, one particular type of baseline in the context of actor-critic methods will be discussed.

2.2.1.4 The Actor-Critic Framework

The actor-critic framework (Andrew G. Barto et al., 1983; Richard S Sutton, McAllester, et al., 2000) combines the policy gradient with TD-learning and bootstrapping. This has two significant advantages: (1) The variance of policy gradient methods is reduced compared to MC-based versions of the policy gradient like *REINFORCE* and (2) it naturally deals with continuous control problems.

In the actor-critic framework, an actor or policy and a critic are learned from the raw experience of the agent in the environment. In continuous state spaces and action spaces, the actor and critic are modeled by function approximators.

The learned critic acts similar to a baseline in the policy gradient and thereby reduces the variance of the estimated gradient. Generally, a baseline b enters the policy gradient in the following way:

$$\nabla_\theta J(\theta) = \int_{\mathcal{S}} \rho(s) \int_{\mathcal{A}} \left(q_\pi(s,\ a)) - b(s)\right) \nabla_\theta \pi(a \mid s;\ \theta)\ \mathrm{d}s\ \mathrm{d}a. \tag{2.30}$$

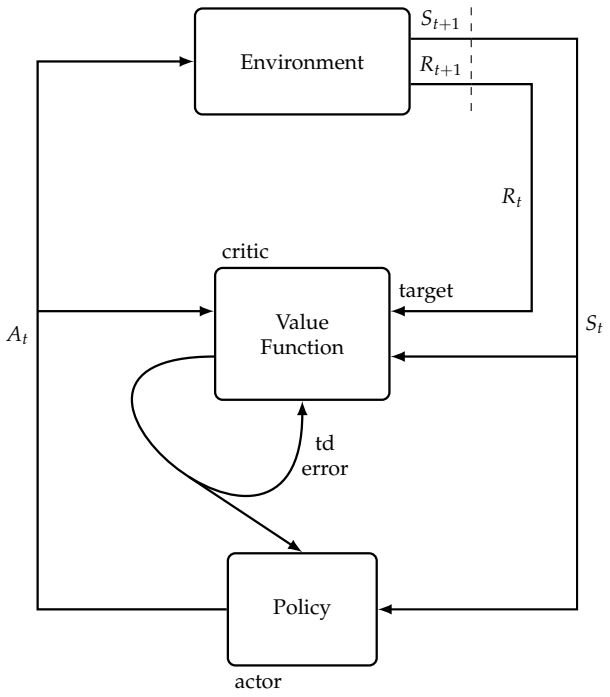


FIGURE 2.6: Schematics of the actor-critic framework. The monolithic agent is split up into an actor and a critic. The actor receives the current state from the environment and produces an action. The critic gets the state and reward from the environment and the action from the agent. The critic updates its current estimate of the state-action value function and provides a learning signal to the agent.

The only requirement for the baseline is that it only depends on the state but not on the action. In that case, it follows that:

$$\begin{aligned} & \int_{\mathcal{S}} \rho(s) \int_{\mathcal{A}} b(s) \nabla_\theta \pi(a \mid s;\, \theta)\, \mathrm{d}s\, \mathrm{d}a \\ = & \int_{\mathcal{S}} \rho(s) b(s) \nabla_\theta \int_{\mathcal{A}} \pi(a \mid s;\, \theta)\, \mathrm{d}s\, \mathrm{d}a \\ = & \int_{\mathcal{S}} \rho(s) b(s) \nabla_\theta 1\, \mathrm{d}s \\ = & \, 0. \end{aligned} \tag{2.31}$$

Hence, the expected value of the policy gradient remains unchanged if an appropriate baseline is added:

$$\begin{aligned} \nabla_\theta J(\theta) &= \int_{\mathcal{S}} \rho(s) \int_{\mathcal{A}} \left(Q_\pi(s,\, a)) - b(s)\right) \nabla_\theta \pi(a \mid s,\, \theta)\, \mathrm{d}s\, \mathrm{d}a \\ &= \int_{\mathcal{S}} \rho(s) \int_{\mathcal{A}} Q_\pi(s,\, a)) \nabla_\theta \pi(a \mid s,\, \theta)\, \mathrm{d}s\, \mathrm{d}a. \end{aligned} \tag{2.32}$$

Nonetheless, the variance of the policy gradient can still be affected by the baseline:

$$\begin{aligned}
&\mathbb{E}_\pi\left[\left(\left(Q_\pi(s,\ a) - b(s)\right)\nabla_\theta \log \pi(a \mid s;\ \theta)\right)^2\right] \\
&\quad - \mathbb{E}_\pi\left[\left(Q_\pi(s,\ a) - b(s)\right)\nabla_\theta \log \pi(a \mid s;\ \theta)\right]^2 \\
=&\mathbb{E}_\pi\left[\left(\left(Q_\pi(s,\ a) - b(s)\right)\nabla_\theta \log \pi(a \mid s,\ \theta)\right)^2\right] \\
&\quad - \mathbb{E}_\pi\left[Q_\pi(s,\ a)\nabla_\theta \log \pi(a \mid s;\ \theta)\right]^2. \qquad (2.33)
\end{aligned}$$

Replacing b with $V_\pi(s)$ in Eq. 2.31 gives rise to the definition of the advantage function:

$$\begin{aligned}
A_\pi(s_t,\ a_t) &= q_\pi(s_t,\ a_t) - b(s) \\
&\propto r(s_t,\ a_t) + V_\pi(s_{t+1}) - V_\pi(s_t). \qquad (2.34)
\end{aligned}$$

The policy is updated by following the gradient that maximizes the advantage, while the critic can be updated from MC rollouts in the environment. If a function approximator is used to model the critic, one can ask whether the function approximator introduces any bias in the policy gradient. It turns out that the policy gradient remains unbiased as long as the compatible function approximation theorem (Richard S Sutton, McAllester, et al., 2000) holds:

Theorem 2.2.5 (Compatible Function Approximator)**.** *Let the critic $Q(s,\ a;\ w)$ be modeled by a function approximator with the model parameters w. The policy gradient*

$$\nabla_\theta J(\theta) = \int_{\mathcal{S}} \rho(s) \int_{\mathcal{A}} Q(s,\ a;\ w)\nabla_\theta \pi(a \mid s,\ \theta). \qquad (2.35)$$

is unbiased, if the critic gradient is compatible with the actor score function

$$\nabla_w Q(s,\ a;\ w) = \nabla_\theta \log \pi(s,\ a;\ \theta) \qquad (2.36)$$

and if the model parameters w minimize

$$\epsilon = \int_{\mathcal{S}} \rho(s) \int_{\mathcal{A}} \pi(s,\ a;\ \theta)(Q_\pi(s,\ a) - Q(s,\ a;\ w))^2 \,\mathrm{d}s\,\mathrm{d}a. \qquad (2.37)$$

It is worth mentioning that all of the results discussed so far are exact only in the tabular case or with linear function approximators. For complex problems like robotic manipulation tasks or challenging games, linear function approximators are typically insufficient to capture all the complicated dependencies. Therefore, one has to resort to more powerful function approximators giving rise to the field of deep RL.

2.2.2 Deep Reinforcement Learning

Deep RL builds on the successes of Deep Learning (DL) in domains such as the classification of natural images (Krizhevsky et al., 2012), natural language processing (Radford et al., 2018), recommendation systems (Elkahky et al., 2015) and many others by applying similar techniques to challenging RL problems like robotic manipulation tasks and complex games.

DL enters RL if state spaces or action spaces become too large to be handled by tabular approaches and the models are too complex for linear function approximators. Popular deep RL algorithms (Mnih, Kavukcuoglu, Silver, Rusu, et al.,

2015; Mnih, Badia, et al., 2016; Haarnoja et al., 2018) use powerful non-linear function approximators such as deep Neural Networks (NNs) and massive amounts of data (in the order of 10^6–10^7 data points) to learn the (action-)value function, the policy, or a model of the MDP. On-policy methods like *Trust Region Policy Optimizatio (TRPO)* (Schulman, Levine, et al., 2015) and *Proximal Policy Optimization (PPO)* (Schulman, Wolski, et al., 2017) optimize a parametrized policy directly via the policy gradient. Off-policy methods like *Deep Deterministic Policy Gradient (DDPG)* (Lillicrap et al., 2016) or *Soft Actor Critic (SAC)* (Haarnoja et al., 2018) learn a value-function and are more akin to actor-critic methods with numerous adjustments to accommodate for the training of deep NNs. In fitted value iteration algorithms (G. J. Gordon, 1995; M. A. Riedmiller, 2005) a value function is learned offline on all the experience collected so far. It, therefore, resembles much more the classical supervised learning setup in ML than the RL setting. One downside of deep RL is that many of the results established in the previous sections do not hold (exactly) once deep NNs are used somewhere inside the methods. Naively applied to the RL setting, they typically result in notoriously unstable training (Tsitsiklis et al., 1997). Furthermore, the convergence of RL methods with deep NNs cannot be guaranteed because of the deadly triad (Richard S Sutton, 1995), which results from the combination of function approximation, bootstrapping, and off-policy training. Over time, many ad-hoc solutions were developed that try to alleviate the risk of divergence during training due to the deadly triad:

Lin (1992) introduces the concept of experience replay, or a replay buffer, to deep RL. In experience replay, all the past interactions with the environment are stored in a dataset from which elements are sampled i.i.d. during training. This removes any temporal correlation between individual samples and smoothens the training distribution over different behaviors. It also increases efficiency because training points are seen more often during training. It can also decouple the behavior policy, or data-collection policy, from the learned policy. In the extreme case, this leads to offline RL (see Levine, Kumar, et al. (2020) for a survey of the field).

Mnih, Kavukcuoglu, Silver, Graves, et al. (2013) uses target networks to compute the Q-targets in the Bellman update. However, in contrast to supervised learning, the targets in the Bellman update depend on the parameters that are being optimized, making the updates notoriously unstable. By replacing the Q-network in the target computation of the Bellman update with a previous, immutable iteration of the Q-network, the training can be stabilized significantly.

Computing the max-operation in the Bellman update can be challenging or even intractable for large or continuous action spaces. Lillicrap et al. (2016) uses the action proposed by the current policy to approximate the max-operation, while Haarnoja et al. (2018) uses samples from a probabilistic policy. *QT-OPT* (Kalashnikov et al., 2018) uses an optimization scheme to optimize for the next action explicitly.

Many more tricks, including delayed policy updates and target policy smoothing (Fujimoto et al., 2018) were developed to ease the training of deep RL agents.

2.2.3 Model-Based Reinforcement Learning

So far, model-free RL methods have been discussed. These methods assume the model to be unknown and therefore learn a policy or value function directly from experience without modeling the system dynamics explicitly.

An alternative approach to utilize the data gathered from interactions with the environment is to learn an explicit model of the MDP. The class of models that can be used to approximate MDPs is extensive, including unstructured and structured

models (Martius and Lampert, 2017; Burgess et al., 2019), state- (Wu et al., 2015), observation- (Ebert et al., 2018) and latent state-transition models (Ha et al., 2018), parametric (Lenz et al., 2015; Fu et al., 2016; Gal et al., 2016) and non-parametric (Kocijan et al., 2004; Nguyen-Tuong et al., 2008) models, and many more. One of the simplest model classes are unstructured state-transition models, i.e., a function $f\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,\ 1]$ that maps states-action tuples to a distribution over next states:

$$S_{t+1} \sim \mathrm{P}(\cdot \mid S_t,\ A_t) = f(S_t,\ A_t;\ \zeta), \tag{2.38}$$

with ζ being the parameters of the model. Equation 2.38 is an approximation of the MDP transition kernel $P^a_{ss'}$ defined in Eq. 2.1. Given a dataset $\mathcal{D} = \{(s_i, a_i, r_i, s_{i+1})\}_{i=0}^{|\mathcal{D}|-1}$ of transitions, the model parameters can be learned via maximum likelihood learning:

$$\zeta \leftarrow \arg\max_{\zeta} \mathbb{E}_{\mathcal{D}} \left[\log f(s_{t+1} \mid s_t,\ a_t;\ \zeta)\right]. \tag{2.39}$$

Once a model of the MDP is learned, it can be used in various ways. The following sections give an overview of the different use cases.

Notation

In optimal control, it is common to denote states by x and control inputs or actions by u. To have a unified notation throughout this work, the RL-based notation is adopted in which states are denoted by s and actions are denoted by a.

2.2.3.1 Model-Based Data Generation

These algorithms are also known as Dyna (Richard S Sutton, 1990; Richard S Sutton, 1991a; Richard S Sutton, 1991b) style algorithms. They iterate between: (1) Data collection with the current policy in the MDP. The interaction data is used to improve the model. (2) Policy improvement by running a model-free algorithm with imagined data from the model.

Notable entries of this type of algorithm are World Models (Ha et al., 2018), SimPLe (Kaiser et al., 2020), and Model-Ensemble Trust-Region Policy Optimization (Kurutach et al., 2018). Janner et al. (2019) provides a guarantee for a monotonic improvement of Dyna style algorithms by constructing a bound on the returns of the policy in the true MDP when using model rollouts for improving the policy:

$$G[\pi] \geq \hat{G}[\pi] - C(\epsilon_m,\ \epsilon_\pi), \tag{2.40}$$

with $G[\pi]$ being the return of the policy in the true MDP and $\hat{G}[\pi]$ being the return of the policy in the approximate MDP. Two sources of errors contribute to the gap C between the actual and approximated returns. First, a generalization error due to the difference in the expected loss and the empirical loss of the supervised learning scheme (PAC generalization bound (Shalev-Shwartz et al., 2014)):

$$\epsilon_m = \max_t \mathbb{E}_{s \sim \rho_t^{\hat{\pi}}} \left[D_{\mathrm{TV}}(D(P^a_{ss'} \parallel f(s' \mid s,\ a;\ \zeta)))\right], \tag{2.41}$$

with D_{TV} being the total variation distance and $\rho_t^{\hat{\pi}}$ being the time-dependent state distribution of the data-collecting policy. Second, the distributional shift due to the policy improvement step after which the policy might encounter states that were not seen before:

$$\epsilon_\pi \geq \max_{s \sim \rho_t^{\pi}} D_{\mathrm{TV}}(D(\pi(\cdot \mid s) \parallel \pi_{\mathcal{D}}(\cdot \mid s))). \tag{2.42}$$

Using these two sources of errors, it follows:

Theorem 2.2.6. *Given Eq. 2.41 and Eq. 2.42. The returns of the policy in the true MDP are bounded by:*

$$G[\pi] \geq \hat{G}[\pi] - \left[\frac{2\gamma r_{\max}(\epsilon_m + 2\epsilon_\pi)}{(1-\gamma)^2} + \frac{4 r_{\max}\epsilon_\pi}{(1-\gamma)}\right]. \tag{2.43}$$

For the proof, see Janner et al. (2019).

2.2.3.2 Model Derivatives

If available, analytical model derivatives (and potentially cost function derivatives) can be used directly for policy search. To this end, gradients of the RL objective (Eq. 2.26) are computed by backpropagating through the frozen forward model (see the dashed lines in Fig. 2.7) in the policy improvement step. Some notable entries in this line of work are *Probabilistic Inference for Learning Control (PILCO)* (M. Deisenroth et al., 2011), *Iterative Linear Quadratic Regulator (iLQR)* (Tassa et al., 2012), and *Guided Policy Search (GPS)* (Levine and Abbeel, 2014).

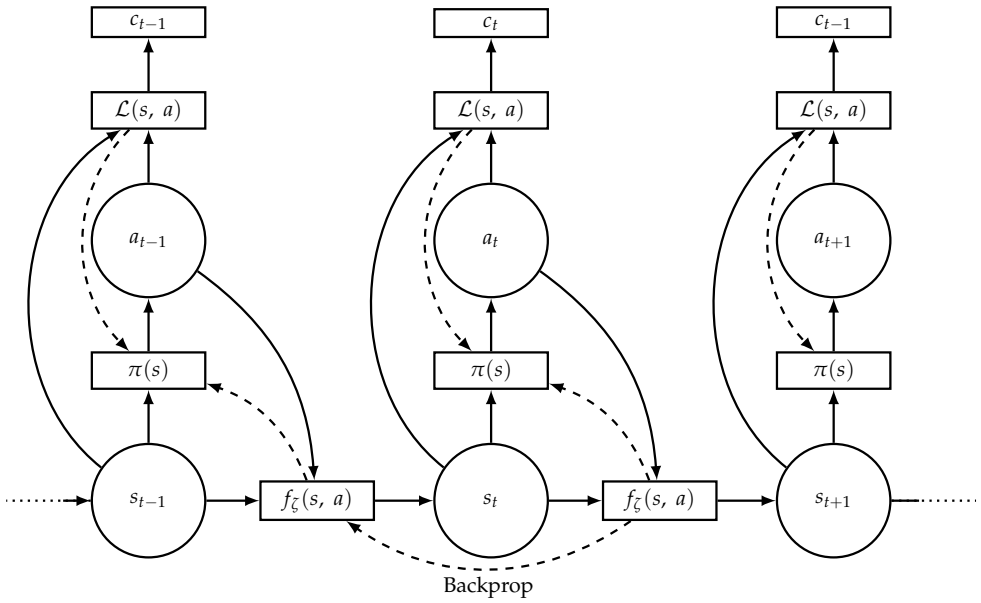


FIGURE 2.7: Graphical depiction of backpropagation through time. In the policy improvement step, the parameters ζ of the forward model f are frozen, and analytical gradients (dashed lines) are computed through the model (and potentially the cost function) to improve the policy.

2.2.3.3 Sampling-Based Planning with Model Predictive Control

The goal of planning-based methods is to optimize an action sequence:

$$(a_h)_{h=0}^{H-1} = (a_0, \ldots, a_{H-1}) \tag{2.44}$$

such that a performance objective:

$$J = \varphi(s_{t'+H}) + \sum_{t=t'}^{t'+H} \mathcal{L}(s_t,\ a_t,\ t), \tag{2.45}$$

with $\varphi\colon \mathbb{R}^{n_s} \to \mathbb{R}$ being a terminal cost function and $\mathcal{L}\colon \mathbb{R}^{n_s} \times \mathbb{R}^{n_a} \times \mathbb{R} \to \mathbb{R}$ being a step-wise cost function, is minimized. The step-wise cost function in planning corresponds to the step-wise reward function in (model-free) RL with a flipped sign. Hence, the maximization problem in RL becomes a minimization problem in planning. The terminal cost corresponds to a value function.

The optimization problem is subject to first-order (stochastic) dynamic constraints:

$$s_{t+1} = f(s_t,\ a_t,\ t,\ \eta(t)), \tag{2.46}$$

input constraints:

$$a_t \in \Omega, \tag{2.47}$$

and algebraic path constraints:

$$h(s_t,\ a_t,\ t) \leq 0, \qquad h\colon \mathbb{R}^{n_s} \times \mathbb{R}^{n_a} \times [t,\ t+H] \to \mathbb{R}^{n_h}. \tag{2.48}$$

In Eq. 2.46, $\eta(t) = \eta(s_t,\ a_t)$ is a random variable modeling the noise in the system. In Eq. 2.47, Ω is an admissible compact region. In contrast to optimal control, where the dynamics is often assumed to be linear, and the cost function is assumed to be convex, in Model-Based Reinforcement Learning (MBRL) the dynamics is learned and highly non-linear. Moreover, the cost function is non-convex. Hence, gradient-free or zero-order optimization methods are a popular choice for optimizing the action sequence in Eq. 2.44.

Model Predictive Control Planning methods typically compute an action sequence offline and execute it in an open-loop fashion. In open-loop control, the entire action sequence is executed in one shot without taking any feedback from the system into account. Only after the action sequence is executed the planner receives feedback from the system. However, in contact-rich robotic control tasks with non-smooth dynamics, a feedback controller or closed-loop policy is preferred because it is much more reactive. *Model Predictive Control (MPC)*, or receding horizon control, transforms an open-loop planning method into a feedback controller according to the following recipe: (1) Optimize a trajectory using model rollouts. (2) Execute only the first action of the optimized trajectory in the environment. (3) Let the system evolve according to the system dynamics. (4) Update the model's internal state with feedback from the system. (5) Repeat.

The name receding horizon control comes from the fact that the planning horizon H is typically much shorter than the task horizon T. Hence, after each iteration of the MPC algorithm, the same problem is solved again with a horizon moved 1 step forward in time.

Random Shooting In Random Shooting (RS) (Richards, 2005; Rao, 2009), many random action sequences are generated by drawing actions from a fixed sampling distribution. The resulting trajectories are ranked according to the performance index J. Then, the best action sequence is selected and executed in the environment. RS can be combined with model-free algorithms to increase asymptotic performance. For instance, Nagabandi et al. (2018) extracted a policy from trajectories generated with RS and further fine-tuned the policies with model-free algorithms.

CEM for Trajectory Optimization In the Cross-Entropy Method (CEM) (R. Y. Rubinstein, 1997) for trajectory optimization (Chua et al., 2018), the fixed sampling distribution is replaced with an adaptive sampling scheme. The most common choice

for the sampling distribution is the Gaussian distribution $a_t \sim \mathcal{N}(\mu_t, \Sigma_t)$ with parameter vectors $\mu_t \in \mathbb{R}^{n_a}$ for the means and $\Sigma_t \in \mathbb{R}^{n_a \times n_a}$ for the covariance matrix. Often, just the variances $\sigma = \text{diag}(\Sigma) = \mathbb{R}^{n_a}$ are used because the vector of variances needs much fewer samples to be estimated faithfully than the matrix of covariances.

The CEM-MPC policy has two optimization loops: (1) The outer optimization loop, or *MPC-iteration*, generates an action for every step in the environment. (2) The inner optimization loop, or *CEM-iteration*, optimizes in each iteration N action sequences $(a_h^n)_{h=0}^{H-1}, n = 1, \ldots, N$. Executing these action sequences inside the model results in N imagined trajectories:

$$\hat{\tau}^n = (s_t, a_0^n, \hat{s}_{t+1}^n, a_1^n, \ldots), \tag{2.49}$$

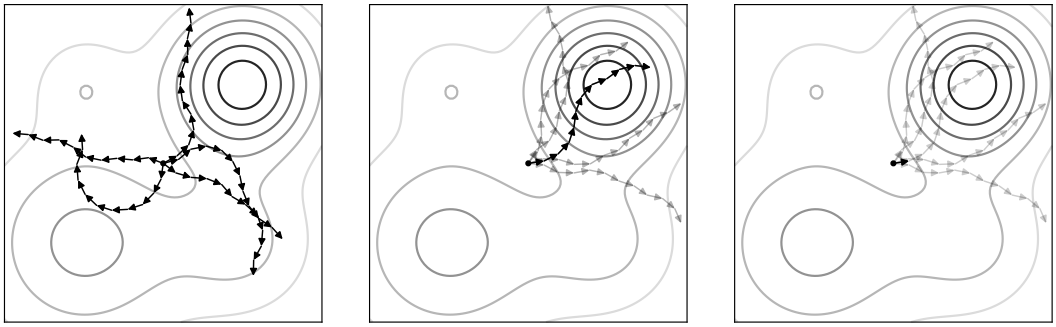
with s_t being the ground-truth MDP state at time step t and $\hat{s}_{t+h+1}$ being the model prediction, or imagined state, at planning step h. The action sequences are ranked according to the performance metric J and the K highest ranking action sequences, or elites, are used to update the parameters of the sampling distribution according to:

$$\mu_i \leftarrow \frac{\sum_{k=1}^{K} a_i^k}{K} \tag{2.50}$$

and:

$$\sigma_i \leftarrow \frac{\sqrt{\sum_{k=1}^{K} (a_i^k - \mu_i)^2}}{K-1}. \tag{2.51}$$

This process is repeated multiple times to optimize the action sequence in the CEM-iteration. Figure 2.8 depicts one loop of the CEM-MPC policy. Figure 2.8A shows the N action sequences (black arrows) that were sampled from the yet uninformed sampling distribution in the first CEM-iteration. From these trajectories, a set of elites gets selected based on their performance w.r.t. J. The optimization landscape induced by J is visualized as contours. Figure 2.8B shows N new trajectories that were sampled from the updated sampling distribution in the second CEM-iteration. Figure 2.8C shows the action of the winning trajectory that eventually gets executed in the environment.



(A) Model rollouts with uninformed sampling distribution.

(B) Model rollouts with updated sampling distribution.

(C) The first action of the best trajectory gets executed.

FIGURE 2.8: The three phases of the CEM-MPC policy. Connected arrows depict model rollouts. The optimization landscape induced by J is visualized as contours.

2.2.3.4 Value-Equivalent Predictions

The idea behind the value-equivalent principle (Grimm et al., 2020) is to learn models m that are specifically tailored to the control setting rather than the most general models possible. According to the value-equivalence principle, two models are value equivalent if they produce the same Bellman updates given a policy π and a set of functions:

Definition 2.2.2 (Value Equivalence (Grimm et al., 2020))**.** *Let Π be a set of policies and let $\mathcal{V}$ be a set of functions. We say that models m and $\tilde{m}$ are value equivalent with respect to Π and $\mathcal{V}$ if and only if:*

$$\mathcal{T}_\pi v = \tilde{\mathcal{T}}_\pi v, \ \forall \pi \in \Pi, v \in \mathcal{V}, \tag{2.52}$$

where $\mathcal{T}_\pi$ and $\tilde{\mathcal{T}}_\pi$ are the Bellman operators induced by m and $\tilde{m}$, respectively.

The general hope is that the models in the class of value-equivalent models are easier to learn than more general models since they make use of task-specific knowledge.

2.2.3.5 Model-Based Reinforcement Learning and Optimal Control

What differentiates MBRL from optimal control is that MBRL typically makes minimal or no prior assumptions about the functional form of the dynamics or the cost function. Instead, in MBRL, models are entirely learned from data and with very general parametric or non-parametric function approximators like NNs or Gaussian Proccesses (GPs). On the contrary, optimal control frameworks such as the *Linear–Quadratic Regulator (LQR)* (Kwakernaak et al., 1969) require a specific functional form of the system dynamics and cost function. Nonetheless, optimal control methods can already involve learned components, for instance, by using system identification (Åström et al., 1971). In system identification, the functional form of the dynamics or the cost function is known apriori. Data collected from the system under consideration is used to fit the analytical models to the system. For instance, if rigid-body dynamics is used to model the movement of a robot, it is common to fit the mass and inertia matrices from data collected from the robot.

2.2.4 Hierarchical Reinforcement Learning

The goal of HRL (Andrew G Barto et al., 2003; Pateria et al., 2021) is to decompose a difficult and potentially long-horizon problem into smaller sub-problems through hierarchical abstractions. Figure 2.9 depicts the general idea of HRL: The low-level policy π^{lo} interacts with the environment by receiving the current MDP state s_t from the environment and sending low-level commands or actions a_t to the environment. The high-level policies $(\pi^{\text{mid}}, \ldots, \pi^{\text{hi}})$ do not interact with the environment directly but only with the policies on the lower levels. The policies in the different levels of the hierarchy are typically conditioned on the action from the policy above (also known as universal policies and similar to the concept of Universal Value Function Approximators (UVFAs) (Schaul et al., 2015)). In that way, the policy on a higher level can modulate the behavior of a policy on a lower level:

$$\pi^i = \pi(\phi^{i-1}(s), \ \pi^{i+1}), \tag{2.53}$$

with $\phi \colon \mathcal{S} \to \mathcal{S}'$ being a state-embedding function mapping from the state-space $\mathcal{S}$ to an abstract state-space $\mathcal{S}'$ and π^i being the policy at the i-th level of the hierarchy.

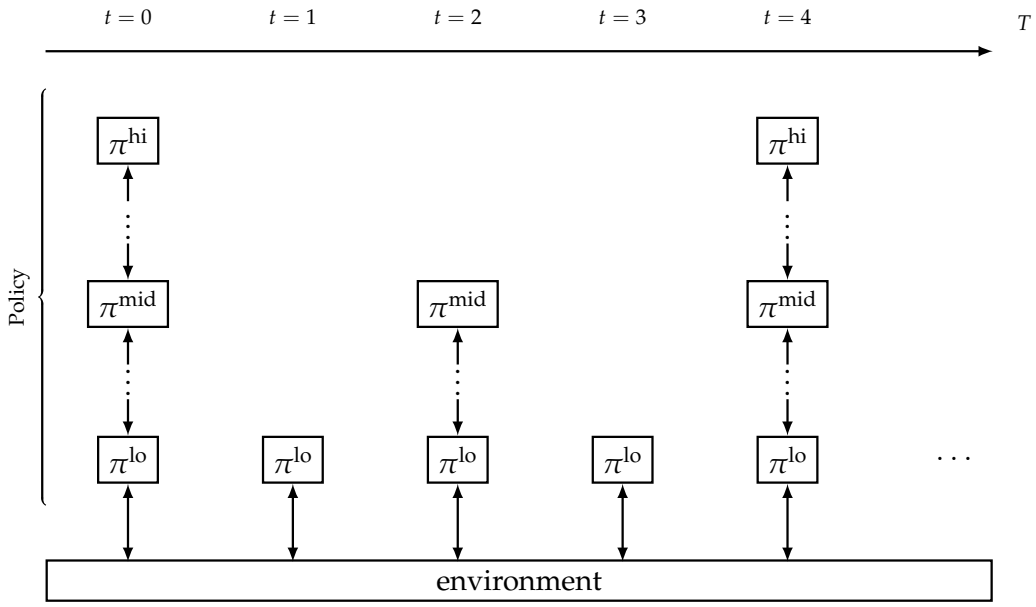


FIGURE 2.9: Graphical depiction of a HRL policy. The low-level policy interacts with the environment at every step. It receives the environment state s_t and sends actions a_t to the environment. The mid- and high-level policies interact only with the lower-level policies on a coarser temporal scale.

Similar to MDPs for RL, Semi-Markov Decision Processes (SMDPs) (Baykal-Gürsoy, 2010) can be used to formalize problems that need to be solved at the higher levels of the HRL policy. SMDPs add an additional timing component to MDPs parametrizing the number of steps for which an action gets executed. Hierarchical abstraction can be achieved in multiple ways:

State Abstraction State abstraction (T. Dietterich, 1999; Jonsson et al., 2000) can be implemented by embedding the MDP state space $\mathcal{S}$ into an abstract state space $\mathcal{S}'$. The transformation from $\mathcal{S}$ to $\mathcal{S}'$ typically involves some form of compression, for instance, by using the information bottleneck (J. Kim et al., 2021) or an attention mechanism (Y. Chen et al., 2019).

Temporal Abstraction Temporal abstraction (Richard S Sutton, Precup, et al., 1999; T. G. Dietterich, 2000) can be achieved by letting the policies operate on different levels of temporal granularity. The lowest-level policy typically interacts with the environment directly in every single step. However, the higher-level policies might act only in fixed time intervals $\Delta t > 1$ or only if certain conditions are met, for instance, if the lower policy reaches a goal set by the higher-level policy. Temporal abstraction can help mitigate the credit assignment problem in two ways: (1) Since the actions of the higher-level policies are temporally extended, rewards in the distant future can be effectively backpropagated over longer time scales. (2) Lower-level policies solve more manageable and shorter sub-problems of the original task such that the credit assignment problem is less pronounced.

Reward Abstraction The policies in the different levels of the hierarchy are learned with RL methods, and each level might receive its unique reward signal. The highest-level RL agent typically receives the original environment reward while all the lower-level policies are trained on internal rewards computed by the higher levels. One of the most straightforward implementations of reward abstraction is to learn goal-reaching policies on the lower levels with goals provided by the higher levels (Dayan et al., 1992). Like temporal abstraction, reward abstraction can help ease the credit assignment problem.

Using hierarchical abstractions can make the HRL problem non-stationary. For instance, a lower-level policy might not be able to reach any targets at the beginning of the training but improves over time, or the execution time of a lower-level policy decreases with more training. The issue of a non-stationary training distribution can be counteracted by subgoal re-labeling (Nachum et al., 2018) or by introducing timed sub-goals (Gürtler et al., 2021).

Instead of learning a complicated policy, HRL can also be used for efficient exploration. Instead of action-level exploration, the higher-level policies can be used for subtask or goal exploration (Jong et al., 2008; Nachum et al., 2018; Forestier et al., 2020).

2.2.5 Intrinsically Motivated Reinforcement Learning

The goal of Intrinsically Motivated Reinforcement Learning (IMRL) and developmental RL is to design agents that explore the environment by setting their own goals instead of solving pre-defined goals or maximizing external rewards. Inspired by how infants and children learn, the hope of IMRL is to design genuinely autonomous agents that build a useful representation of the world and acquire new skills in an open-ended fashion. Aubret et al. (2019) and Colas (2021) provide thorough surveys of the field.

IM is closely related to the concept of embodiment (Bongard et al., 2003; Asada et al., 2009; Cangelosi et al., 2015). The general idea of embodied learning is that the physical constraints of a learning system profoundly influence the representations and skills the system can learn. Therefore, the learning system must be understood as a unit between learning methods and embodiment (Baranes et al., 2013a; Martius, Der, et al., 2013; Gumbsch, Butz, et al., 2019). Another closely related field of research is curriculum learning (Portelas et al., 2020). This field studies how a learning curriculum can be built that facilitates acquiring new and progressively harder skills. For instance, Florensa et al. (2018) learns a Generative Adversarial Network (GAN) to generate goals of intermediate difficulty. The same idea for goal-generating policies is used in Sukhbaatar et al. (2018) and Campero et al. (2021). IMRL based methods can be divided into two major classes:

Knowledge-Based IM Knowledge-Based IM (see Linke et al. (2020) for a survey of the field) is about the agent's belief of how the world works and what it actually observes. For instance, an agent can be motivated to recreate and explore surprising situations (Achiam and Sastry, 2017), where surprise can be defined as the disagreement between the predictions of a learned world model and the actual observations from the environment (Schmidhuber, 1991; Pathak et al., 2017). Approaches based on learning progress (Schmidhuber, 1991; Lopes et al., 2012; K. Kim et al., 2020) concentrate learning efforts on tasks the agent can learn. If a task is too difficult or too easy, the agent's interest in the task vanishes. Novelty (M. Bellemare et al., 2016) or

information gain (Houthooft et al., 2016) can be other measures to drive the intrinsically motivated exploration of an agent. In Schrodt et al. (2017), production rules are learned from sensorimotor experiences to move a video game character in a purely unsupervised fashion.

Competence-Based IM Competence-based IMRL agents are driven by the motivation to maximize their control over the environment, either by learning diverse sets of skills (Mouret et al., 2015) or by reaching self-imposed goals (Baranes et al., 2010; Santucci et al., 2016; Colas et al., 2019; Warde-Farley et al., 2019; Nair, Bahl, et al., 2020; Pong et al., 2020).

IM can also be used to learn better world models (Chitnis et al., 2021). The model can then drive further exploration (Sekar et al., 2020; Mendonca et al., 2021) or it can be used to solve downstream tasks.

3

Autonomous Hierarchical Skill Acquisition with Self-Guided Learning Curriculum

This chapter is based on:

Blaes, Vlastelica, Zhu, Martius (2019). "Control What You Can: Intrinsically Motivated Task-Planning Agent". In: *Advances in Neural Information Processing Systems (NeurIPS)*.

Contents

3.1 Introduction

This chapter presents an intrinsically motivated Hierarchical Reinforcement Learning (HRL) agent that learns to control its environment by utilizing structured models and intrinsically motivated self-play. To enable the agent to gain control over objects from just a few successful tries in compositional multi-object and open-ended environments, its design is inspired by nature.

Many if not all animals show behavior that is acquired not only through learning, but that is innate (Versace et al., 2015), i.e., that is genetically hard-wired (Kanwisher, 2010). Innate behavior increases the chances of survival of an animal and facilitates learning by providing helpful training samples to learn complex skills (Sherman Ross et al., 1957) in a world of countless possibilities. Reflexes are one of the most simple forms of innate behavior and often do not involve higher cognitive processes of the central nervous system. For instance, the patellar or knee-jerk reflex in humans is directly controlled by the spinal cord (Johns, 2014). Other reflexes serve as an initial stimulus to aid the learning of behaviors that are critical for survival. For example, the sucking reflex in mammals promotes the learning of breastfeeding behavior by encouraging the newborn to suck at any object close to the baby's mouth. In that way, the newborn is guaranteed to positively reinforce the sucking behavior once it is close to the mother's breast (Sherman Ross et al., 1957).

In precocial species, newborns show relatively mature behavior (Versace et al., 2015), like the complex escape behavior of prey animals (Miller et al., 2005), right or short after birth. On the contrary, newborns in altricial species are very dependent until long after birth. They have to learn most of their behaviors or skills throughout a prolonged maturation period. They conduct experiments and analyze the statistics of their observations to form intuitive theories about the world (Gopnik et al., 2004). Nonetheless, hard-wired skills like the ability of primates to identify other primates' faces (Scalaidhe et al., 1999) or the well-tuned relationship between teacher (adult) and student (infant) (Aitken, 2018) can facilitate the learning process. Since the opportunities and the number of possible skills to learn are almost endless, infants show a pronounced intrinsic motivation for self-play, often with any objects within their reach. The purpose may not be immediately apparent to us. However, to play is to manipulate, to *gain control*.

Similar to nature, where species with different degrees of pre-structured systems evolved, analogous design directions developed in Machine Learning (ML). One extreme is completely uninformed learning that relies only on data, also called end-to-end learning. The other extreme is hand-designed modules that incorporate prior knowledge about the learning problem, also known as inductive biases, algorithmic biases, or structural biases.

This project studies how inductive biases in the form of structured models can facilitate learning temporally extended, low-level Reinforcement Learning (RL) policies with terminating conditions, also referred to as skills and closely related to options (Richard S Sutton, Precup, et al., 1999). This hybrid architecture between model-based and model-free RL is called the Control What You Can (CWYC) framework. In contrast to the standard approach in model-based RL, which is to learn one unstructured model for all the agent-environment interactions in an end-to-end fashion (M. Deisenroth et al., 2011; Chua et al., 2018), in *CWYC* several structured models are learned that can dynamically adapt to an environment by learning from data. Similar to the innate behaviors and genetically hard-wired structures in animals, the structured models in CWYC serve multiple purposes: (1) The structured models can learn or adapt to the peculiarities of different environments from just a

few training samples compared to the massive amount of training data required to learn unstructured models. (2) Most of the planning complexity, usually done by the low-level control policies, is offloaded to the models allowing the low-level policies to be much simpler composable subroutines. (3) As with infants' sucking reflex or self-play, the structured models can be used to generate situations that create valuable training data for the low-level policies or skills, moving the exploration problem from the low-level control commands to abstract goal spaces and task spaces.

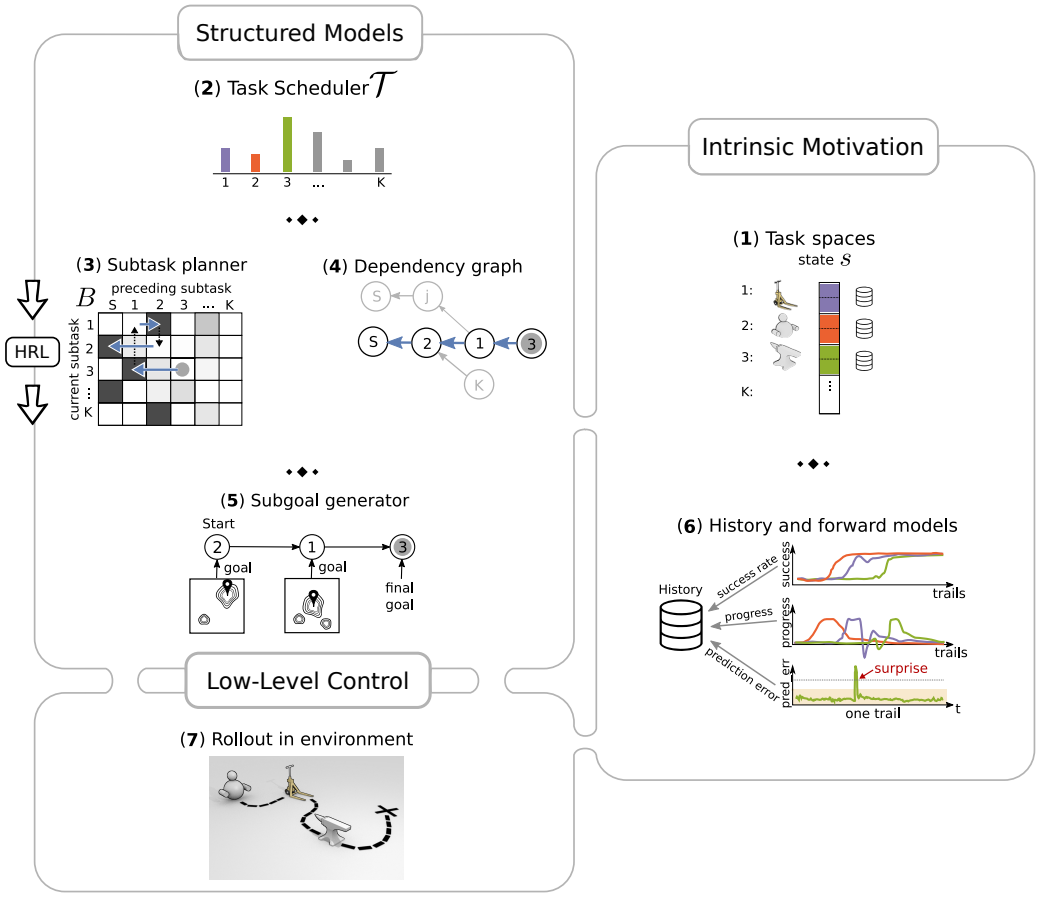


FIGURE 3.1: Overview of the CWYC architecture. CWYC consists of the following components: (1) Several task spaces defined over groups of coordinates in the observation vector. (2) A task scheduler that distributes learning efforts between tasks. (3) A sub-task planner with (4) an associated task dependency graph. (5) Sub-goal proposal networks. (6) Low-level control policies or skills and (8) an intrinsic motivation module that is derived from the history (7) that gets recorded for each skill and trial.

In CWYC, the following inductive biases are introduced to solve compositional object-manipulation tasks in complex environments (see Fig. 3.1): (1) An entity- or object-centric state representation. (2) A task scheduler that allocates time and attention to tasks in which the agent can make progress towards learning them, creating a self-guided learning curriculum similar to Achiam, Edwards, et al. (2018). (3) A subtask planner from which (4) a graph gets derived that models dependencies between compositional multi-object tasks similar to Konidaris et al. (2009). (5) A structure to learn geometric relations between task-relevant objects in the environment (Santoro et al., 2017; Zambaldi et al., 2019). This model is used to generate intermediate goals for potential intermediate subtasks similar to Florensa et al. (2018). (6) An intrinsic

motivation module that models the agent's desire to maximize learning progress. Intrinsic motivation is implemented as a combination of prediction error and learning progress as the primary force that drives self-guided learning (Forestier et al., 2020). The models are organized in a hierarchy and direct the exploration and behavior of (7) the low-level RL policies that get executed in the environment.

Section 3.2 of this chapter introduces the CWYC architecture in full detail. Section 3.3 discusses the environments in which experiments are conducted to empirically demonstrate the effectiveness of the CWYC architecture. The CWYC agent is compared against several baselines which are introduced in section Sec. 3.4. Section 3.5 presents the experimental results and Sec. 3.6 discusses various ablations to the CWYC architecture by showing their impact on the performance of the agent. This chapter closes with a discussion in Sec. 3.7.

3.2 Method

The following sections introduce the CWYC framework in full detail.

3.2.1 Preliminaries

In the following, the MDP formulation introduced in Sec. 2.1 of Ch. 2 is used. The goal of this work is to design an agent that gains control over the environment. In other words, to design an agent that alters the observations $o \in \mathcal{O} = \mathbb{R}^n$ from the environment in arbitrary ways. The observations are grounded in the environment via a perception module $f^{\text{obs}}\colon \mathcal{S} \to \mathcal{O}$ that maps MDP states s to observations $o = f^{\text{obs}}(s)$. Since the perception module grounds observations in the outside world, the agent has to manipulate the environment in order to change the observations. This work focuses on compositional object-manipulation environments in which an agent is supposed to change the location or orientation of randomly scattered objects in the environment. In particular, multi-stage goal-reaching tasks are studied that require the agent to reach a series of task-dependent (sub-)goals. As soon as the agent reaches a subgoal, the agent's focus switches from one task to a potential next task. Overall success is defined as reaching the goal of a final task.

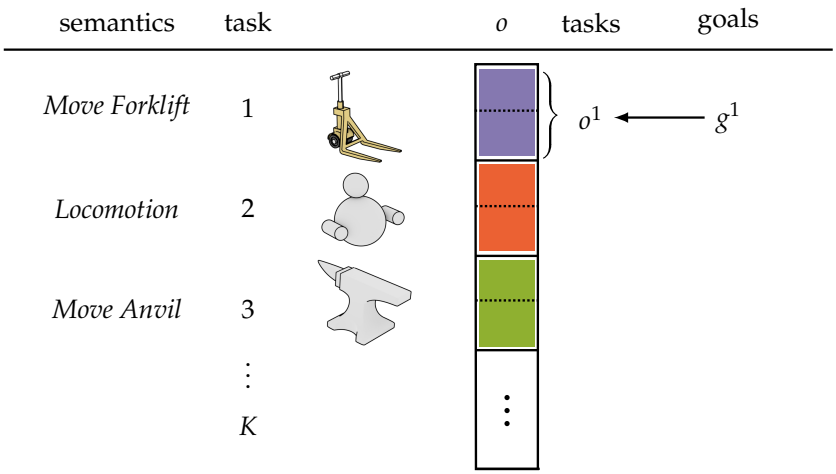


FIGURE 3.2: Schematics of the observation vector. Observations are divided into non-overlapping, simultaneously controllable tasks $o^{\mathcal{T}}$. Every task has a semantic meaning that is unknown to the agent. Independent goals $g^{\mathcal{T}}$ can be set for any of the tasks.

The observation vector is assumed to be pre-partitioned into groups of non-overlapping, simultaneously controllable components referred to as **task spaces** $\mathcal{O}^{\mathcal{T}}$. Here, the object-centric state representation is an inductive bias hard-coded in the observation. The same inductive bias can be introduced by using Graph Neural Networks (GNNs) (Battaglia et al., 2018) or similar structured models. Each task space has a corresponding **goal space** $\mathcal{G}^{\mathcal{T}}$, similar as in Andrychowicz et al. (2017), that specifies a desired target $g^{\mathcal{T}} \in \mathcal{G}^{\mathcal{T}}$ for the task-space vector $o^{\mathcal{T}} \in \mathcal{O}^{\mathcal{T}}$.
Example: *In a potential locomotion task, the agent needs to change its $(x,\ y)$-coordinates in the observation vector by changing its position. In an object manipulation task, the agent has to change the object's $(x,\ y)$-coordinates in the observation vector by moving the object from location a to location b.*

Figure 3.2 shows a graphical depiction of the observation vector and the grouping of coordinates into task spaces with their respective semantic meanings. The semantics of the task spaces is unknown to the agent at all times. All the agent knows is what groups of coordinates belong together. The perception problem, that is, constructing task spaces from high-dimensional image data or other sensor modalities is an orthogonal line of research. Readers interested in representation learning are referred to Péré et al. (2018) and Burgess et al. (2019).

Formally, the observation space factorizes in K object- or entity-centric subspaces $\mathcal{O} = \mathcal{O}^1 \times \ldots \times \mathcal{O}^K$. Each subspace $\mathcal{O}^\mathcal{T}$ has a corresponding goal space denoted by $\mathcal{G}^\mathcal{T} \subseteq \mathcal{O}^\mathcal{T}$. Since manipulation of a vector $o^\mathcal{T} \in \mathcal{O}^\mathcal{T}$ can be interpreted as solving a particular task, e.g., moving an object to a target location $g^\mathcal{T} \in \mathcal{G}^\mathcal{T}$, in the following subspaces are also referred to as task spaces, with $\mathcal{T} \in \{1, \ldots, K\}$. It is assumed that task spaces are non-overlapping and encompass only simultaneously controllable components. Given a specific goal $g^\mathcal{T}$ for a task $\mathcal{T}$, the agent's objective is to create an action sequence that brings $o^\mathcal{T}$ as close as possible to $g^\mathcal{T}$, according to some metric $d^\mathcal{T} : \mathcal{O}^\mathcal{T} \times \mathcal{G}^\mathcal{T} \to \mathbb{R}$. The Euclidean distance between task and goal state

$$d^\mathcal{T} = ||o^\mathcal{T} - g^\mathcal{T}||_2^2 \tag{3.1}$$

is used as a metric in all the experiments discussed in Sec. 3.5. This metric has two advantages: (1) The Euclidean distance is general and does not impose any particular structure on the goal spaces. Therefore, it can be easily applied to any goal space. (2) This particular distance function can be easily computed by the agent because all the relevant information is contained in the observation vector.

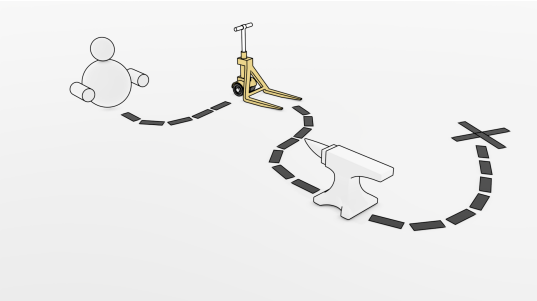


FIGURE 3.3: Sketch of a single trial. At the beginning of each trial, the agent selects a final task $\mathcal{T}$ and a self-imposed goal $g_\mathcal{T}$, e.g., move the anvil to the target location marked by the giant cross. However, to solve a particular task, it might be necessary to solve multiple subtasks, e.g., move to the forklift, pick up the anvil with the forklift, move the anvil with the help of the forklift to its target location. In that case, the agent must create an appropriate chain of subtasks and subgoals that connects the individual subtasks.

All experiments consist of two phases:
In the **intrinsic phase** or **developmental phase**, the agent does not receive any external goals or environment rewards. Instead, the agent has time to freely explore the environment and gain control over as much of the environment as possible. The developmental phase is divided into **trials**. At the beginning of each trial, the environment is reset to a random state and the agent creates a self-imposed goal $g^\mathcal{T}$ for a task $\mathcal{T}$ that maximizes learning progress (see Sec. 3.2.3), e.g., move to location $(x,\ y)$. While the agent tries to solve the task $\mathcal{T}$, it evaluates its action at time step t by computing an internal reward function given by:

$$r(t,\ \mathcal{T}) = -d^\mathcal{T}(t) = -||o^\mathcal{T}(t) - g^\mathcal{T}(t)||_2^2. \tag{3.2}$$

The developmental phase is followed by the **extrinsic phase**, in which the agent is confronted with external goals for any of the tasks $\mathcal{T}$. In this phase, the agent's objective is to solve various given tasks and reach the externally provided goals using skills learned during the developmental phase. The agent's performance is evaluated using the same metric as the agent uses to compute its internal reward.
Example: *Figure 3.3 shows a sketch of a potential trial: The final task is to move a very heavy anvil to the target location marked by a big cross. Alone, the agent cannot move the anvil. It needs a tool. In this case, a forklift is the right tool for the job. To succeed in the task, the agent has to reach the forklift, bring the forklift to the anvil, and eventually move the anvil with the help of the forklift to the target location.*

Note that time t is measured relative to the beginning of a trial. Each trial has a maximum number of $T^{\max}$ time steps.
Learning in the developmental phase is governed by several components as shown in Fig. 3.1. Their detailed interplay is as follows:
Tasks $\mathcal{T}$ (**1**) define groups of components (coordinates) in the observation vector. Each task has a semantic meaning attached to it. The semantic meanings are unknown to the agent.
A task scheduler (**2**) is used by the agent at the beginning of each trial to select a self-imposed task $\mathcal{T}$ (final task) to maximize expected learning progress.
Given a final task, the **task planner** B (**3**) computes a viable sub-task sequence from a learned **task-dependency graph** (**4**).
The **subgoal generators** $G_{\ell \to k}$ (**5**) create a time-dependent goal $g^{\ell}(t)$ for each task transition from task ℓ to task k.
Goal-conditioned control policies (**6**) $\pi^{\mathcal{T}}(o,\ g^{\mathcal{T}})$ control the agent in the environment and encapsulate the individual subroutines or skills.
Between trials, a history of different quantities like the **learning progress** and **surprising events** are recorded in a **per-task history buffer** (**7**).
An **intrinsic motivation** module (**8**) computes the rewards and target signals for the **task scheduler**, **task planner**, and **subgoal generator** based on **learning progress** and **surprise**.

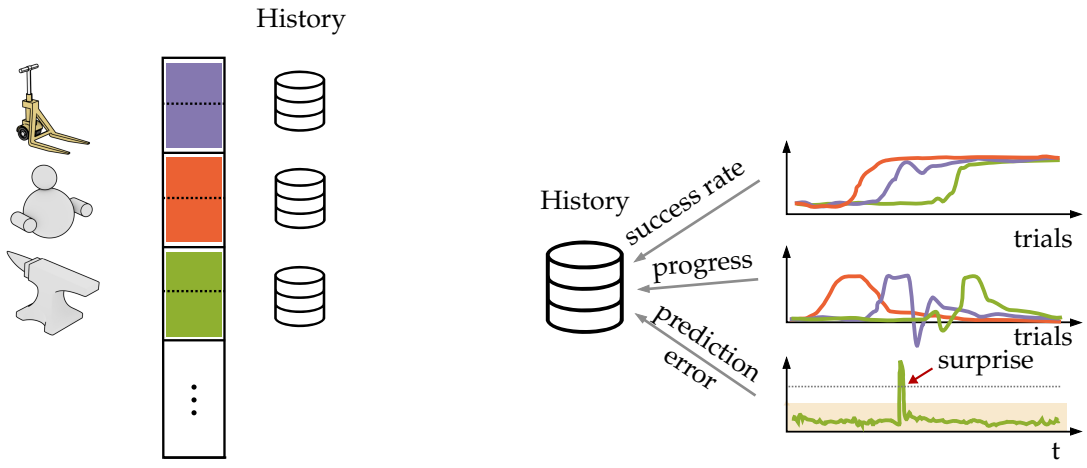
All components are trained concurrently from data collected during the developmental phase without external supervision. Prior knowledge enters only in the form of the predefined task spaces and the internal structure of the models.

3.2.2 Intrinsic Motivation

In the developmental phase, the agent's objective is to gain control over the environment, that is, to succeed in or master all potential tasks in the environment. In trial i, the agent declares success in a self-imposed task $\mathcal{T}^\star$, if $o^{\mathcal{T}^\star}$ is close, up to a precision $\delta^{\mathcal{T}^\star} \geq 0$, to the goal state $g^{\mathcal{T}^\star}$ w.r.t. $d^{\mathcal{T}^\star}$:

$$\text{succ}^{\mathcal{T}^\star}(i) = \begin{cases} 1 & \text{if } d^{\mathcal{T}^\star}(t) \leq \delta^{\mathcal{T}^\star} \text{ for } t \leq T^{\max} \text{ and } \mathcal{T}^\star = \mathcal{T}^{\text{final}}, \\ 0 & \text{otherwise.} \end{cases} \tag{3.3}$$

It is important to note that the agent only succeeds in trial i if it attempts to solve the self-imposed task $\mathcal{T}^\star$ as the final task in a potential sequence of subtasks.
Example: *Coming back to the example depicted in Fig. 3.3 in which the agent is supposed to move the anvil to the target location with the cross. The agent only succeeds in the task if it executes the task sequence: reach forklift, transport forklift to anvil, move anvil to target location. Suppose the agent instead executes the task sequence: get forklift, move forklift to target location. In the process, the agent might pick up the anvil just by pure chance and*



(A) Task spaces. (B) Per-task history buffer.

FIGURE 3.4: Object- or task-based intrinsic motivation: (A) For each task space $\mathcal{O}^{\mathcal{T}}$ several per-trial quantities are stored in a per-task history buffer. (B) The per-task history buffers store quantities like the success rate, the learning progress, and the prediction error of a forward model.

bring it to the cross. In that case, the agent will not succeed in the task because it was not even attempting to move the anvil.

As some goals can be easier to reach than others, the agent does not look at a single trial to measure success, but uses a recent history of Z trials to compute a **success rate** or the **controllability** of the corresponding task:

$$\mathrm{sr}^{\mathcal{T}}(i) = {}^{1}/_{Z} \sum_{i \in S^{\mathcal{T}}} \mathrm{succ}^{\mathcal{T}}(i), \tag{3.4}$$

with $S^{\mathcal{T}}$ being the set of the last Z trial indices in which the agent attempted to solve task $\mathcal{T}$.

To effectively distribute its learning effort between the different tasks, the agent is motivated to maximize its **instantaneous learning progress** on a trial-by-trial basis. The instantaneous learning progress is defined as the derivative of the success rate with respect to task trials:

$$\rho^{\mathcal{T}}(i) = \frac{\Delta \mathrm{sr}^{\mathcal{T}}(i)}{\Delta i}. \tag{3.5}$$

The reason for maximizing the learning progress instead of the success rate directly is the following: Once the agent masters a task, the success rate is a constant function of 1. Therefore, the agent would continue to concentrate its learning efforts on tasks it has mastered already, although no further progress can be made in those tasks. With the learning progress, however, the agent only concentrates on tasks that can improve in terms of their success rate and stops focusing on tasks as soon as no further progress can be made either because the agent has mastered the task or it cannot be learned at all.

Since learning progress can be sparse, especially at the beginning of learning a new task, the agent is also motivated to recreate surprising events it encountered in past trials. The agent deems an event surprising if it was not predicted by an internal forward model $f^{\mathrm{fw}}\colon \mathcal{O} \times \mathcal{A} \to \mathcal{O}$ of the world (see Sec. 2.2.3 in Ch. 2 for more details) that the agent learns over time. The **prediction error** of the forward

model is defined as:

$$e(t) = \|(f^{\text{fw}}(o(t),\ a(t)) - o(t+1)\|_2^2. \tag{3.6}$$

Given the definition of the prediction error, a **surprising event** is defined as (see also Gumbsch, Otte, et al., 2017):

$$\text{surprise}^{\mathcal{T}}(t) = \begin{cases} 1 & \text{if } |\frac{\Delta e^{\mathcal{T}}(t)}{\Delta t}| > \mu^{\mathcal{T}} + \theta \cdot \sigma^{\mathcal{T}}, \\ 0 & \text{otherwise,} \end{cases} \tag{3.7}$$

where $e^{\mathcal{T}}$ is the prediction error in task space $\mathcal{O}^{\mathcal{T}}$ and $\mu^{\mathcal{T}}$ and $\sigma^{\mathcal{T}}$ are the first and second moments of the distribution of the time derivative of the prediction error. The time derivative of the prediction error is assumed to be Gaussian distributed :

$$(e^{\mathcal{T}}(t) - e^{\mathcal{T}}(t-1))/\Delta t = \frac{\Delta e^{\mathcal{T}}(t)}{\Delta t} \sim \mathcal{N}(\mu^{\mathcal{T}}, (\sigma^{\mathcal{T}})^2). \tag{3.8}$$

The parameter θ in Eq. 3.7 is a threshold that defines the confidence interval outside which the agent labels an event as surprising. The following example can explain why surprising events can help to guide the agent's exploration but should be used with care:

Example: *Assuming the agent only knows how to move itself, it will just move around, not knowing how to manipulate other parts of the environment. That means, the agent can neither move the forklift nor the anvil yet. However, whenever the agent moves to the forklift just by chance, the forklift suddenly starts to move and creates a high prediction error or surprise signal. Thus, it is likely that this particular situation is a good starting point for actually solving the "move forklift" task and to continue to explore from this situation. Now an autonomous drone enters the scene, flying out of reach of the agent. The independent movement of the drone will constantly create surprising events for the agent because it can not predict the drone's movement, disregarding how much the agent observes the drone. Thus, the agent's initial urge to control the drone should fade over time since the object is not controllable by the agent.*

With this example in mind, the prediction error should not be the primary driver of the agent's exploration but rather spark initial interest in pursuing a task. It is worth noting that the agent does not have to pursue a particular task ℓ to encounter unanticipated events in that task. The agent can follow any other task k as long as the prediction error in the task ℓ is high enough. In that way, interest in any task can be sparked at any given point in time, even though this task was only of meager interest for the agent so far.

3.2.3 (Self-Imposed) Task Scheduler

During the learning/developmental phase, the agent can decide which task it wants to pursue in each trial. Intuitively, it should be beneficial for the agent to concentrate on tasks in which it can make the most learning progress in or that had surprising events in the past.

The task scheduler (see Fig. 3.1(**2**) and Fig. 3.5) keeps track of the agent's interest in the different tasks and is implemented as a multi-armed bandit. The per-task reward for the multi-armed bandit is defined as:

$$r^{\mathcal{T}}(i) = |\rho^{\mathcal{T}}(i)| + \beta \cdot \text{surprise}^{\mathcal{T}}(i), \tag{3.9}$$

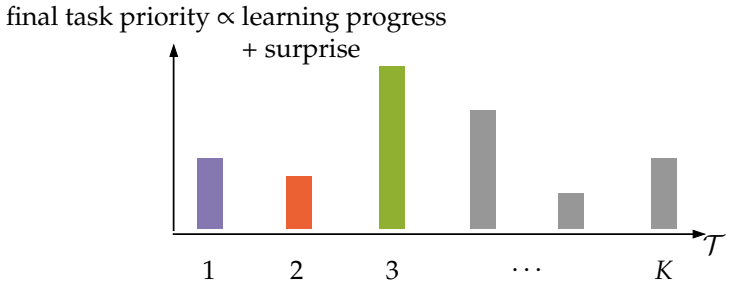


FIGURE 3.5: The agent distributes its learning resources between the different tasks. The priority of pursuing a task is computed as a combination of learning progress and surprise. At the beginning of each trial, the agent selects a final task $\mathcal{T}^\star$. This selection progress is implemented as a multi-armed bandit that tries to maximize the reward specified in Eq. 3.9.

with the trial-based surprise:

$$\text{surprise}^{\mathcal{T}}(i) = \mathbb{1}_{\{\text{surprise}^{\mathcal{T}}(k,\,t)),\,\forall t,\,k<i\}}(i), \tag{3.10}$$

and $\mathbb{1}$ being the indicator function. Equation 3.10 is 1 if $\text{surprise}^{\mathcal{T}}(t) = 1$ for any time step t in any trial $k \leq i$. While the primary quantity to maximize is the learning progress, the surprise signal is added to the reward, with $\beta \ll 1$, to spark an initial interest in the agent to attempt a task that the agent has not explored before but that showed some unexplainable behavior in the past.

The absolute value of the learning progress $|\rho_{\mathcal{T}}(i)|$ is used because the agent should prioritize a task more when it can improve in a task and if performance degrades over time (Baranes et al., 2013b). Initially, the surprise term dominates the reward as $|\rho_{\mathcal{T}}(i)| \to 0$. As soon as actual progress can be made, the learning progress takes the leading role in the reward signal. The reward signal is non-stationary because the learning progress in each task changes over time. To track the non-stationary reward, the task scheduler updates an internal estimate of the reward according to:

$$Q^{\mathcal{T}}(i) = Q^{\mathcal{T}}(i-1) + \alpha(r^{\mathcal{T}}(i) - Q^{\mathcal{T}}(i-1)), \tag{3.11}$$

with the learning rate $\alpha > 0$. Equation 3.11 is a running exponential average of the per-task reward.

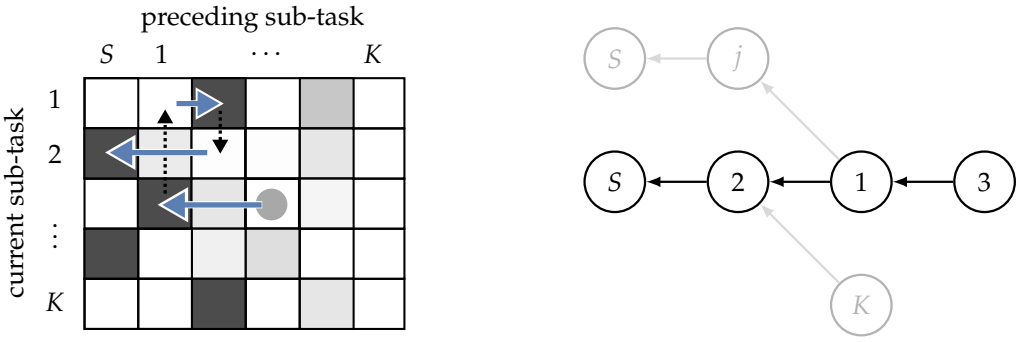
To promote exploration, a stochastic policy with $\mathcal{T}^\star \sim p(\mathcal{T}^\star = \mathcal{T}) = Q^{\mathcal{T}} / \sum_j Q^j$ is used to select tasks.

3.2.4 Task-Planning Architecture

More difficult self-imposed tasks might require one or several subtasks to be performed in a particular order to be solved successfully. A self-imposed task in one trial can be a subtask of an other task in a different trial.

Example: *For instance, if the agent wants to move the anvil, it has to move to the forklift first, then move with the forklift to the anvil, and finally transport the anvil with the forklift to its final position. But just moving to a different location can also be a valid self-imposed task.*

A task planner determines the sequence of subtasks that need to be solved in order to solve a task $\mathcal{T}$. To do so, the task planner keeps track of the time $T_{\ell \to k}$ required to solve a particular task k if it was preceded by task ℓ.



(A) Transition matrix B of the task planner. (B) Task-dependency graph.

FIGURE 3.6: The task-planning module. (A) The transition matrix B of the task planner keeps track of the time $(B)_{k,\ell} \propto T_{\ell \to k}$ required to solve task k if it was preceded by task ℓ. (B) By using backtracking, a task graph is derived from the task planner. The task graph starts from the self-imposed task $\mathcal{T}^\star$ and computes backward all possible sequences of tasks that need to be solved in order to solve $\mathcal{T}^\star$.

Example: *For instance, if the agent attempts to relocate the anvil right after it reaches it, the time to solve the "move anvil" task will be T^{max}. This is because the anvil is too heavy for the agent to be transported on its own. However, if the agent performs the "move forklift" task right before it attempts to move the anvil, the time to solve the "move anvil" task will be smaller than T^{max} because the forklift enables the agent to carry the heavy anvil. Hence, the "move forklift" task is a prerequisite for the "move anvil" task.*

As before, surprising events are used as an additional proxy signal for potential future success.

Example: *If the forklift enables the agent to move the anvil, a high prediction error will occur the first time the agent collides with the anvil while controlling the forklift.*

The values of each task transition is captured by the entry $B_{k,\ell}$ of a transition matrix B (see Fig. 3.6A), where $k \in [1, \ldots, K]$ and $\ell \in [S, 1, \ldots, K]$ enumerate the current and preceding subtasks, respectively, and S represents the "start" of a potential subtask sequence:

$$B_{k,\ell}(i) = \frac{Q_{k,\ell}(i)}{\sum_m Q_{k,m}(i)}, \tag{3.12}$$

with:

$$Q_{k,\ell}(i) = \left\langle 1 - \frac{T_{\ell \to k}(i)}{T^{\max}(i)} + \gamma \cdot \text{surprise}^{\mathcal{T}}(i) \right\rangle. \tag{3.13}$$

In Eq. 3.13, $\langle \cdot \rangle$ denotes a running average and $T_{\ell \to k}$ is the runtime for solving task k if it was proceeded by task ℓ. Without success, $T_{\ell \to k}$ equals $T^{\max}$. Similarly to Eq. 3.9, this quantity is initially dominated by the surprise signals, with $\gamma \ll 1$.

The matrix B can also be cast as an adjacency matrix of an acyclic weighted task graph (see Fig. 3.1(4) and Fig. 3.6B). The task graph contains all potential sequences of subtasks starting in S and ending in $\mathcal{T}^\star$, from which the sequence with the minimum total execution time is selected and executed by the agent. An ϵ-greedy policy is used to encourage the exploration of different subtask sequences.

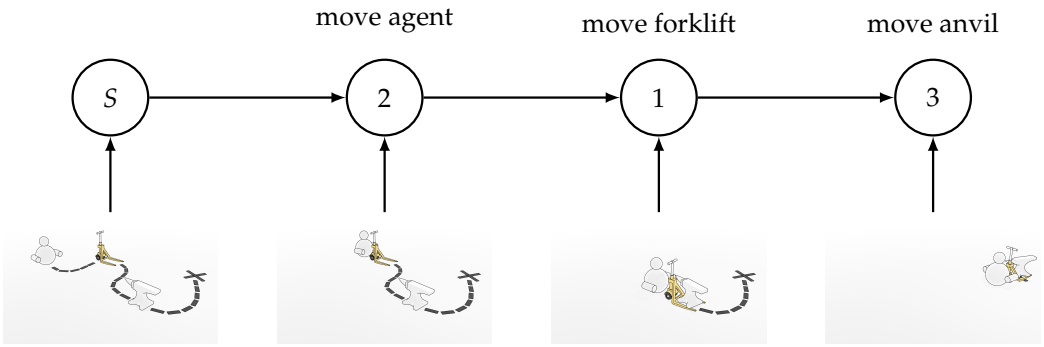


FIGURE 3.7: Example of a potential subtask sequence that solves the "move anvil" task. Once the agent figures out a viable subtask sequence, it also needs to come up with a series of subgoals that connect the subtasks in a meaningful way. The final goal (here shown as a cross) of the last task is sampled independently at the beginning of the trial.

3.2.5 Subgoal Sampling

Each (sub)-task is a goal-reaching problem.

Example: *For instance, in the running example depicted in Fig. 3.3, it is not enough to move the forklift to any position in the environment in order to transport the anvil. Instead, the forklift has to be moved precisely to the position of the anvil to do so, see Fig. 3.7.*

To this end, a goal proposal network is trained for each task transition $G_{\ell \to k}$ in the form of an attention network that can identify pairwise relations among similar observations. The model $G_{\ell \to k} \colon \mathcal{O} \to [0,\ 1]$ associates a value/attention to each coordinate in the observation vector o:

$$G_{\ell \to k}(o) = e^{-\gamma \sum_{a=1}^{n} \sum_{b=a+1}^{n} \| w^1_{ab} o_a + w^2_{ab} o_b + w^3_{ab} \|^2}, \tag{3.14}$$

where $w^1,\ w^2,\ w^3 \in \mathbb{R}^n \times \mathbb{R}^n$, and $\gamma > 0$ are trainable parameters and w^1_{ab} is a single component of the matrix w^1.

To get an intuition about the parametrization of the goal proposal network, consider a particular pair of coordinates $(a,\ b)$ in the observation vector.

Example: *For instance, a could be the agent's x coordinate and b could be the forklift's x coordinate. With $w^1_{a,\,b} = -w^2_{a,\,b} \neq 0$, the model can express that both coordinates have to have a distance of zero or coincide in the proposed goal. The network can also model offsets or global reference points with w^3_{ab}.*

See Fig. 3.8 for the schematics of the network architecture.

The attention maps w^1, w^2 and w^3 are learned via regression:

$$\mathcal{L}_{\ell \to k} = \min_{w^1_{ab}, w^2_{ab}, w^3_{ab}, \gamma} \mathbb{E}_{o \sim \mathcal{D}} \left\| G_{\ell \to k}(o) - r_{\ell \to k}(o) \right\|^2, \tag{3.15}$$

with the target being:

$$r_{\ell \to k}(o_t) = \min(1,\ \text{succ}^k \cdot \Gamma_{\ell \to k}(o_t) + \text{surprise}^k(t)). \tag{3.16}$$

In Eq. 3.16, $\Gamma_{\ell \to k}$ is defined as:

$$\Gamma_{\ell \to k}(o_t) = \begin{cases} 1 & \text{if the agent switches from task } \ell \text{ to task } k \text{ in } o_t, \\ 0 & \text{otherwise.} \end{cases} \tag{3.17}$$

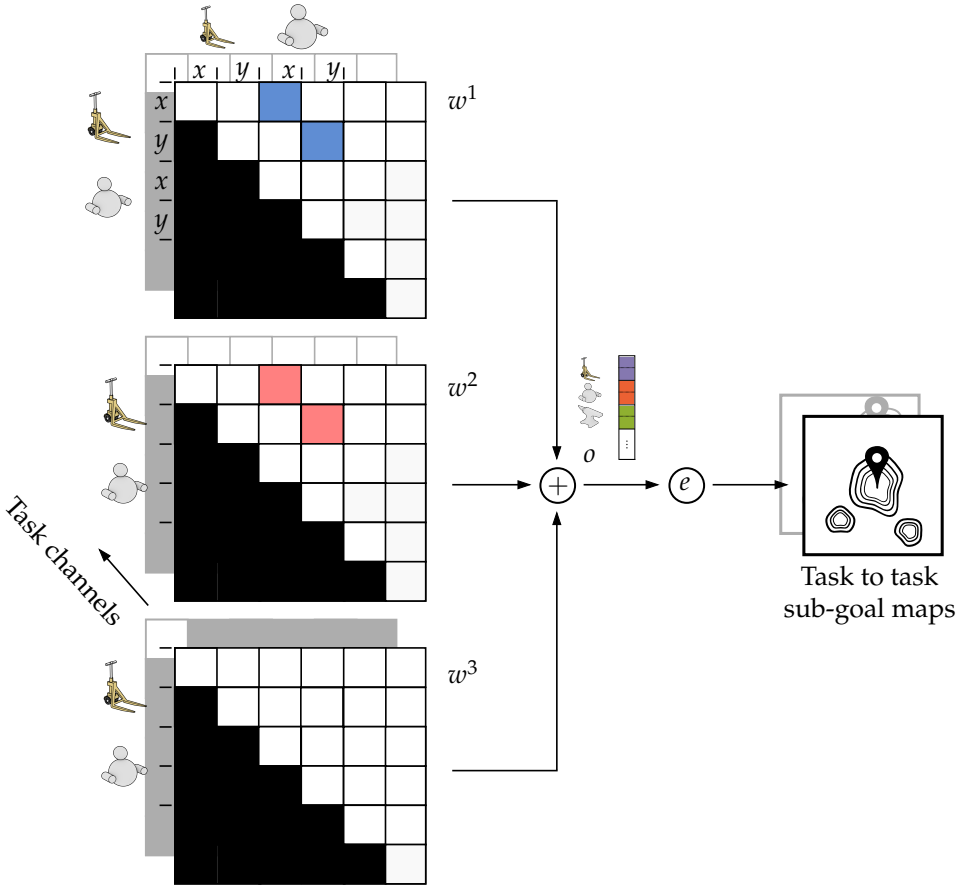


FIGURE 3.8: Network architecture of the goal proposal network. For each task transition, the network learns three attention maps w^1, w^2, and w^3. The w^1 and w^2 model pairwise relations between coordinates in the observation vector. The w^3 models offsets and global reference points. Here, the x- and y coordinates of the agent and the forklift have to coincide in the proposed goal. Only the upper triangular matrix of the attention maps is learned because the matrices are symmetrical.

The intuition behind the target signal (Eq. 3.16) is as follows: An observation counts as a positive training sample for $G_{\ell \to k}$ if the agent transitioned from task ℓ to k, hence $\Gamma_{\ell \to k} = 1$, and the agent succeeds in the following task k. At the beginning of the training, surprise is another source of positive training samples. All other observations are labeled as negative samples during training.

Example: *As in the previously presented example, let* $(a,\ b)$ *be the agent's and forklift's* x *coordinates. The first couple of times the agent moves to the position of the forklift it will cause a surprise signal in the task space of the forklift by changing the forklift's position. The goal sampling network learns from these positive training samples that something interesting happens with the forklift whenever the agent is close to it. Consequently, it will produce goals that lead to such situations. Once the agent succeeds in the "move forklift" task after moving to the forklift location, the goal proposal network will further refine its goal proposals through the success-dependent term in Eq. 3.16.*

The goal proposal network can learn relationships after a few positive examples (in the order of 10), possibly due to its restricted model class. The goal proposal network can also be considered a relational network (Santoro et al., 2017), albeit it is easier to train.

Sampling a goal from the network is done by computing the maximum of the

attention over the observation vector. For each subtask, the goal is selected with the maximal value in the attention map. However, the coordinates I^+ in the observation vector that belong to a task $\mathcal{T}^+$ that has still to be solved in the task chain are fixed, because they can likely not be controlled yet:

$$o^* = \arg\max_{o'} \quad G_{\ell \to k}(o') \qquad \text{subject to } o'_k = o_k,\ \forall k \in I^+. \tag{3.18}$$

The goal for subtask ℓ is then defined as $g^\ell(o) = (o^*)^\ell$. This is a convex program, and its solution can be computed analytically.

In the non-stationary environments discussed in this work, the goal proposal networks are a critical component of the *CWYC* framework that aim to learn relations between entities in the world. The agent sorts observations of the environment into interesting, uninteresting, and undetermined observations. Interesting observations are those in which an unanticipated event (high prediction error) occurs or which lead to success in a subsequent task. All other observations are labeled as uninteresting for a particular task. There is a third class of undetermined observations. Undetermined observations contain similar situations to observations labeled as interesting but do not spark high interest.

Example: *For instance, running into, hence suddenly moving, the forklift might spark interest in the "move forklift" task because of a sudden jump in the prediction error. Therefore, the agent tries to recreate situations in which the agent and the forklift are close. However, after the agent picks up the forklift, the movement of the forklift is entirely determined by the agent's actions resulting in low prediction errors and no surprise. Hence, none of the following observations are labeled as interesting. However, the agent and the forklift are still very close, resulting in conflicting training data. Consequently, this data is considered undetermined by the agent.*

Conclusively, we discard all undetermined transitions within a trial that come after an observation with a positive label.

After removing all data that might prevent the goal proposal networks from learning the proper relations, positive events remain rare compared to the massive body of uninteresting data. Hence, the training data is balanced in each batch during training.

3.2.6 Low-level Control

For each task space $\mathcal{O}^\mathcal{T}$, the *CWYC* agent learns a separate goal-reaching policy $\pi^\mathcal{T}(o_t,\ g^\mathcal{T}(t))$. Policies are trained with the actor-critic framework for continuous control, see Sec. 2.2.1 of Ch. 2 for more details. Specifically, policies are trained with either *Soft Actor Critic (SAC)* (Haarnoja et al., 2018) or *Deep Deterministic Policy Gradient (DDPG)*+*Hindsight Experience Replay (HER)* (Andrychowicz et al., 2017). The goal-reaching policies are temporally extended. They do not have to run until the end of the trial. Instead, each policy has a terminal condition allowing it to terminate early at $T^\mathcal{T} \leq T^{\max}$. A policy $\pi^\mathcal{T}$ terminates if the task-space vector $o^\mathcal{T}$ is close, up to a threshold $\delta^\mathcal{T}$, to the desired goal state $g^\mathcal{T}$ according to:

$$T^\mathcal{T} = \begin{cases} t & \text{if } \|o^\mathcal{T}(t) - g^\mathcal{T}(t)\|_2^2 \leq \delta^\mathcal{T} \text{ for } t \leq T^{\max}, \\ T^{\max} & \text{otherwise..} \end{cases} \tag{3.19}$$

3.3 Environments

The capabilities of the CWYC architecture are tested in two environments. The WAREHOUSE environment is specifically designed to highlight the difficulties in learning compositional multi-stage tasks in environments with rare agent-object and object-object interactions. This environment is based on the running example introduced in the methods section. WAREHOUSE is used to study the different components of the CWYC architecture more thoroughly. The FETCH PICK&PLACE TOOLUSE environment is a robotic object manipulation environment. The following paragraphs present the two environments in more detail.

WAREHOUSE The WAREHOUSE environment is depicted in Fig. 3.9. The environment consists of the following elements:

Robot $(\mathcal{T} = 0)$ The agent's actions directly control a robot in the environment.

Forklift $(\mathcal{T} = 1)$ The forklift can be controlled by the agent as soon as the robot gets close to the forklift location.

Anvil $(\mathcal{T} = 2)$ The anvil is too heavy to be moved by the robot directly. The agent can only move the anvil if it controls the forklift with the robot.

Cone $(\mathcal{T} = 3)$ The cone is an unreliable object, as its behavior changes randomly between trials. In some of the trials, it can be moved by the robot. In other trials it is bolted to the floor and therefore completely immovable.

Drone $(\mathcal{T} = 4)$ An autonomous drone flies around randomly in the environment. It is out of reach of the robot; thus, it cannot be controlled by the agent.

In WAREHOUSE the 16-dimensional continuous Markov Decision Process (MDP) state s and the observation vector o are connected by an identity mapping:

$$o = s = f^{\text{obs}}(s) \in \mathbb{R}^{16}. \tag{3.20}$$

The observation vector is given by:

$$o = [o^0,\ o^1,\ \ldots,\ o^4,\ \dot{x},\ \dot{y},\ p^1,\ \ldots,\ p^4] \tag{3.21}$$

with $[\ldots]$ being the concatenation operator. The $o^{\mathcal{T}} = (x^{\mathcal{T}},\ y^{\mathcal{T}})$ are the different task spaces, with $x^{\mathcal{T}}$ and $y^{\mathcal{T}}$ being the positional coordinates of the respective entity. The task spaces have the following semantic meaning: $(\mathcal{T} = 0)$ *Locomotion*, $(\mathcal{T} = 1)$ *Move Forklift*, $(\mathcal{T} = 2)$ *Move Anvil*, $(\mathcal{T} = 3)$ *Move Cone*, $(\mathcal{T} = 4)$ *Control Drone*. The $(\dot{x},\ \dot{y})$ are the robot's velocities. The indicator variables p^1 to p^4 indicate whether the agent is in possession of the i-th object $(p^i = 1)$ or not $(p^i = 0)$.

The agent controls the robot and any controllable object in possession of the robot by applying forces in the x- and y-axis to the robot:

$$a = (F_x, F_y) \in \mathbb{R}^2. \tag{3.22}$$

The robot's dynamics is modeled as a 2-dimensional double integrator, which is subject to the laws of motion with the application of friction from the environment, rendering the control of the robot non-trivial. The other objects do not have their own dynamics, except the drone. They are either not moving at all, or the robot's

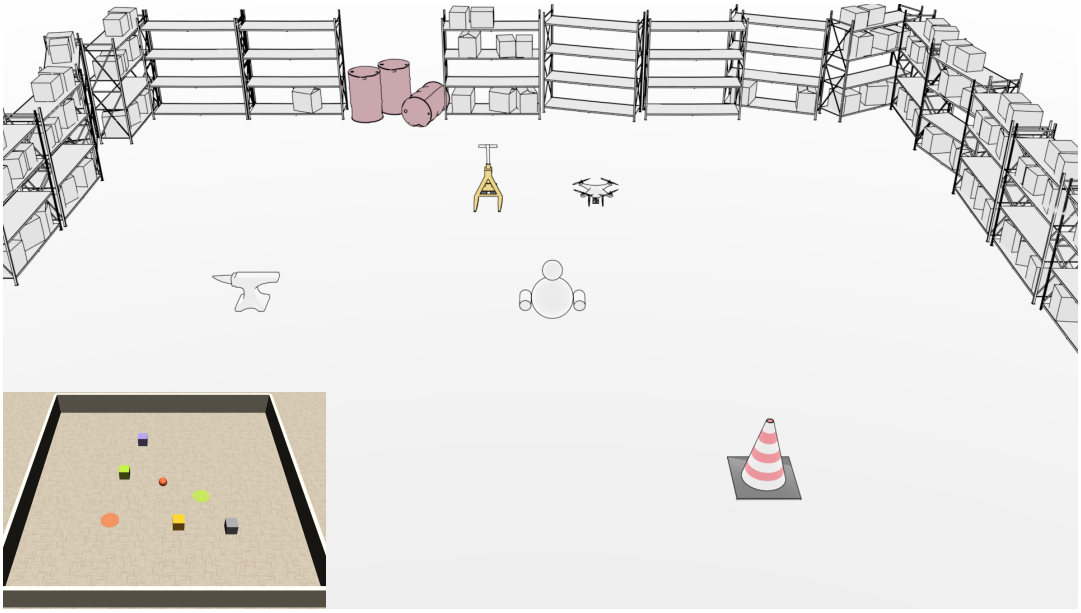


FIGURE 3.9: WAREHOUSE consists of a robot that the agent directly controls. The robot can pick up and move a forklift. A heavy anvil cannot be controlled by the agent directly but only with the help of the forklift. An autonomous drone moves randomly in the environment and cannot be controlled by the agent. A randomly placed cone is bolted to the floor in some of the trials, while in others not. Hence, the robot can move it only in some of the trials. In the bottom left corner is a screenshot of the simulated environment. A sphere represents the robot, while colored blocks represent the other objects.

movement entirely governs their movement. A random policy controls the movement of the drone. Some objects are more difficult to move than others and might depend on other entities in the environment, e.g., the anvil depends on the forklift.

In the developmental phase, the agent computes a reward for each task $\mathcal{T}$ according to:

$$r^{\mathcal{T}}(t) = -d^{\mathcal{T}}(t) = -\|o^{\mathcal{T}}(t) - g^{\mathcal{T}}(t)\|_2^2. \tag{3.23}$$

The reward is partially sparse in the sense that it does not include the distance between robot and object. If the robot tries to relocate an object and is not moving it, the reward signal is constant, making the exploration problem very challenging.

The bottom left overlay in Fig. 3.9 shows the environment in the simulation. The environment is simulated in the MuJoCo physics simulator (Todorov et al., 2012). In the simulation, the robot is visualized as a sphere. The other objects are visualized as colored cubes.

At the beginning of each trial, the environment gets reset. The robot always starts in the middle of the warehouse while all the other objects are spawned at a random location. Thus, it is not sufficient for the agent to learn one static policy. Each trial has a total length of $T_{\text{max}} = 1600$ steps.

The warehouse is relatively large, and objects can be placed quite far apart. Therefore, random interactions between the agent and the objects are infrequent. One major challenge in this environment is the exploration problem, and the agent has to learn from these rare events efficiently.

FETCH PICK&PLACE TOOLUSE In the second environment, the agent controls a 7 Degree of Freedom (DoF) fetch robot. The environment is shown in Fig. 3.10. Besides the robot, the environment consists of a hook-shaped tool and a cube spawned on a table in front of but out of reach of the robotic arm. With its 4 dimensional continuous action space, the agent controls the x-, y-, and z-movement of the end-effector in Cartesian space and the opening and closing of a gripper attached to the end-effector.

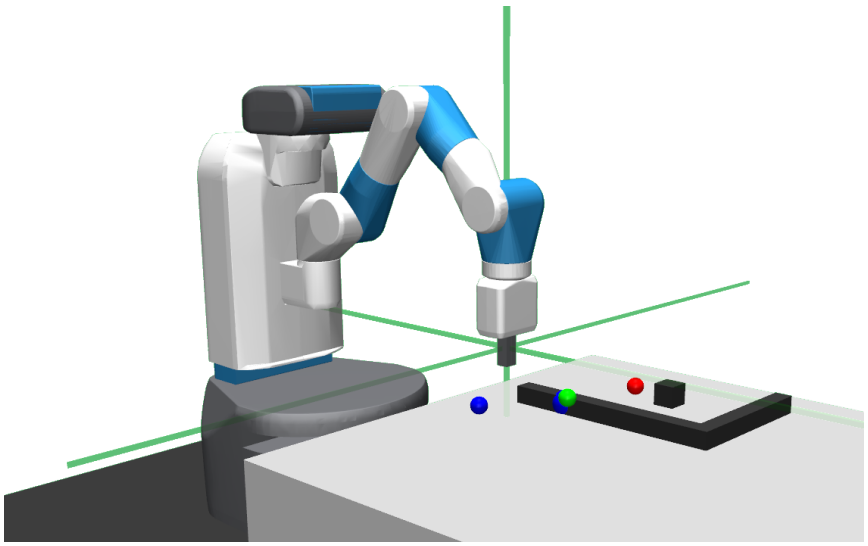


FIGURE 3.10: In the robotic manipulation environment, the agent controls a fetch robot. The robot can use a hook-shaped tool to move a cube to a target location that is otherwise unreachable for the robotic arm.

In this environment, the agent does not observe the MDP state directly, including all the joint angles of the fetch robot. Instead, it receives an observation that contains the Cartesian coordinates of the end-effector and the relative positions between the end-effector and the cube and the end-effector and the tool, among other information. The environment is derived from the FETCH PICK&PLACE environment in the OPENAI GYM (Brockman et al., 2016) collection of RL tasks. Two modifications are applied to the original environment: (1) A tool is added to the environment together with the tool's positional, rotational, and velocity information in the observation vector. (2) The table is extended to allow the cube to be placed out of reach of the robotic arm.

The task spaces are the 3-dimensional positions of the end-effector, the tool (measured at the location of the green sphere in Fig. 3.10), and the cube. The corresponding semantic meanings/tasks are *Move End-Effector*, *Use Hook*, and *Move Cube*.

In the developmental phase, the agent computes its intrinsic reward according to Eq. 3.23. Each trial has a total length of $T_{\max} = 150$ steps.

3.4 Baselines

The CWYC framework is compared against several baselines: First, a version of CWYC in which some of the components are replaced by oracles. Second, a vanilla version of the SAC algorithm with a shaped reward function to guide the exploration. CWYC is also compared against a state-of-the-art hierarchical RL algorithm and an Intrinsic Motivation (IM) baseline. In FETCH PICK&PLACE TOOLUSE, CWYC is compared against vanilla *DDPG+HER* as this algorithm is specifically tuned to the FETCH PICK&PLACE environment. The following sections explain the baselines in more detail.

CWYC with Oracles To assess the upper bound on the performance of CWYC in WAREHOUSE, a baseline is crafted in which all of the learned high-level components including the **task planner**, the **task graph**, and the **subgoal generators** are replaced by oracles.

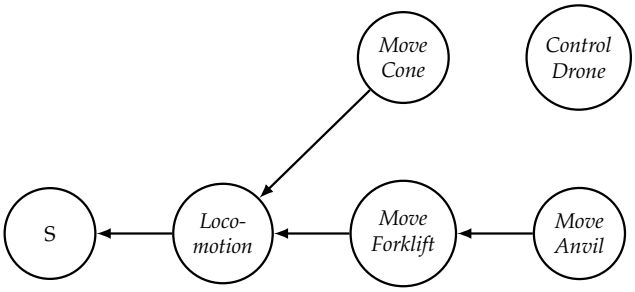


FIGURE 3.11: Oracle task graph. *Locomotion* does not have any prerequisites. *Locomotion* has to be solved first to solve *Move Forklift* and *Move Cone*. *Move Forklift* is a prerequisite for *Move Anvil*. *Control Drone* cannot be solved at all.

Figure 3.11 shows the oracle task graph. *Locomotion* can be solved immediately because the agent's actions directly control the robot. To solve *Move Forklift*, *Locomotion* has to be solved first. The same goes for *Move Cone*. Before *Move Anvil* can be solved, *Move Forklift* has to be solved first. But to solve *Move Forklift*, *Locomotion* has to be solved already. *Control Drone* cannot be solved at all because the drone moves autonomously and is out of reach for the robot.

Figure 3.12 and Fig. 3.13 show the oracle attention weight maps w^1 (left), w^2 (middle) and w^3 (right) for the *Locomotion* to *Move Forklift*, *Move Forklift* to *Move Anvil*, and *Locomotion* to *Move Cone* task transitions, respectively. Again, the $(x^0,\ y^0)$-coordinates correspond to the agent's position. The $(x^k,\ y^k)$-coordinates, with $k = 1, \ldots, 4$, correspond to the objects' positions, i.e., $k = 1$ is the location of the forklift, $k = 2$ is the location of the anvil, $k = 3$ is the location of the cone, and $k = 4$ is the location of the drone.

According to Fig. 3.12, to succeed in *Move Forklift* the positional coordinates of the agent $(x^0,\ y^0)$ and the forklift $(x^1,\ y^1)$ have to be the same after transitioning from *Locomotion* to *Move Forklift*. In other words, the goal proposed by $G_{0\to1}$ is given by $g^0(t) = o^1(t)$. In the case of the *Move Forklift* to *Move Anvil* transition, the positions of the agent $(x^0,\ y^0)$, the forklift $(x^1,\ y^1)$, and the anvil $(x^2,\ y^2)$ have to be the same. Since *Move Anvil* requires *Move Forklift* to be solved and *Move Forklift* needs *Locomotion* to be solved it is already guaranteed that $o^0 = o^1$.

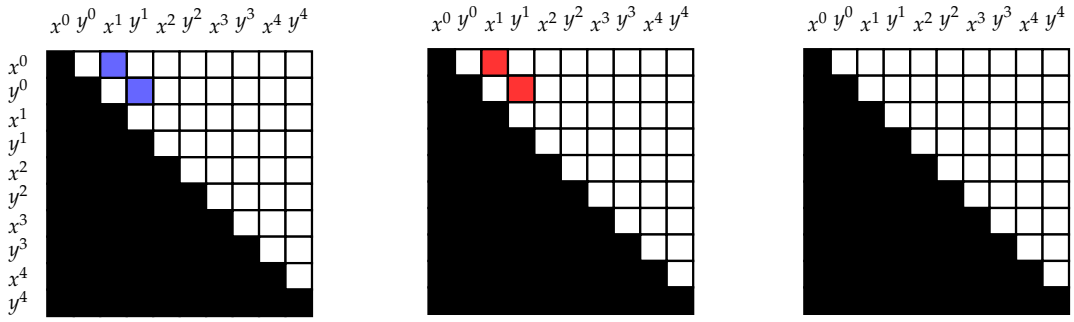
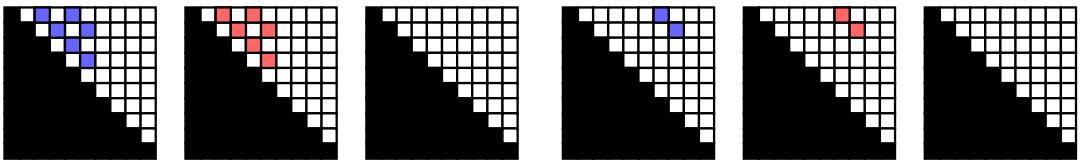


FIGURE 3.12: Oracle attention weight maps w^1 (left), w^2 (middle), and w^3 for the transition between *Locomotion* and *Move Forklift*. At the end of *Locomotion*, the task space $o^0 = (x^0,\ y^0)$ has to be equal to the position of the forklift $g^0 = o^1 = (x^1,\ y^1)$. As a reminder, $(x^k,\ y^k)$ are the positional coordinates of the different entities in the environment, with $k = 0$ being the robot, $k = 1$ being the forklift, $k = 2$ being the anvil, $k = 3$ being the cone, and $k = 4$ being the drone.



(A) *Move Forklift* to *Move Anvil* task transition. (B) *Locomotion* to *Move Cone* task transition.

FIGURE 3.13: Oracle attention weight maps for (A) the *Move Forklift* to *Move Anvil* and (B) the *Locomotion* to *Move Cone* task transitions. Same notation as in Fig. 3.12.

Soft Actor-Critic with Oracle Reward SAC does not make use of hierarchies or IM. As in CWYC, SAC learns one policy per task space. However, instead of using a sequence of policies to solve a single task, SAC has to use a single policy to solve the entire task in one go. That makes it harder for SAC to learn the more challenging tasks such as *Move Anvil* since one policy has to learn a complex trajectory in one shot. For instance, the *Move Anvil* policy has to first pick up the forklift, move with the forklift to the anvil, and eventually transport the anvil to its target location. To mitigate the issue that SAC has to learn much more complex policies and that it has to solve the exploration problem for each task independently, a much more dense/informative reward signal is used for the individual tasks as listed below.

Locomotion The Euclidean distance between the agent's location and the goal location is used.

Move Forklift The Euclidean distance between the forklift's location and the goal location plus the euclidean distance between the agent's location and the forklift's location is used.

Move Anvil The Euclidean distance between the anvil's location and the goal location plus the euclidean distance between the agent's location and the forklift's location as well as the distance between the forklift's location and the location of the anvil is used.

Move Cone The Euclidean distance between the cone's location and the goal location plus the euclidean distance between the agent's location and the cone's location is used.

Control Drone The Euclidean distance between the drone's location and the goal location plus the euclidean distance between the agent's location and the drone's location is used.

In the following, SAC^+ is used to refer to the *SAC* baseline with privileged knowledge.

Hierarchical RL Baseline with Oracle Rewards HIRO (Nachum et al., 2018) is an algorithm designed to solve HRL problems. The *HIRO* agent consists of a goal-conditioned low-level policy μ^{lo} that interacts directly with the environment and a goal-conditioned high-level policy μ^{hi} that interacts with the low-level policy by providing temporally extended goals to the low-level policy. *HIRO* uses *Twin Delayed DDPG (TD3)* (Fujimoto et al., 2018) to learn both μ^{lo} and μ^{hi}. During the training of μ^{hi}, goal relabeling of high-level goals is used to account for sub-optimal low-level policies.

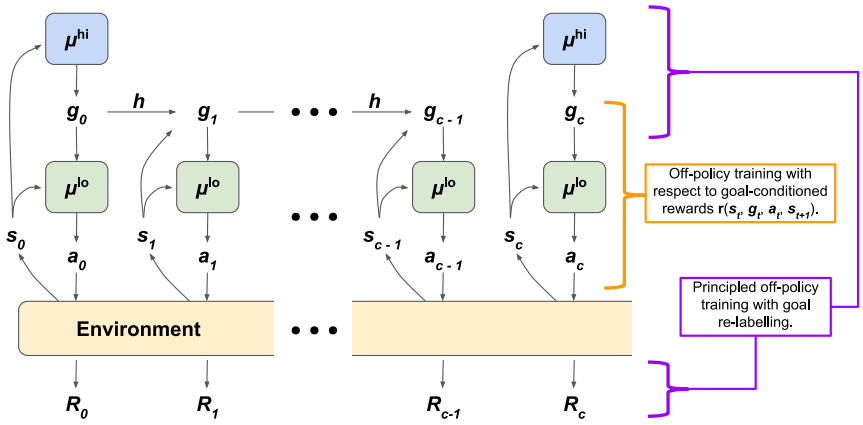


FIGURE 3.14: Schematics of the *HIRO* agent. Source: Nachum et al. (2018)

The goals for the high-level policy are the goals for the individual tasks: *Locomotion*, *Move Forklift*, *Move Anvil*, *Move Cone*, and *Control Drone*. The actions or goals produced by the high-level policy, which are the goals for the low-level policy, span the entire state space following the original implementation. For each task, one pair of low- and high-level policies are instantiated and trained independently of each other.

The high-level policy in *HIRO* is trained with the same shaped and dense reward signals as the ones used for training the SAC^+ agent to account for the challenging exploration problem. The low-level policy gets the same reward signal as in the original implementation. This privileged baseline is referred to by $HIRO^+$.

Intrinsic Motivation Baseline As IM baseline, *ICM* (Pathak et al., 2017) is used. *ICM* consists of two components: (1) A policy π that is learned with *Asynchronous Advantage Actor Critic (A3C)* (Mnih, Badia, et al., 2016) and a reward-generator that generates a curiosity-driven intrinsic reward signal r_t^i. The policy is trained to maximize the intrinsic reward and a potential extrinsic reward r_t^e. Figure 3.15 shows the schematics of the *ICM* agent. The policy is adapted to be goal-conditioned and uses the non-privileged reward signal as an extrinsic reward. An independent policy is learned for each task. In Pathak et al. (2017), the prediction error of a forward model is used as an intrinsic reward. Similarly, here the prediction error of the forward model in the CWYC framework is provided as an intrinsic reward to the learner.

This baseline is called *ICM*^{e}. Also, a version of *ICM* is tested in which the surprise signal is provided as an intrinsic reward which should give a much cleaner signal. This version is referred to by *ICM*^{s}.

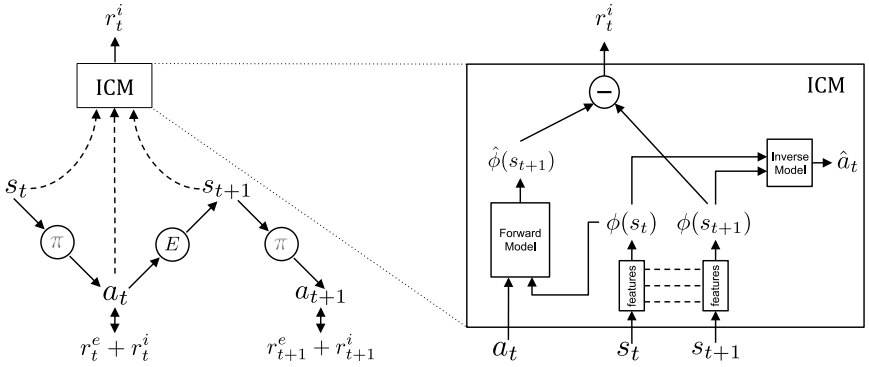


FIGURE 3.15: Schematics of the *ICM* agent. Source: Pathak et al. (2017)

3.5 Experimental Results

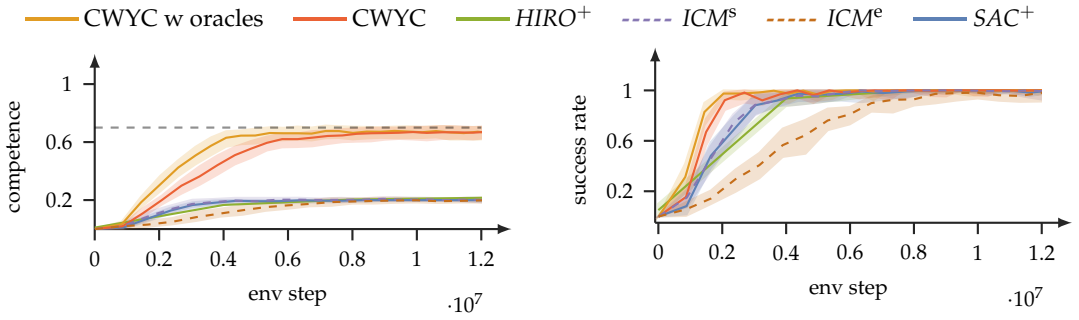
This section presents the experimental results for WAREHOUSE and FETCH PICK&PLACE TOOLUSE. The performance of *CWYC* is compared against the performance of *CWYC* with oracles, *HIRO*$^+$, *SAC*$^+$, *DDPG+HER* (only in FETCH PICK&PLACE TOOLUSE), *ICM*^{s}, and *ICM*^{e}.

3.5.1 Warehouse

Figure 3.16A shows the time evolution of the overall competence of the different agents during the developmental phase. The overall competence is defined as the average success rate among the various tasks:

$$\text{competence}(i) = \frac{1}{|\mathcal{T}|}\sum_{\mathcal{T}} \text{sr}_{\mathcal{T}}(i). \tag{3.24}$$

The means (solid lines) and standard deviations (color bands) from 10 independent runs with different seeds are plotted. In WAREHOUSE, the maximum achievable competence is 70% (broken horizontal line) due to the uncontrollable drone and the unreliable cone (which is controllable only in 50% of the trials).



(A) Overall competence of the agents in WAREHOUSE.

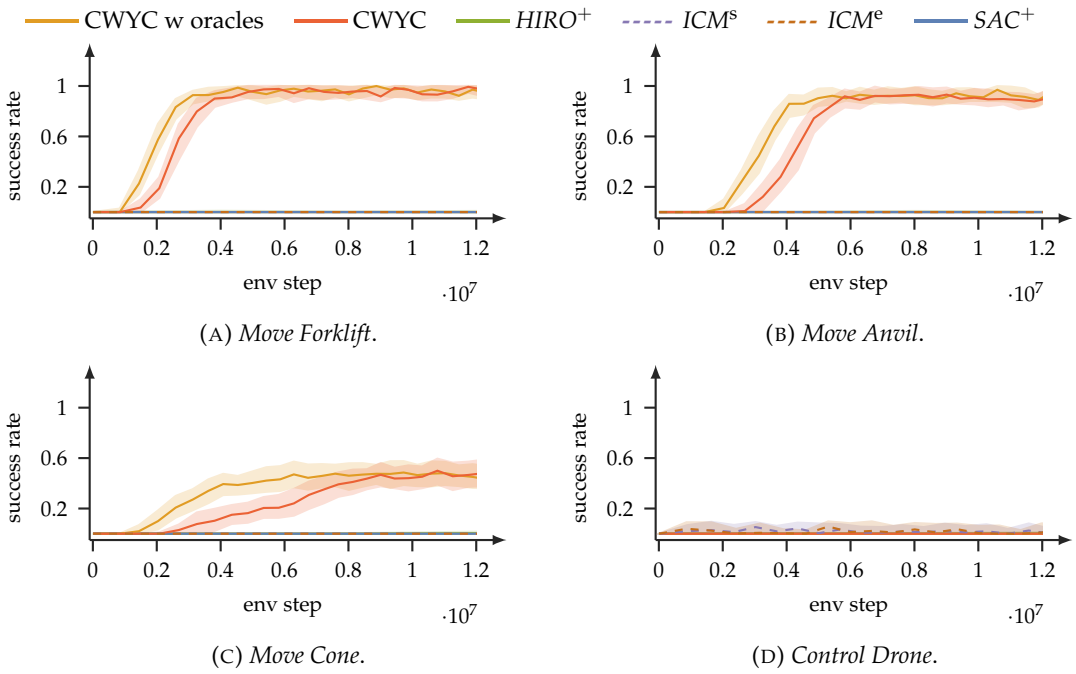
(B) Success rate of the agents in *Locomotion*.

FIGURE 3.16: (A) Overall competence in WAREHOUSE and (B) success rate of the agents in *Locomotion* throughout learning in the developmental phase. The x-axis shows the number of environment steps, i.e., the number of observations/transitioned collected in the environment.

CWYC with oracles achieves the maximum achievable competence the fastest. This is expected since in this baseline the **task planner**, **task graph** and the **goal generators** are replaced by oracles. Therefore, all that is left to learn for the agent are the low-level control policies. It is also important to note that the exploration problem is solved for this baseline as the low-level goal-reaching policies receive optimal goals right from the start of the developmental phase. *CWYC* also manages to reach the maximum achievable performance. However, the *CWYC* agent is slower as it has to learn all its components from experience during the developmental phase. All the other baselines achieve only sub-optimal performance. The competences of *HIRO*$^+$, *SAC*$^+$, *ICM*^{s}, and *ICM*^{e} plateau at 25%.

Figure 3.16B and Fig. 3.17A to Fig. 3.17D show the success rates of the different agents for the individual tasks. All the agents are capable of learning *Locomotion*. This is expected as it is the simplest goal-reaching task in which the agent has direct control over the task space via the robot's movement. In any of the more challenging

tasks, all the baselines fail to learn a successful control policy, while CWYC eventually achieves the same performance as CWYC with oracles.



(A) *Move Forklift*. (B) *Move Anvil*.

(C) *Move Cone*. (D) *Control Drone*.

FIGURE 3.17: Success rate of the agents in (A) *Move Forklift*, (B) *Move Anvil*, (C) *Move Cone*, and (D) *Control Drone* throughout training in the developmental phase.

Why do the baselines have that much trouble in solving the other tasks and how does CWYC manage to successfully solve all of the other tasks (except for the unsolvable *Control Drone* task)? To answer this question, it is insightful to look at the different components of CWYC and what they learned during the developmental phase.

First, let's look at how the different agents allocate their learning resources. As a reminder, all the baselines distribute their learning time equally between the different tasks. This can be inefficient because some of the tasks depend on other tasks and progress in these tasks can only be made after the other tasks are mastered to a certain degree. CWYC on the other hand distributes its learning efforts efficiently between the various tasks to maximize its overall learning progress. Figure 3.18 shows the task prioritization of the task scheduler throughout the developmental phase. At the beginning of the training, the CWYC agent spends most of its learning resources on *Locomotion* since the agent has direct control over the robot and the task does not have any prerequisites. Once the agent masters *Locomotion* at roughly $2 \cdot 10^6$ environment steps (see Fig. 3.16B) the agent starts to lose interest in the task (the probability of selecting *Locomotion* lowers) and begins to shift learning efforts towards *Move Forklift* (the likelihood of selecting the task increases) as *Move Forklift* has *Locomotion* as its only prerequisite. Simultaneously, the interest in *Move Cone* slowly raises as it also has *Locomotion* as its only prerequisite. However, the interest in the task grows much slower than the interest in *Move Forklift* due to its unreliable nature. Once the agent masters *Move Forklift* at around $3 \cdot 10^6$ environment steps, the agent switches to *Move Anvil* because the agent now acquired the skill to use the forklift to move the heavy anvil. Once all the fully controllable tasks are learned and resources free up, the agent begins to concentrate more and more learning efforts on the unreliable

cone. Since the agent does not have control over the autonomous drone, the agent's interest in that task drops quickly and stays at a constant low level.

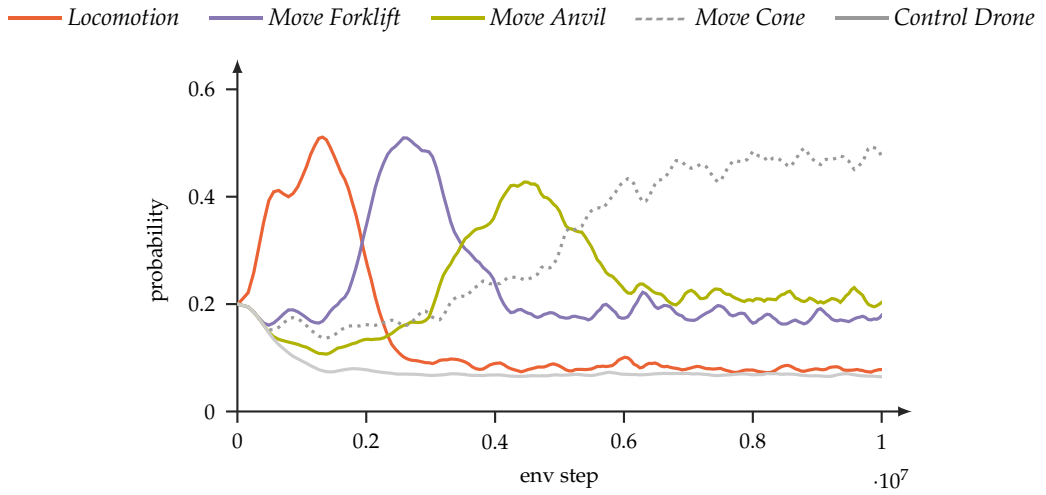


FIGURE 3.18: The learned task prioritization of the *CWYC* agent during the developmental phase. At the beginning of the developmental phase, the task scheduler concentrates most of the agent's learning efforts on *Locomotion*. After the agent masters the task, most of the learning efforts are shifted towards *Move Forklift*. Once the task is solved, the agent eventually concentrates on *Move Anvil*. Once all the other tasks are solved, the agent spends the remaining time learning *Move Cone*. A constant low priority is assigned to *Control Drone*.

All the baselines except for *CWYC* with oracles have to learn the different tasks with a single per-task policy in one shot. For instance, the *Move Anvil* policy has to first move the robot to the forklift, transport the forklift to the anvil, and eventually relocate the anvil with the help of the forklift to its target location. *CWYC*, on the other hand, learns specialized temporally-extended policies for the individual tasks that are only concerned with solving that specific task. The agent can then stitch together these expert policies to solve a final task. This has two advantages: (1) The individual policies can be much simpler as each policy is a simple goal-reaching policy. (2) The exploration problem becomes much easier because the individual policies have to collect only relevant data for solving the simple goal-reaching task.

	start	loco-motion	forklift	anvil	cone	drone
loco-motion	0.2	0.0	0.2	0.2	0.2	0.2
forklift	0.2	0.2	0.0	0.2	0.2	0.2
anvil	0.2	0.2	0.2	0.0	0.2	0.2
cone	0.2	0.2	0.2	0.2	0.0	0.2
drone	0.2	0.2	0.2	0.2	0.2	0.0

(A) Initial task transition matrix.

	start	loco-motion	forklift	anvil	cone	drone
loco-motion	0.9	0.0	0.0	0.0	0.0	0.0
forklift	0.0	0.9	0.0	0.0	0.0	0.0
anvil	0.0	0.0	1.0	0.0	0.0	0.0
cone	0.0	0.8	0.2	0.1	0.0	0.0
drone	0.2	0.3	0.2	0.2	0.2	0.0

(B) Learned task transition matrix.

FIGURE 3.19: (A) Initial and (B) learned task transition matrices for the warehouse environment. The labels within the cells show the probability of executing a task k before a task j.

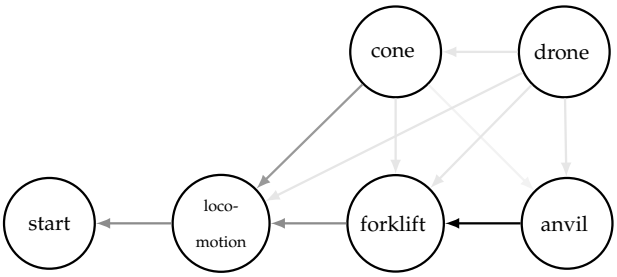


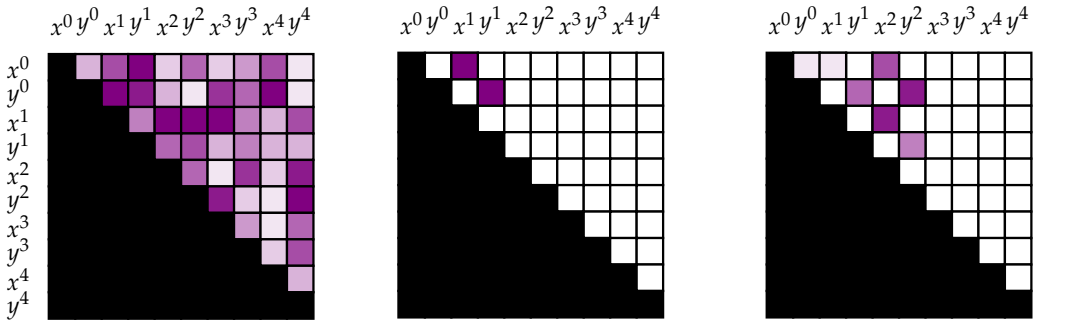
FIGURE 3.20: Learned task graph. The darker an arrow is, the higher the probability for the task transition to occur according to the learned task planner in Fig. 3.19B

Figure 3.19 shows the learned task transition matrix while Fig. 3.20 shows the corresponding learned task graph that is used to stitch together the different policies of the CWYC agent. The numbers in the transition matrix indicate the probability of executing a task k before task j. For instance, the probability of executing *Locomotion* before *Move Cone* is 80%. In comparison, the probability of executing *Move Forklift* before *Move Cone* amounts to 20%. In the task graph, arrows are darker if the probability in the task transition matrix is higher. The CWYC agent correctly learns that *Locomotion* does not have any prerequisites. In the case of *Move Forklift*, the agent has the highest chance of succeeding if it is proceeded by *Locomotion*. *Move Anvil* needs *Move Forklift* to be solved first to be successful. *Move Cone* can be solved if *Locomotion* precedes it. Surprisingly, the agent also assigns a low probability to the *Move Forklift* to *Move Cone* and *Move Anvil* to *Move Cone* transitions. This is because it is not critical for *Move Cone* if one of the other two tasks get solved first. However, as the time of solving *Move Cone* increases with the length of the subtask chain, these two task transitions have a much lower probability of being executed compared to the *Locomotion* to *Move Cone* transition.

As discussed in Sec. 3.2.5, it is not enough for the agent to know in which order the tasks have to be executed, but the agent also has to be in the right state at the end of each subtask to be successful in the next. These "relational funnel states" depend on the time-varying positional relations of the different entities in the environment. The task of the goal or attention network is to learn these relations. Figure 3.21 shows the learned attention weight maps for the *Locomotion* to *Move Forklift* (middle) and *Move Forklift* to *Move Anvil* (right) task transitions next to an initial random (left) attention weight map. Only the relevant part of the attention weight maps are shown in the visualizations, i.e., coordinates like the indicator variables are omitted. Moreover, the w^1 and w^2 attention weight maps belonging to one task transition are visualized in one single map that is computed according to:

$$w = \min\{|w^1|,\ |w^2|\}. \tag{3.25}$$

A non-zero value in the attention weight maps shown in Fig. 3.21A to Fig. 3.21C indicate that the coordinate pairs are involved in the relational attention computation. The weight matrices are initialized with random weights. Consequently, the initial goals produced by the goal proposal network are completely random and change chaotically every time step. Already after a few positive samples (see Sec. 3.2.5 for the definition), the attention network learns the relevant object relations for the different task transitions. For instance, in the *Locomotion* to *Move Forklift* transition (Fig. 3.21B), the relation between the robot and the forklift is important. More concretely, the attention is maximal if the distance between the robot and the forklift



(A) Initial attention weight matrix. (B) *Locomotion* to *Move Forklift*. (C) *Move Forklift* to *Move Anvil*.

FIGURE 3.21: Attention weight matrices of the (A) untrained network, (B) the *Locomotion* to *Move Forklift* task transition, and (C) the *Move Forklift* to *Move Anvil* task transition. The matrices are computed according to Eq. 3.25.

is close to zero. In the *Move Forklift* to *Move Anvil* transition, the relationship between robot and forklift, robot and anvil, and forklift and anvil are important. The attention is maximal if the distance between all three entities is zero.

Figure 3.22 shows the average distance between the learned goal g and the oracle goal $g^\star$ for the *Locomotion* to *Move Forklift* transition in relation to the number of positive samples. It is worth noting that first, the distance between g and $g^\star$ goes to almost zero after just a dozen positive samples. Second, it takes around $1.4 \cdot 10^6$ environment transitions to collect roughly a dozen positive training samples highlighting how rare interactions between the different entities are in the environment.

As discussed in the Methods section of this chapter, it is not enough to solely rely on the learning progress to learn the more difficult tasks because learning progress is quite rare, especially early on when learning a new task. Hence, the prediction error of a forward model is used to guide the exploration of the agent as long as no other training signal is available. Figure 3.23A shows a trajectory of the robot (red) in the environment. At some point in time, the robot collides with the forklift. The trajectory of the two is visualized in purple. Figure 3.23B shows the corresponding prediction error of the forward model. The observation of the moment in which the robot interacts with the forklift is identified as a surprising element. The respective observation is added to the training buffer of the attention network for the *Locomotion* to *Move Forklift* transition as a positive sample. All observations before this event are added as negative examples. Observations after the event are discarded

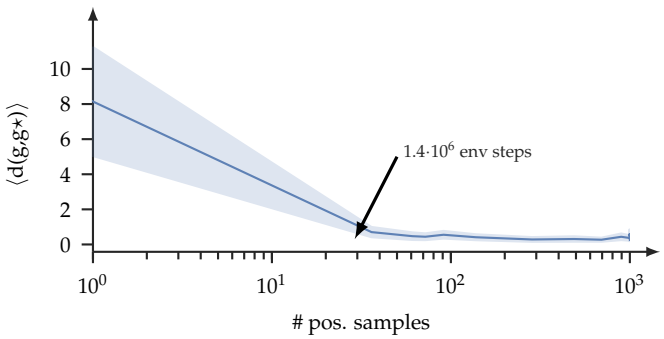
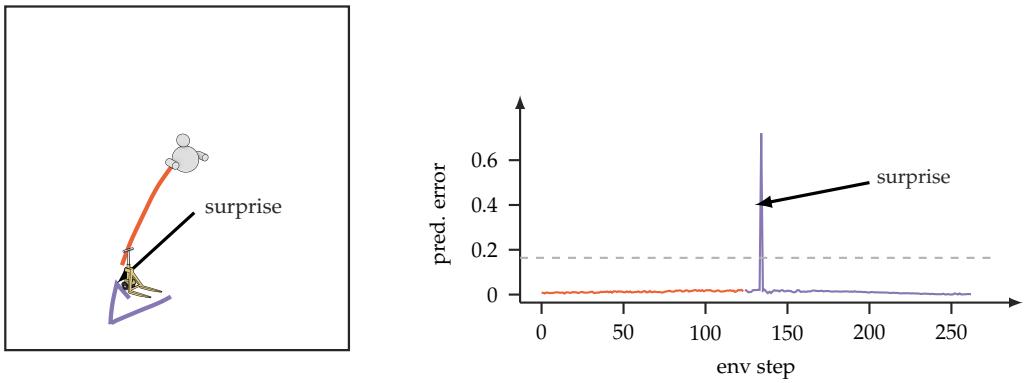


FIGURE 3.22: Distance between the learned goal g and the oracle goal $g^\star$ for the *Locomotion* to *Move Forklift* transition as the number of positive training samples increases.



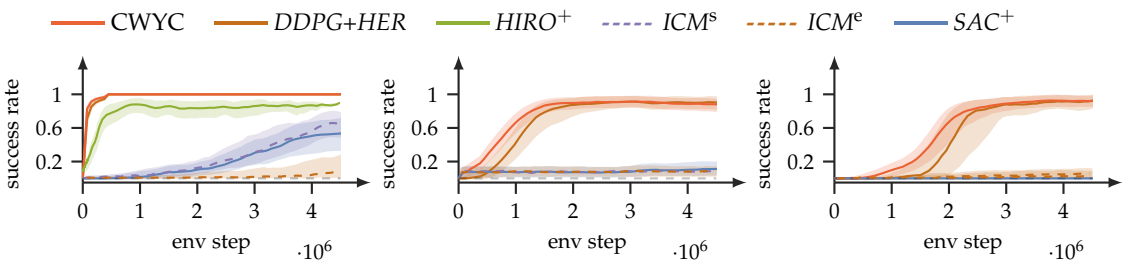
(A) Trajectory. (B) Prediction error.

FIGURE 3.23: (A) Trajectory of the robot (red) and the robot with the forklift (purple) in WAREHOUSE. (B) Corresponding prediction error of the forward model. The moment the robot collides with the forklift is identified as a surprising event as the prediction error exceeds a certain threshold (broken horizontal line).

as they are potentially misleading: They are not identified as a surprising event, but the same relationship between robot and forklift holds as for the observation that is identified as a surprising event. This can lead to conflicting signals during training as discussed in Sec. 3.2.5.

3.5.2 Fetch Pick&Place with Tool

Figure 3.24 shows the success rate for the different agents in FETCH PICK&PLACE TOOLUSE. *CWYC* manages to solve all three tasks close to perfection. Also, *DDPG+HER* manages to solve all three tasks sufficiently as it is a strong baseline for the OPENAI GYM FETCH PICK&PLACE environment. Still, there is a clear margin between the sample efficiency of *DDPG+HER* and *CWYC* that is a strong argument for adding inductive biases to increase the sample efficiency in RL. Again, the inductive biases in the *CWYC* agent are not hand-designed to work in this specific environment. Instead, the structured models in *CWYC* allow the agent to adapt to any environment from experience quickly. This observation becomes much more pronounced if the performance of *CWYC* is compared against the performance of the other baselines. *HIRO*$^+$ manages to solve *Move End-Effector* to a certain satisfaction but fails to learn any of the other tasks. *SAC*$^+$ and *ICM*^{s} struggle to even solve *Move End-Effector* and are not able to solve the *Use Hook* or *Move Cube* tasks. *ICM*^{e} does not manage to solve any of the tasks in FETCH PICK&PLACE TOOLUSE.



(A) *Move End-Effector*. (B) *Use Hook*. (C) *Move Cube*.

FIGURE 3.24: Success rates in the (A) *Move End-Effector*, (B) *Use Hook*, and (C) *Move Cube* tasks in FETCH PICK&PLACE TOOLUSE.

3.6 Ablation Studies

In this section, the impact of individual components of CWYC on the overall competence is studied by performing ablations. Some parts are more crucial for success than others. For instance, the goal proposal network is critical for the agent's success. Without meaningful and consistent goals, the robot moves aimlessly through the environment without a clear goal in mind. Similarly, the task scheduler is essential to solve the more challenging hierarchical tasks. For instance, if the agent tries to solve *Move Anvil* first and only afterwards *Move Forklift*, it will never succeed in any of the two tasks. On the contrary, the impact of the surprise signal and the task scheduler on the overall competence of the CWYC agent is more intricate, and it is not straightforward to evaluate their importance. Thus, in the next two paragraphs, these two components will be discussed in more detail by studying the following ablations of CWYC: CWYC^{s-} refers to a version of CWYC in which the surprise signal is ablated. CWYC‡ refers to a version of CWYC in which all tasks have a uniform, stationary priority of being sampled as the final task.

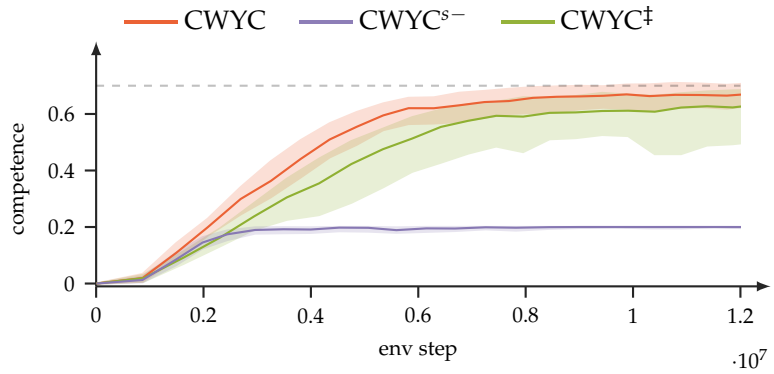
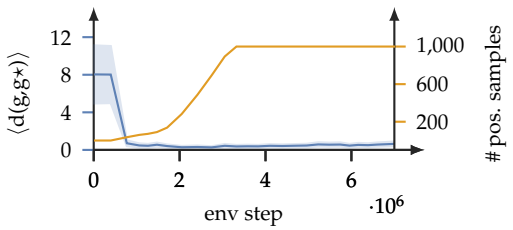


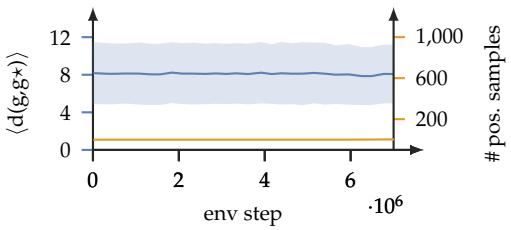
FIGURE 3.25: Overall competence of the CWYC, CWYC^{s-}, and CWYC‡ agents in the warehouse environment.

CWYC without surprise: Figure 3.25 shows the overall competence of CWYC^{s-} compared to the competence of CWYC. The CWYC^{s-} agent achieves only a fraction of the competence reached by CWYC. Only *Locomotion* is learned by CWYC^{s-} successfully as it is the most simple goal-reaching task among all the tasks.

What is the reason for the sub-optimal performance of the CWYC^{s-} agent? Figure 3.26 highlights the problem. Figure 3.26A shows the distance between the goals g proposed by the learned goal proposal network of CWYC and the oracle goals



(A) CWYC.

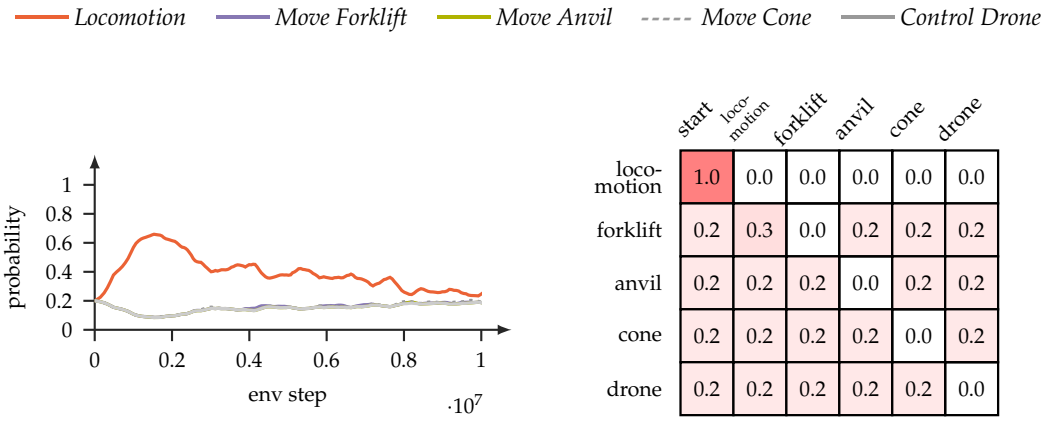


(B) CWYC^{s-}.

FIGURE 3.26: Number of positive samples in the training buffer of the goal proposal network over number of environment steps (yellow) and distance between learned goal g and oracle goal $g^\star$ for the *Locomotion* to *Move Anvil* task transition for (A) CWYC and (B) CWYC^{s-}.

$g^\star$ in comparison to the number of positive training examples for the *Locomotion* to *Move Forklift* transition. At the beginning of the developmental phase, it takes a very long time to get any positive training examples at all. These initial training examples come exclusively from surprising events generated during random collisions between the robot and the forklift. However, once the agent makes progress in learning *Move Forklift*, the learning progress signal leads to a sudden rise in the number of positive training samples at around $0.2 \cdot 10^7$ steps. In the case of the CWYC^{s-} agent, the initial positive training samples from the surprise signal are missing and the agent never comes to a point in which it can make any substantial learning progress in *Move Forklift*. Consequently, the goal proposal network never learns to propose any meaningful goals.

Due to the lack of a meaningful learning signal for the more challenging tasks, the task scheduler and task planner are also not able to adapt to the environment as shown in Fig. 3.27A and Fig. 3.27B. For the simple *Locomotion* task, the CWYC^{s-} agent gets a learning signal and therefore spends most of its learning resources on that task while completely ignoring all the more difficult tasks (Fig. 3.27A). Indeed, the task planner correctly learns that *Locomotion* does not have any prerequisites. However, it does not learn any meaningful task dependencies for all the other tasks (Fig. 3.27B).



(A) Task scheduler. (B) Task planner.

FIGURE 3.27: Learned (A) task scheduler and (B) task planner of the CWYC^{s-} agent.

CWYC without prioritized task scheduling: The effect of a missing task scheduler on the overall performance can be seen in Fig. 3.25. The impact on the overall performance is not as abysmal as for the CWYC^{s-} agent. Nevertheless, the agent still suffers from a significant drop in sample complexity and overall performance compared to CWYC. The reason for the sub-optimal performance of CWYC‡ is that since learning efforts are distributed uniformly between the different tasks, the agent tries to solve the more challenging tasks already early on in training. At this point, it cannot make any substantial learning progress. This sub-optimal distribution of learning resources results in a reduced learning speed.

3.7 Discussion

This chapter presented CWYC, an intrinsically motivated agent that learns to control its environment via self-motivated free play. CWYC can be seen as a hybrid architecture between model-based and model-free RL: The agent learns model-free low-level control policies to solve hierarchical goal-reaching tasks by offloading some of the planning complexity to structured models. The structured models serve multiple purposes: (1) They introduce inductive biases that allow the agent to adapt quickly to an environment from just a few observations. (2) They remove complexity from the low-level controller, and (3) they allow for a structured task-/goal-level exploration in contrast to the unstructured action-based exploration typically used in model-free RL.

The experimental section of this chapter demonstrated in two challenging object manipulation environments that the inductive biases introduced in the CWYC architecture help learn capable control policies where most of the baselines fail. The individual components of CWYC were discussed after they adapted to the specific environments to gain better insights into what they learned and how they facilitate the learning of the low-level controllers. The key components are: (1) A self-guided learning curriculum that distributes attention between tasks to maximize the overall learning progress. (3) An intrinsic motivation module that promotes the self-motivated free play of the agent in the environment and provides valuable learning signals for the other components. (4) A task-level planner with an associated task graph that allows the agent to compose multiple simple goal-reaching policies to solve challenging tasks. (5) A goal-proposal network that proposes goals for the subtasks in the task chain.

The choice of the more intricate inductive biases introduced in the CWYC architecture was justified by ablating these components and studying the impact on the overall performance. Without detecting surprising events, the CWYC agent fails to adapt any of the structured models to the specific environment, resulting in complete failure in any of the more challenging compositional tasks. Furthermore, without a proper learning curriculum, the agent spends too much energy on tasks in which it cannot make any learning progress, slowing down the learning progress significantly.

There are also certain drawbacks of an architecture like CWYC: (1) Imposing biases on the architecture in the form of hand-designed structured models makes it difficult for the architecture to adapt to situations that were not considered by the human designer or to find other optimal solutions that are potentially less intuitive for humans. (2) The disjoint nature of the individual modules increases the likelihood of individual points of failure. The other modules can hardly compensate for the failure of one module in the architecture as each model has its specific purpose. (3) The rigid structure of the architecture does not allow for unconventional solutions and makes concurrent training of the entire architecture more challenging.

Given the two extremes of fully unstructured end-to-end trained architectures and fully hand-design models, CWYC is an example of an architecture that sits between the two extremes and combines the strengths from both worlds. The structured models of the CWYC architecture inherit the benefits of hand-designed models in that they introduce inductive biases that facilitate rapid learning from just a few examples. Yet, these models have enough plasticity to adapt to the particularities of different environments by learning from data.

4

Sample-Efficient Action Planning and Imitation-Based Learning of Neural Network Policies in Model-Based Reinforcement Learning

This chapter is based on:

Pinneri*, Sawant*, Blaes, Martius (2021). "Extracting Strong Policies for Robotics Tasks from Zero-order Trajectory Optimizers", In: *International Conference on Learning Representations (ICLR)*. *equal contributiol.

Pinneri, Sawant, Blaes, Achterhold, Stueckler, Rolinek, Martius (2020), In: "Sample-efficient Cross-Entropy Method for Real-time Planning". In: *Conference on Robot Learning (CoRL)*.

Contents

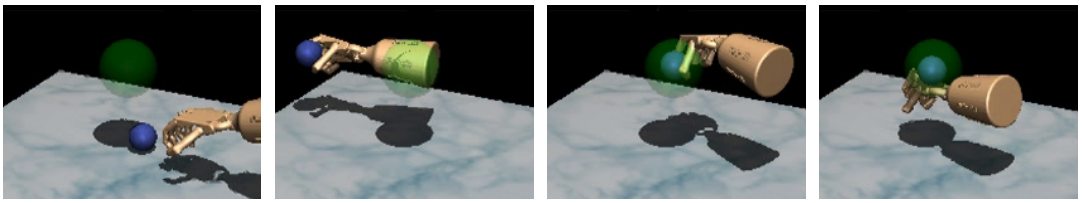
4.1 Introduction

This chapter explores the direct use of models inside control policies to solve challenging robotic manipulation tasks. In the previous chapter, it was assumed that the transition kernel of the Markov Decision Process (MDP) is unknown. Instead of learning an (approximated) model of the transition kernel, the *Control What You Can (CWYC)* agent learned various structured models that captured certain high-level aspects of the system that allowed the agent to efficiently explore the environment and plan for future success in compositional object-manipulation tasks.

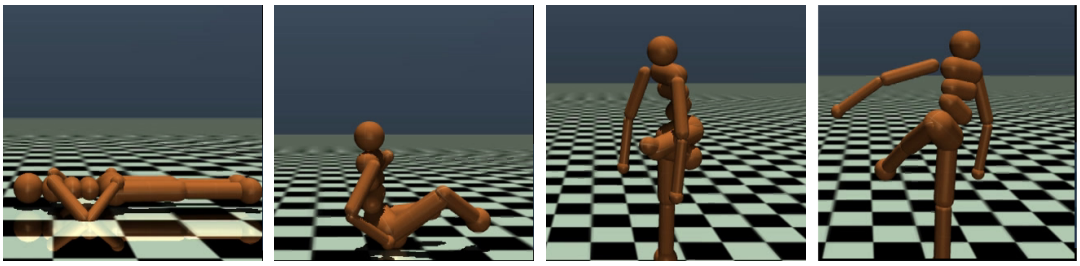
In recent years, using simulators as models for optimizing motion patterns for physical robots has become increasingly accepted in the Reinforcement Learning (RL) community with the emergence of new, highly parallelizable simulators like ISAAC GYM (Makoviychuk et al., 2021) and JAX MD (Schoenholz et al., 2020). Simultaneously, the simulators become more accurate by directly modeling certain simulation aspects from data (Lee et al., 2020). Together with techniques like domain randomization (OpenAI et al., 2019) this enables one-shot sim-to-real transfer. The simulator model can be considered an unstructured model $f : \mathcal{O} \times \mathcal{A} \rightarrow \mathcal{O}$ that maps from raw observations $o_t \in \mathcal{O}$ and actions $a_t \in \mathcal{A}$ to next raw observations o_{t+1} without imposing any additional structure onto the model's output. One way of utilizing the MDP model is to use Model-Based Reinforcement Learning (MBRL) to find planning-based control policies. Section 2.2.3 of chapter Ch. 2 discusses the various ways in which a model of the MDP can be used. Instead of planning in the task space and goal space, this work focuses on planning directly in the joint space as part of an Model Predictive Control (MPC) policy, see Sec. 2.2.3.3 in Ch. 2 for more details. The following two types of policies will be discussed in this chapter:

MPC Policies solve an optimization problem at each step during execution. Planning methods optimize a short trajectory snippet or plan into the future. Only the first action of the plan is executed before planning begins anew in the next time step. That means that all the heavy computation is done during execution time. Because this process can be very slow, MPC policies are typically hard to deploy in real-time operation domains. Despite this shortcoming, recent work in the MBRL community on population-based algorithms and sampling-based methods (G. Williams et al., 2015; Chua et al., 2018; Nagabandi et al., 2018; Hafner et al., 2019; T. Wang et al., 2020) show remarkable results in high-dimensional simulated robotic control tasks. Figure 4.1 shows two behaviors created by an MPC policy: In the DAPG RELOCATE environment, a 24 Degree of Freedom (DoF) simulated SCHADOW hand picks up a ball from a table and juggles it around a target location. In the OPENAI MUJOCO HUMANOID STANDUP environment, a 17 DoF humanoid robot stands up and balances itself in the upright position. These tasks are quite challenging to learn for model-free policy optimization methods, as can be seen in Fig. 4.11.

NN Control Policies, on the other hand, can be learned entirely offline. Assuming certain generalization capabilities, a Neural Network (NN) policy can be queried quickly for any arbitrary point in the state space. In that way, NN policies can offload most of the computational burden into the learning stage. NN policies can be learned in multiple ways. One possibility is to continuously interact with the environment and optimize a policy with policy gradient techniques as discussed in Sec. 2.2.1 of Ch. 2. This is usually referred to as online RL. Another way is to use offline RL or Imitation Learning (IL). In offline RL, access to the environment is prohibited during training. The learning agent has only access to a static offline dataset. Methods for learning NN policies from offline datasets include Behavioral Cloning



(A) DAPG RELOCATE.



(B) OPENAI GYM HUMANOID STANDUP environment.

FIGURE 4.1: Example behaviors found by the *improved Cross-Entropy Method (iCEM)*. (A) In the DAPG RELOCATE environment, a 24 DoF ADROID hand learns to juggle a ball around a target location. (B) In the OPENAI GYM HUMANOID STANDUP environment, a 17 DoF humanoid robot learns to stand up and balance in an upright position.

(BC) or online RL methods (Hussein et al., 2017) with adaptations to the offline setting. Recently, several algorithms (Kumar et al., 2020; Z. Wang et al., 2020; Yu et al., 2021) were proposed specifically designed for the offline RL setting.

Section 4.2.1 of this chapter presents *iCEM*, a sample-based trajectory planning algorithm that is based on the gradient-free Cross-Entropy Method (CEM) (R. Rubinstein, 1999) for trajectory optimization. By adding temporally structured exploration and memory to the optimizer, *iCEM* can be used for rapid and sample efficient planning in high-dimensional robotic control tasks. Moreover, *iCEM* produces close to optimal solutions in cases in which CEM completely fails to deliver any good solution. At the same time, *iCEM* uses only a fraction of the number of samples required by CEM to achieve the same or better performance.

Although *iCEM* is a big step toward making sample-based planning algorithms more suitable for real-time robotic control tasks, the disadvantage of doing most of the heavy computation during runtime remains challenging. Section 4.2.2 of this chapter presents *Adaptive Policy EXtraction (APEX)*, a framework for extracting NN policies from the near-optimal trajectories produced by *iCEM*. The MPC policy acts as a teacher for the NN student policy in *APEX*. Instead of having a purely one-sided influence between teacher and student, *APEX* allows for a mutual influence between the two. In that way, the NN policy improves over time, but the MPC policy also improves with a more capable NN policy. During the execution, the NN policy can be run alone, or the NN policy can be used to warm start the planner inside the MPC policy to decrease the sample complexity and runtime of the planner further.

4.2 Method

The following two sections present the iCEM algorithm for fast sample-based trajectory optimization and *APEX*, an IL-based NN policy extraction scheme with a planning-based teacher. In the following, the standard MDP formulation introduced in Sec. 2.1 of Ch. 2 is used. Additional remarks regarding notation differences between the RL and control literature can be found in Sec. 2.2.3.

4.2.1 Fast Sample-Based Trajectory Optimization

Section 2.2.3.3 of Ch. 2 introduces CEM as a gradient-free sample-based trajectory optimizer that recently gained traction in the model-based RL community, e.g., by the work of Chua et al. (2018), Hafner et al. (2019), and T. Wang et al. (2020). The particular appeal of methods like CEM lies in several important factors: (1) The possibility of optimizing black-box functions. All that is required from a function under consideration is that it can be queried for arbitrary inputs. (2) A lower sensitivity to hyperparameter tuning and thus higher robustness. (3) No requirement of gradient information. (4) A lower susceptibility to local optima.

However, there is a problem intrinsic to the nature of sample-based optimizers, which makes these methods often unsuitable for real-time planning. Planning methods as part of a closed-loop MPC policy do all the heavy computation during runtime. After each step in the environment, a new plan has to be computed incorporating the incoming stream of recent sensory information.

To make sample-based optimization for trajectory planning more applicable for real-time control, iCEM enhances CEM in several ways. The following sections discuss all the modifications.

4.2.1.1 Colored Action Noise Exploration for Broad State-Space Coverage

CEM adapts the parameters θ of its sampling distribution according to the population statistics of an elite set. The elites get selected from the sample population based on a ranking computed by evaluating each sample w.r.t. a performance metric. While the parameters of the sampling distribution are adapted along the action dimension, there is no correlation between actions along the time axis. Applying temporally uncorrelated actions to a robotic system typically leads to only minor deviations of the joints from their initial position, resulting in a poor coverage of the full state space. This is shown by the blue state-space trajectory in Fig. 4.2A for a 1-dimensional point-mass system. However, finding an optimal solution of the trajectory planning problem heavily relies on an initially highly diverse sample population, i.e., in a broad coverage of the state space. The lower the diversity in the sample population, the more samples are needed and the longer the planning horizon has to be in order to find an optimal solution. Consequently, this will increase the computational burden of each planning step in the MPC policy.

In nature, animals revert to different exploration strategies when they need to efficiently explore the space in search for food, rather than plain Brownian exploration. When prey is scarce, animals like sharks or other predatory species produce trajectories which can be described by the so-called Lévy walks (Humphries et al., 2010). Classically, Lévy walks exhibit velocities with long-term correlations and consequentially produce trajectories with higher variance than a Brownian motion (Shlesinger et al., 1982).

A way of analyzing the temporal correlation structure of an action sequence sampled from different types of noises is via the Power Spectral Density (PSD) of the

action sequence:

$$\mathrm{PSD}_{a_0,\ldots,a_{H-1}}(f) \propto \frac{1}{f^\beta}, \tag{4.1}$$

with $(a_i)_{i=0}^{H-1}$ being the action sequence under consideration and f being a particular frequency component of the signal. Intuitively, the PSD quantifies how much each frequency is present in the time series or, in other words, how the energy is distributed between the frequency components in the time series.

For instance, CEM samples temporally independent actions from a multivariate Gaussian sampling distribution $\mathcal{N}(\mu_a,\ \Sigma_a)$, with means μ_a and diagonal covariance matrix $\Sigma = \sigma_a^2 \cdot \mathbb{1}$. This results in a constant power spectrum, with $\beta = 0$ (also known as white noise), as it is shown by the blue curve in Fig. 4.2B.

To achieve a broader coverage of the state space as shown by the pink and brown curves in Fig. 4.2A, a larger exponent β of the power-law distribution can be chosen. As it can be seen by Fig. 4.2B, a larger exponent results in a larger contribution of the lower frequency components in the respective PSD, while higher frequencies get more and more suppressed. This corresponds to less erratic changes in the control outputs; thus, to much smoother behavior of a robot.

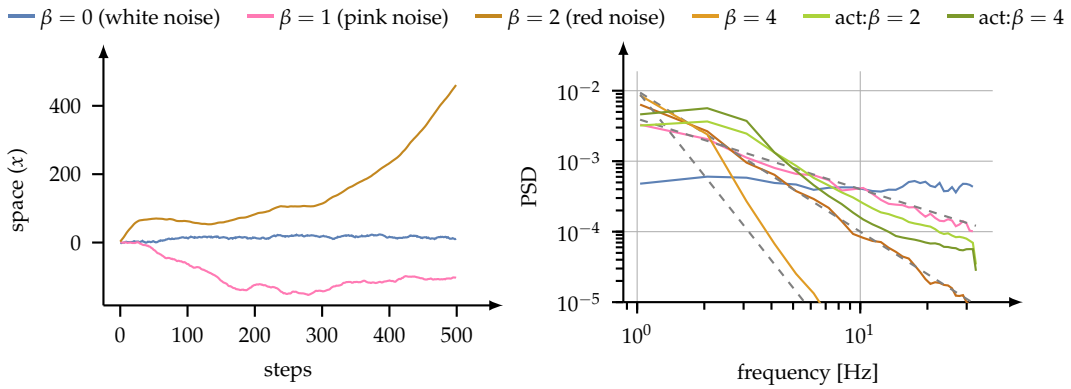
To this end, generalized colored-noise for the action sequence $(a_i)_{i=0}^{H-1}$ is introduced as power-law distributed noise as defined in Eq. 4.1. An exponent $\beta = 0$ corresponds to white noise while an exponent of $\beta > 0$ means that higher frequencies are less prominent in the PSD than lower ones. In signal processing this type of noise is also called colored noise and some exponents have a particular name. For instance, colored noise with an exponent of $\beta = 1$ is known as pink noise, and Brownian or red noise has an exponent of $\beta = 2$.

An efficient implementation of a noise generator (Timmer et al., 1995) is used that is based on the Fast Fourier Transform (Cochran et al., 1967) to produce correlated action sequences. The implementation relies on the fact that the PSD of a time series can be directly modified in the frequency space. To sample actions with a PSD as in Eq. 4.1, the following transformation needs to be applied to the original action sequence $(a_i)_{i=0}^{H-1}$ sampled from a white noise process:

$$\bar{a}_i = \mathcal{F}^{-1}\left[\frac{1}{f^{\beta/2}}\mathcal{F}[a_i]\right] \quad \text{gives} \quad \mathrm{PSD}_{\bar{a}}(f) = \left\|\frac{1}{f^{\beta/2}}\mathcal{F}[a_i](f)\right\|^2 = \frac{1}{f^\beta}\mathrm{PSD}_a(f) \propto \frac{1}{f^\beta}. \tag{4.2}$$

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

Evidence for the benefit of non-constant power spectra in robotic control tasks can be seen in Fig. 4.2B. From frequency response analysis for dynamical systems, it is well known that different dynamical systems, like robotic systems, have very specific Frequency Response Functions (FRFs) (Sinha, 1989). The green curves in Fig. 4.2B show the PSDs of action sequences that allow a humanoid robot to stand up. The two successful runs are far away from white noise ($\beta = 0$) with exponents between $\beta = 2$ and $\beta = 4$.



(A) 1D random walks with colored noise. (B) PSD of different power-law distributions.

FIGURE 4.2: Colored random noise. (A) Random walks with colored noise of different temporal structures. (B) Power spectrum of colored random action sequences for different β and two successful action sequences in the HUMANOID STANDUP task.

4.2.1.2 Reusing Information Between Planning Steps

The MPC policy is closed-loop. Consequently, replanning needs to happen anew in every step to incorporating all the new sensory information coming from the environment. Typically, only information in the form of the updated sampling distributions gets transferred between planning steps, while all the other information, e.g., the actual solution set, gets discarded entirely. According to the parameters used in Chua et al. (2018), this amounts to an average of ∼55000 discarded actions per step.

This is despite two observations: (1) Although there might be a deviation between the imagined model state prediction and the actual system state after executing the first action, this deviation is typically small. (2) The solution set rarely collapses to a singular solution but covers a small neighborhood around the best solution from which the first action gets executed. These two observations make it clear that the previous solution can contain valuable information for the next planning step. To acknowledge this fact, memory is added to iCEM that allows it to reuse some of the data from the previous plan.

Keep Elites

iCEM keeps a fraction of the previous elites between inner CEM-iterations and adds them to the new sample population in the next CEM iteration. Again, previous solutions might still be valid if the deviation between imagined and actual outcomes is small enough and if the elites cover a small neighborhood around the optimal solution. There are a lot of settings, e.g., sparse-reward settings or settings in which certain funnel states have to be reached, in which a very particular solution has to be found. Once such a solution is found by iCEM, it should not lose this solution between CEM iterations.

Example: *In the* FETCH PICK&PLACE *environment shown in Fig. 4.3, the task of the robot arm is to pick up a cube that is randomly spawned on a table in front of the robot and bring it to a target location either on the table or in the air. The reward signal in this environment is partially sparse in that only the distance between the cube and target location is considered but not the distance between end-effector and cube. As long as the end-effector is not in contact with the cube, the reward signal is constant. Consequently, finding a solution*

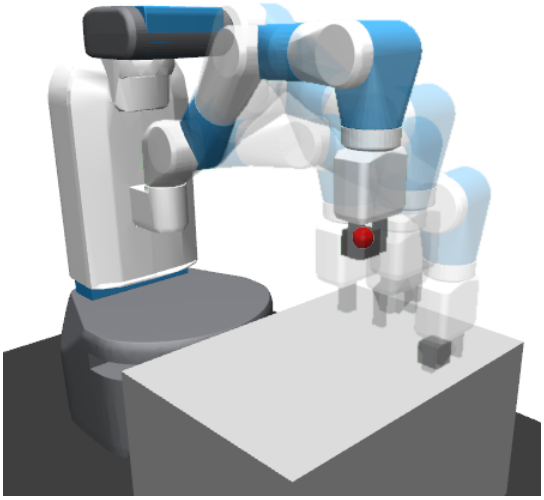


FIGURE 4.3: In the OPENAI GYM FETCH PICK&PLACE environment, a fetch robot has to pick up a cube and bring it to a target location either on the table or in the air. The cube's position on the table is a funnel state through which the robot has to go through to be successful.

in which the end-effector touches the cube is very unlikely with purely sampling-based exploration. Once such a solution is found, it might be the only viable candidate in the sample population, marginalizing out its impact in the update of the parameters of the sampling distribution.

Shift Elites

To not lose a promising solution found in one time step, *iCEM* carries over the elites of the previous planning step to the next. In experiments, it can be observed that a particularly good solution typically survives for a longer time in the elite set over multiple consecutive planning steps. Thus, these solutions are more emphasized in the updated of the sampling distribution parameters. Not the entire elite-set is carried over, though, because that would drastically shrink the variance of the sampling distribution in the first CEM-iteration. Consequently, only a fraction of the elites are kept.

Once some of the elites are carried over to the next time step, the first action needs to be discarded from the action plans because it was already sent to the robot. Ergo, the previous elite plans are too short, as they are now missing the last action. To restore the full planning horizon, iCEM samples a new action from the sampling distribution and appends it to the plans carried over from the previous time step.

4.2.2 Neural Network Policy Extraction

Figure 4.1 shows some of the behaviors found by iCEM in high-dimensional robotic control tasks. This section discusses how high-performing NN policies can be extracted from these behaviors.

Extracting an NN policy from a powerful sample-based optimizer like iCEM is one of the missing pieces to truly bridge the gap between model-based RL in simulation and real-time robotics. As of today, this is still an open challenge (T. Wang et al., 2020).

The following sections discuss the issues that arise during distillation of a multi-modal stochastic teacher into a NN policy. To this end, *APEX* is introduced as an

adaptive policy extraction procedure that integrates iCEM with BC and Dataset Aggregation (DAgger) (Stéphane Ross, G. Gordon, et al., 2011), and a novel adaptive variant of Guided Policy Search (GPS) (Levine and Koltun, 2013). Although the main goal of *APEX* is to distill an NN policy, the specific integration of methods also produces an improving adaptive teacher with higher performance than the original iCEM optimizer.

4.2.2.1 Imitation Learning

BC is used to imitate the actions of the iCEM-MPC teacher policy with an NN student policy. BC is the simplest form of imitation learning in that it minimizes the log-likelihood loss between the teacher action distribution and the student distribution in a purely supervised fashion:

$$\mathcal{L}_\theta = \mathbb{E}_{(s,\, a)\sim\mathcal{D}_{\text{iCEM}}} \left[-\log \pi_\theta(a \mid s) \right], \tag{4.3}$$

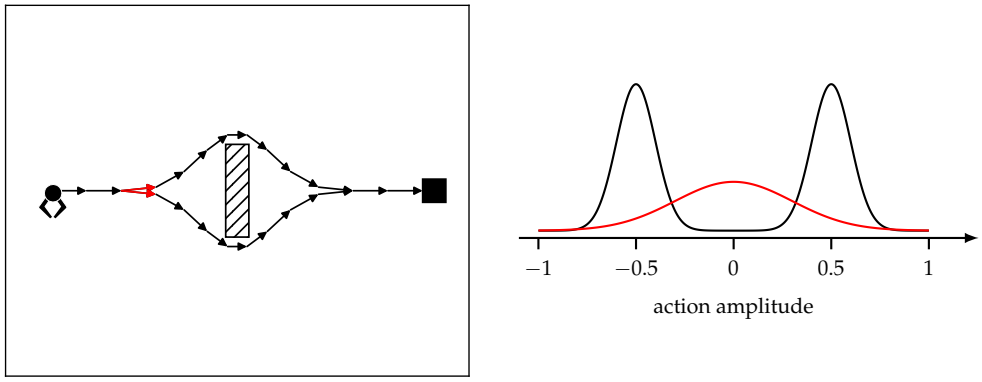
with π_θ being the NN policy with parameters θ and state-action tuples $(s,\ a)$ being sampled from a dataset $\mathcal{D}_{\text{iCEM}}$ of iCEM rollouts.

There is an issue with the vanilla BC loss. It is only minimized on samples from the teacher state-visitation distribution. Thus, as soon as the student leaves the distribution of states visited by the teacher, no guarantees can be given regarding the student's performance. This problem is known as covariate shift (Stéphane Ross and Bagnell, 2010) between teacher and student distribution and arises from the interactive and sequential nature of RL.

DAgger
A straightforward way of mitigating the covariate shift is to use DAgger (Stéphane Ross, G. Gordon, et al., 2011). In DAgger, data is collected not only from the teacher but also from the student. Whenever the student creates data outside of the teacher distribution, i.e., the student visits states not visited by the teacher while executing suboptimal actions, the data gets relabeled with optimal actions from the teacher. This limits the class of teachers to those that can be queried for arbitrary observations such as iCEM.

Guided Policy Search
Even if started from the same initial configuration, *iCEM* can yield vastly different solutions for the same problem due to its stochastic nature and the finite sample size during planning. This renders the NN training very challenging
Example: *In the situation depicted in Fig. 4.4A, a robotic gripper tries to reach a cube far away. Between the gripper's and cube's position is an obstacle that the gripper has to bypass to get the cube. The robotic gripper can decide to take the upper or lower trajectory. Since the iCEM planner is stochastic, sometimes it will find one and some other times the second solution, resulting in the multi-modal distribution (black) shown in Fig. 4.4B. If an NN policy gets trained from the multi-modal distribution shown in Fig. 4.4B, it will learn the average action shown as a red curve. This results in a catastrophic failure as the robotic gripper collides with the obstacle.*

GPS is used to mitigate the multi-modality issue. GPS adds a penalty term to the cost function of the iCEM planner that penalizes large deviations between the action distribution of the NN policy $\pi(a \mid s)$ and the empirical action distribution of the



(A) Model planning. (B) Multi-modal action distribution.

FIGURE 4.4: (A) A gripper robot tries to reach a cube while avoiding the obstacle between gripper and cube. Arrows indicate state transitions. The state transitions highlighted with red arrows mark a bifurcation point. Depending on which action the robot chooses, it will either take the upper or the lower trajectory. (B) Multi-modal action distribution (black) results in the bifurcation point on the left. A value of $a = -0.5$ will lead to the lower trajectory. A value of $a = 0.5$ means that the gripper follows the upper trajectory. By minimizing the BC loss, the NN policy learn the average action (red).

sample population $\mathcal{N}(\mu_a, \sigma_a^2)$:

$$a_t^\star \leftarrow \arg\min_{a_t} J_t(a_t, s_t) + \lambda D_{\mathrm{KL}}(\pi_\theta || \mathcal{N}(\mu_a, \sigma_a^2)), \tag{4.4}$$

with λ being the Lagrange multiplier for the relaxed constrained optimization problem and $D_{\mathrm{KL}}(P||Q)$ being the Kullback–Leibler divergence (Kullback et al., 1951) between distributions P and Q. The μ_a and σ_a^2 are estimated from the sample population.

4.2.2.2 Neural Network Policy Informed Trajectory Optimization

The GPS cost establishes a mutual influence between the NN and MPC policies via the Kullback–Leibler divergence between their respective action distributions. The following sections discuss issues of vanilla GPS, a possible solution to these issues, and other ways of making the connection between the NN and MPC policies tighter.

Adaptive Auxiliary Cost Weighting

There is a fundamental problem with the GPS cost term (as well as with any other auxiliary cost term). Instead of solving the original optimization problem, a modified optimization problem is solved. The new problem has the form of a linear combination between the original cost J and any additional cost terms with a fixed mixing term λ. There are instances in which the modified optimization problem does not yield a viable solution although such a solution exists for the original optimization problem.

Example: *In the situation depicted in Fig. 4.5, the NN policy (shown in red) is suboptimal and proposes a solution in which the robotic gripper collides with the obstacle. Since the weighting term of the GPS cost is fixed, the optimizer (black) cannot account for the suboptimality of the policy and therefore fails to find one of the viable solutions shown in Fig. 4.4A.*

This issue is solved by introducing an adaptive λ_j-term for each auxiliary task C_j^{aux} in the total cost:

$$a_t^\star \leftarrow \arg\min_{a_t} J_t(a_t,\ s_t) + \sum_j \lambda_j C_j^{\text{aux}}(a_t,\ s_t), \tag{4.5}$$

with λ_j being defined as:

$$\lambda_j = c_j \frac{\mathcal{R}(J)}{\mathcal{R}(C_j^{\text{aux}}) + \epsilon} \tag{4.6}$$

and

$$\mathcal{R}(X) = \max_{\text{elite-set}} X - \min_{\text{elite-set}} X\,. \tag{4.7}$$

To understand how the adaptive λ works intuitively, it helps to consider the case in which there is no variance in the main objective J, i.e., $\mathcal{R}(J) = 0$. With a locally flat cost landscape, there is no point in further restricting the solution set of the optimizer by adding any auxiliary costs. The planner should explore the solution space as freely as possible by setting $\lambda_j = 0$ as its main objective is to decrease J. Only if the optimizer gets a proper signal for the original cost, i.e., $\mathcal{R}(J) > 0$, it makes sense to restrict the solution set to respect any further constraints of the optimization problem. The denominator in Eq. 4.6 ensures that the auxiliary cost weight increases the more the solution violates the constraint. The λ_j are upper bounded by the constants $c_j \geq 1$ and $\epsilon \ll 1$ is a regularization constant.

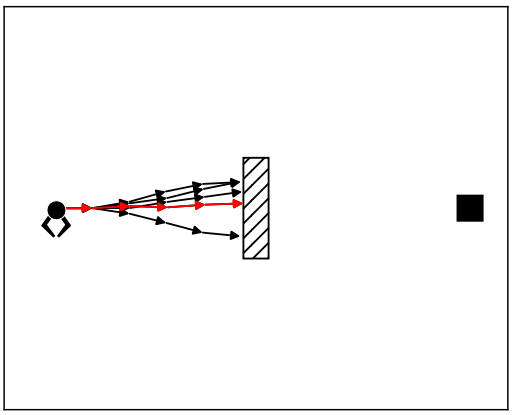


FIGURE 4.5: (Red) Trajectory produced by a sub-optimal NN policy. (Black) MPC trajectories with GPS and a fixed cost weighting.

Policy Informed Shift Initialization
To further strengthen the coupling between the NN and MPC policies, iCEM uses the mean action proposed by the NN policy to initialize the last step of any plan that results from shifting an elite from a previous planning step to the next, see Sec. 4.2.1.2 for more details about shifting elites.

Warmstarting
In addition, the MPC policy gets warmstarted with the NN policy. Two versions of warmstarting are tested: (1) The mean of the iCEM sampling distribution is initialized with the mean of the action distribution of the NN policy. (2) A sample is added

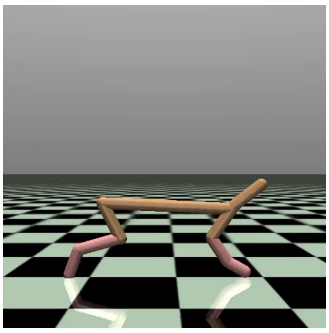
to the iCEM sample population in which the actions along the planning horizon are the mean actions proposed by the NN policy.

Both versions have a slightly different effect on the sampling distribution: Version (1) directly influences the sampling distribution by changing its mean parameter. This can result in a collapse of the sample population to a suboptimal region of the optimization landscape, as discussed in the case of the adaptive auxiliary cost weighting. The effect of version (2) on the sampling distribution is more subtle. The NN policy sample affects the sampling distribution only indirectly if it is part of the elite set and gets used in the update of the sampling distribution parameters. If the NN policy is already well trained, the sample from the NN policy might already solve the task.

The version of GPS with warmstarting is referred to by GPS_{π}.

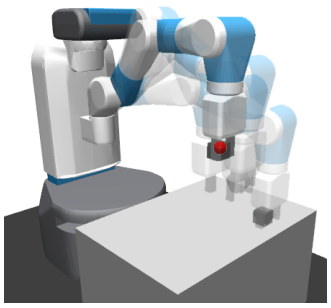
4.3 Environments

The performance of iCEM and *APEX* is studied in various challenging high-dimensional simulated robotic control tasks with continuous observation and action spaces. From OpenAI Gym (Brockman et al., 2016), the HalfCheetah Running (Fig. 4.6a), Humanoid StandUp (Fig. 4.6b) and Fetch Pick&Place (Fig. 4.6c) tasks are considered. In HalfCheetah Running, the goal of a 6 DoF cheetah is to maximize the velocity along the x-axis without falling. The cheetah can only move in the xz-plane. A rolling motion of the cheetah, commonly found by strong optimization schemes, is prohibited by heavily penalizing large angles of the root joint. The observation-space of the HalfCheetah Running environment is 18 dimensional and includes the absolute position of the cheetah, all the joint angles, as well as the joint angular velocities. In Humanoid StandUp, the goal of a 17 DoF humanoid robot is to stand up from a lying position and balance itself in an upright position. The observation-space of Humanoid StandUp is 376-dimensional and includes the humanoid's absolute position, joint angles, and angular velocities, as well as measurements of external forces and mass and inertia information. In Fetch Pick&Place, a 7 DoF Fetch robot has to pick up a cube and bring it to a target location on the ground or in the air. The $(25 + 3 + 3)$-dimensional observation space contains information about the Fetch robot and the cube, as well as their relative positioning. Additionally, the desired and achieved goal states (each 3-dimensional) are appended to the observation vector. The 4 actions control the movement of the end-effector in Cartesian space and the opening and closing of the gripper. The reward signal is partially sparse in that it is computed as the distance between the cube and target location. Thus, the reward is constant as long as the cube is not moving. This makes the exploration problem particularly hard in this environment, since the agent does not receive any feedback as long as it does not make progress on the actual task.



(a) HalfCheetah Running.

(b) Humanoid StandUp .



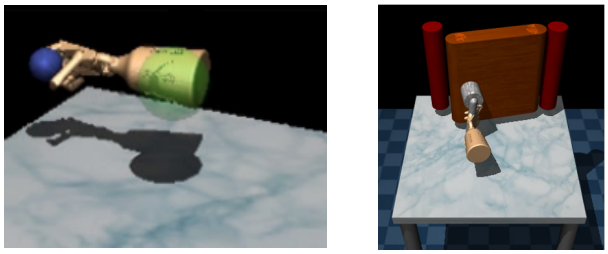
(c) Fetch Pick&Place .

Figure 4.6: OpenAI Gym environments. The performances of iCEM and *APEX* are tested in (a) HalfCheetah Running, (b) Humanoid StandUp and (c) Fetch Pick&Place.

The other set of environments comes from the DAPG project (Rajeswaran et al., 2018) and includes a 24 DoF Adroid hand that has to solve various dexterous hand manipulation tasks. The Adroid hand is not attached to a body and can freely float in the air.

In this work, the Relocate (Fig. 4.7a) and Door (Fig. 4.7b) environments are considered. The task in Relocate is to pick up a ball and bring it to a random target location in the air. The ball is randomly spawned on a table in front of the hand. The reward signal in this environment is the sum of the distance between

the palm and the ball and the distance between the ball and the target location, as well as bounties for lifting the ball and getting the ball close to the target. The 39-dimensional observation space contains information about the hand, the ball, the target location, and their relative relationship. The task in DOOR is to open a door in front of the robot. The reward in this environment is partially sparse. It is computed as a real-valued indicator variable of the door's openness and does not consider the distance between palm and door. The reward signal is constant if the hand cannot open the door at all. Again, this makes the exploration problem in this environment very hard. The 39 dimensional observation space contains information about the hand and the door, including the position of the latch and handle and their relative positions to the hand, as well as an indicator variable for the door's openness.



(A) RELOCATE. (B) DOOR.

FIGURE 4.7: DAPG environments. The performances of iCEM and *APEX* are tested in (A) RELOCATE and (B) DOOR.

4.4 Baselines

This section discusses the different baselines used to compare the performances of iCEM and *APEX* with their respective state-of-the-art counterpart algorithms. Where applicable, the same settings are used across all baselines.

Fast Sample-Based Trajectory Optimization iCEM is compared against the following baselines. The first baseline is the plain CEM as discussed in Sec. 2.2.3.3 of Ch. 2.

The second baseline adds various standard modifications to the vanilla CEM: (1) A momentum term (De Boer et al., 2005) in the refitting of the distributions between the CEM-iterations:

$$\mu_t^{i+1} = \alpha \mu_t^i + (1-\alpha)\mu^{\text{elite}-\text{set}_i}, \tag{4.8}$$

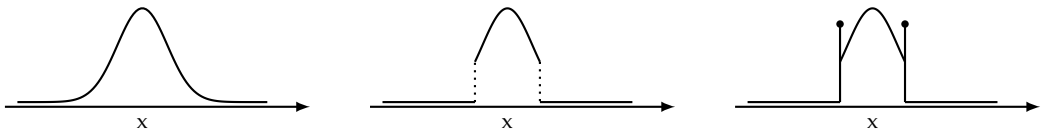
where $\alpha \in [0,1]$ and i is the index of the inner CEM-iterations. The μ_t^i are the current means of the sampling distributions and $\mu^{\text{elite}-\text{set}_i}$ are the estimates of the new means based on the elite set. The reasoning behind the momentum term is that only a small elite set is used to estimate the sampling distribution's many parameters, resulting in a poor signal-to-noise ratio. (2) For sampling bounded actions, a truncated normal distribution (Fig. 4.8B) with suitable bounds is used instead of an unbounded normal distribution (Fig. 4.8A) with clipping (Fig. 4.8C). The difference between truncating and clipping is that in the case of clipping, samples outside of the bounds are clipped to the nearest boundary, resulting in a probability mass concentration around the boundaries as an artifact of the clipping operation. An accept-reject sampling method is used to sample from a truncated normal distribution, e.g., new samples are generated until a sample falls inside the bounds. This avoids the oversampling of values at the boundaries. (3) As discussed in Sec. 2.2.3.3, executing a plan in an MPC fashion means to solve the exact same problem again going from one time step to the next with the horizon shifted by one. As a typical modification (Chua et al., 2018; T. Wang et al., 2020), the initial mean μ_t of the CEM distribution is shift initialized from the optimized μ_{t-1} according to:

$$\mu_t(\cdot, j) = \mu_{t-1}(\cdot, j+1) \qquad \text{for } 1 \leq j \leq h-1 \tag{4.9}$$

$$\mu_t(\cdot, h) = \vec{0}, \tag{4.10}$$

where the parenthesis denote index-access: (action dimension, horizon timestep). This variance of CEM is called CEM_{MPC}. Also *iCEM* applies the same standard modifications to *iCEM* as CEM_{MPC}, with the exception of initializing $\mu_t(\cdot, h)$ in Eq. 4.10 with $\mu_{t-1}(\cdot, h)$.

To make the comparison between iCEM and the baselines more intuitive, an



(A) Normal distribution. (B) Truncated normal distribution. (C) Clipped normal distribution.

FIGURE 4.8: Comparison between (A) a normal distribution, (B) a truncated normal distribution, and (C) a clipped normal distribution.

TABLE 4.1: 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

overall planning budget is defined as the total number of trajectories per step. Table 4.1 shows the different budgets used in the experimental evaluations. For any given budget, the first number indicates the number of internal CEM iterations per step (see Sec. 2.2.3.3 of Ch. 2) and the second number shows the number of sampled imagined trajectories.

Chua et al. (2018) modifies the truncated sampling distribution such that the bounds are always set to 2σ, where σ is adapted to be not larger than $\frac{1}{2}b$, with b being the minimum distance to the action bounds. This variance is referred to as CEM_{PETS}.

NN Policy Extraction *APEX* is compared against several IL baselines, as well as ablations of *APEX*. Since the main interest is in the performance of the extracted NN policy, the same iCEM-MPC teacher policy is used in all the baselines.

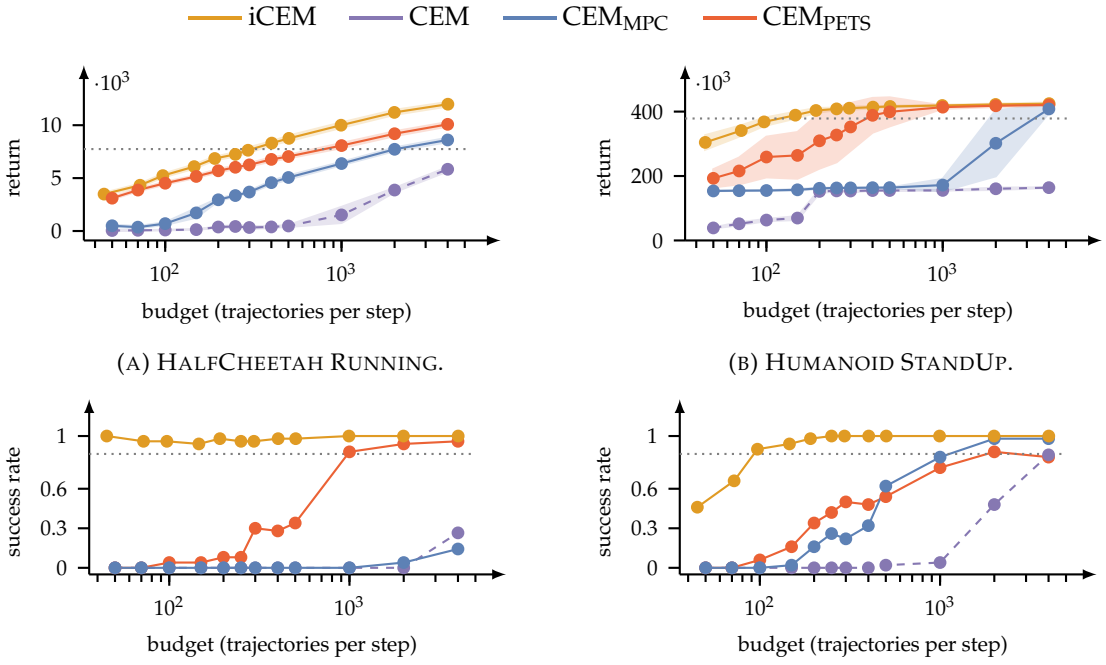
As the simplest baseline, BC is used without any feedback loop between the NN and MPC policies. The BC-DAgger baseline adds DAgger to BC, still without any feedback loop between the NN and MPC policies. Finally, BC is tested with guidance cost (fixed λ) and warm-starting ($\text{BC-GPS}_{\pi}^{\lambda^{\text{fixed}}}$).

For reference, the performance of Soft Actor Critic (SAC) as a model-free RL baseline is provided to get an idea of the difficulty of the learned tasks.

4.5 Experimental Results

The following sections present and discuss the experimental results for *iCEM* and *APEX*.

Fast Sample-Based Trajectory Optimization Figure 4.9 shows the performances of *iCEM* (yellow), CEM (purple), CEM_{MPC} (blue), and CEM_{PETS} (red) for different planning budgets. Again, the budget is defined as the total number of imagined trajectories per step. Intuitively, a higher planning budget means that the planner has access to more computational resources. Results are averaged over 50 independent runs with different seeds. Solid lines show means and the color bands around the solid lines indicate the standard deviations. Notice the log scale on the x-axis. In all the tasks, *iCEM* achieves the best results among all budgets. Even for the extremely low budget settings (around 10 trajectories per step), *iCEM* can solve most of the tasks where the other baselines fail.



(A) HalfCheetah Running. (B) Humanoid StandUp.

(C) Door. (D) Relocate.

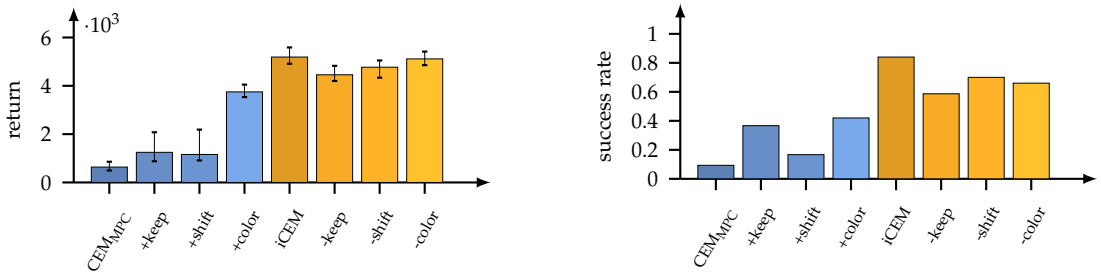
Figure 4.9: Performance of *iCEM*, CEM, CEM_{MPC}, and CEM_{PETS} relative to the planning budget for the (A) HalfCheetah Running, (B) Humanoid StandUp, (C) Door, and (D) Relocate environments. Notice the log-scale on the x-axis.

To quantify the improvements of *iCEM* over the baselines, Table 4.2 compares the performance of *iCEM* with the respective best baseline in each environment. A sample efficiency factor for *iCEM* is reported based on the approximate budget needed to reach 90% of the best baseline performance (at budget 4000). In conclusion, *iCEM* is $2.7 - 21.9\times$ more sample efficient than the baselines, which is a significant improvement. Similarly, it is evaluated how much the performance of *iCEM* improved w.r.t. the best baseline for a given budget (averaged over budgets < 1000). Again, the evaluation reveals that *iCEM* achieves $120 - 1030\%$ of the best baseline performance.

What is the reason for the considerable reduction in sample complexity and gain in performance of *iCEM* compared to the baselines? Figure 4.10 sheds light

on this question by adding individual components to CEM_{MPC} (blue bars) and removing individual components from *iCEM* (yellow bars) in HALFCHEETAH RUNNING and FETCH PICK&PLACE. Among the environments, adding colored noise has the biggest impact on the performance for reasons discussed in Sec. 4.2.1.1. In HALFCHEETAH RUNNING, for instance, running fast requires a fast and coordinated movement of the legs, while in FETCH PICK&PLACE smooth and temporally extended motions typically lead to higher successes. In FETCH PICK&PLACE, keeping the previous elites also has a noticeable impact on the performance. This is due to the partially sparse reward setting; thus, the small likelihood of finding a solution with a non-constant cost signal as discussed in Sec. 4.2.1.2. Once such a solution is found, keeping it over multiple planning steps highly increases the likelihood of success.

While adding individual components to CEM_{MPC} can have a significant impact on the planner's performance, removing individual parts from *iCEM* (yellow bars) does not have the same magnitude of impact. This indicates that not one single component of *iCEM* is responsible for its overall performance but that the interplay between all the additions to CEM_{MPC} is essential to achieve the highest performance.



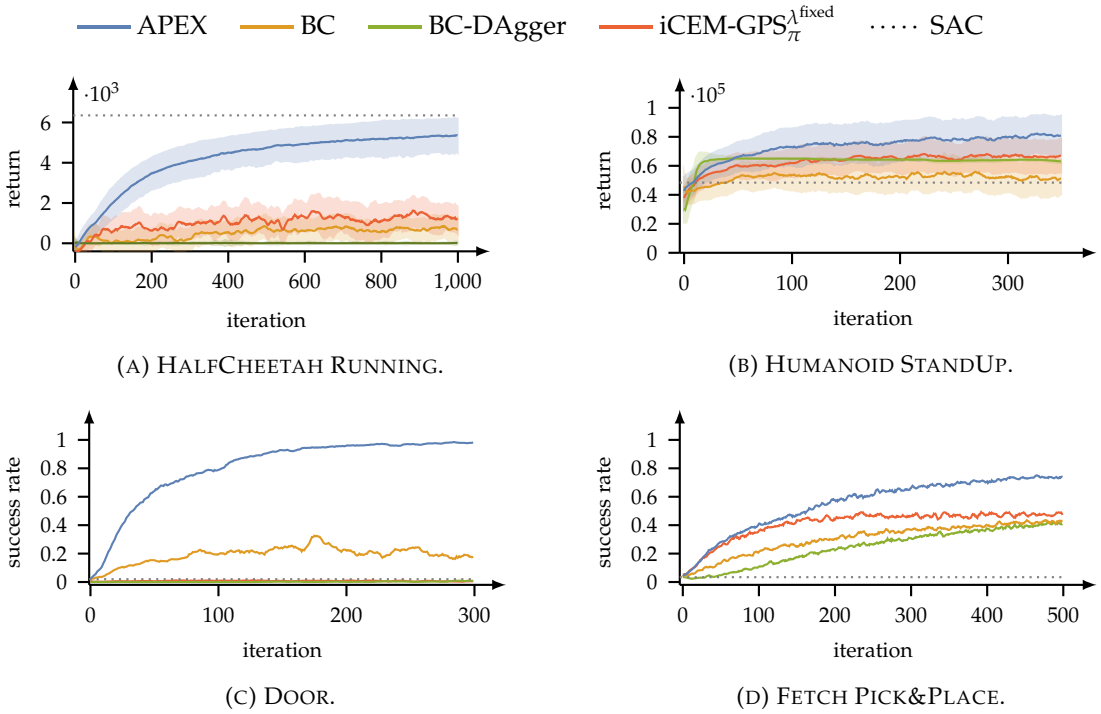
(A) HALFCHEETAH RUNNING. (B) FETCH PICK&PLACE.

FIGURE 4.10: Ablation studies of *iCEM* for (A) HALFCHEETAH RUNNING and (B) FETCH PICK&PLACE. Blue bars show CEM_{MPC} with each improvement added separately. Yellow bars show *iCEM* with each feature removed separately.

NN Policy Extraction This section presents the results of the IL-based NN policy extraction scheme *APEX*. The high-quality data produced by *iCEM* is used as reference trajectories for the BC loss. Additionally, *iCEM* is used inside DAgger to relabel suboptimal actions taken by the NN policy.

TABLE 4.2: Sample efficiency and performance increase of *iCEM* w.r.t. the best baseline. The first four columns consider the budget needed to reach 90% of the best baseline (dashed lines in Fig. 4.9). The last column shows the average improvement over the best baseline in the 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	0.86	45	985	21.9	100–1000	1030%
RELOCATE	0.88	95	1300	13.7	100–1000	413%



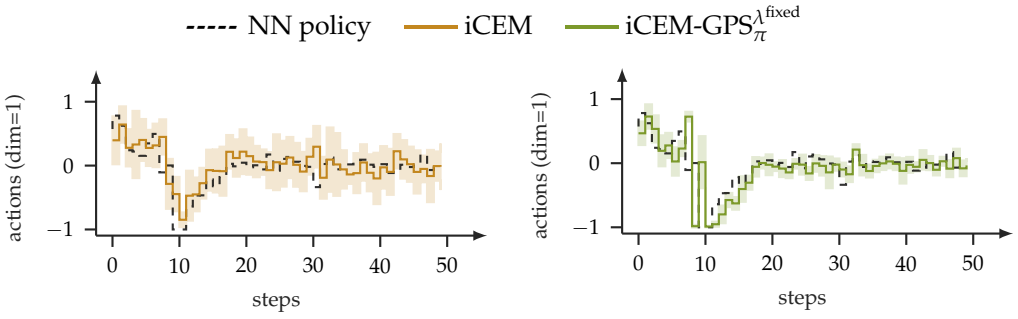
(A) HALFCHEETAH RUNNING. (B) HUMANOID STANDUP.

(C) DOOR. (D) FETCH PICK&PLACE.

FIGURE 4.11: Policy performance on the test environments for APEX and baselines. SAC performance is provided for reference.

Figure 4.11 shows the performance of the NN policy learned with *APEX* (blue) as well as the performance of the NN policies learned with the vanilla BC loss (yellow), BC-DAgger (green), iCEM-GPS$_\pi^{\lambda^{\text{fixed}}}$ (red), and the asymptotic performance of the model-free and off-policy RL baseline SAC (dotted horizontal line). Results are averaged over 10 independent runs with different seeds. In HALFCHEETAH RUNNING, SAC achieves the best asymptotic performance, closely followed by *APEX*. BC and BC-DAgger fail completely to learn a good policy. iCEM-GPS$_\pi^{\lambda^{\text{fixed}}}$ achieves slightly better performance than BC-DAgger but is also not able to match the performance of SAC or *APEX*. Given these results, it can be concluded that adding DAgger or GPS alone to BC does not solve the policy extraction problem to full satisfaction. In HUMANOID STANDUP, SAC only finds the sub-optimal solution of sitting, corresponding to a reward of around 50000. While all other methods achieve better performance than SAC, only the policy learned with *APEX* can stand up. However, even the policy learned with *APEX* fails to balance in an upright position for longer time periods because of the many ways the humanoid robot can fall. In DOOR and FETCH PICK&PLACE, SAC does not learn any reasonable policy. Only *APEX* can learn acceptable NN policies in these environments.

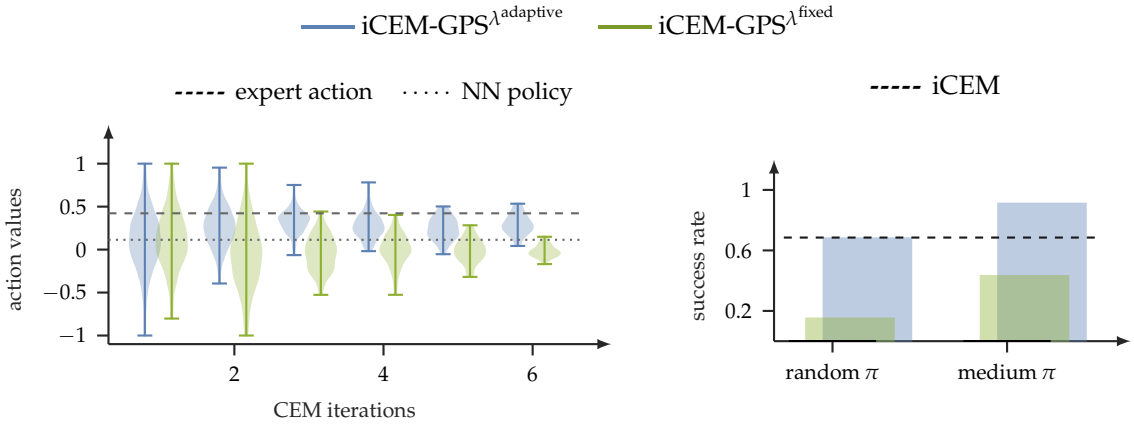
Figure 4.12 and Fig. 4.13 provide insights into why the NN policies learned with *APEX* show much better performances then the ones learned by the baselines. Figure 4.12 compares the variance of the actions produced by multiple runs of DAgger with *iCEM* and iCEM-GPS$_\pi^{\lambda^{\text{fixed}}}$ as teachers on a single policy rollout. Because of the inherently stochastic nature of the unconstrained *iCEM*, the distribution of relabeled actions is very broad (Fig. 4.12A). In comparison, the additional GPS cost in iCEM-GPS$_\pi^{\lambda^{\text{fixed}}}$ leads to solutions of the planner that concentrate more around the solution of the NN policy (Fig. 4.12B). In the case of the unconstrained *iCEM*, this makes learning a good performing NN policy very hard because the BC loss learns an average action from the data. If the data is conflicting, this might lead to catastrophic



(A) iCEM. (B) iCEM-GPS$_\pi^{\lambda\text{fixed}}$.

FIGURE 4.12: Variance of DAgger action relabeling after relabeling the same trajectory 10 times in case of (A) *iCEM* and (B) iCEM-GPS$_\pi^{\lambda\text{fixed}}$ in FETCH PICK&PLACE.

failure. For instance, in the example depicted in Fig. 4.4, an average policy learned from both types of solutions (moving around the obstacle via the upper or lower path) might move directly into the obstacle. Constraining the solution set produced by the planning method to stay close to the solution of the NN policy helps stabilize the training of the policy and minimizes the likelihood of conflicting reference data. However, constraining the planner too much might lead to suboptimal solutions as well if the NN policy is too far away from the optimal solution. Figure 4.13 shows the difference between an adaptive and a fixed λ in the GPS cost for the FETCH PICK&PLACE environment. With a fixed λ, the action variance of iCEM-GPS$_\pi^{\lambda\text{fixed}}$ shrinks too fast and concentrates around the actions proposed by the suboptimal NN policy shown as the dotted horizontal line. In FETCH PICK&PLACE, the optimal solution corresponds to moving the gripper closer to the cube's location. Suppose the NN policy is too far away from this solution and the planner is too constrained. In that case, the planner will not even get close to the cube's position resulting in a completely flat, thus uninformative, cost signal because of the partially sparse reward/cost that depends only on the distance between the cube and target but not on the distance between the gripper and the cube. With an adaptive λ, however,



(A) Action sampling distribution. (B) Success rate.

FIGURE 4.13: (A) When guiding with a weak policy, the action sampling distribution for *iCEM*-GPS with fixed and adaptive λ over multiple CEM-iterations (at predefined time steps). The dashed line shows the action of an expert policy. The dotted line shows the action of a suboptimal policy. (B) The effect of an adaptive λ on the success rate in the FETCH PICK&PLACE environment.

iCEM-GPS$_\pi^{\lambda^{\text{adaptive}}}$ keeps a larger action variance; thus, it is flexible enough to find the optimal actions shown as the broken horizontal line. The performance plot shown in Fig. 4.13B for *iCEM*-GPS with adaptive and fixed λ in FETCH PICK&PLACE confirms the benefit of an adaptive λ in the GPS cost. In the case of a medium NN policy iCEM-GPS$_\pi^{\lambda^{\text{adaptive}}}$ outperforms the pure *iCEM* expert.

Finally, the feedback loop between the NN policy and the *iCEM* planner inside *APEX* needs to be discussed. Figure 4.14 shows the performance of *APEX*'s NN policy (solid blue line) next to the performance of *APEX*'s *iCEM* (broken blue line) throughout training in the HALFCHEETAH RUNNING environment. At the beginning of training, *APEX*'s *iCEM* achieves the same performance as the purely planning-based *iCEM*. This is due to the adaptive λ in the GPS cost that prevents the planner inside *APEX* from collapsing to the suboptimal solution proposed by the NN policy. During training, the NN policy inside *APEX* becomes more capable, also boosting the planner's performance far above the performance of the purely planning-based *iCEM*. In HALFCHEETAH RUNNING, vanilla *iCEM* is not able to beat the performance of SAC, while the NN policy boosted *iCEM* inside *APEX* achieves even higher performance than SAC.

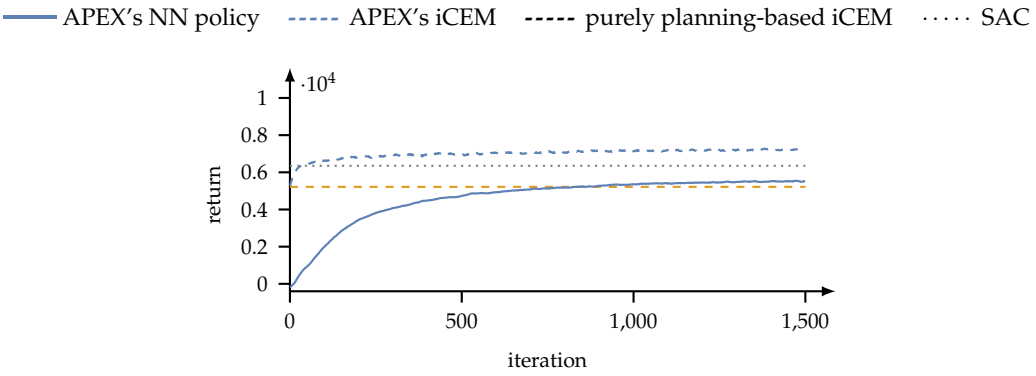


FIGURE 4.14: Interplay between *APEX*'s NN policy and *APEX*'s *iCEM* throughout training in the HALFCHEETAH RUNNING environment.

4.6 Discussion

This chapter presented *iCEM* and *APEX*. *iCEM* aims to close the gap between sample-based global trajectory optimization and real-time robotic control. *iCEM* is based on CEM for MPC. CEM_{MPC} showed to produce compelling solutions to the trajectory optimization problem in simulated environments (Chua et al., 2018; Hafner et al., 2019). Applying sample-based planning methods like CEM in real robotic control tasks is challenging, however, because all the heavy computation is done during the deployment. In particular, if the planner is used inside an *MPC* policy to close the action-perception loop. Executed in closed loop, the CEM optimizer must compute a new plan after each step in the environment to incorporate all the new information from the sensorimotor stream. *iCEM* extends CEM_{MPC} in several ways to decrease the sample complexity in each planning step significantly. (1) From (linear) system identification it is known that (linear and time-invariant) dynamical systems have characteristic FRFs and corresponding transfer functions that are used to efficiently explore the operation space and estimate the parameters of the (linear) system (Sinha, 1989). *iCEM* leverages this knowledge about the frequency-specific response of dynamical systems by replacing the Gaussian action sampling distribution inside CEM with a power-law distribution. While action sequences sampled from a Gaussian distribution have a uniform PSD, lower frequencies in the PSD of action sequences sampled from a power-law distribution are amplified. In comparison, higher frequencies are repressed depending on the exponent of the power-law distribution. In Fig. 4.2 of Sec. 4.2 it was shown that the state-space coverage of a point mass depends on the type of power-low distributed exploration noise. Moreover, it was shown that successful trajectories in HUMANOID STANDUP result from power-law distributed action sequences with an exponent between 2 and 4, while Gaussian distributed ($\beta = 0$) action sequences do not lead to success. Consequently, the experimental results presented in Fig. 4.10 revealed that adding colored noise to CEM_{MPC} leads to the most significant boost in the performance of the optimizer. (2) *iCEM* makes use of the observation that the imagined plans computed in one time step are still mostly valid in the next step because deviations between the imagined and actual observed outcomes are typically small. Moreover, the solution set produced by the planner rarely collapses to a singular solution but covers a small neighborhood around the best solution. To this end, *iCEM* keeps a memory of previously computed plans and reuses them in the next time step. The experimental results shown in Fig. 4.10 suggest that *iCEM* benefits from its memory, especially in environments where the exploration problem is challenging and the likelihood of finding any viable solution is very low. Once such a rare solution is found, the memory allows *iCEM* to remember this solution over multiple time steps; thus, increasing the likelihood of overall success dramatically. All in all, the additions to CEM_{MPC} presented in this chapter make *iCEM* $2.7 - 21.9$ times more sample efficient than the best next baseline and boost the performance by $120 - 1030\%$. That means that *iCEM* can achieve the same performance as the baselines by using much fewer samples, reducing the computational complexity per step significantly.

Table 4.3 shows the runtimes of *iCEM* for different budgets and environments. In the simplest HALFCHEETAH RUNNING environment, *iCEM* already reaches close to real-time performance while using a computationally expensive CPU-based simulator as a model. With highly parallelizable GPU-based simulators like ISAAC GYM or learned NN-based models, the runtime can be reduced much further, pushing *iCEM* even further toward the realm of real-time control (Pinneri et al., 2020).

Suppose a planning method cannot be used directly for control, for instance,

TABLE 4.3: Runtimes for *iCEM* with different compute budgets. Times are given in seconds per env-step (total wall-clock time = time/step × episode length) on a Xeon® Gold 6154 CPU @ 3.00GHz.

		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	

because of limited onboard computational resources. In that case, another way of utilizing the strengths of *iCEM* for (real-time) control is by using IL techniques to extract an NN control policy from the near-optimal trajectories produced by *iCEM*. NN policies have advantages over planning methods. (1) All the computational complexity is moved to the training phase while querying the policy for actions during runtime is computationally cheap. (2) NN policies can be queried on any arbitrary state, even those not seen during training. Yet, no optimality or safety guarantees can be given outside the data distribution. However, learning an NN policy from a stochastic teacher can be challenging and lead to suboptimal results if done naively, as shown for several BC-based IL baselines in Fig. 4.11. In *APEX*, the *iCEM*-MPC policy is used as a teacher to train an NN student policy. The NN policy is trained with a BC loss. To mitigate a potential covariate shift due to different state-visitation distributions between the teacher and the student policy, DAgger is used to relabel the suboptimal actions taken by the student with actions from the teacher. In addition, GPS is used to constrain the solution set of the teacher to be close to the student's solution. As shown by Fig. 4.12, this can help provide more consistent training examples for the BC loss, especially if the teacher is stochastic and does not converge to a single best solution. If applied naively, GPS can result in a suboptimal teacher performance as shown by Fig. 4.13. This effect is especially prevalent early on in training, where the solution found by the student is far away from the optimal solution. With an adaptive λ in the GPS cost, this effect can be attenuated by making the influence of the student on the teacher dependent on the performance of the student.

Despite all these improvements, BC-based IL techniques have some drawbacks. For instance, they are upper bounded by the teacher's performance and cannot be trained from mixed behaviors. Offline-RL is a different type of IL technique that gained traction recently (Kumar et al., 2020; Nair, Dalal, et al., 2020; Z. Wang et al., 2020; Kostrikov et al., 2021). In offline-RL, an RL agent is trained on the sub-MDP induced by a fixed dataset of agent-environment interactions. Offline-RL has several advantages over BC. (1) It can deal with datasets produced by a mixture of behavioral policies. (2) In principle, it can surpass the performance of the behavioral policies that generated the dataset by learning a policy in the sub-MDP that is more optimal. (3) It can learn different and new tasks from the same data using reward relabeling. When this work was published, the field of offline-RL was not mature enough to be used out-of-the-box in *APEX*. In the future, it would be an exciting direction to replace the BC-based IL part in *APEX* with methods from offline-RL.

5

Uncertainty-Aware Planning in Model-Based Reinforcement Learning

This chapter is based on:

Vlastelica*, Blaes*, Pinneri, Martius (2021) "Risk-Averse Zero-Order Trajectory Optimization". In: *Conference on Robot Learning (CoRL)*. *Equal Contribution.

Contents

5.1 Introduction

The works summarized in the previous chapter discussed several improvements to the control side of Model-Based Reinforcement Learning (MBRL). Another major challenge in the MBRL domain is to design or learn accurate dynamics models. Especially if applied to real-world systems, special care is needed to account for the inevitably noisy dynamics of the environment. Noise can originate from unobserved variables, like external perturbations, or irreducible noise in the system, e.g., from unreliable sensor readings. To account for the possible uncertainties about the dynamics of a robotic system and the environment during planning, this work presents *Risk-Averse Zero-Order Trajectory Optimization Method (RAZER)*, a framework for uncertainty estimation and uncertainty-aware planning. *RAZER* consists of two components: (1) A learned model of the Markov Decision Process (MDP) that allows to estimate and disentangle between different types of uncertainties accurately. (2) An uncertainty-aware Model Predictive Control (MPC) planning policy that explicitly handles uncertainties in the optimization objective.

Previous works in the MBRL literature use learned models to estimate uncertainties in parametric (Chua et al., 2018) and non-parametric (M. P. Deisenroth et al., 2013; Kamthe et al., 2018) models. In these methods, uncertainties are merely used for sampling-based estimation of the expected task cost, while most of the information contained in the underlying noise distributions is unused. Also, these methods do not distinguish between the different types of uncertainties present in the learned models, namely aleatoric and epistemic uncertainties (Hora, 1996; Der Kiureghian et al., 2009). The aleatoric or statistical uncertainty stems from any inherent noise in the system under consideration. The aleatoric uncertainty cannot be further reduced regardless of how much more the system is observed. In the context of learned models, epistemic or systemic uncertainty stems from insufficient training data. This type of uncertainty can be reduced by collecting more training data; thus, reducing the model's prediction error to the system's noise level. These two types of uncertainties are well known in the model-free Reinforcement Learning (RL) literature (Mihatsch et al., 2002; Garcıa et al., 2015) and control literature (Arruda et al., 2017; Abraham et al., 2020) but are not well explored in the context of MBRL.

This work presents *RAZER*, a planning-based RL agent that utilizes learned ensembles of probabilistic Neural Networks (NNs) to model the dynamics of the MDP. A carefully designed model architecture allows *RAZER* to differentiate between the aleatoric and the epistemic uncertainties. An MPC policy inside *RAZER* uses these uncertainties explicitly in the optimization objective for uncertainty-aware planning. The performance of the uncertainty-aware planner *RAZER* is compared to a method that only estimates the expected cost via Monte-Carlo (MC) sampling in three domains: (1) In the active learning domain, the goal of the agent is to actively seek states for which the prediction accuracy of the learned model is suboptimal. In this setting, the epistemic uncertainty serves as a proxy measure for information gain (Pfaffelhuber, 1972) and is maximized as an intrinsic reward during planning, similar to how humans and other animals take actions that reduce the uncertainty about the environment (Belger et al., 2018). The epistemic uncertainty is measured in terms of the multi-step ensemble disagreement of the learned forward model. (2) In the uncertainty-aware planning domain, the agent's goal is to optimize the task cost using the learned model while minimizing the uncertainty about the future. Being able to assess the uncertainty about future outcomes of certain actions is an essential ability of living beings (Alhussein et al., 2021) that increases the chance of survival. Thus, it is crucial to equip robots with the same skill to minimize the risk of failure

and allow for safe human-robot interactions. In this setting, the agent uses the model acquired during active learning. (3) In the third domain, the agent's goal is to optimize the task cost while complying with external safety constraints and accounting for the environmental uncertainties. Being able to define explicit constraints in the optimization objective is very important in industrial environments. Furthermore, accounting for the uncertainty in the environment becomes vital in multi-agent environments where the robot has only partial information.

5.2 Method

The following sections present the three main contributions of *RAZER*: (1) The accurate estimation and separation of uncertainties in learned ensembles of probabilistic NNs. (2) An uncertainty-aware trajectory planning method with the explicit handling of aleatoric and epistemic uncertainties in the optimization objective. (3) The explicit handling of probabilistic safety constraints in sampling-based trajectory optimizers with learned models.

5.2.1 Preliminaries

In the following, the standard MDP formulation introduced in Sec. 2.1 of Ch. 2 is used. Moreover, the notation commonly found in the RL literature is adopted as discussed in Sec. 2.2.3.

This work is concerned with the accurate estimation and disentanglement of uncertainties in systems with noisy dynamics. Therefore, it is assumed that the time evolution of states $s \in \mathcal{S} \subseteq \mathbb{R}^{n_s}$ is governed by the following equation:

$$s_{t+1} = f(s_t,\ a_t,\ \eta(t)), \tag{5.1}$$

with $a \in \mathcal{A} \subseteq \mathbb{R}^{n_a}$ being a control input or action, $f\colon \mathcal{S} \times \mathcal{A} \times \mathbb{R}^{n_s} \to \mathcal{S}$ being the noisy system dynamics governing the transition from state s_t to s_{t+1} and

$$\eta(t) = \eta(s_t, a_t) \tag{5.2}$$

being a random variable modeling the noise in the system. No restrictions are put on the functional form of f nor η. For instance, f might be a non-linear function and η might be sampled from any arbitrary distribution and might enter the system dynamics in a non-linear way.

Consequently, without prior knowledge about the system dynamics f, a class of general function approximators, such as NNs, can be used that can be learned purely from data in an end-to-end fashion. Furthermore, since this work is interested in estimating and separating uncertainties in the system arising from the noise model η and the inherently erroneous approximations of the learned models, *RAZER* uses ensembles of probabilistic NNs to estimate and separate these uncertainties.

5.2.2 Ensemble of Probabilistic Neural Networks

RAZER learns a model $f_\theta\colon \mathcal{S} \times \mathcal{A} \to (\mathcal{S} \times \mathbb{R}^{n_s}_{>0})^K$, with parameters θ, that approximates the system dynamics in Eq. 5.1. In many cases, the model does not directly predict the true MDP state s but an observation $o = f^{\text{obs}}(s)$ produced by a perception model $f^{\text{obs}}\colon \mathcal{S} \to \mathcal{O}$. The following analysis remains valid independent of whether the model predicts states or observations.

Like Chua et al. (2018), *RAZER* uses an ensemble of NNs with stochastic outputs as model class for its forward model. Each ensemble member $f^k_{\theta_k} : \mathcal{S} \times \mathcal{A} \to \mathcal{S} \times \mathbb{R}^{n_s}_{>0}$, $k = 1, \ldots, K$, predicts the parameters:

$$f^k_{\theta_k}(s_{t-1},\ a_{t-1}) = (\mu^k_{\theta_k}(t),\ \Sigma^k_{\theta_k}(t)) \tag{5.3}$$

of a multivariate diagonal Gaussian distribution:

$$\hat{S}_t \sim \mathcal{N}(\mu^k_{\theta_k}(t),\ \Sigma^k_{\theta_k}(t)), \tag{5.4}$$

where θ_k are the parameters of the k-th model. The $\mu^k_{\theta_k} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$:

$$\mu^k_{\theta_k}(t) = \mu^k_{\theta_k}(s_{t-1},\ a_{t-1}) \tag{5.5}$$

are the output from the network head estimating the means and $\Sigma^k_{\theta_k} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^{n_s}_{>0}$:

$$\Sigma^k_{\theta_k}(t) = \Sigma^k_{\theta_k}(s_{t-1},\ a_{t-1}) \tag{5.6}$$

are the output of the network head estimating the diagonal covariance matrix $\Sigma^k_{\theta_k} = \sigma^k_{\theta_k} \cdot \mathbb{1}$ of the Gaussian distribution.

The ensemble is trained with the negative log-likelihood loss on the Gaussian outputs of the individual networks:

$$\begin{aligned}\mathcal{L}_\theta &= \sum_k \mathcal{L}^k_{\theta_k} \\ &= \sum_k \mathbb{E}_{(s_t,\ a_t,\ s_{t+1})\sim\mathcal{D}} \Big[-\log(\mathcal{N}(\mu^k_{\theta_k}(t),\ \Sigma^k_{\theta_k}(t))) \Big],\end{aligned} \tag{5.7}$$

with $\mathcal{D}$ being the dataset of state-action transitions.

Although the model is trained on individual transition tuples $(s_t,\ a_t,\ s_{t+1})$ sampled i.i.d. from the dataset $\mathcal{D}$, it is queried on consecutive steps in an auto-regressive fashion during planning. Consequently, any sequence of actions $(a_h)_{h=0}^{H-1}$ defines a predictive distribution over imagined trajectories $\tau = (s_t,\ a_0,\ \hat{S}_{t+1},\ a_1,\ \ldots)$:

$$\psi_\tau(s_t,\ a_{0,\ldots,\,h-1}) = p(\tau \mid s_t,\ a_{0,\ldots,\,h-1}), \tag{5.8}$$

starting from the ground-truth state s_t of the MDP.

Since probabilistic models $f^k_{\theta_k}$ are used for approximating the actual system dynamics in Eq. 5.1, the model predictions (or particles) $\hat{S}^k_{t+h}$ are distributed according to Eq. 5.4. Furthermore, the ensemble of probabilistic models induces an empirical distribution over the parameters $f^k_{\theta_k}(s_{t+h})) = (\mu_\theta(t+h),\ \Sigma_\theta(t+h))$ of the Gaussian distribution at a certain planning step h:

$$\psi_\vartheta(s_{t+h},\ a_h) = p(f_\theta(s_{t+h},\ a_h)). \tag{5.9}$$

This allows the ensemble model to approximate non-trivial and potentially multi-modal distributions of trajectories despite the assumption that the particles are Gaussian distributed.

5.2.3 Uncertainty Estimation with Ensembles of Probabilistic Neural Networks

Every action sequence $(a_h)_{h=0}^{H-1}$ with initial state s_t induces a distribution ψ_τ over trajectories. To efficiently sample from ψ_τ, the sampling procedure **T1** from Chua et al. (2018) is used: At each planning step h, K particles are sampled, one from each ensemble member $f^k_{\theta_k}$:

$$\begin{aligned}\hat{S}^k_{t+h} &\sim \mathcal{N}(\mu^k_{\theta_k}(t+h),\ \Sigma^k_{\theta_k}(t+h)) \\ &= \mathcal{N}(\mu^k_{\theta_k}(\hat{S}^{\pi(k)}_{t+h-1}, a_{t+h-1}), \\ &\qquad \Sigma^k_{\theta_k}(\hat{S}^{\pi(k)}_{t+h-1}, a_{t+h-1})).\end{aligned} \tag{5.10}$$

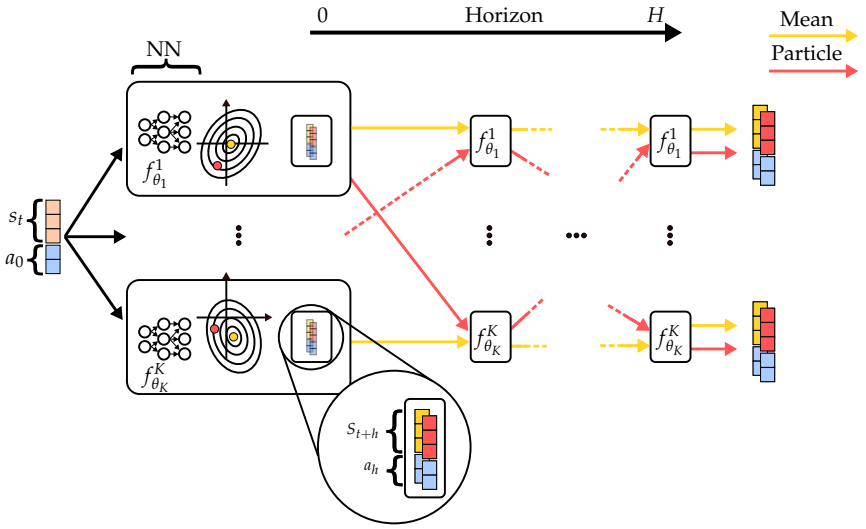


FIGURE 5.1: Probabilistic Ensembles with Trajectory Sampling and Uncertainty Separation (PETSUS). Each ensemble member $f^k_{\theta_k}$ predicts the mean μ^k and diagonal covariance matrix Σ^k of a Gaussian distribution. The networks are repeated along the planning horizon to predict H steps into the future in an auto-regressive fashion. The red pathways correspond to the sampling procedure **T1** proposed in Chua et al. (2018). The yellow pathways are added to disentangle aleatoric and epistemic uncertainties.

Afterward, the order of the particles is rearranged according to the permutation operator π, and the procedure repeats itself in an auto-regressive fashion until the end of the planning horizon H. In Eq. 5.10, $\hat{S}^{\pi(k)}_{t+h-1}$ is the particle from the $\pi(k)$-th model sampled in the previous planning step. For $h = 0$, the ground-truth MDP state $s(t)$ and the first action a_0 are fed into the models. The red pathways in Fig. 5.1 depict the sampling procedure.

In Chua et al. (2018), sampled trajectories are used to perform a Monte Carlo estimation of the expected trajectory cost $\mathbb{E}_{\tau\sim\psi_\tau}[c(\tau)]$. However, the sampling-based estimation does not consider the properties of the trajectory distribution ψ_τ. For instance, ψ_τ might be a high-entropy or heavy-tailed distribution. Sampling from such distributions with a limited number of samples may lead to overly risky or unsafe behavior. Moreover, computing the cost of a trajectory as the expectation over sampled trajectories makes it difficult to differentiate between aleatoric and epistemic uncertainties during planning, nor does it give explicit control over the degree of risk-averseness of the agent.

5.2.4 Separation of Uncertainties

RAZER alleviates these shortcomings by directly looking at the statistical properties of ψ_τ. This allows *RAZER* to differentiate between the epistemic uncertainty, denoted as $\mathfrak{E}$, and the aleatoric uncertainty, denoted as $\mathfrak{A}$.

Aleatoric Uncertainty One way of estimating the aleatoric uncertainty is to look at the ensemble disagreement of the forward model. This, however, is typically a bad estimate as it can be entangled with the epistemic uncertainty of the model. Therefore, RAZER measures aleatoric uncertainty in terms of the differential entropy of the particle sampling distributions defined by Eq. 5.4. More concretely, given the

k-th particle $\hat{S}_{t+h}$ at planning step h, the corresponding differential entropy is defined as:

$$\mathfrak{h}_h^k(\hat{S}_{t+h}) = \mathfrak{h}_h^k[-\log \mathcal{N}(\mu_{\theta_k}^k(t+h),\ \Sigma_{\theta_k}^k(t+h))]. \tag{5.11}$$

Equation 5.11 uses the definition of the differential entropy $\mathfrak{h}[f]$ of a continuous random variable X with probability density function $f(x)$ and support $\mathcal{X}$:

$$\mathfrak{h}[f] = \mathbb{E}[-\ln(f(x))] = -\int_{\mathcal{X}} f(x)\ln(f(x))dx. \tag{5.12}$$

The aleatoric uncertainty $\mathfrak{A}_h$ at planning step h is defined as the mean differential entropy of the particle sampling distributions:

$$\mathfrak{A}_h = \frac{1}{K}\sum_{k=1}^{K} \mathfrak{h}_h^k. \tag{5.13}$$

Since 1-step predictive Gaussian distributions are assumed, Eq. 5.13 is an expectation over differential Gaussian entropies, which can be computed in closed form (Cover, 1999).

An alternative approach for computing the aleatoric uncertainty is to look at the entropy of the distribution of network parameters θ_k induced by the ensemble. However, this is not entirely satisfying since large NNs tend to be over-parametrized. Therefore, the ensemble members might find vastly different solutions to the same optimization problem.

Intuitively, Eq. 5.14 says that the aleatoric uncertainty decreases the more the individual particle sampling distributions are peaked, corresponding to less noisy system dynamics. Conversely, the more noise gets injected into the system, the more "tailedness" the particle sampling distributions and higher the entropies are, resulting in a large aleatoric uncertainty.

Epistemic Uncertainty For estimating the model's epistemic uncertainty, *RAZER* extends the ensemble model from Chua et al. (2018) by an additional forward-path (yellow pathways in Fig. 5.1). In addition to the particle sampling, which involves permutations of the particles after each planning step (shown in red), a so-called "mean path" propagates only the mean predictions of the ensemble members along the planning horizon while keeping the order of the "mean particles" fixed. The resulting network architecture is called *Probabilitic Ensembles with Trajectory Sampling and Uncertainty Separation (PETSUS)*. The additional mean path in *PETSUS* allows *RAZER* to capture the statistical properties of the particle sampling distributions for each ensemble member independently. With this, the epistemic uncertainty $\mathfrak{E}$ is defined as the entropy of the mean particle sampling distribution parameters $f_{\theta_k}^k = (\mu_{\theta_k}^k(t),\ \Sigma_{\theta_k}^k(t))$:

$$\mathfrak{E}_h = \mathcal{H}[f_{\theta_k}^k(\mu_{\theta_k}^k(t+h-1),\ a_{t+h-1})]. \tag{5.14}$$

Notice that in the case of the mean path, the mean prediction $\mu_{\theta_k}^k(t+h-1)$ of the previous step is fed into $f_{\theta_k}^k$ instead of the previous particle $\hat{S}_{t+h-1}$.

Intuitively, Eq. 5.14 can be understood as follows: If every ensemble member faithfully captures the underlying noise distribution of the dynamical system, then the entropy of the distribution over the parameters of the sampling distributions goes to zero. On the contrary, if the moments of the different ensemble members do not match the moments of the noise distribution because of a lack of sufficient training data, the entropy will be non-zero.

An alternative approach for computing the epistemic uncertainty is to calculate the Fisher information metric $\mathcal{I} \coloneqq \mathrm{Var}[\nabla_\theta \log \mathcal{L}(s_{t+1}|s_t, a_t)]$ (Hüllermeier et al., 2021), where $\mathcal{L}$ denotes the likelihood function. However, this tends to be expensive to compute, especially for larger NNs.

5.2.5 Entropy vs. Variance as Uncertainty Measurement

The Gaussian distribution is the maximum entropy distribution for a given variance σ^2. Its entropy scales linearly with $\log(\sigma^2)$ (Cover, 1999). Therefore, the entropy of a Gaussian distribution and its variance can be used interchangeably to estimate the uncertainties. However, experiments showed that utilizing the variance directly causes *RAZER* to be much more risk-averse, which can be explained by the fact that the *log*-term inside the entropy squashes the variance. Moreover, variances are much easier to relate to because they have the same unit as the observations. Consequently, equations Eq. 5.13 and Eq. 5.14 can be formulated in terms of variances:

$$\mathfrak{A}_h = \frac{1}{K} \sum_{k=1}^{K} \Sigma_{\theta_k}^k(t+h) \tag{5.15}$$

and

$$\mathfrak{E}_h = \mathrm{Var}[\mu_{\theta_k}^k(t+h)] + \mathrm{Var}[\Sigma_{\theta_k}^k(t+h)]. \tag{5.16}$$

5.2.6 Probabilistic Safety Constraints

Safety is of utmost importance whenever data-driven control algorithms are applied to real systems. One option of introducing safety constraints in constrained optimization is by applying a constant penalty to constraint-violating imagined trajectories (A. E. Smith et al., 1997). Nevertheless, erroneous stochastic non-linear models often lead to non-trivial predictive distributions making assessing the risk of failure challenging using only a limited number of samples. For this reason, it would be favorable to control the risk of violating the safety constraints by considering the full predictive distribution.

Given a state space $\mathcal{S}$ with a constraint-violation region $\mathcal{C} \subseteq \mathcal{S}$, the probability of the action sequence $(a_h)_{h=0}^{H-1}$ entering the region $\mathcal{C}$ at planning step h is defined as:

$$p(\hat{S}_{t+h} \in \mathcal{C} \mid s_t,\ a_{0,\dots,h-1}) = \int_{\mathcal{C}} \psi_\tau(s \mid s_t,\ a_{0,\dots,h-1})\ \mathrm{d}s. \tag{5.17}$$

In practice, this integral is intractable due to the non-linear propagation of the uncertainties and the potentially non-trivial topology of the constraint-violation region $\mathcal{C}$.

To simplify the computation, only box violations are considered in this work. As a result, each dimension of s is constrained to be outside of the interval $[a, b] \in \{a,\ b \mid a,\ b \in \mathbb{R}^2,\ a < b\}$. Furthermore, $\hat{S}_{t+h}$ is estimated through moment-matching between a diagonal Gaussian distribution $\mathcal{N}(s;\ \hat{\mu},\ \hat{\sigma})$ and the empirical distribution induced by the particle population at planning step h. With this, the probability of particle $\hat{S}$ entering the constraint-violation set $\mathcal{C}$ is given by the integral:

$$p(\hat{S}_{t+h} \in \mathcal{C} \mid s_t,\ a_{0,\dots,h-1}) = \prod_{i=0}^{d} \int_{\mathcal{C}} \mathcal{N}(s_i;\ \hat{\mu}_i,\ \hat{\sigma}_i)\ \mathrm{d}s_i. \tag{5.18}$$

5.2.7 Planning and Control

RAZER uses the *improved Cross-Entropy Method (iCEM)* (see Ch. 4) to generate a finite number of action sequences $(a_h^n)_{h=0}^{H-1}$, $n = 1, \dots, N$, inside an MPC policy. In each time step t, the current ground-truth MDP state s_t and the first actions $a_0^{0,\dots,N}$ are fed through the ensemble resulting in $N \times K$ particles per step, with N being the number of independent action sequences and K being the number of ensemble members. This process is repeated in an auto-regressive fashion until the end of the planning horizon H.

Task Cost Like Chua et al. (2018), *RAZER* computes the task specific cost for an action sequence $(a_h^n)_{h=0}^{H-1}$ according to:

$$c^n(s_t,\ a^n_{0,\dots,\,H-1}) = c(s_t,\ a_0^n) + \sum_{h=1}^{H-1} \frac{1}{K} \sum_{k=1}^{K} c(\hat{S}^{k,n}_{t+h},\ a_h^n). \tag{5.19}$$

Already the expected task cost accounts for the uncertainty in the predictions to a certain degree; thus, encourages risk-averse behavior. For instance, if some of the particles $\hat{S}^{\cdot,n}$ for an action sequence n incur a high cost, the expected cost of the trajectory will increase. This makes it less likely for the action sequence to appear in the elite set. Consequently, the agent will avoid plans that lead to high expected costs and therefore are riskier. Nevertheless, using just the expected trajectory cost has several drawbacks: (1) It relies on a finite sampling size to estimate the expected cost. Especially for heavy-tailed distributions, this can lead to an overly optimistic behavior that might result in catastrophic failure. (2) It does not differentiate between aleatoric and epistemic uncertainties. (3) Since the expected task cost does not differentiate between the different types of uncertainties, they cannot be treated differently, nor can their effect on the agent's behavior be influenced.

RAZER solves these issues by modeling and separating the aleatoric and epistemic uncertainties explicitly in the optimization objective.

Aleatoric Cost With the definition of the aleatoric uncertainty given in Eq. 5.15, the corresponding auxiliary cost term is defined as:

$$c^n_{\mathfrak{A}}(s_t,\ a^n_{0,\dots,\,H-1}) = w_{\mathfrak{A}} \sum_{h=0}^{H-1} \sqrt{\mathfrak{A}^n_h}, \tag{5.20}$$

where $w_{\mathfrak{A}} \geq 0$ controls the risk-averseness of the agent.

Epistemic Cost Similarly, the definition of the epistemic uncertainty in Eq. 5.16 can be used to formulate an auxiliary cost term:

$$c^n_{\mathfrak{E}}(s_t,\ a^n_{0,\dots,\,H-1}) = -w_{\mathfrak{E}} \sum_{h=0}^{H-1} \sqrt{\mathfrak{E}^n_h}, \tag{5.21}$$

where $w_{\mathfrak{E}} \geq 1$ controls the agent's intrinsic motivation to maximize its information gain by exploring yet unexplored states.

Safety Cost Finally, *RAZER*'s capability to efficiently compute violations of probabilistic safety constraints can be used to formulate an auxiliary cost term:

$$c^n_{\mathcal{C}}(s_t,\ a_{0,\dots,\,H-1}) = w_{\mathcal{C}} \sum_{h=0}^{H-1} [\![p(\hat{S}^n_{t+h} \in \mathcal{C}) > \delta]\!] \tag{5.22}$$

in which $w_{\mathcal{C}} \geq$ controls how strongly safety violation are penalized and $\delta \geq 0$ is a threshold on the confidence that the constraint is violated. The $[\![\cdot]\!]$ in Eq. 5.22 is called the Iverson bracket and is 1 if the argument is true and 0 otherwise. An alternative way of implementing safety constraints in sampling-based planners is by changing the ranking function as done in Wen et al. (2018).

Equation 5.20 and 5.21 are computed in the state space while Eq. 5.19 is computed in the task-specific cost space. This makes it difficult to weigh the different terms against each other in the total cost. Alternatively, Eq. 5.20 and Eq. 5.21 can be computed in the task cost space as well. However, the task cost space formulation has several drawbacks: (1) One advantage of MBRL is that the learned models are task agnostic; thus, they can be used to solve a wide variety of tasks. If the epistemic cost is computed in the task cost space, the resulting exploration is coupled to a specific task, and the transferability of the learned model might suffer. (2) On the one hand, computing the aleatoric cost in the task-specific cost space seems reasonable because a failure typically results in a high task cost, for instance, because a human designer adds prior knowledge about failure cases to the cost function. On the other hand, it can be argued that high uncertainty in the state space typically results in control difficulties which should be avoided independent of the task at hand. Additionally, treating aleatoric uncertainty in a task-independent way can reduce the complexity of designing cost functions. Because of these reasons, the state-space formulation is used in all of the presented experiments.

5.3 Environments

The following sections introduce the different environments studied in the experimental section. What differentiates these environments from all the environments studied so far is that their system dynamics is noisy. All environments are simulated with the *MuJoCo* physics engine (Todorov et al., 2012).

THREE BRIDGES This toy environment is specifically designed for showcasing the different qualities of the epistemic and aleatoric uncertainties in planning. The agent controls a 2 Degree of Freedom (DoF) point mass in three dimensional space. The point mass is subject to the laws of motion. With its continuous actions, the agent applies forces in the x- and y-direction to the robot:

$$a = (F_x,\ F_y) \in \mathbb{R}^2. \tag{5.23}$$

The 10 dimensional continuous state vector:

$$s = (x,\ y,\ z,\ a,\ b,\ c,\ d,\ \dot{x},\ \dot{y},\ \dot{z}) \tag{5.24}$$

contains the 3 positional (x to z), 4 quaternion- (a to d) and 3 velocity-based ($\dot{x}$ to $\dot{z}$) agent-centric coordinates.

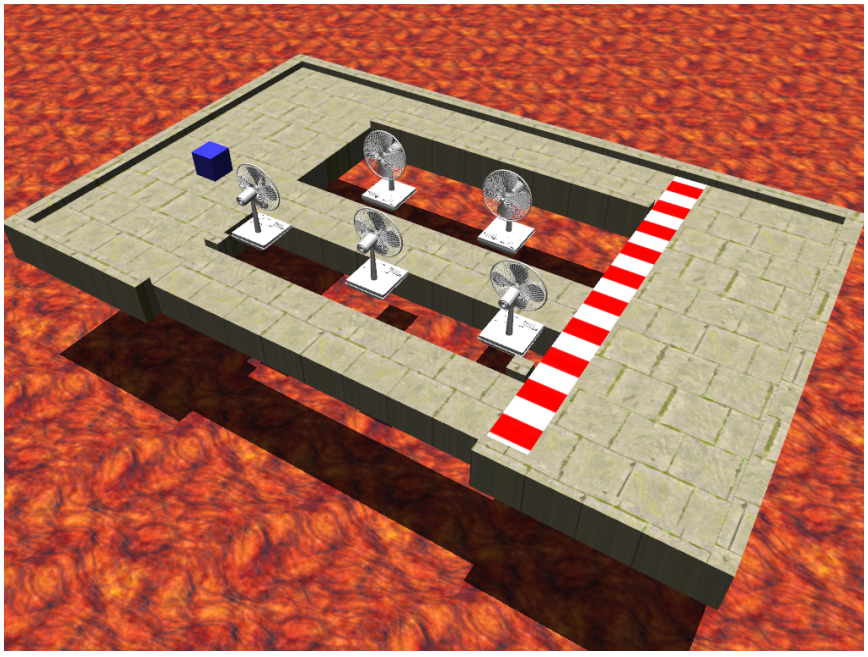


FIGURE 5.2: In the THREE BRIDGES environment, the agent controls a point mass. The point mass is spawned on the left platform and must reach the right platform by crossing one of three bridges.

As shown in Fig. 5.2, THREE BRIDGES consists of the agent controlled point mass (visualized as blue cube), two platforms and three bridges that connect the platforms. Below the platforms and bridges is lava that immediately destroys the robot as soon as it comes in contact with the lava. Walls around the outer edges of the landmass (except for the south edge of the lower bridge and all the edges of the landmass that are inside of it) prevent the robot from falling into the lava.

The agent's task is to steer the point mass from its starting platform on the left to the goal platform on the right by crossing one of the three bridges without falling

into the lava. More precisely, the domain reward is defined as

$$r(s_t, a_t, s_{t+1}) = \begin{cases} -|x_t - x^\star| & \text{if } z_{t+1} \geq -1.5, \\ 0 & \text{if } x_{t+1} \geq x^\star, \\ & \text{and } z_{t+1} \geq -1.5, \\ -1 & \text{otherwise,} \end{cases} \tag{5.25}$$

where $x^\star$ is the x-coordinate of the finish line. The task-specific cost is defined as the negative reward:

$$c(s_t,\ a_t,\ s_{t+1}) = -r(s_t,\ a_t,\ s_{t+1}). \tag{5.26}$$

Intuitively, the agent receives a reward equal to the negative distance to the finish line in every step. The reward is zero as long as the agent is right from the finish line. If the agent steers the point mass into the lava ($z < -1.5$), the agent receives a reward of -1 in every step until the end of the trial.

The noise in THREE BRIDGES is modeled in the form of random external forces (visualized by the fans in the environment) sampled every 5 steps anew from $F^{\text{ext}} \in \mathcal{U}(0,\ F^{\text{ext}}_{\text{max}})$. They are applied to the point mass whenever it is on the middle bridge. Otherwise, the system is entirely deterministic. Since the external forces are strong enough to push the point mass from the central bridge, this is the most dangerous bridge to cross. However, going over the middle bridge is the shortest path from the starting point to the finish, resulting in a minimum task-specific cost. The route over the lower bridge is longer than the path over the middle bridge but safer since the movement of the point mass is entirely deterministic. The upper bridge is the safest because it is broader than the two other bridges, it has a guard on the north edge of the bridge, and the movement of the point mass is fully deterministic. However, the path over the upper bridge is the longest.

NOISY HALFCHEETAH This environment adds simulator state noise to the OPENAI GYM environment HALFCHEETAH. In HALFCHEETAH, the agent controls a 6 DoF cheetah and the goal is to maximize the velocity along the x-axis without flipping over. The observation space in HALFCHEETAH is 18 dimensional and includes the absolute position of the cheetah, the joint angles, as well as the joint angular velocities. The cheetah can only move in the xz-plane. The simulator-state noise gets sampled from a Normal distribution:

$$\eta(t) \sim \mathcal{N}(\mu,\ \Sigma), \tag{5.27}$$

with $\mu = [0, \cdots, 0]^T$ and diagonal covariance matrix $\Sigma = 0.2 \cdot \mathbb{1}$. The noise is added to the simulator state according to:

$$s'_t = s_t + \eta(t) \cdot \mathbb{1}_{\dot{x}>6}, \tag{5.28}$$

with $\mathbb{1}_{(\cdot)}$ being the indicator function and $\dot{x}$ being the velocity in the x-direction. Consequently, simulator state noise is only added if the cheetah's velocity in the x-direction is greater than 6.

To evaluate external safety constraints, a virtual ceiling at height $z = 0.3$ is added to NOISY HALFCHEETAH. In the experiment, the agent has to maximize the task-specific reward while avoiding violations of the safety constraints.

NOISY FETCH PICK&PLACE This environment adds action noise to the OPENAI GYM environment FETCH PICK&PLACE. In FETCH PICK&PLACE the agent controls a 7 DoF fetch robot with the task of picking up a cube and bringing it to a target location either on the table or in the air. The $(25 + 3 + 3)$ dimensional observation space contains information about the fetch robot and the cube as well as their relative positioning. Two 3 dimensional vectors holding the achieved and the desired goal positions are appended to the observation vector. The 4 actions control the movement of the end-effector in Cartesian space and the opening and closing of the gripper. The reward signal is partially sparse in that it is computed from the distance between the cube and the target only and does not contain the distance between the end-effector and the cube. That means the reward signal is constant as long as the end-effector does not touch the cube. The Gaussian distributed action noise $\eta_{a_{\text{gripper}}}(t) \sim \mathcal{N}(\mu, \Sigma)$ is only applied to the action dimension controlling the opening and closing of the gripper. Furthermore, it is only applied if the end-effector's y-position is smaller than 1.67. Figure 5.3 shows the environment. The goal in the environment is to transport the cube from the left side of the table to a target location on the right side of the table that is in the air (red sphere in Fig. 5.3). The initial position and target position are fixed in this environment to keep the focus on the uncertainty-aware planning aspect. In particular, the cube's position is centered at $y = -1.5$ while the target is in the air at $y = 2.0$.

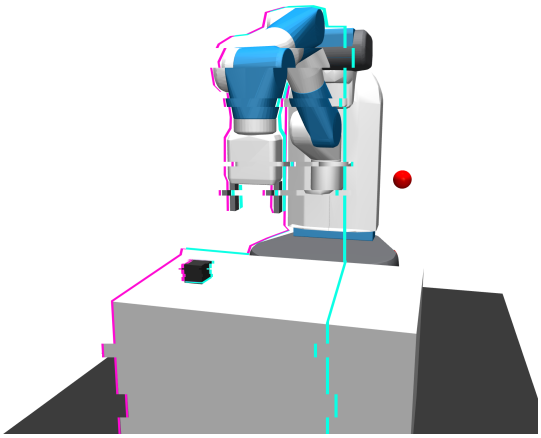


FIGURE 5.3: In NOISY FETCH PICK&PLACE, action noise is applied to the action dimension controlling the opening and closing of the gripper. The noise is applied if the gripper is on the right-hand side (from the robot's perspective) of the vertical cyan line. Otherwise, no action noise is applied. The task in the environment is to transport the cube from the right side of the table to an in-air target location (red sphere) on the left side of the table. The "glitch effect" indicates the region in which action noise is added to the action dimension that controls the opening and closing of the gripper.

Given the difficulty in learning the dynamics of the environment, we concentrate on the planning aspect in NOISY FETCH PICK&PLACE. To this end, experiments are performed with access to the ground-truth model which simulates a learned model. The same noise is applied to the real system and the "mental model". In that way, the mental model simulates a learned model with no epistemic uncertainty and an accurate estimation of the aleatoric uncertainty.

SOLO8 LEANOVEROBJECT In this environment, the agent controls a quadruped robot (Grimminger et al., 2020) and the goal is to track two targets (green markers)

with the purple markers at the front and rear ends of the robot's base (Fig. 5.4) without hitting the red volume in front of the robot. The robot starts in a laying position as shown in the inset of Fig. 5.4. Gaussian distributed action noise $\eta \sim \mathcal{N}(\mu,\ \Sigma)$, with zero mean and diagonal covariance matrix $\Sigma = 0.3 \cdot \mathbb{1}$, is applied to all action dimensions to mimic real-world disturbances. The state-space of the environment is 47 dimensional. It contains the absolute position, rotation, velocity, and angular velocity of the robot's base, 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 markers at the front and back of the robot. The action space is 8 dimensional and controls the relative positions of the joints. The two front legs of the robot are attached to the ground using soft constraints to prevent the robot from jumping uncontrollably, which would make the task much more challenging.

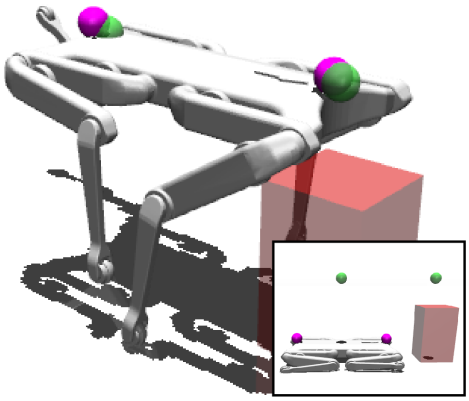


FIGURE 5.4: In SOLO8 LEANOVEROBJECT, a quadruped robot has to stand up from a ground position to track two targets (green spheres) with its front and rear end (purple spheres). The front feet of the robot are attached to the ground. Therefore, it has to lean slightly forward to decrease the tracking error. A fragile object (unsafe region, red cube) is in front of the robot.

As in the NOISY FETCH PICK&PLACE experiments, the ground-truth model is used to simulate a learned model. The same noise is applied in the mental model and the real system to simulate a learned model with no epistemic uncertainty and an accurate estimate of the aleatoric uncertainty.

5.4 Baselines

In the experiment discussed in the next section, the uncertainty-aware planner *RAZER* is compared against *Probabilistic Ensembles with Trajectory Sampling (PETS)* (Chua et al., 2018). *RAZER*'s architecture is an extension of the *PETS* architecture. Hence, both share the same base regarding model learning and sampling-based trajectory optimization. *RAZER* extends *PETS* in two aspects. First, the learned model in *RAZER* can separate and accurately estimate its forward predictions' aleatoric and epistemic uncertainties using the full information of the predictive distributions. *PETS*, on the other hand, relies on sample-based estimations of uncertainties and does not separate the aleatoric from the epistemic uncertainty explicitly. Second, *RAZER* explicitly considers the aleatoric and epistemic uncertainties in its optimization objective. This allows *RAZER* to use the epistemic uncertainty (Eq. 5.20 and Eq. 5.21) in an active learning context to improve the model predictions and the aleatoric uncertainty (Eq. 5.20 and Eq. 5.21) for risk-averse planning. Since *PETS* does not model the two uncertainties explicitly, uncertainties enter the planning only implicitly during the computation of the mean trajectory cost (Eq. 5.19). Furthermore, *RAZER* allows handling probabilistic safety constraints (Eq. 5.22) in a principled way (Eq. 5.2.6).

To make the comparison between *PETS* and *RAZER* fair, both use the same model architecture for the ensemble network, including the same hyperparameters. *PETS*, however, does not make use of the information from the mean particle pathway (yellow pathway in Fig. 5.1). For planning, both *PETS* and *RAZER* use the sampling-based trajectory optimizer *iCEM* with the same set of hyper-parameters.

PETS and *RAZER* are compared in three domains: (1) In the active learning setting (Settles, 2010), it is studied how the learned model can be actively improved by maximizing the epistemic uncertainty as a proxy for information to drive the agent's exploration. (2) In the risk-averse planning setting, the planners' capabilities to cope with noisy system dynamics is studied. (3) The third domain deals with planning under external safety constraints. In this domain, the *PETS* agent receives a high penalty cost whenever a particle violates the safety constraint, while *RAZER* optimizes Eq. 5.22.

5.5 Experimental Results

RAZER is studied in 4 continuous state- and action-space environments and compared against *PETS* in three domains: (1) Active learning, (2) risk-averse planning, and (3) planning under safety constraints.

5.5.1 Active Learning for Model Improvement

If model uncertainties are used for uncertainty-aware planning, they are only meaningful if the model has a good understanding and estimation of the uncertainty landscape. Uncertainty about the model predictions might stem from the noisy system dynamics or the model's erroneous predictions. Learning the parameters of the approximate noise model requires a sufficient amount of diverse data from the system under consideration. With too little or too similar data, the agent might avoid parts of the state space due to overestimating the model's aleatoric uncertainty. However, if the agent underestimates the aleatoric uncertainty, it might enter unsafe regions. By adding the epistemic bonus to the domain-specific cost, *RAZER* can actively seek states with high epistemic uncertainty, that is, states that maximize information gain because no or only little training data exists yet.

(A) RAZER. (B) PETS.

FIGURE 5.5: State-space coverage of (A) *RAZER* and (B) *PETS* during active exploration in THREE BRIDGES. States highlighted with a lighter color are visited earlier in the exploration, while a darker color indicates states visited later in the exploration.

Figure 5.6 shows qualitatively the state coverage of *RAZER* (Fig. 5.5A) and *PETS* (Fig. 5.5B) in THREE BRIDGES. The state coverage is computed by projecting the continuous states to the xy-plane and dividing them into 50 equally spaced bins in the range $-20 \leq x_0 \leq 20$ and $-10 \leq x_1 \leq 15$. To produce Fig. 5.5, the bins contain the first timestep (over the course of the entire training) in which the agent visited the state. If the agent visited the state earlier in the training, the respective state in Fig. 5.5 has a lighter color. States that are visited later have a darker color. Figure 5.6 shows the state coverage over time, computed as the fractions between states visited at least once and the total number of states. In each step, the agents receive the task reward defined by Eq. 5.25. *PETS* finds the solution of taking the lower bridge quite early in the exploration phase and fully commits to this particular solution without exploring any of the other options much further. *RAZER*, on the other hand, receives the epistemic uncertainty cost defined in Eq. 5.21 in addition to the task-specific cost. Hence, exploring states with a large ensemble disagreement is more incentivized. At the end of the active exploration phase, *RAZER* explored

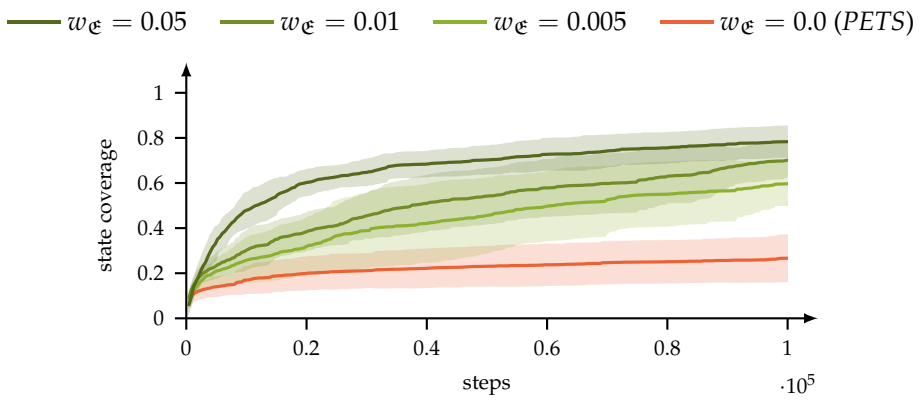


FIGURE 5.6: State-space coverage of *PETS* and *RAZER* for different values of $w_{\mathfrak{E}}$. A larger weight of the epistemic cost encourages *RAZER* to seek states that maximize information gain. Means and standard deviations are computed over 5 independent runs with different seeds.

most of the state space, including all three bridges, sufficiently well to better estimate the global uncertainty landscape. In contrast, *PETS* only explores the lower bridge extensively and therefore lacks sufficient training data to learn a globally accurate model of the uncertainty landscape in THREE BRIDGES.

Figure 5.6 shows quantitatively the state coverage of *PETS* and *RAZER* for different values of $w_{\mathfrak{E}}$. The quantitative result for the state coverage of *PETS* confirms the qualitative observation in Fig. 5.5B. After an initial exploration phase, the state coverage plateaus once the agent finds the solution of taking the lower bridge. *RAZER*, on the other hand, keeps exploring the environment until the end of the active exploration phase. The weight of the epistemic uncertainty cost, $w_{\mathfrak{E}}$, controls how much emphasis the agent puts on active exploration versus solving the actual task. With a large enough $w_{\mathfrak{E}}$, the agent even starts to extensively explore the dangerous middle bridge ignoring the task success altogether. While this leads to inferior task performance and might even be dangerous in the real world, it gives *RAZER* a better estimate of the global uncertainty landscape, which is important for risk-averse planning during deployment, for instance, on a real robot.

5.5.2 Uncertainty-Aware Model-Based Planning

The previous section verified that the epistemic uncertainty cost in *RAZER* indeed encourages the agent to actively seek novel states to understand the global uncertainty landscape better.

This section shows that a planner that explicitly handles aleatoric uncertainties (Eq. 5.13) in the cost (Eq. 5.20) is better at avoiding overly risky behavior compared to a planner that optimizes the mean task cost (Eq. 5.19) exclusively.

THREE BRIDGES This experiment is designed to study the uncertainty-aware planning capabilities of *RAZER* and *PETS*. The same learned model is used for planning to make the comparison between the two methods fair. To properly account for the uncertainties during planning, the planner needs to have a good estimate of the overall uncertainty landscape. To this end, the model learned with *RAZER* and $w_{\mathfrak{E}} = 0.05$ is used since this model is trained with the most diverse data covering most of the state space (see Fig. 5.6).

Figure 5.7 shows the success rate of *PETS* and *RAZER* in THREE BRIDGES. The random forces applied to the point mass on the middle bridge are tuned such that there is only a small but greater than zero chance for the agent to steer the point mass over that bridge without falling into the deadly lava. During exploration, *PETS* avoids the middle bridge altogether (see Fig. 5.5B) because its model overestimates the uncertainties along that path due to a lack of sufficient training data. With the accurate model of *RAZER*, though, *PETS* sometimes sees a chance to cross the middle bridge because of the finite sample size during the computation of the expected task cost per imagined trajectory. If the samples do not contain a trajectory that results in the agent's death, the agent might be tempted to cross the middle bridge as the task cost for this path is minimal. This can result in the agent falling into the lava or sometimes the agent might be lucky and succeeds. In the other case, the samples contain a deadly trajectory resulting in an increased task cost for the path over the middle bridge and the agent chooses to go over the upper or lower bridge. Consequently, *PETS* achieves only a success rate of ∼58%. *RAZER*, on the other hand, handles the aleatoric uncertainty explicitly in the optimization objective and does not rely on sampling to compute it. This has two consequences in THREE BRIDGES: (1) *RAZER* achieves generally higher success rates than *PETS* because *RAZER* can accurately estimate the high probability of failure taking the middle bridge and therefore favors the paths with the lower and upper bridges over the path going over the central bride. (2) Since the aleatoric uncertainty is explicitly modeled in the optimization objective, the risk-averseness of the *RAZER* agent can be varied. According to Fig. 5.7, $w_{\mathfrak{A}}$ has an optimum w.r.t. the success rate in the task at around 0.12 with a success rate of ∼96%. If $w_{\mathfrak{A}}$ is increased above this level, the agent sometimes becomes too afraid to move at all, resulting in a drop in the success rate. On the other hand, if $w_{\mathfrak{A}}$ is smaller, *RAZER* shows more risk-taking behavior and might try to reach the goal platform by taking the middle bridge in favor of a smaller task reward, leading to a lower success rate as well.

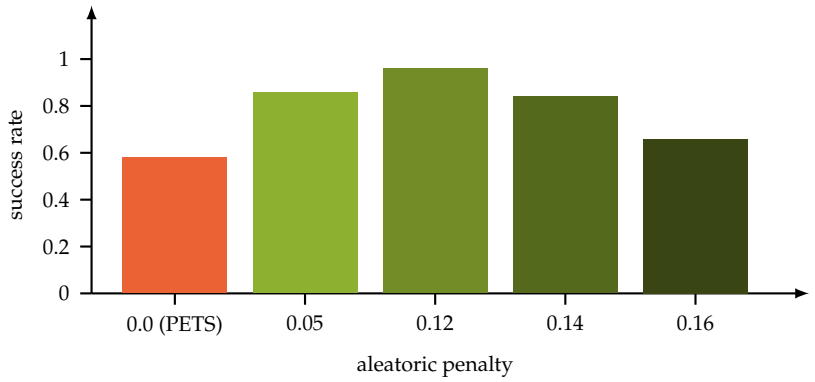


FIGURE 5.7: Sucess rates of *PETS* and *RAZER* in THREE BRIDGES. *RAZER* is evaluated with different values of $w_{\mathfrak{A}}$, resulting in varying levels of risk-averseness. In contrast, the risk-awareness of *PETS* cannot be controlled explicitly.

NOISY HALFCHEETAH Although in this experiment the agents have access to the ground-truth model of the environment (in the form of a "mental" simulation that has the same noise profile as the simulation used for evaluation), *PETS* consistently underestimates the uncertainty in the environment and tends to increase the instantaneous velocity above the point at which the noise sets in and destabilizes the cheetah. This results in a lower average velocity of the *PETS* agent (Fig. 5.8) as the noise

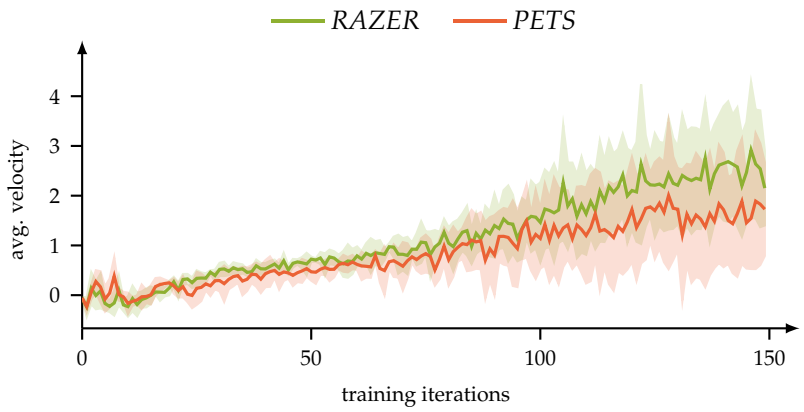


FIGURE 5.8: Average velocity of the *PETS* and *RAZER* agents in NOISY HALFCHEETAH.

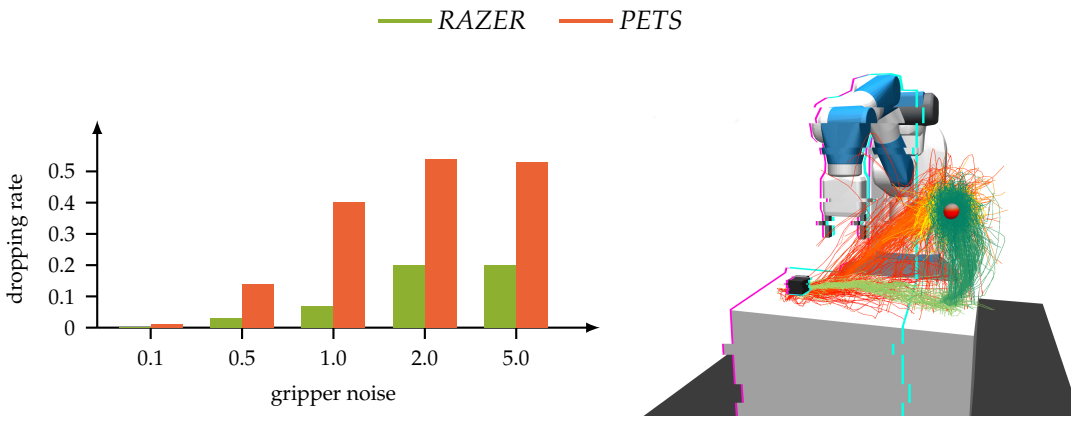
at higher velocities renders the solution found by the planner more and more sub-optimal. On the other hand, *RAZER*, with $w_{\mathfrak{E}} > 0$ and $w_{\mathfrak{A}} > 0$, manages to safely push its average velocity close to the boundary of the noisy region by avoiding overshooting of its instantaneous velocity. On average, this leads to a higher velocity of the *RAZER* agent compared to the *PETS* agent.

NOISY FETCH PICK&PLACE In this experiment, the agent controls a fetch robot and has to bring a cube from one side of the table to an in-air target position at the opposite side of the table. The shortest path from start to goal is in a straight line, meaning the agent has to lift the cube and simultaneously move it to the target location. However, with noise applied to the action controlling the gripper on the side of the table where the cube is spawned, there is a certain probability of dropping the cube along the way. Again, *PETS* optimizes only for the task cost and does not take into account the environment's uncertainty explicitly. This results in an overly optimistic behavior of the *PETS* agent that is close to the optimal solution for the deterministic case (red trajectories in Fig. 5.9B). *RAZER* optimizes for a low aleatoric uncertainty explicitly; therefore, it adopts a much more cautious behavior (green trajectories in Fig. 5.9B), that is, the agent slides the cube on the table and lifts it only in the region in which no action noise is applied. Figure 5.9A shows the dropping rates for *PETS* and *RAZER* for different levels of action noise. *RAZER* maintains a dropping rate lower than 20%, even when considerable noise is applied. The dropping rate of *PETS*, however, increases dramatically with higher action noise.

5.5.3 Planning under External Safety Constraints

This section discusses the experiments dealing with external safety constraints. The *RAZER* agent handles probabilistic safety constraints (Sec. 5.2.6) explicitly in the optimization objective via the auxiliary cost defined in Eq. 5.22. *PETS* does not have an explicit way of handling safety constraints; therefore, it is given a high penalty cost whenever it violates the constraint. In addition to the safety constraints, the agents optimize for the forward velocity of the cheetah.

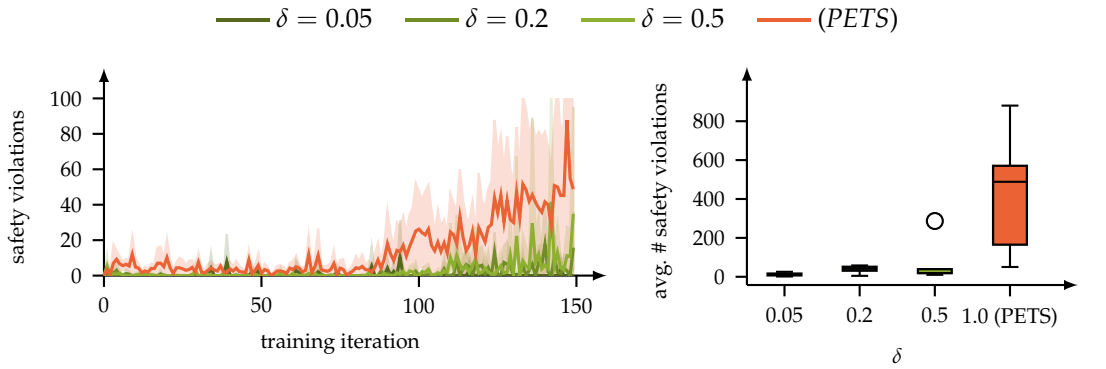
NOISY HALFCHEETAH In this experiment, a safety constraint is implemented as a maximum allowed height of the cheetah's body above the ground, simulating a narrow passage. Figure 5.10A shows the number of safety violations over time and Fig. 5.10B shows the average number of safety violation per step for *PETS* and



(A) Dropping rate (B) Trajectories

FIGURE 5.9: (A) Dropping rates of *PETS* and *RAZER* in NOISY FETCH PICK&PLACE. (B) Cube trajectories produced by *PETS* (red) and *RAZER* (green) in NOISY FETCH PICK&PLACE.

RAZER. With a higher forward velocity of the cheetah (from training iteration 90 onwards), *PETS* cannot avoid violating the safety constraints resulting in a sudden rise in the number of violations. *RAZER*, on the other hand, manages to keep the number of safety violations low. The threshold parameter, δ, allows adjusting the agent's confidence in respecting the constraint. Figure 5.10B shows that the agent is willing to risk more safety violations with a higher δ. A value of $\delta = 1$ corresponds to a sharp decision boundary as it is used here to implement safety constraints in *PETS*.

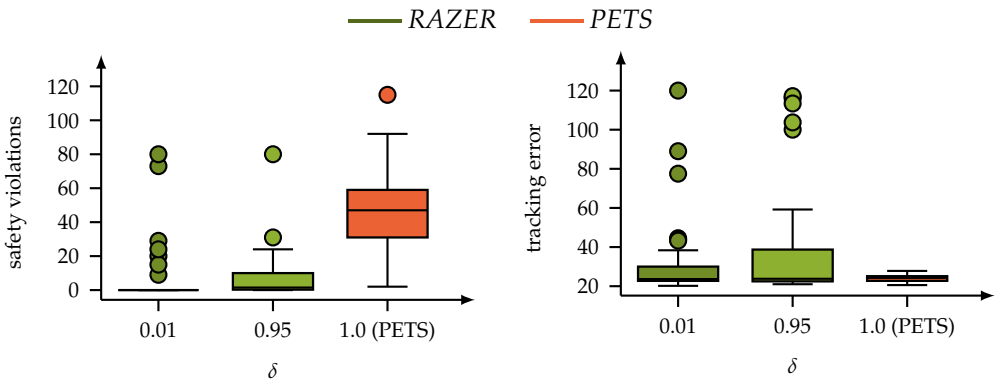


(A) Safety violations over iterations. (B) Average safety violations per step

FIGURE 5.10: (A) Safety violations over iterations and (B) the average number of safety violation per step of *PETS* and *RAZER* in NOISY HALFCHEETAH for different values of δ.

SOLO8 LEANOVEROBJECT In this experiment, the agent has to control the Solo8 robot while tracking two targets (green spheres in Fig. 5.4) with two markers attached to the front and rear ends of its base (purple spheres in Fig. 5.4). At the same time, the agent must avoid entering a rectangular volume in front of the robot (red volume Fig. 5.4). The front feet of the robot are attached to the floor such that the robot has to lean slightly forward to track the points accurately. However, because of the action noise in this environment, leaning too much forward can easily lead to the robot losing balance and falling into the restricted volume. Figure 5.11A shows

the number of safety violations per step for *PETS* and *RAZER* with different values of δ, while Fig. 5.11B shows the corresponding tracking error. Although a high safety violation penalty is added to the loss optimized by *PETS*, it still favors the task cost, that is a low tracking error, over avoiding safety violations. *RAZER*, on the other hand, co-optimizes for a small tracking error and a low count of safety violations in a probabilistic way, allowing it to drive down the number of safety violations while maintaining a reasonable tracking accuracy. However, due to the more delicate optimization objective, satisfying the safety constraint comes with the cost of an increased tracking error (Fig. 5.11B).



(A) Safety violations.

(B) Tracking error.

FIGURE 5.11: (A) Safety violation and (B) tracking error of the *PETS* and *RAZER* agents in SOLO8 LEANOVEROBJECT for different values of δ.

5.6 Discussion

This chapter presented *RAZER* and demonstrated that with its specific model architecture *PETSUS* it is possible to learn accurate probabilistic models of noisy system dynamics that can be used for uncertainty-aware task-oriented planning. A vital component of the introduced *PETSUS* model is that it can separate aleatoric from epistemic uncertainty. As argued earlier in this chapter and empirically evaluated in the experimental section, it is very important to make a clear distinction between the two uncertainties. The aleatoric uncertainty is inherent to the system which is to be modeled; thus, this uncertainty is irreducible. However, the epistemic uncertainty is a function of the model itself and either reflects a lack of enough training data or a too restricted model class. Hence, this type of uncertainty can in principle be reduced. This has important consequences for autonomous robotic agents. In one hand, a robot can use its knowledge about its epistemic uncertainty to improve its mental model. For instance, if the robot has to learn how to manipulate an object reliably, it can quickly start to concentrate on corner cases that need a lot of data to be predicted accurately instead of spending valuable time on easy-to-predict interactions, for instance, in which the robot has full control over the object. In fact, Sancaktar et al. (2022) showed in an unsupervised skill-discovery setting in which an agent maximizes the ensemble disagreement as a proxy for information gain that this is precisely the kind of behavior the agent adopts. An agent can freely play with several cubes in an FETCH PICK&PLACE like construction environment. What is observed in this environment is that the agent very quickly losses interest in lifting the cube after it discovers this behavior. This is because once the agent can lift a cube, it has full control over the object and there is no uncertainty about the future. Throwing the cube around, however, is very unpredictable; hence, the agent spends a lot of time trying to master that skill. On the other hand, knowing where and when the uncertainty stems from an inherently unpredictable system is equally important. Suppose a robot can solve a task or execute a motion in multiple ways. In that case, it should always take the option that minimizes the overall aleatoric uncertainty to maximize its controllability over the environment.

This work has demonstrated that an approach such as *PETS* 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 as purely sample-based approaches struggle to accurately estimate the risk of specific actions especially if the distribution of potential outcomes is heavy-tailed and the number of samples is very low.

Another important domain in which these problems need to be addressed is sim-to-real transfer. With the advent of highly parallelizable simulators like ISAAC GYM it becomes more and more common to train model-free RL agents in simulation to deploy them on real systems in a zero-shot fashion, that is, without further training in the new domain. To get this transfer to work, techniques like domain randomization are used to prevent the agent from overfitting to the simulated environment. However, it is not clear how to use domain randomization in the context of planning methods. Moreover, planning methods are very susceptible to exploiting the particularities of the environment. Accurately modeling uncertainties in simulation and grounding them in the real world might be one way of opening up the sim-to-real pipeline for planning-based policies. However, as of now, this is still an open challenge.

6

DISCUSSION

This thesis presented a collection of works studying nature-inspired inductive biases in model-free and model-based Reinforcement Learning (RL). With the theoretical analysis and empirical results presented, this work supports the long-lasting hypothesis that inductive or structural biases are necessary for sample efficient learning and learning generalizations. Yet, this argument is not supposed to be against data-driven and end-to-end learning approaches. Instead, the argumentation put forward here aligns with the view stated in "The Need for Biases in Learning Generalizations" (Mitchell, 1980):

"Although removing all biases from a generalization system may seem to be a desirable goal, in fact, the result is nearly useless. An unbiased learning system's ability to classify new instances is no better than if it simply stored all the training instances and performed a lookup when asked to classify a subsequent instance."

Likewise, natural intelligent agents like humans and other animals are not born as blank slates. Instead, nature equipped these agents with numerous inductive biases throughout evolution to help them learn more efficiently and survive in the natural world.

This chapter discusses the results presented in the previous chapters in the context of this hypothesis and highlights similarities between the learning behavior of natural and the presented artificial agents. The discussion is an attempt to encourage further studies of inductive biases in artificial agents, hoping to lead to more intelligent and autonomously behaving robots.

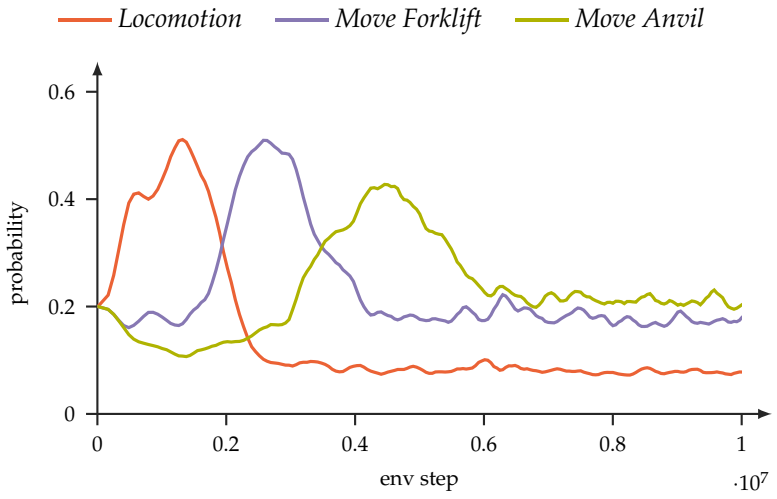


FIGURE 6.1: The learned task prioritization of the Control What You Can (CWYC) agent during the developmental phase in WAREHOUSE. For more information see Fig. 3.18 in Ch. 3.

Control What You Can Chapter 3 introduced CWYC, an RL agent that explores its environment and learns various skills in open-ended environments, that is, environments that do not provide any external goals during a developmental phase. In this phase, the agent is purely driven by the intention to maximize its controllability over the environment. Afterward, the learned skills or subroutines are used in an extrinsic phase for goal-directed behavior. To solve multi-stage object manipulation tasks, several hierarchically organized structured models are learned from data of agent-environment interactions and used by the CWYC agent to manage its learning efforts and plan on the subroutine level. The structured models implement different types of inductive biases.

Figure 6.1 shows the self-guided learning curriculum that is created during the developmental phase in WAREHOUSE. See Sec. 3.3 in Ch. 3 for more details about the environment. At the beginning of the training, the agent concentrates most of its learning efforts on the easiest task, that is, on *Locomotion* since the agent's actions directly control the robot's movement. Once the agent masters *Locomotion*, it switches its learning efforts to *Move Forklift* because the task does not have any prerequisites other than that the robot has to be able to reach the location of the forklift. On the contrary, the anvil can only be relocated with the help of the forklift. Consequently, the agent starts to learn *Move Anvil* only once it masters *Move Forklift*. A similar self-created learning curriculum can be observed in the statistical learning of infants (L. B. Smith et al., 2018). One early driver of this curriculum is the baby's growing sensory-motor competence. From the first intentional movements and reaching towards nearby items to crawling and the dexterous in-hand manipulation of objects, an infant acquires many new abilities over time that give it access to new objects to interact with and new sources of information (James et al., 2014; Ueno et al., 2018).

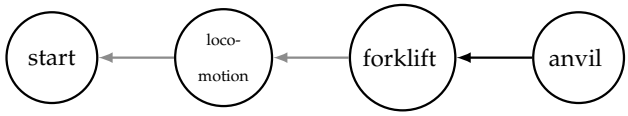


FIGURE 6.2: Learned task graph for *Move Anvil* in WAREHOUSE. For more information see Fig. 3.20 in Ch. 3.

In WAREHOUSE, *Move Anvil* is a rather complex task in which the agent has to go through multiple funnel states or stepping stones. First, the robot has to go to the forklift, then move the forklift to the anvil, and only then can it relocate the anvil to its target location. Instead of solving *Move Anvil* with a single control policy that might become very complicated, the CWYC agent has the ability to decompose the long-horizon task into multiple smaller subproblems that are easier to handle one at a time. The task-dependency graph that the agent learns for *Move Anvil* is shown in Fig. 6.2. The agent learns a simple control policy for each subtasks while the task-planner takes care of the planning at the subtask level and the suitable subtask composition. A similar decomposition of complex tasks into easier-to-manage subtasks can be observed in natural agents. For instance, Capuchin monkeys solve the task of cracking a nut by first carrying the nut to a large stone, then picking up a smaller stone to finally use the small stone as a hammer to crack the nut (de Resende et al., 2008).

Suppose a task requires multiple objects to be manipulated. In that case, it is usually not enough to know which objects must be manipulated but also in which relation the objects have to be with respect to each other. For instance, it is not the same in a tower building task whether two blocks are placed next to or on top of each other. In the example of the Capuchin monkeys, it matters whether the monkey smashes the small stone onto the big stone right next to the nut or right onto the nut itself. In WAREHOUSE, the CWYC agent learns that the relationship between the agent, forklift, and anvil matters for the success in *Move Anvil*. The agent can solve the final task only if the locations of all three entities coincide. Humans and other primates do not know about the importance of object relations right from birth (Marcinowski, Campbell, et al., 2016). For instance, newborns cannot reliably

build non-trivial constructions like towers (Marcinowski, Nelson, et al., 2019) or successfully play the game "shape sorter" in which they have to put all kinds of differently shaped objects through a similarly shaped hole. With training, however, this skill becomes quickly much more reliable (Hayashi and Matsuzawa, 2003) even if the task's difficulty is deliberately increased by altering the properties of the objects in a non-intuitive way (Hayashi and Takeshita, 2009). This might not be surprising since humans and other primates are very talented in utilizing tools.

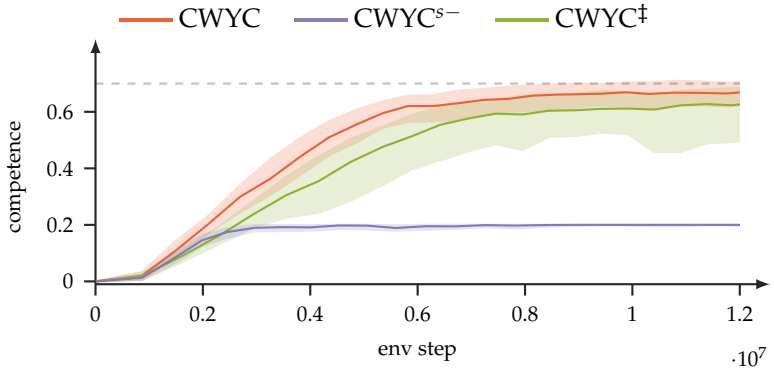
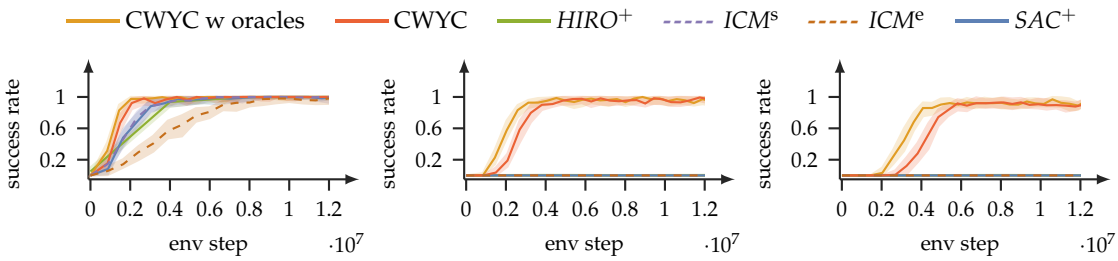


FIGURE 6.3: Overall competence of the CWYC, CWYC^{s-}, and CWYC‡ agents in the warehouse environment. For more details, see Fig. 3.6 in Ch. 3.

The different high-level components of the *CWYC* agent take some of the complexity out of the low-level control policies by capturing certain aspects of the planning problem. At the same time, they help to structure the agent's exploration by sampling goals and tasks instead of random unstructured noise in the action space. What stimulates the agent to explore its environment and learn new skills in the first place, however, is its intrinsic motivation to recreate surprising situations and maximize its learning progress. In Fig. 6.3, it is shown that removing these intrinsically motivated reward signals leads to a significantly reduced learning speed and even to a complete lack of any learning progress in the more complicated tasks. In Fig. 6.3, CWYC^{s-} ablates the surprising signal and CWYC‡ ablates the learning curriculum. Likewise, it is observed that infants seek events that violate their expectations about the world instead of events that are in accordance with their inner beliefs (Stahl et al., 2015). Studies in pedagogy indicate that students are more intrinsically motivated to study if they are in a learning-oriented environment rather than the student's self-perceived ability in a subject (Spinath et al., 2012). Similarly, the *CWYC* agent's interest in a task is not determined by its absolute competence but by how much progress the agent can make in learning the task.

Figure 6.4 shows that even a single control policy per task learned with Soft Actor Critic (SAC) (*SAC*$^{+}$) cannot solve the more complicated *Move Forklift* and *Move Anvil* tasks and that even a hierarchical abstraction (*HIRO*$^{+}$) or intrinsically motivated exploration based on surprise (*ICM*^{s}) or prediction error (*ICM*^{e}) is not enough to solve these tasks. Only the *CWYC* agent can solve these tasks by combining various inductive biases and Intrinsic Motivation (IM) in the different planning stages.

Already the works summarized in Table 6.1 utilize some of the inductive biases also found in the *CWYC* framework to build more autonomously exploring RL agents. What makes the *CWYC* unique is its capability to learn object relations and inter-task dependencies from just a few agent-object and object-object interactions (in the order of a few dozens). The *CWYC* does this by utilizing a particular network structure for the goal proposal network that can learn pairwise object relations very efficiently from data collected by the agent during the developmental phase. With



(A) *Locomotion*. (B) *Move Forklift*. (C) *Move Anvil*.

FIGURE 6.4: Success rate of the agents in (A) *Locomotion*, (B) *Move Forklift*, and (C) *Move Anvil* throughout training in the developmental phase. For more details, see Fig. 3.16 and Fig. 3.17 in Ch. 3.

this, the agent can propose new goals that maximize learning progress without relying on a database of previously visited states. Moreover, with Colas et al. (2019) and Röder et al. (2020), *CWYC* is one of the few methods that use curiosity in a Hierarchical Reinforcement Learning (HRL) setting. In *CWYC*, curiosity and learning progress are used to drive the exploration and a self-imposed learning curriculum that guides the learning of all the other components of the HRL agent.

There are also certain drawbacks of the proposed architecture: (1) Imposing biases on the architecture in the form of hand-designed structured models makes it difficult for the architecture to adapt to situations that were not considered by the human designer or to find other optimal solutions that are potentially less intuitive for humans. For example, Sancaktar et al. (2022) uses Graph Neural Networks (GNNs) to model agent-object and object-object relations and interactions, which in principle can deal with more complex situations because of a non-linear Neural Network (NN) that is used inside the GNN to model the relations and interactions between the nodes in the graph. (2) The disjoint nature of the individual modules increases the likelihood of individual points of failure. The other modules can hardly compensate for the failure of one module in the architecture as each module has its specific purpose. One solution to this problem can be to learn monolithic policies that solve the final tasks in one shot. This can be done by providing all the data collected in the developmental phase to the individual policies and using reward relabeling to train them to solve a specific task, e.g., relocating the anvil. In that way, the policies can compensate for a miscalibrated goal proposal network or subtask planner. Ideally, these components can even be removed altogether during the extrinsic phase. However, that also means that the individual policies become much more complex, making them potentially more challenging to train. In addition, such monolithic policies do not have the same compositional power as the more fine-grained skill

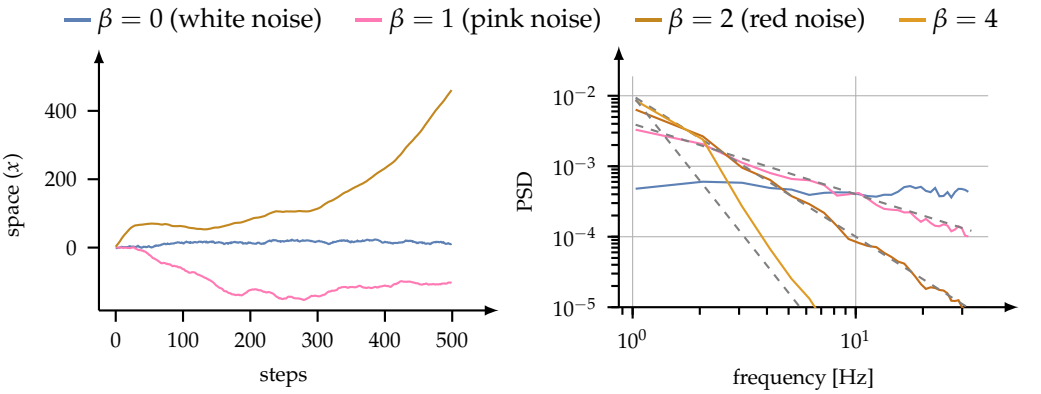
	Intrinsic motivation	Computational methods
CWYC	**learning progress + surprise**	**task-level planning, relational attention**
h-DQN (Kulkarni et al., 2016)	reaching subgoals	HRL, DQN
IMGEP (Forestier et al., 2020)	learning progress	memory-based
CURIOUS (Colas et al., 2019)	learning progress	DDPG, HER, E-UVFA
SAC-X (M. A. Riedmiller et al., 2018)	auxiliary task	HRL, (DDPG-like) PI
Relational RL (Zambaldi et al., 2019)	-	relation net, IMPALA
ICM (Pathak et al., 2017)	prediction error	A3C, ICM
Goal GAN (Florensa et al., 2018)	adversarial goal	GAN, TRPO
Asymmetric self-play (Sukhbaatar et al., 2018)	self-play	Alice/Bob, TRPO, REINFORCE

TABLE 6.1: Table of intrinsically motivated RL agents with the type of IM and the computational method used for learning the policy.

policies making them less suitable for one-shot solving of new tasks. (3) The rigid structure of the architecture does not allow for unconventional solutions and makes concurrent training of the entire architecture more challenging. In future work, exploring more adaptive architectures with looser couplings between the individual modules would be interesting so that the network can decide which modules to use and when.

iCEM Intelligent behavior is not only driven by high-level cognitive processes like the ones discussed in the context of the *CWYC* agent. In most animals, far more primitive behavioral patterns are even more dominant. For instance, it is known that certain species, such as predatory animals (Humphries et al., 2010) or honey bees (Reynolds et al., 2009) fall back to search strategies with long-lasting temporal patterns if resources are scarce. Such search strategies can be described by Lévy walks or Lévy flights (Shlesinger et al., 1982). Also, in robotics, it is known that different robotic systems have specific Frequency Response Functions (FRFs), i.e., their response profile varies with the frequency of alternating stimuli (Sinha, 1989). In motor babbling, random motor commands are sent to a robot to explore its motor-control system in an unsupervised fashion. Sending actions to the robot, that are sampled from a white noise distribution, usually does little in terms of exploration, as shown by the blue state-space trajectory in Fig. 6.5A. Instead, tiny and jittery movements of the robot are observed. One alternative is to send action sequences with different frequency profiles (see Fig. 6.5B for the Power Spectral Density (PSD) of different power-law distributed noise patterns) to the robot to explore a larger part of the operational space as shown by the brown and pink state-space trajectories in Fig. 6.5A. The *improved Cross-Entropy Method (iCEM)* discussed in Ch. 4 uses temporally correlated action samples to increase the sample efficiency and performance of a Model-Based Reinforcement Learning (MBRL) planning method compared to a method that does not produce any temporally correlated action samples. In the empirical studies discussed in Sec. 4.5 of Ch. 4, the temporally correlated exploration of *iCEM* shows a significant advantage over the exploration of vanilla Cross-Entropy Method (CEM) and produces much more coherent behavior among a variety of different environments and robotic manipulation tasks. Moreover, *iCEM* is $2.7 - 21.9\times$ more sample efficient than the next best baseline and, at the same time, achieves $120 - 1030\%$ of the best baseline performance in the considered tasks. That means that *iCEM* can run with a much lower computation budget to achieve the same performance as the baselines. Even if used inside a Model Predictive Control (MPC) policy to close the action-perception loop, this pushes the *iCEM* planning method closer to the realm of real-time control, making it especially suitable for real-world robotic applications and other tasks with long-lasting temporal dependencies. For instance, in Gumbsch, Butz, et al. (2021), *iCEM* is used in conjunction with a particular type of Recurrent Neural Network (RNN) to solve partially observable environments with sparsely changing latent states. In Sancaktar et al. (2022), *iCEM* is used together with a learned GNN for planning trajectories that maximize information gain to improve the model. With the improved model, *iCEM* can perform zero-shot generalization to long-horizon object-manipulation tasks. It is worth noting that temporally-correlated noise is not always the preferable choice for exploration noise. In fact, random search has the advantage of being unbiased and typically works best across many domains. The temporally-correlated exploration noise shines in domains with inertia (e.g., robotics) and sparse-reward long-horizon tasks such as FETCH PICK&PLACE.

One essential ingredient that is still missing to make planning methods such as *iCEM* truly capable of real-time control of real robotic systems are fast models that can be efficiently learned from data. This is still an open challenge, especially for interaction-rich settings with non-smooth dynamics. GNNs (Battaglia et al., 2018) and transformer (Vaswani et al., 2017) like attention networks are just two types of promising model classes that are particularly suited for modeling object-object interactions. *Risk-Averse Zero-Order Trajectory Optimization Method (RAZER)* (see Sec. 5) addresses this challenge by learning a model of probabilistic NNs and by separating and accurately estimating the aleatoric and epistemic uncertainties that arise in the context of model learning. Another avenue that opened up recently is to use highly parallelizable GPU-based simulators such as ISAAC GYM (Makoviychuk et al., 2021) for planning. Some other shortcoming of *iCEM* is its finite planning horizon. This can be especially problematic in sparse-reward tasks or high-frequency control. Terminal value functions are one option to transform the finite-horizon planning problem into an infinite-horizon planning problem (Silver, Schrittwieser, et al., 2017; Lowrey et al., 2019). However, learning a good value function in continuous state spaces and action spaces is a challenge on its own and typically requires abundant data.



(A) 1D random walks with colored noise.

(B) PSD of different power-law distributions.

FIGURE 6.5: Colored random noise. (A) Random walks with colored noise of different temporal structures. (B) Power spectrum of colored random action sequences for different β. See Fig. 4.2 in Ch. 4 for more details.

APEX One other major disadvantage of planning methods such as *iCEM* is that they do all the heavy computation during runtime. NN policies, on the other hand, are expensive to train but are cheap to query on any arbitrary state during runtime. To combine the strength of planning methods and NN-based policies, the *Adaptive Policy EXtraction (APEX)* agent discussed in Ch. 4 learns and runs an NN policy in tandem with the *iCEM* planning method. In this setup, the *iCEM* planner takes the role of the teacher and the NN policy the role of the student in an Imitation Learning (IL) arrangement. In Sec. 4.5, it was empirically shown that a one-sided influence from the teacher to the student might not be sufficient to learn a strong NN policy from an inherently stochastic teacher like the *iCEM* planning method. One reason for this could be that the policy class is not expressive enough to capture the multi-modal output distribution of the teacher. In the student-teacher relationship found in social learning, a good teacher constantly gathers feedback from the student to adapt its teaching strategy. This work showed that such a mutual influence could

also be beneficial in the discussed IL setting. In the case of the *APEX* agent, adaptive Guided Policy Search (GPS) is used to constrain the solution space of the *iCEM* teacher such that it stays close to the behavior of the NN student policy the more competent it gets. Constraining the planning method's solution space helps considerably in stabilizing the training of the NN policy by generating more consistent training data that aligns with the solution already found by the student for the Behavioral Cloning (BC) objective. Moreover, the more tightly coupled teacher-student relationship in the *APEX* agents even improves the solutions of the planning method over time alongside the policy.

Even considering all the improvements added to *APEX* to make the policy learning more stable, BC-based IL methods have been proven to produce suboptimal solutions (Stéphane Ross and Bagnell, 2010). One reason for this suboptimal performance is the covariate shift between the teacher's state visitation distribution and the student's state visitation distribution. Hence, methods such as Dataset Aggregation (DAgger) (Stéphane Ross, G. Gordon, et al., 2011) were proposed to mitigate the covariate shift. However, these methods are also not entirely satisfying as they require an extensive amount of data for relabeling and access to an expert who can be queried for actions on arbitrary states. Recently, a new type of IL methods emerged under the name of offline RL (Kumar et al., 2020; Z. Wang et al., 2020; Yu et al., 2021). The idea in offline RL is to extract a policy from a fixed dataset of agent-environment interactions without the possibility of collecting new data in the environment. In contrast to BC, in offline RL actions are not just reproduced, but RL methods are used to solve the sub-Markov Decision Process (MDP) induced by the fixed dataset. This has several advantages. (1) If a state(-action) value function is learned, the policy can be actively discouraged from taking actions not present in the dataset (Kumar et al., 2020). (2) Offline RL is, in principle, able to learn from multiple or stochastic behavioral policies as the RL loss takes care of choosing the optimal action in each state. (3) While BC merely copies the expert, offline RL has the chance to learn a better policy than the one that generated the data. When this work was published, offline RL was not yet ready for off-the-shelf usage. But with the progress made recently in offline RL, it would be an exciting future direction to pair *APEX* with offline RL methods to extract even stronger policies from the *iCEM* expert trajectories. Another way of combining the optimizer and the NN policy is to use the policy to warmstart the optimization (Jetchev et al., 2013; S. W. Chen et al., 2022) to reduce the runtime of the optimizer further if used for real-time control. But also the opposite direction is possible by coupling RL methods with trajectory optimization to learn an optimal long-term decision-making strategy (Mirchevska et al., 2021).

RAZER Learning good predictive models is important to make real-world robotic control feasible because not all aspects of the real world can be modeled with analytical equations. However, to act robustly in an inherently uncertain environment such as the natural world, any autonomous agent also has to take the uncertainties in the environment into account during planning. The *RAZER* agent presented in Ch. 5 learns a model of the environment that is later used for planning. In contrast to prior work (M. Deisenroth et al., 2011; Chua et al., 2018), the *Probabilitic Ensembles with Trajectory Sampling and Uncertainty Separation (PETSUS)* model inside the *RAZER* agent can separate the two main sources of uncertainty that may arise in the domain of learned models: aleatoric and epistemic uncertainties. While the former type of uncertainty originates from the inherent noise in the system or any unobserved system variables, the latter stems from a lack of enough or good training data. Empirical

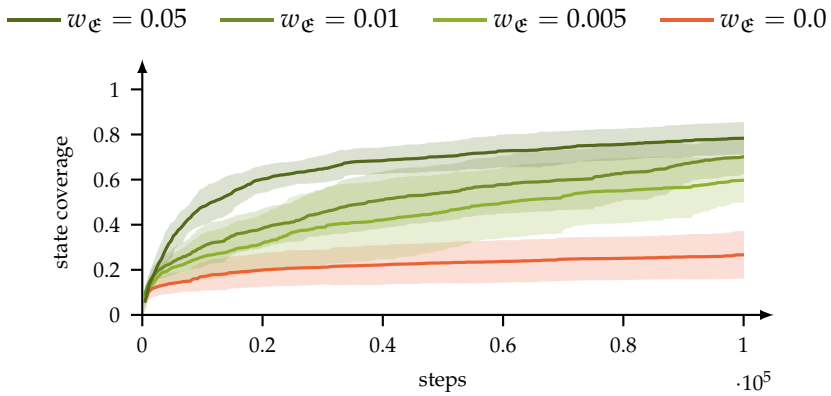
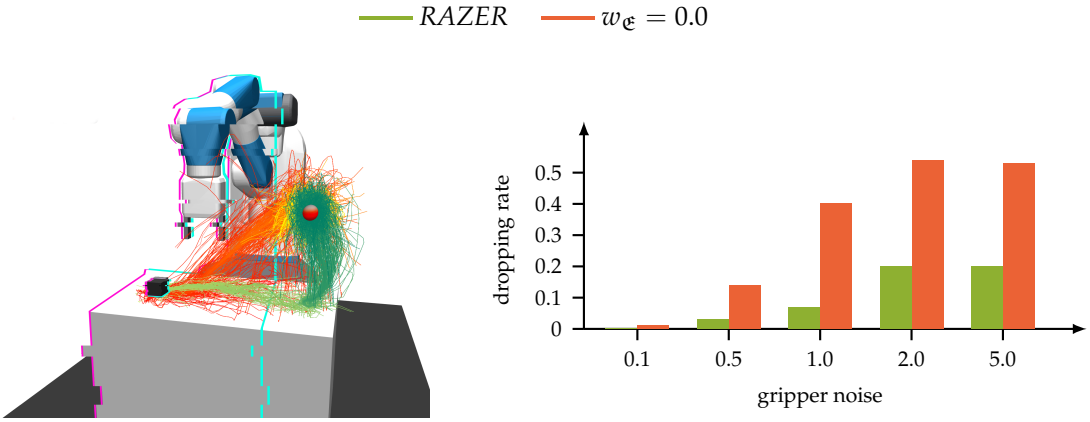


FIGURE 6.6: State-space coverage of an uncertainty-unaware planner and *RAZER* for different values of $w_{\mathfrak{E}}$. A larger weight of the epistemic cost $w_{\mathfrak{E}}$ encourages *RAZER* to seek states for which no or only little training data exists. For more details, see Fig. 5.6 in Ch. 5.

studies in developmental psychology suggest that infants seek new information by actively exploring their environment (Begus et al., 2018) and through social interaction (Bazhydai et al., 2020). In Machine Learning (ML), corresponding information-seeking exploration approaches are studied under the name of active learning (Settles, 2010). The *RAZER* agent implements active learning by seeking states with high epistemic uncertainty, i.e., states that maximize the information gain to improve the model predictions and uncertainty estimates. Figure 6.6 shows the state-space coverage of an uncertainty-unaware planner (red) during the active learning phase versus the state-space coverage of *RAZER* (green). The uncertainty-unaware planner does not have direct access to the model uncertainties and therefore explores the state space only in the vicinity of its solution to the externally provided task. *RAZER*, on the other hand, has direct access to the epistemic uncertainty estimates of the model and uses this information to maximize the information gain during exploration.



(A) Trajectories. (B) Dropping rate.

FIGURE 6.7: Planning in NOISY FETCH PICK&PLACE. (A) The uncertainty-aware planner *RAZER* first pushes the cube out of the noisy region before it lifts the cube to its target location (green). The uncertainty-unaware baseline transports the cube along a straight line between the initial and target position (red), (B) resulting in a much higher dropping rate of the uncertainty-unaware planner compared to *RAZER*. See Fig. 5.9 in Ch. 5 for more details.

Once a good model of the noisy dynamics is learned, the solutions of the uncertainty-aware planner *RAZER* can be actively shaped in goal-directed behavior, as shown by Fig. 6.7. In NOISY FETCH PICK&PLACE, the agent controls a fetch robot with the task of transporting a cube from its initial position on a table to a target position in the air. If the robot's end-effector is right (from the perspective of the robot) from the cyan colored line in Fig. 6.7A, action noise is applied to the robot such that the gripper opens and closes randomly. An uncertainty-unaware planner might take the direct path from the cube's initial position to the target location (red trajectories in Fig. 6.7A), resulting in a high dropping rate as shown in Fig. 6.7B. The uncertainty-aware planner *RAZER*, however, adopts a much safer behavior by first pushing the cube along the table's surface and lifting the cube only if it is right from the cyan-colored line where no action noise is applied (green trajectories in Fig. 6.7A). Consequently, the dropping rate of the *RAZER* agent is much lower even for high levels of action noise.

What makes the design of *RAZER* unique is that the method is very general as it is learned in an entirely data-driven and end-to-end fashion. Hence, *RAZER* can be applied to any system without incorporating much prior knowledge. This is in stark contrast to other methods. In system identification, for instance, the analytical equations governing the state evolution are known and only the system-specific constants are learned from data.

The access to the uncertainty measures of the model's predictions could also help close the sim-to-real gap, for instance, by learning a notion of uncertainty in simulation to act more carefully in parts of the state space of the real world that have high uncertainties. In that way, the robot's dangerous and tedious exploration phase can be done entirely offline and in the safe space of the simulation. During deployment on the real system, the model learned in simulation can then be used for robust planning and control. However, the uncertainties learned in simulation need to be calibrated with respect to the uncertainties incurred in the real system, which is still an unsolved problem.

7

Conclusion & Outlook

The works presented in this thesis discuss nature-inspired inductive biases in Hierarchical Reinforcement Learning (HRL), model-based planning, and model learning for robotic control. The first project relied heavily on human-designed biases. First, in the form of specific architectural choices that can be adapted to an environment from just a few training examples. Second, in the form of design choices regarding the agent's intrinsic motivation that help to structure the exploration in sparse-interaction environments with compositional object-manipulation tasks. Yet, robots might have very different demands, perceive and interact differently with the world, and solve other tasks than humans. Consequently, removing the human designer from the equation and letting data-driven approaches find suitable inductive biases for robotic systems would be desirable. This idea is not new. In the field of meta-learning (Duan et al., 2016; Finn et al., 2017), policies are learned such that they can be adapted to new environments with just a few gradient steps. In neural architecture search (Negrinho et al., 2017; J. X. Wang et al., 2017; Zoph et al., 2017), the architecture of Neural Networks (NNs) and the learning algorithms are optimized directly by data-driven approaches. Entire programs can be synthesized (Balog et al., 2017), for instance, to create sketches of various geometrical shapes (Ganin et al., 2018). In robotics, it would be of great interest to deploy these meta-learning techniques in the constrained optimization spaces of physical systems to find the proper priors for autonomous robotic systems. A closely related research field is continual or lifelong learning (Z. Chen et al., 2018). The idea of life-long learning is that a robot is exposed to a continuous stream of sensorimotor information. In the natural world, some of this information shares some similarities between tasks, while other information is unique for a specific task. Thus, learning a new task should not always start from zero but instead should be embedded into all the previous experience and the already existing competencies of the robot. Combined with the ideas from meta-learning, an interesting future research direction is to identify and learn common inductive biases that are transferable between different robots and tasks and inductive biases that are robot or even task-specific. This repertoire of inductive biases might serve as a prior to accelerate the development of more efficient and self-organized learning robots. Consequently, the remaining works discussed in this thesis look into more data-driven approaches that are applicable to a broader range of problem domains. Especially model-based approaches have the potential to be used inside architectures that implement life-long learning as they provide a straightforward way of continuously incorporating new information to update the inner beliefs of the agent about the world. However, to make model-based approaches truly apt for real-world robotic control, one essential ingredient seems to be still missing: Model architectures that incorporate the necessary biases to effectively model agent-object and object-object interactions commonly found in the real world and that allow for causal reasoning. Similar to how the inductive biases in Convolutional Neural Networks (CNNs) lead to a paradigm shift from hand-designed solutions to data-driven solutions in natural image processing and how transformers did the same in the domain of natural language processing.

A

SUPPLEMENTARY BACKGROUND

A.1 Contraction Mapping

Theorem A.1.1. *There exists a $\gamma \in [0, 1)$ such that for any two value functions $v_1(s)$ and $v_2(s)$ the Bellman operator $\mathcal{T}^\pi$ is a γ-contraction mapping under the sup-norm $\|f(x)\|_\infty = \sup_{x \in \mathcal{X}} |f(x)|$:*

$$\|\mathcal{T}^\pi v_1 - \mathcal{T}^\pi v_2\|_\infty \leq \gamma \|v_1 - v_2\|_\infty \tag{A.1}$$

Proof.

$$\begin{aligned}
\|\mathcal{T}^\pi v_1 - \mathcal{T}^\pi v_2\|_\infty &= \|R(s,\, a) + \gamma \int_{\mathcal{S}} P^\pi_{ss'} v_1(s')\, \mathrm{d}s' \\
&\quad - R(s,\, a) + \gamma \int_{\mathcal{S}} P^\pi_{ss'} v_2(s')\, \mathrm{d}s' \|_\infty \\
&= \gamma \sup_{s' \in \mathcal{S}} \Big| \int_{\mathcal{S}} P^\pi_{ss'} v_1(s')\, \mathrm{d}s' - \int_{\mathcal{S}} P^\pi_{ss'} v_2(s')\, \mathrm{d}s' \Big| \\
&= \gamma \sup_{s'} \Big| \int_{\mathcal{S}} P^\pi_{ss'} v_1(s') - v_2(s')\, \mathrm{d}s' \Big|
\end{aligned} \tag{A.2}$$

Since $0 \leq P^\pi_{ss'} \leq 1$ it follows

$$\|\mathcal{T}^\pi v_1 - \mathcal{T}^\pi v_2\|_\infty = \gamma \sup_{s'} \int_{\mathcal{S}} P^\pi_{ss'} |v_1(s') - v_2(s')|\, \mathrm{d}s' \tag{A.3}$$

From Jensen's inequality (Jensen, 1906) it follows:

$$\begin{aligned}
\|\mathcal{T}^\pi v_1 - \mathcal{T}^\pi v_2\|_\infty &\leq \gamma \int_{\mathcal{S}} P^\pi_{ss'} \sup_{s'} |v_1(s') - v_2(s')|\, \mathrm{d}s' \\
&= \gamma \int_{\mathcal{S}} P^\pi_{ss'}\, \mathrm{d}s' \sup_{s'} |v_1(s') - v_2(s')| \\
&\leq \gamma \sup_{s'} |v_1(s') - v_2(s')| \\
&= \gamma \|v_1(s') - v_2(s')\|_\infty
\end{aligned} \tag{A.4}$$

□

A.2 Proof of the Policy Gradient

Theorem A.2.1 (Policy Gradient Theorem). *For any differentiable policy π_θ and the objective function defined in Eq. 2.26, the policy gradient is proportional to:*

$$\begin{aligned}\nabla_\theta J(\theta) &\propto \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} Q_\pi(s,\ a) \nabla_\theta \pi(a \mid s;\ \theta)\ \mathrm{d}s\ \mathrm{d}a \\ &= \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} Q_\pi(s,\ a) \pi(a \mid s;\ \theta) \nabla_\theta \log \pi(a \mid s;\ \theta)\ \mathrm{d}s\ \mathrm{d}a \\ &= \mathbb{E}_{s\sim\rho,\, a\sim\pi} \left[Q_\pi(s,\ a) \nabla_\theta \log \pi(a \mid s;\ \theta) \right] \end{aligned} \tag{A.5}$$

with $\log \pi(a \mid s;\ \theta)$ being the gradient of the log-likelihood or Score function.

Proof. We start the proof of the policy gradient theorem with

$$\begin{aligned} J(\theta) &= \int_{\mathcal{S}_0} p_0(s_0) V_\pi(s_0)\ \mathrm{d}s_0 \\ &= \int_{\mathcal{S}_0} p_0(s_0) \int_{\mathcal{A}_0} \pi_\theta(a_0 \mid s_0) Q_\pi(s_0,\ a_0)\ \mathrm{d}s_0\ \mathrm{d}a_0. \end{aligned} \tag{A.6}$$

By noting that the start-state distribution $p_0(s_0)$ is independent of θ, it follows:

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta \int_{\mathcal{S}_0} p_0(s_0) V_\pi(s_0)\ \mathrm{d}s_0 \\ &= \int_{\mathcal{S}_0} p_0(s_0) \nabla_\theta V_\pi(s_0)\ \mathrm{d}s_0 \\ &= \int_{\mathcal{S}_0} p_0(s_0) \int_{\mathcal{A}_0} \nabla_\theta \left(\pi_\theta(a_0 \mid s_0) Q_\pi(s_0,\ a_0) \right)\ \mathrm{d}s_0\ \mathrm{d}a_0 \\ &= \int_{\mathcal{S}_0} p_0(s_0) \int_{\mathcal{A}_0} \nabla_\theta \pi_\theta(a_0 \mid s_0) Q_\pi(s_0,\ a_0)\ \mathrm{d}s_0\ \mathrm{d}a_0 \\ &\quad + \int_{\mathcal{S}_0} p_0(s_0) \int_{\mathcal{A}_0} \pi_\theta(a_0 \mid s_0) \nabla_\theta Q_\pi(s_0,\ a_0)\ \mathrm{d}s_0\ \mathrm{d}a_0. \end{aligned} \tag{A.7}$$

Substituting Q_π with its respective Bellman equation yields

$$\begin{aligned} \nabla_\theta J(\theta) = &\int_{\mathcal{S}_0} p_0(s_0) \int_{\mathcal{A}_0} \nabla_\theta \pi_\theta(a_0 \mid s_0) Q_\pi(s_0,\ a_0)\ \mathrm{d}s_0\ \mathrm{d}a_0 \\ &+ \int_{\mathcal{S}_0} p_0(s_0) \int_{\mathcal{A}_0} \pi_\theta(a_0 \mid s_0) \\ &\cdot \nabla_\theta \left(\int_{\mathcal{S}} p(s_1 \mid s_0,\ a_0) \left(R + \gamma \cdot V_\pi(s_1) \right) \right)\ \mathrm{d}s_0\ \mathrm{d}a_0\ \mathrm{d}s_1. \end{aligned} \tag{A.8}$$

With R being independent of θ, we can remove it from the equation and interchange the gradient operation with the integration:

$$\begin{aligned} \nabla_\theta J(\theta) = &\int_{\mathcal{S}_0} p_0(s_0) \int_{\mathcal{A}_0} \nabla_\theta \pi_\theta(a_0 \mid s_0) Q_\pi(s_0,\ a_0)\ \mathrm{d}s_0\ \mathrm{d}a_0 \\ &+ \int_{\mathcal{S}_0} p_0(s_0) \int_{\mathcal{A}_0} \pi_\theta(a_0 \mid s_0) \int_{\mathcal{S}} p(s_1 \mid s_0,\ a_0) \cdot \gamma \\ &\cdot \nabla_\theta V_\pi(s_1)\ \mathrm{d}s_0\ \mathrm{d}a_0\ \mathrm{d}s_1. \end{aligned} \tag{A.9}$$

By reordering the integrals, it follows:

$$\begin{aligned}\nabla_\theta J(\theta) = &\int_{\mathcal{S}_0} p_0(s_0) \int_{\mathcal{A}_0} \nabla_\theta \pi_\theta(a_0 \mid s_0) Q_\pi(s_0,\ a_0)\ \mathrm{d}s_0\ \mathrm{d}a_0 \\ &+ \int_{\mathcal{S}} \int_{\mathcal{S}_0} \gamma \cdot p_0(s_0) \int_{\mathcal{A}_0} \pi_\theta(a_0 \mid s_0) p(s_1 \mid s_0,\ a_0) \\ &\cdot \nabla_\theta V_\pi(s_1)\ \mathrm{d}s_1\ \mathrm{d}s_0\ \mathrm{d}a_0. \end{aligned} \tag{A.10}$$

Using $\int_{\mathcal{A}} \pi(S,\ a;\ \theta) p(S' \mid S,\ a) = p(S \to S',\ 1,\ \pi)$ yields:

$$\begin{aligned}\nabla_\theta J(\theta) = &\int_{\mathcal{S}_0} p_0(s_0) \int_{\mathcal{A}_0} \nabla_\theta \pi_\theta(a_0 \mid s_0) Q_\pi(s_0,\ a_0)\ \mathrm{d}s_0\ \mathrm{d}a_0 \\ &+ \int_{\mathcal{S}} \int_{\mathcal{S}_0} \gamma \cdot p_0(s_0) \cdot p(s_0 \to s_1,\ 1,\ \pi) \\ &\cdot \nabla_\theta V_\pi(s_1)\ \mathrm{d}s_1\ \mathrm{d}s_0\ \mathrm{d}a_0. \end{aligned} \tag{A.11}$$

Notice that at this point, we can recursively substitute Eq. A.7 into Eq. A.11 and get:

$$\begin{aligned}\nabla_\theta J(\theta) &= \sum_{t=0}^{\infty} \int_{\mathcal{S}} \int_{\mathcal{S}_0} \gamma^t \cdot p_0(s_0) \cdot p(s_0 \to s,\ t,\ \pi) \\ &\quad \cdot \int_{\mathcal{A}} \nabla_\theta \pi_\theta(a \mid s) Q_\pi(s,\ a)\ \mathrm{d}s\ \mathrm{d}s_0\ \mathrm{d}a \\ &= \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} \nabla_\theta \pi_\theta(a \mid s) Q_\pi(s,\ a)\ \mathrm{d}s\ \mathrm{d}a. \end{aligned} \tag{A.12}$$

Finally, we can use $\nabla_x \log f(x) = \nabla_x f(x) / f(x)$ (also known as the log trick) and write Eq. A.12 as an expectation:

$$\begin{aligned}\nabla_\theta J(\theta) &= \int_{\mathcal{S}} \rho^\pi(s) \int_{\mathcal{A}} \pi_\theta(a \mid s) \nabla_\theta \log \pi_\theta(a \mid s) Q_\pi(s,\ a)\ \mathrm{d}s\ \mathrm{d}a \\ &= \mathbb{E}_{s \sim \rho, a \sim \pi} \left[Q_\pi(S_t,\ a) \nabla_\theta \log \pi_\theta(a \mid S_t) \right]. \end{aligned} \tag{A.13}$$

□

B

Relational RL

B.1 Algorithm

Algorithm 1: CWYC

for *episode in episodes* **do**
  sample final task $\mathcal{T}^\star \sim \mathcal{T}$
  sample final goal $g^{\mathcal{T}^\star}$ from environment
  compute task chain κ using $(B)_{k,l}$ starting from $\mathcal{T}^\star$; `// κ contains list task indices`
  $i = 1$
  while $t < T^{max}$ *and no success in* $\mathcal{T}^\star$ **do**
    $\mathcal{T}' = \mathcal{T}_{\kappa[i]}$
    if $\mathcal{T}' \neq \mathcal{T}^\star$ **then**
      sample goal $g^{\mathcal{T}'}$ from $G_{\kappa[i],\kappa[i+1]}$; `// Eq. 3.18`
    try to reach $g^{\mathcal{T}'}$ with policy $\pi^{\mathcal{T}'}$
    if $succ^{\mathcal{T}'}$ **then**
      $i = i + 1$; `// next task in task chain`
  store episode in history buffer
  calculate statistics based on history
  train policies for each task
  train task scheduler ; `// Sec. 3.2.3`
  train task planner ; `// Sec. 3.2.4`
  train goal proposal networks ; `// Sec. 3.2.5`

B.2 Training Details and Parameters

TABLE B.1: WAREHOUSE

(A) Training.

Parameter	Value
# parallel rollout worker	5

(B) Environment.

Parameter	Value
arena size	20×20
$T^{\max}$	1600
δ	1.0

(C) SAC.

Parameter	Value
lr	3×10^{-4}
batch size	64
policy type	gaussian
discount	0.99
reward scale	5
target update interval	1
tau (soft update)	5×10^{-3}
action prior	uniform
reg	1×10^{-3}
layer size (π, q, v)	256
# layers (π, q, v)	2
# train iterations	200
buffer size	1×10^{6}

(D) Forward Model.

Parameter	Value
lr	10^{-4}
batch size	64
input	(o_{t-1}, u_{t-1})
confidence interval	5
network type	MLP
layer size	100
θ	5
# layers	9
# train iterations	100

(E) Task Scheduler.

Parameter	Value
$\beta^{\mathcal{T}}$	10^{-1}
lr	10^{-1}
random_eps	0.05
surprise history weighting	0.99

(F) Task Planner.

Parameter	Value
β^{B}	10^{-3}
avg. window size	100
surprise history weighting	0.99
sampling_eps	0.05

(G) Goal Proposal Network.

Parameter	Value
lr	10^{-4}
batch size	64
L1 reg.	0.0
L2 reg.	0.0
γ init	1.0
γ trainable	True
# train iterations	100

TABLE B.2: FETCH PICK&PLACE TOOLUSE

(A) Training.

Parameter	Value
# parallel rollout workers	5

(B) Environment.

Parameter	Value
$T^{\max}$	150
δ	0.05

(C) DDPG+HER.

Parameter	Value
Q_lr	10^{-3}
pi_lr	10^{-3}
batch size	256
polyak	0.95
layer size (π, q)	256
# layers (π, q)	3
# train iterations	80
buffer size	1×10^6
action_l2	1.0
relative goals	false
replay strategy	future
replay_k	4
random_eps	0.3
noise_eps	0.2

(D) Forward Model.

Parameter	Value
lr	10^{-4}
batch size	64
input	(o_{t-1}, u_{t-1})
confidence interval	3
network type	MLP
layer size	100
θ	3
# layers	9
# train iterations	100

(E) Task Scheduler.

Parameter	Value
$\beta^{\mathcal{T}}$	10^{-1}
lr	10^{-1}
random_eps	0.05
surprise history weighting	0.99

(F) Task Planner.

Parameter	Value
β^{B}	10^{-3}
avg. window size	100
surprise history weighting	0.99
sampling_eps	0.05

(G) Goal Proposal Network.

Parameter	Value
lr	10^{-4}
batch size	64
L1 reg.	0.0
L2 reg.	0.0
γ init	1.0
γ trainable	True
# train iterations	30

C

Planning and Control

C.1 Fast Sample-Based Trajectory Optimization

C.1.1 Algorithm

Algorithm 2: 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 Eq. 4.10
  $\sigma_t \leftarrow$ 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. 4.2
    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

C.1.2 Implementation Details and Parameters

TABLE C.1: 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 C.2: Env-dependent Hyperparameter choices.

	iCEM/CEM with ground truth	
horizon h	30	
colored-noise exponent β	0.25	HALFCHEETAH RUNNING
	2.0	HUMANOID STANDUP
	2.5	DOOR
	3.0	FETCH PICK&PLACE
	3.5	RELOCATE

TABLE C.3: Environment settings.

	iCEM/CEM with ground truth	
Episode length	1000	HALFCHEETAH RUNNING
	1000	HUMANOID STANDUP
	200	DOOR
	50	FETCH PICK&PLACE
	200	RELOCATE

C.2 Neural Network Policy Extraction

C.2.1 Algorithm

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**
  $f(a,s) \leftarrow J_t(a,s_t) + \lambda_1 D_{\text{KL}}(\pi_\theta||a) + \lambda_2\|a\|$; // see Eq. 4.4, Eq. 4.5, and Eq. 4.6
  $\tau_{\text{CEM}} \leftarrow n$ Rollout with iCEM$_\pi(f(a,s),\pi_\theta)$; // 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$_\pi(f(a,s),\pi_\theta)$; // 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

C.2.2 Implementation Details and Parameters

TABLE C.4: Expert settings for the considered methods.

	# 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 C.5: 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) $150 \cdot 25$ (FETCH PICK&PLACE), $100 \cdot 10$ (DOOR)				

D

Risk Averse Control

D.1 Algorithm

Algorithm 4: RAZER: Risk-aware and safe CEM-MPC

Parameters:
  N: number of samples; B: 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.2 Implementation Details

D.2.1 Model Learning

TABLE D.1: Model parameters for THREE BRIDGES.

(A) Ensemble parameters.

Parameter	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

(B) Stochastic NN parameters.

Parameter	Value
var_clipping_low	-10.0
var_clipping_high	4
state_dependent_var	True
regularize_automatic_var_scaling	False

(C) Remaining parameters.

Parameter	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

The predicted log variance is bounded by applying (as in Chua et al., 2018, 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, it would be possible to differentiate through this bound to automatically adjust the bounds `max_logvar` and `min_logvar`. However, this can lead to instabilities during training. Hence, these parameters are fixed during training.

The predictive model is trained by alternating between two phases: data collection and model fitting. In THREE BRIDGES, 5 rollouts are collected of length 80 steps and appended to the previous rollouts. Afterwards, the model gets fit to the data for 25 epochs. For NOISY HALFCHEETAH, 1 rollout is collected and the model is fit for 50 epochs. For NOISY FETCH PICK&PLACE and SOLO8 LEANOVEROBJECT the $\hat{f}$ in Fig. 5.1 is replaced by independent instances of noisy ground truth simulators.

D.2.2 Controller Parameters

The parameters used in the controller can be found in Table D.3 and D.4.

TABLE D.2: Model parameters (only differences wrt D.1 are shown) for NOISY HALFCHEETAH environment.

(A) Ensemble parameters.

Parameter	Value
num_layers	4
size	200

(B) Stochastic NN parameters.

Parameter	Value
var_clipping_low	-6.0
state_dependent_var	True

(C) Remaining parameters.

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

D.2.3 Timings

While the code is not tuned for speed specifically, table D.6 provides some timings for a single step in the environment (hyper-parameters are set as specified in Suppl. D.2.1 and Suppl. D.2.2, with num_simulated_trajectories $= 128$ and op_iterations $= 3$).

TABLE D.3: Controller parameters in THREE BRIDGES.

(A) Action sampler parameters.

Parameter	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

(B) Remaining parameters.

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

TABLE D.4: Controller parameters in NOISY HALFCHEETAH (only difference w.r.t. D.3 are shown).

(A) Action sampler parameters.

Parameter	Value
noise_beta	0.25
opt_iterations	4

(B) Remaining parameters.

Parameter	Value
num_simulated_trajectories	120

TABLE D.5: Controller parameters in SOLO8 LEANOVEROBJECT (only difference w.r.t. D.3 are shown).

Action sampler parameters

Parameter	Value
init_std	0.3
noise_beta	3.0

TABLE D.6: 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]
THREE BRIDGES	0.25
NOISY HALFCHEETAH	0.14

BIBLIOGRAPHY

Abraham, Ian et al. (2020). "Model-Based Generalization under Parameter Uncertainty Using Path Integral Control". In: *IEEE Robotics and automation letters* 5.2, pp. 2864–2871 (cit. on pp. 6, 93).

Acharya, Sourya and Samarth Shukla (2012). "Mirror Neurons: Enigma of the Metaphysical Modular Brain". In: *Journal of natural science, biology, and medicine* 3.2, pp. 118–124 (cit. on p. 3).

Achiam, Joshua, Harrison Edwards, et al. (2018). "Variational Option Discovery Algorithms". In: *arXiv:1807.10299* (cit. on p. 38).

Achiam, Joshua and Shankar Sastry (2017). "Surprise-Based Intrinsic Motivation for Deep Reinforcement Learning". In: *arXiv:1703.01732* (cit. on p. 32).

Aitken, Stuart C. (2018). "The Gardener and the Carpenter: What the New Science of Child Development Tells Us about the Relationship between Parents and Children". In: *The AAG Review of Books* 6.2, pp. 101–103 (cit. on pp. 3, 37).

Alhussein, Laith and Maurice A Smith (2021). "Motor Planning under Uncertainty". In: *Elife* 10, e67019 (cit. on pp. 4, 93).

Andrychowicz, Marcin et al. (2017). "Hindsight Experience Replay". In: *Conference on Neural Information Processing Systems (NeurIPS)*, pp. 5048–5058 (cit. on pp. 40, 49).

Arruda, Ermano et al. (2017). "Uncertainty Averse Pushing with Model Predictive Path Integral Control". In: *IEEE/RAS International Conference on Humanoid Robotics (Humanoids)*, pp. 497–502 (cit. on pp. 6, 93).

Asada, Minoru et al. (2009). "Cognitive Developmental Robotics: A Survey". In: *IEEE transactions on autonomous mental development* 1.1, pp. 12–34 (cit. on pp. 5, 32).

Åström, Karl Johan and Peter Eykhoff (1971). "System Identification - A Survey". In: *Automatica* 7.2, pp. 123–162 (cit. on p. 30).

Aubret, Arthur, Laetitia Matignon, and Salima Hassas (2019). "A Survey on Intrinsic Motivation in Reinforcement Learning". In: *arXiv:1908.06976* (cit. on p. 32).

Balog, Matej et al. (2017). "DeepCoder: Learning to Write Programs". In: *International Conference on Learning Representations (ICLR)* (cit. on p. 128).

Banach, Stefan (1922). "Sur Les Opérations Dans Les Ensembles Abstraits et Leur Application Aux Équations Intégrales". In: *Fund. math* 3.1, pp. 133–181 (cit. on p. 17).

Baranes, Adrien and Pierre-Yves Oudeyer (2010). "Intrinsically Motivated Goal Exploration for Active Motor Learning in Robots: A Case Study". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1766–1773 (cit. on p. 33).

— (2013a). "Active Learning of Inverse Models with Intrinsically Motivated Goal Exploration in Robots". In: *Robotics and Autonomous Systems* 61.1, pp. 49–73 (cit. on p. 32).

— (2013b). "Active Learning of Inverse Models with Intrinsically Motivated Goal Exploration in Robots". In: *Robotics and autonomous systems* 61.1, pp. 49–73 (cit. on p. 45).

Barto, Andrew G and Sridhar Mahadevan (2003). "Recent Advances in Hierarchical Reinforcement Learning". In: *Discrete event dynamic systems* 13.1, pp. 41–77 (cit. on p. 30).

Barto, Andrew G., Richard S. Sutton, and Charles W. Anderson (1983). "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems". In: *IEEE Transactions on systems, man, and cybernetics* SMC-13.5, pp. 834–846 (cit. on p. 22).

Battaglia, Peter W. et al. (2018). "Relational Inductive Biases, Deep Learning, and Graph Networks". In: *arXiv: 1806.01261* (cit. on pp. 40, 122).

Baykal-Gürsoy, Melike (2010). "Semi-Markov Decision Processes". In: *Wiley encyclopedia of operations research and management science* (cit. on p. 31).

Bazhydai, Marina, Gert Westermann, and Eugenio Parise (2020). ""I Don't Know but I Know Who to Ask": 12-Month-Olds Actively Seek Information from Knowledgeable Adults". In: *Developmental science* 23.5, e12938 (cit. on p. 124).

Begus, Katarina and Victoria Southgate (2018). "Curious Learners: How Infants' Motivation to Learn Shapes and Is Shaped by Infants' Interactions with the Social World". In: *Active Learning from Infancy to Childhood: Social Motivation, Cognition, and Linguistic Mechanisms*, pp. 13–37 (cit. on p. 124).

Belger, Julia and Juliane Bräuer (2018). "Metacognition in Dogs: Do Dogs Know They Could Be Wrong?" In: *Learning & behavior* 46.4, pp. 398–413 (cit. on pp. 4, 93).

Bellemare, Marc et al. (2016). "Unifying Count-Based Exploration and Intrinsic Motivation". In: *Conference on Neural Information Processing Systems (NeurIPS)*. Vol. 29 (cit. on p. 32).

Bellman, Richard (1957). "A Markovian Decision Process". In: *Journal of mathematics and mechanics*, pp. 679–684 (cit. on p. 13).

Bertsekas, Dimitri (2012). *Dynamic Programming and Optimal Control: Volume I*. Vol. 1. Athena scientific (cit. on p. 11).

Bertsekas, Dimitri P (2011). "Dynamic Programming and Optimal Control 3rd Edition, Volume II". In: *Belmont, MA: Athena Scientific* (cit. on p. 11).

Bongard, Josh C and Rolf Pfeifer (2003). "Evolving Complete Agents Using Artificial Ontogeny". In: *Morpho-Functional Machines: The New Species*, pp. 237–258 (cit. on p. 32).

Brockman, Greg et al. (2016). "OpenAI Gym". In: *arXiv:1606.01540* (cit. on pp. 52, 79).

Bruner, Jerome S. (1973). "Organization of Early Skilled Action". In: *Child development* 44.1, pp. 1–11 (cit. on p. 3).

Burgess, Christopher P. et al. (2019). "MONet: Unsupervised Scene Decomposition and Representation". In: *arXiv:1901.11390* (cit. on pp. 5, 26, 41).

Campero, Andres et al. (2021). "Learning with AMIGo: Adversarially Motivated Intrinsic Goals". In: *International Conference on Learning Representations (ICLR)* (cit. on p. 32).

Cangelosi, Angelo and Matthew Schlesinger (2015). *Developmental Robotics: From Babies to Robots*. MIT press (cit. on p. 32).

Chen, Steven W. et al. (2022). "Large Scale Model Predictive Control with Neural Networks and Primal Active Sets". In: *Autom.* 135, p. 109947 (cit. on p. 123).

Chen, Yilun et al. (2019). "Attention-Based Hierarchical Deep Reinforcement Learning for Lane Change Behaviors in Autonomous Driving". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops* (cit. on p. 31).

Chen, Zhiyuan and Bing Liu (2018). "Lifelong Machine Learning". In: *Synthesis lectures on artificial intelligence and machine learning* 12.3, pp. 1–207 (cit. on p. 128).

Chitnis, Rohan et al. (2021). "GLIB: Efficient Exploration for Relational Model-Based Reinforcement Learning via Goal-Literal Babbling". In: *Conference on Artificial Intelligence (AAAI)*, pp. 11782–11791 (cit. on p. 33).

Chua, Kurtland et al. (2018). "Deep Reinforcement Learning in a Handful of Trials Using Probabilistic Dynamics Models". In: *Conference on Neural Information Processing Systems (NeurIPS)*, pp. 4759–4770 (cit. on pp. xix, 5, 28, 37, 69, 71, 73, 81, 82, 88, 93, 95–98, 100, 106, 123, 142).

Cochran, William T et al. (1967). "What Is the Fast Fourier Transform?" In: *Proceedings of the IEEE* 55.10, pp. 1664–1674 (cit. on p. 72).

Colas, Cédric (2021). "Towards Vygotskian Autotelic Agents : Learning Skills with Goals, Language and Intrinsically Motivated Deep Reinforcement Learning. (Agents Autotéliques Vygostkiens : Buts, Langage et Apprentissage Intrinsèquement Motivé)". PhD thesis. University of Bordeaux, France (cit. on p. 32).

Colas, Cédric et al. (2019). "CURIOUS: Intrinsically Motivated Modular Multi-Goal Reinforcement Learning". In: *International Conference on Machine Learning (ICML)*. Vol. 97, pp. 1331–1340 (cit. on pp. 33, 120).

Cover, Thomas M (1999). *Elements of Information Theory*. John Wiley & Sons (cit. on pp. 5, 98, 99).

Dayan, Peter and Geoffrey E Hinton (1992). "Feudal Reinforcement Learning". In: *Conference on Neural Information Processing Systems (NeurIPS)*. Vol. 5 (cit. on p. 32).

De Boer, Pieter-Tjerk et al. (2005). "A Tutorial on the Cross-Entropy Method". In: *Annals of operations research* 134.1, pp. 19–67 (cit. on p. 81).

de Resende, Briseida Dogo, Eduardo B. Ottoni, and Dorothy M. Fragaszy (2008). "Ontogeny of Manipulative Behavior and Nut-Cracking in Young Tufted Capuchin Monkeys (Cebus Apella): A Perception-Action Perspective". In: *Developmental science* 11.6, pp. 828–840 (cit. on p. 118).

Deisenroth, Marc and Carl E Rasmussen (2011). "PILCO: A Model-Based and Data-Efficient Approach to Policy Search". In: *International Conference on Machine Learning (ICML)*, pp. 465–472 (cit. on pp. 27, 37, 123).

Deisenroth, Marc Peter, Dieter Fox, and Carl Edward Rasmussen (2013). "Gaussian Processes for Data-Efficient Learning in Robotics and Control". In: *IEEE Transactions on pattern analysis and machine intelligence* 37.2, pp. 408–423 (cit. on p. 93).

Der Kiureghian, Armen and Ove Ditlevsen (2009). "Aleatory or Epistemic? Does It Matter?" In: *Structural safety* 31.2, pp. 105–112 (cit. on p. 93).

Dietterich, Thomas (1999). "State Abstraction in MAXQ Hierarchical Reinforcement Learning". In: *Conference on Neural Information Processing Systems (NeurIPS)*. Vol. 12 (cit. on p. 31).

Dietterich, Thomas G (2000). "Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition". In: *Journal of artificial intelligence research* 13, pp. 227–303 (cit. on p. 31).

Duan, Yan et al. (2016). "RL2: Fast Reinforcement Learning via Slow Reinforcement Learning". In: *arXiv:1611.02779* (cit. on p. 128).

Ebert, Frederik et al. (2018). "Visual Foresight: Model-based Deep Reinforcement Learning for Vision-Based Robotic Control". In: *arXiv:1812.00568* (cit. on p. 26).

Elkahky, Ali Mamdouh, Yang Song, and Xiaodong He (2015). "A Multi-View Deep Learning Approach for Cross Domain User Modeling in Recommendation Systems". In: *International Conference on World Wide Web*, pp. 278–288 (cit. on p. 24).

Farley, BWAC and W Clark (1954). "Simulation of Self-Organizing Systems by Digital Computer". In: *Transactions of the IRE professional group on information theory* 4.4, pp. 76–84 (cit. on p. 4).

Finn, Chelsea, Pieter Abbeel, and Sergey Levine (2017). "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *International Conference on Machine Learning (ICML)*. Vol. 70, pp. 1126–1135 (cit. on p. 128).

Florensa, Carlos et al. (2018). "Automatic Goal Generation for Reinforcement Learning Agents". In: *International Conference on Machine Learning (ICML)*. Vol. 80, pp. 1514–1523 (cit. on pp. 32, 38, 120).

Forestier, Sébastien et al. (2020). "Intrinsically Motivated Goal Exploration Processes with Automatic Curriculum Learning". In: *arXiv:1708.02190* (cit. on pp. 32, 39, 120).

Friederici, Angela D. (2011). "The Brain Basis of Language Processing: From Structure to Function". In: *Physiological reviews* 91.4, pp. 1357–1392 (cit. on p. 4).

Fu, Justin, Sergey Levine, and Pieter Abbeel (2016). "One-Shot Learning of Manipulation Skills with Online Dynamics Adaptation and Neural Network Priors". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 4019–4026 (cit. on p. 26).

Fujimoto, Scott, Herke van Hoof, and David Meger (2018). "Addressing Function Approximation Error in Actor-Critic Methods". In: *International Conference on Machine Learning (ICML)*. Vol. 80, pp. 1582–1591 (cit. on pp. 25, 55).

Gal, Yarin, Rowan McAllister, and Carl Edward Rasmussen (2016). "Improving PILCO with Bayesian Neural Network Dynamics Models". In: *Data-Efficient Machine Learning Workshop (ICML)*. Vol. 4, p. 25 (cit. on p. 26).

Ganin, Yaroslav et al. (2018). "Synthesizing Programs for Images Using Reinforced Adversarial Learning". In: *International Conference on Machine Learning (ICML)*. Vol. 80, pp. 1652–1661 (cit. on p. 128).

Garcıa, Javier and Fernando Fernández (2015). "A Comprehensive Survey on Safe Reinforcement Learning". In: *Journal of machine learning research* 16.1, pp. 1437–1480 (cit. on p. 93).

Gopnik, Alison et al. (2004). "A Theory of Causal Learning in Children: Causal Maps and Bayes Nets". In: *Psychological review* 111.1, pp. 3–32 (cit. on pp. 3, 4, 37).

Gordon, Geoffrey J. (1995). "Stable Function Approximation in Dynamic Programming". In: *International Conference on Machine Learning (ICML)*, pp. 261–268 (cit. on p. 25).

Grimm, Christopher et al. (2020). "The Value Equivalence Principle for Model-Based Reinforcement Learning". In: *Conference on Neural Information Processing Systems (NeurIPS)* (cit. on p. 30).

Grimminger, Felix et al. (2020). "An Open Torque-Controlled Modular Robot Architecture for Legged Locomotion Research". In: *IEEE robotics and automation letters* 5.2, pp. 3650–3657 (cit. on p. 104).

Gumbsch, Christian, Martin V. Butz, and Georg Martius (2019). "Autonomous Identification and Goal-Directed Invocation of Event-Predictive Behavioral Primitives". In: *IEEE transactions on cognitive and developmental systems* 13.2, pp. 298–311 (cit. on p. 32).

— (2021). "Sparsely Changing Latent States for Prediction and Planning in Partially Observable Domains". In: *Conference on Neural Information Processing Systems (NeurIPS)*, pp. 17518–17531 (cit. on p. 121).

Gumbsch, Christian, Sebastian Otte, and Martin V. Butz (2017). "A Computational Model for the Dynamical Learning of Event Taxonomies". In: *Annual Meeting of the Cognitive Science Society (CogSci)* (cit. on p. 44).

Gürtler, Nico, Dieter Büchler, and Georg Martius (2021). "Hierarchical Reinforcement Learning with Timed Subgoals". In: *Conference on Neural Information Processing Systems (NeurIPS)*. Vol. 34 (cit. on p. 32).

Ha, David and Jürgen Schmidhuber (2018). "World Models". In: *arXiv:1803.10122* (cit. on p. 26).

Haarnoja, Tuomas et al. (2018). "Soft Actor-Critic: Off-policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *International Conference on Machine Learning (ICML)*. Vol. 80, pp. 1856–1865 (cit. on pp. 25, 49).

Hafner, Danijar et al. (2019). "Learning Latent Dynamics for Planning from Pixels". In: *International Conference on Machine Learning (ICML)*. Vol. 97, pp. 2555–2565 (cit. on pp. 69, 71, 88).

Hayashi, Misato and Tetsuro Matsuzawa (2003). "Cognitive Development in Object Manipulation by Infant Chimpanzees". In: *Animal cognition* 6.4, pp. 225–233 (cit. on p. 119).

Hayashi, Misato and Hideko Takeshita (2009). "Stacking of Irregularly Shaped Blocks in Chimpanzees (Pan Troglodytes) and Young Humans (Homo Sapiens)". In: *Animal cognition* 12 Suppl 1, S49–58 (cit. on p. 119).

Hora, Stephen C (1996). "Aleatory and Epistemic Uncertainty in Probability Elicitation with an Example from Hazardous Waste Management". In: *Reliability engineering & system safety* 54.2-3, pp. 217–223 (cit. on p. 93).

Houthooft, Rein et al. (2016). "Vime: Variational Information Maximizing Exploration". In: *Conference on Neural Information Processing Systems (NeurIPS)*. Vol. 29 (cit. on pp. 5, 33).

Howard, Ronald A (1960). "Dynamic Programming and Markov Processes." In: (cit. on p. 13).

Hüllermeier, Eyke and Willem Waegeman (2021). "Aleatoric and Epistemic Uncertainty in Machine Learning: An Introduction to Concepts and Methods". In: *Machine learning* 110.3, pp. 457–506 (cit. on p. 99).

Humphries, Nicolas E et al. (2010). "Environmental Context Explains Lévy and Brownian Movement Patterns of Marine Predators". In: *Nature* 465.7301, pp. 1066–1069 (cit. on pp. 4, 71, 121).

Hussein, Ahmed et al. (2017). "Imitation Learning: A Survey of Learning Methods". In: *ACM Computing Surveys (CSUR)* 50.2, pp. 1–35 (cit. on p. 70).

James, Karin H et al. (2014). "Young Children's Self-Generated Object Views and Object Recognition". In: *Journal of cognition and development* 15.3, pp. 393–401 (cit. on p. 118).

Janner, Michael et al. (2019). "When to Trust Your Model: Model-based Policy Optimization". In: *Conference on Neural Information Processing Systems (NeurIPS)*, pp. 12498–12509 (cit. on pp. 26, 27).

Jensen, J. L. W. V. (1906). "Sur Les Fonctions Convexes et Les Inégalités Entre Les Valeurs Moyennes". In: *Acta Mathematica* 30.none, pp. 175–193 (cit. on p. 129).

Jetchev, Nikolay and Marc Toussaint (2013). "Fast Motion Planning from Experience: Trajectory Prediction for Speeding up Movement Generation". In: *Auton. Robots* 34.1-2, pp. 111–127 (cit. on p. 123).

Johns, Paul (2014). "Chapter 4 - Sensory and Motor Pathways". In: *Clinical Neuroscience*. Churchill Livingstone, pp. 49–59 (cit. on pp. 3, 37).

Jong, Nicholas K, Todd Hester, and Peter Stone (2008). "The Utility of Temporal Abstraction in Reinforcement Learning." In: *Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 299–306 (cit. on p. 32).

Jonschkowski, Rico and Oliver Brock (2015). "Learning State Representations with Robotic Priors". In: *Autonomous robots* 39.3, pp. 407–428 (cit. on p. 4).

Jonsson, Anders and Andrew Barto (2000). "Automated State Abstraction for Options Using the U-tree Algorithm". In: *Conference on Neural Information Processing Systems (NeurIPS)*. Vol. 13 (cit. on p. 31).

Kaiser, Lukasz et al. (2020). "Model Based Reinforcement Learning for Atari". In: *International Conference on Learning Representations (ICLR)* (cit. on p. 26).

Kalashnikov, D et al. (2018). "Qt-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation". In: *arXiv:1806.10293* (cit. on p. 25).

Kamthe, Sanket and Marc Deisenroth (2018). "Data-Efficient Reinforcement Learning with Probabilistic Model Predictive Control". In: *International Conference on Artificial Intelligence and Statistics*, pp. 1701–1710 (cit. on p. 93).

Kanwisher, Nancy (2010). "Functional Specificity in the Human Brain: A Window into the Functional Architecture of the Mind". In: *Proceedings of the national academy of sciences* 107.25, pp. 11163–11170 (cit. on pp. 3, 37).

Kim, H et al. (2003). "Autonomous Helicopter Flight via Reinforcement Learning". In: *Conference on Neural Information Processing Systems (NeurIPS)*. Vol. 16 (cit. on p. 5).

Kim, Jaekyeom, Seohong Park, and Gunhee Kim (2021). "Unsupervised Skill Discovery with Bottleneck Option Learning". In: *International Conference on Machine Learning (ICML)*. Vol. 139, pp. 5572–5582 (cit. on p. 31).

Kim, Kuno et al. (2020). "Active World Model Learning with Progress Curiosity". In: *International Conference on Machine Learning (ICML)*, pp. 5306–5315 (cit. on pp. 5, 32).

Kocijan, Juš et al. (2004). "Gaussian Process Model Based Predictive Control". In: *IEEE American Control Conference* (cit. on p. 26).

Konidaris, George and Andrew Barto (2009). "Skill Discovery in Continuous Reinforcement Learning Domains Using Skill Chaining". In: *Conference on Neural Information Processing Systems (NeurIPS)* (cit. on p. 38).

Kostrikov, Ilya, Ashvin Nair, and Sergey Levine (2021). "Offline Reinforcement Learning with Implicit Q-learning". In: *arXiv: 2110.06169* (cit. on p. 89).

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet Classification with Deep Convolutional Neural Networks". In: *Conference on Neural Information Processing Systems (NeurIPS)*. Vol. 25 (cit. on p. 24).

Kuhl, Patricia K. et al. (1992). "Linguistic Experience Alters Phonetic Perception in Infants by 6 Months of Age". In: *Science* 255.5044, pp. 606–608 (cit. on p. 3).

Kulkarni, Tejas D. et al. (2016). "Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation". In: *Conference on Neural Information Processing Systems (NeurIPS)*, pp. 3675–3683 (cit. on p. 120).

Kullback, Solomon and Richard A Leibler (1951). "On Information and Sufficiency". In: *The annals of mathematical statistics* 22.1, pp. 79–86 (cit. on p. 76).

Kumar, Aviral et al. (2020). "Conservative Q-learning for Offline Reinforcement Learning". In: *Conference on Neural Information Processing Systems (NeurIPS)* (cit. on pp. 70, 89, 123).

Kurutach, Thanard et al. (2018). "Model-Ensemble Trust-Region Policy Optimization". In: *International Conference on Learning Representations (ICLR)* (cit. on p. 26).

Kwakernaak, Huibert and Raphael Sivan (1969). *Linear Optimal Control Systems*. Vol. 1072. Wiley-interscience (cit. on p. 30).

Lake, Brenden M., Tal Linzen, and Marco Baroni (2019). "Human Few-Shot Learning of Compositional Instructions". In: *Annual Meeting of the Cognitive Science Society (CogSci)*, pp. 611–617 (cit. on p. 5).

Lee, Joonho et al. (2020). "Learning Quadrupedal Locomotion over Challenging Terrain". In: *Science Robotics* 5.47 (cit. on p. 69).

Legare, Cristine H., Susan A. Gelman, and Henry M. Wellman (2010). "Inconsistency with Prior Knowledge Triggers Children's Causal Explanatory Reasoning". In: *Child development* 81.3, pp. 929–944 (cit. on p. 3).

Lenz, Ian, Ross A Knepper, and Ashutosh Saxena (2015). "DeepMPC: Learning Deep Latent Features for Model Predictive Control." In: *Robotics: Science and Systems* (cit. on p. 26).

Levine, Sergey and Pieter Abbeel (2014). "Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics". In: *Conference on Neural Information Processing Systems (NeurIPS)* (cit. on p. 27).

Levine, Sergey and Vladlen Koltun (2013). "Guided Policy Search". In: *International Conference on Machine Learning (ICML)* (cit. on p. 75).

Levine, Sergey, Aviral Kumar, et al. (2020). "Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems". In: *arXiv:2005.01643* (cit. on p. 25).

Lewis, Michael (2012). *Social Cognition and the Acquisition of Self*. Springer Science & Business Media (cit. on p. 3).

Lillicrap, Timothy P. et al. (2016). "Continuous Control with Deep Reinforcement Learning". In: *International Conference on Learning Representations (ICLR)* (cit. on p. 25).

Lin, Long-Ji (1992). *Reinforcement Learning for Robots Using Neural Networks*. Carnegie Mellon University (cit. on p. 25).

Linke, Cam et al. (2020). "Adapting Behavior via Intrinsic Reward: A Survey and Empirical Study". In: *Journal of artificial intelligence research* 69, pp. 1287–1332 (cit. on p. 32).

Lopes, Manuel et al. (2012). "Exploration in Model-Based Reinforcement Learning by Empirically Estimating Learning Progress". In: *Conference on Neural Information Processing Systems (NeurIPS)* (cit. on pp. 5, 32).

Lowrey, Kendall et al. (2019). "Plan Online, Learn Offline: Efficient Learning and Exploration via Model-Based Control". In: *International Conference on Learning Representations (ICLR)* (cit. on p. 122).

Lungarella, Max et al. (2003). "Developmental Robotics: A Survey". In: *Connection science* 15.4, pp. 151–190 (cit. on p. 5).

Makoviychuk, Viktor et al. (2021). "Isaac Gym: High Performance GPU Based Physics Simulation for Robot Learning". In: *Track on Datasets and Benchmarks (NeurIPS)* (cit. on pp. 69, 122).

Marcinowski, Emily C., Julie M. Campbell, et al. (2016). "Do Hand Preferences Predict Stacking Skill during Infancy?" In: *Developmental psychobiology* 58.8, pp. 958–967 (cit. on p. 118).

Marcinowski, Emily C., Eliza Nelson, et al. (2019). "The Development of Object Construction from Infancy through Toddlerhood". In: *Infancy: The official journal of the international society on infant studies* 24.3, pp. 368–391 (cit. on p. 119).

Martius, Georg, Ralf Der, and Nihat Ay (2013). "Information Driven Self-Organization of Complex Robotic Behaviors". In: *PloS one* 8.5, e63400 (cit. on p. 32).

Martius, Georg and Christoph H. Lampert (2017). "Extrapolation and Learning Equations". In: *International Conference on Learning Representations (ICLR)* (cit. on p. 26).

McCorduck, Pamela and Cli Cfe (2004). *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence*. CRC Press (cit. on p. 4).

Meltzoff, Andrew N and M Keith Moore (1977). "Imitation of Facial and Manual Gestures by Human Neonates". In: *Science* 198.4312, pp. 75–78 (cit. on p. 3).

— (1997). "Explaining Facial Imitation: A Theoretical Model". In: *Early development & parenting* 6.3-4, pp. 179–192 (cit. on p. 3).

Mendonca, Russell et al. (2021). "Discovering and Achieving Goals via World Models". In: *Conference on Neural Information Processing Systems (NeurIPS)*. Vol. 34 (cit. on p. 33).

Metcalfe, Janet, Arthur P Shimamura, et al. (1994). *Metacognition: Knowing about Knowing*. MIT press (cit. on p. 4).

Mihatsch, Oliver and Ralph Neuneier (2002). "Risk-Sensitive Reinforcement Learning". In: *Machine learning* 49.2, pp. 267–290 (cit. on p. 93).
Miller, Robert M and Rick Lamb (2005). *The Revolution in Horsemanship and What It Means to Mankind*. The lyons press (cit. on pp. 3, 37).
Minsky, Marvin (1961). "Steps toward Artificial Intelligence". In: *Proceedings of the IRE* 49.1, pp. 8–30 (cit. on p. 15).
Minsky, Marvin Lee (1954). *Theory of Neural-Analog Reinforcement Systems and Its Application to the Brain-Model Problem*. Princeton University (cit. on p. 4).
Mirchevska, Branka et al. (2021). "Amortized Q-learning with Model-Based Action Proposals for Autonomous Driving on Highways". In: *International Conference on Robotics and Automation (ICRA)*, pp. 1028–1035 (cit. on p. 123).
Mitchell, Tom M (1980). *The Need for Biases in Learning Generalizations*. Department of Computer Science, Laboratory for Computer Science Research ... (cit. on pp. 4, 117).
Mnih, Volodymyr, Adria Puigdomenech Badia, et al. (2016). "Asynchronous Methods for Deep Reinforcement Learning". In: *International Conference on Machine Learning (ICML)*, pp. 1928–1937 (cit. on pp. 25, 55).
Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, et al. (2013). "Playing Atari with Deep Reinforcement Learning". In: *arXiv:1312.5602* (cit. on pp. 5, 25).
Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, et al. (2015). "Human-Level Control through Deep Reinforcement Learning". In: *Nature* 518.7540, pp. 529–533 (cit. on p. 24).
Möttönen, Riikka et al. (2005). "Viewing Speech Modulates Activity in the Left SI Mouth Cortex". In: *NeuroImage* 24.3, pp. 731–737 (cit. on p. 3).
Mouret, Jean-Baptiste and Jeff Clune (2015). "Illuminating Search Spaces by Mapping Elites". In: *arXiv:1504.04909* (cit. on p. 33).
Nachum, Ofir et al. (2018). "Data-Efficient Hierarchical Reinforcement Learning". In: *Conference on Neural Information Processing Systems (NeurIPS)* (cit. on pp. xvii, 32, 55).
Nagabandi, Anusha et al. (2018). "Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning". In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7559–7566 (cit. on pp. 28, 69).
Nair, Ashvin, Shikhar Bahl, et al. (2020). "Contextual Imagined Goals for Self-Supervised Robotic Learning". In: *Conference on Robot Learning (CoRL)*, pp. 530–539 (cit. on p. 33).
Nair, Ashvin, Murtaza Dalal, et al. (2020). "Accelerating Online Reinforcement Learning with Offline Datasets". In: *arXiv: 2006.09359* (cit. on p. 89).
Negrinho, Renato and Geoff Gordon (2017). "DeepArchitect: Automatically Designing and Training Deep Architectures". In: *arXiv:1704.08792* (cit. on p. 128).
Nehaniv, Chrystopher L. and Kerstin Dautenhahn, eds. (2007). *Imitation and Social Learning in Robots, Humans and Animals: Behavioural, Social and Communicative Dimensions*. Cambridge University Press (cit. on p. 3).
Nguyen-Tuong, Duy, Jan Peters, and Matthias Seeger (2008). "Local Gaussian Process Regression for Real Time Online Model Learning and Control". In: *Conference on Neural Information Processing Systems (NeurIPS)*, pp. 1193–1200 (cit. on p. 26).
OpenAI et al. (2019). "Solving Rubik's Cube with a Robot Hand". In: *arXiv:1910.07113* (cit. on pp. 5, 69).

Oudeyer, Pierre-Yves (2010). "On the Impact of Robotics in Behavioral and Cognitive Sciences: From Insect Navigation to Human Cognitive Development". In: *IEEE Transactions on autonomous mental development* 2.1, pp. 2–16 (cit. on p. 5).

Pateria, Shubham et al. (2021). "Hierarchical Reinforcement Learning: A Comprehensive Survey". In: *Acm computing surveys* 54.5, pp. 1–35 (cit. on p. 30).

Pathak, Deepak et al. (2017). "Curiosity-Driven Exploration by Self-Supervised Prediction". In: *International Conference on Machine Learning (ICML)*. Vol. 70, pp. 2778–2787 (cit. on pp. xvii, 5, 32, 55, 56, 120).

Péré, Alexandre et al. (2018). "Unsupervised Learning of Goal Spaces for Intrinsically Motivated Goal Exploration". In: *International Conference on Learning Representations (ICLR)* (cit. on p. 41).

Pfaffelhuber, E (1972). "Learning and Information Theory". In: *International journal of neuroscience* 3.2, pp. 83–88 (cit. on pp. 5, 93).

Pinneri, Cristina et al. (2020). "Sample-Efficient Cross-Entropy Method for Real-Time Planning". In: *Conference on Robot Learning (CoRL)*. Vol. 155, pp. 1049–1065 (cit. on p. 88).

Pong, Vitchyr et al. (2020). "Skew-Fit: State-covering Self-Supervised Reinforcement Learning". In: *International Conference on Machine Learning (ICML)*. Vol. 119, pp. 7783–7792 (cit. on p. 33).

Portelas, Rémy et al. (2020). "Automatic Curriculum Learning for Deep RL: A Short Survey". In: *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 4819–4825 (cit. on p. 32).

Puterman, Martin L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics (cit. on p. 18).

Radford, Alec et al. (2018). "Improving Language Understanding by Generative Pre-Training". In: (cit. on p. 24).

Rajeswaran, Aravind et al. (2018). "Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations". In: *Conference on Robotics: Science and Systems (RSS)* (cit. on p. 79).

Rao, Anil V (2009). "A Survey of Numerical Methods for Optimal Control". In: *Advances in the astronautical sciences* 135.1, pp. 497–528 (cit. on p. 28).

Reynolds, A. M. et al. (2009). "Honeybees Use a Levy Flight Search Strategy and Odour-Mediated Anemotaxis to Relocate Food Sources". In: *Behavioral ecology and sociobiology* 64, pp. 115–123 (cit. on p. 121).

Richard E. Bellman (1957). *Dynamic Programming*. Princeton University Press (cit. on p. 17).

Richards, Arthur George (2005). "Robust Constrained Model Predictive Control". PhD thesis. Massachusetts Institute of Technology (cit. on pp. 5, 28).

Riedmiller, Martin A. (2005). "Neural Fitted Q Iteration - First Experiences with a Data Efficient Neural Reinforcement Learning Method". In: *European Conference on Machine Learning (ECML)*. Vol. 3720, pp. 317–328 (cit. on p. 25).

Riedmiller, Martin A. et al. (2018). "Learning by Playing Solving Sparse Reward Tasks from Scratch". In: *International Conference on Machine Learning (ICML)*. Vol. 80, pp. 4341–4350 (cit. on p. 120).

Rizzolatti, Giacomo, Leonardo Fogassi, and Vittorio Gallese (2001). "Neurophysiological Mechanisms Underlying the Understanding and Imitation of Action". In: *Nature reviews neuroscience* 2.9, pp. 661–670 (cit. on p. 3).

Röder, Frank et al. (2020). "Curious Hierarchical Actor-Critic Reinforcement Learning". In: *International Conference on Artificial Neural Networks (ICANN)*. Vol. 12397, pp. 408–419. DOI: 10.1007/978-3-030-61616-8_33 (cit. on p. 120).

Rosenblatt, Frank (1957). *The Perceptron, a Perceiving and Recognizing Automaton Project Para*. Cornell aeronautical laboratory (cit. on p. 4).

Ross, Sherman, Alan E. Fisher, and David King (1957). "Sucking Behavior: A Review of the Literature". In: *The journal of genetic psychology* 91.1, pp. 63–81 (cit. on pp. 3, 37).

Ross, Stéphane and Drew Bagnell (2010). "Efficient Reductions for Imitation Learning". In: *Conference on Artificial Intelligence and Statistics*, pp. 661–668 (cit. on pp. 75, 123).

Ross, Stéphane, Geoffrey Gordon, and Drew Bagnell (2011). "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning". In: *Conference on Artificial Intelligence and Statistics*, pp. 627–635 (cit. on pp. 75, 123).

Rubinstein, Reuven (1999). "The Cross-Entropy Method for Combinatorial and Continuous Optimization". In: *Methodology and computing in applied probability* 1.2, pp. 127–190 (cit. on p. 70).

Rubinstein, Reuven Y (1997). "Optimization of Computer Simulation Models with Rare Events". In: *European journal of operational research* 99.1, pp. 89–112 (cit. on p. 28).

Ruggeri, Azzurra et al. (2019). "Shake It Baby, but Only When Needed: Preschoolers Adapt Their Exploratory Strategies to the Information Structure of the Task". In: *Cognition* 193, p. 104013 (cit. on p. 3).

Saffran, Jenny R., Richard N. Aslin, and Elissa L. Newport (1996). "Statistical Learning by 8-Month-Old Infants". In: *Science* 274.5294, pp. 1926–1928 (cit. on p. 3).

Sancaktar, Cansu, Sebastian Blaes, and Georg Martius (2022). "Curious Exploration via Structured World Models Yields Zero-Shot Object Manipulation". In: *arXiv:2206.11403* (cit. on pp. 113, 120, 121).

Santoro, Adam et al. (2017). "A Simple Neural Network Module for Relational Reasoning". In: *Conference on Neural Information Processing Systems (NeurIPS)*, pp. 4967–4976 (cit. on pp. 38, 48).

Santucci, Vieri Giuliano, Gianluca Baldassarre, and Marco Mirolli (2016). "Grail: A Goal-Discovering Robotic Architecture for Intrinsically-Motivated Learning". In: *IEEE transactions on cognitive and developmental systems (TCDS)* 8.3, pp. 214–231 (cit. on p. 33).

Scalaidhe, S. P., F. A. Wilson, and P. S. Goldman-Rakic (1999). "Face-Selective Neurons during Passive Viewing and Working Memory Performance of Rhesus Monkeys: Evidence for Intrinsic Specialization of Neuronal Coding". In: *Cerebral cortex* 9.5, pp. 459–475 (cit. on p. 37).

Schaul, Tom et al. (2015). "Universal Value Function Approximators". In: *International Conference on Machine Learning (ICML)*, pp. 1312–1320 (cit. on p. 30).

Schmidhuber, Jürgen (1991). "A Possibility for Implementing Curiosity and Boredom in Model-Building Neural Controllers". In: *International Conference on Simulation of Adaptive Behavior: From Animals to Animats (SAB)*, pp. 222–227 (cit. on pp. 5, 32).

Schoenholz, Samuel S. and Ekin D. Cubuk (2020). "JAX, M.D.: A Framework for Differentiable Physics". In: *arXiv:1912.04232* (cit. on p. 69).

Schrodt, Fabian et al. (2017). "Mario Becomes Cognitive". In: *Topics in cognitive science* 9.2, pp. 343–373 (cit. on p. 33).

Schulman, John, Sergey Levine, et al. (2015). "Trust Region Policy Optimization". In: *International Conference on Machine Learning (ICML)*. Vol. 37, pp. 1889–1897 (cit. on p. 25).

Schulman, John, Filip Wolski, et al. (2017). "Proximal Policy Optimization Algorithms". In: *arXiv:1707.06347* (cit. on p. 25).

Sekar, Ramanan et al. (2020). "Planning to Explore via Self-Supervised World Models". In: *International Conference on Machine Learning (ICML)*. PMLR, pp. 8583–8592 (cit. on p. 33).

Settles, Burr (2010). *Active Learning Literature Survey. University of Wisconsin*. Tech. rep. (cit. on pp. 106, 124).

Shalev-Shwartz, Shai and Shai Ben-David (2014). *Understanding Machine Learning: From Theory to Algorithms*. Cambridge university press (cit. on p. 26).

Shlesinger, Michael F, Joseph Klafter, and YM Wong (1982). "Random Walks with Infinite Spatial and Temporal Moments". In: *Journal of statistical physics* 27.3, pp. 499–512 (cit. on pp. 71, 121).

Silver, David, Thomas Hubert, et al. (2017). "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm". In: *arXiv:1712.01815* (cit. on p. 5).

Silver, David, Julian Schrittwieser, et al. (2017). "Mastering the Game of Go without Human Knowledge". In: *Nat.* 550.7676, pp. 354–359 (cit. on p. 122).

Sims, David W. et al. (2008). "Scaling Laws of Marine Predator Search Behaviour". In: *Nature* 451, pp. 1098–1102 (cit. on p. 4).

Sinha, Naresh K. (1989). "System Identification - Theory for the User : Lennart Ljung". In: *Autom.* 25.3, pp. 475–476 (cit. on pp. 72, 88, 121).

Smith, Alice E et al. (1997). "Penalty Functions". In: *Handbook of evolutionary computation* 97.1, p. C5 (cit. on p. 99).

Smith, Linda B. et al. (Apr. 2018). "The Developing Infant Creates a Curriculum for Statistical Learning". In: *Trends in cognitive sciences* 22.4, pp. 325–336. DOI: 10.1016/j.tics.2018.02.004 (cit. on p. 118).

Spinath, Birgit and Ricarda Steinmayr (2012). "The Roles of Competence Beliefs and Goal Orientations for Change in Intrinsic Motivation." In: *Journal of educational psychology* 104.4, p. 1135 (cit. on p. 119).

Stahl, Aimee E and Lisa Feigenson (2015). "Observing the Unexpected Enhances Infants' Learning and Exploration". In: *Science* 348.6230, pp. 91–94 (cit. on pp. 3, 119).

Storck, Jan, Sepp Hochreiter, Jürgen Schmidhuber, et al. (1995). "Reinforcement Driven Information Acquisition in Non-Deterministic Environments". In: *International Conference on Artificial Neural Networks (ICANN)*. Vol. 2, pp. 159–164 (cit. on p. 5).

Sukhbaatar, Sainbayar et al. (2018). "Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play". In: *Conference on Learning Representations (ICLR)* (cit. on pp. 32, 120).

Sutton, Richard S (1990). "Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming". In: *Machine Learning Proceedings*, pp. 216–224 (cit. on pp. 5, 26).

— (1991a). "Dyna, an Integrated Architecture for Learning, Planning, and Reacting". In: *ACM sigart bulletin* 2.4, pp. 160–163 (cit. on pp. 5, 26).

— (1991b). "Planning by Incremental Dynamic Programming". In: *Machine Learning Proceedings*. Elsevier, pp. 353–357 (cit. on pp. 5, 26).

— (1995). "On the Virtues of Linear Learning and Trajectory Distributions". In: *Workshop on Value Function Approximation, Machine Learning Conference*, p. 85 (cit. on p. 25).

Sutton, Richard S, David McAllester, et al. (2000). "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Conference on Neural Information Processing Systems (NeurIPS)*. Vol. 12 (cit. on pp. 22, 24).

Sutton, Richard S, Doina Precup, and Satinder Singh (1999). "Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning". In: *Artificial intelligence* 112.1-2, pp. 181–211 (cit. on pp. 31, 37).

Sutton, Richard S. (1988). "Learning to Predict by the Methods of Temporal Differences". In: *Machine learning* 3.1, pp. 9–44 (cit. on p. 19).

Sutton, Richard S. and Andrew G. Barto (1998). "Reinforcement Learning: An Introduction". In: *IEEE trans. Neural networks* 9.5, pp. 1054–1054 (cit. on p. 11).

Tassa, Yuval, Tom Erez, and Emanuel Todorov (2012). "Synthesis and Stabilization of Complex Behaviors through Online Trajectory Optimization". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4906–4913 (cit. on p. 27).

Timmer, J and M Koenig (1995). "On Generating Power Law Noise." In: *Astronomy and astrophysics* 300, p. 707 (cit. on p. 72).

Todorov, Emanuel, Tom Erez, and Yuval Tassa (2012). "MuJoCo: A Physics Engine for Model-Based Control". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5026–5033 (cit. on pp. 51, 102).

Tsitsiklis, John N and Benjamin Van Roy (1997). "An Analysis of Temporal-Difference Learning with Function Approximation". In: *IEEE transactions on automatic control* 42.5, pp. 674–690 (cit. on p. 25).

Turing, Alan M. (1990). "Computing Machinery and Intelligence". In: *The Philosophy of Artificial Intelligence*. Oxford Readings in Philosophy. Oxford University Press, pp. 40–66 (cit. on p. 4).

Ueno, Moeko et al. (2018). "Crawling Experience Relates to Postural and Emotional Reactions to Optic Flow in a Virtual Moving Room". In: *Journal of motor learning and development* 6.s1, S63–S75 (cit. on p. 118).

Vaswani, Ashish et al. (2017). "Attention Is All You Need". In: *Conference on Neural Information Processing Systems (NeurIPS)*, pp. 5998–6008 (cit. on p. 122).

Versace, Elisabetta and Giorgio Vallortigara (2015). "Origins of Knowledge: Insights from Precocial Species". In: *Frontiers in behavioral neuroscience* 9 (cit. on pp. 3, 37).

Wang, Jane X. et al. (2017). "Learning to Reinforcement Learn". In: *arXiv:1611.05763* (cit. on p. 128).

Wang, Tingwu and Jimmy Ba (2020). "Exploring Model-Based Planning with Policy Networks". In: *International Conference on Learning Representations (ICLR)* (cit. on pp. 69, 71, 74, 81).

Wang, Ziyu et al. (2020). "Critic Regularized Regression". In: *Conference on Neural Information Processing Systems (NeurIPS)* (cit. on pp. 70, 89, 123).

Warde-Farley, David et al. (2019). "Unsupervised Control through Non-Parametric Discriminative Rewards". In: *International Conference on Learning Representations (ICLR)* (cit. on p. 33).

Watkins, Christopher J. C. H. and Peter Dayan (1992). "Q-Learning". In: *Machine learning* 8.3, pp. 279–292 (cit. on pp. 20, 21).

Weisler, Ann and Robert R. McCall (1976). "Exploration and Play: Resume and Redirection". In: *American psychologist* 31.7, pp. 492–508 (cit. on p. 3).

Wen, Min and Ufuk Topcu (2018). "Constrained Cross-Entropy Method for Safe Reinforcement Learning". In: *Conference on Neural Information Processing Systems (NeurIPS)*, pp. 7461–7471 (cit. on p. 101).

Williams, Grady, Andrew Aldrich, and Evangelos Theodorou (2015). "Model Predictive Path Integral Control Using Covariance Variable Importance Sampling". In: *arXiv:1509.01149* (cit. on p. 69).

Williams, Ronald J. (1992). "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Machine learning* 8.3, pp. 229–256 (cit. on p. 22).

Wu, Jiajun et al. (2015). "Galileo: Perceiving Physical Object Properties by Integrating a Physics Engine with Deep Learning". In: *Conference on Neural Information Processing Systems (NeurIPS)*, pp. 127–135 (cit. on p. 26).

Yingjun, Pei and Hou Xinwen (2019). "Learning Representations in Reinforcement Learning: An Information Bottleneck Approach". In: *arXiv:1911.05695* (cit. on p. 5).

Yu, Tianhe et al. (2021). "Combo: Conservative Offline Model-Based Policy Optimization". In: *Conference on Neural Information Processing Systems (NeurIPS)*, pp. 28954–28967 (cit. on pp. 70, 123).

Zambaldi, Vinícius Flores et al. (2019). "Deep Reinforcement Learning with Relational Inductive Biases". In: *International Conference on Learning Representations (ICLR)* (cit. on pp. 38, 120).

Zoph, Barret and Quoc V. Le (2017). "Neural Architecture Search with Reinforcement Learning". In: *International Conference on Learning Representations (ICLR)* (cit. on p. 128).