Methods for Learning Adaptive and Symbolic Forward Models for Control and Planning

Dissertation

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

vorgelegt von
Jan Michael Achterhold
aus Warendorf

Tübingen
2023

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

Tag der mündlichen Qualifikation:	12.04.2024
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr. Jörg-Dieter Stückler
2. Berichterstatter:	Prof. Dr. Matthias Bethge

Abstract

Learning-based methods for sequential decision making, i.e., methods which leverage data, have shown the ability to solve complex problems in recent years. This includes control of dynamical systems, as well as mastering games such as Go and StarCraft. In addition, these methods often promise to be applicable to a wide variety of problems.

A subclass of these methods are model-based methods. They leverage data to learn a model which allows predicting the evolution of a dynamical system to control. In recent research, it was shown that these methods, in contrast to model-free methods, require less data to be trained. In addition, model-based methods allow re-using the dynamics model when the task to be solved has changed, and straightforward adaptation to changes in the system's dynamics.

One particular focus of this thesis is on learning dynamics models which can data-efficiently adapt to changes in the system's dynamics, as well as the efficient collection of data to adapt a learned model. In this regard, two novel methods are presented.

In the application domain of autonomous robot navigation, in which both parameters of the robot and the terrain are subject to change, a novel method comprising an adaptive dynamics model is presented and evaluated on a simulated environment.

A further advantage of model-based methods is the ability to incorporate physical prior knowledge for model design. In this thesis, we demonstrate that leveraging physical prior knowledge is advantageous for the task of tracking and predicting the motion of a table tennis ball, respecting its spin.

However, model-based methods, in particular planning with learned models, have to cope with certain challenges. For long prediction horizons, which are required if the effect of an action is apparent only far in the future, model errors accumulate. In addition, model-based planning is commonly computationally intensive, which is problematic if high-frequency, reactive control is required. In this thesis, a method is presented to alleviate these problems. To this end, we propose a two-layered hierarchical method. Model-based planning is only applied on the higher layer on symbolic abstractions. On the lower-layer, model-free reactive control is used. We show successful application of this method to board games which can only be interacted with through a robotic manipulator, e.g., a robotic arm, which requires high-frequency reactive control.

Zusammenfassung

Lernbasierte Verfahren zur sequenziellen Entscheidungsfindung, d. h. Verfahren, die Informationen aus Daten extrahieren und nutzen, haben in der jüngeren Vergangenheit komplexe Probleme gelöst. Dies umfasst die Regelung dynamischer Systeme und die Beherrschung von Spielen wie Go und StarCraft. Zudem versprechen diese Verfahren, auf eine große Klasse von Problemen anwendbar zu sein.

Eine Unterklasse dieser Verfahren sind modellbasierte Verfahren. In ihnen wird aus Daten ein Modell gelernt, was es ermöglicht, Aussagen über das zukünftige Verhalten des zu regelnden Systems zu treffen. In jüngerer Forschung hat sich gezeigt, dass diese Verfahren im Vergleich zu modellfreien Verfahren, in denen kein explizites Modell über das Verhalten des Systems gelernt wird, dateneffizienter zu lernen sind. Weiterhin ermöglichen modellbasierte Verfahren die Weiternutzung des Systemmodells, wenn sich die zu lösende Aufgabe ändert, und eine einfache Adaption des Systemmodells, wenn sich die Dynamik des Systems ändert.

Diese Arbeit beleuchtet u. a. Aspekte des Lernens von Systemmodellen unter dem besonderen Fokus der dateneffizienten Adaptierbarkeit auf sich ändernde Dynamik sowie das effiziente Sammeln von Daten zur Adaption eines gelernten Modells, wozu zwei neuartige Verfahren vorgestellt werden.

Im Anwendungsfeld der Roboternavigation, in der sowohl Parameter des Roboters als auch der Bodenbeschaffenheit Änderungen unterliegen, wird ein Verfahren basierend auf adaptiven Dynamikmodellen vorgestellt und empirisch in einer Simulationsumgebung untersucht.

Ein weiterer Vorteil modellbasierter Verfahren ist die einfache Einbindung physikalischen Vorwissens in die Modellbildung. In dieser Arbeit wird demonstriert, dass dies vorteilhaft ist für die Modellierung und Vorhersage der Flugbahn eines Tischtennisballs unter Berücksichtigung der Eigenrotation des Tischtennisballs.

Modellbasierte Verfahren, insbesondere modellbasiertes Planen, unterliegen jedoch auch Nachteilen: Bei langen Prädiktionshorizonten, welche erforderlich sind, wenn sich der Effekt einer Aktion erst weit in der Zukunft auswirkt, akkumulieren sich Modellfehler. Zudem ist modellbasiertes Planen sehr rechenintensiv, was für eine hochfrequenten, reaktive Regelung problematisch ist. In dieser Arbeit wird ein Verfahren vorgeschlagen, um diese Probleme des modellbasierten Planens abzuschwächen. Dazu wird in einer zweistufigen Hierarchie nur auf der oberen Ebene modellbasiertes Planen auf symbolischen Abstraktionen genutzt, während auf der unteren Ebene modellfreie, reaktive Regelung zum Einsatz kommt. In dieser Arbeit wird dieses Verfahrens erfolgreich zum Lösen von Brettspielen angewendet, mit denen nur in Form von robotischen Manipulatoren (bspw. ein Roboterarm) interagiert werden kann, welcher eine hochfrequente reaktive Regelung benötigt.

Acknowledgements

First, I want to thank my supervisor Jörg Stückler for guiding me towards interesting topics and research questions, for giving me the freedom to also pursue my own ideas, and for regularly providing feedback during the development of the particular projects. Your knowledge about anything related to robotics, computer vision, and machine learning is really remarkable, and has been a very valuable support during my time in the Embodied Vision group. Thank you!

I also want to express my gratitude to my thesis advisory committee, consisting of Matthias Bethge and Sebastian Trimpe, who gave very valuable advice over the last years. I also want to thank Georg Martius and Martin Butz for being part of the examination committee.

The thesis advisory committee is one of the helpful instruments of the International Max Planck Research School for Intelligent Systems (IMPRS-IS). I thank everyone involved in installing and keeping up the IMPRS-IS, in particular Leila Masri and Sara Sorce, not only for administrative guidance, but also for creating a vibrant community.

The work presented in this thesis would not have been possible without help from many collaborators. I am very thankful that I had the opportunity to advise the master's thesis projects of Nathanael Bosch, Rama Kandukuri, Mohammad Kalim Akram, Suresh Guttikonda, Markus Krimmel, and Lukas Mack. Several of your contributions have shaped this work. In addition, I want to thank the student assistants / interns Jonathan Schmidt and Johannes Meier, who have helped me with implementation work in several projects.

I also very much appreciate the help by Philip Tobuschat and Hao Ma for collecting trajectories on the table tennis setup and the hours we spent in the table tennis lab to get the robotic return working. Additional thanks go to Alexander Dittrich for providing measurement data and insights on the ball launcher, Dieter Büchler for his advice and all his work on the table tennis setup, and Michael Mühlebach for his advice and the careful proofreading of our paper. I very much enjoyed working with you!

I thank all current and past members of the Embodied Vision group and colleagues at the Max Planck Institute in Tübingen for maintaining the pleasant working atmosphere. In particular, I thank Michael Strecke and Jens Kreber for very valuable feedback on this thesis, and Rama Kandukuri for the interesting discussions during coffee breaks.

Thank you, Cathrin Elich, Vincent Stimper, Nathanael Bosch and Michael Strecke for sharing the highs and lows of a PhD. Together with my friends and flatmates Sophie Engelhardt, Alina Happ, Hannah Deininger and Tobias Weiß, you make me feel home in Tübingen.

I want to thank my family, my sister Eva and my parents Gabi and Anton, that I never had to question their support.

Finally, I would like to thank everyone who has been patient with me when I have had to put you off by saying *"Wait a minute, I just need to start some experiments"*.

Jan Achterhold
Tübingen, July 2023

Contents

List of Figures

List of Tables

Notation

Numbers, Vectors, Matrices, Sets

$\mathbb{R}$	Set of real numbers
$\mathbb{R}_{>0}$	Set of real numbers greater than zero
$\mathbb{R}_{\geq 0}$	Set of real numbers greater or equal to zero
$\mathbb{N}$	Set of natural numbers, excluding 0
$\mathbb{N}_0$	Union of natural numbers and $\{0\}$
x, X	A scalar ($x \in \mathbb{R}$, $X \in \mathbb{R}$)
$\boldsymbol{x}$	A column vector ($\boldsymbol{x} \in \mathbb{R}^n$, includes vectors of dimension 1, i.e., scalars)
$\boldsymbol{X}$	A matrix ($\boldsymbol{X} \in \mathbb{R}^{n \times m}$)
$[\boldsymbol{x}]_i$	i^{th} entry of vector $\boldsymbol{x}$ (1-based indexing)
$[\boldsymbol{X}]_{i,j}$	Entry of matrix $\boldsymbol{X}$ at row i and column j (1-based indexing)
$\boldsymbol{X}^\top$	Transpose of $\boldsymbol{X}$
$\boldsymbol{X}^{-1}$	Inverse of square nonsingular matrix $\boldsymbol{X} \in \mathbb{R}^{n \times n}$
X	A finite, ordered collection of elements (tuple), e.g. $\mathrm{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)$
$\mathcal{X}$	A set, e.g. $\mathcal{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3\}$
$\boldsymbol{I}_n$	Identity matrix of size $n \times n$
$\boldsymbol{1}_n$	All-one vector of dimension n
$\boldsymbol{0}_n$	All-zero vector of dimension n
$\boldsymbol{0}_{n \times m}$	All-zero matrix of dimension $n \times m$
$\mathrm{diag}(\boldsymbol{s})$	Square matrix with $\boldsymbol{s}$ on its diagonal and 0 elsewhere

Sequential data

$\boldsymbol{o}_n$	Observation at timestep n
$\boldsymbol{s}_n$	State at timestep n
$\boldsymbol{a}_n$	Action at timestep n
r_n	Reward at timestep n
$x_{m:n}$	Ordered list of elements from m to n (inclusive), i.e., $x_{m:n} := (x_m, x_{m+1}, \ldots, x_{n-1}, x_n)$
$x(t)$	Variable x at continuous time $t \in \mathbb{R}$

Probability distributions

$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	Random variable $\boldsymbol{x}$ is distributed according to a (multivariate) Gaussian distribution with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$
$p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$	$p(\boldsymbol{x})$ is the p.d.f. of a Gaussian distribution
$x \sim \mathcal{U}[a, b]$	$x \in \mathbb{R}$ is uniformly distributed on the closed interval $[a, b]$
$x \sim \mathcal{U}\{1, \ldots, K\}$	$x \in \{1, \ldots, K\}$ is uniformly distributed, i.e., $\Pr[x = k] = \frac{1}{K}$, $\forall k \in \{1, \ldots, K\}$.

$\mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})}[f(\boldsymbol{x})]$	Expectation of $f(\boldsymbol{x})$, with x distributed according to distribution with p.m.f./p.d.f. $p(\boldsymbol{x})$

Information theory

$\mathrm{H}[p(\boldsymbol{x})]$	Shannon entropy of a probability distribution with p.m.f. $p(\boldsymbol{x})$
$\mathrm{H}[p(\boldsymbol{x})]$	Differential entropy of a probability distribution with p.d.f. $p(\boldsymbol{x})$
$\mathrm{H}(\boldsymbol{x})$	Shannon (differential) entropy of a random variable $\boldsymbol{x}$
$\mathrm{H}(\boldsymbol{x} \mid \boldsymbol{y})$	Conditional Shannon (differential) entropy of a random variable $\boldsymbol{x}$ given $\boldsymbol{y}$
$\mathrm{KL}[p(\boldsymbol{x}) \,\|\|\, q(\boldsymbol{x})]$	Kullback-Leibler divergence between a probability distribution with p.d.f./p.m.f. p and a probability distribution with p.d.f./p.m.f. q
$\mathrm{I}(\boldsymbol{x}, \boldsymbol{y})$	Mutual information between two random variables $\boldsymbol{x}$ and $\boldsymbol{y}$, with $\boldsymbol{x}, \boldsymbol{y} \sim p(\boldsymbol{x}, \boldsymbol{y})$
$\mathrm{I}(\boldsymbol{x}, \boldsymbol{y} \mid \boldsymbol{z})$	Conditional mutual information between the random variables $\boldsymbol{x}$ and $\boldsymbol{y}$ given $\boldsymbol{z}$, with $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \sim p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$

All logarithms occurring in computing the Shannon entropy, differential entropy, Kullback-Leibler divergence, mutual information, and conditional mutual information, are to the base e (Euler's number). Consequently, all the above quantities are measured in "nats".

Miscellaneous

$\mathcal{O}(\texttt{<expr>})$	Big O notation; an algorithm has time or memory complexity in the order of `<expr>`, e.g., $\mathcal{O}(mn^3)$.
$\mathrm{ReLU}(\boldsymbol{x})$	Rectified Linear Unit (ReLU). For $\boldsymbol{x} \in \mathbb{R}^D$, $[\mathrm{ReLU}(\boldsymbol{x})]_i = \max([\boldsymbol{x}]_i, 0) \quad \forall i \in \{1, \ldots, D\}$.
$\mathcal{J}$	An objective function for maximization (e.g., evidence lower bound, $\max_\theta \mathcal{J}(\theta)$).
$\mathcal{L}$	An objective function for minimization (e.g., loss function, $\min_\theta \mathcal{L}(\theta)$).
$\mathbb{1}[\texttt{<predicate>}]$	Iverson bracket (Iverson, 1962). Takes value of 1 if `<predicate>` is true, 0 otherwise.
$\boldsymbol{x} = \mathrm{onehot}(n, N)$	Vector $\boldsymbol{x} \in \{0, 1\}^N$ with $[\boldsymbol{x}]_n = 1$, $[\boldsymbol{x}]_m = 0 \;\; \forall m \in \{1, \ldots, N\} \setminus \{n\}$.

Acronyms

p.d.f.	Probability density function
p.m.f.	Probability mass function
BFS	Breadth-first search
BOED	Bayesian optimal experimental design (see Subsection 4.3.1)
CEM	Cross-entropy method (see Subsection 2.6.1)
GP	Gaussian process (see Subsection 3.3.1)
EKF	Extended Kalman Filter (see Subsection 6.3.5)
ELBO	Evidence lower bound (see Subsection 2.5.2)
LGSSM	Linear Gaussian state-space model (see Section 6.3)
MDP	Markov decision process (see Subsection 2.3.1)
ML	Machine learning (see Section 2.4)
NP	Neural Process (see Subsection 4.3.2)
POMDP	Partially observable Markov decision process (see Subsection 2.3.2)
RL	Reinforcement learning (see Subsection 2.3.6)
VAE	Variational Auto-Encoder (see Subsection 2.5.3)

Overview 1.

1.1. Introduction

This thesis revolves around sequential decision making problems, in which an *agent* takes *actions* in an *environment* in order to fulfill some *task*. We assume aspects of the task and environment to be unknown, which the agent has to infer from *data* — i.e., it has to incorporate *learning*.

An example for such a problem is learning to play a board game, e.g., chess, in which an agent learns which moves are more likely to lead to a win than others. When playing a board game, deciding on which moves to make is not the only decision making problem. In addition, the execution of a move itself also requires decision making, e.g., about how to move the hand in order to grasp a chess piece. This may also incorporate learning, e.g., about which movements are suited to reliably grasp the chess piece.

The way humans make decisions can be linked to the dual process theory of cognitive psychology, which suggests two different modes of human thinking (Evans, 1984). Kahneman (2003) denotes them by System 1 (intuition, *thinking fast*) and System 2 (reasoning, *thinking slow*). System 1 is fast, affective, automatic or even unconscious. Exemplarily, a chess player observes a particular opening by his opponent. Through historic plays, he has collected experience on which reacting strategy is promising to win the game, and selects his next move accordingly.

System 2, however, is slow and effortful. Relating to the chess playing example, an example of System 2 thinking is when a player imagines multiple possible moves, evaluates their outcomes, and chooses his next move based on these considerations.

Technical implementations of sequential decision making agents can be classified into *model-free* and *model-based* approaches (Moerland et al., 2023; Sutton, 1991; Sutton and Barto, 2018). Model-free approaches typically share properties of System 1: Action selection is based on (learned) heuristics, and fast. Typically, a mapping (policy) is learned from a current situation (e.g., chess board state) to an action (game move), which bounds the solution to a particular task.

With a model-based approach, multiple possible futures can be imagined and evaluated, which is computationally intensive, but easier to apply to a variety of tasks. For this, a model of the environment is learned, modeling the effect of an action, e.g., the effect of a particular chess move on the board state. Leveraging a model to foresee the consequences of actions and acting accordingly is related to the slow and effortful operations of System 2.

In some environments, in particular in the real world, the dynamics are subject to change — exemplarily, the chess opponent changes his playing style, or the chess piece might be slightly moist, making it more difficult to grasp. A model-based method is more straightforward to adapt to this situation, as the model only encodes information on the environment's dynamics, and not on the task, unlike the policy does in a model-free method. Another advantage of model-based methods is that physical prior knowledge can easily be integrated into the model. This aligns with the classification by Kahneman (2003), stating that operations of System 1 (model-free) "are governed by habit and [. . .] therefore difficult to control or modify", while operations of System 2 (model-based) are "relatively flexible and potentially ruled-governed".

Model-based methods to sequential decision making and their adaptation capabilities form the main body of this thesis. We propose a novel method for adaptive dynamics models which are learned directly from high-dimensional observations such as images in Chapter 3. With the capability to adapt to varying dynamics, the question arises on how to interact with an environment in order to quickly learn how it behaves, and subsequently adapt the model based on this experience. Figuratively speaking, a human might “experiment" with the moist chess piece to learn how is behaves under different graspings, to be able to stably grasp it. A computational method for this active learning of dynamics models is presented in Chapter 4. In Chapter 5 we consider the special case of robot- and terrain-aware navigation, in which a robot, whose dynamics are subject to change, has to navigate over varying terrains. We also address this with adaptive dynamics models. Chapter 6 demonstrates the efficacy of incorporating physical prior knowledge in a model which predicts the motion of a table tennis ball for robotic return. The board game example introduced above serves as a good example for the challenges adressed in Chapter 7. In order to play a board game, not only logical planning capabilities are required (to plan the next move), but also fast and reactive control (for moving a robotic hand). We present a learning agent which captures these two requirements.

Structure of this thesis While the above aims to serve as an informal introduction to model-based sequential decicion making and the challenges adressed in this thesis, after commenting on historical remarks, we will outline the contributions of this thesis in a more technical way. For this, we first briefly introduce some basic terminology as preliminaries for the following discussions. Then, we present general and real-world challenges of sequential decision making (Section 1.4), followed by concepts and approaches to sequential decision making and their relation to the aforementioned challenges in Section 1.5. The contributions of this thesis are summarized in Section 1.6, followed by a list of accompanying publications (Section 1.7) and software releases (Section 1.8). More detailed technical preliminaries are presented in Chapter 2, followed by a detailed discussion of the thesis' contributions in Chapters 3 to 7. Chapter 8 concludes the thesis and outlines future research directions.

1.2. Historical remarks

The research field of sequential decision making has a long history and is approached by a variety of disciplines. Many of its formal roots stem from the area of optimal control, such as the introduction of the Markov decision process (MDP) by Bellman (1957a) and solution approaches based on dynamic programming (Bellman, 1957b). While these approaches assume the dynamics of the environment to be known, the research field of reinforcement learning has had, historically, more emphasis on the (trial and error) learning aspect of decision making (Sutton and Barto, 2018). Nowadays, MDPs and methods from dynamic programming find wide usage in the domain of reinforcement learning.

Over the last years and decades, the combination of learning and sequential decision making has gained even more attention with the advent of powerful algorithms and learning methods for function approximators, subsumed under the term *machine learning*. It has, subsequently, found widespread applications in areas such as gameplay (Mnih et al., 2015; D. Silver et al., 2016; Vinyals et al., 2019), economics (Mosavi et al., 2020), and healthcare (C. Yu et al., 2021).

The application area most closely related to this thesis is that of controlling physical systems, in particular robotic systems (Kober and Peters, 2012; Kroemer et al., 2021).

Artificial intelligence Building sequential decision making agents is strongly related to *artificial intelligence*. While several interpretations of this term exist, for many authors, the concept of an agent which acts either humanly or rationally is a cornerstone of computational and artificial intelligence (Russell and Norvig, 2009). This thesis is focused on *rational agents*, in which the task to fulfill is formalized by the maximization of an objective function.*

1.3. Brief terminology primer

We follow terminology as in reinforcement learning literature (see, e.g, Sutton and Barto (2018)). More details can be found in Chapter 2.

An environment has an internal *state*. The state at the next timestep depends (through the environment's *dynamics*) on the current state and the *action* applied by the agent. Depending on the current state, the environment generates an *observation* with an observation function. An environment can be fully observable or partially observable. In the first case, the current state of the environment can be inferred from a single observation with full certainty — e.g., if the observation *is* the state. In the latter case, the agent estimates a belief state given past observations using a *state estimator*. Every action the agent takes in an environment is judged by a scalar reward, generated by the reward function. The goal of the agent is to act in a way which maximizes the expected discounted sum of rewards, i.e., the expected *return*. A *policy* outputs an action (or a distribution over actions for stochastic policies) given the current environment's state (in fully observable environments), or belief state (in partially observable environments). The policy can be parametric (e.g., a neural network) or non-parametric, e.g., a planner. A *value function* outputs, given the current environment's state (or belief state), and, optionally, the next action, the expected return when following a particular policy. The above setup can be formalized as a *Markov decision process* (Bellman, 1957a; Puterman, 1994).

We consider the environment's dynamics, observation function, and reward function as separate entities. In the general case, the policy and value function implicitly depend on the environment's dynamics and on the reward function.

* Regarding *acting humanly*, in particular, the *Turing test* (Turing, 1950), Russell and Norvig (2009) comment: "The quest for 'artificial flight' succeeded when the Wright brothers and others stopped imitating birds and started using wind tunnels and learning about aerodynamics. Aeronautical engineering texts do not define the goal of their field as making 'machines that fly so exactly like pigeons that they can fool even other pigeons.'"

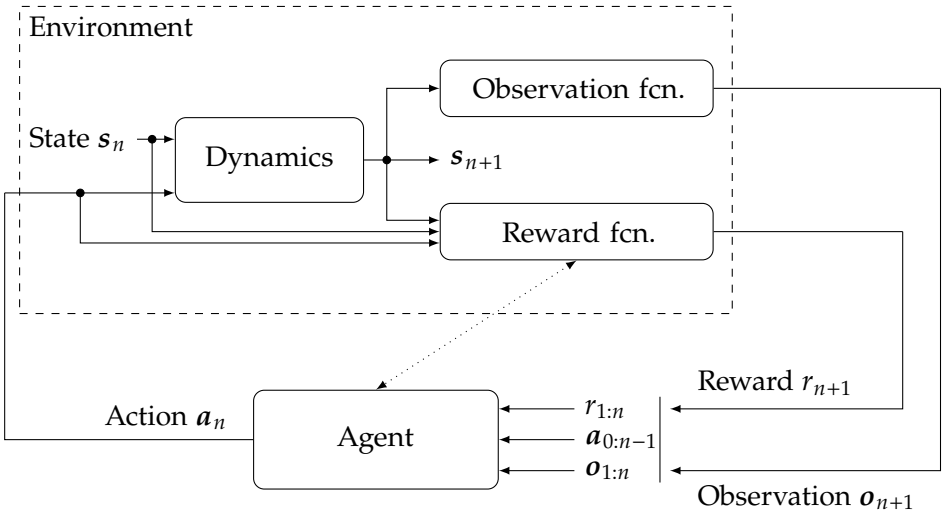


Figure 1.1.: Interplay between an environment and the agent. Based on a sequence of past rewards, actions, and observations, the agent sends the next action to the environment. In the partial observation setting, the state is not available outside the environment. The agent might have full access to the reward function, as denoted by the dotted line.

1.4. Challenges

Some challenges in sequential decision making are of general nature, which occur in all kinds of environments, regardless of whether it is purely virtual (e.g., a simulated physical system), or a real-world physical system (Subsection 1.4.1). Other challenges are especially prevalent in decision making for real-world physical systems, such as robots (Subsection 1.4.2).

1.4.1. General challenges

Credit assignment problem and sparse reward What makes sequential decision making challenging in general is the *credit assignment problem* (Minsky, 1961). It is hard to attribute an outcome to an action if the action's effect is apparent only very far in the future. The credit assignment problem is particularly pronounced in *sparse reward* settings, in which, in the extreme case, the agent is only rewarded once it has reached a final goal, after taking many sequential actions. Exemplarily, one of the first moves in a chess game can decide about a lose or win of the game, several tens or hundreds moves later.

Multiple dynamics and tasks In the standard Markov decision process formulation, an agent aims to optimally solve a single task on a single environment. A more generalist agent is able to act optimally on a set of dynamics and tasks. If the set of dynamics and tasks is known in advance and the agent is not expected to adapt to *novel* tasks, this is termed *multi-task reinforcement learning* in the reinforcement learning literature (T. Yu et al., 2019).

Novel dynamics and tasks As an extension to multi-task reinforcement learning, we can also require the agent to *generalize* to dynamics and tasks which were *unseen* during the initial learning phase. Such approaches are subsumed in the literature under the term *meta reinforcement learning* (T. Yu et al., 2019).

High-dimensional observations High-dimensional observations can be present in real-world and virtual settings. In classical control, sensor selection and -placement plays a central role in the control system design process (see, e.g., Skogestad and Postlethwaite (2005)). Typically, it is assumed that low-dimensional measurements of the system to control are available, such as angles and angular velocities of a robot's joints. However, for some systems, sensor placement is not feasible, and only observations in form of images or video is available.

1.4.2. Challenges in real-world environments

Sample efficiency Collecting data in the real world (e.g., from physical systems, or animal trials) is usually time- and cost-intensive. Different to a simulation, a real-world experiment cannot run faster than real time. Additionally, the experiment may fail and need human intervention to be reset. Also, hardware may fail and needs to be repaired or replaced. Consequently, the learning algorithms should be *sample-efficient*.

Non-stationary dynamics The physical world we are surrounded by is subject to a multitude of variations through biological, chemical and physical effects. Exemplary, over time, weather conditions may change, the behavior of a robotic system varies due to wear and tear of its hardware, or friction parameters change due to corrosion or biofilms on surfaces. These effects may render the agent's learned internal representations invalid, and require adaptation capabilities to novel dynamics.

Real-time control In highly dynamic real-world systems, the design of the decision rule and its implementation on hardware need to allow for *fast* decision making. Different to a simulation, the real world cannot be paused to compute a control action. Exemplarily, the cart-pole system by Jervis and Fallside (1992), used by Deisenroth and Rasmussen (2011), is controlled at a rate of $50\,\mathrm{s}^{-1}$.

Changes in the observation function A third component of the Markov decision process, next to the dynamics and reward function, which might be subject to change especially in real-world environments, is the observation function. Constructing an agent which can cope with changes in the observation function is especially important if images are used as inputs, as disturbances in image inputs are very common (e.g., different lighting conditions or different camera angles).

In this thesis, established concepts and novel approaches are discussed which address each of the above mentioned challenges to some extent, except *changes in the observation function*.

1.5. Established concepts and approaches

In this section, approaches from literature are reviewed, and their relation to the aforementioned challenges is expressed. Along the way, we will pose research questions which are discussed in this thesis.

1.5.1. Model-based vs. model-free

Approaches to sequential decision making can be classified as *model-based* and *model-free* approaches. In model-based approaches, an explicit representation of the transition dynamics of the environment is learned, while in model-free approaches, this is not the case. For the following discussions, we assume the environment to be fully observable or the state estimator to be independent of the dynamics. Otherwise, the state estimator has to provide the same adaptation capabilities as the dynamics model for the agent to be able to adapt to novel dynamics.

Model-free methods In model-free approaches, no component is learned or given which encodes information on the dynamics of the environment explicitly. Instead, the agent learns components which depend simultaneously on both the environment's dynamics and the rewards (such as a parametric policy or value function). Popular model-free approaches are built upon Q-learning (Watkins, 1989), such as Deep Q Networks (DQN) (Mnih et al., 2015), or policy gradient methods, such as Trust Region Policy Optimization (TRPO) (Schulman et al., 2015) and Proximal Policy Optimization (PPO) (Schulman et al., 2017), or a combination of both, such as Soft Actor-Critic (SAC) (Haarnoja et al., 2018). The main advantage of these approaches is that the decision rule (in form of a parametric policy or arg max-operation on the action value (Q) function) can be evaluated with low computational cost, enabling real-time control. As no explicit representation of dynamics is maintained, it is not straightforward to incorporate prior knowledge (e.g., through laws of physics), into the learning algorithm. Model-free methods cannot easily adapt to novel tasks or dynamics, as their solution representation entangles each of these components. A standard model-free agent has to be retrained (including costly interactions with the environment) if the task (reward function) changes, even if the environment's dynamics do not change. Model-free methods are known to be notoriously *sample-inefficient* (see, e.g., comparisons in Chua et al. (2018) and Hafner et al. (2020, 2019)).

Model-based methods Model based methods do maintain an explicit representation of (an approximation to) the environment's dynamics. Examples for such representations are forward dynamics models ($\boldsymbol{s}_n, \boldsymbol{a}_n \rightarrow \boldsymbol{s}_{n+1}$), reverse dynamics models ($\boldsymbol{s}_{n+1}, \boldsymbol{a}_n \rightarrow \boldsymbol{s}_n$), and inverse dynamics models ($\boldsymbol{s}_n, \boldsymbol{s}_{n+1} \rightarrow \boldsymbol{a}_n$). This thesis focuses on forward dynamics models. Model-based methods can further be separated into two branches: Model-based planning and model-based reinforcement learning approaches (we follow the definition by Moerland et al. (2023)). In the first concept, no parametric policy or value function is learned. In the second concept, a parametric policy or value function is learned on top of a dynamics model. As a representation of the dynamics is maintained separately, incorporating prior knowledge about the dynamics of the system is possible. In the extreme case, in environments such as board games, the dynamics are fully known in advance and can be incorporated accordingly (see, e.g., D. Silver et al. (2016)), increasing *sample-efficiency* of the learning process. Model-based methods were shown to be very sample efficient compared to model-free methods, even with only minimal assumptions made on the dynamics of the environment (e.g., Chua et al. (2018), Deisenroth and Rasmussen (2011), and Hafner et al. (2019)). What is termed *model learning* here is strongly related to *system identification* in the (optimal) control community (see Ljung (1986) for an overview).

Model-based planning In model-based planning methods, learned representations depending on the dynamics and the task (reward) are strictly separated. A planner leverages the dynamics

and reward model (or reward function) for planning an action sequence. Advantages of this approach are that (i) the dynamics model and reward model can be adapted separately to novel dynamics and tasks, and (ii) no entangled representations, such as the policy or value function, have to be retrained once the dynamics or task have changed. As a limitation to (i) it is to mention that separate adaptation is only possible if the dynamics model remains to capture the parts of the state space relevant for the task with sufficient accuracy. For complex systems and tasks, in many approaches, only *local* dynamics models are learned, which capture dynamics relevant for a particular task, e.g., through data collection on task-oriented execution of the planner (as in Hafner et al. (2019)). A disadvantage of the model-based planning approach is that planning is a compute-intensive operation and meeting real-time requirements can be challenging. Exemplary methods are Embed2Control (Watter et al., 2015), PlaNet (Hafner et al., 2019), and PETS (Chua et al., 2018).

Model-based reinforcement learning Model-based reinforcement learning combines learning an explicit representation of the environment's dynamics (or leveraging known dynamics) with components which implicitly depend on both the dynamics and the task, such as a parametric policy or value function. Three principled variants of model-based reinforcement learning can be distinguished.

- In the first variant, the model is known, and a value function is learned as a heuristic for planning. This methodology is used exemplarily by D. Silver et al. (2016).
- The second variant learns a parametric policy and/or value function from combined experience which stems from the actual environment and from the learned model (hypothetical experience). This family of approaches dates back to the *Dyna* algorithm (Sutton, 1991).
- As a third variant, a family of approaches was developed which leverage the differentiability of the learned model for updating parameters of a parametric policy through gradient ascent on the cumulative reward. Exemplary methods for this are *PILCO* (Deisenroth and Rasmussen, 2011) and *Dreamer* (Hafner et al., 2020).

Adapting such models to changes in the environment's dynamics and task are not as straightforward as in model-based planning due to the existence of components which hold entangled representations of dynamics and task (parametric policy, value function). However, after adapting the dynamics and/or reward model or replacing the reward function, retraining the entangled representations might be possible leveraging the learned models, without additional data collection on the environment. As the decision rule is expressed by a reactive parametric policy and no time-intensive planning is required, model-free reinforcement learning approaches are less challenging to apply for real-time control tasks than model-based planning methods.

1.5.2. Meta (reinforcement) learning

As outlined in Section 1.4, being adaptable to changes in the task and environment's dynamics (e.g., caused by effects present in the real world) is a desirable property of decision making agents. Also, such agents should be able to adapt with minimal additional data, due to the cost incurred with data collection in the real world, and with minimal compute required for retraining.

Such adaptation problems can be addressed with the methodology of *meta-learning*. The general idea of meta-learning is *learning to learn* (Thrun and Pratt, 1998). It has found broad application in machine learning and neural networks, also outside the domain of sequential decision making (see Hospedales et al. (2022) for an extensive survey). In an initial stage, a meta model is learned. This model is not designed to perform optimally on a particular task, but designed in a way *such that it can data- and compute-efficiently adapt to a target task*. Exemplarily, Finn et al. (2017) proposed two meta-learning schemes for general neural network models, without particular focus on decision making problems. They have in common that, for adaptation, weights of the model are updated. The updates are computed either by the method of steepest descent, or through a recurrent neural network.

In *meta reinforcement learning* (see, e.g., T. Yu et al. (2019) for a definition), a meta-agent is learned in a way such that it can efficiently adapt to its target task and environment. For learning, a distribution of tasks and environments is considered. A meta-agent, which is shared among all tasks and environments, is learned to perform optimally without (zero-shot) or with minimal (few-shot) additional interactions on a previously unseen test environment.

As outlined in Subsection 1.5.1, adapting to novel tasks (in form of given reward functions) is rather straightforward for model-based planning agents, given the dynamics model is sufficiently accurate for the tasks considered. For novel dynamics, the dynamics model needs to be adapted, which is what we will discuss in Subsection 1.5.3.

1.5.3. Adaptive dynamics models

An adaptive dynamics model can leverage learned structures to adapt to novel dynamics without the need for expensive additional data collection or model retraining. One approach to adaptivity are context-conditional models, in which predictions depend on contextual observations. An example for such a model is the Gaussian process dynamics model used by Deisenroth and Rasmussen (2011). However, Deisenroth and Rasmussen (2011) used physical measurements as observations and did not consider high-dimensional observations.

This leads us to the first research question discussed in this thesis:

Research question 1 *Can we learn a Gaussian process dynamics model from images, and how well does it adapt to changes in the dynamics?*

We will detail this question in Chapter 3.

Learning adaptive dynamics models can also be explicitly formulated as a *meta-learning* problem (see Subsection 1.5.2). A meta-dynamics model is learned on a distribution of environments with the ability to adapt, in a sample-efficient manner, specifically to a particular environment instance. The two weight-adaptation schemes proposed by Finn et al. (2017) were successfully applied by Nagabandi et al. (2019) to model-based planning problems. Another approach to adaptive models is to capture its variations in a global latent variable. A general approach to this, formulated in the context of neural networks, are methods from the Neural Process (NP) family (Garnelo et al., 2018b). An exemplary application of a NP dynamics model in model-based reinforcement learning is proposed by Lee et al. (2020), in which a set of past observations modulate the dynamics model through a context variable. A Gaussian process latent variable

model with applications in model-based reinforcement learning was proposed by Sæmundsson et al. (2018).

Related to the problem of an adaptive model is the question how *informative* data should be collected on a novel environment to allow the model to adapt to the actual dynamics. We will discuss this question in Subsection 1.5.4.

1.5.4. Active inference for efficient data collection

The problem of efficient data collection concerns all sequential decision making agents which incorporate some form of data and where data collection incurs any kind of cost. To obtain a better decision making strategy, an agent may even act temporarily suboptimally with respect to its task, in order to collect additional data. Thus, over its lifetime, an agent has to decide between (i) acting optimally with respect to the task, given its current knowledge, and (ii), acting suboptimally with respect to the task, but gaining additional knowledge about the environment dynamics and task (in form of rewards). This is known as the *exploration-exploitation tradeoff* (see, e.g., Sutton and Barto (2018) for a more in-depth discussion). In a nutshell: During the time the agent is busy with data collection it is likely to act suboptimally with respect to its task. Thus, collecting new data should be efficient.

Let us look at this problem from the model-based planning perspective. In order to be able to solve a wide variety of tasks, we aim for a dynamics model which accurately captures the actual dynamics of the system for large parts of the state space. To this end, we need to perform *active inference* in form of experiments in the environment. This is achieved by executing action sequences which excite the environment in a way which is *maximally informative*, i.e., reveals maximum information on the dynamics of the environment. When learning from scratch, the informativeness of actions is determined by what has already been observed, and the structure which is assumed for the dynamics model. For dynamical systems, we need to consider that we cannot query the environment arbitrarily, but are constrained by its dynamics. Thus, finding a maximally informative sequence of actions is again a sequential decision making problem in itself. Buisson-Fenet et al. (2020) formulate such an active inference scheme for learning Gaussian process forward dynamics models from scratch. Sekar et al. (2020) develop a similar method for active learning of a forward model implemented with recurrent neural networks. In the above two approaches, the structure of the dynamics model is defined a-priori through the Gaussian process with its respective mean- and covariance function or the recurrent neural network design.

In many practical cases, however, it can be assumed that the dynamics to model are from some environment from a particular distribution of environments, e.g., inverted pendulums with a pole mass from a specific interval.

With the above, we reach the next research question:

Research question 2 *Instead of leveraging an assumed structure on the environment dynamics, can we learn structure over a distribution of environments, and perform active inference to identify the dynamics of a particular environment instance?*

We will detail this question in Chapter 4.

To approach the above research question, we leverage a context-conditional dynamics model in Chapter 4. The dynamics model contains a latent variable which is inferred from contextual observations to adapt to varying dynamics.

In terrain-aware autonomous robot navigation, certain aspects of the environment are subject to change. This does not only include intrinsic parameters of the robot, such as actuator gains, but also the ground surface (Sonker and Dutta, 2021). Consequently, the agent is not only required to adapt to varying dynamics of the robot, but also to different terrain surfaces depending on the robot's location.

This leads us to the research question:

Research question 3 *Can challenges present in the domain of terrain-aware autonomous robot navigation be tackled with context-conditional dynamics models as presented in Chapter 4?*

We will detail this question in Chapter 5.

1.5.5. Incorporating structure

As mentioned in Subsection 1.5.1, in model-based methods, we can incorporate a-priori knowledge on the dynamics of the environment. In the extreme case, the model is known without the need to incorporate data, for example, as rules of a board game, as in D. Silver et al. (2016). In system identification terminology (see, e.g., Nelles (2001)), such models are called *white-box* models. The other extreme are *black-box* models, in which the model is solely identified from data. From a learning-theory perspective, such models are typically not useful. When fitting a function $f : \mathcal{X} \to \mathcal{Y}, f \in \mathcal{F}$ to a finite set of datapoints $\{\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)}\}_{k=1}^{K}$, without making any restricting assumptions on the function class $\mathcal{F}$ (such as smoothness), no conclusions can be made on the value of the function at unseen points $\boldsymbol{x}^*$ (Luxburg and Schölkopf, 2011). Thus, formally, a truly black box model will only be a lookup table. Models for which some structure is assumed, and, additionally, data is used to identify parameters, are called *gray-box* models. Consequently, nearly all models in which parameters are identified from data are *gray-box* models. Colloquially, the term *black-box model* is also used for models which only make minimal assumptions on the model class, e.g., expressive neural network models. This meaning is adopted in this thesis.

Investigating the properties of blackbox- and graybox methods on a particular task is a central element of the next research question:

Research question 4 *For the particular task of tracking and prediction the motion of a table tennis ball for return with a robotic arm, what are the properties of a physics-informed gray-box method, compared to non-physics informed black-box methods?*

We will detail this question in Chapter 6.

1.5.6. Hierarchy

Incorporating hierarchy into the design of the decision making agent aims at *decomposing* a complex problem into simpler subproblems. A (sub-)agent is rewarded for achieving a subgoal or successfully fulfilling a subtask. By this, credit assignment and sparse reward problems are

attenuated. The need for *hierarchies* in reinforcement learning agents to cope with this problem in long-horizon decision-making settings has already been discovered in early works such as (Dayan and Hinton, 1992; Sutton et al., 1999; Watkins, 1989).

Long-horizon sparse-reward tasks are problematic to both model-free and model-based agents. An agent, which updates its behavior only based on tasks rewards, will, in consequence, not update its behavior when only observing a reward of 0. An agent which acts randomly will continue to act randomly, and the probability of solving the task is that of random chance.

To this end, hierarchical methods were developed which consist of agents on multiple layers. A higher-level agent commands a lower-level agent to fulfill subtasks or reach subgoals. The lower-level agent is rewarded when reaching the subtask, independent of the task reward, and can thus update its behavior accordingly. Thus, meaningful behavior is learned even in absence of task rewards. For the higher-level agent, an action now corresponds to commanding a lower-level agent, effectively reducing the decision horizon of the higher-level agent. Learned subtask solutions can now be chained by the higher-level agent to achieve complex tasks.

In Subsection 1.5.7 we detail methods to learn meaningful behavior even in absence of task rewards.

1.5.7. Learning without task reward

Learning without a task-specific reward is coined *task agnostic learning* in reinforcement learning literature. Task-specific, extrinsic reward is distinguished from a task-agnostic, intrinsic reward.

Task-agnostic pre-training of agent components finds its applications in meta reinforcement learning, accelerating the ability of an agent to adapt to a specific task (e.g., in Sekar et al. (2020)). Also, it is used for pre-training low-level agents in a hierarchical setup. In Sharma et al. (2020), a set of diverse *skills* is trained using an intrinsic reward derived from an information theoretic criterion. A skill is a parametric policy augmented by an additional parameter which modulates its behavior. Jointly, a single-step forward model is learned, modeling the skill dynamics, which can be used in a hierarchical setup for *planning* over skills. However, as the forward model only models single-step transitions, the problem of error accumulation still exists for long-horizon tasks.

This leads us to the final research question discussed in this thesis:

Research question 5 *Can we learn a multi-step forward model and a set of skills jointly for model-based planning on the high-level and fast, reactive control on the lower level? How well does it perform on long-horizon sparse-reward settings?*

We will detail this question in Chapter 7.

1.6. Thesis contributions

In relation to the research questions posed in the previous section, in the following the contributions of this thesis are summarized.

1. In Chapter 3 we present a non-parametric approach to adaptive dynamics models. The dynamics of the environment are modeled in the latent space of a variational autoencoder using Gaussian processes. This allows the model to generalize to previously unseen environment variations such as modified mass and actuator gain parameters of an inverted pendulum. We demonstrate that, for adaptation, it requires fewer data points than a deep learning based baseline model.

2. Aside from formulating adaptive models, we are interested in exciting an environment in a way which is *informative* about the variations present to perform system identification, with the assumption that the environment is from an assumed distribution of environments. In Chapter 4, we interpret this as a problem of Bayesian optimal experimental design (BOED, Chaloner and Verdinelli (1995) and Lindley (1956)), and propose an according identification/calibration algorithm. We show that we obtain a more accurate dynamics model after calibration when using actions to excite the environment proposed by our approach, compared to random actions.

3. In terrain-aware autonomous robot navigation, not only intrinsic parameters of the robot are subject to change, but also the ground surface. In Chapter 5 we present a terrain- and robot-aware dynamics model (TRADYN), which can adapt to changes in intrinsic robot parameters and leverage terrain information for (throttle-) energy-efficient navigation.

4. In Chapter 6 we present a state-of-the-art approach for tracking and predicting the motion of a table tennis ball, incorporating effects caused by the ball's spin and impact with the table. Our approach outperforms existing deep-learning-based baselines through identifying parameters of a physics-based model from data through a filtering process. Our learning-based formulation allows adapting to different settings of the ball launcher to infer the initial spin of the ball, improving the predictive accuracy of the model even further.

5. In Chapter 7 we do not investigate the problem of adaptive models, but rather the issue of solving problems which require long-horizon planning and reactive control. We consider physically embedded board games as a class of problems requiring these two capabilities. Those are challenging for model-free reinforcement learning approach due to the combinatorial complexity of the state space, and challenging for model-based planning due to the accumulation of error in forward model predictions. We present an approach termed SEADS combining model-based planning and model-free reinforcement learning. SEADS is able to solve physically embedded board games with high success rates, in contrast to the baselines considered.

1.7. Publications

Large parts of this thesis have been published in peer-reviewed conference proceedings. Each chapter mentioned in Section 1.6 relates to one of the *main* publications (i) — (v) listed below. The publications are given in the order of chapters of this thesis.

During the course of my PhD, I have contributed to further works in the domain of dynamics model learning and model-based planning (*side* publications). However, these works are not considered as integral parts of this thesis.

1.7.1. Main publications

(i) N. Bosch, **J. Achterhold**, L. Leal-Taixé, and J. Stückler (2020). 'Planning from Images with Deep Latent Gaussian Process Dynamics'. In: *Proceedings of the Learning for Dynamics and Control Conference (L4DC)*. Chapter 3.

(ii) **J. Achterhold** and J. Stueckler (2021). 'Explore the Context: Optimal Data Collection for Context-Conditional Dynamics Models'. In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*. Chapter 4.

(iii) S. Guttikonda, **J. Achterhold**, H. Li, J. Boedecker, and J. Stueckler (2023). 'Context-Conditional Navigation with a Learning-Based Terrain- and Robot-Aware Dynamics Model'. In: *Proceedings of the IEEE European Conference on Mobile Robots (ECMR)*. Chapter 5.

(iv) **J. Achterhold**, P. Tobuschat, H. Ma, D. Büchler, M. Muehlebach, and J. Stueckler (2023). 'Black-Box vs. Gray-Box: A Case Study on Learning Table Tennis Ball Trajectory Prediction with Spin and Impacts'. In: *Proceedings of the Learning for Dynamics and Control Conference (L4DC)*. Chapter 6.

(v) **J. Achterhold**, M. Krimmel, and J. Stueckler (2022). 'Learning Temporally Extended Skills in Continuous Domains as Symbolic Actions for Planning'. In: *Proceedings of the Conference on Robot Learning (CoRL)*. Chapter 7.

1.7.2. Side publications

- C. Pinneri, S. Sawant, S. Blaes, **J. Achterhold**, J. Stueckler, M. Rolínek, and G. Martius (2020). 'Sample-efficient Cross-Entropy Method for Real-time Planning'. In: *Proceedings of the Conference on Robot Learning (CoRL)*.

  The cross-entropy method planning algorithm (CEM, Rubinstein (1999)) finds widespread use in the field of (model-based) planning for robotics and dynamical systems, also in the remainder of this thesis. Its computational complexity however poses a challenge in applying CEM to real-time planning tasks. Pinneri et al. (2020) develop several improvements to CEM, termed iCEM, to reduce its computational footprint and allow use in real-time applications. I have contributed to (Pinneri et al., 2020) by evaluating iCEM on the task of planning from pixels (Hafner et al., 2019).

- R. K. Kandukuri, **J. Achterhold**, M. Möller, and J. Stueckler (2020). 'Learning to Identify Physical Parameters from Video Using Differentiable Physics'. In: *Proceedings of the German Conference on Pattern Recognition (GCPR) and*

  R. K. Kandukuri, **J. Achterhold**, M. Möller, and J. Stueckler (2022). 'Physical Representation Learning and Parameter Identification from Video Using Differentiable Physics'. In: *International Journal of Computer Vision* 130.1.

  Using *differential physics simulators* (e.g. Amos et al. (2018)) as forward models has several advantages over black-box models, e.g. (recurrent) neural networks. It allows for physical interpretability of parameters and latent states, and, due to imposing more structure on the learning problem, may require fewer data to be trained. However, this comes at the expense of the need to define the problem structure beforehand, i.e., to define the physical

scene. In his master's thesis, Rama Kandukuri formulated and implemented algorithms for identifying physical parameters (e.g., masses and friction coefficients) of objects from video data. The thesis was concucted under advisory by Jörg Stückler and me, and examined by Michael Möller at the University of Siegen. The work of this thesis led to the publications (Kandukuri et al., 2020) and (Kandukuri et al., 2022).

1.8. Open-source software releases

In order to allow other researchers to reproduce and build upon our work, we have published the software and data related to results presented in this thesis for public access on GitHub.

- Chapter 3, Bosch et al. (2020): Planning from Images with Deep Latent Gaussian Process Dynamics.
  `https://github.com/EmbodiedVision/dlgpd`.
- Chapter 4, Achterhold and Stueckler (2021): Explore the Context: Optimal Data Collection for Context-Conditional Dynamics Models.
  `https://github.com/EmbodiedVision/explorethecontext`.
- Chapter 5, Guttikonda et al. (2023): Context-Conditional Navigation with a Learning-Based Terrain- and Robot-Aware Dynamics Model.
  `https://github.com/EmbodiedVision/tradyn`
- Chapter 6, Achterhold et al. (2023): Black-Box vs. Gray-Box: A Case Study on Learning Table Tennis Ball Trajectory Prediction with Spin and Impacts.
  `https://github.com/EmbodiedVision/tabletennis-spin-impacts`
- Chapter 7, Achterhold et al. (2022): Learning Temporally Extended Skills in Continuous Domains as Symbolic Actions for Planning.
  - Environment implementations:
    `https://github.com/EmbodiedVision/seads-environments/`
  - Agent implementation:
    `https://github.com/EmbodiedVision/seads-agent`

2. Preliminaries

Before presenting technical background on particular topics, we introduce the concept of *probabilistic graphical models* in Section 2.1, used widely in this thesis. Further important concepts are the Markov process (Subsection 2.2.1) and the related state-space model (Subsection 2.2.2).

The overarching theme of this thesis are sequential decision making problems, basics of which we introduce in Section 2.3, in combination with methods from the field of machine learning, briefly introduced in Section 2.4. Section 2.5 is dedicated to latent variable models.

In model-based planning, model learning and planning are intertwined to solve problems of sequential decision making. In Section 2.6 we give a short overview on the planning methods used in this thesis.

2.1. Probabilistic graphical models

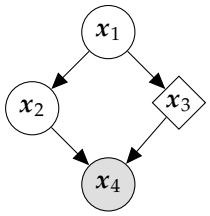
We make repeated use of probabilistic graphical models in this thesis, in particular, Bayesian networks (for reference, see Murphy (2012, chap. 10) or Bishop (2007, chap. 8.1)). A Bayesian network is a directed acyclic graph visualizing a factorization of the joint probability distribution of a set of random variables. The nodes of the graph correspond uniquely to random variables, while edges indicate conditional dependence. Let us, for simplicity, name all random variables in the graphical model $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_V)$ where V is the number of variables. Then, the joint distribution is given by the product of conditional distributions

$$p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_V) = \prod_{v=1}^{V} p(\boldsymbol{x}_v \mid \{\boldsymbol{x}_{v'} \mid v' \in \mathrm{Pa}(v)\}) \tag{2.1}$$

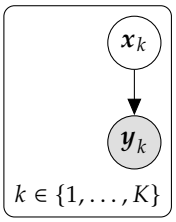
where $\mathrm{Pa}(v)$ are the parent nodes of node v.

We distinguish different types of nodes by visual features. It is important to note that for the factorization of the joint distribution, these visual features are irrelevant. *Observed* random variables are depicted with a shaded background. In contrast, *latent* or *unobserved* random variables have a plain background. A circle depicts a general random variable. A rhombus denotes a special variable which depends deterministically on its parent nodes, i.e., a deterministic mapping $\boldsymbol{x}_v = f(\{\boldsymbol{x}_{v'} \mid v' \in \mathrm{Pa}(v)\})$ exists. We refer to Figure 2.1a for an example.

The plate notation can be related to a generator expression for set of random variables. For each node within the plate, an instance exists for each instance index in the index set given at the bottom of the plate. See Figure 2.1b for an example.



(a) Bayesian network with two latent nodes x_1, x_2, one deterministic node x_3, and one observed node x_4. Joint distribution $p(x_1, \dots, x_4) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1)p(x_4 \mid x_2, x_3)$.



(b) Bayesian network with plate notation. Joint distribution $p(x_1, \dots, x_K, y_1, \dots, y_K) = \prod_{k=1}^{K} p(x_k)p(y_k \mid x_k)$.

Figure 2.1.: Two exemplary Bayesian networks. See Section 2.1 for details.

2.2. Markov processes

2.2.1. Markov process

Mathematically, sequential decision making problems are typically formalized as Markov decision processes. Before introducing the Markov decision process in Subsection 2.3.1, we will first introduce the Markov process, on which the definition of a Markov decision process builds upon.

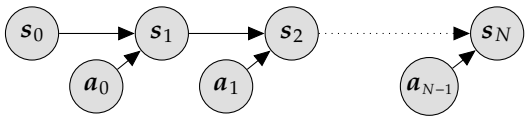


Figure 2.2.: Directed graphical model of an action-conditioned Markov process with states s_n and actions a_n.

Consider the action-conditioned stochastic process given by the directed graphical model in Figure 2.2. We can factorize the joint distribution over states and actions as $p(s_0, \dots, s_N, a_0, \dots, a_{N-1}) = p(s_0) \prod_{n=1}^{N} p(s_n \mid s_{n-1}, a_{n-1})p(a_{n-1})$. The depicted process is termed (first-order) *Markov process** as the probability distribution on s_n (for any $n \in \{1, \dots, N\}$) becomes independent of states $\{s_m \mid m < n-1\}$ and actions $\{a_m \mid m < n-1\}$ when conditioned on s_{n-1} and a_{n-1}, i.e.

$$\begin{aligned} &p(s_n \mid s_0, \dots, s_{n-1}, a_0, \dots, a_{n-1}) \\ &\quad = \frac{p(s_0, \dots, s_n, a_0, \dots, a_{n-1})}{p(s_0, \dots, s_{n-1}, a_0, \dots, a_{n-1})} \\ &\quad = \frac{p(s_0) \prod_{k=1}^{n} p(s_k \mid s_{k-1}, a_{k-1})p(a_{k-1})}{p(s_0) \prod_{k=1}^{n-1} p(s_k \mid s_{k-1}, a_{k-1})p(a_{k-1})p(a_{n-1})} \\ &\quad = p(s_n \mid s_{n-1}, a_{n-1}). \end{aligned}$$

As a consequence, for predicting the state distribution for timestep N given past states and actions, i.e. $p(s_N \mid s_0, \dots, s_{N-1}, a_0, \dots, a_{N-1})$, it suffices to observe the state and action at the immediately preceding timestep s_{N-1}, a_{N-1}. State and action observations for timesteps $n < N-1$ do not convey additional information on the conditional distribution of s_N.

* One also says the process is Markov or fulfills the Markov property.

2.2.2. State-space models

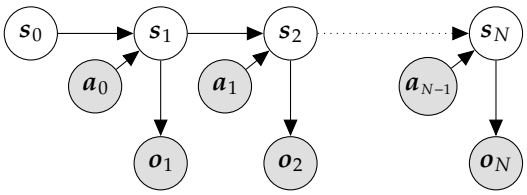


Figure 2.3.: Directed graphical model of a state-space model with latent states s_n, observations o_n, and actions a_n.

An extension of the fully observable Markov process described in Subsection 2.2.1 to partial observability is the *state-space model*. The sequence of random variables $(s_0, \ldots, s_N)$ is distributed according to a Markov process, and conditioned on actions, i.e. $p(s_0, \ldots, s_N, a_0, \ldots, a_{N-1}) = p(s_0) \prod_{n=1}^{N} p(s_n \mid s_{n-1}, a_{n-1}) p(a_{n-1})$. The variables $(s_0, \ldots, s_N)$ are *latent*, i.e., unobserved. However, the random variables $(o_1, \ldots, o_N)$ are observed, with each o_n conditionally distributed as $p(o_n \mid s_n)$. We assume states, actions and observations being real-valued vectors in this section.

The state-space model is characterized by the *initial state density* $p(s_0)$, the *transition model* $p(s_n|s_{n-1}, a_{n-1})$, and the *observation model* $p(o_n|s_n)$. Typically, densities of interest are conditioned on actions $a_0, \ldots, a_{N-1}$, and no particular density $p(a_0, \ldots, a_{N-1})$ on actions is assumed.

The state-space model can constitute a joint transition/observation model for partially observable Markov decision processes (see Subsection 2.3.2). As outlined in Subsection 2.3.4, action decisions in POMDPs are made based on the *belief* of a state s_n, which incorporates all past actions and observations. Thus, we are particularly interested in computing the density $p(s_n \mid o_{\leq n}, a_{0:n-1})$, which is called *filtering* density. By $o_{\leq n}$ we denote that some observations may be unavailable. If all observations are available, $o_{\leq n} = o_{1:n}$. Also, in particular for solving MDPs/POMDPs with planning (see Subsection 2.3.5), we are interested in *prediction* densities of the form $p(s_n \mid o_{<n}, a_{0:n-1})$.

2.3. Sequential decision making

2.3.1. Markov decision process

In a Markov decision process (MDP) (Bellman, 1957a; Puterman, 1994), it is assumed that the transition dynamics of the environment are a Markov process (Subsection 2.2.1). In order to specify a sequential decision making problem, the MDP does not only describe the transition dynamics, but also, encoded through the *reward function*, a particular task to be solved. An *agent*, being an (approximate)† solution to an MDP, aims to maximize future rewards‡.

MDP on finite sets of states and actions We present the definition by Otterlo and Wiering (2012). A discrete-time Markov Decision Process (MDP) on finite sets of states and actions is defined as a four-tuple $(\mathcal{S}, \mathcal{A}, T(s_{n+1}, a_n, s_n), R(s_n, a_n, s_{n+1}))$ where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $T(s_{n+1}, a_n, s_n) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ with $\sum_{s_{n+1} \in \mathcal{S}} T(s_{n+1}, a_n, s_n) = 1$ the

† *Approximate* in the sense of *potentially non-optimal*

‡ This is kept vaguely purposefully here, we refer to the paragraph *Optimality criteria* for more details.

transition function and $R(\boldsymbol{s}_n, \boldsymbol{a}_n, \boldsymbol{s}_{n+1}) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ the reward function. The process is termed *Markov* as the transition function is fully determined by the current state and action and does not depend on further states or actions from the past. Similarly, the reward function only depends on the current state and a single past state and action. With a finite set of states, the transition function encodes the probability of reaching state $\boldsymbol{s}_{n+1}$ when applying action $\boldsymbol{a}_n$ in state $\boldsymbol{s}_n$. The reward function outputs the reward for a particular transition $(\boldsymbol{s}_n, \boldsymbol{a}_n, \boldsymbol{s}_{n+1})$. The agent chooses actions based on a *policy*, which we will introduce later.

MDP on continuous state and action spaces We follow the introduction by Hasselt (2012) for MDPs on state spaces which are subsets of the space of real-valued vectors of dimension D_S, i.e., $\mathcal{S} \subseteq \mathbb{R}^{D_S}$, which is a continuous space. For the following considerations, it is not relevant whether the action space is a finite set or also a subset of real-valued vectors; we will denote it with $\mathcal{A}$. For a continuous state space, the transition dynamics are given by a conditional probability density function $p(\boldsymbol{s}_{n+1} \mid \boldsymbol{s}_n, \boldsymbol{a}_n)$. The reward function is again defined as $R(\boldsymbol{s}_n, \boldsymbol{a}_n, \boldsymbol{s}_{n+1}) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. We deviate from the definition by Hasselt (2012) and for now define the MDP as four-tuple $(\mathcal{S}, \mathcal{A}, p, R)$. The definition by Hasselt (2012) contains a mandatory discount factor γ (see below).

Optimality criteria In order to solve a decision making problem formulated as an MDP, additional information about *how exactly* the agent should maximize its future rewards is required. Typically, an agent aims at maximizing an expected discounted cumulative reward, i.e. expected return, on a finite-time horizon of length N, i.e.,

$$\mathcal{J} = \mathbb{E}\left[\sum_{n=0}^{N-1} \gamma^n R(\boldsymbol{s}_n, \boldsymbol{a}_n, \boldsymbol{s}_{n+1})\right], \tag{2.2}$$

or an infinite-time horizon

$$\mathcal{J} = \mathbb{E}\left[\sum_{n=0}^{\infty} \gamma^n R(\boldsymbol{s}_n, \boldsymbol{a}_n, \boldsymbol{s}_{n+1})\right]. \tag{2.3}$$

The discount factor $\gamma \in [0, 1]$ controls the far-sightedness of the agent. A small discount factor encourages the agent to seek for reward in the near future, while for a discount factor approaching 1, the agent cares less how far the reward is obtained in the future. A discount factor $\gamma < 1$ ensures convergence of the geometric series in Equation 2.3 even for a constant reward (Sutton and Barto, 2018). As the horizon and discount factor have influence on the solution of the MDP, commonly, the MDP four-tuple is extended by the discount factor, and, if applicable, the time horizon.

Policy The solution to an MDP may be obtained and represented in various ways. All solutions have in common, that, at the final stage, a *policy* decides on the action $\boldsymbol{a}_n$ to take in state $\boldsymbol{s}_n$. The policy might be parametric (e.g., a neural network), or non-parametric, e.g., a planner. In the general case, action selection is stochastic, i.e., $\boldsymbol{a}_n \sim \pi(\boldsymbol{a}_n \mid \boldsymbol{s}_n)$, where π is a probability mass function (for a finite set of actions) or probability density function (for a continuous space of actions). In Subsection 2.3.3 we will detail several approaches on how to obtain and represent a policy for sequential decision making problems formulated as Markov devision processes.

At this point, please note the concept of a *random* policy, which is not to be confused with a stochastic policy. While typically not obtained as a solution to an MDP, a random policy randomly selects actions from the action space, e.g., for a real-valued action bounded between $[-a, a]$, $a_n \sim \mathcal{U}[-a, a]$. A random action is commonly used to collect data from an environment without focusing on a particular task.

2.3.2. Partially observable Markov decision processes

In physical systems, the underlying Markovian state $\boldsymbol{s}_n$ usually cannot be directly observed to perform action selection with a policy. Rather, only observations $\boldsymbol{o}_n$ in form of noisy measurements are available. A sequential decision making problem in which the environment's state $\boldsymbol{s}$ is not fully observable (as in Subsection 2.3.1) can be formalized as *partially observable Markov decision process* (POMDP) (Kaelbling et al., 1998). Formally, one must again distinguish between a finite set of observations $\mathcal{O}$ and a continuous space of observations $\mathcal{O} \subseteq \mathbb{R}^{D_\mathrm{O}}$. In the first case, the observation function is defined as $O(\boldsymbol{o}_n, \boldsymbol{s}_n, \boldsymbol{a}_{n-1}) : \mathcal{O} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ with $\sum_{\boldsymbol{o}_n \in \mathcal{O}} O(\boldsymbol{o}_n, \boldsymbol{s}_n, \boldsymbol{a}_{n-1}) = 1$ (Kaelbling et al., 1998). In the latter case, the observation function is a probability density function on the observation space, conditioned on state and action, i.e. $p(\boldsymbol{o}_n \mid \boldsymbol{s}_n, \boldsymbol{a}_{n-1})$ (Spaan, 2012). For notational simplicity we will always refer to the observation function as $p(\boldsymbol{o}_n \mid \boldsymbol{s}_n, \boldsymbol{a}_{n-1})$, being a probability mass function or probability density function, depending on the context. In this thesis, we assume the observation function to be independent of the action, i.e., given by $p(\boldsymbol{o}_n \mid \boldsymbol{s}_n)$ (see Figure 1.1).

2.3.3. Solution strategies to MDPs

With the theory of Markov decision processes, we now have a formal background for the sequential decision making strategies introduced in Subsection 1.5.1, which were

- model-based planning,
- model-based reinforcement learning, and
- model-free reinforcement learning.

In Subsection 2.3.5 we will detail background on model-based planning, and in Subsection 2.3.6 central ideas of reinforcement learning, which are relevant for this thesis.

2.3.4. Solution strategies to POMDPs

As the state is no longer directly observable in POMDPs, action selection has to take place depending on the history of observations $\boldsymbol{o}_{1:n}$ and actions $\boldsymbol{a}_{0:n-1}$. Past observations and actions can be absorbed into a *belief state*. For a finite set of states, the belief state is a probability mass function over the set of states (Kaelbling et al., 1998). For a continuous state space, the belief state is a probability density on the state space (Kochenderfer, 2015), represented by, e.g., its moments (mean and covariance for a Gaussian distribution). The process of forming a belief state from past observations and actions is called *filtering*, with $p(\boldsymbol{s}_n \mid \boldsymbol{o}_{1:n}, \boldsymbol{a}_{0:n-1})$ being the *filtering distribution*. In contrast to the observations, the belief state fulfills the Markov property, such that solution approaches to MDPs can be extended to POMPDs when using the belief state as a Markov state.

2.3.5. Model-based planning

Model-based planning approaches to sequential decision making problems are two-staged (see, e.g., Chua et al. (2018), Hafner et al. (2019), and Moerland et al. (2023)).

In the first stage, the dynamics of the environment are captured with a *dynamics model*. This typically entails learning a forward dynamics model of the form $q(\boldsymbol{s}_{n+1} \mid \boldsymbol{s}_n, \boldsymbol{a}_n)$, which approximates $p(\boldsymbol{s}_{n+1} \mid \boldsymbol{s}_n, \boldsymbol{a}_n)$. If the mapping from states to rewards is unknown, a reward model can also be learned via function approximation, $\widetilde{r}(\boldsymbol{s}_n, \boldsymbol{a}, \boldsymbol{s}_{n+1}) \approx R(\boldsymbol{s}_n, \boldsymbol{a}, \boldsymbol{s}_{n+1})$. We will denote both a known mapping and function approximator by $\widetilde{r}$ in the following.

In the second stage, a *planner* optimizes an objective function $\mathcal{J}$ (e.g., expected return) over a sequence of H actions from the action space starting from an initial state distribution $p_n(\boldsymbol{s}_n)$, i.e.

$$(\boldsymbol{a}_n^*, \dots, \boldsymbol{a}_{n+H-1}^*) = \max_{(\boldsymbol{a}_n, \dots, \boldsymbol{a}_{n+H-1}) \in \mathcal{A}^H} \mathcal{J}((\boldsymbol{a}_n, \dots, \boldsymbol{a}_{n+H-1}), p_n, q, \widetilde{r}). \tag{2.4}$$

The variable H is termed the *horizon* of the planning problem. For details on planning algorithms which implement Equation 2.4 we refer to Section 2.6.

Open-loop planning Based on the sequence of optimal actions computed in Equation 2.4, the simplest approach to solve the associated decision making problem would be to apply it sequentially to the environment, neglecting the resulting states $(\boldsymbol{s}_{n+1}, \dots, \boldsymbol{s}_{n+H})$ (or observations $(\boldsymbol{o}_{n+1}, \dots, \boldsymbol{o}_{n+H})$ for POMPDs). This strategy is called *open-loop* planning. It is, however, problematic for the following reasons:

1. The model q may suffer from *model bias* (Deisenroth, 2010, sec. 3.2), i.e., it does not capture the actual dynamics $p(\boldsymbol{s}_{n+1} \mid \boldsymbol{s}_n, \boldsymbol{a}_n)$ accurately enough in the state regions of interest. During planning, model errors accumulate, making model predictions and thus planned actions at timesteps far into the future very unreliable.

2. In a POMDP, the belief state can be very uncertain if none or only a few observations have been observed, also negatively impacting the task performance for the sequence of actions computed.

Closed-loop planning The above two issues are mitigated by closed-loop planning, also referred to as model-predictive control or receding-horizon control (surveyed by, e.g., García et al. (1989) and Schwenzer et al. (2021)). From the computation in Equation 2.4, only the first action is applied to the environment. The resulting state $\boldsymbol{s}_{n+1}$ (in an MDP) or observation $\boldsymbol{o}_{n+1}$ (in a POMDP) is recorded. In a POMDP, the belief state is updated given the most recent observation. Thus, at any point in time, all information available on the state of the system is incorporated. Consequently, action decisions are not based on predicted states, which curbs the effects of accumulating model errors. Given the updated state or belief, Equation 2.4 is carried out again (starting at $n + 1$) to compute the next optimal action.

2.3.6. Reinforcement learning

In this subsection we give a short overview on basic concepts and terminology of reinforcement learning relevant for this thesis. For a more in-depth treatment we refer to Sutton and Barto (2018). All following definitions are based on the assumption that the sequential decision making problem is formalized as a Markov decision process (Subsection 2.3.1) with an infinite-horizon, discounted optimality criterion as in Equation 2.3. The action space $\mathcal{A}$ and state space $\mathcal{S}$ may be finite sets or continuous spaces.

Action value function The action value function $Q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ gives, for a policy π, the expected return when applying action $\boldsymbol{a}$ in state $\boldsymbol{s}$ and following the policy π afterwards (Watkins, 1989). Thus, it entangles the environment's dynamics, the reward function, and the policy. The optimal action value function Q_* refers to the action value function of an *optimal* policy, i.e., the policy which maximizes the expected return: $Q_*(\boldsymbol{s}, \boldsymbol{a}) = \max_\pi Q_\pi(\boldsymbol{s}, \boldsymbol{a})$ (Sutton and Barto, 2018, eq. 3.16). If we have access to the Q function of the optimal policy Q_*, we can implement the optimal policy π_* as an arg max operation

$$\pi_*(\boldsymbol{s}) = \arg\max_{\boldsymbol{a} \in \mathcal{A}} Q_*(\boldsymbol{s}, \boldsymbol{a}). \tag{2.5}$$

State value function The state value function $V_\pi : \mathcal{S} \to \mathbb{R}$ is strongly related to the action value function. It gives, for a policy π, the expected return when following the policy π from state $\boldsymbol{s}$ onwards. It can be computed from the action value function with

$$V_\pi(\boldsymbol{s}) = \mathbb{E}_{\boldsymbol{a} \sim \pi(\boldsymbol{a} \mid \boldsymbol{s})}[Q_\pi(\boldsymbol{s}, \boldsymbol{a})]. \tag{2.6}$$

Q-learning Based on the concept of dynamic programming (Bellman, 1957b), a consistency relation (Bellman equation) for the optimal action value function can be formulated (see, e.g., Sutton and Barto (2018, eq. 3.20)), which states

$$Q_*(\boldsymbol{s}_n, \boldsymbol{a}_n) = \quad \mathbb{E}_{\boldsymbol{s}_{n+1} \sim p(\cdot \mid \boldsymbol{s}_n, \boldsymbol{a}_n)} \left[r(\boldsymbol{s}_n, \boldsymbol{a}_n, \boldsymbol{s}_{n+1}) + \gamma \max_{\boldsymbol{a}' \in \mathcal{A}} Q_*(\boldsymbol{s}_{n+1}, \boldsymbol{a}') \right]. \tag{2.7}$$

Q-learning is an algorithm presented by Watkins (1989) (for finite state and action spaces) which computes, through interaction with an environment, under some assumptions given by Watkins and Dayan (1992), an optimal action value function. For an observed transition, i.e., a tuple of the form $(\boldsymbol{s}_n, \boldsymbol{a}_n, \boldsymbol{s}_{n+1}, r_{n+1})$ with $r_{n+1} = R(\boldsymbol{s}_n, \boldsymbol{a}_n, \boldsymbol{s}_{n+1})$, and some action value function Q, we can compute an action value $\bar{Q}(\boldsymbol{s}_n, \boldsymbol{a}_n)$ which fulfills the consistency equation in Equation 2.7

$$\bar{Q} = r_{n+1} + \gamma \max_{\boldsymbol{a}' \in \mathcal{A}} Q(\boldsymbol{s}_{n+1}, \boldsymbol{a}'). \tag{2.8}$$

Q is now updated with a step-size parameter α as

$$Q(\boldsymbol{s}_n, \boldsymbol{a}_n) \leftarrow (1 - \alpha) Q(\boldsymbol{s}_n, \boldsymbol{a}_n) + \alpha \bar{Q}. \tag{2.9}$$

The next action to be applied on the environment is chosen via $\boldsymbol{a} = \arg\max_{\boldsymbol{a}'} Q(\boldsymbol{s}, \boldsymbol{a}')$. However, for convergence, all state-action pairs have to keep being visited, making *exploration*, e.g. by choosing a random action occasionally (ϵ-*greedy*), mandatory (Sutton and Barto, 2018).

Maximization bias The action value function $Q(\boldsymbol{s}_n, \boldsymbol{a}_n)$ maintained by Q-learning is an *estimate* to the *true* action value, computed from a finite number of experience samples on the environment. Let $\hat{Q}(\boldsymbol{s}_n, \boldsymbol{a}_n)$ be the true action value. When choosing an action, we aim to choose the action that maximizes the *true* action value $\arg\max_{\boldsymbol{a}} \hat{Q}(\boldsymbol{s}_n, \boldsymbol{a})$. What Q-learning does, however, is to maximize the *estimate* over the action value $\arg\max_{\boldsymbol{a}} Q(\boldsymbol{s}_n, \boldsymbol{a})$ for action selection. This leads to a systematic overestimation of the true action value, named the *maximization bias*. Double Q-learning (Hasselt, 2010) is an approach to mitigate this bias by using different value functions Q^A, Q^B for action selection and value estimation, i.e.

$$\bar{Q}^B = r_{n+1} + \gamma Q^B \left(\boldsymbol{s}_{n+1}, \arg\max_{\boldsymbol{a}' \in \mathcal{A}} Q^A(\boldsymbol{s}_{n+1}, \boldsymbol{a}') \right) \tag{2.10}$$

$$Q^A(\boldsymbol{s}_n, \boldsymbol{a}_n) \leftarrow (1 - \alpha) Q^A(\boldsymbol{s}_n, \boldsymbol{a}_n) + \alpha \bar{Q}^B. \tag{2.11}$$

The functions Q^A and Q^B regularly change roles, i.e., Q^B is used for action selection and Q^A to compute the update on Q^B.

Continuous state spaces In continuous state spaces, function approximators such as neural networks can be used to estimate action values. Let $Q^{\boldsymbol{\theta}}(\boldsymbol{s}, \boldsymbol{a})$, $Q^{\boldsymbol{\theta}'}(\boldsymbol{s}, \boldsymbol{a})$ be two functions estimating Q values from a continuous state $\boldsymbol{s}$, parametrized by $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$, respectively. An extension of double Q learning to continuous state spaces was presented by Hasselt et al. (2016). Again, different value functions are used for action selection and value estimation. The target value is computed as

$$\bar{Q} = r_{n+1} + \gamma Q^{\boldsymbol{\theta}'} \left[\boldsymbol{s}_{n+1}, \arg\max_{\boldsymbol{a}' \in \mathcal{A}} Q^{\boldsymbol{\theta}}(\boldsymbol{s}_{n+1}, \boldsymbol{a}') \right]. \tag{2.12}$$

Given an observed transition $(\boldsymbol{s}_n, \boldsymbol{a}_n, \boldsymbol{s}_{n+1}, r_{n+1})$, a stochastic gradient descent update with learning rate α can be formulated as

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha(\bar{Q} - Q^{\boldsymbol{\theta}}(\boldsymbol{s}_n, \boldsymbol{a}_n)) \nabla_{\boldsymbol{\theta}} Q^{\boldsymbol{\theta}}(\boldsymbol{s}_n, \boldsymbol{a}_n). \tag{2.13}$$

In contemporary Q-learning algorithms with continuous state spaces, the parameters of the target network $\boldsymbol{\theta}'$ usually slowly track the parameters $\boldsymbol{\theta}$ with

$$\boldsymbol{\theta}' \leftarrow \tau \boldsymbol{\theta} + (1 - \tau) \boldsymbol{\theta}' \tag{2.14}$$

with $\tau \ll 1$, see, e.g., Haarnoja et al. (2018) and Lillicrap et al. (2016).

Actor-critic algorithms The Q-learning algorithm presented above only learns an approximation to the true action value function without explicitly representing a policy, and is thus a *value-based* method. Neglecting exploration, actions are chosen through an arg max operation (Equation 2.5). In continuous action spaces, performing an arg max operation over the space of actions is not straightforward and potentially very time-consuming.

A second class of reinforcement learning algorithms, named *policy-based* methods, e.g., REINFORCE (R. J. Williams, 1992), directly learn a (parametrized) representation of a (near-optimal) policy $\pi(\boldsymbol{a} \mid \boldsymbol{s})$, and are directly applicable to continuous action spaces. However, these methods are typically plagued with high-variance gradient estimates during training, slowing down convergence (Konda and Tsitsiklis, 1999).

In *actor-critic* methods (Konda and Tsitsiklis, 1999), both an actor (a policy) and a critic, assessing the quality of an action (e.g., an action value function), are learned, reducing the gradient variance of purely policy-based methods. The actor-critic design enables to extend the concept of Q-learning to continuous action spaces. In algorithms such as deterministic policy gradient (DPG, D. Silver et al. (2014)), deep deterministic policy gradient (DDPG, Lillicrap et al. (2016)), and soft actor-critic (SAC, Haarnoja et al. (2018)), the arg max operation is replaced by a policy, which is updated by gradient ascent in the direction of increasing action values estimated by Q. However, also actor-critic algorithms exist which leverage a state-value function as their critic, e.g., proximal policy optimization (PPO, Schulman et al. (2017)).

Soft actor-critic Soft actor-critic is a model-free reinforcement learning algorithm for sequential decision making problems with continuous state- and action spaces. As the name states, it is an actor-critic method, which maintains an approximation to the state- and action value function *and* a parametrized policy, all implemented as neural networks. To circumvent the maximization bias it maintains a separate *target* network for the state value to compute the update for the action value network. The weights of the target network are updated with exponential moving averages from the weights of the state value network. The main difference of soft actor-critic compared to previous actor-critic methods such as DDPG is that it not only aims to maximize the expected return, but also the expected entropy of the policy π, i.e.,

$$\pi_* = \max_{\pi} \mathcal{J}(\pi) = \max_{\pi} \sum_{n=0}^{N} \mathbb{E}_{(\boldsymbol{s}_n, \boldsymbol{a}_n) \sim \rho_\pi} \left[r(\boldsymbol{s}_n, \boldsymbol{a}_n) + \alpha \mathrm{H}[\pi(\cdot \mid \boldsymbol{s}_n)] \right], \tag{2.15}$$

which is a concept known as *maximum entropy reinforcement learning*. In the above equation, ρ_π represents the joint distribution of states and actions induced by the policy and transition dynamics of the Markov decision process. Experiments by Haarnoja et al. (2018) have shown that SAC outperforms the prior actor-critic algorithms DDPG (Lillicrap et al., 2016) and PPO (Schulman et al., 2017) on a set of baseline tasks both in asymptotic performance and sample-efficiency.

2.4. Machine learning

The main focus of this thesis lies on the combination of learning and sequential decision making. To this end, let us recapitulate the following definition by Mitchell (1997):

Definition 2.4.1 *A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*

Learning an agent which maximizes the expected return when interacting with an environment, based on experience collected on the environment, can thus be seen as a problem of machine

learning. To achieve this, sub-problems can occur which are machine learning problems themselves. Exemplarily, in model-based agents, obtaining a dynamics model (from experience) which should minimize prediction error is also a machine learning problem.

2.4.1. Empirical risk minimization

In many cases, the performance measure is given as an expectation over a random variable following some distribution. For training a dynamics model, this could be the expected prediction error on a distribution of states and actions. However, for training, only a finite set of (independent and identically distributed) samples (the *training set*) is available to *estimate* the expected performance, and the performance is only empirically optimized. In literature, this is known as *empirical risk minimization* (Vapnik, 1991), where risk is a performance measure to minimize. Optimizing the estimated expected performance does not necessarily imply that the expected performance is optimized — a phenomenon known as *overfitting*. To this end, a different, disjoint set of finite samples is required to estimate the expected performance, called *test set*. In the following chapters, we maintain disjoint train- and test sets to train and evaluate the proposed algorithms.

2.4.2. Likelihood maximization

A machine learning problem commonly occurring in this thesis is that of likelihood maximization. Given a probabilistic model $p(\mathrm{X} \mid \boldsymbol{\theta})$ we aim to find a parameter vector $\boldsymbol{\theta}$ which maximizes the likelihood

$$\theta^* = \max_{\boldsymbol{\theta}} p(\mathrm{X} \mid \boldsymbol{\theta}). \tag{2.16}$$

For numerical reasons, typically the logarithm of the likelihood is maximized.

2.5. Latent variable models

In latent variable models (see, e.g., Bishop (2007)), data generation is modeled by a hierarchical process. At the top of the hierarchy, a single or a collection of unobserved (latent) random variables condition the generation of the observed random variables. The state-space model presented in Subsection 2.2.2 is such a latent variable model. For simplicity, in this section, we limit our discussion to a non-sequential latent variable model.

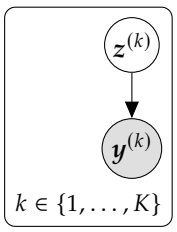


Figure 2.4.: Non-sequential latent variable model.

Let us exemplarily consider a dataset of images of handwritten digits from 0 to 9. For the process of crafting a single image, we may first decide on which digit to draw. Formally, we sample the digit to draw $\boldsymbol{z} \in \{0, \ldots, 9\}$ from the probability distribution $p(\boldsymbol{z})$.[§] Second, we draw the particular digit; however, each drawing may underlie stochastic variations, such as the writing style. We can denote this probabilistically as the

[§] Although $\boldsymbol{z}$ denotes a scalar here, we use a bold font for generality.

conditional distribution $p(\boldsymbol{y} \mid \boldsymbol{z})$ for an image $\boldsymbol{y}$ of digit $\boldsymbol{z}$. This process is represented formally as a directed graphical model for a dataset of K images in Figure 2.4.

While in this example the latent variable $\boldsymbol{z}$ is discrete, $\boldsymbol{z}$ can also be continuous in more complex models to capture a larger variety of variations in the data. In the following, we will consider models with continuous $\boldsymbol{z}$ and $\boldsymbol{y}$, as this is the most relevant case for this thesis.

The latent variable model in Figure 2.4 is fully characterized by the distribution over $\boldsymbol{z}$, which is $p(\boldsymbol{z})$, and the conditional distribution $p(\boldsymbol{y} \mid \boldsymbol{z})$. Given a dataset of observations $\boldsymbol{y}^{(k)}$ with $k \in \{1, \ldots, K\}$, two tasks are of particular interest. First, we are interested in modeling the data generation process, i.e., $p(\boldsymbol{z})$ and $p(\boldsymbol{y} \mid \boldsymbol{z})$. Second, modeling the posterior distribution $p(\boldsymbol{z} \mid \boldsymbol{y})$ is relevant for multiple tasks, such as embedding a higher-dimensional observation into a lower-dimensional latent space.

For general, non-linear latent variable models, computing the posterior distribution $p(\boldsymbol{z} \mid \boldsymbol{y})$ exactly is not tractable. The framework of *variational inference* provides methods to approximate this posterior distribution.

2.5.1. Variational inference

The general idea of variational inference is to frame the problem of approximate inference as an *optimization* problem (see, e.g., Blei et al. (2016) for an introduction). Variational inference describes the process of finding a probability density $q^*(\boldsymbol{z})$ from a family of densities $\boldsymbol{Q}$ which minimizes the Kullback-Leibler divergence to the exact posterior density $p(\boldsymbol{z} \mid \boldsymbol{y})$, i.e.,

$$q^*(\boldsymbol{z}) = \arg\min_{q \in \boldsymbol{Q}} \mathrm{KL}[q(\boldsymbol{z}) \,||\, p(\boldsymbol{z} \mid \boldsymbol{y})]. \tag{2.17}$$

2.5.2. Evidence lower bound

At a first sight, the optimization problem in Equation 2.17 seems to be of limited usefulness. In order to find an optimal $q^*(\boldsymbol{z})$, we need to know $p(\boldsymbol{z} \mid \boldsymbol{y})$, which we assumed to have no access to in the first place. However, with the definition of the Kullback-Leibler divergence between two probability density functions (Cover and Thomas, 2006, p. 251)

$$\mathrm{KL}[q(\boldsymbol{z}) \,||\, p(\boldsymbol{z} \mid \boldsymbol{y})] = \int q(\boldsymbol{z}) \log \frac{q(\boldsymbol{z})}{p(\boldsymbol{z} \mid \boldsymbol{y})} \, \mathrm{d}\boldsymbol{z} \tag{2.18}$$

and Bayes' theorem

$$p(\boldsymbol{z} \mid \boldsymbol{y}) = \frac{p(\boldsymbol{y} \mid \boldsymbol{z}) p(\boldsymbol{z})}{p(\boldsymbol{y})}, \tag{2.19}$$

we arrive at the following expression

$$\mathrm{KL}[q(\boldsymbol{z}) \,||\, p(\boldsymbol{z} \mid \boldsymbol{y})] = \log p(\boldsymbol{y}) - \int q(\boldsymbol{z}) \log p(\boldsymbol{y} \mid \boldsymbol{z}) \, \mathrm{d}\boldsymbol{z} + \mathrm{KL}[q(\boldsymbol{z}) \,||\, p(\boldsymbol{z})]. \tag{2.20}$$

As the KL divergence is non-negative (Cover and Thomas, 2006, p. 252) for all (q, p), it follows

$$\log p(\boldsymbol{y}) \geq \int q(\boldsymbol{z}) \log p(\boldsymbol{y} \mid \boldsymbol{z}) \, \mathrm{d}\boldsymbol{z} - \mathrm{KL}[q(\boldsymbol{z}) \,||\, p(\boldsymbol{z})]. \tag{2.21}$$

As the right-hand side of the inequality in Equation 2.21 lower-bounds the log evidence $\log p(\boldsymbol{y})$, it is termed *evidence lower bound*. When reconsidering Equation 2.20, as the log evidence $\log p(\boldsymbol{y})$ is not a function of q, we can conclude that *maximizing* the evidence lower bound with respect to q *minimizes* $\mathrm{KL}[q(\boldsymbol{z}) \,||\, p(\boldsymbol{z} \mid \boldsymbol{y})]$, which was the objective we initially formulated in Equation 2.17.

2.5.3. Variational autoencoder

Recent advances in deep neural networks allow for complex observation models $p(\boldsymbol{y} \mid \boldsymbol{z})$, such as conditional image generation models. However, even for a small neural network with a single nonlinear hidden layer, the posterior $p(\boldsymbol{z} \mid \boldsymbol{y})$ is intractable (Kingma and Welling, 2014).

Given a dataset of observations $\mathcal{Y} = \{\boldsymbol{y}^{(k)}\}_{k=1}^{K}$ one may resort to sampling-based approximate inference techniques, yielding an approximate posterior $q(\boldsymbol{z}^{(k)} \mid \boldsymbol{y}^{(k)})$ for each datapoint. This approach is computationally expensive for large datasets. It can not leverage previously performed computations for performing inference on datapoints *within* the set of observations $\mathcal{Y}$ nor on novel, *unseen* datapoints.

To share computations across datapoints and for the ability to generalize to unseen data, Kingma and Welling (2014) propose to learn a *parametric* model to perform approximate inference, which maps an observation $\boldsymbol{y}$ to parameters of a probability distribution, e.g., mean and covariance matrix of a multivariate Gaussian distribution for a continuous latent variable $\boldsymbol{z} \in \mathbb{R}^D$. The idea of sharing computation among separate inference tasks is known as *amortized inference* (Gershman and Goodman, 2014; Stuhlmüller et al., 2013).

2.5.4. Reparametrization trick

Recapitulating the evidence lower bound in Equation 2.21, we observe that it contains an expectation term and a Kullback-Leibler divergence term. The expectation term is of the form

$$\mathbb{E}_{z \sim q(z \mid \boldsymbol{\phi})} \left[\log p(\boldsymbol{y} \mid \boldsymbol{z}, \boldsymbol{\theta}) \right] = \int q(\boldsymbol{z} \mid \boldsymbol{\phi}) \log p(\boldsymbol{y} \mid \boldsymbol{z}, \boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{z}, \tag{2.22}$$

where we have made the dependence of p and q on parameters $\boldsymbol{\theta}$, $\boldsymbol{\phi}$ explicit. For gradient-based optimization, it is required to compute gradients of the above expectation with respect to $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$. In Kingma and Welling (2014) a method is presented, called the *reparameterization trick*, which provides Monte Carlo estimates of gradients of expectations such as the above.

For generality, in the following, f is an arbitrary function depending on the random variable $\boldsymbol{z} \sim q(\boldsymbol{z} \mid \boldsymbol{\phi})$ and on a parameter $\boldsymbol{\theta}$, and E is the expectation

$$E = \mathbb{E}_{z \sim q(z \mid \boldsymbol{\phi})} \left[f(\boldsymbol{z}, \boldsymbol{\theta}) \right]. \tag{2.23}$$

Differentiating Equation 2.23 with respect to $\boldsymbol{\theta}$ is straightforward, with

$$\frac{\partial}{\partial \boldsymbol{\theta}} E = \mathbb{E}_{z \sim q(z \mid \boldsymbol{\phi})} \left[\frac{\partial}{\partial \boldsymbol{\theta}} f(\boldsymbol{z}, \boldsymbol{\theta}) \right]. \tag{2.24}$$

An unbiased Monte Carlo estimate for the gradient $\frac{\partial}{\partial \boldsymbol{\theta}} E$ is given by

$$\frac{1}{K} \sum_{k=1}^{K} \frac{\partial}{\partial \boldsymbol{\theta}} f(\boldsymbol{z}^{(k)}, \boldsymbol{\theta}) \quad \text{with} \quad \boldsymbol{z}^{(k)} \sim q(\boldsymbol{z} \mid \boldsymbol{\phi}). \tag{2.25}$$

Differentiating Equation 2.23 with respect to parameters $\boldsymbol{\phi}$ of the density q is more involved, as we cannot, without further modifications, note down $\frac{\partial}{\partial \boldsymbol{\phi}} f(\boldsymbol{z}^{(k)}, \boldsymbol{\theta})$.

The *reparameterization trick* (Kingma and Welling, 2014) proposes to make the dependence of $\boldsymbol{z}^{(k)}$ on $\boldsymbol{\phi}$ explicit through a differentiable transformation g, such that

$$E = \mathbb{E}_{\boldsymbol{\epsilon} \sim p_\epsilon(\boldsymbol{\epsilon})} \left[f(g(\boldsymbol{\epsilon}, \boldsymbol{\phi}), \boldsymbol{\theta}) \right]. \tag{2.26}$$

Now we can obtain an unbiased estimate of the gradient of E w.r.t. $\boldsymbol{\phi}$ analogously to Equation 2.25, i.e.,

$$\frac{1}{K} \sum_{k=1}^{K} \frac{\partial}{\partial \boldsymbol{\phi}} f(g(\boldsymbol{\epsilon}, \boldsymbol{\phi}), \boldsymbol{\theta}) \quad \text{with} \quad \boldsymbol{\epsilon}^{(k)} \sim p_\epsilon(\boldsymbol{\epsilon}). \tag{2.27}$$

2.6. Planning and trajectory optimization

The basic idea of model-based planning and model-predictive control is to find an action sequence $(\boldsymbol{a}_n^*, \ldots, \boldsymbol{a}_{n+H-1}^*)$ which maximizes a performance objective $\mathcal{J}$, i.e.,

$$(\boldsymbol{a}_n^*, \ldots, \boldsymbol{a}_{n+H-1}^*) = \max_{(\boldsymbol{a}_n, \ldots, \boldsymbol{a}_{n+H-1}) \in \mathcal{A}^H} \mathcal{J}(\boldsymbol{a}_n, \ldots, \boldsymbol{a}_{n+H-1}). \tag{2.28}$$

In control and planning tasks, the objective $\mathcal{J}$ typically contains terms which depend on intermediate states of the dynamical system $\boldsymbol{s}_n, \boldsymbol{s}_{n+1}, \ldots, \boldsymbol{s}_{n+H}$. The dependencies between states can, e.g., be modeled by a learned forward dynamics model $\boldsymbol{s}_{n+1} \sim q(\boldsymbol{s}_{n+1} \mid \boldsymbol{s}_n, \boldsymbol{a}_n)$.

In *shooting* algorithms, an initial state distribution $p(\boldsymbol{s}_n)$ (or samples thereof) are propagated with forward dynamics $\boldsymbol{s}_{n+1} \sim q(\boldsymbol{s}_{n+1} \mid \boldsymbol{s}_n, \boldsymbol{a}_n)$, and state-dependent terms are evaluated on propagated state distributions or state samples. In contrast, in *collocation* algorithms, intermediate states are added to the variables to optimize, and compliance with the dynamics of the system is added as a constraint to the optimization problem (Betts, 1998; Rybkin et al., 2021; Tedrake, 2023).

In general, optimization algorithms can be distinguished by up to which order gradient information (i.e., derivatives) is incorporated into the optimization process. A zero-order optimization algorithm is *gradient-free*, i.e., no gradient information is incorporated.

2.6.1. Cross-entropy method

The cross-entropy method is a zero-order stochastic optimization algorithm, which has its roots in literature on rare event probability estimation with the cross-entropy measure (Boer et al., 2005; Rubinstein, 1996, 1999). To optimize an objective function $\mathcal{J}(\boldsymbol{x})$, first, a set of *candidates* is sampled from an initial distribution $p(\boldsymbol{x})$. A subset of *elite* candidates, which yield the highest

values on $\mathcal{J}(\boldsymbol{x})$, is used to adjust (re-fit) the distribution $p(\boldsymbol{x})$. From the re-fit distribution, a new set of candidates is sampled, and the iteration continues, typically until a pre-defined limit of optimization iterations is reached.

In model-based planning algorithms, the cross-entropy method is commonly employed for action selection in a *shooting* setting. We provide pseudocode of the CEM algorithm for trajectory optimization, as used in Chapters 3 to 5 in this thesis, in Algorithm 1.

Algorithm 1: Cross-entropy method (CEM) for trajectory optimization

Input: Objective function $\mathcal{J} : \mathcal{A}^H \to \mathbb{R}, \ \mathcal{A} = [-a_{\max}, a_{\max}]^{D_\mathrm{A}}$
Planning horizon H,
Maximal action magnitude $a_{\max}$,
Number of optimization iterations T,
Number of candidates N_{cand},
Number of elites N_{elites}
Result: Optimal action sequence $(\boldsymbol{a}_1^*, \dots, \boldsymbol{a}_H^*)$

Initialize $\boldsymbol{\mu}_n = \mathbf{0}_{D_\mathrm{A}}$, $\boldsymbol{\sigma}_n^2 = \mathbf{1}_{D_\mathrm{A}} \cdot a_{\max}^2 \ \forall n \in \{1, \dots, H\}$;
for $t = \{1, \dots, T\}$ **do**
  Sample N_{cand} candidate sequences
    $(\boldsymbol{a}_1^k, \dots, \boldsymbol{a}_H^k), k \in \{1, \dots, N_{\text{cand}}\}$ with $\hat{\boldsymbol{a}}_n^k \sim \mathcal{N}(\boldsymbol{\mu}_n, \boldsymbol{\sigma}_n^2)$, $\boldsymbol{a}_n^k = \text{clip}(\hat{\boldsymbol{a}}_n^k, -a_{\max}, a_{\max})$;
  Evaluate objective $\mathcal{J}_k = \mathcal{J}(\boldsymbol{a}_1^k, \dots, \boldsymbol{a}_H^k)$;
  Obtain set of elites $\mathcal{S}_{\text{elites}} \subset \{1, \dots, N_{\text{cand}}\}$
    with $|\mathcal{S}_{\text{elites}}| = N_{\text{elites}}$, $\mathcal{J}_{k'} \geq \mathcal{J}_k \ \ \forall k' \in \mathcal{S}_{\text{elites}}, k \in \{1, \dots, N_{\text{cand}}\} \setminus \mathcal{S}_{\text{elites}}$;
  Re-fit beliefs
    $\boldsymbol{\mu}_n \leftarrow \frac{1}{N_{\text{elites}}} \sum_{k \in \mathcal{S}_{\text{elites}}} \boldsymbol{a}_n^k$;
    $[\boldsymbol{\sigma}_n^2]_d \leftarrow \frac{1}{N_{\text{elites}}-1} \sum_{k \in \mathcal{S}_{\text{elites}}} [\boldsymbol{a}_n^k - \boldsymbol{\mu}_n]_d^2 \quad \forall d \in \{1, \dots, D_\mathrm{A}\}$;
end
Set $(\boldsymbol{a}_1^*, \dots, \boldsymbol{a}_H^*) \leftarrow (\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_H)$;

Despite the fact that with contemporary machine learning frameworks gradient computation is straightforward, zero-order (gradient-free) trajectory optimization algorithms are a common choice when combined with learned high-capacity, e.g. neural network based, dynamics models (Tedrake, 2023). Examples of learned dynamics models with CEM trajectory optimization can be found in Chua et al. (2018), Hafner et al. (2019), and T. Wang and Ba (2020). Tedrake (2023) formulates possible explanations for this. First, population-based methods, such as CEM, benefit from the *single instruction multiple data* parallelism provided by contemporary hardware such as GPUs, in which the time complexity for evaluating the objective function grows sublinearly with the number of candidates. Second, population-based methods are less prone to fall into local minima. When high-capacity function approximators are used to model system dynamics, this problem may be even more pronounced compared to simple physics-based dynamics.

3. Deep Latent Gaussian Process Dynamics Models

Declaration of contributions

The contents of this chapter are based on the peer-reviewed conference publication

N. Bosch, **J. Achterhold**, L. Leal-Taixé, and J. Stückler (2020). 'Planning from Images with Deep Latent Gaussian Process Dynamics'. In: *Proceedings of the Learning for Dynamics and Control Conference (L4DC)* (Bosch et al., 2020).

The above publication was presented as a poster presentation at the *Learning for Dynamics and Control Conference (L4DC) 2020, online event*.

Author contributions are as follows:

	Scientific ideas	Data generation	Analysis & Interpretation	Paper writing
Nathanael Bosch	45 %	40 %	45 %	33 %
Jan Achterhold	30 %	60 %	40 %	33 %
Laura Leal-Taixé	5 %	0 %	5 %	0 %
Jörg Stückler	20 %	0 %	10 %	33 %

Parts of this project were conducted as a Master's thesis by Nathanael Bosch, conducted under advisory by Jan Achterhold and Jörg Stückler at the MPI-IS in Tübingen, and examined by Laura Leal-Taixé at TU Munich.

Nathanael Bosch handed in his Master's thesis titled *Learning Gaussian Process Dynamics Models from Visual Observations for Control* at TU Munich on October 15th, 2019.

Jan Achterhold and Jörg Stückler developed the scientific idea to learn a Gaussian process model in the latent space of a variational autoencoder. Jan Achterhold proposed to approach transfer learning problems. Nathanael Bosch developed the lower-bound loss formulation and implemented an initial version of the DLGPD algorithm. Jan Achterhold numerically stabilized the DLGPD algorithm and conducted the final experiments, including implementation and evaluation of the PlaNet baselines. Jan Achterhold proposed and implemented the idea to model rewards by a Gaussian process model. Laura Leal-Taixé gave feedback on the project in regular meetings. Nathanael Bosch, Jan Achterhold and Jörg Stückler wrote the paper.

We provide additional materials, including the conference poster, at `https://dlgpd.is.tue.mpg.de/`. We provide our implementation at `https://github.com/EmbodiedVision/dlgpd`.

3.1. Introduction

In Subsection 1.5.1 we have reviewed model-free and model-based approaches to sequential decision making, and their particular advantages and disadvantages.

We have seen that one advantage of model-based methods, compared to model-free methods, is that, as long as the model is accurate enough for regions of the state space which are relevant for the set of tasks to consider, a set of tasks can be covered by reusing the learned dynamics model. Thus, model-based agents provide a way to approach the *novel tasks* challenge (Subsection 1.4.1), without requiring additional interaction with the environment.

As a second advantage, past research has shown that model-based methods can reduce the amount of environment samples required for training the agent, compared to model-free approaches (Chua et al., 2018; Deisenroth and Rasmussen, 2011; Hafner et al., 2019), even for a single task (c.f., the *sample-efficiency* challenge in Subsection 1.4.2).

Both advantages makes these methods especially attractive for applications in robotics and real-world settings, as collecting experience in real environments, such as driving a vehicle or moving a robot, is often time- and resource-consuming, and mistakes can incur significant costs. This is aggravated by the fact that environment dynamics in real-world environments are likely to be non-stationary (c.f., the *non-stationarity* challenge in Subsection 1.4.2).

A method which provides a particularly sample-efficient approach to sequential decision making is PILCO (Deisenroth and Rasmussen, 2011). PILCO is a model-based reinforcement learning method. Based on a learned dynamics model, implemented by Gaussian process regression, policy search is conducted to learn a parametrized policy. A central role for contributing to the sample-efficiency of PILCO was attributed to the Gaussian process regression model, as it handles uncertainty in dynamics caused by scarcity of data in a principled way, thereby mitigating model bias. PILCO assumes observations of the environment to be available as low-dimensional measurements. However, in many problems of interest the underlying state of the world is only indirectly observable through images (c.f., the *high-dimensional observations* challenge in Subsection 1.4.1).

As Gaussian process regression is a method which makes predictions based on contextual observations, it is a promising approach to adapt to non-stationarities in the environment's dynamics, mitigating the need of retraining the dynamics model.

In this chapter, we aim to extend PILCO to high-dimensional observations such as images. The dynamics of an environment are modeled by a Gaussian process in the latent space of a variational autoencoder. In our experiments, we put a special emphasis on the data efficiency of adapting to non-stationarities in the environments dynamics.

Specifically, we make the following contributions:

- We combine Gaussian processes (GPs) with neural networks to learn latent dynamics models from visual observations. All parts of the proposed deep latent Gaussian process dynamics (DLGPD) model can be trained jointly by optimizing a lower bound on the likelihood of transitions in the image space.

- We integrate the learned system dynamics with learning a reward function and use the models for model-predictive control. In our experiments, the predictions of the learned dynamics model enable the agent to successfully solve an inverted pendulum swingup task.
- We demonstrate that the latent Gaussian process dynamics model allows the agent to quickly adapt to environments with modified system dynamics from only a few rollouts. Our approach compares favorably to the purely deep-learning based baseline PlaNet (Hafner et al., 2019) in this transfer learning experiment.

3.2. Related work

Bayesian nonparametric Gaussian process models (Rasmussen and C. K. I. Williams, 2006) are a popular choice for dynamics models in reinforcement learning (RL) (Deisenroth et al., 2009; Ko et al., 2007; Rasmussen and Kuss, 2003; J. M. Wang et al., 2005). When the low-dimensional states of the environment are available to the agent, PILCO (Deisenroth and Rasmussen, 2011) achieves remarkable sample efficiency and is able to solve a swingup task in a real cart-pole system with only 17.5 seconds of interaction. Deep PILCO (Gal et al., 2016) replaces GPs in PILCO with Bayesian neural networks to learn the dynamics model. The method is not demonstrated to learn an embedding of high dimensional image observations but directly operates on low-dimensional state representations.

Model-free deep RL algorithms have shown good performance in image-based domains (Lillicrap et al., 2016; Mnih et al., 2015), but they commonly require a large number of interactions. On the other hand, model-based RL can often be more data-efficient. Many such algorithms learn low-dimensional abstract state representations (Lesort et al., 2018) and model the system dynamics in the learned latent space (Fraccaro et al., 2017; Karl et al., 2017). Some approaches such as E2C (Watter et al., 2015), RCE (Banijamali et al., 2018) or SOLAR (M. Zhang et al., 2019) learn locally-linear latent transitions and plan for actions based on the linear quadratic regulator (LQR). In comparison, we learn a non-local, non-linear dynamics model to select actions by planning in latent space. PlaNet (Hafner et al., 2019) learns a recurrent encoder and a latent neural transition model to efficiently plan in latent space. All components of PlaNet are modeled through deep neural networks. We propose to model state transitions with GPs to reduce the number of model parameters, provide better uncertainty estimates, and generally increase the data-efficiency.

Our formulation provides a data-efficient way to transfer a learned dynamics model to different system dynamics. Closely related are meta-learning approaches that also learn to transfer between different properties for the same task, e.g. Killian et al. (2017), Sæmundsson et al. (2018), and Al-Shedivat et al. (2018). While our models are not specifically trained for transferability, it is an inherent property of our formulation.

3.3. Preliminaries

This section is not part of Bosch et al. (2020).

Of particular relevance for this chapter are preliminaries on model-based planning (see Subsection 1.5.1), state-space models (see Subsection 2.2.2), Gaussian process regression (see

Subsection 3.3.1), variational inference and variational auto-encoders (see Section 2.5) and the cross-entropy method (CEM) planning algorithm (see Section 2.6).

3.3.1. Gaussian processes

A Gaussian process (GP, see e.g. Rasmussen and C. K. I. Williams (2006) for reference) describes the distribution of a finite collection of scalar quantities $y_n \in \mathbb{R}, \boldsymbol{y} = [y_1, \dots, y_N]^\top$, paired with corresponding inputs $\boldsymbol{x}_n \in \mathcal{X}, \mathrm{X} = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_N)$. It is fully described by a *mean function* $m : \mathcal{X} \to \mathbb{R}$ and a *covariance function* $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. By

$$\boldsymbol{m}(\mathrm{X}) = [m(\boldsymbol{x}_1), m(\boldsymbol{x}_2), \dots, m(\boldsymbol{x}_N)]^\top \tag{3.1}$$

we abbreviate the evaluation of the mean function for a collection of inputs X. We denote the pairwise evaluation of the covariance function on a collection of inputs X as

$$\boldsymbol{K}(\mathrm{X}, \mathrm{X}) = \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & k(\boldsymbol{x}_1, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ k(\boldsymbol{x}_2, \boldsymbol{x}_1) & k(\boldsymbol{x}_2, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_2, \boldsymbol{x}_N) \\ \vdots & \vdots & \ddots & \vdots \\ k(\boldsymbol{x}_N, \boldsymbol{x}_1) & k(\boldsymbol{x}_N, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{bmatrix}. \tag{3.2}$$

The covariance function k must fulfill the property that its pairwise evaluation $\boldsymbol{K}(\mathrm{X}, \mathrm{X})$ on arbitrary collections of input points yields a matrix which is positive semidefinite and thus is a valid *covariance matrix*. Note that the above definitions include possibly *infinite* input sets $\mathcal{X}$. Exemplarily, $\boldsymbol{x}$ might refer to continuous time, in which $\mathcal{X} \subseteq \mathbb{R}$, or a real-valued vector, in which $\mathcal{X} \subseteq \mathbb{R}^D$.

Definition 3.3.1 *A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution (Rasmussen and C. K. I. Williams, 2006).*

With the above definitions, the finite collection of scalar quantities $\boldsymbol{y}$ at input locations X is jointly Gaussian distributed with mean $\boldsymbol{m}(\mathrm{X})$ and covariance matrix $\boldsymbol{K}(\mathrm{X}, \mathrm{X})$, i.e.,

$$\boldsymbol{y} \sim \mathcal{N}\left(\boldsymbol{m}(\mathrm{X}), \boldsymbol{K}(\mathrm{X}, \mathrm{X})\right). \tag{3.3}$$

Regression Let us now investigate how we can leverage the idea of Gaussian processes for the task of function regression from noisy measurements. We slightly restrict the setting and assume inputs to be vectors of dimension D, i.e., $\boldsymbol{x}_i \in \mathbb{R}^D, \mathrm{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N)$, as this is the most relevant setting for the modeling of dynamical systems. We assume measurements to be generated by an underlying, unknown function $f : \mathbb{R}^D \to \mathbb{R}$, which are additionally perturbed by zero-mean Gaussian noise with variance σ_{n}^2, i.e.

$$\hat{y} = f(\boldsymbol{x}) + \epsilon \quad \epsilon \sim \mathcal{N}(0, \sigma_{\mathrm{n}}^2). \tag{3.4}$$

In probabilistic regression, we are interested in the (joint) distribution of a collection of unobserved function values $\boldsymbol{f}^* \in \mathbb{R}^M$ at inputs $\mathrm{X}^* = (\boldsymbol{x}_1^*, \dots, \boldsymbol{x}_M^*)$, given perturbed observations $\hat{\boldsymbol{y}} \in \mathbb{R}^N$ at inputs $\mathrm{X} = [\boldsymbol{x}_1^*, \dots, \boldsymbol{x}_N^*]$. Formally, this corresponds to the conditional distribution $p(\boldsymbol{f}^* \mid \mathrm{X}^*, \hat{\boldsymbol{y}}, \mathrm{X})$.

According to the Gaussian process formulation with additive Gaussian observation noise, observed and unobserved function values are jointly Gaussian distributed with

$$\begin{bmatrix} \hat{\boldsymbol{y}} \\ \boldsymbol{f}^* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{m}(\mathrm{X}) \\ \boldsymbol{m}(\mathrm{X}^*) \end{bmatrix}, \begin{bmatrix} \boldsymbol{K}(\mathrm{X},\mathrm{X}) + \sigma_\mathrm{n}^2 \boldsymbol{I} & \boldsymbol{K}(\mathrm{X},\mathrm{X}^*) \\ \boldsymbol{K}(\mathrm{X}^*,\mathrm{X}) & \boldsymbol{K}(\mathrm{X}^*,\mathrm{X}^*) \end{bmatrix} \right), \tag{3.5}$$

corresponding to the distribution $p(\boldsymbol{f}^*, \hat{\boldsymbol{y}} \mid \mathrm{X}^*, \mathrm{X})$. We obtain $p(\boldsymbol{f}^* \mid \mathrm{X}^*, \hat{\boldsymbol{y}}, \mathrm{X})$ from $p(\boldsymbol{f}^*, \hat{\boldsymbol{y}} \mid \mathrm{X}^*, \mathrm{X})$ through conditioning, which again yields a Gaussian distribution

$$\boldsymbol{f}^* \sim \mathcal{N}\left(\boldsymbol{\mu}_{f^*}, \boldsymbol{\Sigma}_{f^*}\right) \tag{3.6}$$

with posterior mean and covariance matrix given by

$$\boldsymbol{\mu}_{f^*} = \boldsymbol{m}(\mathrm{X}^*) + \boldsymbol{K}(\mathrm{X}^*,\mathrm{X})[\boldsymbol{K}(\mathrm{X},\mathrm{X}) + \sigma_\mathrm{n}^2 \boldsymbol{I}]^{-1}(\hat{\boldsymbol{y}} - \boldsymbol{m}(\mathrm{X}^*)) \tag{3.7}$$

$$\boldsymbol{\Sigma}_{f^*} = \boldsymbol{K}(\mathrm{X}^*,\mathrm{X}^*) - \boldsymbol{K}(\mathrm{X}^*,\mathrm{X})[\boldsymbol{K}(\mathrm{X},\mathrm{X}) + \sigma_\mathrm{n}^2 \boldsymbol{I}]^{-1}\boldsymbol{K}(\mathrm{X},\mathrm{X}^*). \tag{3.8}$$

Covariance function The covariance function $k(\boldsymbol{x}_1, \boldsymbol{x}_2)$ encodes how much the corresponding output values (y_1, y_2) correlate. The covariance function is also commonly called *kernel*. The choice of the mean and covariance function (and its hyperparameters) encodes a-priori knowledge on the stochastic process. In regression, if we expect the underlying function to be smooth, our choice of covariance function should reflect this. Similarly, if we expect our function to be periodic, we should choose a periodic covariance function. A widely used covariance function for vector-valued inputs which poses a smoothness prior on the underlying function is the *squared exponential kernel* (Rasmussen and C. K. I. Williams, 2006)

$$k(\boldsymbol{x}_p, \boldsymbol{x}_q) = \alpha^2 \exp\left(-\frac{1}{2}(\boldsymbol{x}_p - \boldsymbol{x}_q)^\top \boldsymbol{\Lambda}(\boldsymbol{x}_p - \boldsymbol{x}_q)\right) \tag{3.9}$$

In our case, $\boldsymbol{\Lambda} = \mathrm{diag}(\boldsymbol{l})^{-2}$, where $\boldsymbol{l}$ is a vector of *characteristic lengthscales*. The factor α^2 is termed *outputscale*.

Model selection Gaussian process regression is a non-parametric approach. In parametric approaches, such as neural network regression, the training data $(\mathrm{X}, \hat{\boldsymbol{y}})$ is used to fit parameters of a function approximator, which in turn is used to make predictions on unseen datapoints. In contrast, in Gaussian process regression, the training data is directly leveraged to make predictions by conditioning a Gaussian distribution. However, similar to parametric approaches, the Gaussian process regression has *hyperparameters*. Among these hyperparameters are the choice of mean- and covariance function and their parameters, and the choice and parameters of the observation model. Generally, the problem of choosing hyperparameters is referred to as *model selection*. There are two prominent approaches to model selection for Gaussian process regression, which is leave-one-out cross validation and marginal likelihood maximization (Rasmussen and C. K. I. Williams, 2006). We will focus here on the marginal likelihood for quantifying model fit, as it emerges in the derivation of the training objective of the model proposed in Chapter 3. For computing the marginal likelihood, function values $\boldsymbol{f}$ are marginalized with

$$p(\hat{\boldsymbol{y}} \mid \mathrm{X}, \boldsymbol{\theta}_f, \boldsymbol{\theta}_y) = \int p(\hat{\boldsymbol{y}} \mid \boldsymbol{f}, \mathrm{X}, \boldsymbol{\theta}_y) p(\boldsymbol{f} \mid \mathrm{X}, \boldsymbol{\theta}_f) \, \mathrm{d}\boldsymbol{f}. \tag{3.10}$$

The term $p(\boldsymbol{f} \mid \mathrm{X}, \boldsymbol{\theta}_f)$ is given by the Gaussian process *prior*, which is $\mathcal{N}(\boldsymbol{m}(\mathrm{X}), \boldsymbol{K}(\mathrm{X}, \mathrm{X}))$. The parameter vector $\boldsymbol{\theta}_f$ thus contains parameters of the mean and covariance function. The observation likelihood $p(\hat{\boldsymbol{y}} \mid \boldsymbol{f}, \mathrm{X}, \boldsymbol{\theta}_y)$, is, as defined in Equation 3.4, also Gaussian

$$p(\hat{\boldsymbol{y}} \mid \boldsymbol{f}, \mathrm{X}, \boldsymbol{\theta}_y) = \mathcal{N}\left(\boldsymbol{f}, \sigma_{\mathrm{n}}^2 \boldsymbol{I}\right), \tag{3.11}$$

where $\boldsymbol{\theta}_y = \sigma_{\mathrm{n}}^2$. The marginal likelihood is thus be given as

$$p(\hat{\boldsymbol{y}} \mid \mathrm{X}, \boldsymbol{\theta}_f, \boldsymbol{\theta}_y) = \mathcal{N}\left(\hat{\boldsymbol{y}} \mid \boldsymbol{m}(\mathrm{X}), \boldsymbol{K}(\mathrm{X}, \mathrm{X}) + \sigma_{\mathrm{n}}^2 \boldsymbol{I}\right). \tag{3.12}$$

3.4. Method

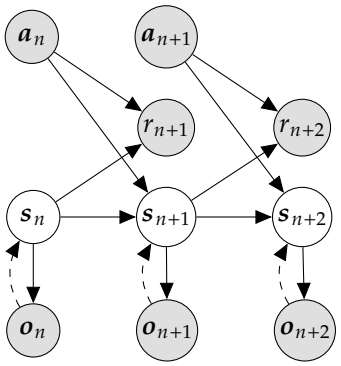
We propose a novel approach for learning system dynamics from high-dimensional image observations. We combine the advantages of deep representation learning with Gaussian processes for data-efficient Bayesian modeling of the latent system dynamics. For control, the agent requires a model of the dynamical system as well as a reward model and an encoder to infer its belief over latent states from observations. In the following, we formulate our learning framework which learns all these components jointly from applied actions, and observed images and rewards. We also propose our approach for model-predictive control and transfer learning based on our learned models.

3.4.1. Deep Gaussian process state-space models

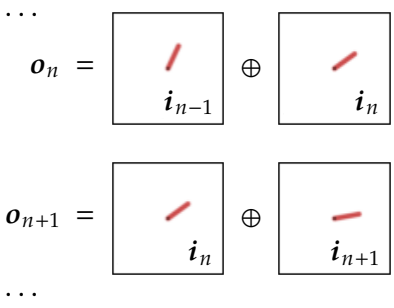
We consider dynamical systems with latent states $\boldsymbol{s} \in \mathbb{R}^{D_{\mathrm{S}}}$, actions $\boldsymbol{a} \in \mathbb{R}^{D_{\mathrm{A}}}$, and observations $\boldsymbol{o} \in [0, 1]^{D_{\mathrm{O}}}$. We call $p(\boldsymbol{s}_{n+1} \mid \boldsymbol{s}_n, \boldsymbol{a}_n) = \mathcal{N}(\boldsymbol{s}_{n+1} \mid g(\boldsymbol{s}_n, \boldsymbol{a}_n), \boldsymbol{\Sigma}_g)$ the state-transition model for discrete time steps $n \in \mathbb{N}$ with state-transition function g and output noise covariance matrix $\boldsymbol{\Sigma}_g = \mathrm{diag}(\boldsymbol{\sigma}_{\mathrm{n},g}^2)$. The observation model $p(\boldsymbol{o}_n \mid \boldsymbol{s}_n)$ is associated with the observation function h, where $\mathbb{E}[\boldsymbol{o}_n] = h(\boldsymbol{s}_n)$ is the expected observation; we will give an analytical expression for the full likelihood later. In addition, we consider a control task specified by a reward model $p(r_{n+1} \mid \boldsymbol{s}_n, \boldsymbol{a}_n) = \mathcal{N}(r_{t+1} \mid r(\boldsymbol{s}_n, \boldsymbol{a}_n), \sigma_{\mathrm{n},r}^2)$ with reward function $r(\boldsymbol{s}_n, \boldsymbol{a}_n)$ and output noise covariance $\sigma_{\mathrm{n},r}^2$. Figure 3.1a illustrates the generative process.

A possible approach to learn the above models would be to restrict g, h, and r to families of parametric functions such as deep neural networks. However, these usually require large training datasets in order to avoid overfitting. On the other hand, Bayesian nonparametric methods such as GPs often perform well with smaller datasets. We combine their respective properties and choose different types of methods for the different model components. First, we model the transition function through a GP $g \sim \mathcal{GP}(\mu_g(\cdot), k_g(\cdot, \cdot))$, with mean function $\mu_g : (\boldsymbol{s}_n, \boldsymbol{a}_n) \mapsto \boldsymbol{s}_n$ and squared exponential* (SE) kernel function k_g. For state dimensionality $D_S > 1$ we model each output dimension of the transition function with a conditionally independent GP. Similarly, we chose a GP reward model $r \sim \mathcal{GP}(r_{\min}, k_r(\cdot, \cdot))$ where $r_{\min}$ is the minimal reward observed in the collected training data, and k_r the SE kernel. We model the observation function (decoder) h with a transposed-convolutional network. With $\boldsymbol{h}_n = h(\boldsymbol{s}_n)$, the observation likelihood $p(\boldsymbol{o}_n \mid \boldsymbol{s}_n)$

* See Subsection 3.3.1 on the squared exponential kernel.



(a) State-space model with latent states $\boldsymbol{s}$, conditioned on actions $\boldsymbol{a}$, yielding observations $\boldsymbol{o}$ and rewards r. We use a simplified approximate filtering density $p(\boldsymbol{s}_n \mid \boldsymbol{o}_{\leq n}) \approx q(\boldsymbol{s}_n \mid \boldsymbol{o}_n)$ (dashed arrows).



(b) We model observations $\boldsymbol{o}_n$ as stacked subsequent images $\boldsymbol{i}_n \oplus \boldsymbol{i}_{n-1}$, in order to infer velocity information from a single observation.

Figure 3.1.: Generative state-space model (a) for observations, which we model as stacked images (b).

is given by

$$p(\boldsymbol{o}_n \mid \boldsymbol{s}_n) = \prod_{i=1}^{D_\mathrm{O}} (h_{n,i})^{o_{n,i}} \cdot (1 - h_{n,i})^{(1-o_{n,i})}, \tag{3.13}$$

where $h_{n,i}$ is the i^{th} element of $\boldsymbol{h}_n$ ($o_{n,i}$ is the i^{th} element of $\boldsymbol{o}_n$). For a state-space model in general, to obtain an estimate of the latent state $\boldsymbol{s}_n$ (the filtering density $p(\boldsymbol{s}_n \mid \boldsymbol{o}_{\leq n}, \boldsymbol{a}_{<n})$), past observations and actions have to be incorporated (see Subsection 2.2.2). Here, we make the simplifying assumption that from a single observation the filtering density can be sufficiently approximated (see Figure 3.1a), using a learned a probabilistic encoder of the form $q(\boldsymbol{s}_n \mid \boldsymbol{o}_n) = \mathcal{N}\left(\boldsymbol{s}_n \mid f_{\boldsymbol{\mu}}(\boldsymbol{o}_n), \boldsymbol{D}\boldsymbol{D}^\top\right)$ with $\boldsymbol{D} = \mathrm{diag}(f_\sigma(\boldsymbol{o}_n))$. The location and scale parameters $f_{\boldsymbol{\mu}}(\boldsymbol{o}_n)$, $f_\sigma(\boldsymbol{o}_n)$ are inferred by an encoder neural network f. We refer to Appendix A.2 for architectural details on the encoder f and decoder h.

To enable state inference from a single observation, we consider observations $\boldsymbol{o}_n$ to consist of two stacked subsequent frames $\boldsymbol{i}_{n-1} \oplus \boldsymbol{i}_n$ (see Figure 3.1b). This allows for approximate inference of positions and velocities, which is sufficient to fully describe the state of the pendulum task considered in our experiments. Note that for different environments with more complex states or with long-term dependencies it might not be possible to infer a state from two subsequent frames and hence be necessary to choose a more general encoder $q(\boldsymbol{s}_n \mid \boldsymbol{o}_{\leq n}, \boldsymbol{a}_{<n})$, e.g. by filtering through a recurrent encoder as done by Hafner et al. (2019).

3.4.2. Training objective

We jointly learn all parameters of the model, which include the weights of the neural networks and the hyperparameters of the GP, from interactions with the environment. Consider transitions $(\boldsymbol{o}_n, \boldsymbol{a}_n, \boldsymbol{o}_{n+1}, r_{n+1})$ collected by interacting with the environment. To model the covariances of individual data points we group the observed transitions into a joint training dataset $\mathrm{D} = (\mathrm{O}, \mathrm{A}, \mathrm{O}', \mathrm{R}')$, with $\mathrm{O} = (\boldsymbol{o}_1, \ldots, \boldsymbol{o}_{N-1})$, $\mathrm{A} = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{N-1})$, $\mathrm{O}' = (\boldsymbol{o}_2, \ldots, \boldsymbol{o}_N)$, and $\mathrm{R}' = (r_2, \ldots, r_N)$. We further define latent states $\mathrm{S} = (\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{N-1})$ and $\mathrm{S}' = (\boldsymbol{s}_2, \ldots, \boldsymbol{s}_N)$.

With the encoder $q(\boldsymbol{s}_n \mid \boldsymbol{o}_n)$ we factorize the approximate filtering density as $q(\mathrm{S} \mid \mathrm{O}) = \prod_{n=1}^{N-1} q(\boldsymbol{s}_n \mid o_n)$ and $q(\mathrm{S}' \mid \mathrm{O}') = \prod_{n=2}^{N} q(\boldsymbol{s}_n \mid \boldsymbol{o}_n)$. Similarly, with our observation model we can

write $p(\mathrm{O} \mid \mathrm{S}) = \prod_{n=1}^{N-1} p(\boldsymbol{o}_n \mid \boldsymbol{s}_n)$ and $p(\mathrm{O}' \mid \mathrm{S}') = \prod_{n=2}^{N} p(\boldsymbol{o}_n \mid \boldsymbol{s}_n)$. Finally, since we model the state transitions and the rewards through GPs, the densities $p(\mathrm{S}' \mid \mathrm{S}, \mathrm{A})$ and $p(\mathrm{R}' \mid \mathrm{S}, \mathrm{A})$ are from multivariate Gaussian distributions which we do not factorize over individual transitions.

We marginalize the data likelihood $p(\mathrm{O}', \mathrm{R}' \mid \mathrm{O}, \mathrm{A})$ in the following way:

$$
\begin{aligned}
&p(\mathrm{O}', \mathrm{R}' \mid \mathrm{O}, \mathrm{A}) \\
&\quad = \iint p(\mathrm{O}' \mid \mathrm{S}')\, p(\mathrm{S}' \mid \mathrm{S}, \mathrm{A})\, p(\mathrm{R}' \mid \mathrm{S}, \mathrm{A})\, q(\mathrm{S} \mid \mathrm{O})\, \mathrm{dS}\, \mathrm{dS}' && (3.14) \\
&\quad = \mathbb{E}_{q(\mathrm{S}' \mid \mathrm{O}')q(\mathrm{S} \mid \mathrm{O})} \left[p(\mathrm{O}' \mid \mathrm{S}')\, p(\mathrm{S}' \mid \mathrm{S}, \mathrm{A})\, p(\mathrm{R}' \mid \mathrm{S}, \mathrm{A})\, \frac{1}{q(\mathrm{S}' \mid \mathrm{O}')} \right]. && (3.15)
\end{aligned}
$$

With Jensen's inequality we obtain our training objective as a lower bound on the log-likelihood:

$$
\begin{aligned}
&\log p(\mathrm{O}', \mathrm{R}' \mid \mathrm{O}, \mathrm{A}) \\
&\quad \geq \underbrace{\mathbb{E}_{q(\mathrm{S}' \mid \mathrm{O}')} \left[\log p(\mathrm{O}' \mid \mathrm{S}') \right]}_{\text{(I): Reconstruction}} + \underbrace{\mathbb{E}_{q(\mathrm{S}' \mid \mathrm{O}')} \left[-\log q(\mathrm{S}' \mid \mathrm{O}') \right]}_{\text{(II): Encoder regularization}} \\
&\qquad + \underbrace{\mathbb{E}_{q(\mathrm{S}' \mid \mathrm{O}')q(\mathrm{S} \mid \mathrm{O})} \left[\log p(\mathrm{S}' \mid \mathrm{S}, \mathrm{A}) \right]}_{\text{(III): State transitions}} + \underbrace{\mathbb{E}_{q(\mathrm{S} \mid \mathrm{O})} \left[\log p(\mathrm{R}' \mid \mathrm{S}, \mathrm{A}) \right]}_{\text{(IV): Reward}}.
\end{aligned} \tag{3.16}
$$

We refer to Bosch et al. (2020) for a more detailed derivation. The four terms of the derived lower bound have readily interpretable roles: The first term (I) describes a (negative) reconstruction loss. Since the decoder parametrizes a Bernoulli distribution over pixel values, it is equivalent to the negative binary cross-entropy loss. The second term (II) corresponds to the *differential entropy* of the encoder $q(\boldsymbol{s}_n \mid \boldsymbol{o}_n)$ and can be interpreted as a regularization term on the encoder. For multivariate Gaussian distribution with diagonal covariance matrix $\boldsymbol{\Sigma} = \mathrm{diag}([\sigma_1^2, \ldots, \sigma_D^2])$, the differential entropy is proportional to $\sum_i \log(\sigma_i)$. Thus, the term prevents vanishing variances. Term (III) describes the likelihood of transitions in latent state space in expectation over the encoder. Since the state transitions are modeled with a GP, the inner likelihood corresponds to the so-called *marginal log-likelihood* (MLL), which is a common training objective for hyperparameter selection in Gaussian process regressions (Rasmussen and C. K. I. Williams, 2006). Similarly, (IV) shows the MLL for the reward GP, again in expectation over the encoder. For the loss terms (I), (III), and (IV) we estimate the outer expectations using a single reparametrized sample (Kingma and Welling, 2014; Rezende et al., 2014). The differential entropy (II) can be computed analytically, since the encoder provides a multivariate Gaussian distribution.

For computational tractability, we maximize the lower bound in Equation 3.16 over batched subsamples of our training dataset, which for Gaussian processes can be theoretically justified by the subset-of-data-approximation (Hayashi et al., 2020; Liu et al., 2020). To improve training stability, we normalize the encoded states $(\mathrm{S}, \mathrm{S}')$ batch-wise to zero mean and unit variance before passing them to the transition- and reward GPs. For testing we compute fixed normalization parameters from the full training dataset. Before decoding predicted states the normalization is reversed. Furthermore, we limit the signal-to-noise ratio of the squared exponential kernels of transition and reward GPs by minimizing an additional penalty term (see Appendix A.1).

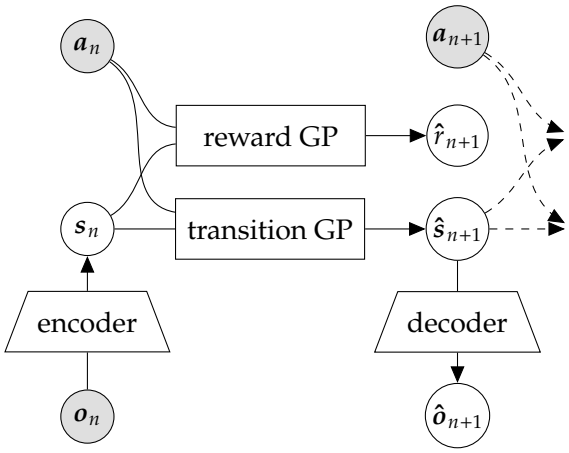


Figure 3.2.: In general, for prediction, we iteratively perform mean prediction with the transition Gaussian process, starting from the expected state $\mathbb{E}_{s_n \sim q(s_n \mid o_n)}[s_n]$ inferred from o_n using the encoder. For control, we additionally regress the reward using the reward GP. For video prediction, we can decode the predicted states $\hat{s}_n$ using the decoder.

3.4.3. Posterior inference with latent Gaussian processes

To compute the predictive distributions of the transition model and reward model, the GPs need to be conditioned on *evidence*. With the training data $\mathrm{D} = (\mathrm{O}, \mathrm{A}, \mathrm{O}', \mathrm{R}')$ and the encoder $q(\boldsymbol{s} \mid \boldsymbol{o})$ we compute latent states $(\mathrm{S}, \mathrm{S}')$, using reparametrized samples (Kingma and Welling, 2014; Rezende et al., 2014) of the predicted distribution as the state representations. For an arbitrary but known state $\boldsymbol{s}_n^*$ and action $\boldsymbol{a}_n^*$ we then obtain the posteriors $p\left(\boldsymbol{s}_{n+1}^* \mid \boldsymbol{s}_n^*, \boldsymbol{a}_n^*, ((\mathrm{S}, \mathrm{A}), \mathrm{S}')\right)$ and $p\left(r_{t+1}^* \mid \boldsymbol{s}_n^*, \boldsymbol{a}_n^*, ((\mathrm{S}, \mathrm{A}), \mathrm{R}')\right)$ through standard posterior GP inference (Rasmussen and C. K. I. Williams, 2006).

3.4.4. Model-predictive control

We employ our learned system dynamics for model-predictive control (MPC) (García et al., 1989). Key ingredients for MPC are our learned transition and reward models as well as the encoder which infers latent states from image observations. The observation model is not required for planning. Fig. 3.2 illustrates prediction with our probabilistic model. We use the cross-entropy method (CEM)[†] (Boer et al., 2005; Rubinstein, 1996) to search for the action sequence that maximizes the expected sum of rewards $\mathbb{E}\left[\sum_{t=1}^{T} r_t\right]$. Starting from the mean state encoding of the most recent observation, we compute a state trajectory by forward-propagating the predicted mean. We then approximate the expected reward by averaging the mean prediction of the reward model for 5 samples of the marginal state distribution for every timestep.

3.5. Experiments

We demonstrate our learning-based control approach on the inverted pendulum (OpenAI Gym Pendulum-v0 (Brockman et al., 2016)), a classical problem of optimal control and continuous reinforcement learning. The goal of this task is to swingup an inverted pendulum from its resting (hanging-down) position. Due to bounds on the motor torques, a straight upswing of the

[†] See Subsection 2.6.1 for details on the cross-entropy planning method.

pendulum is not possible, and the agent has to plan multiple swings to reach and balance the pendulum around the upward equilibrium.

For data collection, we excite the system with uniformly sampled random actions $a_t \sim \mathcal{U}[-2, 2]$. We initialize the system's state with angles $\theta_0 \sim \mathcal{U}[-\pi, \pi]$ and angular velocities $\dot{\theta}_0 \sim \mathcal{U}[-8, 8]$. We collect 500 rollouts for training and 3 pools of evidence rollouts, containing 200 rollouts each. Each rollout contains 28 transitions.

We represent latent states as 3-dimensional real-valued vectors $\boldsymbol{s}_n \in \mathbb{R}^3$. The observations consist of two subsequent images stacked channel-wise, with image size 64×64 pixels and RGB color channels. We model the probabilistic encoder $q(\boldsymbol{s}_n \mid \boldsymbol{o}_n)$ by a convolutional neural network with two output heads for mean and standard deviation. The decoder $p(\boldsymbol{o}_n \mid \boldsymbol{s}_n)$ mapping from the latent space to the observation space is implemented by a transposed-convolutional neural network. For more details on the architecture, see Appendix A.2.

For implementation, we use PyTorch (Paszke et al., 2019) and GPyTorch (Gardner et al., 2018).

3.5.1. Training

The lengthscales of the squared exponential kernel are initialized as $l = \text{softplus}(0) \approx 0.693$. The outputscales of the transition GPs are initialized to $\alpha^2 = 1$, the output noise variances to $\sigma_{\text{n}}^2 = 0.2$. We pose Gamma$(1, 5)$ priors on the outputscales and lower bound the outputscales to 10^{-2}. For the reward GP, the outputscale α^2_{reward} is initialized to the variance of the rewards in the first batch, the output noise variance to $0.2 \cdot \alpha^2_{\text{reward}}$. All output noise variances are lower bounded by $\alpha^2 \cdot 10^{-3}$. The mean function for the reward GP is set constant as the minimum of all collected rewards, to predict a minimal reward for unseen regions of the state space. All parameters of our model are jointly optimized with Adam (Kingma and Ba, 2015), with a learning rate of 10^{-3} and a batch-size of 1024. To encourage the encoder to learn an embedding for the forward modelling task, we stop gradients from the reward model to the encoder. We train all models for 2000 epochs. To observe the effect of neural network initialization and training data shuffling, we report control performance on three separately trained models.

We train PlaNet (Hafner et al., 2019) on the same training data like our model (500 pendulum rollouts with random initialization and random actions), plus additionally 200 rollouts which is the maximum number of evidence rollouts we use, which gives 700 rollouts in total. Based on the best average performance on 5 validation rollouts, we choose a model after 3.8 million steps of training. This model serves as the base model for fine-tuning on data from modified environments. For fine-tuning, we evaluate all models every 20k steps for the first 100k steps and every 100k steps for up to 1 million steps, and report results for models where the mean performance is best.

3.5.2. Pendulum swingup

Performance on the swingup task is evaluated on a system randomly initialized with angle $\theta_0 \sim \mathcal{U}[\pi - 0.05, \pi + 0.05]$ (pole hanging downwards) and angular velocity $\dot{\theta}_0 \sim \mathcal{U}[-0.05, 0.05]$. As evaluation metric, we report the achieved cumulative reward over 150 steps. In order to apply the DLGPD model on prediction and control tasks, the transition and reward GPs have to be conditioned on encoded observations, actions and rewards collected from previous environment

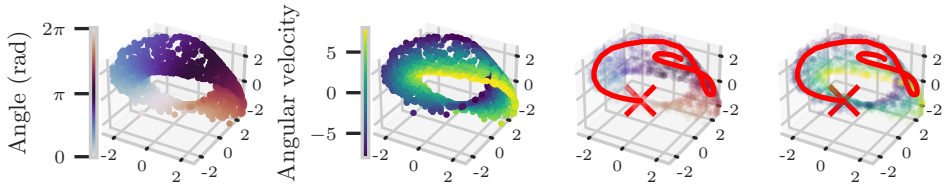


Figure 3.3.: Visualizing the 3-dimensional latent space of the learned embedding. States are colored according to true physical states of the pendulum (angle and angular velocity). On the two rightmost panels, an MPC-planned trajectory for swingup is shown, with × marking the final state.

interactions. For this we use subsets of rollouts (between 10 and 200 rollouts) from the *evidence* pools. We use CEM for planning (see Subsection 3.4.4) with a planning horizon of 20 steps. We report results for 3 control trials on 3 trained models conditioned on a subset of each of the 3 evidence pools (i.e. 27 runs per subset size). The control performance results in terms of cumulative reward are depicted in Figure 3.4(a). We observe that our approach achieves higher average cumulative reward than PlaNet in this environment already for a small set of evidence rollouts (≥ 20). In Figure 3.5(a) we additionally report the success rate on the pendulum swingup task. As there is no common definition of "success" for the *Pendulum-v0* environment, we defined a swingup trial to be successful if the last 25 steps of 150 total steps exceed a reward of -1 (0 is the maximum achievable reward for this environment, and -16.27 the minimum achievable reward). Both PlaNet and DLGPD achieve a success rate of 100 %. Fig. 3.3 shows a learned latent embedding and a trajectory followed by the CEM planner.

3.5.3. Transfer learning

By modeling the state transitions through a GP, we can learn new state-transition functions of systems with different dynamical properties in a very data-efficient way, as long as the other model components (observation model, reward model, encoder) can be re-used. In particular, we observed that the agent does not require additional training and that it is sufficient to replace the evidence in the transition GP (see Subsection 3.4.3) with new data of the modified environment, e.g. collected by a random policy. In the following, we investigate the sample efficiency of our model for adapting to environments with changed physical parameters using new random rollouts. We compare our approach with PlaNet (Hafner et al., 2019) which is fine-tuned on the same rollouts.

To evaluate adaptation capabilities to environments with changed intrinsic properties, we derive three variants of the *Pendulum-v0* environment. For the *inverted action* environment, we flip the sign of the action before passing it to the original environment. Second, we make the pole lighter by reducing its mass from $m = 1$ to $m = 0.2$; we also increase the pole's weight to $m = 1.5$. Results of PlaNet variants and DLGPD conditioned on evidence from the modified and the original environments (matching/mismatching) are shown in Figures 3.4(b-d). For inverted actions, our approach achieves significantly higher cumulative reward even for a small number of rollouts in the evidence. PlaNet clearly performs less well with the same amount of training data. With a lower mass than the original pendulum, MPC with both modelling approaches still achieves the swingup without additional data. For increased mass, our approach achieves higher

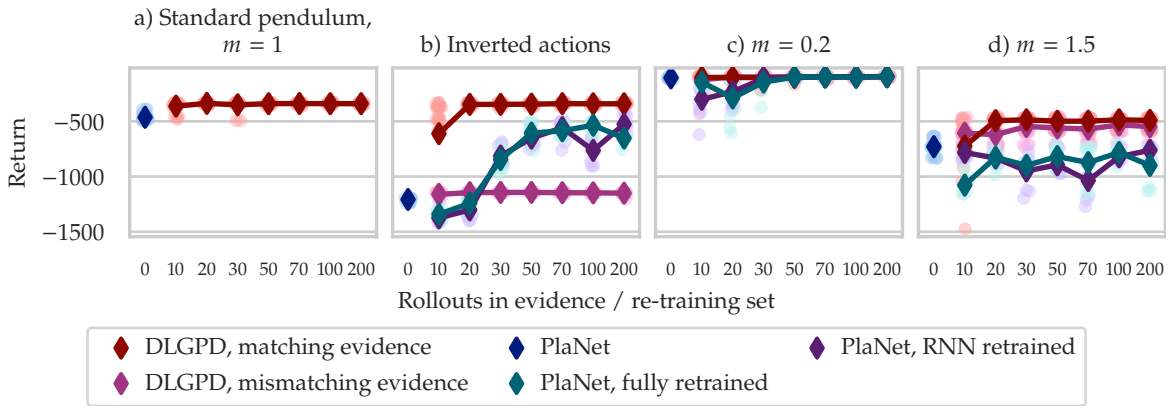


Figure 3.4.: Cumulative rewards for PlaNet and our DLGPD model for swingup of the inverted pendulum in different settings. For the detailed discussion see Subsection 3.5.2 for (a) and Subsection 3.5.3 for (b-d).

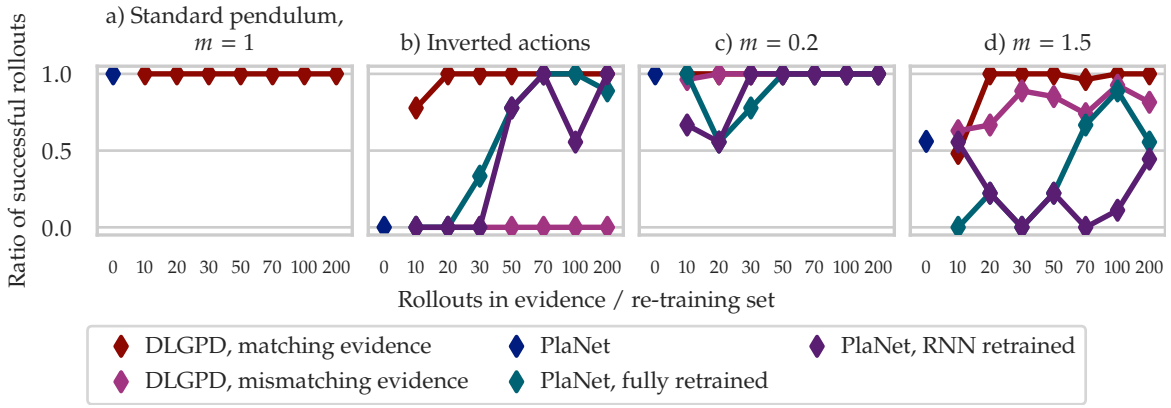


Figure 3.5.: Success rate for PlaNet and our DLGPD model for swingup of the inverted pendulum in different settings. See Subsection 3.5.3 for details.

cumulative reward than PlaNet with only a few extra rollouts. In addition to the cumulative reward, we also evaluated our experiments with respect to the ratio of successful rollouts. Please see Figures 3.5(b-d) for a visualization of the results. We observe that DLGPD matches or outperforms the success rate of PlaNet for evidence sizes ≥ 20 in all settings and is able to adapt to changes in the environment with only a few extra rollouts.

3.6. Limitations

This section is, in large parts, not part of Bosch et al. (2020).

The main limitation of the presented approach concerns applicability to more complex systems than the presented inverted pendulum.

In follow-up experiments to Bosch et al. (2020), we were not able to successfully apply the approach to perform a swing-up on the CartPole environment from the DeepMind Control Suite (Tassa et al., 2018). As a main reason, we identify the need for many datapoints in the Gaussian process conditional to sufficiently capture the system dynamics, which is prohibitive in terms of memory- and computational requirements. A sparse parametric approximation, such as pseudo-inputs (Snelson and Ghahramani, 2005), would reduce these requirements, however, at the cost of losing the ability to straightforwardly adapt to novel dynamics by simply replacing the

Gaussian process conditional. Sæmundsson et al. (2018) present a Gaussian process based latent variable model for adaptive dynamics models. Thus, a potential direction for future research is to extend the approach by Sæmundsson et al. (2018) to image inputs. However, different to our approach, this requires training on multiple environments to achieve adaptability, while our approach is trained on a single environment. An alternative to modeling the dynamics of the environment globally is to limit the datapoints in the conditional to regions of the state space which are important for a particular task. This could be achieved through collecting training data intermittently while trying to solve the task, as in PILCO (Deisenroth and Rasmussen, 2011) and PlaNet (Hafner et al., 2019).

Second, related to the first limitation, the computational complexity of forward predictions is in the order of $\mathcal{O}(N^2)$, where N is the number of datapoints in the Gaussian process conditional. For large N, planning is very computationally intensive, which poses a challenge to meet real-time requirements on physical systems. To alleviate this problem, one could learn a model-free agent from hypothetical experience sampled from the learned forward model, as in Dyna-like approaches (Sutton, 1991), or back-propagate through the forward model, as in Hafner et al. (2020). However, to adapt to novel environment dynamics, these representations have to be retrained. This incurs additional computational operations, but no further costly interaction with the real-world system.

A further simplifying assumption is that a Markovian belief state can be estimated from two subsequent image observations. However, extending the proposed method to more complex, e.g., recurrent, encoder architectures, is straightforward.

Lastly, for planning, we forward propagate mean states without propagating their uncertainty. A probabilistic treatment of the forward prediction, e.g. through moment matching or Monte Carlo sampling, might further improve data-efficiency.

3.7. Conclusion

We propose DLGPD, a dynamics model learning approach which combines deep neural networks for representation learning from images with Gaussian processes for modelling dynamics and rewards in the latent state representation. We jointly train all model parameters from example rollouts in the environment and demonstrate model-predictive control on the inverted pendulum swingup task. Our latent GP transition model allows for data-efficient transfer to tasks with modified pendulum dynamics without additional training, by conditioning the transition GP on rollouts from the modified environment. In comparison to a state-of-the-art purely deep learning based approach (PlaNet, Hafner et al. (2019)) our method demonstrates superior performance in data-efficiency for transfer learning.

Scaling and evaluating our approach on more complex tasks, environments and eventually robotic systems is an interesting topic for future research, and we refer to Section 3.6 for challenges in this regard.

4. Active Inference for Adaptive Dynamics Models

Declaration of contributions

The contents of this chapter are based on the peer-reviewed conference publication

J. Achterhold and J. Stueckler (2021). 'Explore the Context: Optimal Data Collection for Context-Conditional Dynamics Models'. In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)* (Achterhold and Stueckler, 2021).

The above publication was presented as a poster presentation at the *International Conference on Artificial Intelligence and Statistics (AISTATS) 2021, online event*.

Author contributions are as follows:

	Scientific ideas	Data generation	Analysis & Interpretation	Paper writing
Jan Achterhold	80 %	100 %	80 %	75 %
Jörg Stückler	20 %	0 %	20 %	25 %

Jan Achterhold conceived the idea of using the uncertainty in a Neural Process based dynamics model for calibration / system identification. Jörg Stückler regularly provided feedback on the approach during its development. Jan Achterhold implemented the algorithm and performed the experiments. Jan Achterhold and Jörg Stückler wrote the paper.

We provide additional materials, including the conference poster, at `https://explorethecontext.is.tue.mpg.de/`. We provide our implementation at `https://github.com/EmbodiedVision/explorethecontext`.

4.1. Introduction

Building a sample-efficient model-based agent which can efficiently adapt to non-stationarities being present in the environment has two dimensions. First, the dynamics model itself needs to be adaptive, such as the contextual Gaussian process dynamics model presented by Deisenroth and Rasmussen (2011), which we leveraged for transfer learning in Chapter 3. Second, the *data collection process* needs to be efficient. A sequence of actions executed on an environment in order to adapt a dynamics model should carefully be chosen to reveal the maximum amount of information and minimize redundancies.

In this chapter, we focus on the efficiency of the data collection process.

In Chapter 3, the prior distribution over dynamics is defined by a Gaussian process prior with a squared exponential kernel, which expresses an assumption of smoothness in the dynamics. A posterior predictive model is obtained through conditioning on transitions from the environment (the *context observations*).

Here, we also leverage contextual models to implement adaptivity. However, we depart from the Gaussian process formulation due to its computational complexity, and build our dynamics model upon the Neural Process (NP) (Garnelo et al., 2018b) framework.

The Neural Process formulation differs from the Gaussian process formulation in two ways. First, instead of manually defining structure of dynamics in form of a prior over functions, here, the structure is learned by dynamics from a given distribution over environments (e.g., pendulum environments with varying pole masses). Second, an environment-specific latent variable captures the specifics of the dynamics of a particular environment instance. We assume this latent variable to be unobserved, and inferred through a set of context transitions. Overall, the model is context-adaptive as the Gaussian process models mentioned above.

This process can also be interpreted as a form of meta-learning (Finn et al., 2017; Thrun and Pratt, 1998), as we learn to adapt a shared dynamics model (conditioned by a latent variable) to a specific environment.

Neural Process based dynamics models were previously proposed by several authors. B. Zhu et al. (2020) demonstrate that a Neural Process based dynamics model allows to infer underlying physical quantities through the context-conditional latent variable. Lee et al. (2020) encode past transitions into a latent variable, yielding a dynamics model which can adapt to changes in dynamics. Both works do not explicitly make use of the *uncertainty* of the latent variable. Our main contribution presented in this chapter is to leverage this uncertainty for *active system identification*.

We ask: Given an environment from a family of environments (e.g., the family of pendulums with varying pole masses), what action sequence should one apply to the (unknown) environment to identify it as quickly as possible within that family (e.g., to infer the mass of the pendulum's pole)? An optimization procedure seeks for transitions of the environment being most informative for the latent variable, while respecting the dynamical constraints of the environment.

The latent variable formulation allows us to formulate a calibration procedure which optimizes for a sequence of actions that minimizes uncertainty in the latent context variable. We formulate an open-loop and model-predictive calibration algorithm to plan for the optimal sequence of actions to execute.

On an illustrative toy environment (Subsection 4.5.2), we show that the uncertainty in the latent context variable corresponds to the informativeness of the set of context observations about the latent factors.

We also apply our method to a modified "Pendulum" environment from OpenAI Gym (Brockman et al., 2016) and a modified "MountainCar" environment (Moore, 1990) which we both extended by varying properties of the underlying dynamics (Subsections 4.5.3 and 4.5.4). Our algorithm exhibits a reasonable and explainable behavior on these environments. The calibration procedure we propose yields a calibration sequence which outperforms a random calibration sequence in terms of prediction accuracy of the calibrated dynamics model, and in terms of planning performance when using the calibrated dynamics model in a model-predictive control setting.

In summary, our contributions are as follows:

- We apply the framework of Neural Processes (Garnelo et al., 2018b) with a probabilistic context encoder to formulate a latent dynamics model. We demonstrate in experiments that

the probabilistic model yields meaningful posterior uncertainties in the context variable given observations of dynamical systems.

- Based on this probabilistic formulation, we develop an information-theoretic calibration scheme based on expected information gain (EIG) and model-predictive control (MPC). We further demonstrate that our calibration scheme outperforms a baseline which generates actions randomly, in terms of prediction accuracy and planning performance of the calibrated model.

4.2. Related work

Neural Processes The idea of context-conditional modeling of distributions over functions using a permutation invariant context embedding was first introduced by Garnelo et al. (2018a) as Conditional Neural Processes (CNPs). The authors apply CNPs on image completion and classification tasks. While Garnelo et al. (2018a) mainly use a deterministic belief for the context encoding, also a latent variable model is introduced, but a deterministic influence of the latent context encoding on function predictions remains. Garnelo et al. (2018b) present a more formal treatment of the latent variable model formulation coined Neural Processes (NPs). Sequential Neural Processes (Singh et al., 2019) extend the original Neural Processes formulation by a temporal dynamics model on the latent context embedding. Applications of the Neural Process framework for dynamics model learning have recently been proposed by B. Zhu et al. (2020), who show that the latent variable relates to underlying physical quantities of simulated systems, and Lee et al. (2020), who formulate adaptive dynamics models for model-based reinforcement learning tasks. In contrast to these works, in our work the *uncertainty* in the latent variable is leveraged for active system identification / calibration.

Dynamics Model Learning In the seminal PILCO approach (Deisenroth and Rasmussen, 2011), Gaussian Process (GP) dynamics models are learned for control tasks such as cart-poles. Fraccaro et al. (2017) propose an approach for modelling systems with partial observability through linear time-dependent models, with state inference performed by Kalman filtering in the latent space of a variational autoencoder. To enable planning in the learned dynamics models, Watter et al. (2015) learn locally linear models. Probabilistic dynamics models in combination with planning based on the cross-entropy method (Rubinstein, 1999) have shown to outperform model-free reinforcement learning approaches in terms of sample efficiency (Chua et al., 2018) and can be trained directly on image representations (Hafner et al., 2019). These methods do not consider variations in the underlying system which can be explained through context variables. We develop a probabilistic context-dependent dynamics model and an information-theoretic planning scheme for calibration based on model-predictive control.

Meta-Learning Neural- and Gaussian processes can be seen as types of meta learning algorithms which facilitate few-shot learning of the data distribution (Garnelo et al., 2018a,b). Meta learning of dynamics models has been explored in the domain of model-based reinforcement learning (Nagabandi et al., 2019; Sæmundsson et al., 2018). Nagabandi et al. (2019) propose to combine gradient-based (Finn et al., 2017) and recurrence-based (Duan et al., 2016) meta learning for online adaptation of the dynamics model. Different to our approach, this meta

learning scheme does not explicitly model the dynamics model dependent on a context variable. Calibration on the target system requires fine-tuning the deep neural network. We propose a deep probabilistic model and a learning scheme which allows for inferring such a context variable through calibration. Similar to our approach, Sæmundsson et al. (2018) include a latent context variable into a probabilistic hierarchical dynamics model which they choose to model using Gaussian processes. The method uses probabilistic inference to determine a Gaussian context variable from data. We use a deep encoder to regress probabilistic beliefs on the context variable and propose an information-theoretic approach for dynamics model calibration.

Active Learning and Exploration The method we propose for system calibration differs in substantial points from what is termed *exploration* in reinforcement learning. While in exploration the goal of the agent is to visit previously unseen regions of the state space for potentially finding behaviors yielding higher returns, we assume to stay in the domain of systems we observed during training. However, some concepts from exploration approaches in reinforcement learning translate to our method, especially those based on uncertainty- and information-theoretic active learning principles (Epshteyn et al., 2008; Golovin et al., 2010; Sekar et al., 2020; Shyam et al., 2019; Tschantz et al., 2020). Buisson-Fenet et al. (2020) develop an active learning approach for GP dynamics models exploiting the GP predictive uncertainty. Popular examples of active learning strategies are expected error reduction (EER) (Roy and McCallum, 2001) or expected information gain (EIG) (Lindley, 1956; MacKay, 1992). Our information-theoretic calibration approach is formulated based on a variant of EIG.

4.3. Preliminaries

This section is not part of Achterhold and Stueckler (2021).

4.3.1. Optimal Design of Experiments

In an experiment, one tries to obtain *information* on a subject. Exemplarily, medical researchers perform clinical trials to gain information on the response of patients on a particular treatment or drug, or social scientists perform questionnaires to survey political sentiments.

Oftentimes, experiments are costly, and to perform experiments which are actually informative about the quantities of interest is crucial. The set of choices on how to conduct an experiment, e.g., the selection of test subjects and questionnaire questions, are referred to its *design*.

The field of *Optimal Design of Experiments* (Pukelsheim, 2006) is concerned with mathematically approaching the question of how a (sequence of) experiments should be designed such that it is maximally informative. This ultimately helps in reducing the number of experiments required to obtain the desired quantities of interest. Bayesian optimal experimental design (see Chaloner and Verdinelli (1995) and E. G. Ryan et al. (2016) for an overview) specifically leverages Bayesian methods for this purpose.

For further treatment, let us first formalize the process of an experiment (see Figure 4.1): An experimenter chooses an experiment design $\boldsymbol{\eta}$ from the space of designs $\mathcal{D}$, in order to gain

information on parameters $\boldsymbol{\beta} \in \mathcal{B}$ of the experiment's subject. After performing an experiment, the experimenter observes an outcome $\boldsymbol{y} \in \mathcal{Y}$.

A single experiment might not be sufficient to estimate the parameters of interest. In this case, a *sequence* of experiments needs to be designed. Each new design can be based on the outcomes of the previous experiments $\boldsymbol{y}$. In literature, this is referred to as *adaptive* or *sequential* experimental design (E. G. Ryan et al. (2016) also contains a survey on sequential methods).

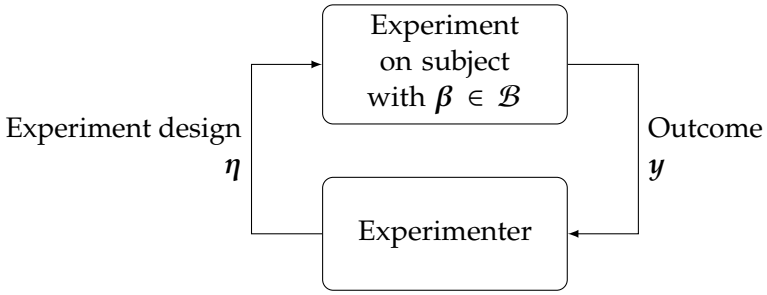


Figure 4.1.: Sequential experiment process

The above problem formulation resembles that of a *sequential decision making process* as introduced at the very beginning of this thesis in Chapter 1. The experimenter is an agent. The subject and the process of experimentation comprise the environment, which is commanded with a particular experiment design $\boldsymbol{\eta}$ (the action), and outputs the outcome $\boldsymbol{y}$ (the observation).

However, an optimality criterion still needs to be defined, which judges the quality of an experiment executed. To judge the quality of a single experiment, Lindley (1956) propose the *Expected Information Gain (EIG)*. The information gain (IG) measures the reduction in Shannon (differential) entropy of the distribution on the parameter vector $\boldsymbol{\beta}$ from before the experiment $\mathrm{H}[p(\boldsymbol{\beta})]$ to after the experiment $\mathrm{H}[p(\boldsymbol{\beta} \mid \boldsymbol{y}, \boldsymbol{\eta})]$, i.e.,

$$\mathrm{IG}(\boldsymbol{y}, \boldsymbol{\eta}) = \mathrm{H}[p(\boldsymbol{\beta})] - \mathrm{H}[p(\boldsymbol{\beta} \mid \boldsymbol{y}, \boldsymbol{\eta})]. \tag{4.1}$$

This information gain can be measured *after* $\boldsymbol{y}$ has been observed, i.e., after the experiment has been conducted. The experimental design, however, happens *before* conducting the experiment. At this point, the random variable $\boldsymbol{y}$ is unknown. To this end, Lindley (1956) proposes to use the expected information gain instead, with

$$\mathrm{EIG}(\boldsymbol{\eta}) = \mathbb{E}_{\boldsymbol{y} \sim p(\boldsymbol{y} \mid \boldsymbol{\eta}, \boldsymbol{\beta}) p(\boldsymbol{\beta})} \left[\mathrm{H}[p(\boldsymbol{\beta})] - \mathrm{H}[p(\boldsymbol{\beta} \mid \boldsymbol{y}, \boldsymbol{\eta})] \right], \tag{4.2}$$

taking the expectation over outcomes $\boldsymbol{y}$.

The expected information gain is particularly well suited for the problem of sequential experimental design, as the posterior of the previous experiment can be used as prior $p(\boldsymbol{\beta})$ for the next experiment (E. G. Ryan et al., 2016).

In this chapter, we evaluate a sequence of experiment designs, which are *actions* in our case, by their overall expected information gain on a latent variable $\boldsymbol{\beta}$, i.e.,

$$\mathrm{EIG}(\boldsymbol{\eta}_{1:N}) = \mathbb{E}_{\boldsymbol{y}_{1:N} \sim p(\boldsymbol{y}_{1:N} \mid \boldsymbol{\eta}_{1:N}, \boldsymbol{\beta}) p(\boldsymbol{\beta})} \left[\mathrm{H}[p(\boldsymbol{\beta})] - \mathrm{H}[p(\boldsymbol{\beta} \mid \boldsymbol{y}_{1:N}, \boldsymbol{\eta}_{1:N})] \right], \tag{4.3}$$

and, in a model-based planning fashion, optimize this action sequence leveraging a learned model of the environment and a planner.

In related work, Foster et al. (2019) propose variational approximations for EIG estimation to infer an informative sequence of experiments. Differently, we learn a context-dependent dynamics model which facilitates EIG estimation from the latent context posterior under dynamics constraints. However, their approach relates to our method as they proposed to use function approximators for amortized inference, which is similar to our context encoder, giving an amortized posterior for the latent context variable.

4.3.2. Neural Processes

In Section 2.5 we have seen latent variable models for single datapoints, and in Subsection 2.5.3 an implementation of a latent variable model with function approximators as observation- and recognition models. In this section, we will present the idea of *Neural Processes* (Garnelo et al., 2018b), in which the latent variable $\boldsymbol{\beta}$ parametrizes a *function* $g_\theta(\boldsymbol{x}, \boldsymbol{\beta})$. As in variational autoencoders (Kingma and Welling, 2014), the observation- and recognition model are implemented with neural networks.

Observational data, indexed over $k \in \{1, \dots, K\}$, is assumed to be generated by $\boldsymbol{y}^{(k)} = f(\boldsymbol{\alpha}, \boldsymbol{x}^{(k)}) + \boldsymbol{\epsilon}^{(k)}$. The unobserved variable $\boldsymbol{\alpha} \in \mathcal{P}_\alpha$ modulates the behavior of the function f. Gaussian additive noise $\boldsymbol{\epsilon}^{(k)} \sim \mathcal{N}(0, \boldsymbol{Q}_\epsilon)$, which is independent among the datapoints, perturb the observations. The covariance matrix $\boldsymbol{Q}_\epsilon$ can be constant, as in Garnelo et al. (2018b), or, more generally, depending on the input $\boldsymbol{x}^{(k)}$ and latent variable $\boldsymbol{\alpha}$, e.g., with $\boldsymbol{Q}_\epsilon = \operatorname{diag} \sigma_\epsilon^2(\boldsymbol{\alpha}, \boldsymbol{x}^{(k)})$, where σ_ϵ^2 is a function $\sigma_\epsilon^2 : \mathcal{P}_\alpha \times \mathbb{R}^D \to \mathbb{R}^D_{\geq 0}$.

In the general case, the function f, latent variable $\boldsymbol{\alpha}$ and covariance matrix $\boldsymbol{Q}_\epsilon$ are unknown. We introduce the superscript $\cdot^\alpha$ for $\boldsymbol{x}$ and $\boldsymbol{y}$ to denote that $\boldsymbol{y}^{(\alpha,k)}$ is generated from $\boldsymbol{x}^{(\alpha,k)}$ and $\boldsymbol{\alpha}$. To model the process, an observation model is employed, which models observations from a particular function instance parametrized by $\boldsymbol{\alpha}$ as $\boldsymbol{y}^{(\alpha,k)} = g_\theta(\boldsymbol{\beta}^{(\alpha)}, \boldsymbol{x}^{(\alpha,k)}) + \boldsymbol{\omega}^{(k)}$ with Gaussian additive noise $\boldsymbol{\omega}^{(k)} \sim \mathcal{N}(0, \boldsymbol{Q}_\omega)$. The variable $\boldsymbol{\theta}$ captures parameters of the function g. Again, $\boldsymbol{Q}_\omega$ can be constant as in Garnelo et al. (2018b), or depend on $\boldsymbol{\beta}^{(\alpha)}$ and $\boldsymbol{x}^{(\alpha,k)}$. The parameter $\boldsymbol{\theta}$ also captures the properties of $\boldsymbol{Q}_\omega$. The latent variable $\boldsymbol{\beta}^{(\alpha)}$ is inferred from a *context set* C^α, containing pairs of $(\boldsymbol{y}^{(\alpha,k)}, \boldsymbol{x}^{(\alpha,k)})$ generated by $f(\cdot, \boldsymbol{\alpha})$

$$C^\alpha = \{(\boldsymbol{y}^{(\alpha,k)}, \boldsymbol{x}^{(\alpha,k)})\}_{k \in C_{\text{idx}}}, \tag{4.4}$$

indexed by the index set C_{idx}. As the true posterior $p_\theta(\boldsymbol{\beta} \mid C^\alpha)$ is intractable, for inference on $\boldsymbol{\beta}$, a recognition model $q_{\text{ctx}}(\boldsymbol{\beta} \mid C^\alpha)$ is employed as in Subsection 2.5.3, called *context encoder*.

For a function instance $\boldsymbol{\alpha}$ and a given context set C^α, the training objective is to find $\boldsymbol{\theta}$ which maximizes the log-likelihood of target values D_y^α at target locations D_x^α

$$\log p_\theta(\mathrm{D}_y^\alpha \mid \mathrm{D}_x^\alpha, C^\alpha) \tag{4.5}$$

with

$$\mathrm{D}_x^\alpha = [\boldsymbol{x}^{(\alpha,k)}]_{k \in \mathcal{D}_{\text{idx}}}, \quad \mathrm{D}_y^\alpha = [\boldsymbol{y}^{(\alpha,k)}]_{k \in \mathcal{D}_{\text{idx}}}, \tag{4.6}$$

indexed by the index set $\mathcal{D}_{\text{idx}}$. The context index set and target index set are disjoint.

We slightly deviate from the derivations in Garnelo et al. (2018b). For notational convenience, we combine target locations and target values to the target data

$$\mathrm{D}^\alpha = (\mathrm{D}_x^\alpha, \mathrm{D}_y^\alpha). \tag{4.7}$$

We generalize the objective in Equation 4.5 to maximizing

$$\log p_\theta(\mathrm{D}^\alpha \mid C^\alpha) = \log p_\theta(\mathrm{D}_y^\alpha, \mathrm{D}_x^\alpha \mid C^\alpha) = \log p_\theta(\mathrm{D}_y^\alpha \mid \mathrm{D}_x^\alpha, C^\alpha) + \log p(\mathrm{D}_x^\alpha \mid C^\alpha). \tag{4.8}$$

However, we assume $p(\mathrm{D}_x^\alpha \mid C^\alpha)$ to be constant, such that the two objectives are equivalent up to an offset.

The method should be capable of handling different context set sizes. Therefore, the overall objective is an expectation over the parameter $\boldsymbol{\alpha}$, sampled target data of fixed size T, and a sampled context set of random size C. With that, the final objective is written as

$$\max_{\boldsymbol{\theta}} \mathbb{E}_{\substack{\boldsymbol{\alpha}\in\Omega_\alpha, C\in\Omega_C, \\ \mathrm{D}^\alpha, C^\alpha \sim \Omega_{f^\alpha}(C,T)}} \log p_\theta(\mathrm{D}^\alpha \mid C^\alpha). \tag{4.9}$$

In the above equations, Ω_α and Ω_C denote distributions on the latent parameter $\boldsymbol{\alpha}$ and the context set size C, respectively, and $\Omega_{f^\alpha}(C, T)$ is a distribution over context- and target sets with prescribed sizes.

As in Subsection 2.5.3, the approach Garnelo et al. (2018b) choose for the optimization problem in Equation 4.9 is via variational inference. First, a variational inference objective is posed to obtain an approximate posterior density $q_{\mathrm{ctx}}(\boldsymbol{\beta} \mid \mathrm{D}^\alpha, C^\alpha)$ for the true posterior $p(\boldsymbol{\beta} \mid \mathrm{D}^\alpha, C^\alpha)$ with

$$\boldsymbol{\theta}^*, \boldsymbol{\phi}^* = \min_{\boldsymbol{\theta},\boldsymbol{\phi}} \mathrm{KL}[q_{\mathrm{ctx}}(\boldsymbol{\beta} \mid \mathrm{D}^\alpha, C^\alpha) \,||\, p_\theta(\boldsymbol{\beta} \mid \mathrm{D}^\alpha, C^\alpha)]. \tag{4.10}$$

Here, $\boldsymbol{\phi}$ denotes the parameters of the recognition model $q_{\mathrm{ctx}}(\boldsymbol{\beta} \mid \mathrm{D}^\alpha, C^\alpha)$.

With the conditioned version of Bayes' theorem

$$p_\theta(\boldsymbol{\beta} \mid \mathrm{D}^\alpha, C^\alpha) = \frac{p_\theta(\mathrm{D}^\alpha \mid \boldsymbol{\beta}, C^\alpha) p_\theta(\boldsymbol{\beta} \mid C^\alpha)}{p_\theta(\mathrm{D}^\alpha \mid C^\alpha)} \tag{4.11}$$

and the assumption that the target data D^α is independent of the context set C^α given $\boldsymbol{\beta}$, i.e. $p_\theta(\mathrm{D}^\alpha \mid \boldsymbol{\beta}, C^\alpha) = p_\theta(\mathrm{D}^\alpha \mid \boldsymbol{\beta})$, we arrive at the following bound for the predictive posterior log-likelihood (following derivations in Subsection 2.5.1)

$$\log p_\theta(\mathrm{D}^\alpha \mid C^\alpha) \geq \mathbb{E}_{\boldsymbol{\beta}\sim q_{\mathrm{ctx}}(\boldsymbol{\beta} \mid \mathrm{D}^\alpha, C^\alpha)} \left[\log p_\theta(\mathrm{D}^\alpha \mid \boldsymbol{\beta})\right] - \mathrm{KL}\left[q_{\mathrm{ctx}}(\boldsymbol{\beta} \mid \mathrm{D}^\alpha, C^\alpha) \,||\, p_\theta(\boldsymbol{\beta} \mid C^\alpha)\right]. \tag{4.12}$$

In Garnelo et al. (2018b), $q_{\mathrm{ctx}}(\boldsymbol{\beta} \mid C^\alpha)$ is also used to approximate the intractable posterior density $p_\theta(\boldsymbol{\beta} \mid C^\alpha)$, yielding

$$\log p_\theta(\mathrm{D}^\alpha \mid C^\alpha) \gtrapprox \mathbb{E}_{\boldsymbol{\beta}\sim q_{\mathrm{ctx}}(\boldsymbol{\beta} \mid \mathrm{D}^\alpha, C^\alpha)} \left[\log p_\theta(\mathrm{D}^\alpha \mid \boldsymbol{\beta})\right] - \mathrm{KL}\left[q_{\mathrm{ctx}}(\boldsymbol{\beta} \mid \mathrm{D}^\alpha, C^\alpha) \,||\, q_{\mathrm{ctx}}(\boldsymbol{\beta} \mid C^\alpha)\right]. \tag{4.13}$$

Le et al. (2018) and Volpp et al. (2021) point out that, due to this approximation, Equation 4.13 is not a proper evidence lower bound. As in Subsection 2.5.3, the right-hand side of Equation 4.13

is maximized with respect to $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ for jointly learning the observation model and recognition model (context encoder).

Permutation invariance As it encodes context data, the recognition model q_{ctx} is called context encoder. Its input is a *set* of contextual observations. Consequently, the output of the context encoder should not depend on the *order* of context observations — it should be invariant to permutations in the context observations. This requires a careful architectural design of the context encoder. Garnelo et al. (2018b) leverage ideas from *Deep Sets* (Zaheer et al., 2017), which achieves permutation invariant set encodings through a permutation invariant aggregation operation (e.g., sum, mean, max) on embeddings of set elements.

Conditional distribution Similar to Equation 4.8, $\log p_\theta(\mathrm{D}^\alpha \mid \boldsymbol{\beta})$ decomposes to

$$\log p_\theta(\mathrm{D}^\alpha \mid \boldsymbol{\beta}) = \log p_\theta(\mathrm{D}_y^\alpha \mid \mathrm{D}_x^\alpha, \boldsymbol{\beta}) + \log p_\theta(\mathrm{D}_x^\alpha \mid \boldsymbol{\beta}). \tag{4.14}$$

The factor $p_\theta(\mathrm{D}_y^\alpha \mid \mathrm{D}_x^\alpha, \boldsymbol{\beta})$ thus, in combination with the context encoder $q_{\text{ctx}}(\boldsymbol{\beta} \mid C^\alpha)$, allows making *contextual* predictions.

Relation to Gaussian processes Overall, the possibility to make contextual prediction with invariance to permutations in the context set, the fact that the combination of latent variable $\boldsymbol{\beta}$ and the observation model $p_\theta(\mathrm{D}_y^\alpha \mid \mathrm{D}_x^\alpha, \boldsymbol{\beta})$ defines a distribution over functions, and the ability to estimate uncertainty in its predictions, makes the Neural Process model behave similarly to a Gaussian process regression model (see Subsection 3.3.1). However, compared to Gaussian processes, the Neural Process features a computational advantage. The computational complexity of Gaussian process regression scales cubically $\mathcal{O}(N^3)$ to the number of datapoints N in the conditional (Rasmussen and C. K. I. Williams, 2006). The Neural Process regression scales linearly $\mathcal{O}(N)$ in the number of datapoints in the conditional.

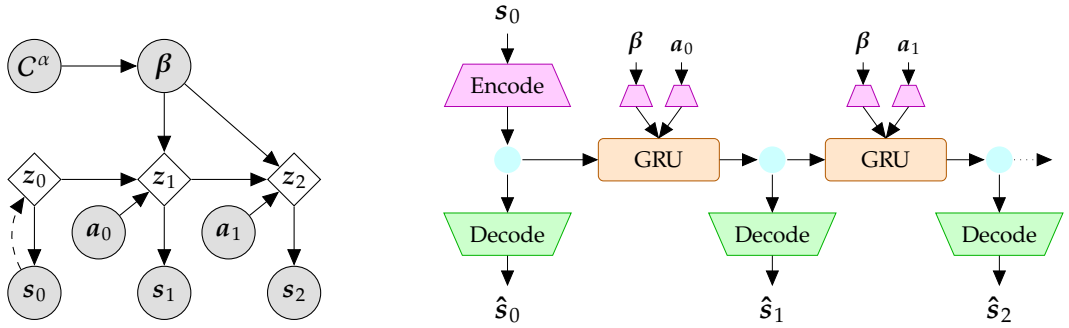
4.4. Method

We assume that the observed data of a dynamical system is generated by the following parametrized Markovian discrete-time state-space model

$$\boldsymbol{s}_{n+1} = f(\boldsymbol{s}_n, \boldsymbol{a}_n, \boldsymbol{\alpha}) + \boldsymbol{\epsilon}_n \tag{4.15}$$

with $n \in \mathbb{N}_0$, $\boldsymbol{s}_n \in \mathcal{S} \subseteq \mathbb{R}^{D_\mathrm{S}}$, $\boldsymbol{a}_n \in \mathcal{A} \subseteq \mathbb{R}^{D_\mathrm{A}}$, $\boldsymbol{\alpha} \in \mathcal{P}_\alpha$, $\boldsymbol{\epsilon}_n \sim \mathcal{N}(0, \boldsymbol{Q}_n)$, and $\boldsymbol{Q}_n$ a diagonal covariance matrix. We assume states to be fully observable. In the following, the term *rollout of length* N refers to a sequence of states and actions $\mathrm{R} = (\boldsymbol{s}_0, \boldsymbol{a}_0, \boldsymbol{s}_1, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{N-1}, \boldsymbol{s}_N)$, with Equation 4.15 holding for each *transition* $(\boldsymbol{s}_n, \boldsymbol{a}_n, \boldsymbol{s}_{n+1})$. The parameter $\boldsymbol{\alpha}$ refers to unobserved system parameters which modulate the dynamics of the system. Exemplarily, $\boldsymbol{\alpha}$ may contain actuator gains and friction coefficients of a robotic system.

The target data D^α consists of a rollout *chunk* of length T, which is a subsequence of a rollout, i.e., $\mathrm{D}^\alpha = (\boldsymbol{s}_n, \boldsymbol{a}_n, \boldsymbol{s}_{n+1}, \ldots, \boldsymbol{a}_{n+T-1}, \boldsymbol{s}_{n+T})$ of the system for fixed parameters $\boldsymbol{\alpha}$. Context data $C^\alpha = \{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}^+)\}$ with $\boldsymbol{s}^+ = f(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\alpha}) + \boldsymbol{\epsilon}$ is a set of transitions generated by the system with parameters $\boldsymbol{\alpha}$. We aim at determining an approximate probabilistic context-conditional dynamics



(a) The action-conditioned Markovian dynamics are modeled deterministically in a latent space. Observations s_n are conditioned on their respective latent state z_n. The dynamics are conditioned on actions a_n and a latent context belief β which encodes an observed context set C^α. For prediction, the initial latent state is inferred from the first observation s_0 (dashed line).

(b) Implementation of the context-conditional forward dynamics model. The initial state s_0 is embedded as hidden state of a gated recurrent unit (GRU) cell. The GRU makes a single-step forward prediction in the latent space using embeddings of the context variable β and action a as additional inputs. Latent states are mapped to Gaussian distributions on the state space for decoding.

Figure 4.2.: Graphical model (a) and implementation (b) of the proposed context-conditional forward dynamics.

model $q_{\text{fwd}}(s_n \mid s_{0:n-1}, a_{0:n-1}, C^\alpha)$ which maximizes (an approximation to) the expected data log-likelihood

$$\max_{\theta} \mathbb{E}_{\substack{\alpha\in\Omega_\alpha, C\in\Omega_C, \\ \mathrm{D}^\alpha, C^\alpha \sim \Omega_{f^\alpha}(C,T)}} \log p_\theta(\mathrm{D}^\alpha \mid C^\alpha) \tag{4.16}$$

where Ω_α is a distribution of parameter values α, Ω_C is a distribution of context set sizes, and $\Omega_{f^\alpha}(C, T)$ is a distribution over target data and context sets with context set size C and target chunk length T. The vector θ parametrizes the generative model.

4.4.1. Latent context variable

Since we cannot directly observe α and also do not know its representation, we introduce a latent variable $\beta \in D_\beta$ whose representation we learn and which encodes context information corresponding to α contained in the context set,

$$\log p_\theta(\mathrm{D}^\alpha \mid C^\alpha) = \log \int p_\theta(\mathrm{D}^\alpha \mid \beta) p_\theta(\beta \mid C^\alpha)\, \mathrm{d}\beta. \tag{4.17}$$

We model $p_\theta(\mathrm{D}^\alpha \mid \beta)$ with the *forward transition model* $q_{\text{fwd}}(\hat{s}_{n+H} \mid \hat{s}_n, a_{n:n+H-1}, \beta)$ and approximate $p_\theta(\beta \mid C^\alpha)$ by $q_{\text{ctx}}(\beta \mid C^\alpha)$, the *context encoder*. For a graphical depiction of our modeling approach, see Figure 4.2.

4.4.2. Transition model

We model sequences in a conditional auto-encoder scheme, in which we obtain predictive distributions for immediate reconstructions, single-step predictions, and multi-step predictions. Given a sequence of states $s_{0:N}$, actions $a_{0:N-1}$ and a latent context variable β, we approximate

the likelihood under the predictive model q as

$$p_\theta(\mathrm{D}^\alpha \mid \boldsymbol{\beta}) \approx \prod_{n=0}^{N} q_{\mathrm{fwd}}(\hat{\boldsymbol{s}}_n = \boldsymbol{s}_n \mid \boldsymbol{s}_n) \cdot \prod_{n=1}^{N} q_{\mathrm{fwd}}(\hat{\boldsymbol{s}}_{n|n-1} = \boldsymbol{s}_n \mid \boldsymbol{s}_{n-1}, \boldsymbol{a}_{n-1}, \boldsymbol{\beta}) \cdot \prod_{n=2}^{N} q_{\mathrm{fwd}}(\hat{\boldsymbol{s}}_{n|0} = \boldsymbol{s}_n \mid \boldsymbol{s}_0, \boldsymbol{a}_{0:n-1}, \boldsymbol{\beta}). \tag{4.18}$$

The above equations correspond to reconstruction, single-step, and multi-step prediction likelihoods. The transition model $q_{\mathrm{fwd}}(\hat{\boldsymbol{s}}_{n+H} \mid \hat{\boldsymbol{s}}_n, \boldsymbol{a}_{n:n+H-1}, \boldsymbol{\beta})$ is given as follows. We assume non-linear dynamics parametrized by $\boldsymbol{\beta}$ and disturbed by additive zero-mean Gaussian noise $\boldsymbol{\epsilon}$

$$\hat{\boldsymbol{s}}_{n+H} = h(\boldsymbol{s}_n, \boldsymbol{a}_n, \ldots, \boldsymbol{a}_{n+H-1}, \boldsymbol{\beta}) + \boldsymbol{\epsilon}. \tag{4.19}$$

We implement the dynamics by a recurrent GRU cell (Cho et al., 2014) which operates in an embedding space. The encoders e_{s}, e_{a} and e_β lift, respectively, the state $\boldsymbol{s}_n$, action $\boldsymbol{a}_n$ and latent context variable $\boldsymbol{\beta}$ to the embedding space in which the GRU operations are performed

$$\begin{aligned} \boldsymbol{z}_n &= e_{\mathrm{s}}(\boldsymbol{s}_n), \\ \boldsymbol{z}_{n+1} &= h_{\mathrm{RNN}}(\boldsymbol{z}_n, [e_{\mathrm{a}}(\boldsymbol{a}_n), e_\beta(\boldsymbol{\beta})]) \end{aligned} \tag{4.20}$$

where $[\cdot, \cdot]$ denotes concatenation. The decoders $d_{\mathrm{s},\mu}$ and d_{s,σ^2} map the propagated hidden state back to a distribution on the state space

$$\hat{\boldsymbol{s}}_{n+H} \sim \mathcal{N}\left(d_{\mathrm{s},\mu}(\boldsymbol{z}_{n+H}), \mathrm{diag}(d_{\mathrm{s},\sigma^2}(\boldsymbol{z}_{n+H}))\right). \tag{4.21}$$

4.4.3. Context encoder

The context encoder $q_{\mathrm{ctx}}(\boldsymbol{\beta} \mid C^\alpha)$ encodes a set of transitions C^α into a Gaussian belief over the latent context variable

$$q_{\mathrm{ctx}}(\boldsymbol{\beta} \mid C^\alpha) = \mathcal{N}\left(\boldsymbol{\beta} \mid d_{\beta,\mu}(\boldsymbol{z}_\beta), \mathrm{diag}(d_{\beta,\sigma^2}(\boldsymbol{z}_\beta))\right) \tag{4.22}$$

As a first step of the encoding procedure, every transition is lifted separately into an embedding space $\mathbb{R}^{D_{\mathrm{F}}}_{\geq 0}$ using the transition encoder e_{trans}, which is non-negative due to its terminal ReLU activation function

$$e_{\mathrm{trans}} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^{D_{\mathrm{F}}}_{\geq 0}, \ \boldsymbol{z} = e_{\mathrm{trans}}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}^+). \tag{4.23}$$

Then, the embedded transitions are aggregated by max pooling, making the aggregated embedding invariant to the ordering of transitions in the context set C^α, which is a common set embedding technique (Garnelo et al., 2018a; Zaheer et al., 2017)

$$[\boldsymbol{z}_\beta]_i = \max_{(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}^+) \in C^\alpha} [e_{\mathrm{trans}}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}^+)]_i, \tag{4.24}$$

where $[\cdot]_i$ denotes the i-th element of a vector. For an empty context set, we fix $\boldsymbol{z}_\beta = 0$. The mean and diagonal elements of the covariance matrix of the Gaussian belief over $\boldsymbol{\beta}$ are computed from the aggregated embedding using multilayer perceptrons

$$d_{\beta,\mu} : \mathbb{R}^{D_{\mathrm{F}}}_{\geq 0} \to \mathbb{R}^{D_\beta}, \quad d_{\beta,\sigma^2} : \mathbb{R}^{D_{\mathrm{F}}}_{\geq 0} \to \mathbb{R}^{D_\beta}_{>0}. \tag{4.25}$$

While $d_{\beta,\mu}$ follows a standard architecture, we structure d_{β,σ^2} to resemble the behavior of Bayesian inference of a latent variable given noisy measurements (Murphy, 2012). In this setting, adding a datapoint to the set of measurements cannot increase uncertainty over the latent variable. Due to the non-negativity of e_{trans} and the monotonicity of the max pooling operation, this can be achieved by requiring $d_{\beta,\sigma^2}(\cdot)$ to be monotonically decreasing in the sense

$$\begin{aligned} [d_{\beta,\sigma^2}(\boldsymbol{z})]_i &\geq [d_{\beta,\sigma^2}(\boldsymbol{z} + \boldsymbol{\Delta}_{\boldsymbol{z}})]_i \\ \forall \boldsymbol{z} \in \mathbb{R}_{\geq 0}^{D_{\mathrm{F}}}, \boldsymbol{\Delta}_{\boldsymbol{z}} &\in \mathbb{R}_{\geq 0}^{D_{\mathrm{F}}}, i \in \{1, \ldots, D_\beta\}. \end{aligned} \tag{4.26}$$

To model this strictly positive, monotonically decreasing function, we squash the negated output of a multilayer perceptron having non-negative weights and activations, $d_{\text{non-neg}}(\boldsymbol{z}_{\boldsymbol{\beta}})$ through a Softplus function $[x]^{\oplus} = \log(1 + e^x)$ (elementwise for vectors)

$$d_{\beta,\sigma^2}(\boldsymbol{z}_{\boldsymbol{\beta}}) = [-d_{\text{non-neg}}(\boldsymbol{z}_{\boldsymbol{\beta}})]^{\oplus}. \tag{4.27}$$

We fix the scale of the latent belief by forcing the latent belief for an empty context set to the unit Gaussian $\mathcal{N}(0, \boldsymbol{I})$ with a KL divergence penalty term

$$\mathcal{L}_{\mathrm{KL}}(C^\alpha) = \mathrm{KL}(q_{\mathrm{ctx}}(\boldsymbol{\beta} \mid C^\alpha = \{\}) \,||\, \mathcal{N}(0, \boldsymbol{I})). \tag{4.28}$$

4.4.4. Evidence maximization

For general non-linear dynamics models, the integral in Equation 4.17 cannot be solved analytically. Garnelo et al. (2018b) derive a lower bound on the log evidence

$$\begin{aligned} &\log p_\theta(\mathrm{D}^\alpha \mid C^\alpha) \geq \\ &\quad \mathbb{E}_{\boldsymbol{\beta} \sim q_{\mathrm{ctx}}(\boldsymbol{\beta} \mid \mathrm{D}^\alpha \cup C^\alpha)} \left[\log p_\theta(\mathrm{D}^\alpha \mid \boldsymbol{\beta})\right] - \mathrm{KL}(q_{\mathrm{ctx}}(\boldsymbol{\beta} \mid \mathrm{D}^\alpha \cup C^\alpha) \,||\, p_\theta(\boldsymbol{\beta} \mid C^\alpha)) \end{aligned} \tag{4.29}$$

which is further approximated by approximating $p_\theta(\boldsymbol{\beta} \mid C^\alpha)$ by $q_{\mathrm{ctx}}(\boldsymbol{\beta} \mid C^\alpha)$ (Garnelo et al., 2018b; Le et al., 2018; Volpp et al., 2021). With a slight abuse of notation, the expression $\mathrm{D}^\alpha \cup C^\alpha$ forms a union of all transitions in the target data chunk D^α and all transitions in the context set C^α. In this work, we train our predictive model directly for multistep prediction by using an appropriate approximation to the likelihood

$$\begin{aligned} &\mathcal{J}_{\mathrm{logll}}(\mathrm{D}^\alpha, \boldsymbol{\beta}) = \\ &\sum_{n=0}^{N} \log q_{\mathrm{fwd}}(\hat{\boldsymbol{s}}_n = \boldsymbol{s}_n \mid \boldsymbol{s}_n) + \sum_{n=1}^{N} \log q_{\mathrm{fwd}}(\boldsymbol{s}_n \mid \boldsymbol{s}_{n-1}, \boldsymbol{a}_{n-1}, \boldsymbol{\beta}) + \sum_{n=2}^{N} \log q_{\mathrm{fwd}}(\boldsymbol{s}_n \mid \boldsymbol{s}_0, \boldsymbol{a}_{0:n-1}, \boldsymbol{\beta}), \end{aligned} \tag{4.30}$$

see Equation 4.18, and a factor λ_{KL} for the KL divergence term as in Higgins et al. (2017), yielding the approximate lower bound

$$\begin{aligned} &\log p_\theta(\mathrm{D}^\alpha \mid C^\alpha) \gtrapprox \\ &\quad \mathbb{E}_{\boldsymbol{\beta} \sim q_{\mathrm{ctx}}(\boldsymbol{\beta} \mid \mathrm{D}^\alpha \cup C^\alpha)} \left[\mathcal{J}_{\mathrm{logll}}(\mathrm{D}^\alpha, \boldsymbol{\beta})\right] - \lambda_{\mathrm{KL}} \mathrm{KL}(q_{\mathrm{ctx}}(\boldsymbol{\beta} \mid \mathrm{D}^\alpha \cup C^\alpha) \,||\, q_{\mathrm{ctx}}(\boldsymbol{\beta} \mid C^\alpha)). \end{aligned} \tag{4.31}$$

The expectation operator in Equation 4.31 is approximated by a single sample $\boldsymbol{\beta} \sim q_{\mathrm{ctx}}(\boldsymbol{\beta} | \mathrm{D}^\alpha \cup C^\alpha)$. Conclusively, for a single pair of target chunk D^α and context set C^α, the training loss we minimize

with respect to parameters of the forward model q_{fwd} and context encoder q_{ctx} states

$$\mathcal{L}(\mathrm{D}^{\alpha}, C^{\alpha}) = -\left(\mathcal{J}_{\text{logll}}(\mathrm{D}^{\alpha}, \boldsymbol{\beta}) - \lambda_{\text{KL}}\mathrm{KL}\left[q_{\text{ctx}}(\boldsymbol{\beta} \mid \mathrm{D}^{\alpha} \cup C^{\alpha}) \,||\, q_{\text{ctx}}(\boldsymbol{\beta} \mid C^{\alpha})\right]\right) + \mathcal{L}_{\text{KL}}(C^{\alpha}) \tag{4.32}$$

with $\mathcal{L}_{\text{KL}}$ defined in Equation 4.28.

As the latent context belief $q_{\text{ctx}}(\boldsymbol{\beta} \mid C^{\alpha})$ is a multivariate Gaussian distribution with diagonal covariance matrix, the KL divergence term $\mathrm{KL}(q_{\text{ctx}}(\boldsymbol{\beta} \mid \mathrm{D}^{\alpha} \cup C^{\alpha}) \,||\, q_{\text{ctx}}(\boldsymbol{\beta} \mid C^{\alpha}))$ decomposes into a sum of KL divergences between scalar Gaussian distributions

$$\mathrm{KL}(q_{\text{ctx}}(\boldsymbol{\beta} \mid \mathrm{D}^{\alpha} \cup C^{\alpha}) \,||\, q_{\text{ctx}}(\boldsymbol{\beta} \mid C^{\alpha})) = \sum_{i} \mathrm{KL}(q_{\text{ctx}}([\boldsymbol{\beta}]_i \mid \mathrm{D}^{\alpha} \cup C^{\alpha}) \,||\, q_{\text{ctx}}([\boldsymbol{\beta}]_i \mid C^{\alpha})). \tag{4.33}$$

We clip $\mathrm{KL}(q_{\text{ctx}}([\boldsymbol{\beta}]_i \mid \mathrm{D}^{\alpha} \cup C^{\alpha}) \,||\, q_{\text{ctx}}([\boldsymbol{\beta}]_i \mid C^{\alpha}))$ at a minimum of 0.1 during training. This avoids local minima during the beginning of training, in which the KL divergence approaches 0 through $q_{\text{ctx}}(\boldsymbol{\beta} \mid \cdot)$ modeling a constant distribution independent of the context observations, while other loss components are not properly minimized.

We detail the minimization of the objective in Equation 4.32 on empirical samples of $\mathrm{D}^{\alpha}, C^{\alpha}$ in Subsection 4.5.1.

4.4.5. Computing optimal action sequences for calibration

We will now discuss algorithms to utilize the learned models from above to define optimal calibration schemes. We refer to “calibration" as the process of inferring unknown latent variables governing the dynamics of a system, while assuming that the model is a member of the family of dynamics models seen during training.

We formulate finding optimal actions to apply to a system for calibration as a Bayesian optimal experimental design problem with an information-theoretic utility function (Chaloner and Verdinelli, 1995; Lindley, 1956). We choose an action sequence $\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{N-1}$ to maximize the expected information gain

$$\begin{aligned} &\mathrm{EIG}(\boldsymbol{a}_{0:N-1} \mid \boldsymbol{s}_0, \mathcal{T}_0) = \\ &\quad \mathbb{E}_{\mathrm{R} \sim q_{\text{fwd}}(\mathrm{R} \mid \boldsymbol{s}_0, \boldsymbol{a}_{0:N-1}, \mathcal{T}_0)}\left[\mathrm{H}[q_{\text{ctx}}(\boldsymbol{\beta} \mid \mathcal{T}_0)] - \mathrm{H}[q_{\text{ctx}}(\boldsymbol{\beta} \mid \mathcal{T}_0 \cup \text{chop}(\mathrm{R}))]\right] \end{aligned} \tag{4.34}$$

where $\boldsymbol{s}_0$ is the current state of the system, $\mathcal{T}_0$ are already observed transitions on the system to calibrate and $\mathrm{H}[\cdot]$ represents differential entropy. The belief over the latent context variable $q_{\text{ctx}}(\boldsymbol{\beta} \mid \mathcal{T})$ after observing a set of calibration transitions $\mathcal{T}$ is given by the context encoder. The imagined rollout $\mathrm{R} = (\boldsymbol{s}_0, \boldsymbol{a}_0, \hat{\boldsymbol{s}}_1, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{N-1}, \hat{\boldsymbol{s}}_N)$ is chopped into a set of transitions $\text{chop}(\mathrm{R}) = \{(\boldsymbol{s}_0, \boldsymbol{a}_0, \hat{\boldsymbol{s}}_1), (\hat{\boldsymbol{s}}_1, \boldsymbol{a}_1, \hat{\boldsymbol{s}}_2), \ldots, (\hat{\boldsymbol{s}}_{N-1}, \boldsymbol{a}_{N-1}, \hat{\boldsymbol{s}}_N)\}$. The distribution of imagined rollouts of the system $q_{\text{fwd}}(\mathrm{R} \mid \boldsymbol{s}_0, \boldsymbol{a}_0, \ldots, \boldsymbol{a}_{N-1}, \mathcal{T}_0)$ is generated from the multi-step transition model while marginalizing out the prior belief over the latent context variable

$$\begin{aligned} &q_{\text{fwd}}(\mathrm{R} \mid \boldsymbol{s}_0, \boldsymbol{a}_{0:N-1}, \mathcal{T}_0) = \\ &\quad \int q_{\text{fwd}}(\mathrm{R} \mid \boldsymbol{s}_0, \boldsymbol{a}_{0:N-1}, \boldsymbol{\beta}) q_{\text{ctx}}(\boldsymbol{\beta} \mid \mathcal{T}_0) \mathrm{d}\boldsymbol{\beta}. \end{aligned} \tag{4.35}$$

We get a Monte Carlo approximation to the expectation in Equation 4.34 by approximate sampling from the above distribution. To obtain a sample, we first sample from the latent context

distribution $\boldsymbol{\beta}_0 \sim q_{\text{ctx}}(\boldsymbol{\beta} \mid \mathcal{T}_0)$. Next, we form the imagined rollout R as defined above, where $\hat{\boldsymbol{s}}_n$ are mean predictions from the learned transition model, i.e. $\hat{\boldsymbol{s}}_n = \mathbb{E}_{q_{\text{fwd}}(\hat{\boldsymbol{s}}_n \mid \boldsymbol{s}_0, \boldsymbol{a}_{0:n-1}, \boldsymbol{\beta}_0)}[\hat{\boldsymbol{s}}_n]$.

The action sequence which the EIG maximization yields is most informative for the latent context variable (minimizes the posterior entropy) *given the a-priori belief*, i.e., taking the a-priori uncertainty about the context variable into account.

Open-loop calibration In the open-loop calibration case, we initialize the set of already observed transitions as the empty set $\mathcal{T}_0 = \{\}$. We then optimize for a sequence of actions $\boldsymbol{a}_0^*, \dots, \boldsymbol{a}_{H-1}^*$ to fulfill

$$\begin{aligned} &\boldsymbol{a}_0^*, \dots, \boldsymbol{a}_{H-1}^* = \\ &\quad \underset{\boldsymbol{a}_0, \dots, \boldsymbol{a}_{H-1} \in \mathcal{A}^H}{\arg\max} \ \text{EIG}(\boldsymbol{a}_0, \dots, \boldsymbol{a}_{H-1} \mid \boldsymbol{s}_0, \mathcal{T}_0 = \{\}) \end{aligned} \tag{4.36}$$

using the cross-entropy method (Rubinstein (1999), see Subsection 2.6.1). We call H calibration horizon.

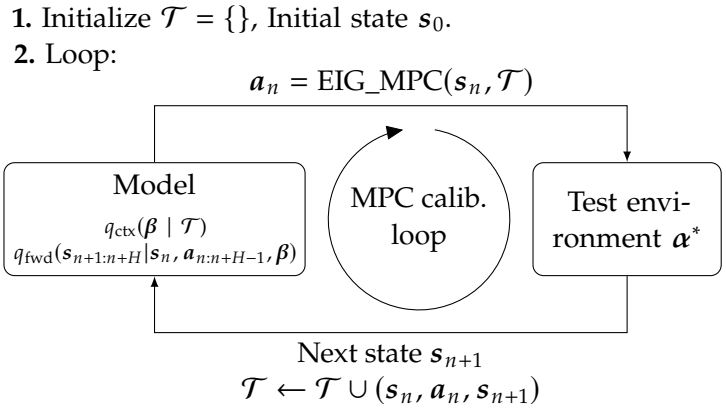
MPC calibration The open-loop calibration scheme computes a static action sequence at the beginning of the calibration procedure. The planned action sequence is not updated during calibration. However, knowledge about the system obtained through already executed system interactions may be valuable to re-plan the remaining calibration actions. Therefore, we propose a closed-loop calibration method which resembles a model-predictive control scheme known from feedback control theory, which we term *MPC calibration*. We refer to Subsection 2.3.5 for a discussion on open-loop and closed-loop planning. For MPC calibration, we compute an open-loop calibration sequence *at every timestep*, with $\mathcal{T}_n$ containing already observed transitions ($\mathcal{T}_0 = \{\}$). From this calibration sequence, the first action is applied to the system and the resulting transition is appended to $\mathcal{T}_n$, giving $\mathcal{T}_{n+1}$. This is repeated for a fixed number of timesteps N. Let $\boldsymbol{s}_n$ be the current state of the system and $\mathcal{T}_n$ contain already observed transitions. At each timestep, we optimize

$$\begin{aligned} &\boldsymbol{a}_n^*, \dots, \boldsymbol{a}_{n+H-1}^* = \\ &\quad \underset{\boldsymbol{a}_n, \dots, \boldsymbol{a}_{n+H-1} \in \mathcal{A}^H}{\arg\max} \ \text{EIG}(\boldsymbol{a}_n, \dots, \boldsymbol{a}_{n+H-1} \mid \boldsymbol{s}_n, \mathcal{T}_n), \end{aligned} \tag{4.37}$$

and apply $\boldsymbol{a}_n^*$ as next action to the system. We abbreviate Equation 4.37 as

$$\boldsymbol{a}_n^* = \text{EIG\_MPC}(\boldsymbol{s}_n, \mathcal{T}_n) \tag{4.38}$$

in case we are just interested in the first action $\boldsymbol{a}_n^*$. The planning horizon H is upper bounded by the maximal planning horizon H_{max} and the remaining steps to reach the calibration horizon as $H = \min(H_{\text{max}}, N - n)$. Due to the repeated optimization, the MPC calibration scheme is more computationally intensive than the open-loop scheme. We refer to Figure 4.3 for a visualization of the MPC calibration scheme.



To calibrate a model for a previously unseen test environment $\boldsymbol{\alpha}^*$, we iteratively compute an optimal action $\boldsymbol{a}_n$ using the expected information gain (EIG) objective. The action $\boldsymbol{a}_n$ is applied to the environment. The resulting transition is recorded as the tuple $(\boldsymbol{s}_n, \boldsymbol{a}_n, \boldsymbol{s}_{n+1})$, which is appended to $\mathcal{T}$. Finally, we obtain the calibrated model by conditioning on all transitions obtained from the calibration rollout $\mathcal{T}$.

Figure 4.3.: Model predictive control (MPC) based calibration loop.

4.5. Experiments

We evaluate our approach for learning and calibrating context-dependent dynamics models on an illustrative toy problem, a modified Pendulum environment from OpenAI Gym (Brockman et al., 2016) and a MountainCar environment with randomly sampled terrain profiles.

4.5.1. General procedure

Data collection For each experiment, we collect random data from the respective environments. For this, we first sample K environment instances, varying in their hidden parameters $\boldsymbol{\alpha}$. Then, we simulate two rollouts of length 100 on each sampled environment instance, by applying independently sampled actions starting from a random initial state. We term these rollouts $\mathrm{R}_A^{(k)}, \mathrm{R}_B^{(k)}$, where $k \in \{1, \ldots, K\}$. More specific details are given in the "Data collection" paragraph of each experiment.

Model training From the pre-generated data, we form a batch of L losses as given in Equation 4.32 for stochastic gradient descent minimization as follows

$$\mathcal{L}_{\text{batch}} = \sum_{l=1}^{L} \mathcal{L}(\mathrm{D}^{(l)}, C^{(l)}). \tag{4.39}$$

For each item l, first, an index k is sampled uniformly from $\mathcal{U}\{1, \ldots, K\}$. The target chunk $\mathrm{D}^{(l)}$ is of length 50, taken from $\mathrm{R}_A^{(k)}$, with the index of the initial state randomly sampled. For the context set, first, a context set size C is randomly sampled $C \sim \mathcal{U}\{0, \ldots, C_{\text{max}}\}$. $C^{(l)}$ is a context set of size C sampled from transitions in $\mathrm{R}_A^{(k)}, \mathrm{R}_B^{(k)}$. We refer to Appendix B.2.1 for details on the context set sampling procedure.

The batch size L, maximal context set size C_{max} and number of training environments instances K varies per environment and is detailed in the respective subsections.

We train all models using the Adam optimizer (Kingma and Ba, 2015). We evaluate the models after training for 50k steps for the toy problem and 100k steps for Pendulum and MountainCar*. We perform all evaluations on 3 independently trained models per environment and model configuration. We set the KL divergence scaling factor in Equation 4.31 to $\lambda_{\mathrm{KL}} = 5$. We train our models using the Adam optimizer (Kingma and Ba, 2015) with parameters $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1\mathrm{e}^{-4}$ and a learning rate of $1\mathrm{e}^{-3}$. We scale gradients such that the vector of concatenated gradients has a maximal 2-norm of 1000.

4.5.2. Toy system

First, we introduce an illustrative toy-problem showcasing the fundamental principles of our method. It is a discrete-time dynamical system in state-space notation

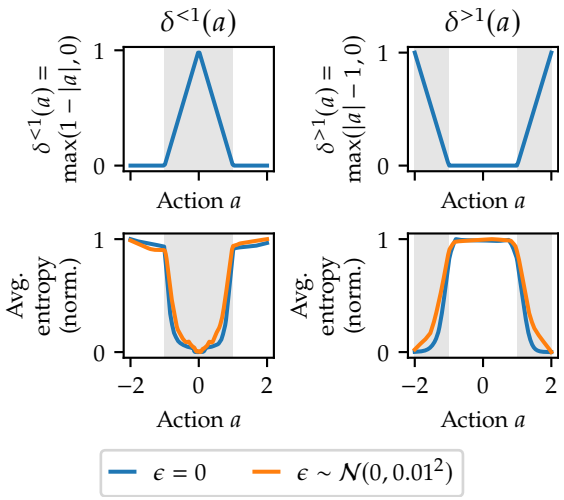
$$\boldsymbol{s}_{n+1} = \begin{pmatrix} 0.8 & 0.2 \\ -0.2 & 0.8 \end{pmatrix} \boldsymbol{s}_n + \begin{pmatrix} \alpha \\ 0 \end{pmatrix} \delta(a_n) + \boldsymbol{\epsilon}_n \tag{4.40}$$

with $\boldsymbol{\epsilon}_n \sim \mathcal{N}(0, \boldsymbol{I} \cdot (0.01)^2)$, $\boldsymbol{s}_0 \sim \mathcal{N}(0, \boldsymbol{I})$, $\alpha \in [-1, 1]$, $a_n \in [-2, 2]$. The state transition matrix leads to a damped oscillation when the system is not excited. A non-linear function δ squashes the action a before it is applied to the system. Inference about the parameter α from empirical system trajectories is only possible when at least one transition $(\boldsymbol{s}_n, a_n, \boldsymbol{s}_{n+1})$ is observed with $\delta(a_n) \neq 0$. As squashing functions we define $\delta^{<1}(a) = \max(1 - |a|, 0)$ and $\delta^{>1}(a) = \max(|a| - 1, 0)$, with the superscript denoting which a are mapped to non-zero values, thus, being informative for the value of α. We train separate models for each squashing function $\delta^{<1}(a)$, $\delta^{>1}(a)$. As we assume homoscedastic noise in this experiment, we learn a constant d_{s,σ^2} independent of $\boldsymbol{z}$. We use a batchsize $L = 64$ and maximal context set size $C_{\max} = 10$.

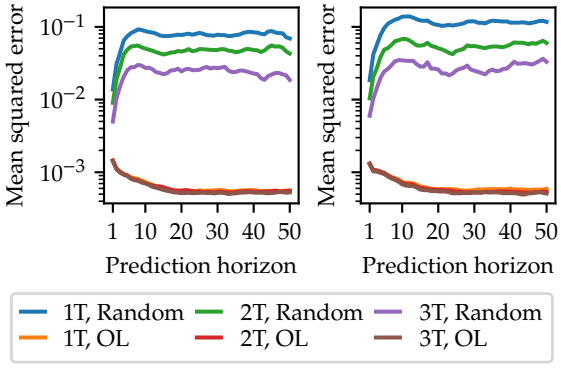
Data collection To collect training and validation data, we sample $K = 6000$ values of $\alpha \sim \mathcal{U}[-1, 1]$ and generate a rollout pair for each α, with the initial state sampled from $\boldsymbol{s}_0 \sim \mathcal{N}(0, \boldsymbol{I})$ and the control inputs from $a_n \in \mathcal{U}[-2, 2]$.

Evaluation As a first evaluation, we visualize the entropy of the latent context variable for informative and non-informative context sets. We generate random transitions $(\boldsymbol{s}_0, a_0, \boldsymbol{s}_1)$ with $\boldsymbol{\beta} \sim q_{\mathrm{ctx}}(\boldsymbol{\beta} \mid C^\alpha = \{\})$, $\boldsymbol{s}_0 = 0$, $a_0 \sim \mathcal{U}[-2, 2]$ using the learned transition models for both squashing functions $\delta^{<1}(a)$ and $\delta^{>1}(a)$. We use the learned model instead of the known ground-truth model for generating transitions to simulate the first step of a calibration procedure, in which the true model is unknown. From each transition, we construct a single-element context set and compute the latent context belief using the learned context encoder. In Figure 4.4a, we plot the entropy of the latent context belief for each action a, averaged over samples from $\boldsymbol{\beta}$. For the posed systems with $\delta^{<1}(a)$ ($\delta^{>1}(a)$), a single-element context is informative for α if and only if it contains a transition with an action $|a| < 1$ ($|a| > 1$). We observe that the entropy is minimized correspondingly for our learned model which demonstrates that the context encoder predicts uncertainty well.

* In (Achterhold and Stueckler, 2021) it has been mistakenly written that models with minimal validation loss over the course of training were selected for the final evaluations. However, due to an implementation error discovered in January 2024, effectively, the models at the end of the training procedure were used for evaluation. The repository at `https://github.com/EmbodiedVision/explorethecontext` contains a separate branch, fixing this error. Qualitatively, we observe that the two variants yield similar results.



(a) Characteristic behavior of the latent context belief encoder on the toy problem. First row: Depiction of the action squashing functions $\delta^{<1}(a)$, $\delta^{>1}(a)$ (effective action magnitude). Action regions which are informative for inferring the hidden parameter α are shaded in gray ($\delta(a) \neq 0$). Second row: Average entropy (normalized to $[0, 1]$) of the latent context belief $\mathrm{H}[q_{\mathrm{ctx}}(\beta \mid C = \{\boldsymbol{s}, a, \boldsymbol{s}^+\})]$ for actions a in systems with Gaussian transition noise $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{I} \cdot (0.01)^2)$ (orange) and $\boldsymbol{\epsilon} = 0$ (blue). Non-informative actions yield a high entropy of the latent context belief, for informative actions, the entropy negatively correlates to the effective action magnitude. Without transition noise, the entropy attains its minimum faster for increasing effective action magnitude as α can be inferred from low-magnitude actions with low variance.



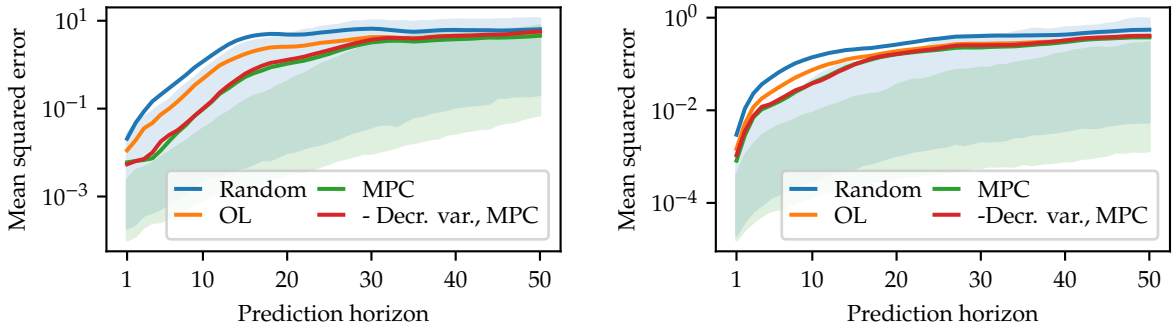
(b) Evaluation of model prediction error for the toy problem. Depicted is the prediction error (lower is better) of models with random and optimal (open-loop) calibration with $\{1, 2, 3\}$ calibration transitions, for both action squashing schemes $\delta^{<1}(a)$ (left) and $\delta^{>1}(a)$ (right).

Figure 4.4.: Evaluation of the toy-problem environment (see Subsection 4.5.2).

In a second experiment, we evaluate the model accuracy in terms of prediction error after calibration on a set of random rollouts for 3 independently trained models. We sample 50 system instances with $\alpha \sim \mathcal{U}[-1, 1]$, and for each instance we generate 20 random rollouts. For calibration, we limit the maximum number of transitions to $\{1, 2, 3\}$ and perform open-loop calibration (see Subsection 4.4.5) on each system instance. As baseline, we use randomly sampled transitions for calibration. Each calibration rollout is initialized at $\boldsymbol{s}_0 = 0$, for the random calibration rollouts we uniformly sample actions from $\mathcal{U}[-2, 2]$. In Figure 4.4b we depict the mean squared prediction error of the learned models for open-loop optimal and random calibration sequences of different lengths. The prediction error is significantly lower for optimally calibrated models compared to randomly calibrated models. As a single informative transition is sufficient to calibrate the system, MPC calibration has no advantage over open-loop calibration for this system.

4.5.3. OpenAI Gym Pendulum

In addition to the toy problem presented above, we evaluate our calibration method on a modified Pendulum environment from OpenAI Gym (Brockman et al., 2016). The Pendulum environment is a simulation of an inverted pendulum subject to gravitational force and actuable



(a) Pendulum

(b) MountainCar

Figure 4.5.: Prediction error (lower is better) of the learned (a) Pendulum and (b) MountainCar models, either conditioned on calibration data obtained with a random rollout (blue), open-loop calibration (orange), MPC calibration (green). For the red curve, we train models without the strictly-decreasing variance constraint in the context encoder and perform MPC calibration. We plot mean (line) and 20% / 80% quantiles (shaded area, for random and MPC calibration only for visual clarity) over 3000 rollouts. Our proposed calibration schemes reduce prediction error compared to random calibration. MPC calibration compares favorably to open-loop calibration. Enforcing the decreasing variance constraint in the context encoder slightly reduces model error after calibration for Pendulum. For MountainCar, both model variants perform similarly. Calibration rollouts contain 30 transitions for the Pendulum and 50 transitions for the MountainCar environment.

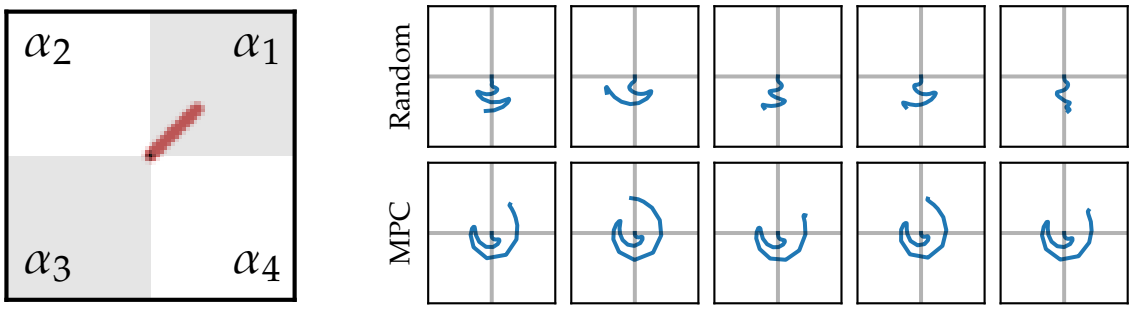
by a motor in the central rotary joint (see Figure 4.6a). Due to torque limits, the pendulum cannot reach an upright pose without following a swing-up trajectory.

After the motor torque u is clipped to a maximum magnitude of 2, we multiply it with an angle-dependent factor $u \leftarrow u \cdot s_i \cdot \alpha_i$. The sign s_i and factor α_i are sampled independently for every quadrant of the pendulum system ($i \in \{1, 2, 3, 4\}$) from $\alpha_i \sim \mathcal{U}[0.5, 2]$, $s_i \sim \mathcal{U}\{-1, 1\}$ (see Figure 4.6a). We use a batchsize $L = 512$ and maximal context set size $C_{\max} = 50$.

Data collection To collect training and validation data, we first sample $K = 110000$ environments with s_i and α_i sampled as above. For each environment, we generate 2 independent rollouts with $u_n \sim \mathcal{U}[-2, 2]$ and $\boldsymbol{s}_0 \sim \mathcal{U}[-\pi, \pi] \times \mathcal{U}[-8, 8]$, with the first dimension being the pendulum angle and the second dimension its angular velocity.

Evaluation The initial state of the pendulum for calibration is sampled from $\mathcal{U}[\pi - 0.05, \pi + 0.05] \times \mathcal{U}[-0.05, 0.05]$, i.e. with the pole pointing nearly downwards with small angular velocity. In contrast to the toy problem, we use the MPC calibration scheme with a CEM planning horizon of $H_{\max} = 20$ and a calibration rollout length of 30. To approximate the expected information gain from Equation 4.34, we use 20 Monte Carlo samples. We generate 50 environments for calibration with independently sampled s_i, α_i. For each environment, we generate 20 random rollouts with $\boldsymbol{s}_0$, u_n sampled as in the "Data collection" paragraph. On each environment, we apply our proposed calibration schemes and compare the predictive performance of the learned dynamics model for open-loop, MPC and random calibration. For random calibration, we sample random actions $u_n \sim \mathcal{U}[-2, 2]$.

Figure 4.6b depicts the angle of the pendulum over time for random and MPC calibration rollouts. Due to gravitational forces and inertia of the pendulum, the rollouts only cover the lower two quadrants. Thus, with these random calibration rollouts, the dynamics in the upper two quadrants cannot be inferred. In contrast, the MPC rollouts exhibit a swing-up behavior to reason about the dynamics in all four quadrants. Note that this behavior solely comes from the



(a) Pendulum dynamics vary per quadrant. The pendulum pole is depicted in red.

(b) Pendulum angle extruded over time (starting at center) for random and MPC calibration rollouts. In contrast to random calibration rollouts, MPC calibration rollouts cover all quadrants to infer their dynamics.

Figure 4.6.: Properties of the Pendulum environment.

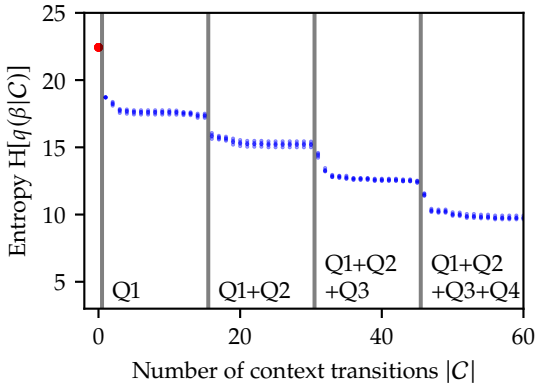
objective to minimize entropy over the latent context variable and not through explicit modelling, e.g. via rewards.

In Figure 4.5a we show the prediction error of the learned dynamics model when calibrated in the open-loop, MPC and random scheme. For each predicted state we compute the mean squared error to the ground-truth state (with the angle represented as $[\sin(\theta), \cos(\theta)]$) and plot its statistics (mean and quantiles) aggregated over 50 randomly sampled systems, 20 rollouts per system and 3 independently trained models, giving 3000 rollouts. Similarly to the results from Subsection 4.5.2, the prediction error for models calibrated with the proposed calibration schemes is lower than when calibrated with random system interventions.

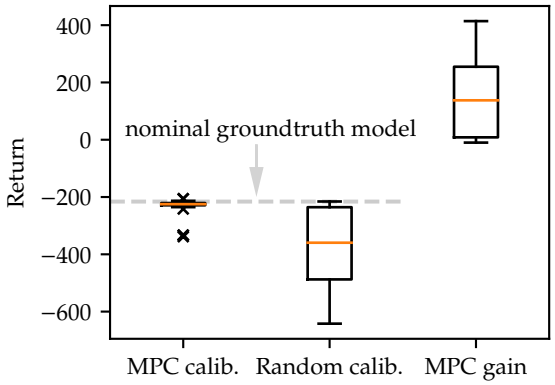
As for the pendulum task in principle a single transition is sufficient to infer the dynamics in the quadrant covered by the transition, one can expect that (1) the entropy of the latent context belief decreases for an increasing number of quadrants covered by a context set; (2) adding transitions from an already covered quadrant to the context set does not lead to a strong reduction of entropy; (3) the entropy is minimized when all four quadrants are covered. Our learned context encoder features all of those properties, as we evaluate in Figure 4.7a. The figure depicts how the information (entropy) of the latent context belief behaves for different context sets covering different numbers of quadrants. See Figure 4.7a for more details.

Control experiment In this experiment, we evaluate the performance of the calibrated models for serving as forward dynamics models in a model predictive control setting. The task is to swing-up the Pendulum into an upright pose, which requires careful trajectory planning as, due to torque limits, the Pendulum cannot be driven to the upright pose directly. Consequently, the calibrated dynamics model has to yield accurate predictions for successfully solving the task. Planning is conducted using CEM with a planning horizon of 20 and a manually constructed swing-up reward function from the OpenAI Gym Pendulum implementation (Brockman et al., 2016). For all experiments, the number of transitions in a calibration rollout is set to 30.

As depicted in Figure 4.7b, planning with a dynamics model conditioned on data collected using the MPC calibration scheme gives similar performance (return) to using a ground-truth model of the Pendulum environment. In contrast, planning with a model conditioned on randomly sampled transitions yields a significantly lower performance.



(a) Entropy of the latent context belief mainly depends on the number of covered quadrants by the context set and is barely affected by the number of context transitions larger one in each quadrant. We randomly generate 15 transitions with $u \sim \mathcal{U}[-1, 1]$ in all four quadrants (Q1–Q4) of several differently parametrized pendulum environments and sequentially add them to the context set, starting with quadrant 1. Each dot corresponds to the entropy $H[q_{\text{ctx}}(\beta \mid C)]$ of the encoded context set C, limited to the number of transitions given on the x-axis. The red dot corresponds to an empty context set. The entropy of the latent context belief saturates when transitions from already explored quadrants are added to the context set and is decreased by adding transitions from unobserved quadrants.



(b) Swingup task cumulative reward (return) on 50 randomly sampled environment instances using a learned context-conditional model and an analytic reward function for planning. The learned model is conditioned on 30 transitions collected using the MPC calibration scheme or random sampling, respectively. “MPC gain" is gain in return when using MPC calibration scheme instead of random calibration. “Nominal groundtruth model" refers to the baseline performance when planning using groundtruth dynamics as forward model with $\alpha_i = 1.25$.

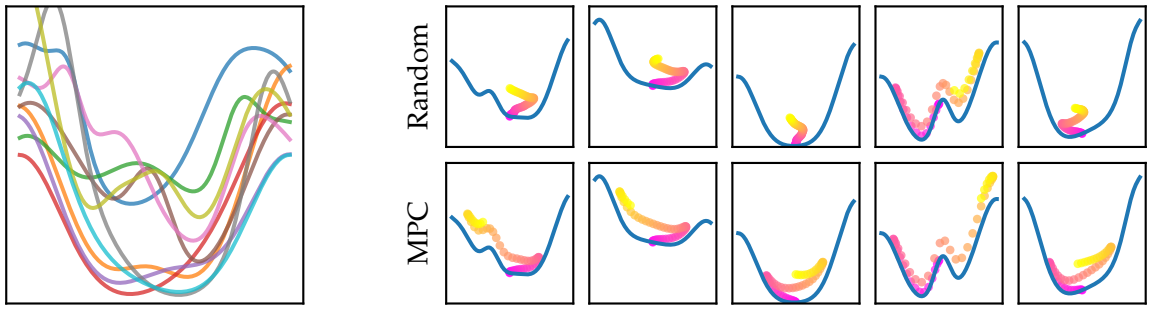
Figure 4.7.: Extended evaluations on the Pendulum environment. Qualitative evaluation of the entropy of the latent context belief (a) and task performance when using the calibrated forward model for model-predictive control (b).

4.5.4. MountainCar environment

The MountainCar environment was first introduced by Moore (1990) and is a common benchmarking problem for reinforcement learning algorithms. In its original formulation, the task is to steer a car from a valley to a hill on a 2D profile. Due to throttle constraints on the car, the agent has to learn to gain momentum by first steering in the opposite direction of the goal. We extend the MountainCar environment to randomly sampled 2D terrain shapes, train context-conditional dynamics models on random rollouts and evaluate our proposed calibration routines. Figure 4.8a illustrates randomly sampled terrain profiles. The terrain profile of the MountainCar environment is generated by a linear combination of Gaussian functions $g(x; l, w) = \exp\left(-\frac{1}{2}\frac{(x-l)^2}{w^2}\right)$

$$y = 0.5 \cdot g(x; -1, 0.3) + 0.5 \cdot g(x; 1, 0.3) + \sum_{n=1}^{N} h_n \cdot g(x; l_n, w_n) \tag{4.41}$$

with $x \in [-1, 1]$, $N \sim \mathcal{U}\{2, \dots, 7\}$, $h_n \sim \mathcal{U}[0.1, 0.3]$, $l_n \sim \mathcal{U}[-1.5, 1.5]$, $w_n \sim \mathcal{U}[0.1, 0.5]$. The Markovian state of the environment is represented by the current horizontal position and horizontal velocity. An external tangential acceleration $u \in [-3, 3]$ can be applied to the car. We locally approximate the dynamics of the car as a sliding block on an inclined plane with friction, where the slope of the plane is given by the average of the profile’s gradient at the current point and the gradient at the simulated next point, akin to Heun’s method for solving ordinary differential equations (Butcher, 2016). We use a batchsize $L = 512$ and maximal context



(a) Sampled terrain profiles of the MountainCar environment.

(b) Random (top) vs MPC (bottom) calibration rollouts on 5 sampled MountainCar profiles (pink: t=1, yellow: t=50). The MPC rollouts cover larger extents of the profile to infer its shape. For better visibility, rollouts are extruded over time towards the positive y-direction.

Figure 4.8.: Properties of the MountainCar environment.

set size $C_{\text{max}} = 50$.

Data collection For training and validation data collection, we generate $k = 60000$ MountainCar environment instances (50000 training / 10000 validation) with randomly sampled terrain profiles. On each environment instance, we generate two randomly initialized ($x_0 \sim \mathcal{U}[-0.8, 0.8]$, $\dot{x}_0 \sim \mathcal{U}[-2, 2]$) rollouts with random actions $u_n \sim \mathcal{U}[-3, 3]$.

Evaluation All calibration rollouts contain 50 transitions and are initialized at $x_0 = \dot{x}_0 = 0$. We depict the resulting random and MPC calibration rollouts in Figure 4.8b. Similar to the Pendulum environment, the MPC calibration exhibits an explorative behavior to gain information about the terrain shape. The prediction error is evaluated on 50 randomly sampled terrain profiles, 20 randomly generated rollouts per profile and 3 independently trained models, giving 3000 rollouts in total. The model prediction error is depicted in Figure 4.5b for different calibration schemes. Again, our proposed calibration schemes compare favorably to using random transitions for calibration. The CEM planning horizon for MPC calibration for this environment is set to $H_{\text{max}} = 30$ in our experiments.

4.5.5. Ablation studies

In Appendix B.4, we show studies on the performance of the calibrated models when using multiple rollouts for calibration and/or varying the number of transitions in a calibration trajectory. We find that (i) the advantage of increasing the trajectory length vanishes at some point, e.g., in the Pendulum case, when all quadrants have been observed. Using multiple calibration rollouts is not advantageous for the Pendulum environment: If the calibration trajectories are too short, the upper quadrants will be missed in all of them. In the MountainCar case, multiple rollouts are advantageous, potentially due to the fact that this allows the exploration of regions which were previously unexplored as the car got stuck in a local minimum. For more details, we refer to Appendix B.4.

4.6. Limitations

This section is not part of Achterhold and Stueckler (2021).

The main limitation of the proposed approach is the assumption that the environment is fully observable. This design decision was made on purpose. For a partially observable environment, a belief state has to be formed based on a sequence of past observations. If the belief state is in a learned latent space, information on the dynamics of the system might be captured also by the latent state, instead of being fully captured by the global latent variable $\boldsymbol{\beta}$. Potentially, this is not an issue, as it might lead to a representation of $\boldsymbol{\beta}$ which only captures variations which can *not* be inferred from past observations. However, this hypothesis needs to be investigated by further research. Interpreting $\boldsymbol{\beta}$ and its uncertainty, as done in this chapter, is less straightforward if information on the variations of the dynamics are also captured by the latent state, which is why this case is not considered here.

4.7. Conclusion

In this chapter, we propose a learning method for context-dependent probabilistic dynamics models and an information-theoretic calibration approach to adapt the dynamics models to target environments from experience. We apply the framework of Neural Processes (Garnelo et al., 2018b) with a probabilistic context encoder to formulate our latent dynamics model and propose a learning approach that learns meaningful predictions of the uncertainty in the context variable. Our calibration approach uses expected information gain in the latent context variable as optimization criterion for model-predictive control. For evaluation, we construct an illustrative toy environment, on which we can easily distinct informative and non-informative actions for inferring a latent parameter. Experiments on this toy problem reveal the characteristics of the probabilistic encoding for informative and non-informative actions. We also introduce parametrized pendulum and mountain car environments to demonstrate our method on more complex systems. On these systems, our proposed calibration methods yield models with significantly lower prediction errors compared to random calibration. In future work, we plan to extend our method for learning dynamics models of real systems and for model-based control and reinforcement learning. Modelling partially observable environments also is an interesting avenue of future research.

5. Context-Conditional Terrain-Aware Robot Navigation

Declaration of contributions

The contents of this chapter are based on the peer-reviewed conference publication

 'Context-Conditional Navigation with a Learning-Based Terrain- and Robot-Aware Dynamics Model'. In: *Proceedings of the IEEE European Conference on Mobile Robots (ECMR)* (Guttikonda et al., 2023).

The above publication was presented as an **oral** presentation at the *European Conference on Mobile Robots, 2023, Coimbra, Portugal*.

Author contributions are as follows:

	Scientific ideas	Data generation	Analysis & Interpretation	Paper writing
Suresh Guttikonda	20 %	60 %	30 %	20 %
Jan Achterhold	35 %	40 %	40 %	50 %
Haolong Li	5 %	0 %	5 %	5 %
Joschka Boedecker	5 %	0 %	10 %	0 %
Jörg Stückler	35 %	0 %	15 %	25 %

Parts of this project were conducted as a Master's thesis by Suresh Guttikonda, conducted under advisory by Jan Achterhold, Haolong Li and Jörg Stückler at the MPI-IS in Tübingen, and examined by Joschka Boedecker and Abhinav Valada at the University of Freiburg.

Suresh Guttikonda handed in his Master's thesis titled *Learning a Context-Conditional and Terrain-Aware Kino-Dynamics Model for Autonomous Mobile Robots* at the University of Freiburg on September 1st, 2022.

Jörg Stückler proposed extending the context-conditional dynamics model presented in *Explore the Context* (Achterhold and Stueckler, 2021) for the task of terrain-aware navigation with mobile robots which possess varying internal parameters. Suresh Guttikonda, Jörg Stückler and Jan Achterhold developed the idea of performing terrain lookup during planning. Suresh Guttikonda implemented the terrain profile generation algorithm, and performed experimental studies combining a robot model from a physics simulator, the dynamics model proposed by (Achterhold and Stueckler, 2021), and the cross-entropy method planning algorithm. For the publication, Jan Achterhold replaced the physics simulator by a simple unicycle-based dynamics model with Runge Kutta integration, and implemented, executed, and analyzed all presented experiments. All authors regularly discussed the approach. Jan Achterhold, Jörg Stückler, and Suresh Guttikonda wrote the paper.

We provide our implementation at `https://github.com/EmbodiedVision/tradyn`.

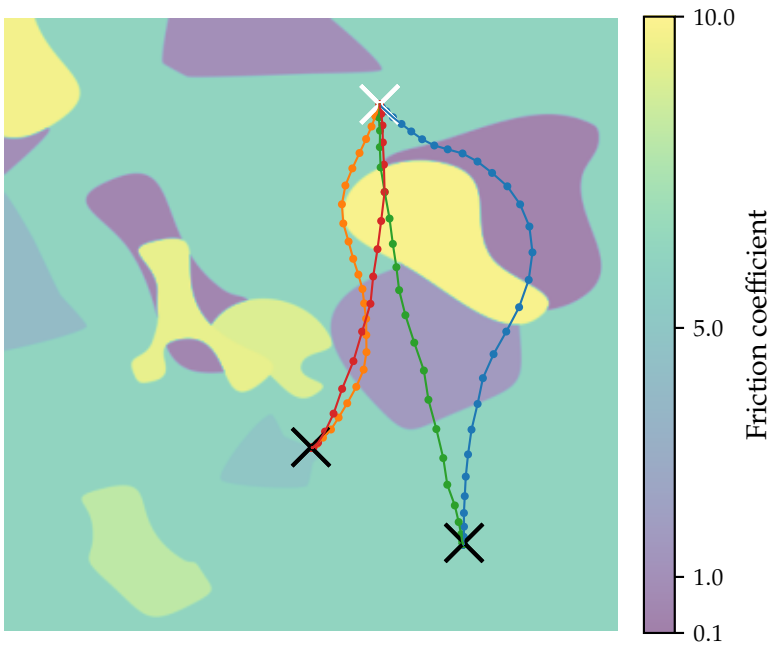


Figure 5.1.: Terrain- and robot-aware control-efficient navigation. We propose a method for control-cost optimal navigation with learned dynamics models. Our method can adapt to varying, unobserved properties of the robot, such as the mass, and spatially varying properties of the terrain, such as the friction coefficient. In the above example of navigating from a single starting point (white cross) to two different goals (black cross), as a result, our method circumvents areas of high friction coefficient and favors areas of low-friction coefficient. As the dissipated energy also depends on the mass of the robot, a heavy robot ($m = 4\,\text{kg}$, blue, orange) is allowed to take longer detours to the goal than a light robot ($m = 1\,\text{kg}$, green, red).

5.1. Introduction

Autonomous mobile robot navigation — the robot's ability to reach a specific goal location — has been an attractive research field over several decades, with applications ranging from self-driving cars, warehouse and service robots, to space robotics. In certain situations, e.g., weeding in agricultural robotics or search and rescue operations, robots operate in harsh and unstructured outdoor environments with limited or no human supervision to complete their task. During such missions, the robot needs to navigate over a wide variety of terrains with changing types, such as grass, gravel, or mud with varying slope, friction, and other characteristics. These properties are often hard to fully and accurately model beforehand (Sonker and Dutta, 2021). Moreover, properties of the robot itself can change during operation due to battery consumption, weight changes, or wear and tear of the robot. Thus, the robot needs to be able to adapt to both changes in robot-specific and terrain-specific properties.

In this work, we develop a novel context-conditional learning approach which captures robot-specific and terrain-specific properties from interaction experience and environment maps. To adapt to robot-specific parameters, we leverage concepts for adaptive dynamics models presented in Chapter 4, in which variations in dynamics are captured by a latent context variable. The context variable is inferred online from observed state transitions. The terrain features are extracted from an environment map and additionally included as conditional variable for the dynamics model.

We develop and evaluate our approach in a 2D simulation of a mobile robot modeled as a point mass with unicycle driving dynamics that depend on a couple of robot-specific and

terrain-specific parameters. Terrains are defined by regions in the map with varying properties. We demonstrate that our context-aware dynamics model learning approach can capture the varying robot and terrain properties well, while a dynamics model without context-awareness achieves less accurate prediction and planning performance.

In summary, in this chapter, we contribute the following:

1. We propose a terrain- and robot-aware probabilistic deep forward dynamics model (TRADYN) which can adapt to robot- and terrain-specific properties that influence the mobile robot's dynamics.
2. We demonstrate in a 2D simulation environment that these adaptation capabilities are crucial for the predictive performance of the dynamics model.
3. The learned context-aware dynamics model is used for robot navigation using model-predictive control. This way, efficient paths can be planned that take robot and terrain properties into account (see Figure 5.1).

5.2. Related work

Some approaches to terrain-aware navigation use semantic segmentation for determining the category of terrain and use this information to only navigate on segments of traversable terrain (Valada et al., 2017; Yang et al., 2018). Z. Zhu et al. (2019) propose to use inverse reinforcement learning to learn the control costs associated with traversing terrain from human expert demonstrations. These methods, however, do not learn the dynamical properties of the robot on the terrain classes explicitly like our methods.

In BADGR (Kahn et al., 2021) a predictive model is learned of future events based on the current RGB image and control actions, which can be used for planning navigation trajectories. The predicted events are collision, bumpiness, and position. The model is trained from sample trajectories in which the events are automatically labelled. Grigorescu et al. (2021) learn a vision-based dynamics model which encodes camera images into a state observation for model-predictive control. Different to our approach, however, the method does not learn a model that can capture a variety of terrain- and robot-specific properties jointly. Siva et al. (2021) learn an offset model from the predicted to the actual behavior of the robot from multimodal terrain features determined from camera, LiDAR, and IMU measurements. In Xiao et al. (2021) a method for learning an inverse kinodynamics model from inertial measurements is proposed to handle high-speed motion planning on unstructured terrain. Sikand et al. (2022) use contrastive learning to embed visual features of terrain with similar traversability properties close in the feature space. The terrain features are used for learning preference-aware path planning. Different to our study, the above approaches do not distinguish terrain- and robot-specific properties and model them concurrently.

Several approaches for learning action-conditional dynamics models have been proposed in the machine learning and robotics literature in recent years. In the seminal work PILCO (Deisenroth and Rasmussen, 2011), Gaussian processes are used to learn to predict subsequent states, conditioned on actions. The approach is demonstrated for balancing and swinging up a cart-pole. Several approaches learn latent embeddings of images and predict future latent states conditioned on actions using recurrent neural networks (Finn et al., 2016; Hafner et al., 2019; Lenz et al., 2015;

Oh et al., 2015). The models are used in several of these works for model-predictive control and planning. Learning-based dynamics models are also popular in model-based reinforcement learning (see e.g. Nagabandi et al. (2018)). Shaj et al. (2020) propose action-conditional recurrent Kalman networks which implement observation and action-conditional state-transition models in a Kalman filter with neural networks. While these approaches can model context from past observations in the latent state of the recurrent neural network, some approaches allow for incorporating an arbitrary set of context observations to infer a context variable (Lee et al., 2020) or a probability distribution thereon (Achterhold and Stueckler, 2021). In this chapter, we base our approach on the context-conditional dynamics model learning approach in Achterhold and Stueckler (2021) to infer the distribution of a context variable of robot-specific parameters using Neural Processes (Garnelo et al., 2018b).

5.3. Preliminaries

This section shortly recapitulates the dynamics model presented in Chapter 4.

We build our approach on the context-conditional probabilistic neural dynamics model of *Explore the Context* (EtC, Achterhold and Stueckler (2021), Chapter 4). In EtC, the basic assumption is that the dynamical system can be formulated by a Markovian discrete-time state-space model

$$\boldsymbol{s}_{n+1} = f(\boldsymbol{s}_n, \boldsymbol{a}_n, \boldsymbol{\alpha}) + \boldsymbol{\epsilon}_n,\ \boldsymbol{\epsilon}_n \sim \mathcal{N}(0, \boldsymbol{Q}_n), \tag{5.1}$$

where $\boldsymbol{s}_n$ is the state at timestep n, $\boldsymbol{a}_n$ is the control input, and $\boldsymbol{\alpha}$ is a latent, *unobserved* variable which modulates the dynamics, e.g., robot or terrain parameters. Gaussian additive noise is modeled by $\boldsymbol{\epsilon}_n$, having a diagonal covariance matrix $\boldsymbol{Q}_n$. Not only $\boldsymbol{\alpha}$ is assumed to be unknown, but also the function f itself. To model the system dynamics, EtC thus introduces an approximate forward dynamics model q_{fwd}. To capture the environment-specific properties $\boldsymbol{\alpha}$, the learned dynamics model is conditioned on a latent context variable $\boldsymbol{\beta} \in \mathbb{R}^{D_\beta}$. A probability density on $\boldsymbol{\beta}$ is inferred from interaction experience on the environment, represented by C transitions $(\boldsymbol{s}_+ \leftarrow \boldsymbol{s}, \boldsymbol{a})$ following Equation 5.1 and collected in a *context* set $C^\alpha = \{(\boldsymbol{s}^{(k)}, \boldsymbol{a}^{(k)}, \boldsymbol{s}_+^{(k)})\}_{k=1}^{C}$. A learned *context encoder* $q_{\text{ctx}}(\boldsymbol{\beta} \mid C^\alpha)$ infers a density on $\boldsymbol{\beta}$. The target chunk $\mathrm{D}^\alpha = (\boldsymbol{s}_n, \boldsymbol{a}_n, \boldsymbol{s}_{n+1}, \ldots, \boldsymbol{a}_{n+T-1}, \boldsymbol{s}_{n+T})$ of length T is a subsequence of a trajectory on the environment. Both context set and target chunk are generated on the same environment instance $\boldsymbol{\alpha}$. For a pair of target chunk and context set, the learning objective is to maximize the marginal log-likelihood

$$\log p_\theta(\mathrm{D}^\alpha \mid C^\alpha) = \log \int p_\theta(\mathrm{D}^\alpha \mid \boldsymbol{\beta})\, p_\theta(\boldsymbol{\beta} \mid C^\alpha)\, d\boldsymbol{\beta}. \tag{5.2}$$

with respect to the parameters $\boldsymbol{\theta}$. Overall, we aim to maximize $\log p_\theta(\mathrm{D}^\alpha \mid C^\alpha)$ in expectation over the distribution of environments Ω_α, context sizes C, and a distribution of pairs of context sets and target chunks $\Omega_{f^\alpha}(C, T)$, i.e.

$$\max \mathbb{E}_{\substack{\boldsymbol{\alpha}\in\Omega_\alpha, C\in\Omega_C, \\ \mathrm{D}^\alpha, C^\alpha \sim \Omega_{f^\alpha}(C,T)}} \log p_\theta(\mathrm{D}^\alpha \mid C^\alpha). \tag{5.3}$$

The term $p_\theta(\mathrm{D}^\alpha|\boldsymbol{\beta})$ is modeled by single-step and multi-step prediction factors and reconstruction factors, all implemented by the approximate dynamics model $q_{\text{fwd}}\left(\boldsymbol{s}_n \mid \boldsymbol{s}_0, \boldsymbol{a}_{0:n-1}, \boldsymbol{\beta}\right)$, while $p_\theta(\boldsymbol{\beta} \mid C^\alpha)$ is approximated by $q_{\text{ctx}}(\boldsymbol{\beta} \mid C^\alpha)$.

Technically, the forward dynamics model is implemented with gated recurrent units (GRU, Cho et al. (2014)) in a latent space. The initial state $\boldsymbol{s}_0$ is encoded into a hidden state $\boldsymbol{z}_0$. The control input $\boldsymbol{a}$ and context variable $\boldsymbol{\beta}$ are encoded into feature vectors and passed as inputs to the GRU

$$\boldsymbol{z}_0 = e_{\mathrm{s}}(\boldsymbol{s}_0) \tag{5.4}$$

$$\mathbf{z}_{n+1} = \mathrm{GRU}\left(\boldsymbol{z}_n, [e_{\mathrm{a}}(\boldsymbol{a}_n), e_{\beta}(\boldsymbol{\beta})]\right) \tag{5.5}$$

where e_{s}, e_{a}, and e_{β} are neural network encoders. The (predicted) latent state $\mathbf{z}_n$ is decoded into a Gaussian distribution in the state space

$$\hat{\mathbf{s}}_n \sim \mathcal{N}\left(d_{s,\mu}(\mathbf{z}_n), d_{s,\sigma^2}(\mathbf{z}_n)\right) \tag{5.6}$$

using neural networks $d_{s,\mu}$, d_{s,σ^2}.

The context encoder gets as input a set of state-action-state transitions C^{α} with flexible size C. The context encoder is implemented by first encoding each transition in the context set independently using a transition encoder e_{trans}, and, for permutation invariance, aggregating the encodings using a dimension-wise max operation. This yields the aggregated latent variable $\boldsymbol{z}_{\beta}$. Lastly, a Gaussian density over the context variable $\boldsymbol{\beta}$ is predicted from the aggregated encodings

$$q_{\mathrm{ctx}}(\boldsymbol{\beta} \mid C^{\alpha}) = \mathcal{N}\left(\boldsymbol{\beta} \mid d_{\beta,\mu}(\boldsymbol{z}_{\beta}), \mathrm{diag}(d_{\beta,\sigma^2}(\boldsymbol{z}_{\beta}))\right) \tag{5.7}$$

with neural network decoders $d_{\beta,\mu}$, d_{β,σ^2}. The network d_{β,σ^2} is designed so that the predicted variance is strictly positive and decreases monotonically when adding context observations.

To form a tractable loss, the marginal log likelihood in Equation 5.2 is (approximately, see Le et al. (2018)) bounded using the evidence lower bound

$$\begin{aligned} &\log p_{\theta}(\mathrm{D}^{\alpha} \mid C^{\alpha}) \gtrapprox \\ &\quad \mathbb{E}_{\boldsymbol{\beta} \sim q_{\mathrm{ctx}}(\boldsymbol{\beta} \mid \mathrm{D}^{\alpha} \cup C^{\alpha})}\left[\log p_{\theta}(\mathrm{D}^{\alpha} \mid \boldsymbol{\beta})\right] - \lambda_{KL}\, \mathrm{KL}\left(q_{\mathrm{ctx}}(\boldsymbol{\beta} \mid \mathrm{D}^{\alpha} \cup C^{\alpha}) \,\|\, q_{\mathrm{ctx}}(\boldsymbol{\beta} \mid C^{\alpha})\right). \end{aligned} \tag{5.8}$$

similar to Neural Processes (Garnelo et al., 2018b). For training the dynamics model and context encoder, the approximate bound in Equation 5.8 is maximized by stochastic gradient ascent on empirical samples for target chunks and context sets. Samples are drawn from trajectories generated on a training set of environments.

By collecting context observations at test time, and inferring $\boldsymbol{\beta}$ using $q_{\mathrm{ctx}}(\boldsymbol{\beta} \mid C^{\alpha})$, the dynamics model $q_{\mathrm{fwd}}(\boldsymbol{s}_n \mid \boldsymbol{s}_0, \boldsymbol{a}_{0:N-1}, \boldsymbol{\beta})$ can adapt to a particular environment instance $\boldsymbol{\alpha}$ (called *calibration*).

5.4. Method

In the modeling assumption of EtC, changes in the dynamics among different instances of environments are captured in a global latent variable $\boldsymbol{\alpha}$ (see Equation 5.1) which is unobserved. In terrain-aware robot navigation, among different environments, the terrain varies (with the terrain layout captured by $\alpha_{\mathrm{terrain}}$), in addition to robot-specific parameters such as actuator gains (captured by $\boldsymbol{\alpha}_{\mathrm{robot}}$). In principle, both effects can be absorbed into a single latent variable

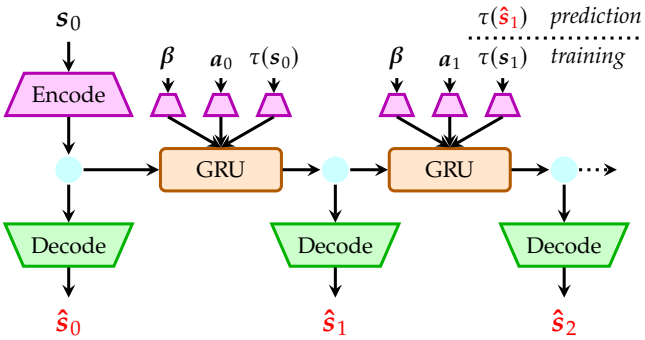


Figure 5.2.: Architecture of our proposed terrain- and robot-aware forward dynamics model (TRADYN). The initial state of the robot s_0 is embedded as hidden state of a gated recurrent unit (GRU) cell. The GRU makes a single-step forward prediction in the latent space using embeddings of the context variable $\boldsymbol{\beta}$, action $\boldsymbol{a}$ and terrain observation $\boldsymbol{\tau}$ as additional inputs. Latent states are mapped to Gaussian distributions on the robot's observation space for decoding. While during training the actual terrain observation $\tau(\boldsymbol{s}_n)$ is used, during prediction, the map τ is queried at predicted robot locations $\tau(\hat{\boldsymbol{s}}_n)$. See Section 5.4 for details.

$\boldsymbol{\alpha} = (\alpha_{\text{terrain}}, \boldsymbol{\alpha}_{\text{robot}})$. Here, we make more specific assumptions, and assume the terrain-specific properties to be captured in a state-dependent function $\alpha_{\text{terrain}}(\boldsymbol{s}_n)$.

5.4.1. Terrain- and Robot-Aware Dynamics Model

Conclusively, we assume the following environment dynamics

$$\boldsymbol{s}_{n+1} = f(\boldsymbol{s}_n, \boldsymbol{a}_n, \boldsymbol{\alpha}_{\text{robot}}, \alpha_{\text{terrain}}(\boldsymbol{s}_n)) + \boldsymbol{\epsilon}_n \tag{5.9}$$

with $\boldsymbol{\epsilon}_n \sim \mathcal{N}(0, \boldsymbol{Q}_n)$ as in Equation 5.1. In our case of terrain-aware robot navigation, $\boldsymbol{s}_n$ refers to the robot state at timestep n, $\boldsymbol{a}_n$ are the control inputs, $\boldsymbol{\alpha}_{\text{robot}}$ captures (unobserved) properties of the robot (mass, actuator gains), and $\alpha_{\text{terrain}}(\boldsymbol{s}_n)$ captures the spatially dependent terrain properties (e.g., friction). While we assume α_{terrain} to be unobserved, we assume the existence of a *known map* of *terrain features* $\tau_{\text{terrain}}(\boldsymbol{s}_n)$, which can be queried at any $\boldsymbol{s}_n$ to estimate the value of $\alpha_{\text{terrain}}(\boldsymbol{s}_n)$. Exemplarily, τ_{terrain} may yield visual terrain observations, which relate to friction coefficients.

As we retain the assumption of EtC that $\boldsymbol{\alpha}_{\text{robot}}$ is not directly observable, we condition the multi-step forward dynamics model on the latent variable $\boldsymbol{\beta}$. In addition, we condition on observed terrain features $\boldsymbol{\tau}_{0:n-1}$, i.e.,

$$\hat{\boldsymbol{s}}_n \sim q_{\text{fwd}}(\boldsymbol{s}_n \mid \boldsymbol{s}_0, \boldsymbol{a}_{0:n-1}, \boldsymbol{\beta}, \boldsymbol{\tau}_{0:n-1}). \tag{5.10}$$

We obtain $\boldsymbol{\tau}_{0:n-1}$ differently for training and prediction. During training, we evaluate τ at ground-truth states, i.e. $\boldsymbol{\tau}_0 = \tau(\boldsymbol{s}_0), \boldsymbol{\tau}_1 = \tau(\boldsymbol{s}_1)$, etc. During prediction, we do not have access to ground-truth states, and obtain $\boldsymbol{\tau}_{0:n-1}$ auto-regressively from predictions as $\boldsymbol{\tau}_0 = \tau(\boldsymbol{s}_0), \boldsymbol{\tau}_1 = \tau(\hat{\boldsymbol{s}}_1)$, etc.

To capture terrain-specific properties, we extend EtC as follows. We introduce an additional encoder e_τ which encodes a terrain feature $\boldsymbol{\tau}$. The encoded value is passed as input to the GRU, such that Equation 5.5 is updated to

$$\boldsymbol{z}_{n+1} = \text{GRU}\left(\boldsymbol{z}_0, [e_\tau(\boldsymbol{\tau}_n), e_a(\boldsymbol{a}_n), e_\beta(\boldsymbol{\beta})]\right). \tag{5.11}$$

Also, the context set is extended to contain terrain features

$$C^{\alpha} = \{(\boldsymbol{s}^{(k)}, \tau(\boldsymbol{s}^{(k)}), \boldsymbol{a}^{(k)}, \boldsymbol{s}_{+}^{(k)}, \tau(\boldsymbol{s}_{+}^{(k)}))\}_{k=1}^{C}. \tag{5.12}$$

We refer to Figure 5.2 for a depiction of our model.

For each training example, the context set size C is uniformly sampled in $\{0, \dots, 50\}$. The target chunk length is $N = 50$. As in EtC (Achterhold and Stueckler, 2021), we set $\lambda_{KL} = 5$. The dimensionality of the latent variable $\boldsymbol{\beta}$ is 16. For details on the networks $(e_{\text{a}}, e_{\beta}, d_{s,\mu}, d_{s,\sigma^2}, e_{\text{trans}}, d_{\beta,\mu}, d_{\beta,\sigma^2})$ we refer to Chapter 4 and Appendix B, as we strictly follow the architecture described therein. The additional encoder network we introduce, e_{β}, follows the architecture of e_{τ} and e_{a}. It contains a single hidden layer with 200 units and ReLU activations, and an output layer which maps to an embedding of dimensionality 200.

5.4.2. Path Planning and Motion Control

We use TRADYN in a model-predictive control setup. The model q_{fwd} yields state predictions $\hat{\boldsymbol{s}}_{1:H}$ for an initial state $\boldsymbol{x_0}$ and controls $\boldsymbol{a}_{0:H-1}$. For calibration, i.e., inferring $\boldsymbol{\beta}$ from a context set C with the context encoder q_{ctx}, calibration transitions are collected on the target environment prior to planning. This allows adapting to varying robot parameters. The predictive terrain feature lookup (see Figure 5.2) with $\tau(\boldsymbol{s})$ allows adapting to varying terrains. We use the cross-entropy method (CEM, Rubinstein (1999), see Section 2.6) for planning. We aim to reach the target position with minimal throttle control energy, given by the sum of squared throttle commands during navigation. This gives rise to the following planning objective, which penalizes high throttle control energy and a deviation of the robot's terminal position to the target position $\boldsymbol{p}^*$:

$$J(\boldsymbol{a}_{0:H-1}, \hat{\boldsymbol{s}}_{1:H}) = \frac{1}{2} \sum_{n=0}^{H-1} u_{\text{throttle},n}^2 + ||[\hat{p}_{\text{x},H}, \hat{p}_{\text{y},H}]^{\top} - \boldsymbol{p}^*||_2^2. \tag{5.13}$$

In our CEM implementation, we normalize the distance term in Equation 5.13 to have zero mean and unit variance over all CEM candidates, to trade-off control- and distance cost terms even under large terrain variations. At each step, we only apply the first action and plan again from the resulting state in a receding horizon scheme.

5.5. Experiments

5.5.1. Simulation environment

Simulated Robot Dynamics

We perform experiments in a 2D simulation with a unicycle-like robot setup where the continuous time-variant 2D dynamics with position $\mathbf{p} = [p_{\text{x}}, p_{\text{y}}]^{\top}$, orientation φ', and directional velocity v, for control input $\boldsymbol{a} = [a_{\text{throttle}}, a_{\text{steer}}]^{\top} \in [-1, 1]^2$, are given by

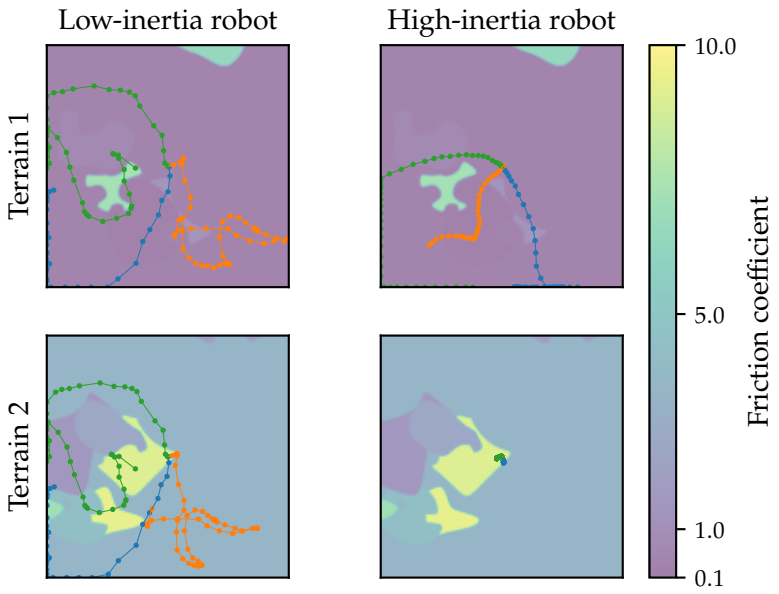


Figure 5.3.: Exemplary rollouts (length 50) on two different terrain layouts (rows) and for two exemplary robot configurations (low-inertia, high-inertia) (columns). Rollouts start from the center; actions are sampled time-correlated. The low-inertia robot has minimal mass $m = 1$ and maximal control gains $k_{\text{throttle}} = 1000$, $k_{\text{steer}} = \pi/4$. The high-inertia robot has maximal mass $m = 4$ and minimal control gains $k_{\text{throttle}} = 500$, $k_{\text{steer}} = \pi/8$. Equally colored trajectories (●, ●, ●) correspond to identical sequences of applied actions. See Subsection 5.5.1 for details.

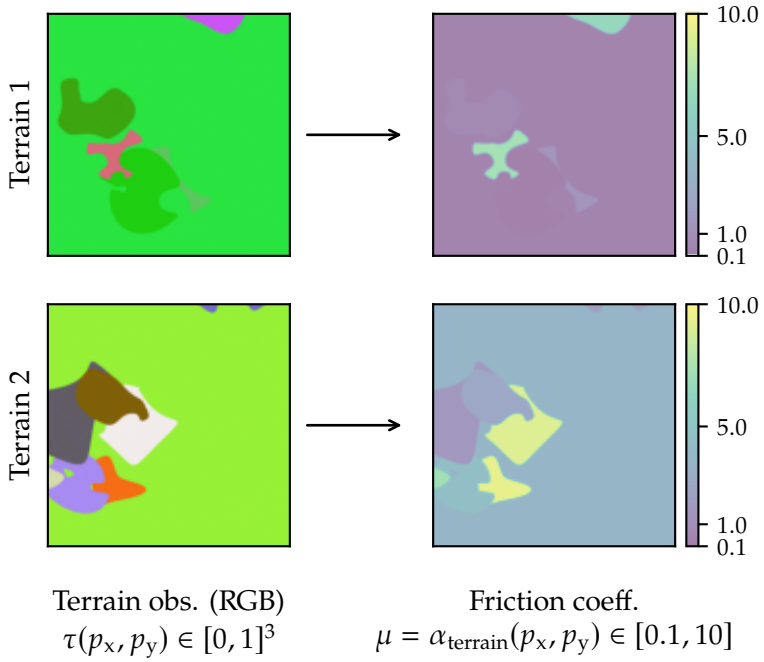


Figure 5.4.: Relationship of RGB terrain features τ (left column) to friction coefficient μ (right column). See Subsection 5.5.1 for details.

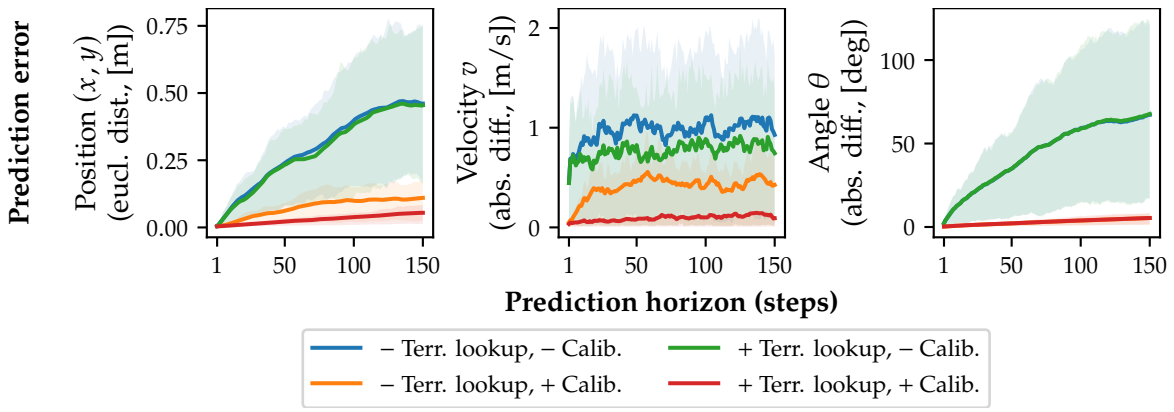


Figure 5.5.: Prediction error evaluation for the proposed model and its ablations (no terrain lookup / no calibration), plotted over the prediction horizon (number of prediction steps). From left to right: Positional error (euclidean distance), velocity error* (absolute difference), angular error (absolute difference). Depicted are the mean and 20%, 80% percentiles over 150 evaluation rollouts for 5 independently trained models per model variant. Our approach with terrain lookup and calibration clearly outperforms the other variants in position and velocity prediction (left and center panel). For predicting the angle (right panel), terrain friction is not relevant, which is why the terrain lookup brings no advantage. However, calibration is important for accurate angle prediction. See Subsection 5.5.3 for details.

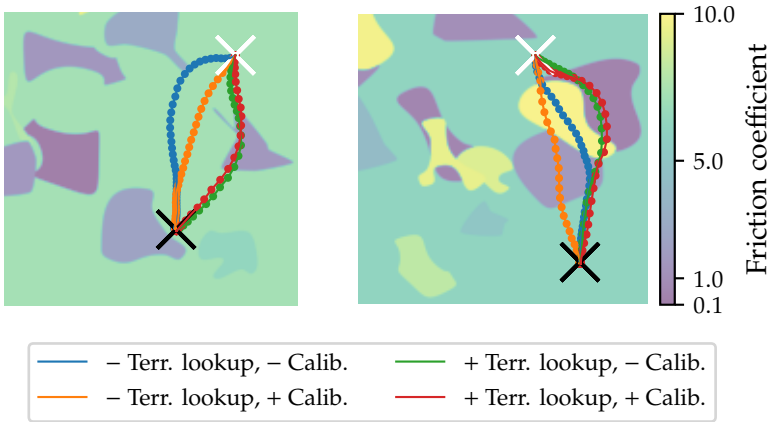
$$\begin{aligned} \dot{\boldsymbol{p}}(t) &= \begin{bmatrix} \cos\varphi' & \sin\varphi' \end{bmatrix}^\top v(t) \\ \dot{v}(t) &= \frac{1}{m}(F_{\text{throttle}} + F_{\text{fric}}) \\ F_{\text{throttle}} &= a_{\text{throttle}}\, k_{\text{throttle}} \\ F_{\text{fric}} &= -\operatorname{sign}(v(t))\, \mu\, m\, g. \end{aligned} \tag{5.14}$$

As our method does not use continuous-time observations, but only discrete-time samplings with stepsize $\Delta_T = 0.01\,\text{s}$, we approximate the state evolution between two timesteps as follows. First, we apply the change in angle as $\varphi' = \varphi(t + \Delta_T) = \varphi(t) + a_{\text{steer}}\, k_{\text{steer}}$. We then query the terrain friction coefficient μ at the position $\boldsymbol{p}(t)$. With the friction coefficient μ and angle φ' we compute the evolution of position and velocity with Equation 5.14. The existence of the friction term in Equation 5.14 requires an accurate integration, which is why we solve the initial value problem in Equation 5.14 numerically using an explicit Runge Kutta (RK45) method, yielding $\boldsymbol{p}(t + \Delta_T)$ and $v(t + \Delta_T)$. Our simulated system dynamics are deterministic. To avoid discontinuities, we represent observations* of the above system as $\boldsymbol{s}(t) = [p_\text{x}(t), p_\text{y}(t), v(t)/5, \cos\varphi(t), \sin\varphi(t)]^\top$. We use $g = 9.81\,\text{m}\,\text{s}^{-2}$ as gravitational acceleration. Positions $\boldsymbol{p}(t)$ are clipped to the range $[0, 1]\,\text{m}$; the directional velocity $v(t)$ is clipped to $[-5, 5]\,\text{m}\,\text{s}^{-1}$. The friction coefficient $\mu = \alpha_{\text{terrain}}(p_\text{x}, p_\text{y}) \in [0.1, 10]$ depends on the terrain layout α_{terrain} and the robot's position. The mass m and control gains $k_{\text{throttle}}, k_{\text{steer}}$ are robot-specific properties, we refer to Table 5.1 for their value ranges.

Terrain layouts

To simulate the influence of varying terrain properties on the robots' dynamics, we programmatically generate 50 terrain layouts for training the dynamics model and 50 terrain layouts for testing (i.e., in prediction- and planning evaluation). For generating terrain k, we first generate an

* As a correction to Guttikonda et al. (2023), we clarify that the velocity observation is normalized, i.e., $\boldsymbol{s}(t) = [\ldots, v(t)/5, \ldots]^\top$. Guttikonda et al. (2023) report the velocity error on the normalized velocity ($v/5$). In Figure 5.5 (center panel), for clarity, the error on the non-normalized velocity v is shown.



Variant	left terrain		right terrain	
	Thr. ctrl. energy	Target dist [mm]	Thr. ctrl. energy	Target dist [mm]
● −T, −C	4.08	7.82	3.96	2.69
● −T, +C	3.47	4.89	2.78	4.87
● +T, −C	2.01	4.57	1.88	5.14
● +T, +C	2.02	5.55	1.43	6.66

Figure 5.6.: Exemplary navigation trajectories and their associated throttle control energy and final distance to the target (see table). The robot starts at the white cross, the goal is marked by a black cross. With terrain lookup (● +T, -C and ● +T, +C), our method circumvents areas of high friction coefficient (i.e., high energy dissipation), resulting in lower throttle control energy (see table). Enabling calibration (+C) further reduces throttle control energy on the right terrain. See Subsection 5.5.4 for details.

Terrain lookup		no		yes	
	Calibration	no	yes	no	yes
no	no		-0.036 **0.115** 0.514	0.024 **0.166** 0.559	0.089 **0.235** 0.912
	yes	-0.514 **-0.115** 0.036		-0.096 **0.042** 0.302	0.007 **0.112** 0.543
yes	no	-0.559 **-0.166** -0.024	-0.302 **-0.042** 0.096		-0.021 **0.034** 0.299
	yes	-0.912 **-0.235** -0.089	-0.543 **-0.112** -0.007	-0.299 **-0.034** 0.021	

Figure 5.7.: Comparison of model variants with and without terrain lookup and calibration†. $E^{v}_{k,i}$ denotes the throttle control energy for method v on navigation task $k \in \{1, \dots, 150\}$ for a trained model with seed $i \in \{1, \dots, 5\}$. We show statistics (20% percentile, median, 80% percentile) on the set of pairwise comparisons of control energies $\{E^{\text{row}}_{k,i_1} - E^{\text{col}}_{k,i_2} \mid \forall k \in \{1, \dots, 150\}, i_1 \in \{1, \dots, 5\}, i_2 \in \{1, \dots, 5\}\}$. Significant ($p < 0.05$) results are printed **bold** (see Subsection 5.5.4). Exemplarily, both performing terrain lookup and calibration (last row) yields navigation solutions with significantly lower throttle control energy (negative numbers) compared to all other methods (columns). See Subsection 5.5.4 for details.

† The code to compute the numbers in Figure 5.7 as reported by Guttikonda et al. (2023) contained an implementation error, which caused only models of equal seeds (i.e., $i_1 = i_2$ in the caption of Figure 5.7) to be compared, but not across different seeds. Here, corrected numbers are reported, which slightly differ from those by Guttikonda et al. (2023) (max. abs. difference in median ctrl. energy: 0.004). The significance statements and variant order w.r.t. median control energy (+T,+C < +T,−C < −T,+C < −T,−C) remain unchanged. The repository at `https://github.com/EmbodiedVision/tradyn` contains a separate branch, fixing this error.

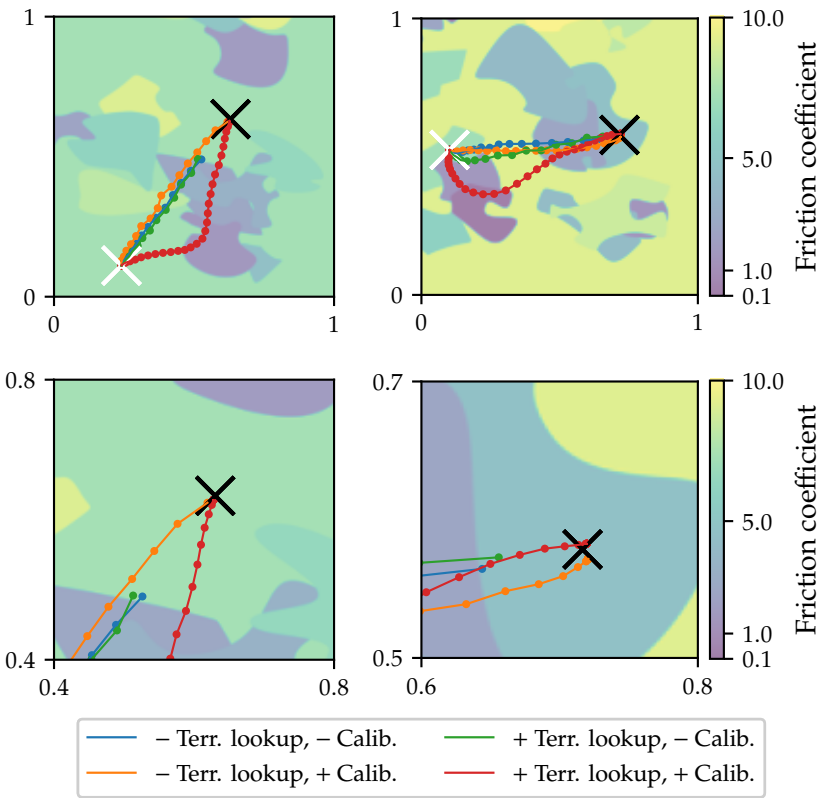


Figure 5.8.: Failure cases for the non-calibrated models. The top row shows the full terrain of extent $[0, 1]$m. The bottom row is zoomed around the goal. In the cases shown, planning with the non-calibrated models does not succeed in reaching the goal marked by the black cross within the given step limit of 50 steps, in contrast to the calibrated models. See Subsection 5.5.4 for details.

unnormalized feature map $\hat{\tau}^{(k)}$, from which we compute $\alpha^{(k)}_{\text{terrain}}$ and the normalized feature map $\tau^{(k)}$. The unnormalized feature map is represented by a 2D RGB image of size 460 px × 460 px. For its generation, first, a background color is randomly sampled, followed by sequentially placing randomly sampled patches with cubic bezier contours. The color value $(r, g, b) \in \{0, \dots, 255\}^3$ at each pixel maps to the friction coefficient $\mu = \alpha_{\text{terrain}}(p_x, p_y)$ through bitwise left-shifts $\ll$ as

$$\begin{aligned} \eta &= ((r \ll 16) + (g \ll 8) + b)/(2^{24} - 1) \\ \alpha_{\text{terrain}}(p_x, p_y) &= 0.1 + (10 - 0.1)\eta^2. \end{aligned} \tag{5.15}$$

The agent can observe the normalized terrain color $\tau^{(k)}(p_x, p_y) \in [0, 1]^3$ with $\tau^{(k)}(p_x, p_y) = \hat{\tau}^{(k)}(p_x, p_y)/255$, and can query $\tau^{(k)}(p_x, p_y)$ at arbitrary p_x, p_y. The simulator has direct access to $\mu = \alpha^{(k)}_{\text{terrain}}(p_x, p_y)$. We denote the training set of terrains $\mathcal{A}_{\text{train}} = \{\alpha^{(k)}_{\text{terrain}} \mid k \in \{1, \dots, 50\}\}$ and the test set of terrains $\mathcal{A}_{\text{test}} = \{\alpha^{(k)}_{\text{terrain}} \mid k \in \{51, \dots, 100\}\}$. We refer to Figure 5.4 for a visualization of two terrains and the related friction coefficients.

Environment instance

The robot's dynamics depends on the terrain α_{terrain} as it is the position-dependent friction coefficient, and the robot-specific parameters $\boldsymbol{\alpha}_{\text{robot}} = (m, k_{\text{throttle}}, k_{\text{steer}})$. A fixed tuple $(\alpha_{\text{terrain}}, \boldsymbol{\alpha}_{\text{robot}})$ forms an environment *instance*.

Table 5.1.: Robot-specific properties.

Property	Min.	Max.
Mass m [kg]	1	4
Throttle gain k_{throttle}	500	1000
Steer gain k_{steer}	$\pi/8$	$\pi/4$

Trajectory generation

We require the generation of trajectories at multiple places of our algorithm for training and evaluation: To generate training data, to sample candidate trajectories for the cross-entropy planning method, to generate calibration trajectories, and to generate trajectories for evaluating the prediction performance. One option would be to generate trajectories by independently sampling actions from a Gaussian distribution at each timestep. However, this Brownian random walk significantly limits the space traversed by such trajectories (Pinneri et al., 2020). To increase the traversed space, Pinneri et al. (2020) propose to use time-correlated (colored) noise with a power spectral density $\text{PSD}(f) \propto \frac{1}{f^{\omega}}$, where f is the frequency. We use $\omega = 0.5$ in all our experiments.

Exemplary rollouts

We visualize exemplary rollouts on different terrains and with different robot parametrizations in Figure 5.3. We observe that both the terrain-dependent friction coefficient μ, as well as the robot properties, have a significant influence on the shape of the trajectories, highlighting the importance of a model to be able to adapt to these properties.

5.5.2. Model training

We train our proposed model on a set of precollected trajectories on different terrain layouts and robot parametrizations. First, we sample a set of 10000 unique terrain layout / robot parameter settings to generate training trajectories. For validation, a set of additional 5000 settings is used. On each setting, we generate two trajectories, used later during training to form the target chunk D^{α} and context set C^{α}, respectively. Terrain layouts are sampled uniformly from the training set of terrains, i.e. $\mathcal{A}_{\text{train}}$. Robot parameters are sampled uniformly from the parameter ranges given in Table 5.1. The robots' initial state $\boldsymbol{x_0} = [p_{\text{x},0}, p_{\text{y},0}, v_0, \varphi_0]^{\top}$ is uniformly sampled from the ranges $p_{\text{x},0}, p_{\text{y},0} \in [0, 1]$, $v_0 \in [-5, 5]$, $\varphi_0 \in [0, 2\pi]$. Each trajectory consists of 100 applied actions and the resulting states. We use time-correlated (colored) noise to sample actions (see previous paragraph). We follow the training procedure described in Achterhold and Stueckler (2021), except we evaluate the models after 100k training steps.

Model ablation

As an ablation to our model, we only input the terrain features $\boldsymbol{\tau}$ at the current and previously visited states of the robot as terrain observations, but do not allow for terrain lookups in a map at future states during prediction. We will refer to this ablation as *No(−) terrain lookup* in the following.

5.5.3. Prediction evaluation

In this section we evaluate the prediction performance of our proposed model. To this end, we generate 150 test trajectories of length 150, on the *test* set of terrain layouts $\mathcal{A}_{\text{test}}$. Robot parameters are uniformly sampled as during data collection for model training. The robot's initial position is sampled from $[0.1, 0.9]^2$, the orientation from $[0, 2\pi]$. The initial velocity is fixed to 0. Actions are sampled with a time-correlated (colored) noise scheme. In case the model is *calibrated*, we additionally collect a small trajectory for each trajectory to be predicted, consisting of 10 transitions, starting from the same initial state $\boldsymbol{x_0}$, but with different random actions. Transitions from this trajectory form the context set C, which is used by the context encoder $q_{\text{ctx}}(\boldsymbol{\beta} \mid C)$ to output a belief on the latent context variable $\boldsymbol{\beta}$. In case the model is *not calibrated*, the distribution is given by the context encoder for an empty context set, i.e. $q_{\text{ctx}}(\boldsymbol{\beta} \mid C = \{\})$. We evaluate two model variants; first, our proposed model which utilizes the terrain map $\tau(p_{\text{x}}, p_{\text{y}})$ for lookup during predictions, and second, a model for which the terrain observation is concatenated to the robot observation. All results are reported on 5 independently trained models. Figure 5.5 shows that our approach with terrain lookup and calibration clearly outperforms the other variants in position and velocity prediction. As the evolution of the robot's angle is independent of terrain friction, for angle prediction, only performing calibration is important.

5.5.4. Planning evaluation

Aside the prediction capabilities of our proposed method, we are interested whether it can be leveraged for efficient navigation planning. To evaluate the planning performance, we generate 150 navigation tasks, similar to the above prediction tasks, but with an additional randomly sampled target position $\boldsymbol{p}^* \in [0.1, 0.9]^2$ for the robot. We perform receding horizon control as described in Subsection 5.4.2.

Again, we evaluate four variants of our model. We compare models with and without the ability to perform terrain lookups. Additionally, we evaluate the influence of calibration, by either collecting 10 additional calibration transitions for each planning task setup, or not collecting any calibration transition ($C = \{\}$), giving four variants in total.

As we have trained five models with different seeds, over all models, we obtain 750 navigation results. We count a navigation task as *failed* if the final Euclidean distance to the goal exceeds 5 cm.

We evaluate the efficiency of the navigation task solution by the sum of squared throttle controls over a fixed trajectory length of $N = 50$ steps, which we denote as $E = \sum_{n=0}^{N-1} u_{\text{throttle},n}^2$. We introduce super- and subscripts $E_{k,i}^{v}$ to refer to model variant v, planning task index k and model seed i. Please see Figures 5.6 and 5.7 for results comparing the particular variants. For pairwise comparison of control energies E we leverage the Wilcoxon signed-rank test with a p-value of 0.05. We can conclude that, regardless of calibration, performing terrain lookups yields navigation solutions with significantly lower throttle control energy. The same holds for performing calibration, regardless of performing terrain lookups. Lowest control energy is obtained for both performing calibration and terrain lookup.

We refer to Table 5.2 for statistics on the number of failed tasks and final distance to the goal. As can be seen, our terrain- and robot-aware approach yields overall best performance in Euclidean distance to the goal and succeeds in all runs in reaching the goal. Planning with non-calibrated

Table 5.2.: Distance to goal (median and 20% / 80% percentiles) and failure rate for 5 cm distance threshold to goal. Variants are with/without terrain lookup (±T) and with/without calibration (±C). Our full approach (+T, +C) yields best performance in reaching the goal and succeeds in all runs.

Variant	Euclidean distance to goal [mm]			Failed tasks
	P20	median	P80	
● −T, −C	3.00	5.19	8.67	6/750
● −T, +C	3.13	5.23	7.65	**0/750**
● +T, −C	2.33	4.22	6.49	14/750
● +T, +C	**2.16**	**3.85**	**5.61**	**0/750**

models variants occasionally fails, i.e., the goal is not reached. We show such failure cases in Figure 5.8.

5.6. Limitations

This section is not part of Guttikonda et al. (2023).

Main limitations of this work are the assumption that the robot's state is fully observable (as in Achterhold and Stueckler (2021), Chapter 4), and that the environment's terrain map is fully available a-priori. Thus, an interesting avenue of future work is to consider partially observable environments, which requires forming a belief state from past observations, e.g. through a recurrent encoder as in Hafner et al. (2019). However, this might lead to information on the dynamics of the system to be captured by a latent belief state instead of the latent variable $\boldsymbol{\beta}$, as discussed in Section 4.6. Furthermore, the assumption of an a-priori available environment map might be alleviated by learning a map from visual observations, commonly referred to as *visual simultaneous localization and mapping (visual SLAM)* (e.g., surveyed by Macario Barros et al. (2022)). Combining the active learning perspective from Achterhold and Stueckler (2021) (Chapter 4) for identifying dynamics models with visual SLAM might allow building a (visual) *active SLAM* (e.g., surveyed by Placed et al. (2023)) method.

5.7. Conclusion

In this chapter, we propose a forward dynamics model which can adapt to variations in unobserved variables that govern the system's dynamics such as robot-specific properties, as well as to spatial variations. We train our model on a simulated unicycle-like robot, which has varying mass and actuator gains. In addition, the robot's dynamics are influenced by instance-wise and spatially varying friction coefficients of the terrain, which are only indirectly observable through terrain observations. In 2D simulation experiments, we demonstrate that our model can successfully cope with such variations through calibration and terrain lookup. It exhibits smaller prediction errors compared to model variants without calibration and terrain lookup, and yields solutions to navigation tasks which require lower throttle control energy. In future work, we plan to extend our novel learning-based approach for real-world robot navigation problems with partial observability, noisy state transitions, and noisy observations.

Case study: Gray-box vs. Black-box for Table Tennis Ball Trajectory Modeling

6.

Declaration of contributions

The contents of this chapter are based on the publication

J. Achterhold, P. Tobuschat, H. Ma, D. Büchler, M. Muehlebach, and J. Stueckler (2023). 'Black-Box vs. Gray-Box: A Case Study on Learning Table Tennis Ball Trajectory Prediction with Spin and Impacts'. In: *Proceedings of the Learning for Dynamics and Control Conference (L4DC)* (Achterhold et al., 2023).

The above publication was presented as a poster presentation at the *Learning for Dynamics and Control Conference (L4DC) 2023, Philadelphia, PA, USA*.

Author contributions are as follows:

	Scientific ideas	Data generation	Analysis & Interpretation	Paper writing
Jan Achterhold	45 %	70 %	50 %	60 %
Philip Tobuschat	15 %	20 %	5 %	0 %
Hao Ma	15 %	10 %	5 %	10 %
Dieter Büchler	5 %	0 %	10 %	0 %
Michael Mühlebach	10 %	0 %	15 %	15 %
Jörg Stückler	10 %	0 %	15 %	15 %

This project was set out by Michael Mühlebach and Jörg Stückler to track and predict table tennis ball motion including spin and impacts. Jan Achterhold proposed and implemented the idea of learning the dynamics and observation model parameters through approximate likelihood (computed by an extended Kalman filter) maximization. The dynamics model (ODE) had already been, for a prior approach, formulated by Hao Ma. Philip Tobuschat was responsible for collecting the offline trajectory data, which was used for training and evaluation. Philip Tobuschat and Hao Ma helped with executing the robotic return experiments. Jan Achterhold implemented the algorithm and performed the simulation experiments on the proposed algorithm and the baselines. All authors regularly discussed the approach. Jan Achterhold, Hao Ma, Michael Mühlebach, and Jörg Stückler wrote the paper.

We provide our implementation at `https://github.com/EmbodiedVision/tabletennis-spin-impacts`.

6.1. Introduction

Playing table tennis with a robot is a long-standing challenge in robotics research (Andersson, 1989). A table tennis playing robot can be considered a sequential decision making agent which is subject to several of the challenges presented in Section 1.4. First, as it is a real-world system,

any learning algorithm has to cope with scarcity of data. Second, partial observability and noisy measurements are properties of such systems: Exemplarily, information on the ball's trajectory is only available as noisy position measurements.

As a single measurement is insufficient to estimate a Markovian belief state on which robot actions can be computed, multiple past measurements have to be filtered (Subsection 2.2.2).

Instead of learning a parametric policy yielding robot actions directly from the filtered belief state, in this chapter we choose an approach which separately estimates the ball's state, predicts the future trajectory, and computes robot actions based on a predicted hitting point.

Our particular focus is on estimating the ball's state from position measurements and predicting its future trajectory. An important difficulty of this task are nonlinear effects arising from drag, spin, and impacts with the table. Approaches to this problem can be distinguished by the amount of physical knowledge being incorporated in the estimation- and prediction model's design process: Black-box models, such as high-capacity function approximators like neural networks, only make use of minimal physical knowledge. Gray-box models, on the other hand, more explicitly incorporate structure in form of laws of physics in their design process (see Subsection 1.5.5). We propose a gray-box model which builds on existing knowledge about the physical dynamics of a flying ball, including drag and spin effects. Our method is based on the extended Kalman filter (EKF) and includes various parameters which are trained from offline data. This gray-box approach demonstrates superior prediction performance compared to two deep-learning based (black-box) baselines in our experiments. We demonstrate that our approach allows incorporating and learning black-box components. We show that estimating the ball's initial spin from parameters of the ball launcher with a neural network, which is jointly learned with other parameters of the dynamics, drastically improves prediction performance over an uninformed initialization of the initial spin. Lastly, we perform experiments for returning balls with a pneumatic artificial muscular robot based on trajectory predictions obtained with our proposed approach, and achieve a return rate of 29/30 (97.7 %).

6.2. Related work

6.2.1. Time series models

Models for time series can be categorized into white-box, gray-box, and black-box models. Black-box models follow a purely statistical, data-driven approach without incorporation of prior physical knowledge on the system to model. In contrast, white-box models are purely based on a-priori system knowledge, without incorporation of data. In gray-box models, both physical a-priori knowledge and data are used for model design and the identification of its parameters. For many dynamical systems, it is difficult or impossible to achieve accurate white-box modeling due to unmodelled effects or variable parameters. Incorporation of data enables black- and gray-box models to adapt to the specifics of the system at hand, which is why we will focus on these two model classes in the following. Inferring dynamics models and their parameters from data is classically referred to as *system identification*, see (Ljung, 1986) for an overview.

Black-box models Black-box time series models often leverage a latent space formulation. They comprise an encoder-decoder pair mapping to and from the latent space, and a forward model in the latent space. Recurrent cells such as long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) or gated recurrent units (GRU) (Cho et al., 2014) are commonly used as latent forward models. Such purely deterministic models can be extended by stochastic nodes to handle noisy observations and transitions, as in variational recurrent neural networks (VRNN) (Chung et al., 2015) or stochastic RNNs (SRNN) (Fraccaro et al., 2016). Several latent sequence models have been proposed which allow for state estimation through filtering, such as the Kalman-VAE (Fraccaro et al., 2017), Recurrent Kalman Networks (Becker et al., 2019), and Deep Variational Bayes Filters (Karl et al., 2017). Hafner et al. (2019) present a recurrent (variational) state-space model (RSSM) which they use for model-based reinforcement learning. We use RSSM as a black-box baseline. Girin et al. (2021) present a comprehensive overview of latent sequence models.

Gray-box models In our work, we follow a gray-box approach, yielding lower prediction error than a black-box baseline and allowing for physical interpretation of the estimated quantities. In contrast to the black-box models discussed above, gray-box models incorporate prior knowledge about the system to model, such as the laws of physics. One line of work in this direction are *differentiable physics engines* (Avila Belbute-Peres et al., 2018). These engines enable gradient-based system identification from data for physical systems with a given structure. Our approach combines ideas from system identification with machine learning. It is therefore capable of exploiting prior knowledge from physics, while also learning parameters of the filter, dynamics, and a state initialization neural network from data.

6.2.2. Table tennis ball trajectory modeling

Approaches for table tennis ball trajectory modelling and prediction can also be categorized as white-box, black-box, and gray-box models. A common **white-box** approach is to use an aerodynamic model of the ball respecting gravity, drag, and Magnus forces (Andersson, 1989) and a physics-grounded rebound / impact model (Nakashima et al., 2010) in an extended / unscented Kalman filter (Koç et al., 2018; Mülling et al., 2010; Tebbe et al., 2018; Q. Wang et al., 2014; Y. Zhang et al., 2015; Z. Zhang et al., 2010). Common **black-box** models approximate the ball trajectories with polynomial curves which are fitted to recorded data (H. Li et al., 2012; Lin et al., 2020; Matsushima et al., 2005; Tebbe et al., 2018). We compare our approach to the deep-learning based black-box approach by Gómez-González et al. (2020) which leverages a variational auto-encoder architecture for table tennis ball trajectory prediction. The ball's spin constitutes a particular challenge for table tennis ball trajectory prediction, since it is hard to infer from position measurements of the trajectory. As a result, prior works have resorted to detecting the ball's spin by following the brand logo on the ball (Y. Zhang et al., 2015) or by equipping the racket with an inertial sensor (Blank et al., 2017). In our **gray-box** approach, we use information from the ball's launch process to initialize the spin. More precisely, we learn the parameters of a neural network that relates the ball launcher settings to the initial spin of the ball. This is shown to drastically improve the quality and accuracy of the predicted ball trajectories.

6.3. Preliminaries

This section is not part of Achterhold et al. (2023).

6.3.1. Linear Gaussian state-space models

For particular types of state-space models (see Subsection 2.2.2), analytically tractable filtering and prediction densities exist. One of such models is the linear Gaussian state-space model (LGSSM).

The linear Gaussian state-space model is characterized by the initial state $\boldsymbol{s}_0$ being Gaussian distributed

$$p(\boldsymbol{s}_0) = \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0), \tag{6.1}$$

The mapping between subsequent states is linear, parametrized by the matrices $\boldsymbol{A}_n$ and $\boldsymbol{B}_n$, and perturbed with additive timewise uncorrelated Gaussian *system* or *transition* noise $\boldsymbol{\zeta}_n$. Overall, the subsequent state is given by

$$\boldsymbol{s}_{n+1} = \boldsymbol{A}_n \boldsymbol{s}_n + \boldsymbol{B}_n \boldsymbol{a}_n + \boldsymbol{\zeta}_n, \text{ with } \boldsymbol{\zeta}_n \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{Q}_n). \tag{6.2}$$

The mapping between states and observations is also linear, parametrized by the matrix $\boldsymbol{C}_n$,

$$\boldsymbol{o}_n = \boldsymbol{C}_n \boldsymbol{s}_n + \boldsymbol{\epsilon}_n, \text{ with } \boldsymbol{\epsilon}_n \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{R}_n), \tag{6.3}$$

and perturbed with timewise uncorrelated Gaussian *measurement* or *observation* noise $\boldsymbol{\epsilon}_n$.

6.3.2. Kalman Filter

The Kalman filter allows computing filtering and prediction densities for the Linear Gaussian state-space model given above in analytically closed form, with Gaussian

- **prediction density**

$$p(\boldsymbol{s}_n \mid \boldsymbol{o}_{<n}, \boldsymbol{a}_{0:n-1}) = \mathcal{N}(\boldsymbol{s}_n \mid \boldsymbol{\mu}_{n|<n}, \boldsymbol{\Sigma}_{n|<n}) \tag{6.4}$$

- and **filtering density**

$$p(\boldsymbol{s}_n \mid \boldsymbol{o}_{\leq n}, \boldsymbol{a}_{0:n-1}) = \mathcal{N}(\boldsymbol{s}_n \mid \boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n). \tag{6.5}$$

The prediction density at timestep n is conditioned on observations up to (excluding) timestep n, hence the subscript $\cdot_{n|<n}$ for the mean and covariance matrix. The filtering density at timestep n does include all observation up to and including timestep n.

In the **prediction step**, the Kalman filter computes $\boldsymbol{\mu}_{n|<n}$, $\boldsymbol{\Sigma}_{n|<n}$ given $\boldsymbol{\mu}_{n-1}$, $\boldsymbol{\Sigma}_{n-1}$, which are parameters of a prediction (i.e., $\boldsymbol{\mu}_{n-1} = \boldsymbol{\mu}_{n-1|<n-1}$, $\boldsymbol{\Sigma}_{n-1} = \boldsymbol{\Sigma}_{n-1|<n-1}$), filtering, or the initial state $(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$ density.

$$\begin{aligned} \boldsymbol{\mu}_{n|<n} &= \boldsymbol{A}_{n-1} \boldsymbol{\mu}_{n-1} \\ \boldsymbol{\Sigma}_{n|<n} &= \boldsymbol{A}_{n-1} \boldsymbol{\Sigma}_{n-1} \boldsymbol{A}_{n-1}^\top + \boldsymbol{Q}_{n-1}. \end{aligned} \tag{6.6}$$

In the **update step**, to obtain the filtering density at timestep n, parameters of a prediction density (including observations up to timestep $n-1$, i.e. $\boldsymbol{\mu}_{n|<n}, \boldsymbol{\Sigma}_{n|<n}$), are updated with the observation at timestep n

$$\begin{aligned} \boldsymbol{v}_n &= \boldsymbol{o}_n - \boldsymbol{C}_n \boldsymbol{\mu}_{n|<n} \\ \boldsymbol{S}_n &= \boldsymbol{C}_n \boldsymbol{\Sigma}_{n|<n} \boldsymbol{C}_n^\top + \boldsymbol{R}_n \\ \boldsymbol{K}_n &= \boldsymbol{\Sigma}_{n|<n} \boldsymbol{C}_n^\top \boldsymbol{S}_n^{-1} \\ \boldsymbol{\mu}_n &= \boldsymbol{\mu}_{n|<n} + \boldsymbol{K}_n \boldsymbol{v}_n \\ \boldsymbol{\Sigma}_n &= \boldsymbol{\Sigma}_{n|<n} - \boldsymbol{K}_n \boldsymbol{S}_n \boldsymbol{K}_n^\top, \end{aligned} \tag{6.7}$$

yielding $\boldsymbol{\mu}_n$ and $\boldsymbol{\Sigma}_n$.

In a **recursion scheme**, the initial state density is predicted forward in time with Equation 6.6 until the first observation becomes available, which is then used to obtain a filtering density with Equation 6.7. This filtering density is again predicted forward in time until the next observation arrives, and the recursion continues.

6.3.3. LGSSMs as Gaussian processes

From basic properties of the Gaussian distribution we can follow that any finite-length sequence of states $\boldsymbol{s}_{1:N}$ and measurements $\boldsymbol{o}_{1:N}$ generated by a linear Gaussian state-sspace model (LGSSM) are jointly Gaussian distributed. By the marginalization property of the Gaussian distribution, this also holds for subsets of states and measurements. We can conclude that an LGSSM thereby fulfills the conditions of Definition 3.3.1, and is thus a *Gaussian process*. In Subsection 3.3.1, we have seen that analytical solutions for posterior inference in Gaussian processes exist. However, their computational complexity scales cubically with the number of datapoints N, i.e., $\mathcal{O}(N^3)$. Albeit the presented solution is generally applicable to posterior inference in Gaussian process regression, it does not take into account the particular factorization of the LGSSM, which allows for more efficient computation. The *Kalman filter* (Kalman, 1960) leverages this structure with recursive computations, and scales linearly with the number of measurements $\mathcal{O}(N)$. The relationship between Gaussian process regression and Kalman filters is well known in literature. Exemplarily, Hartikainen and Särkkä (2010) study how to leverage the Kalman filter for performing efficient temporal Gaussian process regression exactly for Matérn kernels, and approximately for squared exponential kernels.

6.3.4. Inference in time series models with intractable filtering densities

In the general case, no tractable prediction and filtering densities exist. In these cases, one has to resort to approximations. The extended Kalman filter (see, e.g., Jazwinski (1970) and Särkkä (2013)) linearizes nonlinear transition- and observation models, and applies the Kalman filter on the linearized models. We will discuss the extended Kalman filter in Subsection 6.3.5 in more detail.

6.3.5. Extended Kalman filter

See Särkkä (2013) for a derivation of the extended Kalman filter prediction and update equations.

The extended Kalman filter allows computing approximate prediction and filtering densities for non-linear transition and observation models. However, in the form presented here, it still assumes additive Gaussian noise and an initial state which is Gaussian distributed. Formally,

$$p(\boldsymbol{s}_0) = \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) \tag{6.8}$$
$$\boldsymbol{s}_{n+1} = g(\boldsymbol{s}_n, \boldsymbol{a}_n) + \boldsymbol{\zeta}_n, \text{ with } \boldsymbol{\zeta}_n \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{Q}_n) \tag{6.9}$$
$$\boldsymbol{o}_n = h(\boldsymbol{s}_n) + \boldsymbol{\epsilon}_n, \text{ with } \boldsymbol{\epsilon}_n \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{R}_n). \tag{6.10}$$

Here, $g : \mathbb{R}^{D_\mathrm{S}} \times \mathbb{R}^{D_\mathrm{A}} \to \mathbb{R}^{D_\mathrm{S}}$ and $h : \mathbb{R}^{D_\mathrm{S}} \to \mathbb{R}^{D_\mathrm{o}}$ are the transition model function and observation model function, respectively. In order to apply the extended Kalman filter, these functions are required to be differentiable.

The extended Kalman filter is based on two central ideas:

- Prediction and filtering densities are approximated by Gaussian densities

$$p(\boldsymbol{s}_n \mid \boldsymbol{o}_{<n}, \boldsymbol{a}_{0:n-1}) \simeq \mathcal{N}(\boldsymbol{s}_n \mid \boldsymbol{\mu}_{n|<n}, \boldsymbol{\Sigma}_{n|<n}) \tag{6.11}$$
$$p(\boldsymbol{s}_n \mid \boldsymbol{o}_{\leq n}, \boldsymbol{a}_{0:n-1}) \simeq \mathcal{N}(\boldsymbol{s}_n \mid \boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n). \tag{6.12}$$

- The non-linear transition and observation model functions $g(\boldsymbol{s}_n, \boldsymbol{a}_n)$ and $h(\boldsymbol{s}_n)$ are *linearized* at the mean of the distribution over $\boldsymbol{s}_n$, which allows to employ Kalman filtering *locally*, i.e., at the particular linearization of g and h.

We denote the transition and observation models linearized at $(\boldsymbol{s} = \boldsymbol{\mu}, \boldsymbol{a})$ as

$$\widetilde{g}_{\boldsymbol{\mu}}(\boldsymbol{s}, \boldsymbol{a}) = g(\boldsymbol{\mu}, \boldsymbol{a}) + \boldsymbol{G}(\boldsymbol{\mu}, \boldsymbol{a}) \cdot (\boldsymbol{s} - \boldsymbol{\mu}), \quad \widetilde{h}_{\boldsymbol{\mu}}(\boldsymbol{s}) = h(\boldsymbol{\mu}) + \boldsymbol{H}(\boldsymbol{\mu}) \cdot (\boldsymbol{s} - \boldsymbol{\mu}) \tag{6.13}$$

with the Jacobians

$$\boldsymbol{G}(\boldsymbol{\mu}, \boldsymbol{a}) = \frac{\partial g(\boldsymbol{s}', \boldsymbol{a})}{\partial \boldsymbol{s}'}|_{\boldsymbol{s}'=\boldsymbol{\mu}}, \quad \boldsymbol{H}(\boldsymbol{\mu}) = \frac{\partial h(\boldsymbol{s}')}{\partial \boldsymbol{s}'}|_{\boldsymbol{s}'=\boldsymbol{\mu}}. \tag{6.14}$$

In analogy to the Kalman filter prediction step in Equation 6.6, the extended Kalman filter **prediction step** is given by

$$\begin{aligned} \boldsymbol{\mu}_{n|<n} &= g(\boldsymbol{\mu}_{n-1}, \boldsymbol{a}_{n-1}) \\ \boldsymbol{\Sigma}_{n|<n} &= \boldsymbol{G}(\boldsymbol{\mu}_{n-1}, \boldsymbol{a}_{n-1}) \boldsymbol{\Sigma}_{n-1} \boldsymbol{G}^\top(\boldsymbol{\mu}_{n-1}, \boldsymbol{a}_{n-1}) + \boldsymbol{Q}_{n-1}. \end{aligned} \tag{6.15}$$

Similar to Equation 6.7, the extended Kalman filter **update step** is given by

$$\begin{aligned} \boldsymbol{v}_n &= \boldsymbol{o}_n - h(\boldsymbol{\mu}_{n|<n}) \\ \boldsymbol{S}_n &= \boldsymbol{H}(\boldsymbol{\mu}_{n|<n}) \boldsymbol{\Sigma}_{n|<n} \boldsymbol{H}^\top(\boldsymbol{\mu}_{n|<n}) + \boldsymbol{R}_n \\ \boldsymbol{K}_n &= \boldsymbol{\Sigma}_{n|<n} \boldsymbol{H}^\top(\boldsymbol{\mu}_{n|<n}) \boldsymbol{S}_n^{-1} \\ \boldsymbol{\mu}_n &= \boldsymbol{\mu}_{n|<n} + \boldsymbol{K}_n \boldsymbol{v}_n \\ \boldsymbol{\Sigma}_n &= \boldsymbol{\Sigma}_{n|<n} - \boldsymbol{K}_n \boldsymbol{S}_n \boldsymbol{K}_n^\top. \end{aligned} \tag{6.16}$$

If $g(\boldsymbol{s}, \boldsymbol{a}) = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{B}\boldsymbol{a}$, $h(\boldsymbol{s}) = \boldsymbol{C}\boldsymbol{s}$, i.e., the state-space model is linear, the extended Kalman filter yields prediction and filtering densities identical to those of the Kalman filter.

6.3.6. Parameter identification with the KF/EKF

As outlined above, filtering a sequence of observations corresponds to a recursive application of predict- and update steps. The densities obtained after the predict step, so-called *one-step prediction densities*, can be leveraged to compute the marginal likelihood (marginalized over the hidden states) of an observation sequence (see Särkkä (2013) for a derivation)

$$p(\boldsymbol{o}_1, \ldots, \boldsymbol{o}_N \mid \boldsymbol{\theta}). \tag{6.17}$$

In the case of an extended Kalman filter, it is an approximation to the actual marginal likelihood, as the filtering- and prediction densities are approximated with Gaussian densities. The likelihood depends on the parameters of the stochastic process $\boldsymbol{\theta}$, which are, e.g., moments of the initial state distribution, transition noise covariance, observation noise covariance, observation model function and transition model function. This allows to *identify* these parameters from a finite set of observations by likelihood maximization (see Subsection 2.4.2). As the one-step prediction densities are a by-product of the filtering process, as filtering itself, computing the marginal likelihood has a computational complexity of order $\mathcal{O}(N)$.

6.4. Method

We present a gray-box method based on the extended Kalman filter (EKF), which includes parameters that are learned from data. In our notation, scalars are denoted by lowercase letters (σ), vectors are denoted by bold lowercase letters ($\boldsymbol{\sigma}$), and matrices are denoted by bold uppercase letters ($\boldsymbol{\Sigma}$). The operator $\operatorname{diag}(\boldsymbol{x})$ forms a square matrix with $\boldsymbol{x}$ on its diagonal. The operator $[x]^{\oplus}$ applies the softplus function and adds a constant: $[x]^{\oplus} = \log(1 + e^x) + 10^{-6}$ (elementwise for vectors).

6.4.1. Physical model

We assume the ball's dynamics in free-flight (not impacting with the table) to follow the ordinary differential equation (ODE)

$$\dot{\boldsymbol{v}}(t) = -k_{\mathrm{d}} \left\|\boldsymbol{v}(t)\right\|_2 \boldsymbol{v}(t) + k_{\mathrm{m}}(\boldsymbol{\omega}(t) \times \boldsymbol{v}(t)) + \mathbf{g} \tag{6.18}$$

with linear velocity $\boldsymbol{v}^\top(t) = (v_{\mathrm{x}}(t), v_{\mathrm{y}}(t), v_{\mathrm{z}}(t))$ and its Euclidean norm $\|\boldsymbol{v}(t)\|_2$, angular velocity (spin) $\boldsymbol{\omega}^\top(t) = (\omega_{\mathrm{x}}(t), \omega_{\mathrm{y}}(t), \omega_{\mathrm{z}}(t))$, drag coefficient k_{d}, Magnus effect coefficient k_{m} and gravitational acceleration $\mathbf{g}^\top = \left(0, 0, -9.802\,\mathrm{m\,s^{-2}}\right)$. We model the table impact by a linear map that relates the pre- and post-impact velocity $\boldsymbol{v}$ and spin $\boldsymbol{\omega}$ as

$$((\boldsymbol{v}^+)^\top, (\boldsymbol{\omega}^+)^\top)^\top = \mathbf{C}((\boldsymbol{v}^-)^\top, (\boldsymbol{\omega}^-)^\top)^\top, \quad \mathbf{C} \in \mathbb{R}^{6\times 6} \tag{6.19}$$

where the superscripts $^+$ ($^-$) indicate the linear/angular velocities directly after (before) table impact.

6.4.2. Discrete-time state-space model

We formulate a discrete-time state-space model for the ball trajectory for filtering and prediction.

Free flight We introduce a state-space model for the ball dynamics with state

$$\boldsymbol{z}(t) = (\boldsymbol{p}^\top(t), \boldsymbol{v}^\top(t), \boldsymbol{\omega}^\top(t), a_\mathrm{d}(t), a_\mathrm{m}(t))^\top \in \mathbb{R}^{11} \tag{6.20}$$

where $\boldsymbol{p} \in \mathbb{R}^3$ is the position of the ball's center in Cartesian coordinates. The variables a_d, a_m parameterize the drag and Magnus coefficients $k_\mathrm{d}(t) = a_\mathrm{d}^2(t) + \epsilon, k_\mathrm{m}(t) = a_\mathrm{m}^2(t) + \epsilon$ to avoid explicit non-negativity constraints and stabilize training. We choose $\epsilon = 0.05$ in our experiments. As in Equation 6.18, $\boldsymbol{v} \in \mathbb{R}^3$ and $\boldsymbol{\omega} \in \mathbb{R}^3$ relate to linear and angular velocity, respectively. To model free-flight phases, we time-discretize the ODE in Equation 6.18 by Euler's method

$$\begin{aligned}
\boldsymbol{p}(t+\Delta_t) &= \boldsymbol{p}(t) + \Delta_t \boldsymbol{v}(t) \\
\boldsymbol{v}(t+\Delta_t) &= \boldsymbol{v}(t) + \Delta_t \left(-k_\mathrm{d}(t)||\boldsymbol{v}(t)||\boldsymbol{v}(t) + k_\mathrm{m}(t)(\boldsymbol{\omega}(t) \times \boldsymbol{v}(t)) + \mathbf{g}\right) \\
\boldsymbol{\omega}(t+\Delta_t) &= \boldsymbol{\omega}(t), \quad a_\mathrm{d}(t+\Delta_t) = a_\mathrm{d}(t), \quad a_\mathrm{m}(t+\Delta_t) = a_\mathrm{m}(t)
\end{aligned} \tag{6.21}$$

The function $\boldsymbol{z}(t+\Delta_t) = g_\mathrm{free}(\boldsymbol{z}(t), \Delta_t)$ abbreviates Equation 6.21. To refer to states at discrete time indices $n \in \{1, \ldots, N\}$ we use the notation $\boldsymbol{z}_{n+1} = g_\mathrm{free}(\boldsymbol{z}_n, \Delta_T)$. In our setting, $\Delta_T = \frac{1}{180\,\mathrm{s}^{-1}} \approx 5.56\,\mathrm{ms}$ as the cameras of the video tracking system are triggered with a fixed frequency of $180\,\mathrm{s}^{-1}$.

Impact model An impact of the ball with the table occurs within a discrete-time increment from n to $n+1$ if the lower edge of the ball (with radius r) at $p_{\mathrm{z},n+1} - r$ penetrates the table, i.e., $p_{\mathrm{z},n+1} - r < \mathrm{z}_\mathrm{table}$. We approximate the time of impact $\Delta_\mathrm{imp} \in [0, \Delta_T]$ with a simplified model to avoid numerical instabilities. It incorporates the velocity of the ball at the last discrete timestep before the impact $v_{\mathrm{z},n}$, the gravitational acceleration g_z, and the height difference to the table $h = -((p_{\mathrm{z},n} - r) - \mathrm{z}_\mathrm{table})$ such that $\Delta_\mathrm{imp} = -(v_{\mathrm{z},n} + \sqrt{v_{\mathrm{z},n} \cdot v_{\mathrm{z},n} + 2g_\mathrm{z}h})/g_\mathrm{z}$. The state of the ball just before the impact is given by $\boldsymbol{z}^- = g_\mathrm{free}(\boldsymbol{z}_n, \Delta_\mathrm{imp})$. At the time of impact, the velocity and spin are updated according to Equation 6.19, yielding the state $\boldsymbol{z}^+$ after impact. We denote this by $\boldsymbol{z}^+ = \boldsymbol{C}'\boldsymbol{z}^-$ with $\boldsymbol{C}' = \mathrm{blockdiag}(\boldsymbol{I}_3, \boldsymbol{C}, 1, 1)$. After the impact a free-flight phase follows, such that at the next discrete timestep, the state is $\boldsymbol{z}_{n+1} = g_\mathrm{free}(\boldsymbol{z}^+, \Delta_T - \Delta_\mathrm{imp})$.

Joint model We denote our discrete-time forward step, incorporating free flight and impacts, as

$$\boldsymbol{z}_{n+1} = g(\boldsymbol{z}_n) = \begin{cases} g_\mathrm{free}(\boldsymbol{z}_n, \Delta_T) & \text{if } \left[g_\mathrm{free}(\boldsymbol{z}_n, \Delta_T)\right]_\mathrm{z} - r \geq \mathrm{z}_\mathrm{table} \\ g_\mathrm{free}(\boldsymbol{z}^+, \Delta_T - \Delta_\mathrm{imp}) & \text{otherwise,} \end{cases} \tag{6.22}$$

where $[\boldsymbol{z}]_\mathrm{z}$ extracts the z-coordinate of the position in state $\boldsymbol{z}$.

6.4.3. Extended Kalman Filter (EKF)

For filtering and prediction, we assume the ball dynamics as in Equation 6.22 with additive Gaussian noise

$$\hat{\boldsymbol{z}}_{n+1} = g(\hat{\boldsymbol{z}}_n) + \boldsymbol{\zeta}, \quad \boldsymbol{\zeta} \sim \mathcal{N}(0, \mathrm{diag}([\boldsymbol{\sigma}_q]^{\oplus})), \quad \boldsymbol{\sigma}_q \in \mathbb{R}^{11}. \tag{6.23}$$

We obtain measurements of the ball's center $\boldsymbol{m}_n \in \mathbb{R}^3$ through a vision tracking system, which we assume to be perturbed by additive Gaussian measurement noise, i.e. $\boldsymbol{m}_n = \hat{\boldsymbol{p}}_n + \boldsymbol{\epsilon}$, $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathrm{diag}([\boldsymbol{\sigma}_r]^{\oplus}))$ with $\boldsymbol{\sigma}_r \in \mathbb{R}^3$. We estimate a belief of the state $p(\hat{\boldsymbol{z}}_n \mid \boldsymbol{m}_{1:n})$, given past position measurements $\boldsymbol{m}_{1:n}$, with an extended Kalman filter. Occasionally, the vision tracking system is unable to compute a position estimate (e.g. due to occlusions), which leads to missing measurements. To this end, we introduce the operator $\tau(n)$, which maps to the index of the n^{th} *available* measurement.

State initialization We are interested in initializing the state belief at the time when the second measurement is available, i.e. $p(\hat{\boldsymbol{z}}_{\tau(2)} | \boldsymbol{m}_{\tau(1)}, \boldsymbol{m}_{\tau(2)})$. The expected ball's position is estimated by the second measurement, $\boldsymbol{p} = \boldsymbol{m}_{\tau(2)}$, and the expected velocity $\boldsymbol{v}$ by a finite difference approximation $\boldsymbol{v} = (\boldsymbol{m}_{\tau(2)} - \boldsymbol{m}_{\tau(1)})/(\Delta_T(\tau(2) - \tau(1)))$. The initial values for the position and velocity covariance are learned and denoted by $\boldsymbol{\Sigma}_p = \mathrm{diag}([\boldsymbol{\sigma}_p]^{\oplus})$, $\boldsymbol{\Sigma}_v = \mathrm{diag}([\boldsymbol{\sigma}_v]^{\oplus})$. Before a table impact has happened, we can relate the ball's spin to the launcher parameters (as we assume the spin to be constant within the free-flight phase). After an impact has happened, this is no longer possible, as the impact changes the initial spin of the ball. We indicate by $\mathbf{1}_{\text{ai}}$ whether $\boldsymbol{m}_{\tau(2)}$ is taken after an impact. To compute a belief for the initial spin, we first assume the launcher's head to be oriented horizontally along the x-axis. For this launch orientation, we compute a "canonical spin" and its covariance depending on the motor parameters $\boldsymbol{s}_{\text{m}}$ (see Subsection 6.5.1), which we denote by $\boldsymbol{\omega}_{\to \text{x}} = f^{\omega}(\boldsymbol{s}_{\text{m}}, \Psi_f)$, $\boldsymbol{\Sigma}_{\omega_{\to \text{x}}} = \mathrm{diag}([f^{\boldsymbol{\Sigma}_\omega}(\boldsymbol{s}_{\text{m}}, \Psi_f)]^{\oplus})$. The functions f^{ω}, $f^{\boldsymbol{\Sigma}_\omega}$ are implemented by a neural network with two heads with parameters Ψ_f. The azimuthal launch orientation can be changed by rotating the whole launcher frame by ϕ_f and by rotating the launcher's head by ϕ_l (see Figure 6.2c). The elevational launch orientation can be changed by rotating the launcher's head by θ_l. To obtain the initial spin in the world coordinate system, we rotate the "canonical" spin $\boldsymbol{\omega}_{\to \text{x}}$ accordingly. We absorb all rotations in a rotation matrix $\mathbf{R}_{\text{rot}}(\phi_f + \phi_l, \theta_l)$, such that $\boldsymbol{\omega} = \mathbf{R}_{\text{rot}} \boldsymbol{\omega}_{\to \text{x}}$, $\boldsymbol{\Sigma}_\omega = \mathbf{R}_{\text{rot}} \boldsymbol{\Sigma}_{\omega_{\to \text{x}}} \mathbf{R}_{\text{rot}}^{\top}$. These considerations only hold true for the free-flight phase after ball launch. After a table impact has happened, we cannot directly relate the ball spin to the launcher parameters. Therefore, in this case, we set the moments of the initial spin $\boldsymbol{\omega} = \mathbf{0}$, $\boldsymbol{\Sigma}_\omega = \mathrm{diag}([\boldsymbol{\sigma}_{\omega,\text{ai}}]^{\oplus})$. During training, we obtain the angles ϕ_l, θ_l from piecewise linear regression models which map from launcher parameters $\boldsymbol{s}_\phi$, $\boldsymbol{s}_\theta$ to ϕ_l, θ_l. We obtained these models on the training split of trajectories recorded from the *default* launcher orientation. For the default orientation, we assume the launcher to shoot balls in the $-$y direction, i.e. $\phi_f = -90°$. When evaluating the filter in simulation or on the real robot, we infer the total azimuthal launch angle $\phi_f + \phi_l$ from the first two measurements of the trajectory, to avoid measuring the orientation of the launcher frame ϕ_f. In summary, we provide the information $(\phi_f, \phi_l, \theta_l, \boldsymbol{s}_{\text{m}}, \mathbf{1}_{\text{ai}})$ of the ball launch to the model. As an ablation, we initialize $\boldsymbol{\omega} = \mathbf{0}, \boldsymbol{\Sigma}_\omega = \mathrm{diag}([\boldsymbol{\sigma}_\omega]^{\oplus})$, not depending on this information. For the drag and Magnus effect coefficient, we learn the mean and covariance of the initial state. The full initial state belief incorporating the first two available measurements is thus given by $\boldsymbol{\mu}_{\tau(2)|\tau(1:2)} = (\boldsymbol{p}^{\top}, \boldsymbol{v}^{\top}, \boldsymbol{\omega}^{\top}, a_{\text{d}}, a_{\text{m}})^{\top}$, $\boldsymbol{\Sigma}_{\tau(2)|\tau(1:2)} = \mathrm{blockdiag}(\boldsymbol{\Sigma}_p, \boldsymbol{\Sigma}_v, \boldsymbol{\Sigma}_\omega, [\sigma_{a_{\text{d}}}]^{\oplus}, [\sigma_{a_{\text{m}}}]^{\oplus+})$.

In summary, the parameters of our gray-box model are

$$\Psi = \left(\mathbf{C}, \sigma_q, \sigma_r, \sigma_p, \sigma_v, \boldsymbol{\psi}_f, \sigma_\omega, \boldsymbol{\sigma}_{\omega,\mathrm{ai}}, a_\mathrm{d}, a_\mathrm{m}, \sigma_{a_\mathrm{d}}, \sigma_{a_\mathrm{m}}\right).$$

For the initial values of the parameters in Ψ and details on the spin initialization network, please see Appendix C.1.

Prediction step We follow the standard extended Kalman filter prediction step, yielding the prediction mean and covariance matrix $\boldsymbol{\mu}_{n+1|1:n} = g(\boldsymbol{\mu}_{n|1:n})$, $\boldsymbol{\Sigma}_{n+1|1:n} = \mathbf{J}\boldsymbol{\Sigma}_{n|1:n}\mathbf{J}^\top + \mathbf{Q}$, where $\mathbf{J}$ is the Jacobian matrix of the transition model, $\mathbf{J} = \frac{\partial}{\partial \boldsymbol{z}_n} g(\boldsymbol{z}_n)|_{\boldsymbol{z}_n = \boldsymbol{\mu}_{n|1:n}}$.

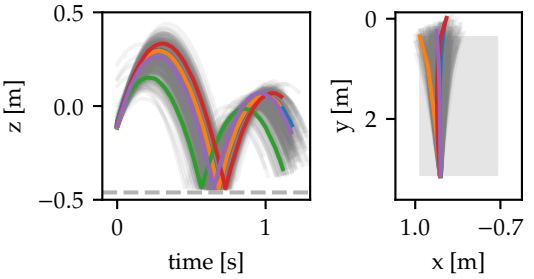
Correction step In the case of missing observations, we perform multiple prediction steps before correcting the state belief with the next available measurement. We now assume that at timestep $n+1$ a measurement is available. For the correction step, we first compute the Kalman gain $\mathbf{K}$ with the observation matrix $\mathbf{H} = [\mathbf{I}_3, \mathbf{0}^{3\times 8}]$ as $\mathbf{K} = \boldsymbol{\Sigma}_{n+1|1:n}\mathbf{H}^\top(\mathbf{H}\boldsymbol{\Sigma}_{n+1|1:n}\mathbf{H}^\top + \mathbf{R})^{-1}$. From this, we obtain the corrected moments as $\boldsymbol{\mu}_{n+1|1:n+1} = \boldsymbol{\mu}_{n+1|1:n} + \mathbf{K}(\mathbf{m}_{n+1} - \mathbf{H}\boldsymbol{\mu}_{n+1|1:n})$, $\boldsymbol{\Sigma}_{n+1|1:n+1} = (\mathbf{I} - \mathbf{K}\mathbf{H})\boldsymbol{\Sigma}_{n+1|1:n}$.

Learning For notational simplicity, we again assume that there are no missing observations. Let $\mathcal{P} = \{(\hat{\mathbf{m}}_1^k, \ldots, \hat{\mathbf{m}}_{L_k}^k)\}_{k=1}^K$ denote the set of training trajectories, consisting of K sequences of ball position measurements, each sequence being of length L_k. The chunk operator expands a single trajectory into $L+1-N$ chunks of length $N = 50$: $\mathrm{chunk}(\hat{\mathbf{m}}_1, \ldots, \hat{\mathbf{m}}_L) = \{(\mathbf{m}_i, \ldots, \mathbf{m}_{i+N-1})\}_{i=1}^{L+1-N}$. By $\mathcal{P}_c = \bigcup_{k=1}^K \mathrm{chunk}(\hat{\mathbf{m}}_1^k, \ldots, \hat{\mathbf{m}}_{L_k}^k)$ we denote the set of training chunks and $\Delta(\mathcal{P}_c)$ the distribution over training chunks with uniform probability. For learning the filter parameters Ψ, we maximize their expected marginal log-likelihood under the training chunk distribution, that is, $\max_\Psi \mathbb{E}_{\mathbf{m}_{1:N}\sim\Delta(\mathcal{P}_c)} \log p(\mathbf{m}_{3:N} \mid \mathbf{m}_1, \mathbf{m}_2, \Psi)$. As we do not aim to learn a generative model of chunks but are only interested in applying the learned model for filtering and prediction, we additionally condition the marginal log-likelihood on the first two measurements, since we use these for initializing the filter. The marginal log-likelihood $\log p(\mathbf{m}_{3:N} \mid \mathbf{m}_1, \mathbf{m}_2, \Psi)$ can be decomposed (Särkkä, 2013) as follows

$$\begin{aligned} &\log p(\mathbf{m}_{3:N} \mid \mathbf{m}_1, \mathbf{m}_2, \Psi) \\ &\quad= \sum_{n=3}^{N} \log p(\mathbf{m}_n \mid \mathbf{m}_{1:n-1}, \Psi) \\ &\quad= \sum_{n=3}^{N} \log \mathcal{N}(\mathbf{m}_n \mid \boldsymbol{\mu}_{n|1:n-1}, \boldsymbol{\Sigma}_{n|1:n-1}) \end{aligned} \tag{6.24}$$

which allows for an iterative computation with complexity $O(N)$ when filtering the chunk $(\mathbf{m}_1, \ldots, \mathbf{m}_N)$. We maximize the expected marginal log-likelihood on batches of chunks with batchsize 64 using the Adam optimizer (Kingma and Ba, 2015) with learning rate $5 \cdot 10^{-3}$.

(a) Setup with the ball launcher (left) and the robot arm (right).



(b) Dataset of recorded table tennis trajectories, launched from the "default" orientation. Five randomly selected trajectories are colored.

Figure 6.1.: Experimental setup (a) and visualization of recorded trajectories (b).

6.5. Experiments

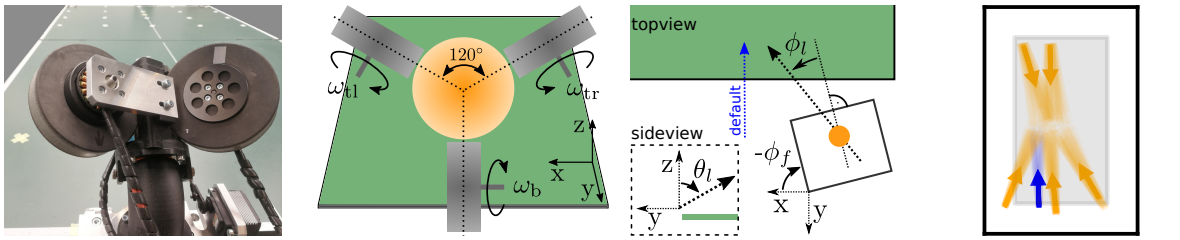
We conduct several experiments to answer the following research questions: **Q1:** How large is the prediction error of the EKF model, and how does it compare to black box baselines? **Q2:** Does supplying the launch parameters (launch direction, launcher motor speeds) to the predictive model improve prediction performance for the EKF and the RSSM baseline? We use a neural network to infer the initial ball spin from motor parameters, leading us to **Q3:** Does the spin inferred from the launcher parameters relate to a simple spin model derived from physical principles? Finally, we are interested in the ratio of balls returned by a robot arm (Büchler et al., 2016) when using the proposed model for trajectory prediction (**Q4**).

6.5.1. Setup

In Figure 6.1 we show our experimental setup. A ball launcher shoots table tennis balls towards a robot arm that is actuated with pneumatic artificial muscles (Büchler et al., 2016). The position of the ball is measured using four RGB cameras as described in Gómez-González et al. (2019). Details of the ball launcher are described in Subsection 6.5.1. The robot is only used for the return experiment in Subsection 6.5.6. For training the predictive models, we use recorded trajectories (Subsection 6.5.1).

Launcher details

The ball launcher follows the design of Dittrich et al. (2022). It accelerates the table tennis ball using three rubber wheels which are actuated by brushless motors. The azimuthal and elevational angle of the launcher's head can be adjusted with servo motors to change the direction of the launch. The launcher frame can be freely positioned and rotated about the z-axis, as parameterized by the angle ϕ_f. We refer to Figure 6.2 for more details on the launcher, including its geometry and the launch angles. The angular velocity of the top-left, top-right, and bottom motor are controlled through three actuation parameters $\boldsymbol{s}_\mathrm{m} = (s_\mathrm{m,tl}, s_\mathrm{m,tr}, s_\mathrm{m,b})^\top \in [0,1]^3$. The mapping from actuation parameters to angular motor velocity is nonlinear, see Figure 6.3. The azimuthal (ϕ_l) and elevational (θ_l) launch angles can be controlled through two parameters $s_\phi \in [0,1], s_\theta \in [0,1]$. We fit piecewise linear functions to the launch angles of the recorded



(a) Ball launcher.

(b) Schematic drawing (rear view).

(c) Schematic drawing (top-down/side view).

(d) Launch directions (top-down view). → *default*, → *unseen* launch direction.

Figure 6.2.: Experimental setup of the ball launcher. (a) Photo of the launcher (taken in the direction of ball launch, with table in background), (b) a schematic drawing of the launcher with rotating wheels (gray) and ball (orange), (c) frame (ϕ_f) and launch angles (ϕ_l, θ_l), (d) shoot directions for the "default" and "unseen" configurations.

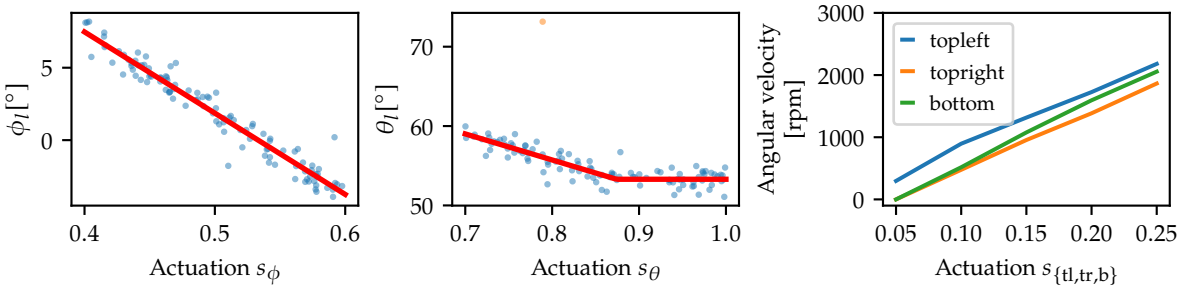
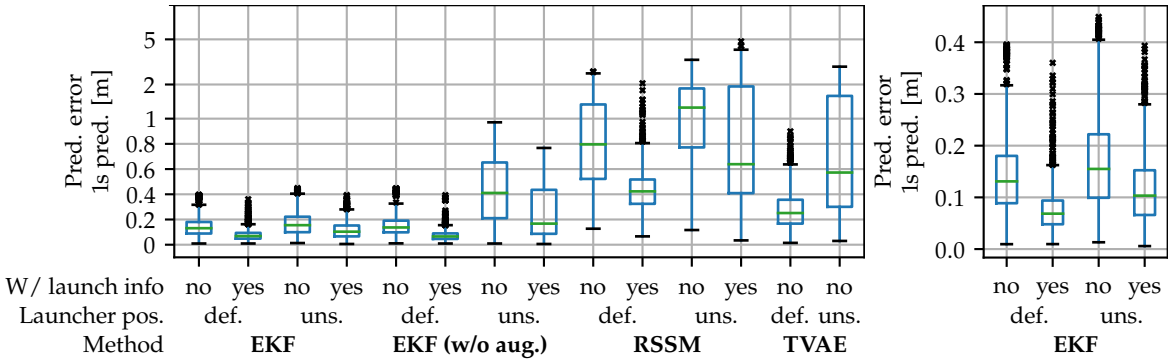


Figure 6.3.: Relation of launcher parameters $\boldsymbol{s} \in [0,1]^5$ to launch angles ϕ_l, θ_l and angular velocity of launcher wheels. The left two panels show the result of piecewise linear regression (red) to initial trajectory angles (blue); the right panel shows angular velocity of wheels depending on the actuation. The outlier (orange, center panel) is excluded.

trajectories to find a mapping from the actuation parameters $s_\phi \in [0,1]$, $s_\theta \in [0,1]$ to launch angles ϕ_l, θ_l (see Figure 6.3). The azimuthal launch direction can further be changed by rotating the launcher's frame about the z-axis by the angle ϕ_f. For the *default* orientation, we oriented the launcher such that it shoots along the negative y-axis for $\phi_l = 0$, i.e., $\phi_f = -90°$.

Data recording

For collecting trajectories for training, validation, and testing, we position the launcher at six different positions and orientations (see Figure 6.2d). We term one particular position/orientation *default*, which we use both for training and testing, and the other five *unseen*, which we use for testing only. On the *default* orientation we collect 334 trajectories, which we split in 108 for training, 63 for validation, and 163 for testing. For each of the five *unseen* configurations we collect 30 trajectories which are used for testing only. For each trajectory we randomly sample the launcher parameters uniformly from $s_\phi \in [0.4, 0.6]$, $s_\theta \in [0.7, 1.0]$, $s_{\text{tl}} \in [0.095, 0.155]$ (default), $s_{\text{tl}} \in [0.105, 0.165]$ (unseen), $s_{\{\text{tr,b}\}} \in [0.135, 0.195]$ (default), $s_{\{\text{tr,b}\}} \in [0.145, 0.205]$ (unseen). To simulate different launcher orientations, we optionally augment the training data by rotating each trajectory by a random angle $\phi_f \in [0, 2\pi]$ about the z-axis at the point with minimal z coordinate. We add 19 rotated trajectories for each existing trajectory to the training set, which forms the augmented dataset.



(a) All methods (change of scale from linear to log at $y = 1$). **(b)** Detailed view for EKF.

Figure 6.4.: Prediction error for various methods on the test set of default (def.) and unseen (uns.) launcher positions, with and without ball launch information (launch info), for a prediction horizon of one second (see Subsection 6.5.2). Depicted statistics are over the prediction errors for ten independently trained models. All models are trained on the augmented dataset, except "EKF w/o aug.". The EKF model outperforms the RSSM (Hafner et al., 2019) and TVAE (Gómez-González et al., 2020) models (a). The EKF's prediction error can further be reduced by providing ball launch information (b).

6.5.2. Prediction performance

Evaluation protocol The predictive performance of the investigated models is quantified by measuring the prediction error when filtering until one second before the trajectory ends, and predicting the remaining part of the trajectory. For shorter trajectories, we filter at least ten measurements. The prediction error for each sequence is given by the maximum Euclidean distance between the last five prediction-measurement pairs.

Baselines As a first baseline, we train a recurrent state-space model, taken from a re-implementation of the PlaNet model by Hafner et al. (2019), implemented by Arulkumaran (2021). We inherit the standard parameters except for the "free nats" parameter, which we determined empirically as 0.3 for minimal average prediction error on the validation split of the *default* dataset. Optionally, we pass the same ball launch information used in the EKF model for state initialization as action to the RSSM model. We train the model for 100,000 steps. As a second baseline, we train a trajectory variational auto-encoder (TVAE) from Gómez-González et al. (2020), using the provided implementation by Gómez-González (2022). We use a model length of 250 as our longest trajectory is 235 steps. We train the model until the validation loss increases.

Results We refer to Figure 6.4 for a visualization of the results. We observe that augmenting the training data is important for the EKF approach presented herein to generalize to the *unseen* launcher positions (Figure 6.4a). In all settings, the EKF approach shows superior performance compared to the RSSM and TVAE baselines. Figure 6.4b shows that initializing the spin using ball launch information reduced the prediction error drastically, both on the *default* and *unseen* launcher configurations. We show representative filtering and prediction results on three trajectories in Figure 6.5.

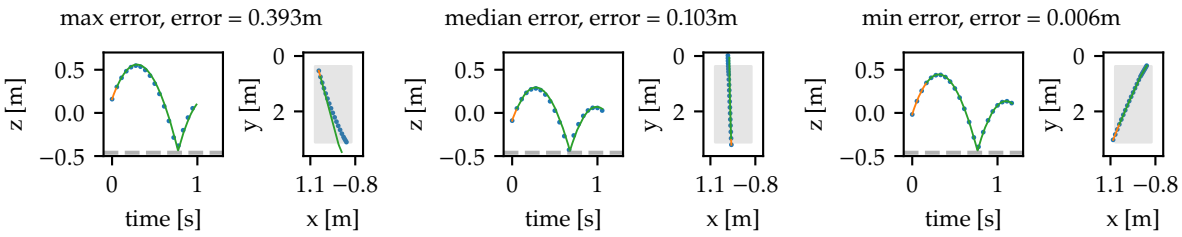
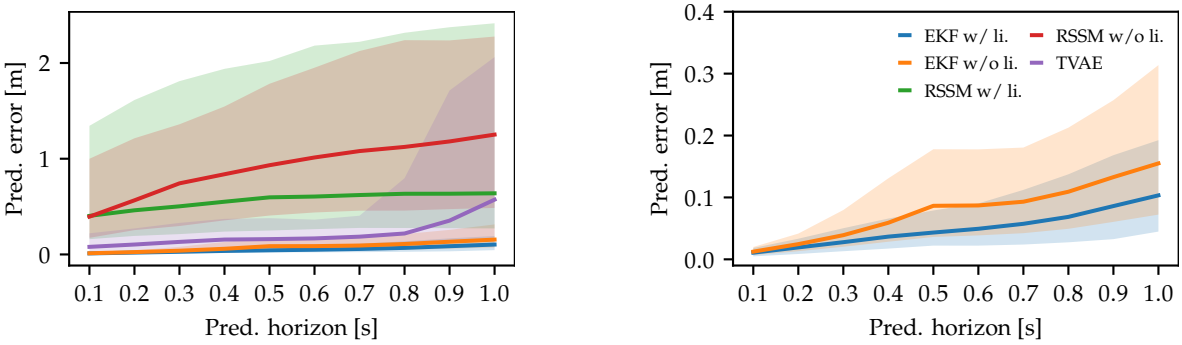


Figure 6.5.: EKF filtering / prediction results on the *unseen* launcher orientations (with ball launch information). We filter until one second before the end of the trajectory (orange) and predict the remaining one second (green). Measurements are colored blue. We show every tenth measurement for visual clarity. Shown are the trajectories with max. / median / min. prediction error at the end of the trajectory from all *unseen* launcher configurations, over ten independently trained models. The median prediction error is 10.3 cm.



(a) All methods.

(b) EKF methods only (note different scaling of y-axis).

Figure 6.6.: Prediction error for varying prediction horizons. We evaluated the prediction error on 150 evaluation trajectories from the *unseen* dataset, for 10 independently trained models per method. We show the median error as a solid line, the shaded area covers values between the 10^{th} and 90^{th} percentile. We compare variants with (w/ li.) and without (w/o li.) providing launch information.

6.5.3. Prediction error for varying prediction horizons

We refer to Figure 6.6 for a visualization of the prediction error for varying prediction horizons. We filter the trajectory from its beginning until the respective prediction horizon remains. For each trajectory, the prediction error is given by the maximum Euclidean distance over the last five measurement-prediction pairs, as stated in Subsection 6.5.2. The proposed EKF method clearly outperforms the baselines RSSM (Hafner et al., 2019) and TVAE (Gómez-González et al., 2020) for all considered horizons. Supplying ball launch information (w/ li.) generally reduces the prediction error.

6.5.4. Spin evaluation

With this experiment, we aim to verify the plausibility of spins which are estimated by the learned neural network $f^{\omega}(\boldsymbol{s}_{\text{m}}, \boldsymbol{\psi}_f)$ given launcher parameters $\boldsymbol{s}_{\text{m}}$ (see Subsection 6.4.3). For this, we formulate a simple model for the ball spin, which is derived from the geometry of the launcher (see Figure 6.2b). We model the ball's spin for a launcher which is oriented to shoot

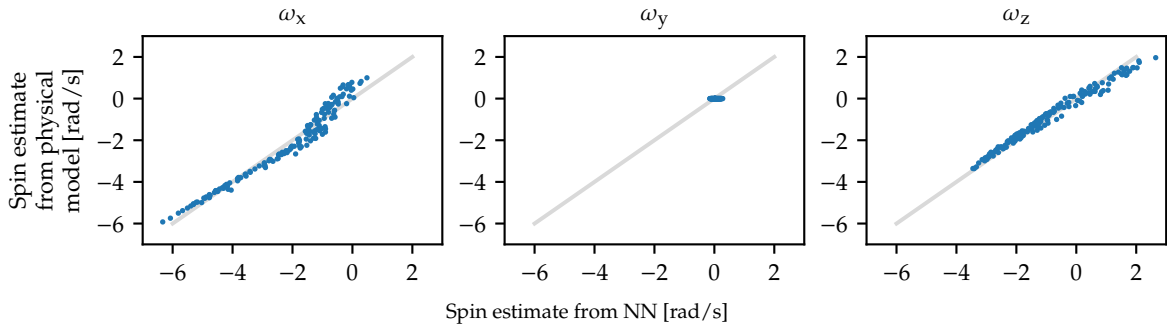


Figure 6.7.: Correlation of spin values estimated by the neural network (NN) $f^{\omega}(\boldsymbol{s}_{\mathrm{m}}, \boldsymbol{\psi}_f)$ from motor actuations $\boldsymbol{s}_{\mathrm{m}}$ and spin values computed with Equation 6.25. Both estimated spin values highly correlate, indicating physical plausibility of spins estimated by $f^{\omega}(\boldsymbol{s}_{\mathrm{m}}, \boldsymbol{\psi}_f)$, see Subsection 6.5.4 for details.

balls in the −y direction as

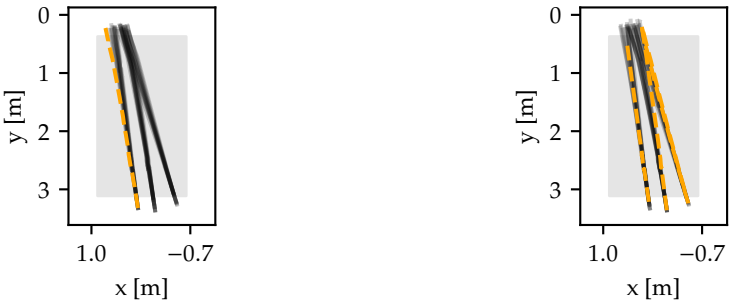
$$\begin{aligned}\left(\omega_{\mathrm{x},\overrightarrow{-y}}, \omega_{\mathrm{y},\overrightarrow{-y}}, \omega_{\mathrm{z},\overrightarrow{-y}}\right)^{\top} = \\ \omega_{\mathrm{tl}}\alpha\left(\frac{1}{2}, 0, -\frac{\sqrt{3}}{2}\right)^{\top} + \omega_{\mathrm{tr}}\beta\left(\frac{1}{2}, 0, \frac{\sqrt{3}}{2}\right)^{\top} + \omega_{\mathrm{b}}\gamma(-1, 0, 0)^{\top}.\end{aligned} \tag{6.25}$$

The reasoning behind the model is that every motor adds a spin component to the ball, which is the motor speed $(\omega_{\mathrm{tl}}, \omega_{\mathrm{tr}}, \omega_{\mathrm{b}})$ scaled by a constant (α, β, γ). We note that the bottom motor causes a negative spin about the x-axis. The direction of the spin of the top motors is obtained by rotating the bottom-motor spin unit vector $(-1, 0, 0)^{\top}$ by 120° (240°) about the y-axis. For this experiment, we obtain the values for $\omega_{\mathrm{tl,tr,b}}$ from measurements for the angular launcher wheel velocity given the actuation parameters (see Figure 6.3). For all *test* trajectories from the *default* dataset, we first compute the spin for a launch in x direction with $f^{\omega}(\boldsymbol{s}_{\mathrm{m}}, \boldsymbol{\psi}_f)$. We rotate this spin by −90° about the z-axis to obtain the spin in −y direction, as in Equation 6.25. Finally, we obtain the parameters α, β, γ by minimizing a squared error between the rotated spins from f^{ω} and the spins estimated by Equation 6.25. In Figure 6.7 we show that the two spin estimates highly correlate, indicating that the values $f^{\omega}(\boldsymbol{s}_{\mathrm{m}}, \boldsymbol{\psi}_f)$ indeed relate to the actual spin of the ball. It is important to note that without additional physical information, we can determine the spin only up to a scaling factor. This is because we both learn $k_{\mathrm{m}} = a_{\mathrm{m}}^2 + \epsilon$ and $\boldsymbol{\omega}$ which appear as a product in Equation 6.18. The actual spin could be inferred as $\boldsymbol{\omega}^* = k_{\mathrm{m}}\boldsymbol{\omega}/k_{\mathrm{m}}^*$ with $k_{\mathrm{m}}^* = C_{\mathrm{m}}\rho A r/(2m)$ for known values of the ball's mass m, Magnus lift coefficient C_{m}, ball radius r, air density ρ and cross-sectional area $A = \pi r^2$.

6.5.5. Parameter analysis

For the impact matrix $\boldsymbol{C}$ we learn

$$\boldsymbol{C} = \left[\begin{array}{c|c} \boldsymbol{C}_{\mathrm{vv}} & \boldsymbol{C}_{\mathrm{v}\omega} \\ \hline \boldsymbol{C}_{\omega\mathrm{v}} & \boldsymbol{C}_{\omega\omega} \end{array}\right] = \left[\begin{array}{ccc|ccc} \mathbf{0.54} & -0.01 & -0.00 & -0.01 & \mathbf{0.12} & 0.01 \\ 0.01 & \mathbf{0.55} & 0.00 & \mathbf{-0.11} & -0.01 & -0.01 \\ -0.01 & -0.00 & \mathbf{-0.92} & 0.01 & 0.00 & 0.00 \\ \hline 0.19 & \mathbf{-1.40} & 0.01 & \mathbf{-0.20} & 0.09 & -0.03 \\ \mathbf{1.44} & 0.14 & -0.03 & -0.01 & \mathbf{-0.12} & -0.01 \\ 0.01 & -0.05 & \mathbf{0.15} & 0.05 & -0.05 & \mathbf{1.37} \end{array}\right]. \tag{6.26}$$



(a) Launch information is provided to the EKF: 29/30 launches are returned.

(b) Launch information is *not* provided to the EKF: 26/30 launches are returned.

Figure 6.8.: Trajectories of balls launched towards the robot arm (at $y \approx 0$), with unreturned trajectories colored orange. See Subsection 6.5.6 for details.

We interpret $\boldsymbol{C}$ to be composed of four submatrices, mapping velocities and spins before to velocities and spins after impact. For each row in each submatrix, we highlight the dominating component. Submatrix $\boldsymbol{C}_{\mathrm{vv}}$ maps velocities before impact to velocities after impact. Velocities are dampened in all directions x, y, z and have no interacting effects. As expected, only the velocity in z-direction flips sign. Submatrix $\boldsymbol{C}_{\mathrm{v}\omega}$ maps spins before impact to velocities after impact. A positive spin about the y-axis increases the x-velocity after impact ($\omega_{\mathrm{y}}\uparrow \rightarrow v_{\mathrm{x}}\uparrow$); a positive spin about the x-axis decreases the y-velocity after impact ($\omega_{\mathrm{x}}\uparrow \rightarrow v_{\mathrm{y}}\downarrow$). Investigating $\boldsymbol{C}_{\omega\mathrm{v}}$, analogously, a negative y-velocity increases spin about the x-axis ($v_{\mathrm{y}}\downarrow \rightarrow \omega_{\mathrm{x}}\uparrow$), and a positive x-velocity increases spin about the y-axis ($v_{\mathrm{x}}\uparrow \rightarrow \omega_{\mathrm{y}}\uparrow$). The abovementioned relations between spins and velocities before and after impact are physically plausible, considering the geometry of the table setup. A negative velocity in z direction (i.e., towards the table) reduces the spin about z as the bottomright entry of $\boldsymbol{C}_{\omega\mathrm{v}}$ is positive (0.15). This relation is reasonable due to friction effects. That our simplified linear model only approximately captures physical reality becomes apparent when investigating $\boldsymbol{C}_{\omega\omega}$, i.e., the mapping of spins before impact to spins after impact. The bottomright entry of $\boldsymbol{C}_{\omega\omega}$ suggests that the spin about the z-axis is amplified by the impact. We hypothesize that this effect cancels with the spin attenuation due to negative z-velocities. We leave posing constraints on $\boldsymbol{C}$, e.g. for rotational symmetries, and nonlinear impact effects, for future work.

For the learned values of the remaining parameters, please see Appendix C.1.2.

6.5.6. Return performance

We evaluate the performance of our ball motion predictions by intercepting and returning balls with a four-degrees-of-freedom robot arm, where each degree of freedom is controlled by a pair of pneumatic artificial muscles (PAMs) (Büchler et al., 2023, 2016). The robot arm is controlled by a learning-based iterative control framework for trajectory tracking (Ma et al., 2022). A table tennis racket is attached to the robot arm in order to return balls. As the ping-pong ball flies through the air, its position, velocity, and spin are continuously estimated with the EKF presented herein. The future evolution of the ball's states is then simulated in a receding horizon scheme using the learned model of the ball dynamics, with the latest state estimate as the initial condition. This predicted ball trajectory is used to determine the interception of the ball with the racket, which we represent as a pair of position (the interception position of the ball and the racket) and

time (the time of interception). The predicted trajectory, and therefore the interception point, is repeatedly recalculated, as the state estimate is updated and improved with new measurements of the ball. The robot arm successfully returns 29 of 30 (97.7 %) launched balls, leveraging the EKF for filtering and prediction presented above. When the launcher information is not used to initialize the spin (see Subsection 6.4.3), 26 of 30 launched balls are returned. We refer to Figure 6.8 for a visualization of returned and unreturned trajectories.

6.6. Limitations

This section is not part of Achterhold et al. (2023).

In this chapter, to predict the ball's trajectory, we assume information on the ball launch process to be available. In our case, this information is available as parameters of the ball launcher. However, tracking and predicting the ball motion should also be possible for balls played by humans. In literature, evidence is presented that for anticipating the ball trajectory, humans, similar to our method, also leverage additional cues. Exemplarily, results by Zhao et al. (2018) indicate that advanced table tennis players are able to better anticipate ball trajectories by observed body kinematics, compared to novice players. Klein-Soetebier et al. (2020) conclude that table tennis players can discriminate different ball rotations by the impact sound of the ball on the racket, and that artificially removing these cues by wearing headphones negatively impacts the player's performance. Conclusively, in future work, we aim to replace the ball launcher parameters by detections of the racket in through inertial measurement data (Blank et al., 2017) and/or video (Gao et al., 2021), human pose estimation (Kulkarni and Shenoy, 2021), or auditory cues.

6.7. Conclusion

Based on a physically grounded model for the aerodynamic behavior of a flying ball respecting Magnus and drag effects and the extended Kalman filter, we have designed a filter and predictive model for table tennis ball trajectories. As we fit the parameters of the filter on offline data, no tedious tuning of initial, transition, and observation covariances is required. Our formulation allows for learning a neural model which estimates the ball's initial spin from ball launch information. This drastically improves the performance of long-term predictions compared to an uninformed initialization. Our results also support the findings of other works, which state that the ball's spin can only insufficiently be estimated from ball position measurements alone (Blank et al., 2017; Y. Zhang et al., 2015). Our method could constitute groundwork for future research which, e.g., incorporates information on the racket movement to estimate the ball's spin and predict its future trajectory with high accuracy. A further interesting avenue for future work is to include ball pose measurements, e.g. by detecting the brand logo (Y. Zhang et al., 2015) or other patterns (Gossard et al., 2023), into the model.

7. Learning Temporally Extended Skills for Planning

Declaration of contributions

The contents of this chapter are based on the peer-reviewed conference publication

J. Achterhold, M. Krimmel, and J. Stueckler (2022). 'Learning Temporally Extended Skills in Continuous Domains as Symbolic Actions for Planning'. In: *Proceedings of the Conference on Robot Learning (CoRL)* (Achterhold et al., 2022).

The above publication was presented as an **oral** presentation at the *Conference on Robot Learning (CoRL) 2022, Auckland, New Zealand*.

Author contributions are as follows:

	Scientific ideas	Data generation	Analysis & Interpretation	Paper writing
Jan Achterhold	70 %	90 %	80 %	75 %
Markus Krimmel	0 %	10 %	0 %	5 %
Jörg Stückler	30 %	0 %	20 %	20 %

Jörg Stückler proposed to investigate learning combinations of symbolic representations and low-level (subsymbolic) skills. Jan Achterhold conceived the idea of learning skill-conditioned policies jointly with a forward model, modeling the effect of skill execution, through mutual information maximization. Jan Achterhold conceived the idea of using physically embedded single-player board games as benchmarking environments. Jörg Stückler regularly provided feedback on the approach during its development. Jan Achterhold implemented the majority of the algorithm, except the BFS planner, which was implemented by Markus Krimmel. Jan Achterhold performed and analyzed the experiments. Jan Achterhold and Jörg Stückler wrote the majority of the paper, Markus Krimmel contributed pseudocode for the planning algorithm.

We provide additional materials at `https://seads.is.tue.mpg.de`. We provide our implementation at `https://github.com/EmbodiedVision/seads-environments` (physically embedded single-player board game environments) and `https://github.com/EmbodiedVision/seads-agent` (SEADS agent).

7.1. Introduction

Reinforcement learning (RL) agents have been applied to difficult continuous control and discrete planning problems such as the DeepMind Control Suite (Tassa et al., 2018), StarCraft II (Vinyals et al., 2019), or Go (D. Silver et al., 2016) in recent years. Despite this tremendous success, tasks which require both continuous control capabilities and long-horizon discrete planning are classically approached with task and motion planning (Garrett et al., 2021). These problems still pose significant challenges to RL agents (Mirza et al., 2020). An exemplary class of environments

which require both continuous-action control and long-horizon planning are *physically embedded games* as introduced by Mirza et al. (2020). In these environments, a board game is embedded into a physical manipulation setting. A move in the board game can only be executed indirectly through controlling a physical manipulator such as a robotic arm. We simplify the setting of Mirza et al. (2020) and introduce physically embedded *single-player* board games which do not require to model the effect of an opponent. Our experiments support the findings of Mirza et al. (2020) that these environments are challenging to solve for existing flat and hierarchical RL agents. In this chapter, we propose a novel hierarchical RL agent for such environments which learns skills and their effects in a known symbolic abstraction of the environment.

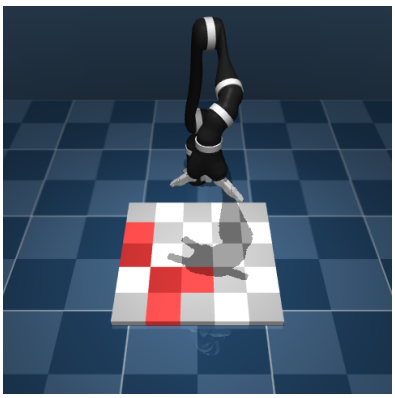


Figure 7.1.: The *LightsOutJaco* environment.

A concrete example for a proposed embedded single-player board game is the *LightsOutJaco* environment (see Figure 7.1). Pushing a field on the *LightsOut* board toggles the illumination state (*on* or *off*) of the field and its non-diagonal neighboring fields. A field on the board can only be pushed by the end effector of the *Jaco* robotic arm. The goal is to reach a board state in which all fields are *off*. The above example also showcases the two concepts of *state* and *action* abstraction in decision making (Konidaris, 2019). A state abstraction function $\Phi(\boldsymbol{s}_t)$ only retains information in state $\boldsymbol{s}_t$ which is relevant for a particular decision making task. In the LightsOut example, to decide which move to perform next (i.e., which field to push), only the illumination state of the board is relevant. A *move* can be considered an *action abstraction*: A skill, i.e. high-level action (e.g., push top-left field), comprises a sequence of low-level actions required to control the robotic manipulator.

We introduce a two-layer hierarchical agent which assumes a discrete state abstraction $\boldsymbol{z}_t = \Phi(\boldsymbol{s}_t) \in \mathcal{Z}$ to be known and observable in the environment, which we in the following refer to as *symbolic observation*. In our approach, we assume that state abstractions can be defined manually for the environment. For LightsOut, the symbolic observation corresponds to the state of each field (on/off). We provide the state abstraction as prior knowledge about the environment and assume that skills induce changes of the abstract state. Our approach then learns a diverse set of skills for the given state abstraction as action abstractions and a corresponding forward model which predicts the effects of skills on abstract states. In board games, these abstract actions relate to *moves*. We jointly learn the predictive forward model q_θ and *skill policies* $\pi(\boldsymbol{a}_t \mid \boldsymbol{s}_t, k)$ for low-level control through an objective which maximizes the number of symbolic states reachable from any state of the environment (diversity) and the predictability of the effect of skill execution. The forward model q_θ can be leveraged to plan a sequence of skills to reach a particular state of the board (e.g., all fields off), i.e. to solve tasks. We evaluate our approach using two single-player board games in environments with varying complexity in continuous control. We demonstrate that our agent learns skill policies and forward models suitable for solving the associated tasks with high success rate and compares favorably with other flat and hierarchical RL baseline agents. We also demonstrate our agent playing LightsOut with a real robot.

In summary, we contribute the following:

- We formulate a novel RL algorithm, which, based on a state abstraction of the environment and an information-theoretic objective, jointly learns a diverse set of continuous-action

skills and a forward model capturing the temporally abstracted effect of skill execution in symbolic states. We term our agent SEADS for Symbolic Effect-Aware Diverse Skills.

- We demonstrate the superiority of our approach compared to other flat and hierarchical baseline agents in solving complex physically-embedded single-player games, requiring high-level planning and continuous control capabilities.

7.2. Related work

Diverse skill learning and skill discovery Discovering general skills to control the environment through exploration without task-specific supervision is a fundamental challenge in RL research. DIAYN (Eysenbach et al., 2019) formulates skill discovery using an information-theoretic objective as reward. The agent learns a skill-conditioned policy for which it receives reward if the target states can be well predicted from the skill. VALOR (Achiam et al., 2018) proposes to condition the skill prediction model on the complete trajectory of visited states. Warde-Farley et al. (2019) train a goal-conditioned policy to reach diverse states in the environment. Variational Intrinsic Control (Gregor et al., 2017) proposes to use an information-theoretic objective to learn a set of skills which can be identified from their initial and target states. Relative Variational Intrinsic Control (Baumli et al., 2021) seeks to learn skills relative to their start state, aiming to avoid skill representations that merely tile the state space into goal state regions. Both approaches do not learn a forward model on the effect of skill execution like our approach. Sharma et al. (2020) propose a model-based RL approach (DADS) which learns a set of diverse skills and their dynamics models using mutual-information-based exploration. While DADS learns skill dynamics as immediate behavior $q(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, k)$, we learn a transition model on the effect of skills $q(\boldsymbol{z}_T|\boldsymbol{z}_0, k)$ in a symbolic abstraction, thereby featuring temporal abstraction.

Hierarchical reinforcement learning Hierarchical RL can overcome sparse reward settings and time extended tasks by breaking the task down into subtasks. Some approaches such as methods based on MAXQ (Dietterich, 2000; Z. Li et al., 2017) assume prior knowledge on the task-subtask decomposition. In SAC-X (Riedmiller et al., 2018), auxiliary tasks assist the agent in learning sparse reward tasks and hierarchical learning involves choosing between tasks. Florensa et al. (2017) propose to learn a span of skills using stochastic neural networks for representing policies. The policies are trained in a task-agnostic way using a measure of skill diversity based on mutual information. Specific tasks are then tackled by training an RL agent based on the discovered skills. Feudal approaches (Dayan and Hinton, 1992) such as HIRO (Nachum et al., 2018) and HAC (Levy et al., 2019) train a high-level policy to provide subgoals for a low-level policy. In our method, we impose that a discrete state-action representation exists in which learned skills are discrete actions, and train the discrete forward model and the continuous skill policies jointly. Several approaches to hierarchical RL are based on the options framework (Sutton et al., 1999) which learns policies for temporally extended actions in a two-layer hierarchy. Learning in the options framework is usually driven by task rewards. Recent works extend the framework to continuous spaces and discovery of options (e.g. Bacon et al. (2017) and Bagaria and Konidaris (2020)). HiPPO (A. C. Li et al., 2020) develops an approximate policy gradient method for hierarchies of actions. HIDIO (J. Zhang et al., 2021) learns task-agnostic options using a measure of diversity of the skills. In our approach, we also learn task-agnostic (for the given state abstraction) hierarchical representations using a measure of intrinsic motivation. However, an

important difference is that we do not learn high-level policies over options using task rewards, but learn a skill-conditional forward model suitable for planning to reach a symbolic goal state. Jointly, continuous policies are learned which implement the skills. Several approaches combine symbolic planning in a given domain description (state and action abstractions) with RL to execute the symbolic actions (Guan et al., 2022; Illanes et al., 2020; Kokel et al., 2021; Lyu et al., 2019; M. R. K. Ryan, 2002). Similar to our approach, the approach by Guan et al. (2022) learns low-level skill policies using an information-theoretic diversity measure which implement known symbolic actions. Differently, we learn the action abstraction and low-level skills given the state abstraction.

Representation learning for symbolic planning Some research has been devoted to learning representations for symbolic planning. Konidaris et al. (2018) propose a method for acquiring a symbolic planning domain from a set of low-level options which implement abstract symbolic actions. In James et al. (2020) the approach is extended to learning symbolic representations for families of SMDPs which describe options in a variety of tasks. Our approach learns action abstractions as a set of diverse skills given a known state abstraction and a termination condition which requires abstract actions to change abstract states. Toro Icarte et al. (2019) learn structure and transition models of finite state machines through reinforcement learning. Ugur and Piater (2015) acquire symbolic forward models for a predefined low-level action repertoire in a robotic manipulation context. Chitnis et al. (2022) concurrently learn transition models on the symbolic and low levels from demonstrations provided in the form of hand-designed policies, and use the learned models for bilevel task and motion planning. The approach also assumes the state abstraction function to be known. In T. Silver et al. (2022) a different setting is considered in which the symbolic transition model is additionally assumed known and skill policies that execute symbolic actions are learned from demonstrations. Other approaches such as DeepSym (Ahmetoglu et al., 2022) or LatPlan (Asai and Fukunaga, 2018) learn mappings of images to symbolic states and learn action-conditional forward models. In Asai and Fukunaga (2018) symbolic state-action representations are learned from image observations of discrete abstract actions (e.g. moving puzzle tiles to discrete locations) which already encode the planning problem. Our approach concurrently learns a diverse set of skills (discrete actions) based on an information-theoretic intrinsic reward and the symbolic forward model. Differently, in our approach low-level actions are continuous.

7.3. Method

Our goal is to learn a hierarchical RL agent which (i) enables high-level, temporally abstract planning to reach a particular goal configuration of the environment (as given by a symbolic observation) and (ii) features continuous control policies to execute the high-level plan. Let $\mathcal{S}, \mathcal{A}$ denote the state and action space of an environment, respectively. In general, by $\mathcal{Z} = \{0,1\}^D$ we denote the space of discrete symbolic environment observations $\boldsymbol{z} \in \mathcal{Z}$ and assume the existence of a state abstraction $\Phi : \mathcal{S} \to \mathcal{Z}$. The dimensionality of the symbolic observation D is environment-dependent. For the *LightsOutJaco* environment, the state $\boldsymbol{s} = [\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{z}] \in \mathcal{S}$ contains the robot arms' joint positions and velocities $(\boldsymbol{q}, \dot{\boldsymbol{q}})$ and a binary representation of the board $\boldsymbol{z} \in \{0,1\}^{5\times5}$. The action space $\mathcal{A}$ is equivalent to the action space of the robotic manipulator. In the *LightsOutJaco* example, it contains the target velocity of all actuable joints. The discrete

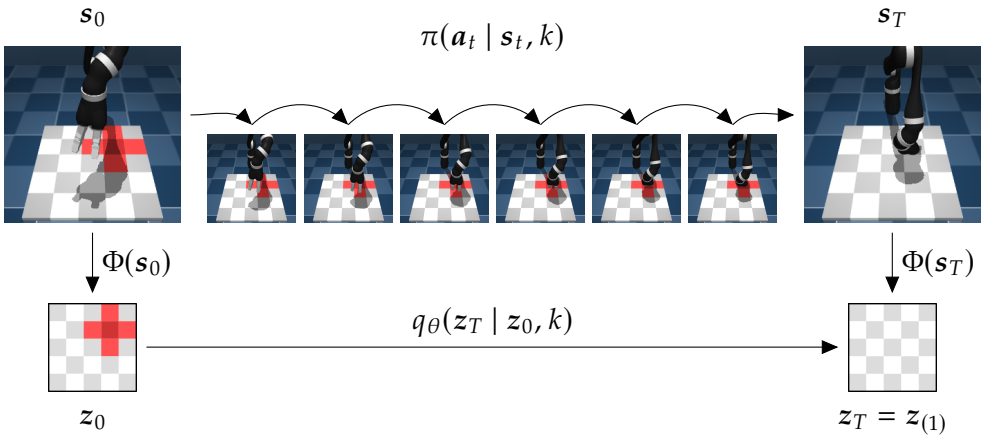


Figure 7.2.: Symbolic abstraction Φ and temporal skill abstraction demonstrated on the *LightsOutJaco* environment. The symbolic observation z represents the discrete state of the board while s contains both the board state and the state of the *Jaco* manipulator. Executing skill k by applying the skill policy $\pi(\cdot|\cdot, k)$ until termination leads to a change of the state of the board, which is modeled by q_θ with a single action k.

variable $k \in \mathcal{K}, \mathcal{K} = \{1, \ldots, K\}$ refers to a particular *skill*, which we will detail in the following. The number of skills K needs to be set in advance, but can be chosen larger than the number of actual skills.

We equip our agent with symbolic planning and plan execution capabilities through two components.

First, a forward model

$$\hat{z} = \arg\max_{z'} q_\theta(z' \mid z, k) \tag{7.1}$$

allows to *enumerate* all possible symbolic successor states $\hat{z}$ of the current symbolic state z by iterating over the discrete variable k. This allows for node expansion in symbolic planners.

Second, a family of discretely indexed policies (*skills*)

$$\pi : \mathcal{A} \times \mathcal{S} \times \mathcal{K} \to \mathbb{R}, \; a_t \sim \pi(a_t \mid s_t, k) \tag{7.2}$$

aims to steer the environment into a target state s_T for which it holds that $\Phi(s_T) = \hat{z}$, given that $\Phi(s_0) = z$ and $\hat{z} = \arg\max_{z'} q_\theta(z' \mid z, k)$ (see Figure 7.2).

7.3.1. Skill policies

We can relate this discretely indexed family of policies to a set of K *options* (Sutton et al., 1999). An option is formally defined as a triple $O_k = (\mathcal{S}_k^I, \beta_k, \pi_k)$ where $\mathcal{S}_k^I \subseteq \mathcal{S}$ is the set of states in which option k is applicable, $\beta_k(s_0, s_t) : \mathcal{S} \times \mathcal{S} \to [0, 1]$ parametrizes a Bernoulli probability of termination in state s_t when starting in s_0 and $\pi_k(a_t \mid s_t) : \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the option policy on the action space $\mathcal{A}$. We will refer to the option policy as *skill policy* in the following. We assume that all options are applicable in all states, i.e., $\mathcal{S}_k^I = \mathcal{S}$. An option terminates if the symbolic state has changed between s_0 and s_t or a timeout is reached, i.e., $\beta_k(s_0, s_t) = \mathbb{1}[(\Phi(s_0) \neq \Phi(s_t)) \vee (t = t_{\max})]$. To this end, we append a normalized counter $t/t_{\max}$ to the state s_t.

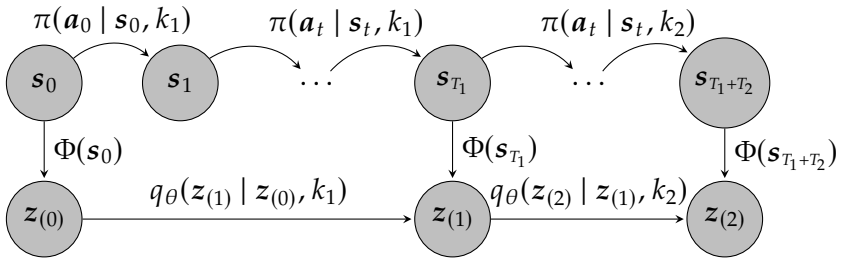


Figure 7.3.: Temporal abstraction induced by skills $\pi(\cdot \mid \cdot, k)$ with associated forward model q_θ on symbolic observations $\boldsymbol{z}_{(n)}$. Executing a skill until termination can be interpreted as a *single* action k transforming the symbolic observation $\boldsymbol{z}_{(n)} \rightarrow \boldsymbol{z}_{(n+1)}$ denoted by the bracketed $(\cdot)$ time indices.

We define the operator apply as

$$\boldsymbol{s}_T = \text{apply}(E, \pi, \boldsymbol{s}_0, k) \tag{7.3}$$

which applies the skill policy $\pi(\boldsymbol{a}_t \mid \boldsymbol{s}_t, k)$ until termination on environment E starting from initial state $\boldsymbol{s}_0$ and returns the terminal state $\boldsymbol{s}_T$. We also introduce a bracketed time notation which abstracts the effect of skill execution from the number of steps T taken until termination

$$\boldsymbol{s}_{(n)} = \text{apply}(E, \pi, \boldsymbol{s}_{(n-1)}, k) \tag{7.4}$$

with $n \in \mathbb{N}_0$. The apply operator can thus be rewritten as $\boldsymbol{s}_{(1)} = \text{apply}(E, \pi, \boldsymbol{s}_{(0)}, k)$ with $\boldsymbol{s}_{(0)} = \boldsymbol{s}_0$, $\boldsymbol{s}_{(1)} = \boldsymbol{s}_T$. We refer to Figure 7.3 for a visualization of the bracket notation.

7.3.2. Symbolic forward model

The symbolic forward model $q_\theta(\boldsymbol{z}_T \mid \boldsymbol{z}_0, k)$ aims to capture the relation of $\boldsymbol{z}_0$, k and $\boldsymbol{z}_T$ for $\boldsymbol{s}_T = \text{apply}(E, \pi, \boldsymbol{s}_0, k)$ with $\boldsymbol{z}_0 = \Phi(\boldsymbol{s}_0)$, $\boldsymbol{z}_T = \Phi(\boldsymbol{s}_T)$. It factorizes over the symbolic observation as

$$q_\theta(\boldsymbol{z}_T \mid k, \boldsymbol{z}_0) = \prod_{d=1}^{D} q_\theta([\boldsymbol{z}_T]_d \mid \boldsymbol{k}, \boldsymbol{z}_0) = \prod_{d=1}^{D} \text{Bernoulli}([\boldsymbol{z}_T]_d \mid [\boldsymbol{\alpha}_T(\boldsymbol{z}_0, k)]_d) \tag{7.5}$$

where D is the dimensionality of the symbolic observation*. We assume $\boldsymbol{z} \in \mathcal{Z}$ to be a binary vector with $\mathcal{Z} = \{0, 1\}^D$. The Bernoulli probabilities $\boldsymbol{\alpha}_T(\boldsymbol{z}_0, k) : \mathcal{Z} \times \mathcal{K} \rightarrow (0, 1)^D$ are predicted by a learnable neural component. We use a neural network f_θ to parameterize the probability of each dimension in $\boldsymbol{z}_0$ to flip $\boldsymbol{p}_{\text{flip}} = f_\theta(\boldsymbol{z}_0, k)$, which simplifies learning if the *change* in symbolic state only depends on k and is independent of the current state. Let $\boldsymbol{\alpha}_T$ be the probability that the binary state is "true" (for each dimension), then $\boldsymbol{\alpha}_T = (1 - \boldsymbol{z}_0) \cdot \boldsymbol{p}_{\text{flip}} + \boldsymbol{z}_0 \cdot (1 - \boldsymbol{p}_{\text{flip}})$. The input to the neural network is the concatenation $[\boldsymbol{z}_0, \text{onehot}(k, K)]$. We use a multilayer perceptron with two hidden layers with ReLU nonlinearities, each having 256 units.

7.3.3. Objective

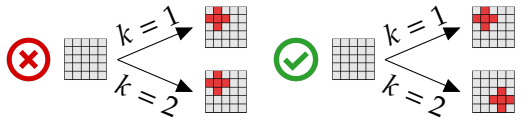
For any state $\boldsymbol{s}_0 \in \mathcal{S}$ with associated symbolic state $\boldsymbol{z}_0 = \Phi(\boldsymbol{s}_0)$ we aim to learn K skills $\pi(\boldsymbol{a}_t \mid \boldsymbol{s}_t, k)$ which maximize the diversity in the set of reachable successor states $\{\boldsymbol{z}_T^k = \Phi(\text{apply}(E, \pi, \boldsymbol{s}_0, k)) \mid k \in \mathcal{K}\}$. Jointly, we aim to model the effect of skill execution with the

* The index operator $[\boldsymbol{x}]_d$ returns the d^{th} element of vector $\boldsymbol{x}$.

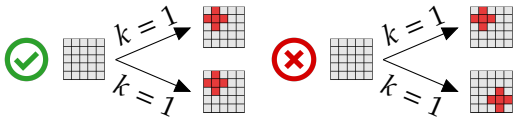
forward model $q_\theta(z_T \mid z_0, k)$. Inspired by Variational Intrinsic Control (Gregor et al., 2017) we take an information-theoretic perspective and maximize the mutual information $\mathrm{I}(z_T, k \mid z_0)$ between the skill index k and the symbolic observation z_T at skill termination given the symbolic observation z_0 at skill initiation, i.e.,

$$\max \mathrm{I}(z_T, k \mid z_0) = \max \left(\mathrm{H}(z_T \mid z_0) - \mathrm{H}(z_T \mid z_0, k)\right). \tag{7.6}$$

The intuition behind this objective function is that we encourage the agent to (i) reach a diverse set of terminal observations z_T from an initial observation z_0 (by maximizing the conditional entropy $\mathrm{H}(z_T \mid z_0)$, see Figure 7.4a) and (ii) behave predictably such that the terminal observation z_T is ideally fully determined by the initial observation z_0 and skill index k (by minimizing $\mathrm{H}(z_T \mid z_0, k)$, see Figure 7.4b).



(a) Diversity objective ($\mathrm{H}(z_T \mid z_0)$ ↑): Distinct skills should lead to distinct state transitions.



(b) Predictability objective ($\mathrm{H}(z_T \mid z_0, k)$ ↓): Identical skills should lead to identical state transitions.

Figure 7.4.: Visualization of the diversity and predictability objectives.

We reformulate the objective as an expectation over tuples (s_0, k, s_T) by employing the mapping function Φ as

$$\mathrm{I}(z_T, k \mid z_0) = \mathbb{E}_{(s_0,k,s_T)\sim P}\left[\log \frac{p(z_T \mid z_0, k)}{p(z_T \mid z_0)}\right] \tag{7.7}$$

with $z_T := \Phi(s_T)$, $z_0 := \Phi(s_0)$ and replay buffer P.

Similar to (Sharma et al., 2020) we derive a lower bound on the mutual information, which is maximized through the interplay of a RL problem and maximum likelihood estimation. To this end, we first introduce a variational approximation $q_\theta(z_T \mid z_0, k)$ to the transition probability $p(z_T \mid z_0, k)$, which we model by a neural component.

We decompose the mutual information as

$$\mathrm{I}(z_T, k \mid z_0) = \mathbb{E}_{(s_0,k,s_T)\sim P}\left[\log \frac{q_\theta(z_T \mid z_0, k)}{p(z_T \mid z_0)}\right] + \underbrace{\mathbb{E}_{(s_0,k,s_T)\sim P}\left[\log \frac{p(z_T \mid z_0, k)}{q_\theta(z_T \mid z_0, k)}\right]}_{\mathrm{KL}(p(z_T \mid z_0,k) \,||\, q_\theta(z_T \mid z_0,k))} \tag{7.8}$$

giving rise to the lower bound

$$\mathrm{I}(z_T, k \mid z_0) \geq \mathbb{E}_{(s_0,k,s_T)\sim P}\left[\log \frac{q_\theta(z_T \mid z_0, k)}{p(z_T \mid z_0)}\right] \tag{7.9}$$

whose maximization can be interpreted as a sparse-reward RL problem with reward

$$\hat{R}_T(k) = \log \frac{q_\theta(z_T \mid z_0, k)}{p(z_T \mid z_0)}. \tag{7.10}$$

We approximate $p(z_T \mid z_0)$ as

$$p(z_T \mid z_0) \approx \sum_{k'} q_\theta(z_T \mid z_0, k') p(k' \mid z_0) \tag{7.11}$$

and assume k uniformly distributed and independent of z_0, i.e. $p(k' \mid z_0) = \frac{1}{K}$. This yields a tractable reward

$$R_T(k) = \log \frac{q_\theta(z_T \mid z_0, k)}{\sum_{k'} q_\theta(z_T \mid z_0, k')} + \log K. \tag{7.12}$$

In Subsection 7.3.5 we describe modifications we apply to the intrinsic reward R_T which improve the performance of our proposed algorithm.

To tighten the lower bound, the KL divergence term in eq. (7.8) has to be minimized. Minimizing the KL divergence term corresponds to "training" the symbolic forward model q_θ by maximum likelihood estimation of the parameters $\boldsymbol{\theta}$ using gradient ascent

$$\nabla_\theta \mathbb{E}_{(\boldsymbol{s}_0, k, \boldsymbol{s}_T) \sim P} \left[\log \frac{p(z_T \mid z_0, k)}{q_\theta(z_T \mid z_0, k)} \right] = -\nabla_\theta \mathbb{E}_{(\boldsymbol{s}_0, k, \boldsymbol{s}_T) \sim P} \left[\log q_\theta(z_T \mid z_0, k) \right]. \tag{7.13}$$

7.3.4. Training procedure

The main training loop of our proposed SEADS agent consists of intermittent episode collection and training the skill-conditioned policy π and symbolic forward model q_θ (see Algorithm 2).

For episode collection we first sample a skill from a uniform distribution over skills $k \sim \mathcal{U}\{1, \ldots, K\}$, reset the environment, collect the corresponding episode and append it to a long-term ($\text{Episodes}_{\text{buffer}}$) and a short-term ($\text{Episodes}_{\text{recent}}$) episode buffer, holding the $N_{\text{buffer}} = 2048$ / $N_{\text{recent}} = 256$ most recent episodes, respectively.

In the following we refer to the symbolic forward model as a general *skill model*. We introduce this terminology, as, in Appendix D.7, we present results where the skill model is a skill discriminator instead of a forward model (akin to Variational Intrinsic Control, Gregor et al. (2017)). For training the skill policy and skill model we combine a sample of 256 episodes from the long-term buffer and all 256 episodes from the short-term buffer. To account for mismatches between policy executions and the predictions of the symbolic forward model, we *relabel* episodes. Please see the following paragraphs for details on episode collection, relabelling, skill policy- and skill model training.

Episode collection (collect_episode)

The operator collect_episode works similar to the apply operator defined in Equation 7.3. It applies the skill policy $\pi(\boldsymbol{a}_t \mid \boldsymbol{s}_t, k)$ iteratively until termination. However, the operator returns all intermediate states $\boldsymbol{s}_0, \ldots, \boldsymbol{s}_T$ and actions $\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{T-1}$. Let Episodes be a collection of M episodes and $i \in \{1, \ldots, M\}$ denote a single episode Ep from this collection. This episode is a tuple $\text{Ep} = (k^i, \boldsymbol{s}_0^i, \ldots, \boldsymbol{s}_{T^i}^i, \boldsymbol{a}_0^i, \ldots, \boldsymbol{a}_{T^i-1}^i)$ where T^i is the episode length. A skill rollout terminates either if an environment-dependent step-limit is reached, or when a change in the symbolic observation $z_t \neq z_0$ is observed. In our experiments we collect 32 episodes per epoch (i.e., $N_{\text{episodes}} = 32$).

Algorithm 2: SEADS training loop

Input: Environment E, Number of skills K, Number of epochs N_{epochs}, Number of new episodes per epoch N_{episodes}, Episode buffer sizes N_{buffer}, N_{recent}
Result: Trained skill-conditioned policy $\pi(a \mid s, k)$ and forward model $q_\theta(z_T \mid z_0, k)$
$\text{Episodes}_{\text{buffer}} = []$, $\text{Episodes}_{\text{recent}} = []$
for $n_{\text{epoch}} = 1$ **to** N_{epochs} **do**
  for $n_{\text{episode}} = 1$ **to** N_{episodes} **do**
    Sample $k \sim k \sim \mathcal{U}\{1, \dots, K\}$
    $s_0 = E.\text{reset}()$
    $\text{Ep} = \text{collect\_episode}(E, \pi, s_0, k)$
    $\text{Episodes}_{\text{buffer}}.\text{append}(\text{Ep})$, $\text{Episodes}_{\text{recent}}.\text{append}(\text{Ep})$
  end for
  $\text{Episodes}_{\text{buffer}} \leftarrow \text{Episodes}_{\text{buffer}}[\text{-}N_{\text{buffer}}:]$ {Keep N_{buffer} most recent episodes}
  $\text{Episodes}_{\text{recent}} \leftarrow \text{Episodes}_{\text{recent}}[\text{-}N_{\text{recent}}:]$
  $\text{Episodes} = \text{sample}(\text{Episodes}_{\text{buffer}}, N = 256) \cup \text{Episodes}_{\text{recent}}$
  $\text{Episodes}_{\text{SM}} \leftarrow \text{relabel}(\text{Episodes}, p = 1.0)$
  $\text{update\_skill\_model}(\text{Episodes}_{\text{SM}})$
  $\text{Episodes} = \text{sample}(\text{Episodes}_{\text{buffer}}, N = 256) \cup \text{Episodes}_{\text{recent}}$
  $\text{Episodes}_{\text{SP}} \leftarrow \text{relabel}(\text{Episodes}, p = 0.5)$
  $\text{update\_skill\_policy}(\text{Episodes}_{\text{SP}})$
end for

Relabelling (relabel**)**

Early in training, the symbolic transitions caused by skill executions mismatch the predictions of the symbolic forward model. We can in *hindsight* increase the match between skill transitions and forward model by replacing the actual k^i which was used to collect the episode i by a different k^i_*. In particular, we aim to replace k^i by k^i_* which has highest probability $k^i_* = \max_k q_\theta(k \mid \boldsymbol{z}^i_{T^i}, \boldsymbol{z}^i_0)$. However, this may lead to an unbalanced distribution over k^i_* after relabelling, which is no longer uniform. To this end, we introduce a constrained relabelling scheme as follows. We consider a collection of episodes indexed by $i \in \{1, \dots, M\}$ and compute skill log-probabilities for each episode which we denote by

$$Q^i_k = \log q_\theta(k \mid \boldsymbol{z}^i_0, \boldsymbol{z}^i_{T^i}) \tag{7.14}$$

where

$$q_\theta(k \mid \boldsymbol{z}^i_0, \boldsymbol{z}^i_{T^i}) = \frac{q_\theta(\boldsymbol{z}^i_{T^i} \mid \boldsymbol{z}^i_0, k)}{\sum_{k'} q_\theta(\boldsymbol{z}^i_{T^i} \mid \boldsymbol{z}^i_0, k')}. \tag{7.15}$$

We find a relabeled skill for each episode $(k^1_*, \dots, k^M_*)$ which maximizes the scoring $\max_{(k^1_*, \dots, k^M_*)} \sum_i Q^i_{k^i_*}$ under the constraint that the counts of re-assigned skills $(k^1_*, \dots, k^M_*)$ and original skills $(k^1, \dots, k^M)$ match, i.e.

$$\sum_{i=1}^{M} \mathbb{1}[k^i_* = k] = \sum_{i=1}^{M} \mathbb{1}[k^i = k] \quad \forall k \in \{1, \dots, K\}, \tag{7.16}$$

which is to ensure that after relabelling no skill is over- or underrepresented. This problem can be formulated as a linear sum assignment problem which we solve using the Hungarian method (Kuhn, 1955; Munkres, 1957).

For each episode in the argument of the relabel operator, we sample a Bernoulli variable with success probability of p, indicating whether it may be relabeled. For training the skill model we relabel all episodes ($p = 1$), while for the skill policies we only allow half of the episodes to be relabeled ($p = 0.5$). The idea is to train the skill policies also on *negative* examples of skill executions with small rewards. Episodes in which the symbolic observation did not change are excluded from relabelling for the skill policies, as for those, the reward is constant $R(k) = -2\log(K) \quad \forall k \in \mathcal{K}$ (see Subsection 7.3.5).

Relabelling experience in hindsight to improve sample efficiency is a common approach in goal-conditioned (Andrychowicz et al., 2017) and hierarchical (Levy et al., 2019) RL.

The union of potentially relabelled episodes and episodes which were excluded from relabelling form the updated buffer which is returned by the relabel operator.

Skill policy update (update_skill_policy**)**

The skill policies π are implemented and updated via the soft actor-critic (SAC) algorithm (Haarnoja et al., 2018). A buffer of transitions, with each transition being of the form

$$T = ([\boldsymbol{s}_t^i, \text{onehot}(k^i, K)], \boldsymbol{a}_t^i, [\boldsymbol{s}_{t+1}^i, \text{onehot}(k^i, K)], r_{t+1}^i), \tag{7.17}$$

is formed from all episodes in the episode collection $\text{Episodes}_{\text{SP}}$. The intrinsic reward r_{t+1}^i is set to zero except for the last transition in an episode ($t + 1 = T^i$), in which $r_{t+1}^i = R(k^i)$ according to Subsection 7.3.5. The operator $[\cdot, \cdot]$ denotes a concatenation of a state and one-hot encoding of the skill index.

The skill policies are updated with 16 steps per epoch on batches comprising 128 randomly sampled transitions from the transition buffer. For architectural details on the SAC skill policies, see Appendix D.2.

Skill model update (update_skill_model**)**

We train the skill model on batches of tuples of the form $(\boldsymbol{z}_0, k, \boldsymbol{z}_T)$, constructed from episodes in the episode buffer $\text{Episodes}_{\text{SM}}$. To sample a batch $\mathcal{B}$, we first sample a set $\mathcal{M}$ of 32 indices, each in $\{1, \ldots, M\}$, where M is the number of episodes in $\text{Episodes}_{\text{SM}}$. The batch $\mathcal{B}$ is then formed as $\mathcal{B} = \{(\boldsymbol{z}_0^i, k^i, \boldsymbol{z}_{T^i}^i)\}_{i \in \mathcal{M}}$. The skill model is trained to minimize an expected loss

$$\mathcal{L} = \mathbb{E}_{\mathcal{B}} \left[\sum_{(\boldsymbol{z}_0, k, \boldsymbol{z}_T) \in \mathcal{B}} \ell(\boldsymbol{z}_0, k, \boldsymbol{z}_T) \right] \tag{7.18}$$

for randomly sampled batches $\mathcal{B}$ of transition tuples. We optimize the skill model parameters $\boldsymbol{\theta}$ using the Adam (Kingma and Ba, 2015) optimizer on four randomly sampled batches per epoch. We use a learning rate of 10^{-3}. The instance-wise loss ℓ to be minimized corresponds to the negative log-likelihood $\ell = -\log q_\theta(\boldsymbol{z}_T \mid \boldsymbol{z}_0, k)$ for symbolic forward models or $\ell = -\log q_\theta(\boldsymbol{k} \mid \boldsymbol{z}_0, \boldsymbol{z}_T)$ for the VIC ablation (see Appendix D.7).

7.3.5. Reward improvements

The reward in eq. (7.12) can be denoted as

$$R(k) = \log q_\theta(k \mid z_0, z_T) + \log K. \tag{7.19}$$

For numerical stability, we define a lower bounded term

$$\bar{Q}_k = \text{clip}(\log q_\theta(k \mid z_0, z_T), \text{min} = -2\log(K)) \tag{7.20}$$

and write $R^0(k) = \bar{Q}_k + \log K$.

In our experiments, we observed that occasionally the agent is stuck in a local minimum in which (i) the learned skills are not unique, i.e., two or more skills $k \in \mathcal{K}$ cause the same symbolic transition $z_0 \rightarrow z_T$. In addition, (ii), occasionally, not all possible symbolic transitions are discovered by the agent.

To tackle (i) we reinforce the policy π with a positive reward if and only if no other skill k' better fits the symbolic transition $(z_0 \rightarrow z_T)$ generated by $\text{apply}(E, \pi, \boldsymbol{s}_0, k)$, i.e.,

$$R^{\text{norm}}(k) = \bar{Q}_k - \text{top2}_{k'} \bar{Q}_{k'} \tag{7.21}$$

which we call **second-best normalization**. The operator $\text{top2}_{k'}$ selects the second-highest value of its argument for $k' \in \mathcal{K}$. We define $R^{\text{base}}(k) = R^{\text{norm}}(k)$ except for the "No second-best norm." ablation where $R^{\text{base}}(k) = R^0(k)$.

To improve (ii) the agent obtains a **novelty bonus** for transitions $(z_0 \rightarrow z_T)$ which are not modeled by the symbolic forward model for *any* k' by

$$R(k) = R^{\text{base}}(k) - \max_{k'} \log q_\theta(z_T \mid z_0, k'). \tag{7.22}$$

For the "No novelty bonus" ablation, we set $R(k) = R^{\text{base}}(k)$.

If the symbolic state does not change (i.e., $z_T = z_0$), we set $R(k) = -2\log(K)$, which is the minimum attainable reward due to the clipping operation in Equation 7.20.

7.3.6. Planning and skill execution

A task is presented to our agent as an initial state of the environment $\boldsymbol{s}_0$ with associated symbolic observation z_0 and a symbolic goal z^*. First, we leverage our learned symbolic forward model q_θ to plan a sequence of skills $k_1, \ldots, k_N$ from z_0 to z^* using breadth-first search (BFS). We use the mode of the distribution over z' for node expansion in BFS:

$$\text{successor}_{q_\theta}(z, k) = \text{argmax}_{z' \in \mathcal{Z}}\, q_\theta(z' \mid z, k). \tag{7.23}$$

After planning, the sequence of skills $[k_1, \ldots, k_N]$ is iteratively applied to the environment through $s_{(n)} = \text{apply}(E, \pi, \boldsymbol{s}_{(n-1)}, k_n)$. Inaccuracies of skill execution (leading to different symbolic observations than predicted) can be coped with by replanning after each skill execution.

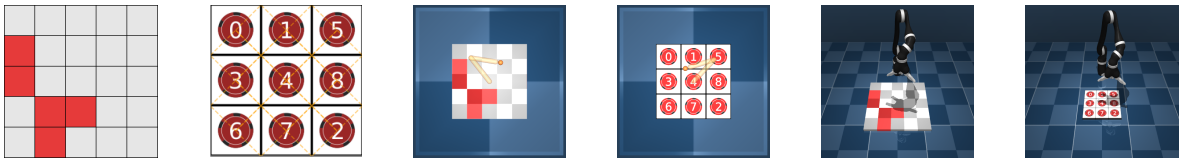
Algorithm 3: Task solution (without replanning). bfs_plan denotes breadth-first search over a sequence of skills to transition $z_{(0)}$ to z^*, leveraging the symbolic forward model q_θ. Nodes are expanded in BFS via the function $\text{successor}_{q_\theta} : \mathcal{Z} \times \mathcal{K} \to \mathcal{Z}$.

Input: environment E, skill-conditioned policy π, symbolic forward model q_θ, initial state $s_{(0)} = s_0$, symbolic goal z^*, symbolic mapping function Φ
Output: boolean success
$N, [k_1, \ldots, k_N] = \text{bfs\_plan}(q_\theta, z_{(0)} = \Phi(s_{(0)}), z^*)$
for $n = 1$ **to** N **do**
  $s_{(n)} = \text{apply}(E, \pi, s_{(n-1)}, k_i)$
end for
$\text{success} = (\Phi(s_{(N)}) == z^*)$

Both single-outcome (mode) determinisation and replanning are common approaches to probabilistic planning (Yoon et al., 2007). We provide pseudocode for task solution without replanning in Algorithm 3, and vice versa, with replanning, in Algorithm 4.

Algorithm 4: Task solution (with replanning, max. 10 tries). bfs_plan denotes breadth-first search over a sequence of skills to transition $z_{(n)}$ to z^*, leveraging the symbolic forward model q_θ. Nodes are expanded in BFS via the function $\text{successor}_{q_\theta} : \mathcal{Z} \times \mathcal{K} \to \mathcal{Z}$.

Input: environment E, skill-conditioned policy π, symbolic forward model q_θ, initial state $s_{(0)} = s_0$, symbolic goal z^*, symbolic mapping function Φ
Output: boolean success
$n \leftarrow 0$, success = False
for $m = 1$ **to** 10 **do**
  $N, [k_1, \ldots, k_N] \leftarrow \text{bfs\_plan}(q_\theta, z_{(n)} = \Phi(s_{(n)}), z^*)$
  for $i = 1$ **to** N **do**
    $n \leftarrow n + 1$
    $\hat{z}_{(n)} \leftarrow \text{successor}_{q_\theta}(\Phi(s_{(n-1)}), k_i)$ {Predict the symbolic state when applying skill k_i}
    $s_{(n)} \leftarrow \text{apply}(E, \pi, s_{(n-1)}, k_i)$
    $z_{(n)} \leftarrow \Phi(s_{(n)})$
    if $z_{(n)} \neq \hat{z}_{(n)}$ **then**
      Break {If the actual symbolic state differs from the predicted state, we replan}
    end if
  end for
  $\text{success} = (\Phi(s_{(N)}) == z^*)$
  if success **then**
    Break {Done}
  end if
end for



LightsOut-Cursor *TileSwapCursor* *LightsOut-Reacher* *TileSwap-Reacher* *LightsOutJaco* *TileSwapJaco*

Figure 7.5.: *LightsOut* (with on (red) and off (gray) fields) and *TileSwap* (fields in a rhombus are swapped if pushed inside) board games embedded into physical manipulation settings. A move in the board game can only indirectly be executed through controlling a manipulator.

7.4. Experiments

7.4.1. Environments

We evaluate our proposed agent on a set of physically-embedded game environments. We follow ideas from Mirza et al. (2020), but consider single-player board games, which, in principle, enable full control over the environment without the existence of an opponent. We chose *LightsOut* and *TileSwap* as board games, which are embedded in a physical manipulation scenario with *Cursor*, *Reacher* or *Jaco* manipulators (see Figure 7.5).

The *LightsOut* game consists of a 5×5 board of fields. Each field has a binary illumination state of *on* or *off*. By pushing a field, its illumination state and the state of the (non-diagonally) adjacent fields toggles. At the beginning of the game, the player is presented a board where some fields are on and the others are off. The task of the player is to determine a set of fields to push to obtain a board where all fields are off. The symbolic observation in all *LightsOut* environments represents the illumination state of all 25 fields on the board $\mathcal{Z} = \{0, 1\}^{5\times5}$.

In *TileSwap* a 3×3 board is covered by chips numbered from 0 to 8 (each field contains exactly one chip). Initially, the chips are randomly assigned to fields. Two chips can be swapped if they are placed on (non-diagonally) adjacent fields. The game is successfully finished after a number of swap operations if the chips are placed on the board in ascending order. In all *TileSwap* environments, the symbolic observation represents whether the i-th chip is located on the j-th field $\mathcal{Z} = \{0, 1\}^{9\times9}$.

A board game move ("push" in *LightsOut*, "swap" in *TileSwap*) is triggered by the manipulator's end effector touching a particular position on the board. We use three manipulators of different complexity. Here, we only give a short introduction and refer to Appendix D.1 for more details.

Cursor The *Cursor* manipulator can be navigated on the 2D game board by commanding x and y displacements. The board coordinates are $x, y \in [0, 1]$, the maximum displacement per timestep is $\Delta x, \Delta y = 0.2$. A third action triggers a push (*LightsOut*) or swap (*TileSwap*) at the current position of the cursor.

Reacher The *Reacher* manipulator (Tassa et al., 2018) consists of a two-link arm with two rotary joints. The position of the end effector in the 2D plane can be controlled by external torques applied to the two rotary joints. As for the *Cursor* manipulator, an additional action triggers a game move at the current end effector coordinates.

Table 7.1.: Number of feasible board configurations for varying solution depths. No feasible board configurations exist with solution depth > 16.

	solution depth							
	1	2	3	4	5	6	7	8
LightsOut	25	300	2300	12650	53130	176176	467104	982335
TileSwap	12	88	470	1978	6658	18081	38936	65246
	solution depth							
	9	10	11	12	13	14	15	16
LightsOut	1596279	1935294	1684446	1004934	383670	82614	7350	0
TileSwap	83000	76688	48316	18975	4024	382	24	1

Table 7.2.: Number of initial board configurations for varying solution depths and dataset splits.

		solution depth				
		1	2	3	4	5
LightsOut	train	7	99	785	4200	17849
	test	18	201	1515	8450	35281
	total	25	300	2300	12650	53130
TileSwap	train	7	31	179	683	2237
	test	5	57	291	1295	4421
	total	12	88	470	1978	6658

Jaco The *Jaco* manipulator (Campeau-Lecours et al., 2017) is a 9-DoF robotic arm whose joints are velocity-controlled at $10\,\mathrm{s}^{-1}$. It has an end-effector with three “fingers" which can touch the underlying board to trigger game moves. The arm is reset to a random configuration above the board around the board’s center after a game move.

By combining the games of *LightsOut* and *TileSwap* with the *Cursor*, *Reacher* and *Jaco* manipulators we obtain six environments.

As step limit for skill execution, we set 10 steps on *Cursor* and 50 steps in *Reacher* and *Jaco* environments.

We ensure disjointness of board configurations used for training and testing through a hashing algorithm (see Appendix D.1.4 for more details).

Solution depth We quantify the difficulty of a particular board configuration by the number of moves required to solve the game (the *solution depth*). To find board configurations with prescribed solution depth, we employ a breadth-first search (BFS) beginning from the goal board configuration (all fields *off* in *LightsOut*, ordered fields in *TileSwap*). Board configurations are expanded through applying feasible actions (12 for *TileSwap*, 25 for *LightsOut*). Once a new board configuration is observed for the first time, its solution depth corresponds to the current BFS step. By this, we find all feasible board configurations for *LightsOut* and *TileSwap*, and their corresponding solution depths (see Table 7.1). In Table 7.2 we show the sizes of the training and test split for *LightsOut* and *TileSwap* environments for solution depths in $\{1, \ldots, 5\}$.

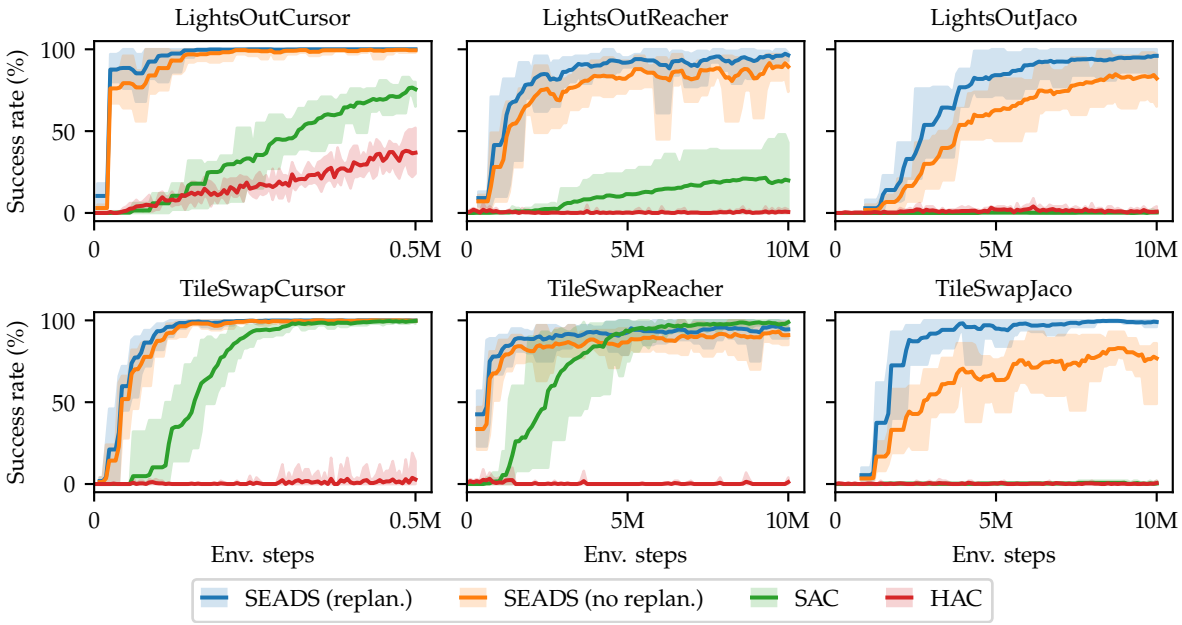


Figure 7.6.: Quantitative evaluation of SEADS. Success rate of the proposed SEADS agent and baseline methods on *LightsOut*, *TileSwap* games embedded in *Cursor*, *Reacher*, *Jaco* environments. SEADS performs comparably or outperforms the baselines on all tasks. The solid line depicts the mean, shaded area min. and max. of 10 (SEADS, SAC on *Cursor*) / 5 (HAC, SAC on *Reacher*, *Jaco*) independently trained agents.

Research questions With our experiments we aim at answering the following research questions:

1. How well does the SEADS agent perform on the proposed physically embedded single-player games, compared to baseline agents? See Subsection 7.4.2.
2. Are the learned skills interpretable? What trajectories do they follow? How many steps do they take before terminating? See Subsection 7.4.3.
3. How many steps are required in the environments to solve a game of particular solution depth? See Subsection 7.4.4.
4. How many unique skills are detected by SEADS? See Subsection 7.4.5.
5. What is the influence of the design decisions (e.g., number of skills K) on the performance of SEADS? See Subsection 7.4.6.
6. How well does SEADS perform on environments with additional challenges (Subsection 7.4.7), including a real-world setup (Subsection 7.4.8)?
7. What are the limitations of SEADS with respect to the games' solution depths (Subsection 7.4.9)?

We approach these questions in the following subsections.

7.4.2. Task performance evaluation

To evaluate the task performance of our agent and baseline agents, we initialize the environments such that the underlying board game requires at maximum 5 moves (pushes in *LightsOut*, swaps

in *TileSwap*) to be solved.[†]

We evaluate each agent on 20 examples for each number of moves in $\{1, \ldots, 5\}$ required to solve the game. We consider a task to be successfully solved if the target board configuration was reached (all fields *off* in *LightsOut*, ordered field in *TileSwap*). For SEADS, we additionally count tasks as "failed" if planning exceeds a wall time limit of 60 seconds. We evaluate both planning variants, with and without replanning.

As an instance of a flat (non-hierarchical) agent we evaluate the performance of Soft Actor-Critic (SAC, Haarnoja et al. (2018)). The SAC agent receives the full environment state $\boldsymbol{s} \in \mathcal{S}$ which includes the symbolic observation (board state). It obtains a reward of 1 if it successfully solved the game and 0 otherwise.

In contrast to the Soft Actor-Critic agent, the SEADS agent leverages the decomposition of state $\boldsymbol{s} \in \mathcal{S}$ and symbolic observation $\boldsymbol{z} \in \mathcal{Z}$. For a fair comparison to a hierarchical agent, we consider Hierarchical Actor-Critic (HAC, Levy et al. (2019)), which, similar to SEADS, can also leverage the decomposition of $\boldsymbol{s}$ and $\boldsymbol{z}$. We employ a two-level hierarchy, in which the high-level policy sets *symbolic* subgoals $\boldsymbol{z} \in \mathcal{Z}$ to the low-level policy, thereby leveraging the access to the symbolic observation.

We refer to Appendices D.5 and D.6 for implementation details on the SAC and HAC baselines, respectively.

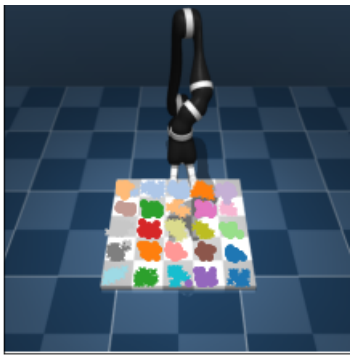
Figure 7.6 visualizes the performance of SEADS and the baselines. On all environments, SEADS performs similar or outperforms the baselines, with the performance difference being most pronounced on the *Jaco* environments, on which SAC and HAC do not make any progress. On the *Cursor* environments, SEADS achieves a success rate of 100%. On the remaining environments, the average success rate (with replanning) is 95.8% (*LightsOutReacher*), 95.5% (*TileSwapReacher*), 94.9% (*LightsOutJaco*), 98.8% (*TileSwapJaco*). As the number of environment steps varies per model checkpoint due to varying skill lengths, the planning performance results are determined at environment steps strictly below 500k (*Cursor*) / 10M (*Reacher*, *Jaco*) steps for which, for every seed, at least one checkpoint exists with the same or higher number of environment steps.

7.4.3. Skill trajectories and -lengths

Contact points and skill trajectories We observe that SEADS learns *game moves* as distinct skills. Exemplarily, on the *LightsOutJaco* and *LightsOutReacher* environments, different skills relate to pushing different fields on the *LightsOut* game board (see Figure 7.7 for a visualization of the end-effector's contact points). In Figure 7.8 we provide a visualization of skill trajectories for the *LightsOutJaco* and *TileSwapJaco* environments. Skill trajectories for the *Cursor* and *Reacher* environments can be found in Appendix D.3.

Skill lengths We report the distribution of skill lengths (i.e., number of actions applied to the manipulator per skill) for successfully solved board instances in Figure 7.9.

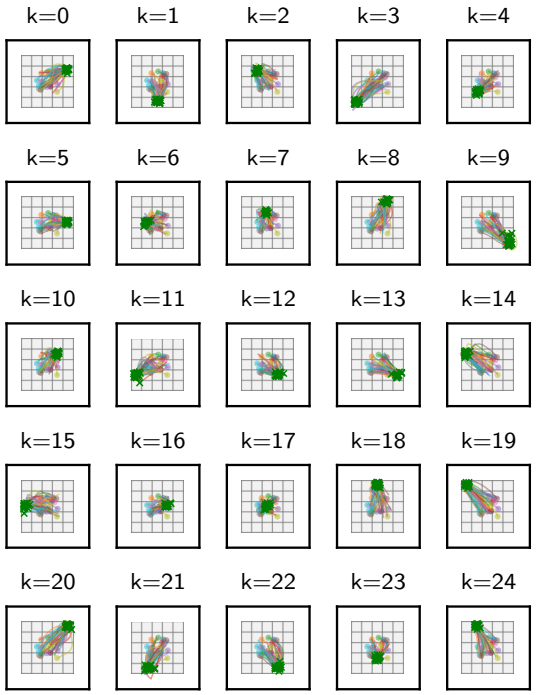
[†] We limit the solution depth to 5 here due to the computational complexity of breadth-first search in the symbolic domain for larger solution depths. We investigate larger solution depths in Subsection 7.4.9.



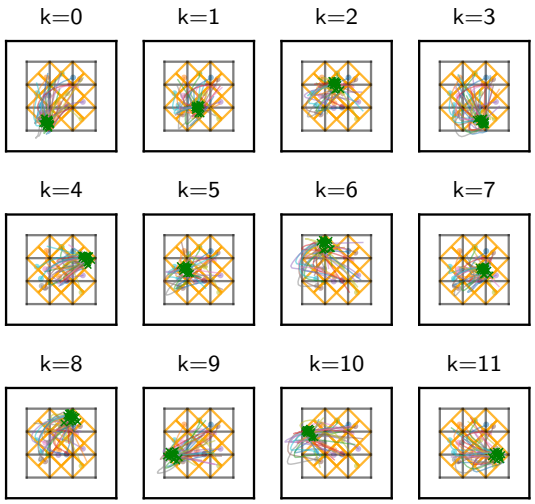
(a) Contact points of *Jaco* end effector.

(b) Contact points of *Reacher* end effector.

Figure 7.7.: Contact points of the *Jaco* (*Reacher*) end effector in the *LightsOutCursor* (*LightsOutReacher*) environments when executing skill $k \in \{1, \ldots, 25\}$ on 20 different initializations of the environment. Each skill is assigned a unique color/marker combination. We show the agent performance after 1×10^7 environment steps. We observe that the SEADS agent learns to push individual fields as skills.



(a) Skill trajectories on *LightsOutJaco*.



(b) Skill trajectories on *TileSwapJaco*.

Figure 7.8.: Trajectories in *Jaco*-embedded environments for skills k on 20 different environment initializations. Colored lines show the x, y-coordinates of the Jaco hand, with the circular marker indicating the start position of the skill. Green markers indicate contact locations with the board.

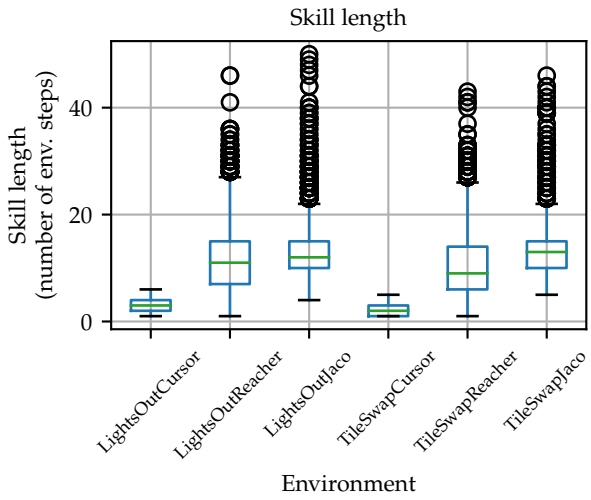


Figure 7.9.: Analysis of skill lengths (number of manipulator steps executed within a single skill) for different environments. Box-plots show the 25%/75% quartiles and median (green line). Whiskers extend to the farthest datapoint within the 1.5-fold interquartile range. Outliers are plotted as circles.

While skill executions on the *Cursor* environments are typically short (median 3/2 manipulator actions for *LightsOutCursor*/*TileSwapCursor*), the *Reacher* and *Jaco* environments require a higher number of manipulator actions per skill (median 11/12/9/13 for *LightsOutReacher*/*LightsOutJaco*/*TileSwapReacher*/*TileSwapJaco*).

7.4.4. Solution length

In Figure 7.10 we provide an analysis how many low-level environment steps (i.e., manipulator actions) are executed to solve instances of the presented physically embedded board games. We show results for 10 trained agents and 20 initial board configurations for each solution depth in $\{1, \ldots, 5\}$. We only report results on board configurations which are successfully solved by SEADS. We observe that even for a solution depth of 5, for the Cursor environments, only relatively few environment steps are required in total to solve the board game ($\approx$ 15 for *LightsOutCursor*, $\approx$ 10 for *TileSwapCursor*). In contrast, the more complex *Reacher* and *Jaco* environments require significantly more interactions to be solved, with up to 400 steps executed on *TileSwapJaco* for a solution depth of 5 and enabled re-planning.

7.4.5. Skill detection performance

In this subsection, we investigate how many distinct skills are learned by SEADS. If not all possible moves within the board games are learned as skills (25 for *LightsOut*, 12 for *TileSwap*), some initial configurations can become unsolvable for the agent, negatively impacting task performance. To count the number of learned skills we apply each skill $k \in \{1, \ldots, K\}$ on a fixed initial state $\boldsymbol{s}_0$ of the environment E until termination (i.e., apply$(E, \boldsymbol{s}_0, \pi, k)$). Among these K skill executions we count the number of unique game moves being triggered. We report the average number of unique game moves for $N = 100$ distinct initial states $\boldsymbol{s}_0$, after training on 5×10^5 (*Cursor*) / 1×10^7 (*Reacher*, *Jaco*) environment interactions. On the *Cursor* environments SEADS detects nearly all possible game moves (in average, 24.9 of 25 possible in *LightsOutCursor*, 12 of 12 in *TileSwapCursor*). For *Reacher* almost all moves are found (24.3/11.8). In

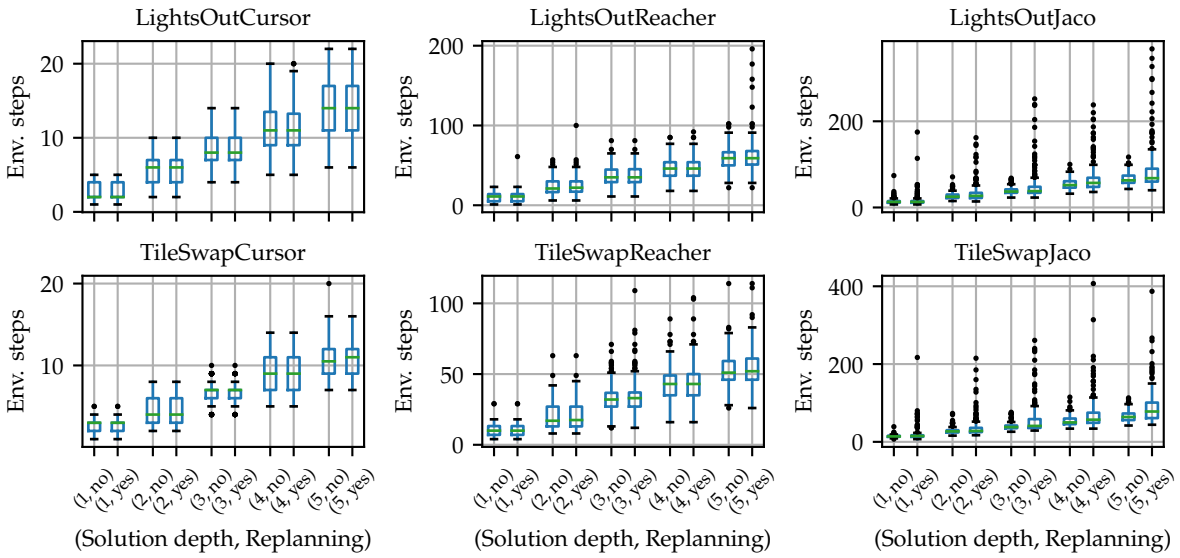


Figure 7.10.: Analysis of solution lengths (total number of manipulator steps required to solve the physically embedded board games) for different environments, solution depths and planning with/without re-planning. Box-plots show the 25%/75% quartiles and median (green line). Whiskers extend to the farthest datapoint within the 1.5-fold interquartile range. Outliers are plotted as circles.

the *Jaco* environments some moves are missing occasionally (23.6/11.5). We visualize the results, including ablations (see Subsection 7.4.6 for details), in Figure 7.11. We demonstrate superior performance compared to a baseline skill discovery method (Variational Intrinsic Control, Gregor et al. (2017)) in Appendix D.7.

7.4.6. Ablation study

We substantiate our agent design decisions through an ablation study, in which we compare the number of unique skills (game moves) detected for several variants of SEADS. For the ablation study, we remove parts from our agent to quantify their impact on performance. This includes training SEADS without the proposed *relabelling*, *second-best normalization* and *novelty bonus*. We visualize the results of this study in Figure 7.11 and provide numerical results in Table 7.3. We found all of these innovations to be important for the performance of SEADS, with the difference to the full SEADS agent being most prominent in the *LightsOutJaco* environment. We also observe that the “More skills" variant (equivalent to SEADS, but with $K = 30$ for *LightsOut*, $K = 15$ for *TileSwap*) yields a similar number of detected unique game moves as SEADS, which is an encouraging result, justifying to over-estimate the number of skills K in situations where it is unknown. We conducted additional significance tests (see below paragraph for details), after which we can state that we did not find any ablation to perform significantly better than SEADS, nor SEADS to perform significantly worse than any ablation (including the “More skills" ablation).

Significance test We perform one-sided Mann-Whitney U tests (Mann and Whitney, 1947) to conclude about significance of our results. On each environment, for each of the 10 independently trained SEADS agents, we obtain a set of 10 samples on the average number of detected skills. Analogously, we obtain such a set of 10 samples for every ablation. We aim at finding ablations which either (i) detect significantly more skills than SEADS or (ii) detect significantly fewer skills

Table 7.3.: Number of average unique game moves detected by SEADS and its ablations, with mean and standard deviation on 10 independently trained agents. Ablations which perform significantly worse than SEADS are colored red. We did not find any ablation to perform significantly better than SEADS, nor SEADS to perform significantly worse than any ablation (including the "More skills" ablation). We refer to Subsection 7.4.6 for details.

	Cursor		Reacher		Jaco	
	LightsOut	TileSwap	LightsOut	TileSwap	LightsOut	TileSwap
SEADS	24.94 ± 0.06	11.99 ± 0.02	24.3 ± 0.28	11.81 ± 0.13	23.64 ± 1.04	11.54 ± 0.27
No sec.-best norm.	24.74 ± 0.42	11.98 ± 0.03	24.42 ± 0.32	11.78 ± 0.11	22.28 ± 0.92	9.26 ± 1.17
No nov. bonus	24.83 ± 0.32	11.99 ± 0.03	23.75 ± 0.71	11.81 ± 0.09	20.24 ± 1.46	11.58 ± 0.29
No relab.	24.72 ± 0.39	12.0 ± 0.02	16.58 ± 4.62	3.62 ± 3.2	19.26 ± 1.0	11.53 ± 0.12
No forw. mod. relab.	24.9 ± 0.07	11.99 ± 0.02	22.73 ± 0.94	11.77 ± 0.28	11.5 ± 3.47	8.64 ± 1.83
No SAC relabelling	24.88 ± 0.09	11.98 ± 0.03	22.94 ± 1.51	11.76 ± 0.15	17.32 ± 2.54	11.05 ± 0.48
More skills	24.97 ± 0.03	11.99 ± 0.02	24.36 ± 0.36	11.8 ± 0.12	23.9 ± 0.47	11.47 ± 0.41

than SEADS on a particular environment. We reject null hypotheses for $p < 0.01$. For (i), we first set up the null hypothesis that the distribution underlying the SEADS samples is stochastically greater or equal to the distribution underlying the ablation samples. For ablations on which this null hypothesis can be rejected it holds that they detect significantly more skills than SEADS. *As we cannot reject the null hypothesis for any ablation, no ablation exists which detects significantly more skills than SEADS.* For (ii), we set up the null hypothesis that the distribution underlying the SEADS samples is stochastically less or equal to the distribution underlying the ablation samples. *For all ablations there exists at least one environment in which we can reject the null hypothesis to (ii), indicating that all ablations contribute significantly to the performance of SEADS on at least one environment*. The results of the significance test are highlighted in Table 7.3.

7.4.7. Environments with additional challenges

In this subsection, we investigate environments posing additional challenges to the SEADS agent.

LightsOutJaco with a 3D board In addition to the *LightsOutJaco* environment presented in Subsection 7.4.1, we introduce an additional environment *LightsOut3DJaco*. While in *LightsOutJaco* the LightsOut board is a flat plane, in *LightsOut3DJaco* the fields are elevated/recessed depending on their distance to the board's center (see Figure 7.12a). This poses an additional challenge to the agent, as it has to avoid to push fields with its fingers accidentally during skill execution. Despite the increased complexity of *LightsOut3DJaco* over *LightsOutJaco*, we observe similar results in terms of detected game moves (Figure 7.12b) and task performance (Figure 7.12c). Both environments can be solved with a high success rate of 94.9% (*LightsOutJaco*) and 96.3% (*LightsOut3DJaco*). We follow the same evaluation protocols as in Subsection 7.4.2 and Subsection 7.4.5.

LightsOutBoard with more fields and spacing In this experiment, we investigate how well SEADS performs on environments in which a very large number of skills has to be learned, and where noisy executions of already learned skills uncover new skills with low probability. To this end, we modified the *LightsOutCursor* environment to have more fields (and thereby, more skills to be learned), and introduced a spacing between the tiles, which makes detecting new skills

more challenging. For a fair comparison, we keep the total actionable area in all environments constant, which introduces an empty area either around the board or around the tiles (see Figure 7.13(a-e)). We make two main observations: First, as presumed, learning skills in the *LightsOutCursor* environment with spacing between tiles requires more environment interactions than for adjacent tiles (see Figure 7.13f). Second, for boards up to size 9×9, a large majority of skills is found after 1.5 million environment steps of training ($5 \times 5 : 24.9/25$, $7 \times 7 : 48.6/49$, $9 \times 9 : 76.8/81$). The *LightsOutCursor* environment with boardsize 13×13 poses a challenge to SEADS with 119.3/169 skills detected after 1.5 million environment steps (see Figure 7.13). The numbers reported are averages over 5 independently trained agents.

7.4.8. Robot experiment

To evaluate the applicability of our approach on a real-world robotic system, we set up a testbed with a uArm Swift Pro robotic arm which interacts with a tablet using a capacitive pen (see Figure 7.15a). The SEADS agent commands a displacement $|\Delta x|, |\Delta y| \leq 0.2$ and an optional pushing command as in the *Cursor* environments. The board state is communicated to the agent through the tablet's USB interface. We manually reset the board *once* at the beginning of training, and do not interfere in the further process. After training for $\approx 160k$ interactions (≈ 43.5 hours) the agent successfully solves all boards in a test set of 25 board configurations (5 per solution depth in $\{1, \ldots, 5\}$). We refer to Figure 7.15b for a visualization of the success rate of SEADS over the course of training and to Figure 7.14 for a visualization of skills learned after $\approx 220k$ environment interactions.‡ Please see Appendix D.4 for more details on the robot experiment setup.

7.4.9. Large solution depth analysis

In Figure 7.16 we present an analysis for solving *LightsOut* tasks with solution depths > 5 using the learned SEADS agent on *LightsOutCursor*. We observe a high mean success rate of $\geq 98\%$ for solution depths ≤ 8. However, the time required for the breadth-first search (BFS) planner to find a feasible plan increases from $\approx 2.02s$ for solution depth 5 to $\approx 301.8s$ for solution depth 9. We abort BFS planning if the list of nodes to expand exceeds a size of ≈ 16 GB memory usage. All experiments were conducted on Intel® Xeon® Gold 5220 CPUs with a clock rate of 2.20 GHz.

‡ We refer to the project page at `https://seads.is.tue.mpg.de/` for a video on the real robot experiment.

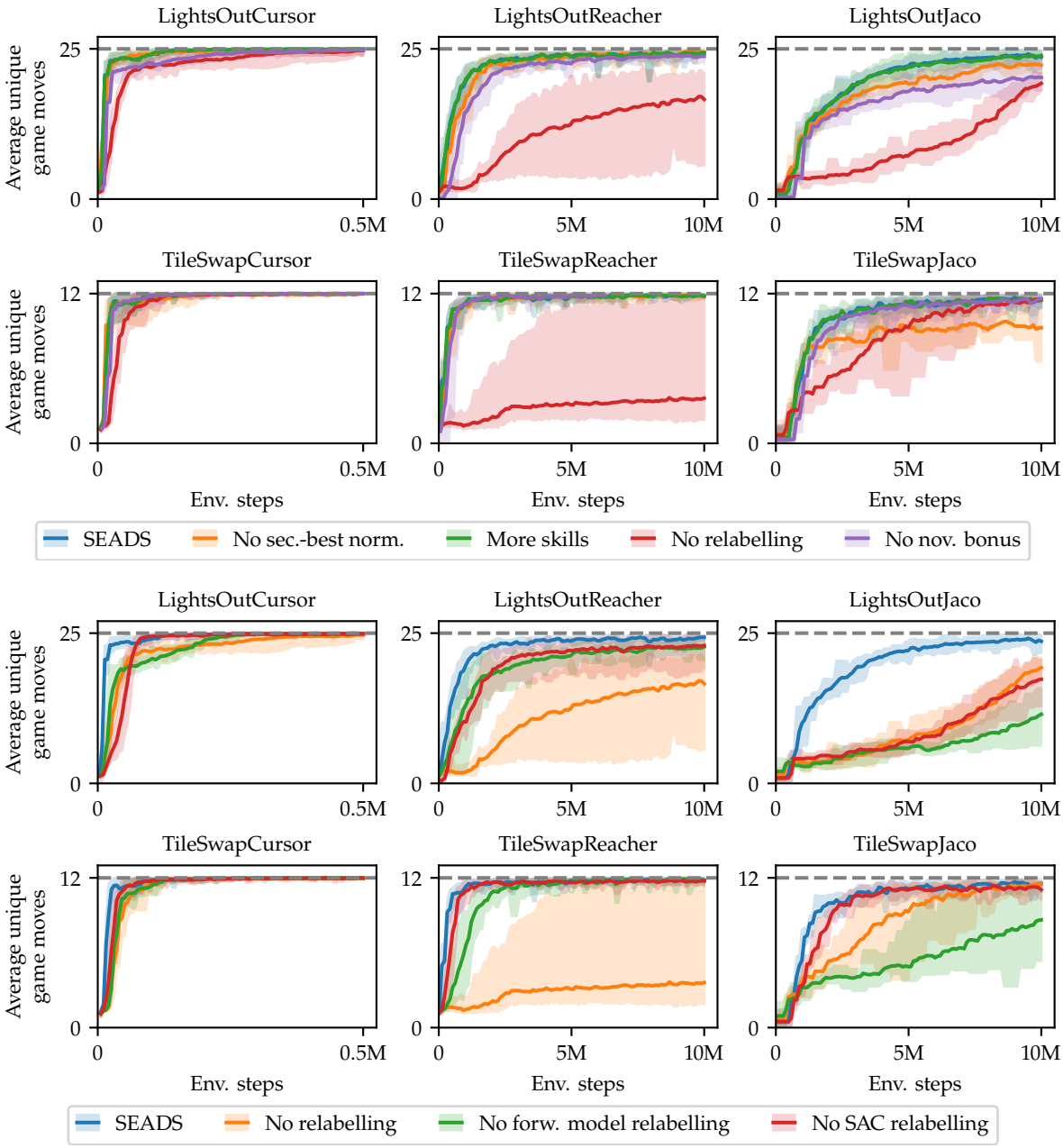
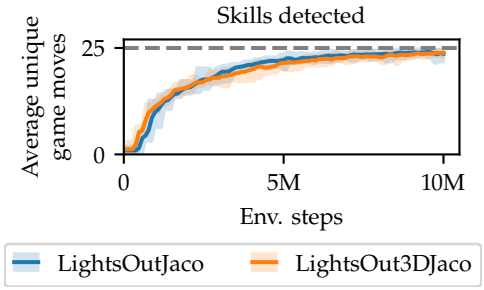
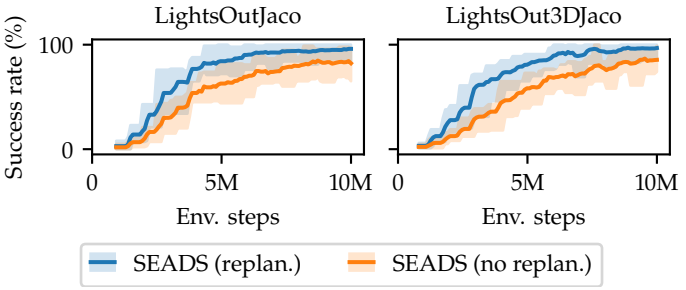


Figure 7.11.: Skill detection performance of SEADS. Top row: Number of learned unique game moves for ablations of SEADS. The solid line depicts the mean, shaded area min. and max. of 10 independently trained agents. Bottom row: Relabelling ablation analysis. Relabelling both for the forward model and SAC agent training are important for the performance of SEADS.

(a) *LightsOut3DJaco*: A variant of the *LightsOutJaco* environment with elevated fields. In comparison to the *LightsOutJaco* environment this environment poses an additional challenge to the agent, as it has to avoid to push fields with its fingers accidentally during skill execution. We show the execution of a skill which has learned to push the center field for $T = \{0, 4, 8, 12, 14\}$.

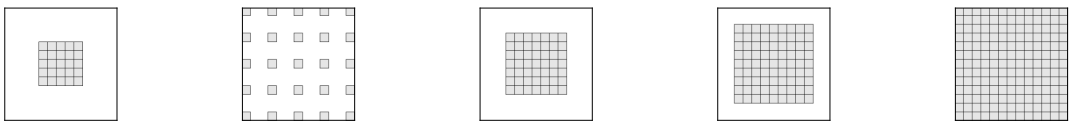


(b) Average number of unique game moves detected on *LightsOutJaco* and *LightsOut3DJaco*.

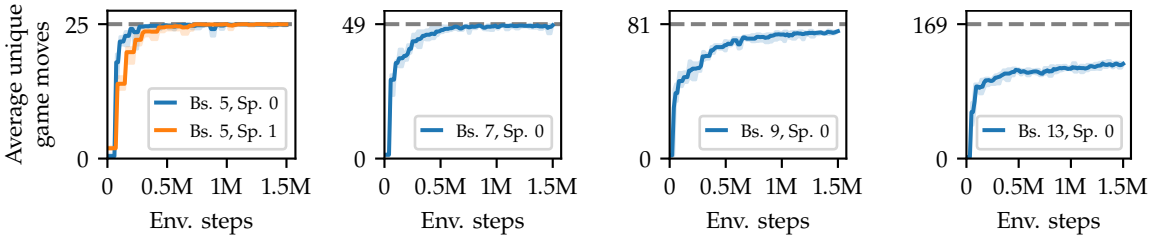


(c) SEADS task performance on *LightsOutJaco* and *LightsOut3DJaco*, with and without replanning.

Figure 7.12.: Evaluation on number of (a) detected game moves and (b) task performance on *LightsOut3DJaco* (see Subsection 7.4.7 for details).



(a) 5×5, no spacing **(b)** 5×5, with spacing **(c)** 7×7, no spacing **(d)** 9×9, no spacing **(e)** 13×13, no spacing



(f) Detected moves, 5×5 **(g)** Detected moves, 7×7 **(h)** Detected moves, 9×9 **(i)** Detected moves, 13×13

Figure 7.13.: Detected average unique game moves (skills) on *LightsOutCursor* environment for different board sizes (Bs.) (5×5, 7×7, 9×9, 13×13) and spacing (Sp.) (boards visualized in (a-e)). The introduced spacing slows down skill learning (f). While for boards until size 9×9 nearly all skills are found (f-h), the 13×13 environment poses a challenge to SEADS (i). Solid line: mean / Shaded area: min/max over 5 independently trained agents.

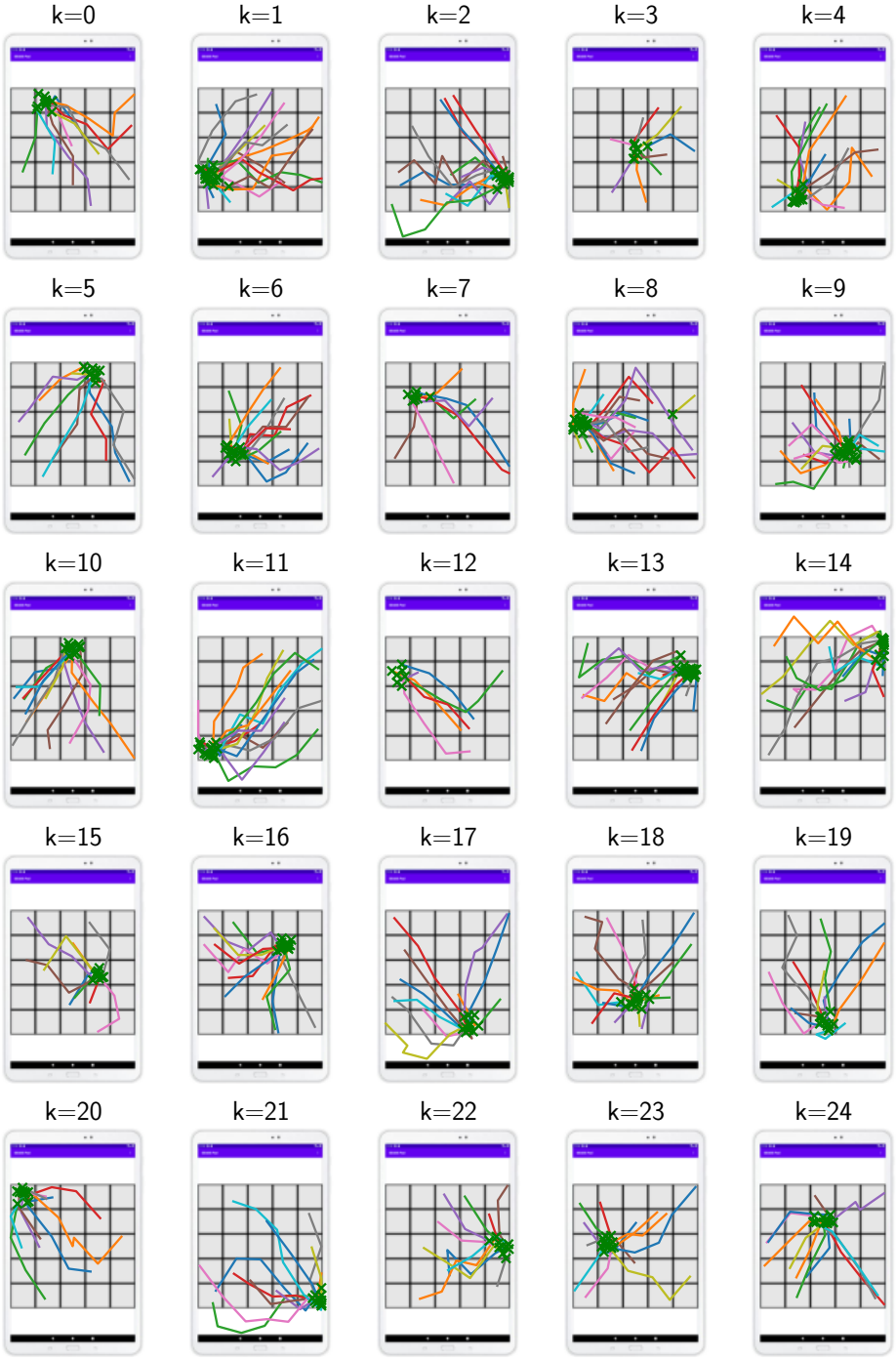
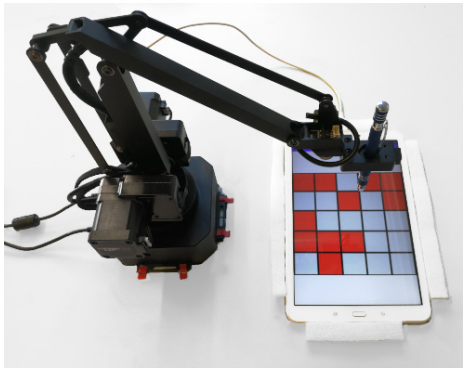
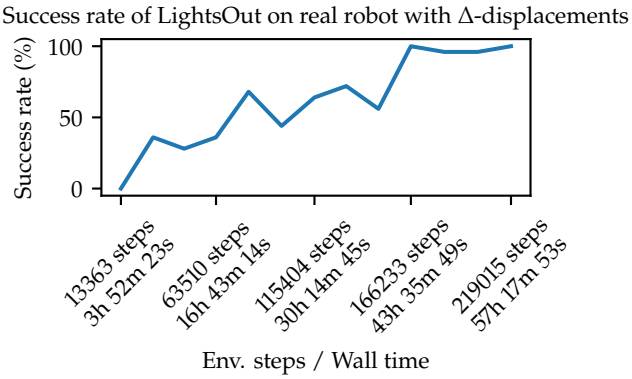


Figure 7.14.: Visualization of 320 trajectories executed on the uArm Swift robotic arm with positional displacement actions, after the agent has been trained for $\approx 220k$ environment interactions. Each subpanel shows trajectories for a specific symbolic action k. Green markers indicate push locations. Similar to the other environments, SEADS has learned to push individual fields as skills.



(a) Robot setup.



(b) Task success rate during the course of training SEADS on the *LightsOut* game embedded into a physical manipulation scenario with the uArm Swift Pro robotic arm. We evaluate the task performance as success rate on 25 board configurations for each reported step (5 instances per solution depth in $\{1, \ldots, 5\}$).

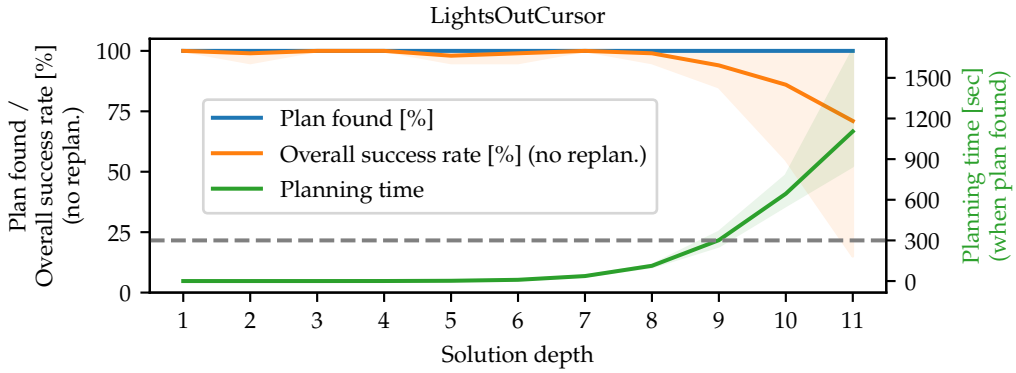


Figure 7.16.: Analysis of the *LightsOutCursor* environment for solution depths > 5. “Plan found" refers to the ratio of problem instances where BFS finds a feasible plan. “Overall success rate" quantifies the ratio of problem instances for which a feasible plan was found and also successfully executed by the low-level policy. The “Planning time" refers to the wall-time the BFS planner runs, for the cases in which it finds a feasible plan. The solid line refers to the mean, shaded area to min/max over 5 independently trained agents on 20 problem instances for each solution depth.

7.5. Assumptions and limitations

Our approach assumes that the state abstraction is known, the symbolic observation z is provided by the environment, and that the continuous state is fully observable. Learning the state abstraction too is an interesting direction for future research. The breadth-first search planner we use for planning on the symbolic level exhibits scaling issues for large solution depths; e.g., for LightsOut it exceeds a 5-minute threshold for solution depths (number of initial board perturbations) ≥ 9. In future work, more efficient heuristic or probabilistic planners could be explored. Currently, our BFS planner produces plans which are optimal with respect to the number of skills executed. Means for predicting and taking the skill execution cost into account for planning could be pursued in future work. In the more complex environments (*Reacher*, *Jaco*) we observe our agent to not learn all possible skills reliably, in particular for skills for which no transitions exist in the replay buffer. In future work one could integrate additional exploration objectives which incentivize to visit unseen regions of the state space. Also, the approach is currently limited to settings such as in board games, where all symbolic state transitions should be mapped to skills. It is an open research question how our skill exploration objective could be combined with demonstrations or task-specific objectives to guide the search for symbolic actions and limit the search space in more complex environments.

7.6. Conclusion

We present an agent which, in an unsupervised way, learns diverse skills in complex physically embedded board game environments which relate to *moves* in the particular games. We assume a state abstraction from continuous states to symbolic states known and observable to the agent as prior information, and that skills lead to changes in the symbolic state. The jointly learned forward model captures the temporally extended *effects* of skill execution. We leverage this forward model to plan over a sequence of skills (moves) to solve a particular task, i.e., bring the game board to a desired state. We demonstrate that with this formulation we can solve complex physically embedded games with high success rate, that our approach compares favorably with other flat and hierarchical RL algorithms, and also transfers to a real robot. Our approach provides an unsupervised learning alternative to prescribing the action abstraction and pretraining each skill individually before learning a forward model from skill executions. In future research, our approach could be combined with state abstraction learning to leverage its full potential.

8. Summary & Outlook

8.1. Summary

In this thesis, we present approaches to challenges which affect model-based sequential decision making agents, particularly in real-world environments.

In Chapter 1, we outlined the necessity of real-world agents to adapt to changes in the environment's dynamics. This has motivated us to develop a novel, adaptive dynamics model based on Gaussian process regression in the latent space of a variational auto-encoder, termed *Deep Latent Gaussian Process Dynamics* (DLGPD, Chapter 3). The method allows decision making directly from high-dimensional data such as images, which is an active area of contemporary research (Ha and Schmidhuber, 2018; Hafner et al., 2020, 2019; Yarats et al., 2021). Differently to the aforementioned approaches, our method can systematically handle changes in the environment's dynamics. We demonstrate on a Pendulum environment that our approach requires less additional data than the purely deep-learning based approach PlaNet (Hafner et al., 2019) to adapt to changes in the environment's dynamics.

Another important aspect of learning-based real-world decision making, as outlined in Chapter 1, is sample efficiency — collecting data in real-world environments typically incurs significant costs. In environments which are subject to variations, this does not only create the need for adaptive dynamics models, but also for interacting with the environment in a way which allows to quickly identify the dynamics of a particular environment. This can be formulated as a Bayesian Optimal Experimental Design problem (Chaloner and Verdinelli, 1995), as we outlined in Chapter 4. In Chapter 4, we formulated an adaptive dynamics model based on the Neural Process (Garnelo et al., 2018b) framework, which captures environment variations in a global latent variable. Through data collection, we aim to maximize the Expected Information Gain (Lindley, 1956), effectively minimizing the uncertainty in the global latent variable. We show that we can compute action sequences which allow to efficiently identify the dynamics of environments underlying diverse variations, in contrast to randomly sampled actions.

In the area of terrain-aware navigation, not only intrinsic parameters of the robot are subject to change, e.g., actuator gains, but also the terrain typically varies depending on the location of the robot. In Chapter 5, we presented our approach *TRADYN* which can cope with both types of variations, based on the adaptive dynamics model presented in Achterhold and Stueckler (2021) (Chapter 4). We evaluated the adaptation capabilities of our proposed approach on a simulated unicycle-like robot, and show the importance of being able to adapt to varying robot parameters and terrain properties.

In all approaches discussed so far, physical prior knowledge has only played a minor rule in the model design process. Exemplarily, in Chapter 3, a smoothness prior is posed on the latent dynamics through the use of a squared-exponential covariance function. Such kinds of models are typically referred to as black-box models, in contrast to gray-box models, in which physical prior knowledge is more explicitly incorporated (Nelles, 2001). Constraining the model class through physical prior knowledge poses the risk of the model not accurately capturing the actual dynamics, e.g., through unmodeled effects. On the other hand, strong regularization

through physical prior knowledge can decrease the amount of training data required to learn a well-generalizing model. In Chapter 6, we leveraged physical prior knowledge to learn a model which allows for filtering and predicting the motion of a table tennis ball for robotic return. With this gray-box approach, we achieve state-of-the-art predictive accuracy, outperforming black-box baselines.

In Chapter 7, we focus on two further challenges of decision making in real-world systems. First, in real-world decision making, some systems, such as robotic arms, require reactive, high-frequency control. With model-based planning methods, this can be more challenging to achieve than with model-free methods (Hafner et al., 2020; Pinneri et al., 2020). A second challenge in sequential decision making we focus on in Chapter 7 is the credit assignment problem: The effect of an action might become apparent only far in the future. One prime example of environments which are subject to both challenges are *physically embedded board games* (Mirza et al., 2020), in which a robotic manipulator is supposed to play a board game. A positive reward signal is only given when the game is won, although this signal depends on actions applied far in the past. Additionally, the robotic manipulator has to be controlled at a high frequency. In Chapter 7, we designed a hierarchical agent termed *SEADS*, which combines advantages from model-based and model-free methods with intrinsic motivation to autonomously learn *skills* which cause diverse and predictable transitions in an abstraction of the environment's state. We show in Chapter 7 that our agent outperforms other flat and hierarchical agents on complex embedded board games.

In conclusion, this thesis presents several approaches which advance the state-of-the-art in several directions linked to model-based decision-making. This includes adaptive dynamics models and their use for active system identification and terrain- and robot-aware navigation, a state-of-the-art approach for predicting table tennis ball trajectories, and a novel hierarchical agent combining the advantages of model-free and model-based methods. With these advances, this thesis paves the way for potential future work in a variety of directions.

8.2. Future work

8.2.1. Real-world experiments

With our contributions DLGPD (Chapter 3), Explore the Context (Chapter 4), and TRADYN (Chapter 5) we have taken a step towards sequential decision making agents which can cope with problems being apparent in the real world. This mainly concerns the adaptivity of dynamics models, active system identification, and robot- and terrain aware navigation. However, we have made some simplifying assumptions in the development of these approaches, which do not necessarily hold in real-world environments. We will comment on them and potential remedies in the following paragraphs.

Gaussian process latent variable models from images While we have shown successful application of the adaptive, Gaussian process based dynamics model presented in Chapter 3 to a system with relatively simple dynamics (inverted pendulum), we found it challenging to model complex systems. This is due to the computational complexity of Gaussian process regression. One potential direction of future work is to use the latent variable model presented

in Sæmundsson et al. (2018). It uses pseudo-inputs (Snelson and Ghahramani, 2005) to reduce computational complexity. The model by Sæmundsson et al. (2018) could be used as a dynamics model in the latent space, with jointly learned image encoders and decoders, as proposed in Chapter 3. This, however, would require learning the dynamics model on a family of environments underlying the variations to consider.

Partially observable environments and adaptive dynamics models The formulations in Chapter 4 assume full observability of the environment, which does not hold in general in real-world environments. In the case of partial observability, a belief state has to be formed from past observations. In this case, however, not all factors which explain the variations in the environments might be captured by the global latent variable. Instead, only factors which can *not* be inferred from the set of past observations might be captured. While this is not problematic in principle, it makes interpreting the global latent variable more challenging.

Learning a map for terrain-aware robot navigation For the terrain- and robot-aware navigation approach presented in Chapter 5, the same considerations regarding partial observability hold as for Chapter 4, as it uses the same dynamics model. In addition, we assume the environment terrain map to be known a-priori. To relax this assumption, in future work, the approach presented in Chapter 5 could be combined with (visual) simultaneous localization and mapping (SLAM) to learn the map (see, e.g., Macario Barros et al. (2022)). A further interesting direction for future work, leveraging the ideas of active inference presented in Chapter 4, could be to extend this further to *active SLAM* (e.g., surveyed by Placed et al. (2023)). In active SLAM, the idea is that the robot takes actions in order to obtain informative observations about the map.

8.2.2. Simulation-to-real transfer with adaptive dynamics models

A concrete application example of the approach presented in Chapter 4, which might be worth investigating in future work, is that of simulation-to-real (Sim2Real) transfer. Dynamics models learned from data generated by a simulator face the challenge that the *real* system may behave slightly differently than the simulation, making the dynamics model less accurate. The approach presented in Chapter 4 could be used to tackle this challenge. First, an adaptive dynamics model is learned in simulation. Subsequently, with the approach presented in Chapter 4, experience is collected in the real environment, in order to calibrate the adaptive dynamics model. Marco et al. (2017) similarly propose combining data from simulation and the real world in an active learning scheme based on Bayesian optimization. However, their objective is to tune the parameters of a linear quadratic regulator in order to minimize a control cost function, instead of learning a dynamics model.

8.2.3. Visual and auditory stroke cues for table tennis trajectory ball prediction

In Chapter 6, we demonstrate that we can learn a reasonable mapping from the ball launcher parameters to the initial spin of the ball. Several studies indicate that human players also leverage additional cues to estimate the table tennis ball spin, such as visual and auditory cues (Klein-Soetebier et al., 2020; Zhao et al., 2018). An interesting direction of future work could be to replace the ball launcher information by such cues, e.g., by estimating the racket movement

through inertial measurement data (Blank et al., 2017) or video (Gao et al., 2021), human pose estimation (Kulkarni and Shenoy, 2021), or auditory cues. Additionally, ball pose measurements, e.g., by detecting the brand logo (Y. Zhang et al., 2015) or other patterns (Gossard et al., 2023), could be included.

8.2.4. Jointly learning skills and state abstractions

A strong assumption made in Chapter 7 is that the state abstraction function $\Phi(s)$ is known. *Learning* the state abstraction function, e.g., by an auto-encoder with a symbolic latent space, as in Ahmetoglu et al. (2022), is an interesting direction for future research. However, learning task-agnostic diverse and predictable skills jointly with their corresponding abstraction is an ill-posed problem without further regularization. In the example of physically embedded board games, the learned state abstraction might include the state of the manipulator. In this case, a diverse set of skills could just move the manipulator in distinct ways, without performing actual game moves (e.g., pushing fields on the LightsOut board), making the actual game unsolvable. Therefore, further regularization needs to be included, such as task rewards or slowness priors (S. Li et al., 2021) on the state abstraction. Instead of learning an abstraction from scratch, leveraging separately trained models providing annotations, e.g., in the form of scene graphs (see Chang et al. (2023) for a survey), might also help to mitigate the ill-posedness.

8.2.5. Learning diverse skills from offline data

Learning from offline data for sequential decision making has become an active research topic over the last years (Levine et al., 2020), also in the area of learning skills (Chebotar et al., 2021; Rosete-Beas et al., 2022). It promises to be able to leverage the vast amount of data being available on the internet, and thereby reduce the amount of specific interaction required in a particular environment to a minimum. The objective that the effect of temporally extended skills should be maximally diverse and predictive (evaluated under a forward model), as proposed in Chapter 7, might be leveraged to learn a set of skills from offline data. As an abstraction, using language annotations might be worth investigating, e.g., as provided by the dataset by Mees et al. (2022).

Appendix

Appendix: Deep Latent Gaussian Process Dynamics Models

A.

A.1. Signal-to-noise ratio (SNR) regularization

As covariance functions (between targets) for the transition GPs and reward GP we choose radial basis function (RBF) kernels

$$k(x_i, x_j) = \alpha^2 \exp\left(-\frac{1}{2}(x_i - x_j)^\top \Lambda^{-1}(x_i - x_j)\right) + \delta_{ij}\sigma^2 \tag{A.1}$$

with outputscale $\alpha > 0$, additive noise covariance σ^2, characteristic length-scales $\Lambda = \text{diag}([l_1^2, \ldots, l_D^2])$ and indicator function δ_{ij} ($\delta_{ij} = 1$ if $i = j, \delta_{ij} = 0$ otherwise). To improve numerical stability, we regularize the ratio of outputscale and noise covariance of the RBF kernels of the transition GPs (denoted by $\mathcal{K}_{\text{trans}}$) and reward GP (denoted by K_{reward}) by minimizing a signal-to-noise penalty

$$\mathcal{L}_{\text{SNR}} = \sum_{k \in \mathcal{K}_{\text{trans}} \cup K_{\text{reward}}} \left[\frac{\log(\text{SNR}_k)}{\log(\tau)}\right]^p \tag{A.2}$$

with $\text{SNR}_k = \alpha_k / \sigma_k, \tau = 10, p = 8$. A similar regularization can be found in the PILCO implementation (Deisenroth et al., 2013) (with $\tau = 1000, p = 30$). The final loss we minimize is thus

$$\mathcal{L} = -\mathcal{L}_{\text{lower-bound}} + \mathcal{L}_{\text{SNR}} \tag{A.3}$$

with

$$\begin{aligned}\mathcal{L}_{\text{lower-bound}} = \mathbb{E}_{q(S' \mid O')}\left[\log p(O' \mid S')\right] + \mathbb{E}_{q(S' \mid O')}\left[-\log q(S' \mid O')\right] \\ + \mathbb{E}_{q(S' \mid O')q(S \mid O)}\left[\log p(S' \mid S, A)\right] + \mathbb{E}_{q(S \mid O)}\left[\log p(R' \mid S, A)\right]\end{aligned} \tag{A.4}$$

(see Equation 3.16).

A.2. Encoder and decoder architecture

For mapping from observation space to latent space and vice versa, we use (transposed) convolutional neural networks with ReLU non-linearities. The network architecture is similar to the encoder and decoder used in (Hafner et al., 2019). The encoder parametrizes a normal distribution by its mean μ and standard deviation σ.

Table A.1.: Encoder architecture.

Input: 2 channel-wise concatenated RGB images ($6 \times 64 \times 64$)
Conv2D (32 filters, kernel size 4×4, stride 2) + ReLU Conv2D (64 filters, kernel size 4×4, stride 2) + ReLU Conv2D (128 filters, kernel size 4×4, stride 2) + ReLU Conv2D (256 filters, kernel size 4×4, stride 2) + ReLU
μ: Linear ($1024 \rightarrow 3$), σ: Softplus(Linear($1024 \rightarrow 3$) + 0.55) + 0.01

Table A.2.: Decoder architecture.

Input: 3-dimensional latent variable
Linear($3 \rightarrow 1024$) + ReLU ConvTranspose2D (128 output channels, kernel size 5×5, stride 2) + ReLU ConvTranspose2D (64 output channels, kernel size 5×5, stride 2) + ReLU ConvTranspose2D (32 output channels, kernel size 6×6, stride 2) + ReLU ConvTranspose2D (6 output channels, kernel size 6×6, stride 2) + Sigmoid
Output: 2 channel-wise concatenated RGB images ($6 \times 64 \times 64$)

Appendix: Active Inference for Adaptive Dynamics Models

B.

In the following, we provide supplementary details and analysis of the approach presented in Chapter 4.

B.1. Architectural details

In the following, we describe the architectural details of our model. Notation-wise, $[\cdot;\cdot;\ldots]$ denotes a sequence of neural network layers. `Linear`(M, N) indicates a linear layer with M input features and N output features, `ReluLinear`(M, N) is a linear layer with non-negative weights $vy = \text{ReLU}(\boldsymbol{W})\boldsymbol{x} + \boldsymbol{b}$. `ReLU` and `Tanh` represent ReLU and hyperbolic tangent nonlinearities, respectively. `Negate` is a negation of the input features $\boldsymbol{y} = -\boldsymbol{x}$. `SoftplusOffset`$(\gamma)$ symbolizes a softplus nonlinearity with additive offset: $y = \ln(1 + e^x) + \gamma$, which is applied elementwise.

B.1.1. Transition model

The transition model consists of encoders e_s, e_a and e_β to lift state (dimensionality D_S), action (dimensionality D_A) and latent context variable (dimensionality D_β) to an embedding space with dimensionality $D_\text{E} = 200$. A GRU cell (Cho et al., 2014) operates in the embedding space to model the dynamics. The decoders parameterize mean and diagonal covariance on the state space given a propagated embedding. Due to the constant additive noise assumption in the toy problem environment, in those experiments, the diagonal state space covariance of the model is learned as a constant and not modeled by a decoder.

State encoder e_s:
[`Linear`$(D_\text{S}, 200)$; `ReLU`(); `Linear`$(200, D_\text{E})$; `Tanh`()]

Action encoder e_a:
[`Linear`$(D_\text{A}, 200)$; `ReLU`(); `Linear`$(200, D_\text{E})$; `ReLU`()]

Latent context encoder $g_{\boldsymbol{\beta}}$:
[`Linear`$(D_\beta, 200)$; `ReLU`(); `Linear`$(200, D_\text{E})$; `ReLU`()]

GRU cell h_RNN:
GRU cell with input dimension $2 \cdot D_\text{E}$ (concatenation of action and latent context), state dimension D_E.

State decoder (mean) $d_{\text{s},\mu}$:
[`Linear`$(D_\text{E}, 200)$; `ReLU`(); `Linear`$(200, D_\text{S})$]

State decoder (diagonal covariance) d_{s,σ^2}:
[`Linear`$(D_\text{E}, 200)$; `ReLU`(); `Linear`$(200, D_\text{S})$; `SoftplusOffset`(1e^{-4})]

B.1.2. Context encoder

Components of the context encoder are the transition encoder and latent context decoder networks for mean and (diagonal) standard deviation. The dimensionality of the transition embedding space D_{F} is 32 for the toy problem and 128 for all other experiments. In ablation experiments in which we do not enforce the variance of the context encoder to be strictly decreasing with the number of context observations (referred to as "- Decr. variance"), we replace the `ReluLinear` layers by standard `Linear` layers.

Transition encoder:
[`Linear`($2D_{\mathrm{S}} + D_{\mathrm{A}}$, 200); `ReLU`(); `Linear`(200, D_{F}); `ReLU`()]

Latent context decoder (mean):
[`Linear`(D_{F}, 200); `ReLU`(); `Linear`(200, D_{β})]

Latent context decoder (diagonal standard deviation):
[`ReluLinear`(D_{F}, 200); `ReLU`(); `ReluLinear`(200, D_{β}); `Negate`(); `SoftplusOffset`($1e^{-2}$)]

B.2. Training details

Table B.1 gives values for the hyperparameters used for each experiment.

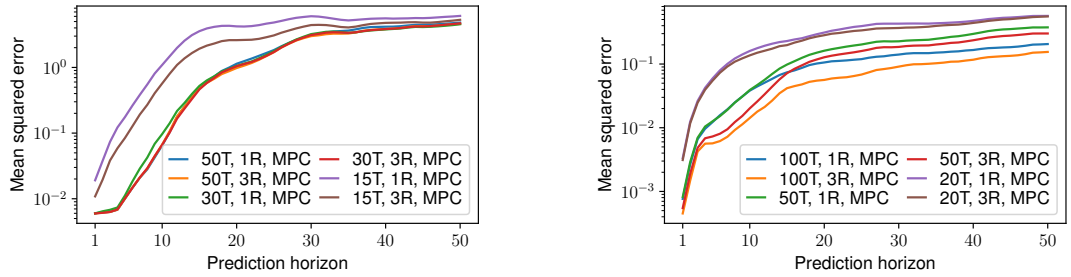
Table B.1.: Hyperparameters for the toy problem, Pendulum and MountainCar experiments.

Parameter	Toy Problem	Pendulum	MountainCar
Number of training steps	50k	100k	100k
Latent context dimensionality D_{β}	1	16	16
Transition embedding space dimensionality D_{F}	32	128	128

B.2.1. Data sampling

As detailed in the respective environment sections, for data collection, we first randomly sample instances of the parameterized toy problem, Pendulum, and MountainCar environments. On each environment instance, we generate two independent rollouts R_A, R_B, each with a randomly sampled initial state and randomly sampled actions. We use rollout pairs from 5k instances of the toy problem, 100k instances of the Pendulum environment and 50k instances of the MountainCar environment as training data. For computing a validation loss during training, we generate rollout pairs from additional 1k toy problem, 10k Pendulum and 10k MountainCar environment samples. Environment instances used to evaluate the performance of the predictive model after calibration do not overlap with instances used for training and validation.

The target chunk D^{α} is a random, contiguous subsequence of length 50 of rollout R_A. Each transition in the context set C^{α} is independently sampled from the rollout R_A with a probability $p_{\mathrm{ctx-from-A}}$ or from the rollout R_B with a probability $p_{\mathrm{ctx-from-B}} = 1 - p_{\mathrm{ctx-from-A}}$. For the toy problem experiments, we fix $p_{\mathrm{ctx-from-A}} = 0$. For the Pendulum and MountainCar experiments, we set $p_{\mathrm{ctx-from-A}} = 0.5$ for the first 30k training steps, then reduce it linearly to $p_{\mathrm{ctx-from-A}} = 0$ until step 60k, and keep it at this value until the end of training. We motivate this scheduling



(a) Pendulum **(b)** MountainCar

Figure B.1.: Prediction error (lower is better) of the learned (a) Pendulum and (b) MountainCar models, for MPC calibration procedures with varying number of transitions per rollout (xT) and varying number of calibration rollouts (xR). Each line represents the mean squared error over 3000 rollouts.

strategy to simplify the learning problem by increasing the average amount of context observations which are informative for the target chunk. When the context set and target chunk are sampled from different rollouts, they may cover disjoint parts of the state space, increasing the average amount of non-informative transitions in the context set.

B.2.2. Validation loss computation

We randomly sample 5 batches from the validation data (with a batchsize of 64 for the toy problem and 512 for Pendulum and MountainCar) to construct a validation dataset (see Appendix B.2.1). For sampling the validation batches, we fix $p_{\text{ctx-from-A}} = 0$. The validation loss is calculated by applying the loss objective used for training (Equation 4.32) on the validation dataset. We report results on models yielding the lowest validation loss within the given number of training steps.

B.3. Planning details

To plan an optimal action sequence for calibration, we use a planning algorithm based on the cross-entropy method (Rubinstein, 1999) (CEM, see Subsection 2.6.1). We set the number of optimization iterations $T = 10$, number of candidates $N_{\text{cand}} = 1000$ and number of elite candidates $N_{\text{elites}} = 100$. The planning horizon is task dependent, during Open-Loop calibration, we use the full calibration horizon as planning horizon ($N = 30$ for the Pendulum, $N = 50$ for the MountainCar). During MPC calibration, the planning horizon is given by H as defined in Subsection 4.4.5.

B.4. Ablation experiment: Number of calibration interactions

As an ablation experiment, we investigate the relation between the model prediction error of a calibrated model and the number of system interactions performed during calibration. To this end, we vary the number of calibration transitions per rollout and the number of calibration rollouts we perform for calibrating a single system. In case of multiple rollouts, we add the transitions of previous calibration rollouts to the set of already observed system transitions $\mathcal{T}_0$ in the MPC calibration scheme.

We vary the number of transitions per rollout as $\{15, 30, 50\}$ transitions for the Pendulum environment and $\{20, 50, 100\}$ transitions for the MountainCar environment. The number of calibration rollouts is selected from $\{1, 3\}$. The results reported in Section 4.5 use a single rollout with 30 transitions for the Pendulum environment and 50 transitions for the MountainCar environment.

See Figure B.1 for a depiction of the prediction error of the calibrated models.

For the Pendulum environment, we observe that short calibration rollouts with 15 transitions yield significantly worse prediction results compared to rollouts of length 30 or 50, even when performing multiple calibration rollouts. With too short rollouts, the calibration sequence can not swing-up the Pendulum to cover all (especially the upper two) quadrants. On the other hand, longer calibration rollouts (with 50 transitions) or more calibration attempts (3 rollouts with 30 transitions) do not yield significantly better results than a single calibration rollout with 30 transitions because all quadrants have already been covered.

For the MountainCar environment, an environment which exhibits more complex dynamics variations than the Pendulum environment, we observe that more calibration data yields models with lower prediction error for short-horizon predictions (< 15 steps). However, for long-horizon predictions, a single long rollout (100 transitions) performs better than 3 short rollouts (50 transitions), although the total amount of calibration transitions is higher in the latter case. We hypothesize that long calibration rollouts accurately explore regions which long system rollouts reach (for long prediction horizons) and thus perform better for the long-horizon case than short calibration rollouts.

Appendix: Table Tennis Ball Trajectory Modeling

C.

In the following, we provide extended results and architectural details of our table tennis ball trajectory filtering and prediction approach. In Appendix C.1 we provide details on the architecture, in particular the initial values for the learned EKF parameters, and the network for estimating the initial spin. Appendix C.2 describes the computation of the Jacobian of the forward model $\boldsymbol{J} = \frac{\partial}{\partial \boldsymbol{z}} g(\boldsymbol{z})$ (cf. Equation 6.22).

C.1. Architectural details

C.1.1. Parameter initialization

In this section, we provide initial values for the parameters

$$\Psi = \left\{ \mathbf{C}, \sigma_q, \sigma_r, \sigma_p, \sigma_v, \boldsymbol{\psi}_f, \sigma_\omega, \sigma_{\omega,\mathrm{ai}}, a_\mathrm{d}, a_\mathrm{m}, \sigma_{a_\mathrm{d}}, \sigma_{a_\mathrm{m}} \right\}$$

we optimize during training.

We initialize the impact matrix $\boldsymbol{C}$ with the assumption that the velocities in x- and y-direction do not change during impact, while the velocity in z direction flips sign. We assume the spin to stay constant in all dimensions. This yields

$$\boldsymbol{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

as initial value for $\boldsymbol{C}$. The parameter $\boldsymbol{\psi}_f$ refers to the parameters of the spin initialization network, which is detailed in Appendix C.1.3. Initial values for the remaining parameters are given in Table C.1.

C.1.2. Learned parameters

In Table C.2, we provide the values of Ψ after training for an exemplary EKF model (trained on the augmented dataset, with providing ball launch information).

C.1.3. Spin initialization network architecture

The neural network for initializing the initial spin in canonical launch direction $\boldsymbol{w}_{\rightarrow x}$ based on actuation parameters $\boldsymbol{s}_m \in [0,1]^3$ is parametrized as follows

$$\boldsymbol{w}_{\rightarrow x,\mu,\sigma} = \boldsymbol{W}_2 \max(\boldsymbol{W}_1 \boldsymbol{s}_m, 0),$$

Table C.1.: Initial values for parameters of the extended Kalman filter. The operator $\text{inv}_{[\cdot]^\oplus}(x)$ inverts the softplus function $[\cdot]^\oplus$, such that, e.g., the initial value for the initial position covariance is $\Sigma_p = \text{diag}([\boldsymbol{\sigma}_p]^\oplus) = \boldsymbol{I}_3 \cdot 10^{-4}$. The operator $[\cdot : \cdot]$ denotes 1-based slicing, including start and end point. E.g., $\boldsymbol{\sigma}_q[1:3]$ refers to the transition variance of the ball's position. Note that $\boldsymbol{\sigma}_\omega, \boldsymbol{\sigma}_{\omega,\text{ai}}$ are only used when the spin initialization network is not used or after an impact has happened. In these cases, we assume the spin to highly vary, which is why we chose the values of $\boldsymbol{\sigma}_\omega, \boldsymbol{\sigma}_{\omega,\text{ai}}$ to be comparably large.

Parameter name	Initial value	Parameter name	Initial value
Initial state variance		Transition variance	
$\boldsymbol{\sigma}_p$	$\text{inv}_{[\cdot]^\oplus}(10^{-4})$	$\boldsymbol{\sigma}_q[1:3]$	$\text{inv}_{[\cdot]^\oplus}(10^{-4})$
$\boldsymbol{\sigma}_v$	$\text{inv}_{[\cdot]^\oplus}(10^{-2})$	$\boldsymbol{\sigma}_q[4:6]$	$\text{inv}_{[\cdot]^\oplus}(10^{-2})$
$\boldsymbol{\sigma}_\omega, \boldsymbol{\sigma}_{\omega,\text{ai}}$	$\text{inv}_{[\cdot]^\oplus}(1)$	$\boldsymbol{\sigma}_q[7:9]$	$\text{inv}_{[\cdot]^\oplus}(10^{-3})$
$\boldsymbol{\sigma}_{a_\text{d}}$	$\text{inv}_{[\cdot]^\oplus}(10^{-2})$	$\boldsymbol{\sigma}_q[10]$	$\text{inv}_{[\cdot]^\oplus}(10^{-2})$
$\boldsymbol{\sigma}_{a_\text{m}}$	$\text{inv}_{[\cdot]^\oplus}(10^{-2})$	$\boldsymbol{\sigma}_q[11]$	$\text{inv}_{[\cdot]^\oplus}(10^{-2})$
$\boldsymbol{\sigma}_r$	$\text{inv}_{[\cdot]^\oplus}(10^{-3})$		
Initial state mean			
a_d	$\sqrt{0.1}$		
a_m	$\sqrt{0.1}$		

Table C.2.: Learned values for parameters of the extended Kalman filter, for a model which leverages launch information (thus, $\boldsymbol{\sigma}_\omega$ is not used).

Parameter name	Initial value	Parameter name	Initial value
Initial state variance		Transition variance	
$[\boldsymbol{\sigma}_p]^+$	$(1.38, 1.63, 1.00)^\top \cdot 10^{-6}$	$[\boldsymbol{\sigma}_q[1:3]]^+$	$(1.00, 1.00, 1.00)^\top \cdot 10^{-6}$
$[\boldsymbol{\sigma}_v]^+$	$(0.12, 0.14, 0.09)^\top$	$[\boldsymbol{\sigma}_q[4:6]]^+$	$(1.16, 1.19, 1.10)^\top \cdot 10^{-6}$
$[\boldsymbol{\sigma}_{\omega,\text{ai}}]^+$	$(0.19, 3.28, 0.12)^\top$	$[\boldsymbol{\sigma}_q[7:9]]^+$	$\begin{pmatrix} 1.73 \cdot 10^{-3} \\ 1.31 \cdot 10^{-3} \\ 1.00 \cdot 10^{-6} \end{pmatrix}$
$[\boldsymbol{\sigma}_{a_\text{d}}]^+$	$5.70 \cdot 10^{-6}$	$[\boldsymbol{\sigma}_q[10]]^+$	$1.16 \cdot 10^{-6}$
$[\boldsymbol{\sigma}_{a_\text{m}}]^+$	$3.10 \cdot 10^{-3}$	$[\boldsymbol{\sigma}_q[11]]^+$	$1.30 \cdot 10^{-3}$
$[\boldsymbol{\sigma}_r]^+$	$(1.02, 1.03, 1.00)^\top \cdot 10^{-6}$		
Initial state mean			
a_d	0.2168		
a_m	$1.22 \cdot 10^{-5}$		

with $\boldsymbol{w}_{\to x,\mu,\sigma} \in \mathbb{R}^6$, $\boldsymbol{W}_1 \in \mathbb{R}^{256\times 3}$, $\boldsymbol{W}_2 \in \mathbb{R}^{6\times 256}$. The max operator is applied elementwise. The initial mean $\boldsymbol{w}_{\to x}$ is taken as the first three dimensions of $\boldsymbol{w}_{\to x,\mu,\sigma}[1:3]$, the latter three dimensions parametrize the initial covariance as $\boldsymbol{\Sigma}_{\omega\to x} = \text{diag}([\boldsymbol{w}_{\to x,\mu,\sigma}[4:6]]^\oplus)$.

C.2. Derivatives

For the extended Kalman filter, we need to compute derivatives (Jacobians) of the forward dynamics $\mathbf{J} = \frac{\partial}{\partial \boldsymbol{z}_n} g(\boldsymbol{z}_n)$. For computational considerations, we manually implemented the derivatives. The dynamics depend on whether an impact has happened. We restate Equation 6.22

with $z^+ = h(z^-) = C' z^-$:

$$z_{n+1} = g(z_n) = \begin{cases} g_{\text{free}}(z_n, \Delta_T) & \left[g_{\text{free}}(z_n, \Delta_T)\right]_z - r \geq z_{\text{table}} \\ g_{\text{free}}(h(g_{\text{free}}(z_n, \Delta_{\text{imp}})), \Delta_T - \Delta_{\text{imp}}) & \text{otherwise.} \end{cases}$$

The function h maps the state prior to the impact $z^- = g_{\text{free}}(z_n, \Delta_{\text{imp}})$ to the state after the impact $z^+ = h(z^-)$. We abbreviate the remaining time after the impact as $\Delta_{\text{rem}} = \Delta_T - \Delta_{\text{imp}}$. First, we consider free-flight dynamics, and compute the Jacobian

$$J_z(z_n, \Delta_t) = \frac{\partial}{\partial z'_n} g_{\text{free}}(z'_n, \Delta_t)|_{z'_n = z_n} \tag{C.1}$$

for an arbitrary timespan Δ_t given by the components

	$\cdot/\partial p_n$	$\cdot/\partial v_n$	$\cdot/\partial \omega_n$	$\cdot/\partial a_{\text{d},n}$	$\cdot/\text{d}a_{\text{m},n}$
$\partial p_{n+1}/\cdot$	I	$\Delta_t I$	0	0	0
$\partial v_{n+1}/\cdot$	0	Eq. C.2	Eq. C.3	Eq. C.4	Eq. C.5
$\partial \omega_{n+1}/\cdot$	0	0	I	0	0
$\partial a_{\text{d},n+1}/\cdot$	0	0	0	1	0
$\partial a_{\text{m},n+1}/\cdot$	0	0	0	0	1

with the following velocity derivatives

$$\frac{\partial v_{n+1}}{\partial v_n} = I - \Delta_t a_{\text{d},n}^2 \frac{\partial(||v_n|| v_n)}{\partial v_n} + \Delta_t a_{\text{m},n}^2 \frac{\partial(\omega_n \times v_n)}{\partial v_n} \tag{C.2}$$

$$\frac{\partial v_{n+1}}{\partial \omega_n} = \Delta_t a_{\text{m},n}^2 \frac{\partial(\omega_n \times v_n)}{\partial \omega_n} \tag{C.3}$$

$$\frac{\partial v_{n+1}}{\partial a_{\text{d},n}} = -2\Delta_t a_{\text{d},n} ||v_n|| v_n \tag{C.4}$$

$$\frac{\partial v_{n+1}}{\partial a_{\text{m},n}} = 2\Delta_t a_{\text{m},n} (\omega_n \times v_n). \tag{C.5}$$

Let us now consider the case where an impact has happened. First, it is important to note that the impact time Δ_{imp} (and thus also the remaining time Δ_{rem}) depends on z_n, in particular $v_{z,n}$, as

$$\Delta_{\text{imp}} = -\left(v_{z,n} + \sqrt{v_{z,n}^2 + 2 g_z h}\right) / g_z.$$

We can thus write

$$z_{n+1} = g_{\text{free}}(h(g_{\text{free}}(z_n, \Delta_{\text{imp}}(z_n))), \Delta_{\text{rem}}(z_n))$$

and obtain

$$\frac{\partial z_{n+1}}{\partial z_n} = \underbrace{\frac{\partial g_{\text{free}}(z^+, \Delta_{\text{rem}})}{\partial z^+}}_{J1} \underbrace{\frac{\partial z^+}{\partial z_n}}_{J2} + \underbrace{\frac{\partial g_{\text{free}}(z^+, \Delta_{\text{rem}})}{\partial \Delta_{\text{rem}}}}_{J3} \underbrace{\frac{\partial \Delta_{\text{rem}}}{\partial z_n}}_{J4}$$

with $z^+ = h(g_{\text{free}}(z_n, \Delta_{\text{imp}}(z_n)))$. The term $(J1) = \boldsymbol{J}_z(z^+, \Delta_{\text{rem}})$ is given by the Jacobian in Equation C.1. For later use, we introduce the Jacobian

$$\boldsymbol{J}_{\Delta_t}(z_n, \Delta_t) = \frac{\partial}{\partial \Delta_t'} g_{\text{free}}(z_n, \Delta_t')|_{\Delta_t'=\Delta_t}, \tag{C.6}$$

given as

$$\boldsymbol{J}_{\Delta_t}(z_n, \Delta_t) = (\boldsymbol{v}_n^\top, (-a_{\text{d},n}^2 ||\boldsymbol{v}_n|| \boldsymbol{v}_n + a_{\text{m},n}^2(\boldsymbol{\omega}_n \times \boldsymbol{v}_n) + \boldsymbol{g})^\top, \boldsymbol{0}, 0, 0)^\top.$$

With Equation C.6, $(J3)$ is given by $\boldsymbol{J}_{\Delta_t}(z^+, \Delta_{\text{rem}})$. Next, we decompose $J2$:

$$\frac{\partial z^+}{\partial z_n} = \underbrace{\frac{\partial h(z^-)}{\partial z^-}}_{J2.1} \underbrace{\frac{\partial z^-}{\partial z_n}}_{J2.2}$$

As $h(z^-) = \boldsymbol{C}' z^-$, it follows that $(J2.1) = \boldsymbol{C}'$. For $(J2.2)$ we have to take into account that $g_{\text{free}}(z_n, \Delta_{\text{imp}}(z_n))$ is a function of the state z_n and of the impact time, which again is a function of z_n:

$$\frac{\partial z^-}{\partial z_n} = \frac{\mathrm{d} g_{\text{free}}(z_n, \Delta_{\text{imp}}(z_n))}{\mathrm{d} z_n} = \underbrace{\frac{\partial g_{\text{free}}(z_n, \Delta_{\text{imp}})}{\partial z_n}}_{J2.2.1} + \underbrace{\frac{\partial g_{\text{free}}(z_n, \Delta_{\text{imp}})}{\partial \Delta_{\text{imp}}}}_{J2.2.2} \underbrace{\frac{\partial \Delta_{\text{imp}}}{\partial z_n}}_{J2.2.3}$$

The terms $(J2.2.1)$, $(J2.2.2)$ are given by the Jacobians $\boldsymbol{J}_z(z_n, \Delta_{\text{imp}})$ and $\boldsymbol{J}_{\Delta_t}(z_n, \Delta_{\text{imp}})$, respectively. For $(J2.2.3)$, let us recapitulate the computation of the impact time

$$\Delta_{\text{imp}} = -\left(v_{\text{z},n} + \sqrt{v_{\text{z},n}^2 + 2 g_{\text{z}} h}\right) / g_{\text{z}}$$

with $h = -((p_{\text{z},n} - r) - \text{z}_{\text{table}})$, i.e.,

$$\begin{aligned}\Delta_{\text{imp}} &= -\left(v_{\text{z},n} + \sqrt{v_{\text{z},n}^2 - 2 g_{\text{z}}((p_{\text{z},n} - r) - \text{z}_{\text{table}})}\right) / g_{\text{z}} \\ &= -\left(v_{\text{z},n} + \sqrt{v_{\text{z},n}^2 - 2 g_{\text{z}} p_{\text{z},n} + 2 g_{\text{z}} r + 2 g_{\text{z}} \text{z}_{\text{table}}}\right) / g_{\text{z}}.\end{aligned}$$

For the derivative w.r.t z_n $(J2.2.3)$ it follows that

$$\frac{\partial \Delta_{\text{imp}}}{\partial z_n} = [0, 0, \frac{\partial \Delta_{\text{imp}}}{\partial p_{\text{z},n}}, 0, 0, \frac{\partial \Delta_{\text{imp}}}{\partial v_{\text{z},n}}, 0, 0, 0, 0, 0]$$

with

$$\frac{\partial \Delta_{\text{imp}}}{\partial p_{\text{z},n}} = \frac{1}{\sqrt{v_{\text{z},n}^2 - 2 g_{\text{z}} p_{\text{z},n} + 2 g_{\text{z}} r + 2 g_{\text{z}} \text{z}_{\text{table}}}},$$

$$\frac{\partial \Delta_{\text{imp}}}{\partial v_{\text{z},n}} = \frac{-1}{g_{\text{z}}}\left(1 + \frac{v_{\text{z},n}}{\sqrt{v_{\text{z},n}^2 - 2 g_{\text{z}} p_{\text{z},n} + 2 g_{\text{z}} r + 2 g_{\text{z}} \text{z}_{\text{table}}}}\right).$$

From $\Delta_{\text{rem}} = \Delta_T - \Delta_{\text{imp}}$, it follows that $\frac{\partial \Delta_{\text{rem}}}{\partial z_n} = -\frac{\partial \Delta_{\text{imp}}}{\partial z_n}$, which finally gives $(J4)$.

Appendix: Learning Temporally Extended Skills for Planning

D.

In the following we provide supplementary details and analysis of the approach. We give details on the implementation of the single-player board game environments in Appendix D.1, including splitting initial board configurations in train- and test splits (Appendix D.1.4). Details on the implementation of the SEADS agent are given in Appendix D.2. Skill trajectories on the *Cursor* and *Reacher* environments are shown in Appendix D.3. Appendix D.4 provides additional details on the real robot experiment. Finally, we give additional details on the SAC, HAC, and VIC baselines (Appendices D.5 and D.7) and the hyperparameter search for SAC and HAC (Appendix D.8).

D.1. Environment details

D.1.1. Cursor environments

At the beginning of an episode, the position of the cursor is reset to $x \sim \mathcal{U}(0, 1)$, $y \sim \mathcal{U}(0, 1)$. After a game move, the cursor is not reset. Both *LightsOut* and *TileSwap* board extents are $(x, y) \in [0, 1] \times [0, 1]$.

D.1.2. Reacher environments

At the beginning of an episode, the two joints of the Reacher are set to random angles $\theta \sim \mathcal{U}(0, 2\pi)$, $\phi \sim \mathcal{U}(0, 2\pi)$. After a game move, the joints are not reset. Both *LightsOut* and *TileSwap* board extents are $(x, y) \in [-0.15, 0.15] \times [-0.15, 0.15]$. The control simulation timestep is 0.02 s, and we use an action repeat of 2.

D.1.3. Jaco environments

At the beginning of an episode and after a game move the Jaco arm is randomly reset above the board. The tool's (end effector) center point is randomly initialized to $x \sim \mathcal{U}(-0.1, 0.1)$, $y \sim \mathcal{U}(-0.1, 0.1)$, $z \sim \mathcal{U}(0.2, 0.4)$ with random rotation $\theta \sim \mathcal{U}(-\pi, \pi)$. The *LightsOut* board extent is $(x, y) \in [-0.25, 0.25] \times [-0.25, 0.25]$, the *TileSwap* board extent $(x, y) \in [-0.15, 0.15] \times [-0.15, 0.15]$. We use a control timestep of 0.1 s in the simulation.

D.1.4. Train-/Test-split

In order to ensure disjointness of board configurations in train and test split we label each board configuration based on a hash remainder. For the hashing algorithm we first represent the current board configuration as comma-separated string, e.g. $s =$ "1, 1, 0, . . . , 0" for *LightsOut* and

s = "1, 0, 2, . . . , 8" for *TileSwap*. Then, this string is passed through a CRC32 hashing function, yielding the split based on an integer division remainder

$$\text{split} = \begin{cases} \text{train} & \text{CRC32}(s) \bmod 3 = 0 \\ \text{test} & \text{CRC32}(s) \bmod 3 \in \{1, 2\} \end{cases} \tag{D.1}$$

D.2. SEADS architectural details

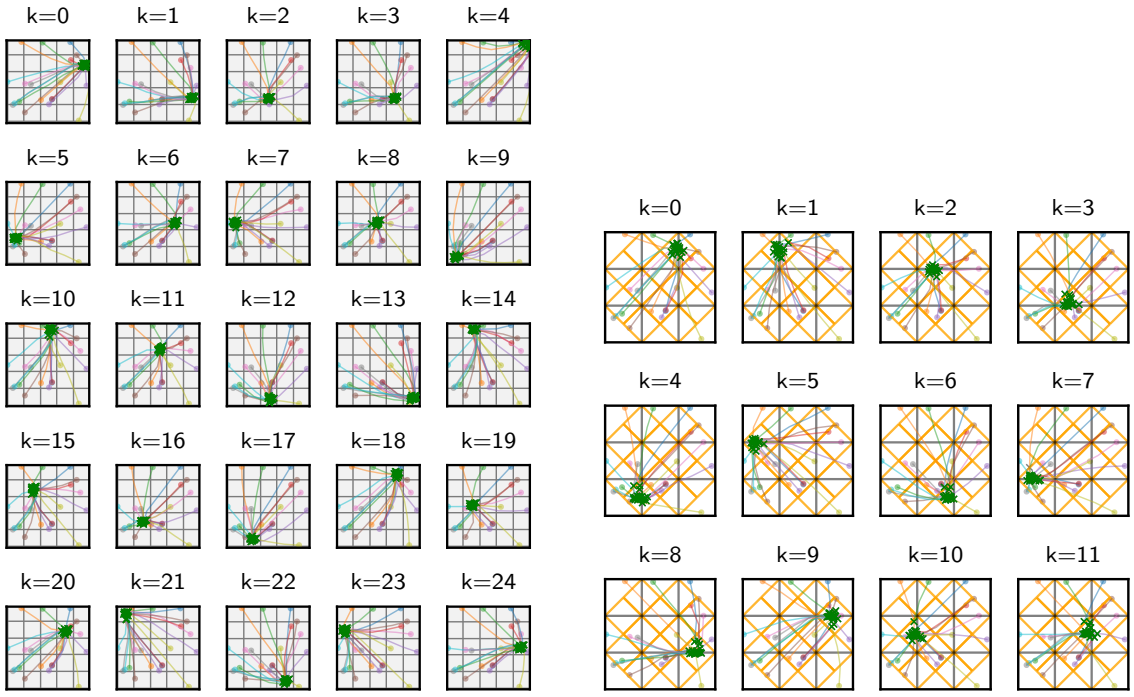
SAC agent

We use an open-source soft actor-critic implementation (Tandon, 2021) in our SEADS agent. Policy and critic networks are modeled by a multilayer perceptron with two hidden layers with ReLU activations. For hyperparameters, please see the table below.

{*LightsOut*, *TileSwap* }-	*Cursor*	*Reacher*	*Jaco*	Robot (LightsOut)
Learning rate	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$
Target smoothing coeff. τ	0.005	0.005	0.005	0.005
Discount factor γ	0.99	0.99	0.99	0.99
Hidden dim.	512	512	512	512
Entropy target α	0.1	0.01	0.01	0.1
Automatic entropy tuning	no	no	no	no
Distribution over actions	Gaussian	Gaussian	Gaussian	Gaussian

D.3. Skill trajectories

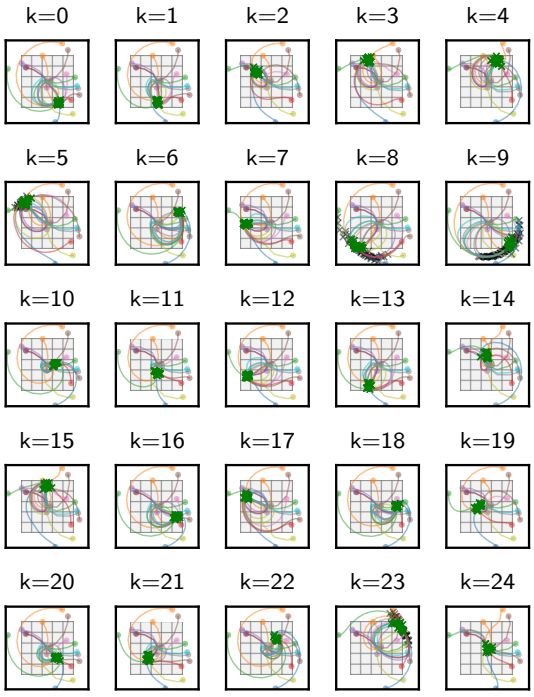
In Figures D.1 and D.2 we provide visualizations of trajectories executed by the learned skills on the *Cursor* and *Reacher* environments, respectively.



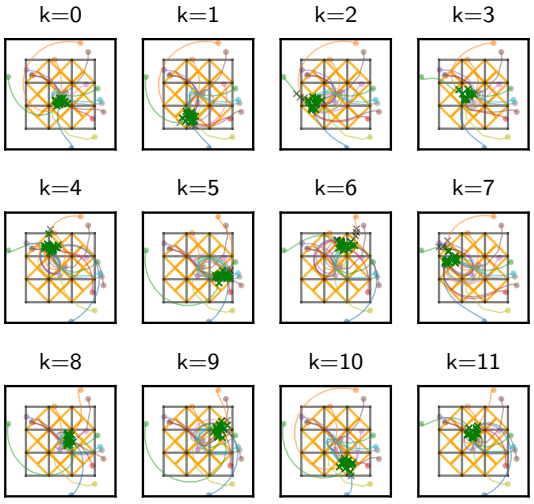
(a) Skill trajectories on *LightsOutCursor*.

(b) Skill trajectories on *TileSwapCursor*.

Figure D.1.: Trajectories in *Cursor*-embedded environments for skills k on 20 different environment initializations. Colored lines show the x, y-coordinates of the Cursor, with the circular marker indicating the start position of the skill. Black markers indicate locations where a “push" was executed, and green markers where the push has caused a change in the symbolic state.



(a) Skill trajectories on *LightsOutReacher*.



(b) Skill trajectories on *TileSwapReacher*.

Figure D.2.: Trajectories in *Reacher*-embedded environments for skills k on 20 different environment initializations. Colored lines show the x, y-coordinates of the Reacher end-effector, with the circular marker indicating the start position of the skill. Black markers indicate locations where a "push" was executed, and green markers where the push has caused a change in the symbolic state.

D.4. Real robot experiment

For the real robot experiment we use a uArm Swift Pro robotic arm that interacts with a *LightsOut* board game. The board game runs on a Samsung Galaxy Tab A6 Android tablet with a screen size of 10.1 inches.

We mapped the screen plane excluding the system status bar and action bar of the app (blue bar) to normalized coordinates $(x, y) \in [0, 1]$. A third $z \in [0, 1]$ coordinate measures the perpendicular distance to the screen plane, with $z = 1$ approximately corresponding to a distance of 10 cm to the screen. To control the robot arm, we use the Python SDK from (uArm-Developer, 2021), which allows to steer the end effector to $\overrightarrow{X} = (X, Y, Z)$ target locations in a coordinate frame relative to the robot's base.

As the robot's base is not perfectly aligned with the tablet's surface, e.g. due to the rear camera, we employed a calibration procedure. We measured the location of the four screen corners in (X, Y, Z) coordinates using the SDK's `get_position` method (by placing the end effector holding the capacitive pen on the particular corners) and fitted a plane to these points minimizing the squared distance. We reproject the measured points onto the plane and compute a perspective transform by pairing the reprojected points with normalized coordinates $(x, y) \in \{0, 1\} \times \{0, 1\}$. To obtain robot coordinates (X, Y, Z) from normalized coordinates (x, y, z) we first apply the perspective transform on (x, y), yielding $\hat{X} = (X, Y, Z = 0)$. We subsequently add the plane's normal to (X, Y, Z) scaled by z and an additional factor which controls the distance to the tablet's surface for $z = 1$.

The state of the board is communicated to the host machine running SEADS via USB through the logging functionality of the Android Debug Bridge. The whole system including robotic arm and Android tablet is interfaced as an `OpenAI Gym` (Brockman et al., 2016) environment.

The action space of the environment is 3-dimensional $a = (\Delta x, \Delta y, p)$, with the first two actions being positional displacement actions $\Delta x, \Delta y \in [-0.2, 0.2]$ and the third action $p \in [-1, 1]$ indicating whether a "push" should be executed. The displacement actions represent incremental changes to the robotic arm's end effector position. In normalized coordinates the end effector is commanded to steer to $(\text{clip}(x + \Delta x, 0, 1), \text{clip}(y + \Delta y, 0, 1), z = 0.3)$, where (x, y) are the current coordinates of the end effector. If the push action p exceeds a threshold $p > 0.6$, first the end effector is displaced, followed by a push, which is performed by sending the target coordinates $(x, y, z = 0), (x, y, z = 0.3)$ to the arm subsequently. Thus, the SEADS agent has to learn temporally extended skills which first locate the end effector above a particular board field and then execute the push.

D.5. SAC baseline

We train the SAC baseline in a task-specific way by giving a reward of 1 to the agent if the board state has reached its target configuration and 0 otherwise. At the beginning of each episode, we first sample the difficulty of the current episode which corresponds to the number of moves required to solve the game (solution depth S). For all environments S is uniformly sampled from $\{1, \ldots, 5\}$. For all *Cursor* environments we impose a step limit $T_{\text{lim}} = 10 \cdot S$, for *Reacher* and *Jaco* $T_{\text{lim}} = 50 \cdot S$. This corresponds to the number of steps a single skill can make in SEADS

multiplied by S. We use a replay buffer which holds the most recent 1 million transitions and train the agent with a batchsize of 256. The remaining hyperparameters (see table below) are identical to the SAC component in SEADS; except for an increased number of hidden units and an additional hidden layer (i.e., three hidden layers) in the actor and critc networks to account for the planning the policy has to perform. In each epoch of training we collect 8 samples from each environment which we store in the replay buffer. We performed a hyperparameter search on the number of agent updates performed in each epoch N and entropy target values α. We also experimented with skipping updates, i.e., collecting 16 (for $N = 0.5$) or 32 (for $N = 0.25$) environment samples before performing a single update. We found that performing too many updates leads to unstable training (e.g., $N = 4$ for *LightsOutCursor*). For all results and optimal settings per environment, we refer to Appendix D.8.1. For the SAC baseline we use the same SAC implementation from (Tandon, 2021) which we use for SEADS.

{*LightsOut*, *TileSwap* }-	*Cursor*	*Reacher*	*Jaco*
Learning rate	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$
Target smoothing coeff. τ	0.005	0.005	0.005
Discount factor γ	0.99	0.99	0.99
Hidden dim.	512	512	512
Entropy target α	*tuned* (see Appendix D.8.1)		
Automatic entropy tuning	no	no	no
Distribution over actions	Gaussian	Gaussian	Gaussian

D.6. HAC baseline

For the HAC baseline we adapt the official code release (Levy, 2020). We modify the architecture to allow for a two-layer hierarchy of policies in which the higher-level policy commands the lower-level policy with *discrete* subgoals (which correspond to the symbolic observations z in our case). This requires the higher-level policy to act on a discrete action space $\mathcal{A}_{\text{high}} = \mathcal{Z}$. The lower-level policy acts on the continuous actions space $\mathcal{A}_{\text{low}} = \mathcal{A}$ of the respective manipulator (*Cursor*, *Reacher*, *Jaco*). To this end, we use a discrete-action SAC agent for the higher-level policy and a continuous-action SAC agent for the lower-level policy. For the higher-level discrete SAC agent we parameterize the distribution over actions as a factorized reparametrizable RelaxedBernoulli distribution, which is a special case of the Concrete (Maddison et al., 2017) / Gumbel-Softmax (Jang et al., 2017) distribution. We use an open-source SAC implementation (Tandon, 2021) for the SAC agent on both levels and extend it by a RelaxedBernoulli distribution over actions for the higher-level policy.

D.6.1. Hyperparameter search

We performed an extensive hyperparameter search on all 6 environments ([*LightsOut*, *TileSwap*] $\times$ [*Cursor*, *Reacher*, *Jaco*]) for the HAC baseline. We investigated a base set of entropy target values $\alpha_{\text{low}}, \alpha_{\text{high}} \in \{0.1, 0.01, 0.001, 0.0001\}$ for both layers separately. On the *Cursor* environments we refined these sets in regions of high success rates. We performed a hyperparameter search on the temperature parameter τ of the RelaxedBernoulli distribution on the *Cursor* environments with $\tau \in \{0.01, 0.05, 0.1, 0.5\}$ and found $\tau = 0.1$ to yield the best results. For experiments on

the *Reacher* and *Jaco* environments we then fixed the parameter $\tau = 0.1$. We report results on parameter sets with highest average success rate on 5 individually trained agents after 5×10^5 (*Cursor*) / 1×10^7 (*Reacher*, *Jaco*) environment interactions. We refer to Appendix D.8.2 for a visualization of all hyperparameter search results.

Parameters for high-level policy

	all environments
Learning rate	$3 \cdot 10^{-4}$
Target smoothing coeff. τ	0.005
Discount factor γ	0.99
Hidden layers for actor/critic	2
Hidden dim.	512
Entropy target α_{high}	*tuned* (see Appendix D.8.2)
Automatic entropy tuning	no
Distribution over actions	RelaxedBernoulli
RelaxedBernoulli temperature τ	*tuned* (see Appendix D.8.2)

Parameters for low-level policy

	all environments
Learning rate	$3 \cdot 10^{-4}$
Target smoothing coeff. τ	0.005
Discount factor γ	0.99
Hidden layers for actor/critic	2
Hidden dim.	512
Entropy target α_{high}	*tuned* (see Appendix D.8.2)
Automatic entropy tuning	no
Distribution over actions	Gaussian

D.7. VIC baseline

We compare to Variational Intrinsic Control (VIC, Gregor et al. (2017)) as a baseline method of unsupervised skill discovery. It is conceptually similar to our method as it aims to find skills such that the mutual information $\mathcal{I}(s_T, k \mid s_0)$ between the skill termination state s_T and skill k is maximized given the skill initiation state s_0. To this end it jointly learns a skill policy $\pi(s_t \mid a_t, k)$ and *skill discriminator* $q_\theta(k \mid s_0, s_T)$. We adopt this idea and pose a baseline to our approach in which we model $q_\theta(k \mid z_0, z_T)$ *directly* with a neural network, instead of modelling $q_\theta(k \mid z_0, z_T)$ indirectly through a forward model $q_\theta(z_T \mid z_0, k)$. The rest of the training process including its hyperparameters is identical to SEADS. We implement $q_\theta(k \mid z_0, z_T)$ by a neural network which outputs the parameters of a categorical distribution and is trained by maximizing the log-likelihood $\log q_\theta(k \mid z_0^i, z_{T^i}^i)$ on transition tuples $(z_0^i, k_i, z_{T^i}^i)$ (see Subsection 7.3.4). We

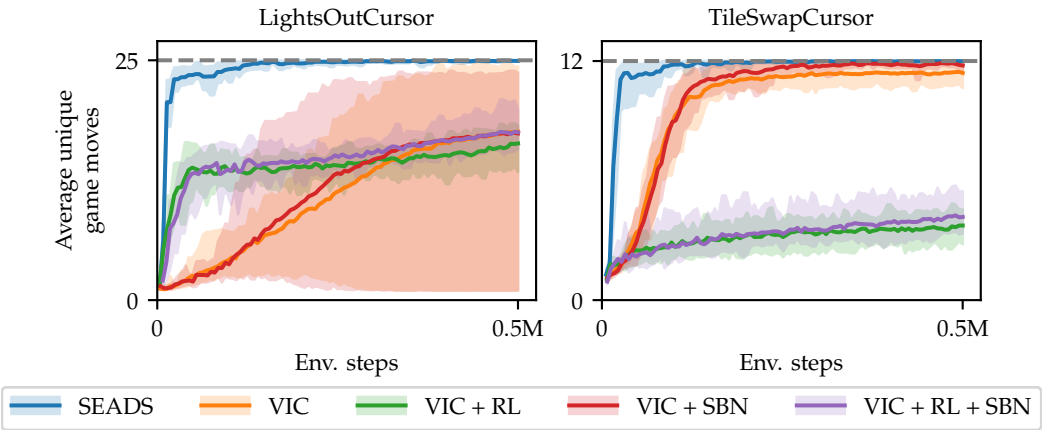


Figure D.3.: Number of discovered skills on the *LightsOutCursor*, *TileSwapCursor* environments for the SEADS agent and variants of VIC (Gregor et al., 2017). Only SEADS discovers all skills reliably on both environments. See Appendix D.7 for details.

experimented with different variants of passing $(z_0^i, z_{T^i}^i)$ to the network: (i) concatenation $[z_0^i, z_{T^i}^i]$ and (ii) concatenation with XOR $[z_0^i, z_{T^i}^i, z_0^i \text{ XOR } z_{T^i}^i]$. We only found the latter to show success during training. The neural network model contains two hidden layers of size 256 with ReLU activations (similar to the forward model). We also evaluate variants of VIC which are extended by our proposed *relabelling scheme* and *second-best reward normalization*. In contrast to VIC, our SEADS agent discovers all possible game moves reliably in both *LightsOutCursor* and *TileSwapCursor* environments, see Figure D.3 for details. Our proposed second-best normalization scheme (+*SBN*, sec. 3) slightly improves performance of VIC in terms of convergence speed (*LightsOutCursor*) and variance in number of skills detected (*TileSwapCursor*). The proposed relabelling scheme (+*RL*, sec. 3) does not improve (*LightsOutCursor*) or degrades (*TileSwapCursor*) the number of detected skills.

D.8. Results of hyperparameter search on HAC and SAC baselines

D.8.1. Results of SAC hyperparameter search

Please see Figures D.4 to D.9 for a visualization of the SAC hyperparameter search results.

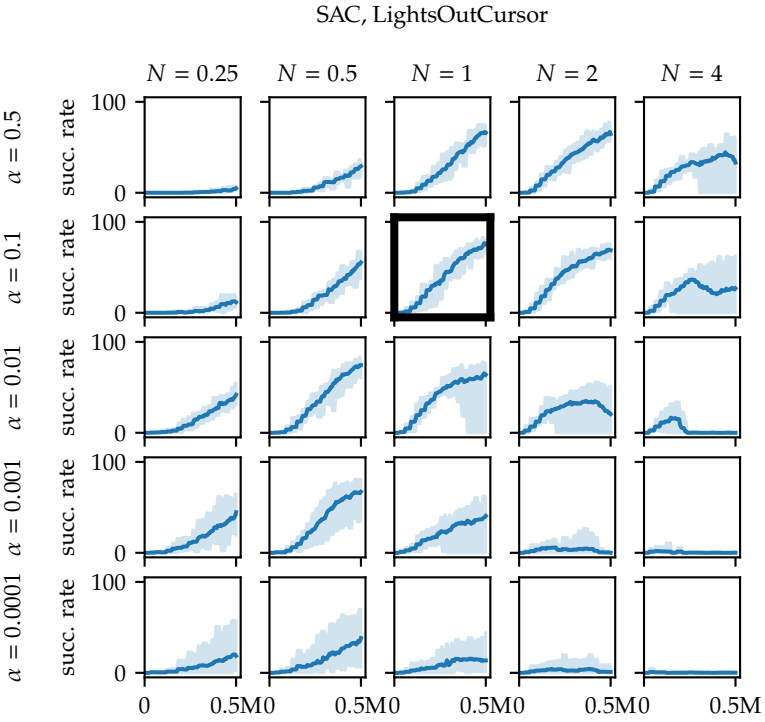


Figure D.4.: Test performance of SAC agents on the *LightsOutCursor* environment for varying number of update steps per epoch (N) and parameters α. We evaluate 5 individual agents per configuration. The best configuration is marked in **bold** ($N = 1$, $\alpha = 0.1$).

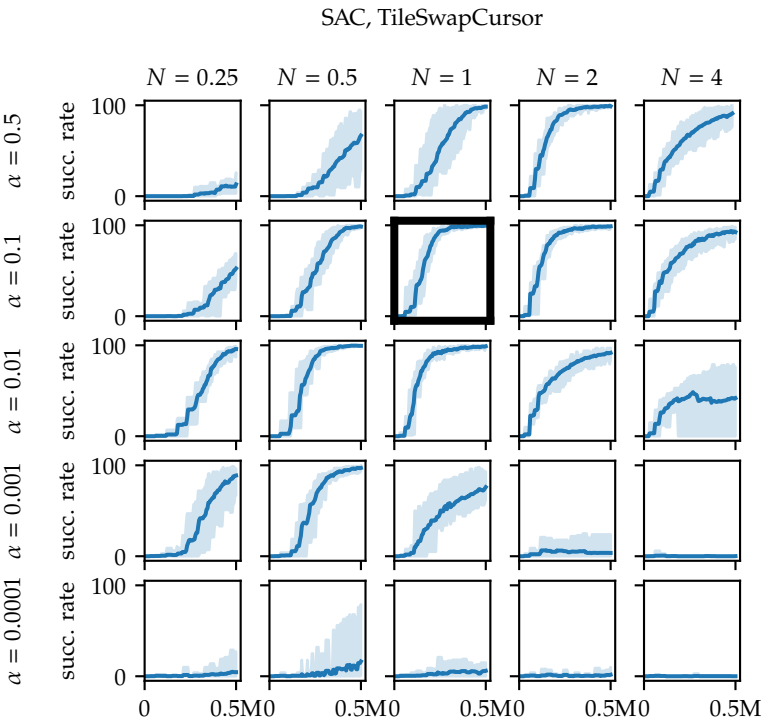


Figure D.5.: Test performance of SAC agents on the *TileSwapCursor* environment for varying number of update steps per epoch (N) and parameters α. We evaluate 5 individual agents per configuration. The best configuration is marked in **bold** ($N = 1$, $\alpha = 0.1$).

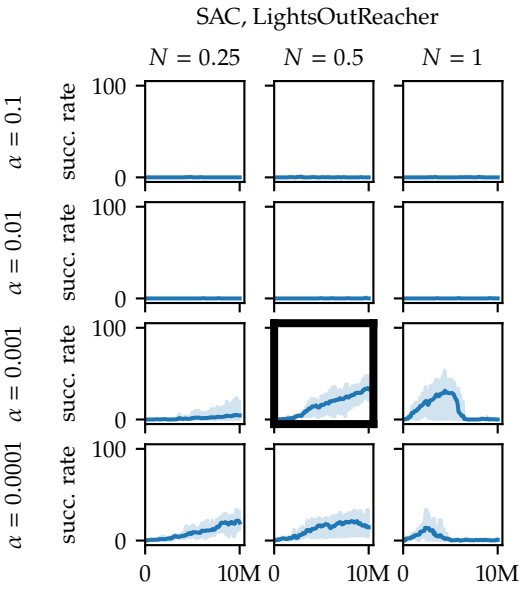


Figure D.6.: Test performance of SAC agents on the *LightsOutReacher* environment for varying number of update steps per epoch (N) and parameters α. We evaluate 5 individual agents per configuration. The best configuration is marked in **bold** ($N = 0.5$, $\alpha = 0.001$).

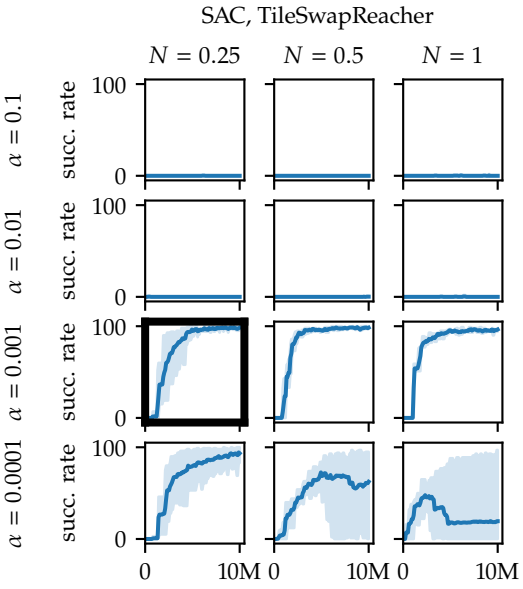


Figure D.7.: Test performance of SAC agents on the *TileSwapReacher* environment for varying number of update steps per epoch (N) and parameters α. We evaluate 5 individual agents per configuration. The best configuration is marked in **bold** ($N = 0.25$, $\alpha = 0.001$).

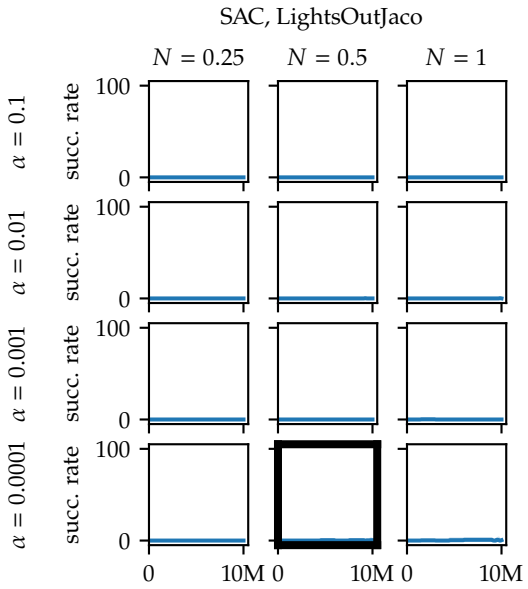


Figure D.8.: Test performance of SAC agents on the *LightsOutJaco* environment for varying number of update steps per epoch (N) and parameters α. We evaluate 5 individual agents per configuration. The best configuration is marked in **bold** ($N = 0.5$, $\alpha = 0.0001$).

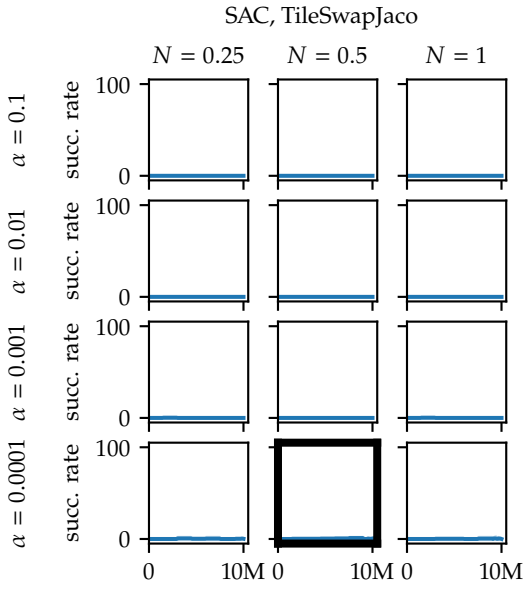
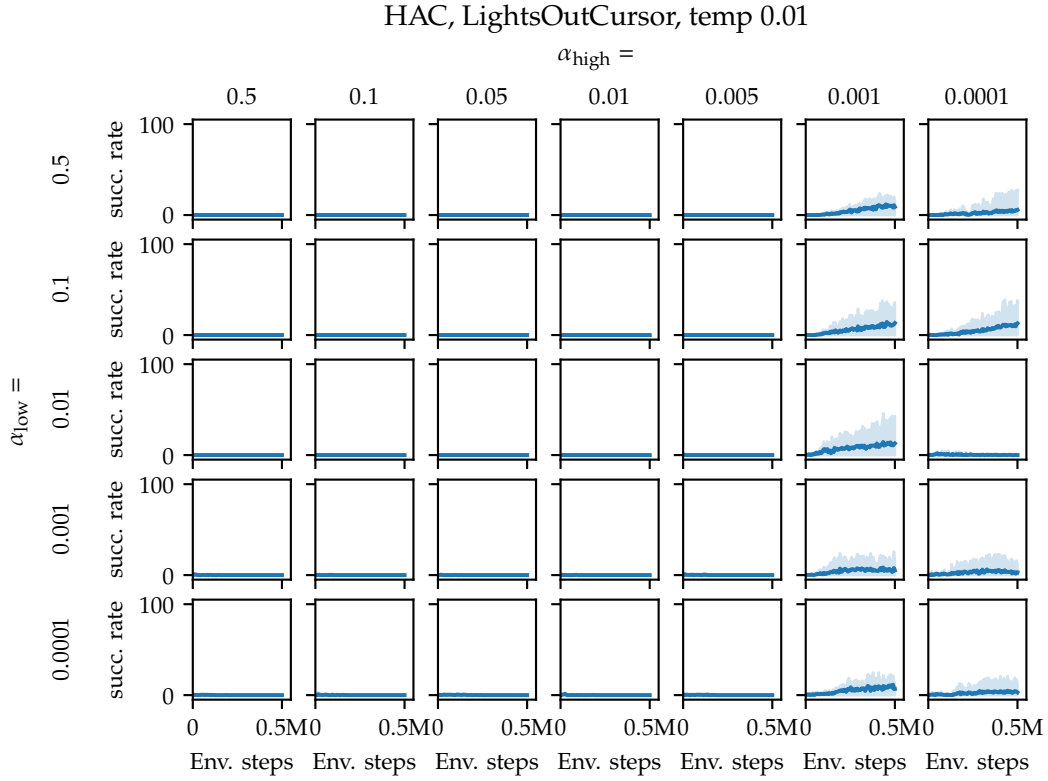
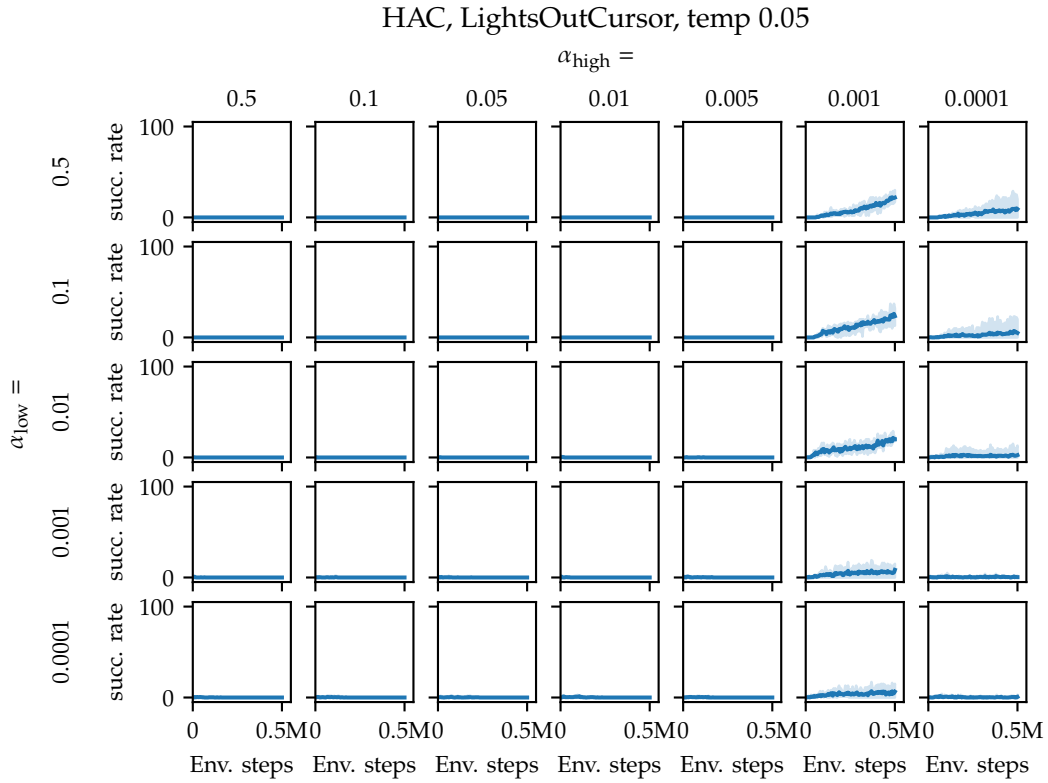


Figure D.9.: Test performance of SAC agents on the *TileSwapJaco* environment for varying number of update steps per epoch (N) and parameters α. We evaluate 5 individual agents per configuration. The best configuration is marked in **bold** ($N = 0.5$, $\alpha = 0.0001$).

D.8.2. Results of HAC hyperparameter search

Please see Figures D.9 to D.13 for a visualization of the HAC hyperparameter search results.





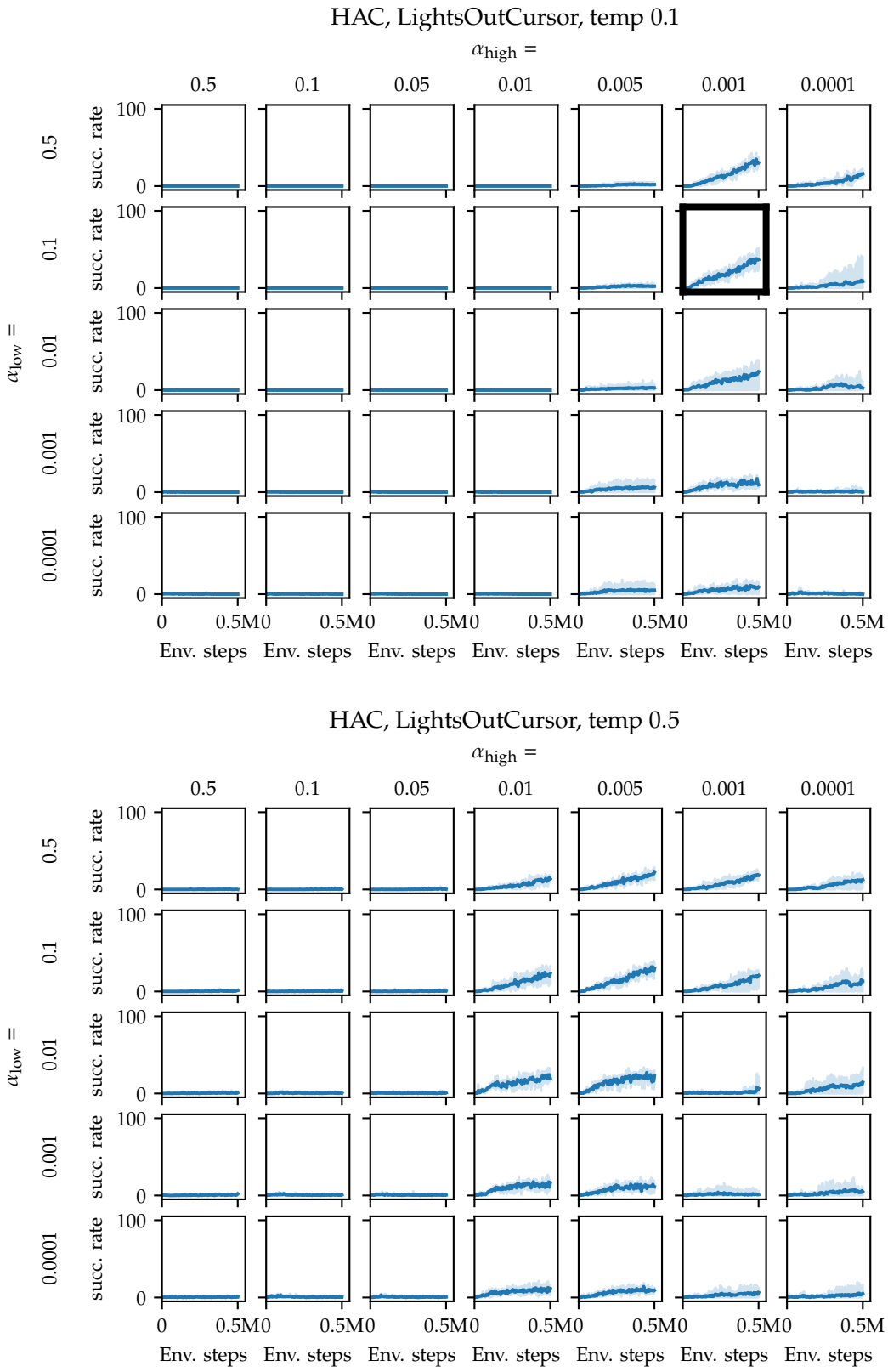
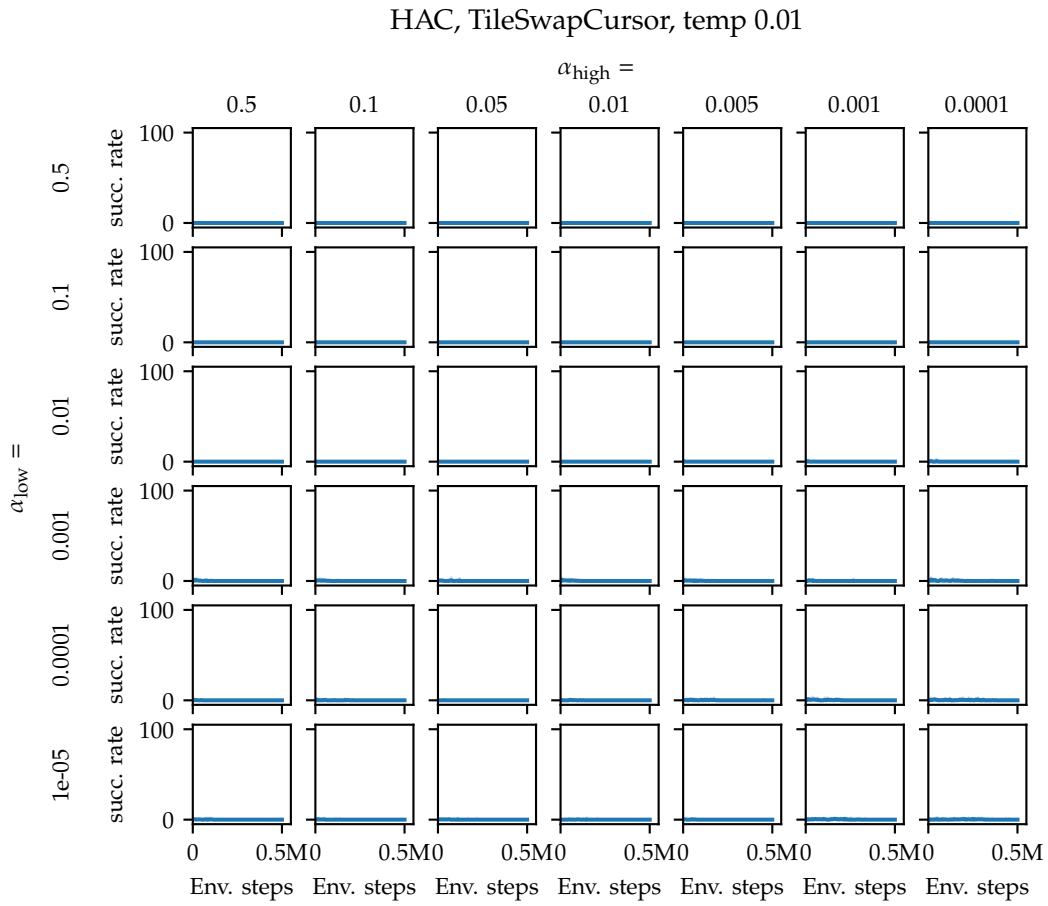
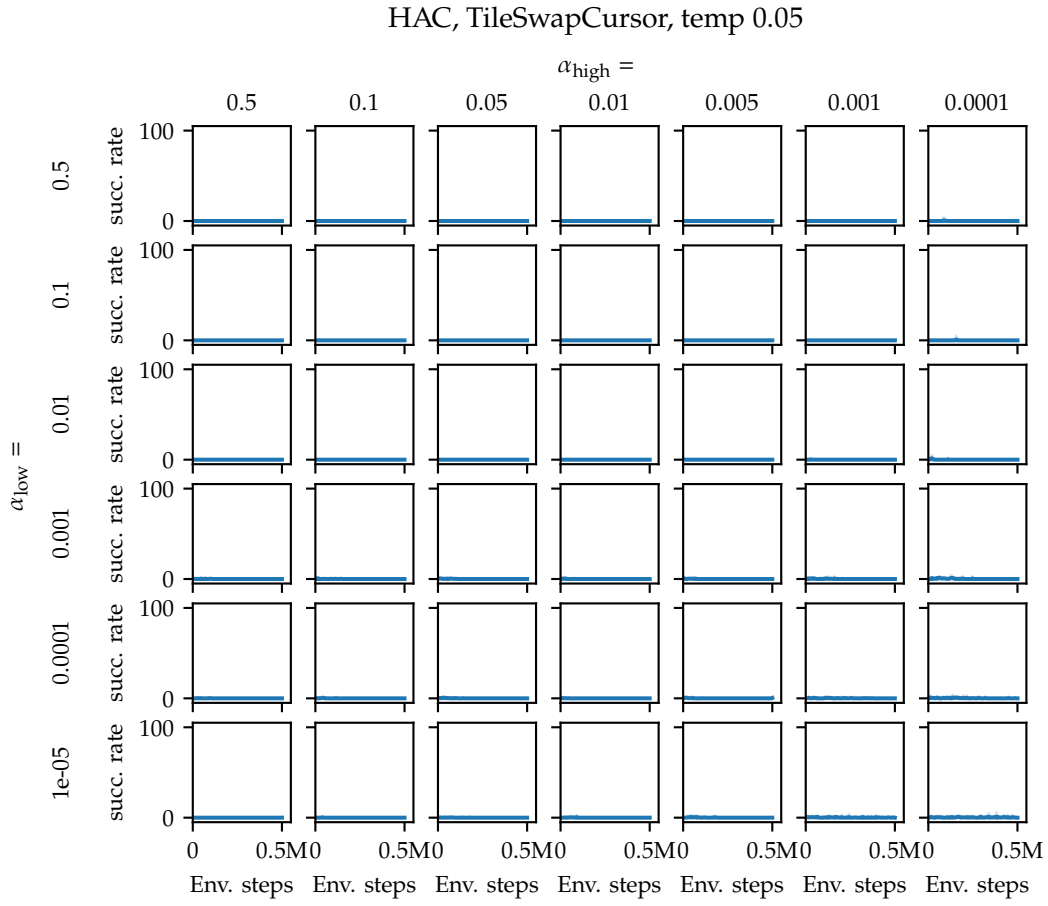
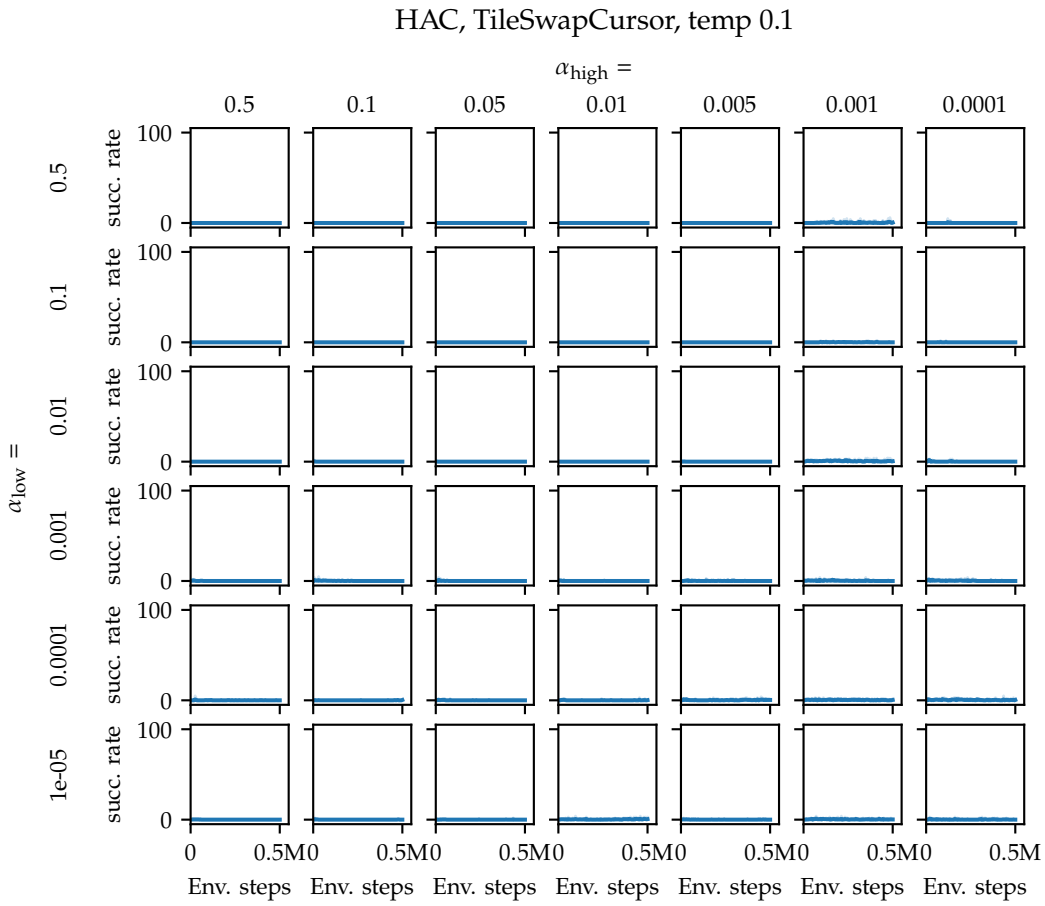


Figure D.9a: Test performance of HAC agents on the *LightsOutCursor* environment for varying values for Relaxed-Bernoulli temperature τ and entropy targets $\alpha_{\text{high}}, \alpha_{\text{low}}$. We evaluate 5 individual agents per configuration. The best configuration is marked in **bold** ($\tau = 0.1$, $\alpha_{\text{low}} = 0.1$, $\alpha_{\text{high}} = 0.001$).


HAC, TileSwapCursor, temp 0.01
α_{high} =
0.5
0.1
0.05
0.01
0.005
0.001
0.0001
α_{low} =
succ. rate
100
0
Env. steps
0
0.5M


HAC, TileSwapCursor, temp 0.05
α_{high} =
0.5
0.1
0.05
0.01
0.005
0.001
0.0001
α_{low} =
succ. rate
100
0
Env. steps
0
0.5M



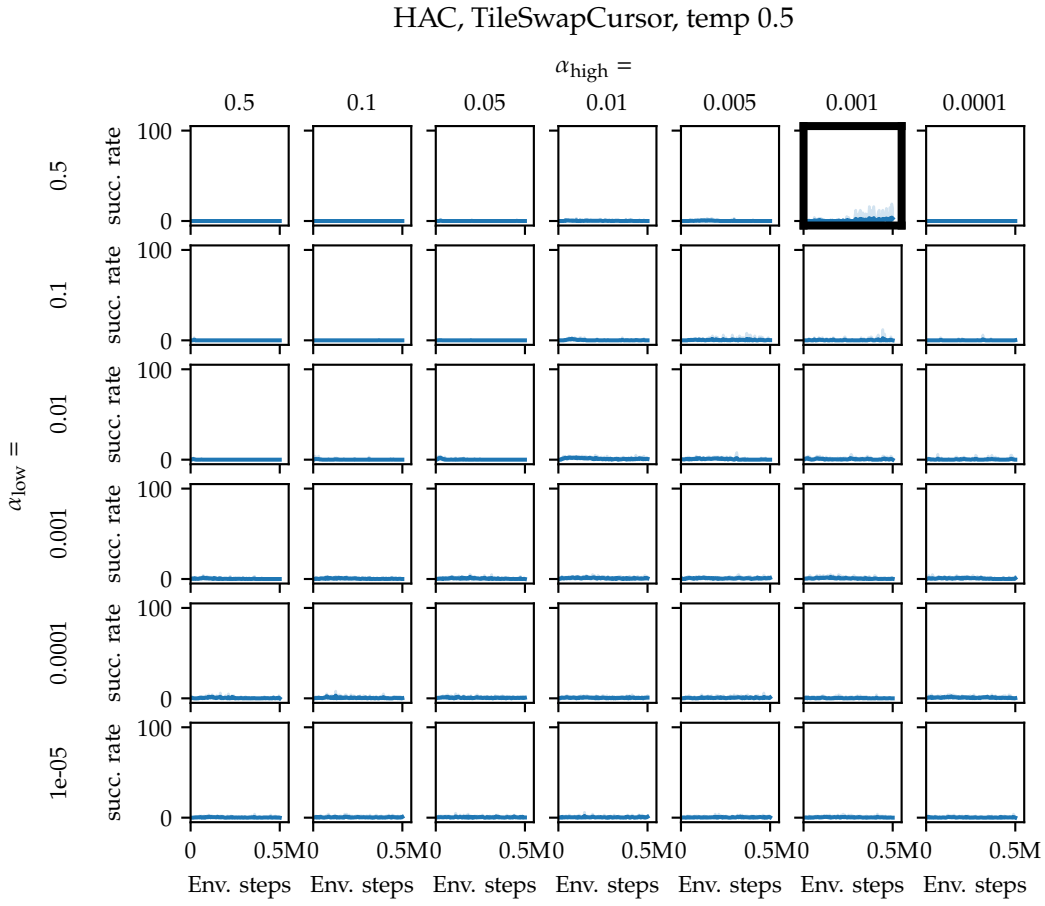


Figure D.9b: Test performance of HAC agents on the *TileSwapCursor* environment for varying values for Relaxed-Bernoulli temperature τ and entropy targets $\alpha_{\text{high}}, \alpha_{\text{low}}$. We evaluate 5 individual agents per configuration. The best configuration is marked in **bold** ($\tau = 0.5$, $\alpha_{\text{low}} = 0.5$, $\alpha_{\text{high}} = 0.001$).

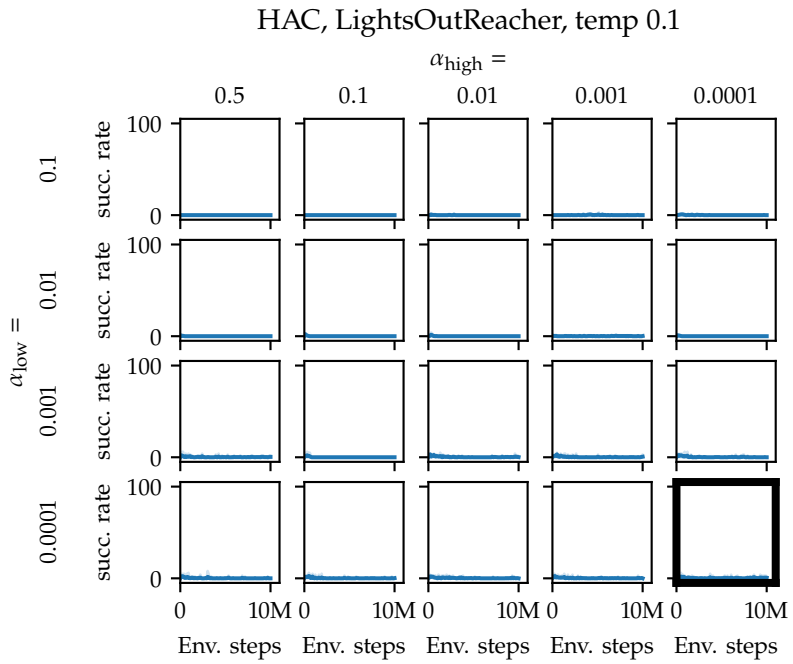


Figure D.10.: Test performance of HAC agents on the *LightsOutReacher* environment for varying values for the entropy targets $\alpha_{\text{high}}, \alpha_{\text{low}}$ and fixed RelaxedBernoulli temperature $\tau = 0.1$. We evaluate 5 individual agents per configuration. The best configuration is marked in **bold** ($\tau = 0.1$, $\alpha_{\text{low}} = 0.0001$, $\alpha_{\text{high}} = 0.0001$).

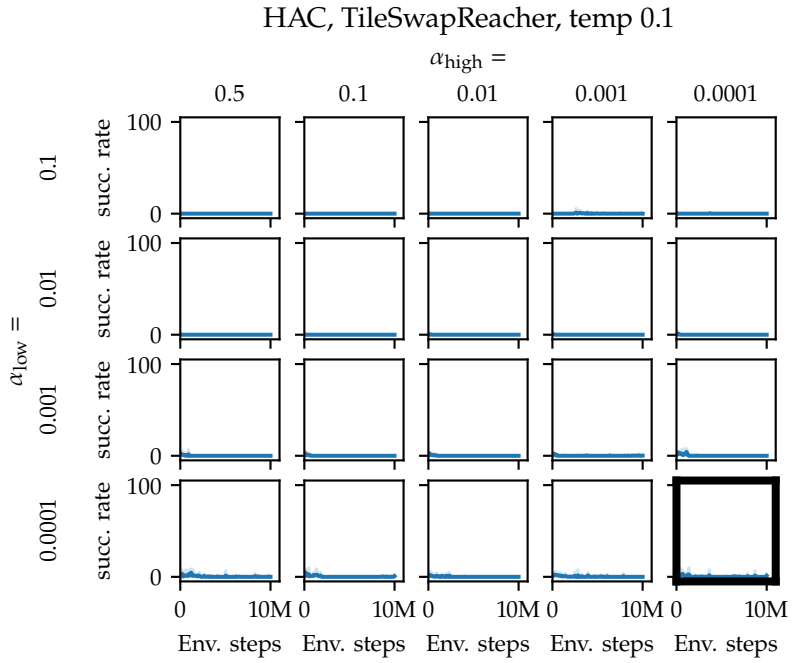


Figure D.11.: Test performance of HAC agents on the *TileSwapReacher* environment for varying values for the entropy targets $\alpha_{\text{high}}, \alpha_{\text{low}}$ and fixed RelaxedBernoulli temperature $\tau = 0.1$. We evaluate 5 individual agents per configuration. The best configuration is marked in **bold** ($\tau = 0.1$, $\alpha_{\text{low}} = 0.0001$, $\alpha_{\text{high}} = 0.0001$).

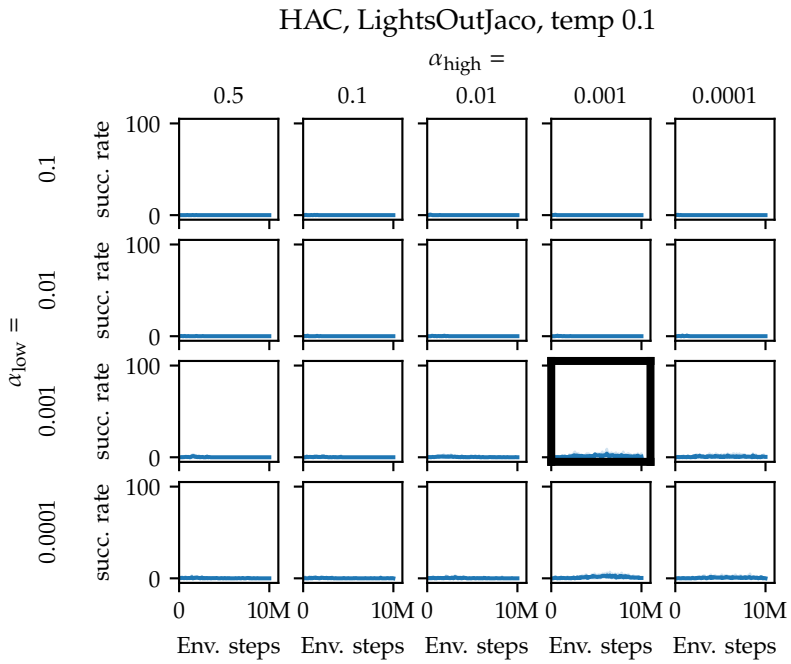


Figure D.12.: Test performance of HAC agents on the *LightsOutJaco* environment for varying values for the entropy targets α_{high}, α_{low} and fixed RelaxedBernoulli temperature $\tau = 0.1$. We evaluate 5 individual agents per configuration. The best configuration is marked in **bold** ($\tau = 0.1$, $\alpha_{\text{low}} = 0.001$, $\alpha_{\text{high}} = 0.001$).

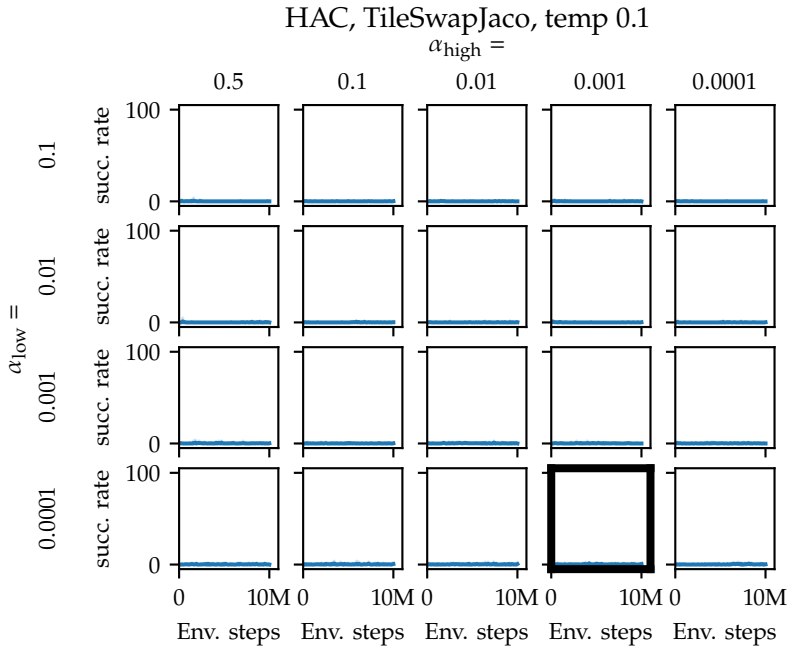


Figure D.13.: Test performance of HAC agents on the *TileSwapJaco* environment for varying values for the entropy targets α_{high}, α_{low} and fixed RelaxedBernoulli temperature $\tau = 0.1$. We evaluate 5 individual agents per configuration. The best configuration is marked in **bold** ($\tau = 0.1$, $\alpha_{\text{low}} = 0.0001$, $\alpha_{\text{high}} = 0.001$).

Bibliography

Achiam, J., H. Edwards, D. Amodei, and P. Abbeel (2018). 'Variational Option Discovery Algorithms'. In: *CoRR* abs/1807.10299 (cited on page 99).

Achterhold, J., M. Krimmel, and J. Stueckler (2022). 'Learning Temporally Extended Skills in Continuous Domains as Symbolic Actions for Planning'. In: *Proceedings of the Conference on Robot Learning (CoRL)* (cited on pages 13, 14, 97).

Achterhold, J. and J. Stueckler (2021). 'Explore the Context: Optimal Data Collection for Context-Conditional Dynamics Models'. In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)* (cited on pages 13, 14, 43, 46, 57, 63, 68, 71, 76, 78, 123).

Achterhold, J., P. Tobuschat, H. Ma, D. Büchler, M. Muehlebach, and J. Stueckler (2023). 'Black-Box vs. Gray-Box: A Case Study on Learning Table Tennis Ball Trajectory Prediction with Spin and Impacts'. In: *Proceedings of the Learning for Dynamics and Control Conference (L4DC)* (cited on pages 13, 14, 79, 82, 95).

Ahmetoglu, A., M. Y. Seker, J. H. Piater, E. Öztop, and E. Ugur (2022). 'DeepSym: Deep Symbol Generation and Rule Learning for Planning from Unsupervised Robot Interaction'. In: *Journal of Artificial Intelligence Research* 75, pp. 709–745 (cited on pages 100, 126).

Amos, B., I. Rodríguez, J. Sacks, B. Boots, and J. Z. Kolter (2018). 'Differentiable MPC for End-to-end Planning and Control'. In: *Advances in Neural Information Processing Systems (NeurIPS)* (cited on page 13).

Andersson, R. L. (1989). 'Understanding and applying a robot ping-pong player's expert controller'. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (cited on pages 79, 81).

Andrychowicz, M., D. Crow, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba (2017). 'Hindsight Experience Replay'. In: *Advances in Neural Information Processing Systems (NeurIPS)* (cited on page 106).

Arulkumaran, K. (2021). *PlaNet*. `https://github.com/Kaixhin/PlaNet`. commit `28c8 491b c01e 8f1b 9113 0074 9e04 c308 c03d b051` (cited on page 91).

Asai, M. and A. Fukunaga (2018). 'Classical Planning in Deep Latent Space: Bridging the Subsymbolic-Symbolic Boundary'. In: *Proceedings of the Conference on Artificial Intelligence (AAAI)* (cited on page 100).

Avila Belbute-Peres, F. de, K. A. Smith, K. R. Allen, J. Tenenbaum, and J. Z. Kolter (2018). 'End-to-End Differentiable Physics for Learning and Control'. In: *Advances in Neural Information Processing Systems (NeurIPS)* (cited on page 81).

Bacon, P.-L., J. Harb, and D. Precup (2017). 'The Option-Critic Architecture'. In: *Proceedings of the Conference on Artificial Intelligence (AAAI)* (cited on page 99).

Bagaria, A. and G. Konidaris (2020). 'Option Discovery using Deep Skill Chaining'. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cited on page 99).

Banijamali, E., R. Shu, M. Ghavamzadeh, H. Bui, and A. Ghodsi (2018). 'Robust Locally-Linear Controllable Embedding'. In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)* (cited on page 31).

Baumli, K., D. Warde-Farley, S. Hansen, and V. Mnih (2021). 'Relative Variational Intrinsic Control'. In: *Proceedings of the Conference on Artificial Intelligence (AAAI)* (cited on page 99).

Becker, P., H. Pandya, G. Gebhardt, C. Zhao, C. J. Taylor, and G. Neumann (2019). 'Recurrent Kalman Networks: Factorized Inference in High-Dimensional Deep Feature Spaces'. In: *Proceedings of the International Conference on Machine Learning (ICML)* (cited on page 81).

Bellman, R. (1957a). 'A Markovian Decision Process'. In: *Journal of Mathematics and Mechanics* 6.5, pp. 679–684 (cited on pages 2, 3, 17).

– (1957b). *Dynamic Programming*. 1st ed. Princeton University Press (cited on pages 2, 21).

Betts, J. T. (1998). 'Survey of Numerical Methods for Trajectory Optimization'. In: *Journal of Guidance, Control, and Dynamics* 21.2, pp. 193–207 (cited on page 27).

Bishop, C. M. (2007). *Pattern recognition and machine learning*. 5th ed. Information science and statistics. Springer (cited on pages 15, 24).

Blank, P., B. H. Groh, and B. M. Eskofier (2017). 'Ball speed and spin estimation in table tennis using a racket-mounted inertial sensor'. In: *Proceedings of the ACM International Symposium on Wearable Computers (ISWC)* (cited on pages 81, 95, 126).

Blei, D. M., A. Kucukelbir, and J. D. McAuliffe (2016). 'Variational Inference: A Review for Statisticians'. In: *CoRR* abs/1601.00670 (cited on page 25).

Boer, P.-T. de, D. P. Kroese, S. Mannor, and R. Y. Rubinstein (2005). 'A Tutorial on the Cross-Entropy Method'. In: *Annals of Operations Research* 134.1, pp. 19–67 (cited on pages 27, 37).

Bosch, N., J. Achterhold, L. Leal-Taixé, and J. Stückler (2020). 'Planning from Images with Deep Latent Gaussian Process Dynamics'. In: *Proceedings of the Learning for Dynamics and Control Conference (L4DC)* (cited on pages 13, 14, 29, 31, 36, 40).

Brockman, G., V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba (2016). 'OpenAI Gym'. In: *CoRR* abs/1606.01540 (cited on pages 37, 44, 56, 58, 60, 143).

Büchler, D., R. Calandra, and J. Peters (2023). 'Learning to Control Highly Accelerated Ballistic Movements on Muscular Robots'. In: *Robotics and Autonomous Systems (RAS)* 159 (cited on page 94).

Büchler, D., H. Ott, and J. Peters (2016). 'A Lightweight Robotic Arm with Pneumatic Muscles for Robot Learning'. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (cited on pages 89, 94).

Buisson-Fenet, M., F. Solowjow, and S. Trimpe (2020). 'Actively Learning Gaussian Process Dynamics'. In: *Proceedings of the Learning for Dynamics and Control Conference (L4DC)* (cited on pages 9, 46).

Butcher, J. C. (2016). *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons, Ltd (cited on page 61).

Campeau-Lecours, A., H. Lamontagne, S. Latour, P. Fauteux, V. Maheu, F. Boucher, C. Deguire, and L.-J. C. L'Ecuyer (2017). 'Kinova Modular Robot Arms for Service Robotics Applications'.

In: *International Journal of Robotics Applications and Technologies (IJRAT)* 5.2, pp. 49–71 (cited on page 110).

Chaloner, K. and I. Verdinelli (1995). 'Bayesian Experimental Design: A Review'. In: *Statistical Science* 10.3, pp. 273–304 (cited on pages 12, 46, 54, 123).

Chang, X., P. Ren, P. Xu, Z. Li, X. Chen, and A. Hauptmann (2023). 'A Comprehensive Survey of Scene Graphs: Generation and Application'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.1, pp. 1–26 (cited on page 126).

Chebotar, Y., K. Hausman, Y. Lu, T. Xiao, D. Kalashnikov, J. Varley, A. Irpan, B. Eysenbach, R. Julian, C. Finn, and S. Levine (2021). 'Actionable Models: Unsupervised Offline Reinforcement Learning of Robotic Skills'. In: *Proceedings of the International Conference on Machine Learning, (ICML)* (cited on page 126).

Chitnis, R., T. Silver, J. B. Tenenbaum, T. Lozano-Perez, and L. P. Kaelbling (2022). 'Learning Neuro-Symbolic Relational Transition Models for Bilevel Planning'. In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)* (cited on page 100).

Cho, K., B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio (2014). 'Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation'. In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)* (cited on pages 52, 69, 81, 131).

Chua, K., R. Calandra, R. McAllister, and S. Levine (2018). 'Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models'. In: *Advances in Neural Information Processing Systems (NeurIPS)* (cited on pages 6, 7, 20, 28, 30, 45).

Chung, J., K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio (2015). 'A Recurrent Latent Variable Model for Sequential Data'. In: *Advances in Neural Information Processing Systems (NeurIPS)* (cited on page 81).

Cover, T. M. and J. A. Thomas (2006). *Elements of information theory*. 2nd ed. Wiley (cited on page 25).

Dayan, P. and G. E. Hinton (1992). 'Feudal Reinforcement Learning'. In: *Advances in Neural Information Processing Systems (NeurIPS)* (cited on pages 11, 99).

Deisenroth, M. P. (2010). 'Efficient Reinforcement Learning using Gaussian Processes'. PhD thesis. Karlsruher Institut für Technologie (cited on page 20).

Deisenroth, M. P., A. McHutchon, J. Hall, and C. E. Rasmussen (2013). *PILCO Code Documentation v0.9*. `http://mlg.eng.cam.ac.uk/pilco/release/pilcodocV0.9.pdf` (cited on page 129).

Deisenroth, M. P. and C. E. Rasmussen (2011). 'PILCO: A Model-Based and Data-Efficient Approach to Policy Search'. In: *Proceedings of the International Conference on Machine Learning (ICML)* (cited on pages 5–8, 30, 31, 41, 43, 45, 67).

Deisenroth, M. P., C. E. Rasmussen, and J. Peters (2009). 'Gaussian process dynamic programming'. In: *Neurocomputing* 72.7-9, pp. 1508–1524 (cited on page 31).

Dietterich, T. G. (2000). 'Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition'. In: *Journal of Artificial Intelligence Research* 13, pp. 227–303 (cited on page 99).

Dittrich, A., J. Schneider, S. Guist, B. Schölkopf, and D. Büchler (2022). 'AIMY: An Open-source Table Tennis Ball Launcher for Versatile and High-fidelity Trajectory Generation'. In: *CoRR* abs/2210.06048 (cited on page 89).

Duan, Y., J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel (2016). 'RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning'. In: *CoRR* abs/1611.02779 (cited on page 45).

Epshteyn, A., A. Vogel, and G. DeJong (2008). 'Active reinforcement learning'. In: *Proceedings of the International Conference on Machine Learning (ICML)* (cited on page 46).

Evans, J. S. B. T. (1984). 'Heuristic and analytic processes in reasoning*'. In: *British Journal of Psychology* 75.4, pp. 451–468 (cited on page 1).

Eysenbach, B., A. Gupta, J. Ibarz, and S. Levine (2019). 'Diversity is All You Need: Learning Skills without a Reward Function'. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cited on page 99).

Finn, C., P. Abbeel, and S. Levine (2017). 'Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks'. In: *Proceedings of the International Conference on Machine Learning (ICML)* (cited on pages 8, 44, 45).

Finn, C., I. Goodfellow, and S. Levine (2016). 'Unsupervised Learning for Physical Interaction through Video Prediction'. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Barcelona, Spain (cited on page 67).

Florensa, C., Y. Duan, and P. Abbeel (2017). 'Stochastic Neural Networks for Hierarchical Reinforcement Learning'. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cited on page 99).

Foster, A., M. Jankowiak, E. Bingham, P. Horsfall, Y. W. Teh, T. Rainforth, and N. D. Goodman (2019). 'Variational Bayesian Optimal Experimental Design'. In: *Advances in Neural Information Processing Systems (NeurIPS)* (cited on page 48).

Fraccaro, M., S. Kamronn, U. Paquet, and O. Winther (2017). 'A Disentangled Recognition and Nonlinear Dynamics Model for Unsupervised Learning'. In: *Advances in Neural Information Processing Systems (NeurIPS)* (cited on pages 31, 45, 81).

Fraccaro, M., S. K. Sønderby, U. Paquet, and O. Winther (2016). 'Sequential Neural Models with Stochastic Layers'. In: *Advances in Neural Information Processing Systems (NeurIPS)* (cited on page 81).

Gal, Y., R. McAllister, and C. E. Rasmussen (2016). 'Improving PILCO with Bayesian neural network dynamics models'. In: *Data-Efficient Machine Learning workshop, International Conference on Machine Learning (ICML)* (cited on page 31).

Gao, Y., J. Tebbe, and A. Zell (2021). 'Robust Stroke Recognition via Vision and IMU in Robotic Table Tennis'. In: *Proceedings of the International Conference on Artificial Neural Networks (ICANN)* (cited on pages 95, 126).

García, C. E., D. M. Prett, and M. Morari (1989). 'Model predictive control: Theory and practice — A survey'. In: *Automatica* 25.3, pp. 335–348 (cited on pages 20, 37).

Gardner, J. R., G. Pleiss, K. Q. Weinberger, D. Bindel, and A. G. Wilson (2018). 'GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration'. In: *Advances in Neural Information Processing Systems (NeurIPS)* (cited on page 38).

Garnelo, M., D. Rosenbaum, C. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. J. Rezende, and S. M. A. Eslami (2018a). 'Conditional Neural Processes'. In: *Proceedings of the International Conference on Machine Learning (ICML)* (cited on pages 45, 52).

Garnelo, M., J. Schwarz, D. Rosenbaum, F. Viola, D. J. Rezende, S. M. A. Eslami, and Y. W. Teh (2018b). 'Neural Processes'. In: *CoRR* abs/1807.01622 (cited on pages 8, 44, 45, 48–50, 53, 63, 68, 69, 123).

Garrett, C. R., R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez (2021). 'Integrated Task and Motion Planning'. In: *Annual Review of Control, Robotics, and Autonomous Systems* 4.1, pp. 265–293 (cited on page 97).

Gershman, S. and N. D. Goodman (2014). 'Amortized Inference in Probabilistic Reasoning'. In: *Proceedings of the Annual Meeting of the Cognitive Science Society (CogSci)* (cited on page 26).

Girin, L., S. Leglaive, X. Bie, J. Diard, T. Hueber, and X. Alameda-Pineda (2021). 'Dynamical Variational Autoencoders: A Comprehensive Review'. In: *Foundations and Trends in Machine Learning* 15.1-2, pp. 1–175 (cited on page 81).

Golovin, D., A. Krause, and D. Ray (2010). 'Near-Optimal Bayesian Active Learning with Noisy Observations'. In: *Advances in Neural Information Processing Systems (NeurIPS)* (cited on page 46).

Gómez-González, S. (2022). *Trajectory Forecasting using Deep Conditional Generative Models*. `https://github.com/sebasutp/trajectory_forcasting`. commit `39ae 9368 63eb e525 82cd 2c91 e19e 5f11 b514 3db1` (cited on page 91).

Gómez-González, S., Y. Nemmour, B. Schölkopf, and J. Peters (2019). 'Reliable Real-Time Ball Tracking for Robot Table Tennis'. In: *Robotics* 8.4 (cited on page 89).

Gómez-González, S., S. Prokudin, B. Schölkopf, and J. Peters (2020). 'Real Time Trajectory Prediction Using Deep Conditional Generative Models'. In: *IEEE Robotics and Automation Letters (RA-L)* 5.2, pp. 970–976 (cited on pages 81, 91, 92).

Gossard, T., J. Tebbe, A. Ziegler, and A. Zell (2023). 'SpinDOE: A ball spin estimation method for table tennis robot'. In: *CoRR* abs/2303.03879 (cited on pages 95, 126).

Gregor, K., D. J. Rezende, and D. Wierstra (2017). 'Variational Intrinsic Control'. In: *International Conference on Learning Representations (ICLR), Workshop Track Proceedings* (cited on pages 99, 103, 104, 115, 145, 146).

Grigorescu, S., C. Ginerica, M. Zaha, G. Macesanu, and B. Trasnea (2021). 'LVD-NMPC: A learning-based vision dynamics approach to nonlinear model predictive control for autonomous vehicles'. In: *International Journal of Advanced Robotic Systems* 18.3 (cited on page 67).

Guan, L., S. Sreedharan, and S. Kambhampati (2022). 'Leveraging Approximate Symbolic Models for Reinforcement Learning via Skill Diversity'. In: *Proceedings of the International Conference on Machine Learning (ICML)* (cited on page 100).

Guttikonda, S., J. Achterhold, H. Li, J. Boedecker, and J. Stueckler (2023). 'Context-Conditional Navigation with a Learning-Based Terrain- and Robot-Aware Dynamics Model'. In: *Proceedings of the IEEE European Conference on Mobile Robots (ECMR)* (cited on pages 13, 14, 65, 73, 74, 78).

Ha, D. and J. Schmidhuber (2018). 'Recurrent World Models Facilitate Policy Evolution'. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Curran Associates, Inc., pp. 2451–2463 (cited on page 123).

Haarnoja, T., A. Zhou, P. Abbeel, and S. Levine (2018). 'Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor'. In: *Proceedings of the International Conference on Machine Learning (ICML)* (cited on pages 6, 22, 23, 106, 112).

Hafner, D., T. P. Lillicrap, J. Ba, and M. Norouzi (2020). 'Dream to Control: Learning Behaviors by Latent Imagination'. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cited on pages 6, 7, 41, 123, 124).

Hafner, D., T. P. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson (2019). 'Learning Latent Dynamics for Planning from Pixels'. In: *Proceedings of the International Conference on Machine Learning (ICML)* (cited on pages 6, 7, 13, 20, 28, 30, 31, 35, 38, 39, 41, 45, 67, 78, 81, 91, 92, 123, 129).

Hartikainen, J. and S. Särkkä (2010). 'Kalman filtering and smoothing solutions to temporal Gaussian process regression models'. In: *IEEE International Workshop on Machine Learning for Signal Processing* (cited on page 83).

Hasselt, H. van (2010). 'Double Q-learning'. In: *Advances in Neural Information Processing Systems (NeurIPS)* (cited on page 22).

– (2012). 'Reinforcement Learning in Continuous State and Action Spaces'. In: *Reinforcement Learning: State-of-the-Art*. Ed. by M. Wiering and M. van Otterlo. Springer, pp. 207–251 (cited on page 18).

Hasselt, H. van, A. Guez, and D. Silver (2016). 'Deep Reinforcement Learning with Double Q-Learning'. In: *Proceedings of the Conference on Artificial Intelligence (AAAI)* (cited on page 22).

Hayashi, K., M. Imaizumi, and Y. Yoshida (2020). 'On Random Subsampling of Gaussian Process Regression: A Graphon-Based Analysis'. In: *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)* (cited on page 36).

Higgins, I., L. Matthey, A. Pal, C. P. Burgess, X. Glorot, M. M. Botvinick, S. Mohamed, and A. Lerchner (2017). 'beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework'. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cited on page 53).

Hochreiter, S. and J. Schmidhuber (1997). 'Long Short-Term Memory'. In: *Neural Computation* 9.8, pp. 1735–1780 (cited on page 81).

Hospedales, T. M., A. Antoniou, P. Micaelli, and A. J. Storkey (2022). 'Meta-Learning in Neural Networks: A Survey'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.9, pp. 5149–5169 (cited on page 8).

Illanes, L., X. Yan, R. T. Icarte, and S. A. McIlraith (2020). 'Symbolic Plans as High-Level Instructions for Reinforcement Learning'. In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)* (cited on page 100).

Iverson, K. E. (1962). 'A programming language'. In: *Proceedings of the spring joint computer conference (AFIPS)* (cited on page xviii).

James, S., B. Rosman, and G. Konidaris (2020). 'Learning Portable Representations for High-Level Planning'. In: *Proceedings of the International Conference on Machine Learning (ICML)* (cited on page 100).

Jang, E., S. Gu, and B. Poole (2017). 'Categorical Reparameterization with Gumbel-Softmax'. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cited on page 144).

Jazwinski, A. H. (1970). *Stochastic processes and filtering theory*. Academic Press New York (cited on page 83).

Jervis, T. T. and F. Fallside (1992). 'Pole Balancing on a Real Rig Using a Reinforcement Learning Controller'. In: *Technical Report CUED/F-INFENG/TR 115* (cited on page 5).

Kaelbling, L. P., M. L. Littman, and A. R. Cassandra (1998). 'Planning and Acting in Partially Observable Stochastic Domains'. In: *Artificial Intelligence* 101.1-2, pp. 99–134 (cited on page 19).

Kahn, G., P. Abbeel, and S. Levine (2021). 'BADGR: An Autonomous Self-Supervised Learning-Based Navigation System'. In: *IEEE Robotics and Automation Letters* (cited on page 67).

Kahneman, D. (2003). 'A perspective on judgment and choice: Mapping bounded rationality.' In: *American Psychologist* 58.9, pp. 697–720 (cited on page 1).

Kalman, R. E. (1960). 'A New Approach to Linear Filtering and Prediction Problems'. In: *Journal of Basic Engineering* 82.1, pp. 35–45 (cited on page 83).

Kandukuri, R. K., J. Achterhold, M. Möller, and J. Stueckler (2020). 'Learning to Identify Physical Parameters from Video Using Differentiable Physics'. In: *Proceedings of the German Conference on Pattern Recognition (GCPR)* (cited on pages 13, 14).

– (2022). 'Physical Representation Learning and Parameter Identification from Video Using Differentiable Physics'. In: *International Journal of Computer Vision* 130.1, pp. 3–16 (cited on pages 13, 14).

Karl, M., M. Soelch, J. Bayer, and P. van der Smagt (2017). 'Deep Variational Bayes Filters: Unsupervised Learning of State Space Models from Raw Data'. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cited on pages 31, 81).

Killian, T. W., S. Daulton, F. Doshi-Velez, and G. D. Konidaris (2017). 'Robust and Efficient Transfer Learning with Hidden Parameter Markov Decision Processes'. In: *Advances in Neural Information Processing Systems (NeurIPS)* (cited on page 31).

Kingma, D. P. and J. Ba (2015). 'Adam: A Method for Stochastic Optimization'. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cited on pages 38, 57, 88, 106).

Kingma, D. P. and M. Welling (2014). 'Auto-Encoding Variational Bayes'. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cited on pages 26, 27, 36, 37, 48).

Klein-Soetebier, T., B. Noël, and S. Klatt (2020). 'Multimodal perception in table tennis: the effect of auditory and visual information on anticipation and planning of action'. In: *International Journal of Sport and Exercise Psychology* 19, pp. 834–847 (cited on pages 95, 125).

Ko, J., D. J. Klein, D. Fox, and D. Hähnel (2007). 'Gaussian Processes and Reinforcement Learning for Identification and Control of an Autonomous Blimp'. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (cited on page 31).

Kober, J. and J. Peters (2012). 'Reinforcement Learning in Robotics: A Survey'. In: *Reinforcement Learning*. Vol. 12. Springer, pp. 579–610 (cited on page 3).

Koç, O., G. Maeda, and J. Peters (2018). 'Online optimal trajectory generation for robot table tennis'. In: *Robotics and Autonomous Systems (RAS)* 105, pp. 121–137 (cited on page 81).

Kochenderfer, M. J. (2015). *Decision Making Under Uncertainty: Theory and Application*. The MIT Press (cited on page 19).

Kokel, H., A. Manoharan, S. Natarajan, B. Ravindran, and P. Tadepalli (2021). 'RePReL: Integrating Relational Planning and Reinforcement Learning for Effective Abstraction'. In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)* (cited on page 100).

Konda, V. R. and J. N. Tsitsiklis (1999). 'Actor-Critic Algorithms'. In: *Advances in Neural Information Processing Systems (NeurIPS)* (cited on page 23).

Konidaris, G. (2019). 'On the necessity of abstraction'. In: *Current Opinion in Behavioral Sciences* 29, pp. 1–7 (cited on page 98).

Konidaris, G., L. P. Kaelbling, and T. Lozano-Perez (2018). 'From Skills to Symbols: Learning Symbolic Representations for Abstract High-Level Planning'. In: *Journal of Artificial Intelligence Research* 61.1, pp. 215–289 (cited on page 100).

Kroemer, O., S. Niekum, and G. Konidaris (2021). 'A Review of Robot Learning for Manipulation: Challenges, Representations, and Algorithms'. In: *Journal of Machine Learning Research* 22, pp. 1–82 (cited on page 3).

Kuhn, H. W. (1955). 'The Hungarian method for the assignment problem'. In: *Naval research logistics quarterly* 2.1, pp. 83–97 (cited on page 105).

Kulkarni, K. M. and S. Shenoy (2021). 'Table Tennis Stroke Recognition Using Two-Dimensional Human Pose Estimation'. In: *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPR Workshops)* (cited on pages 95, 126).

Le, T. A., H. Kim, M. Garnelo, D. Rosenbaum, J. Schwarz, and Y. W. Teh (2018). 'Empirical Evaluation of Neural Process Objectives'. In: *Third Workshop on Bayesian Deep Learning* (cited on pages 49, 53, 69).

Lee, K., Y. Seo, S. Lee, H. Lee, and J. Shin (2020). 'Context-aware Dynamics Model for Generalization in Model-Based Reinforcement Learning'. In: *Proceedings of the International Conference on Machine Learning (ICML)* (cited on pages 8, 44, 45, 68).

Lenz, I., R. A. Knepper, and A. Saxena (2015). 'DeepMPC: Learning Deep Latent Features for Model Predictive Control'. In: *Robotics: Science and Systems* (cited on page 67).

Lesort, T., N. D. Rodríguez, J.-F. Goudou, and D. Filliat (2018). 'State representation learning for control: An overview'. In: *Neural Networks* 108, pp. 379–392 (cited on page 31).

Levine, S., A. Kumar, G. Tucker, and J. Fu (2020). 'Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems'. In: *CoRR* abs/2005.01643 (cited on page 126).

Levy, A. (2020). *Hierarchical-Actor-Critc-HAC-*. `https://github.com/andrew-j-levy/Hierarchical-Actor-Critc-HAC-`. commit `e240 2577 991d 3522 206e c40e 3dc9 5e48 5f15 97b7` (cited on page 144).

Levy, A., G. D. Konidaris, R. P. Jr., and K. Saenko (2019). 'Learning Multi-Level Hierarchies with Hindsight'. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cited on pages 99, 106, 112).

Li, A. C., C. Florensa, I. Clavera, and P. Abbeel (2020). 'Sub-policy Adaptation for Hierarchical Reinforcement Learning'. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cited on page 99).

Li, H., H. Wu, L. Lou, K. Kühnlenz, and O. Ravn (2012). 'Ping-pong robotics with high-speed vision system'. In: *Proceedings of the IEEE International Conference on Control, Automation, Robotics & Vision (ICARCV)* (cited on page 81).

Li, S., L. Zheng, J. Wang, and C. Zhang (2021). 'Learning Subgoal Representations with Slow Dynamics'. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cited on page 126).

Li, Z., A. Narayan, and T.-Y. Leong (2017). 'An Efficient Approach to Model-Based Hierarchical Reinforcement Learning'. In: *Proceedings of the Conference on Artificial Intelligence (AAAI)* (cited on page 99).

Lillicrap, T. P., J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra (2016). 'Continuous control with deep reinforcement learning'. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cited on pages 22, 23, 31).

Lin, H.-I., Z. Yu, and Y.-C. Huang (2020). 'Ball Tracking and Trajectory Prediction for Table-Tennis Robots'. In: *Sensors* 20.2 (cited on page 81).

Lindley, D. V. (1956). 'On a Measure of the Information Provided by an Experiment'. In: *The Annals of Mathematical Statistics* 27.4, pp. 986–1005 (cited on pages 12, 46, 47, 54, 123).

Liu, H., Y.-S. Ong, X. Shen, and J. Cai (2020). 'When Gaussian Process Meets Big Data: A Review of Scalable GPs'. In: *IEEE Transactions on Neural Networks and Learning Systems* 31.11, pp. 4405–4423 (cited on page 36).

Ljung, L. (1986). *System Identification: Theory for the User*. Prentice-Hall, Inc. (cited on pages 6, 80).

Luxburg, U. von and B. Schölkopf (2011). 'Statistical Learning Theory: Models, Concepts, and Results'. In: *Handbook of the History of Logic, Vol. 10: Inductive Logic*. Elsevier North Holland, pp. 651–706 (cited on page 10).

Lyu, D., F. Yang, B. Liu, and S. Gustafson (2019). 'SDRL: Interpretable and Data-Efficient Deep Reinforcement Learning Leveraging Symbolic Planning'. In: *Proceedings of the Conference on Artificial Intelligence (AAAI)* (cited on page 100).

Ma, H., D. Büchler, B. Schölkopf, and M. Muehlebach (2022). 'A Learning-based Iterative Control Framework for Controlling a Robot Arm with Pneumatic Artificial Muscles'. In: *Robotics: Science and Systems* (cited on page 94).

Macario Barros, A., M. Michel, Y. Moline, G. Corre, and F. Carrel (2022). 'A Comprehensive Survey of Visual SLAM Algorithms'. In: *Robotics* 11.1 (cited on pages 78, 125).

MacKay, D. J. C. (1992). 'Information-Based Objective Functions for Active Data Selection'. In: *Neural Computation* 4.4, pp. 590–604 (cited on page 46).

Maddison, C. J., A. Mnih, and Y. W. Teh (2017). 'The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables'. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cited on page 144).

Mann, H. B. and D. R. Whitney (1947). 'On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other'. In: *The Annals of Mathematical Statistics* 18.1, pp. 50–60 (cited on page 115).

Marco, A., F. Berkenkamp, P. Hennig, A. P. Schoellig, A. Krause, S. Schaal, and S. Trimpe (2017). 'Virtual vs. Real: Trading Off Simulations and Physical Experiments in Reinforcement Learning

with Bayesian Optimization'. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (cited on page 125).

Matsushima, M., T. Hashimoto, M. Takeuchi, and F. Miyazaki (2005). 'A Learning Approach to Robotic Table Tennis'. In: *IEEE Transactions on Robotics (T-RO)* 21.4, pp. 767–771 (cited on page 81).

Mees, O., L. Hermann, E. Rosete-Beas, and W. Burgard (2022). 'CALVIN: A Benchmark for Language-Conditioned Policy Learning for Long-Horizon Robot Manipulation Tasks'. In: *IEEE Robotics and Automation Letters (RA-L)* 7.3, pp. 7327–7334 (cited on page 126).

Minsky, M. (1961). 'Steps toward Artificial Intelligence'. In: *Proceedings of the IRE* 49.1, pp. 8–30 (cited on page 4).

Mirza, M., A. Jaegle, J. J. Hunt, A. Guez, S. Tunyasuvunakool, A. Muldal, T. Weber, P. Karkus, S. Racanière, L. Buesing, T. P. Lillicrap, and N. Heess (2020). 'Physically Embedded Planning Problems: New Challenges for Reinforcement Learning'. In: *CoRR* abs/2009.05524 (cited on pages 97, 98, 109, 124).

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill New York (cited on page 23).

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis (2015). 'Human-level control through deep reinforcement learning'. In: *Nature* 518.7540, pp. 529–533 (cited on pages 2, 6, 31).

Moerland, T. M., J. Broekens, A. Plaat, and C. M. Jonker (2023). 'Model-based Reinforcement Learning: A Survey'. In: *Foundations and Trends in Machine Learning* 16.1, pp. 1–118 (cited on pages 1, 6, 20).

Moore, A. (1990). 'Efficient Memory-based Learning for Robot Control'. PhD thesis. Trinity Hall, Cambridge (cited on pages 44, 61).

Mosavi, A., Y. Faghan, P. Ghamisi, P. Duan, S. F. Ardabili, E. Salwana, and S. S. Band (2020). 'Comprehensive Review of Deep Reinforcement Learning Methods and Applications in Economics'. In: *Mathematics* 8.10 (cited on page 2).

Mülling, K., J. Kober, and J. Peters (2010). 'Simulating Human Table Tennis with a Biomimetic Robot Setup'. In: *Proceedings of the International Conference on Simulation of Adaptive Behavior (SAB)* (cited on page 81).

Munkres, J. (1957). 'Algorithms for the assignment and transportation problems'. In: *Journal of the society for industrial and applied mathematics* 5.1, pp. 32–38 (cited on page 105).

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press (cited on pages 15, 53).

Nachum, O., S. Gu, H. Lee, and S. Levine (2018). 'Data-Efficient Hierarchical Reinforcement Learning'. In: *Advances in Neural Information Processing Systems (NeurIPS)* (cited on page 99).

Nagabandi, A., G. Kahn, R. S. Fearing, and S. Levine (2018). 'Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning'. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (cited on page 68).

Nagabandi, A., I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn (2019). 'Learning to Adapt in Dynamic, Real-World Environments through Meta-Reinforcement Learning'. In:

Proceedings of the International Conference on Learning Representations (ICLR) (cited on pages 8, 45).

Nakashima, A., Y. Ogawa, Y. Kobayashi, and Y. Hayakawa (2010). 'Modeling of rebound phenomenon of a rigid ball with friction and elastic effects'. In: *Proceedings of the American Control Conference (ACC)* (cited on page 81).

Nelles, O. (2001). *Nonlinear System Identification*. Springer (cited on pages 10, 123).

Oh, J., X. Guo, H. Lee, R. L. Lewis, and S. Singh (2015). 'Action-Conditional Video Prediction using Deep Networks in Atari Games'. In: *Advances in Neural Information Processing Systems (NeurIPS)* (cited on page 67).

Otterlo, M. van and M. Wiering (2012). 'Reinforcement Learning and Markov Decision Processes'. In: *Reinforcement Learning: State-of-the-Art*. Ed. by M. Wiering and M. van Otterlo. Springer, pp. 3–42 (cited on page 17).

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala (2019). 'PyTorch: An Imperative Style, High-Performance Deep Learning Library'. In: *Advances in Neural Information Processing Systems (NeurIPS)* (cited on page 38).

Pinneri, C., S. Sawant, S. Blaes, J. Achterhold, J. Stueckler, M. Rolínek, and G. Martius (2020). 'Sample-efficient Cross-Entropy Method for Real-time Planning'. In: *Proceedings of the Conference on Robot Learning (CoRL)* (cited on pages 13, 76, 124).

Placed, J. A., J. Strader, H. Carrillo, N. Atanasov, V. Indelman, L. Carlone, and J. A. Castellanos (2023). 'A Survey on Active Simultaneous Localization and Mapping: State of the Art and New Frontiers'. In: *IEEE Transactions on Robotics* 39.3, pp. 1686–1705 (cited on pages 78, 125).

Pukelsheim, F. (2006). *Optimal Design of Experiments*. Society for Industrial and Applied Mathematics (cited on page 46).

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley (cited on pages 3, 17).

Rasmussen, C. E. and M. Kuss (2003). 'Gaussian Processes in Reinforcement Learning'. In: *Advances in Neural Information Processing Systems (NeurIPS)* (cited on page 31).

Rasmussen, C. E. and C. K. I. Williams (2006). *Gaussian processes for machine learning*. MIT Press (cited on pages 31–33, 36, 37, 50).

Rezende, D. J., S. Mohamed, and D. Wierstra (2014). 'Stochastic Backpropagation and Approximate Inference in Deep Generative Models'. In: *Proceedings of the International Conference on Machine Learning (ICML)* (cited on pages 36, 37).

Riedmiller, M. A., R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. V. de Wiele, V. Mnih, N. Heess, and J. T. Springenberg (2018). 'Learning by Playing Solving Sparse Reward Tasks from Scratch'. In: *Proceedings of the International Conference on Machine Learning (ICML)* (cited on page 99).

Rosete-Beas, E., O. Mees, G. Kalweit, J. Boedecker, and W. Burgard (2022). 'Latent Plans for Task-Agnostic Offline Reinforcement Learning'. In: *Proceedings of the Conference on Robot Learning (CoRL)* (cited on page 126).

Roy, N. and A. McCallum (2001). 'Toward Optimal Active Learning through Sampling Estimation of Error Reduction'. In: *Proceedings of the International Conference on Machine Learning (ICML)* (cited on page 46).

Rubinstein, R. Y. (1996). 'Optimization of Computer Simulation Models with Rare Events'. In: *European Journal of Operations Research* 99, pp. 89–112 (cited on pages 27, 37).

– (1999). 'The Cross-Entropy Method for Combinatorial and Continuous Optimization'. In: *Methodology And Computing In Applied Probability* 1.2, pp. 127–190 (cited on pages 13, 27, 45, 55, 71, 133).

Russell, S. and P. Norvig (2009). *Artificial Intelligence: A Modern Approach*. 3rd ed. Prentice Hall Press (cited on page 3).

Ryan, E. G., C. C. Drovandi, J. M. McGree, and A. N. Pettitt (2016). 'A Review of Modern Computational Algorithms for Bayesian Optimal Design'. In: *International Statistical Review* 84.1, pp. 128–154 (cited on pages 46, 47).

Ryan, M. R. K. (2002). 'Using Abstract Models of Behaviours to Automatically Generate Reinforcement Learning Hierarchies'. In: *Proceedings of the International Conference on Machine Learning (ICML)* (cited on page 100).

Rybkin, O., C. Zhu, A. Nagabandi, K. Daniilidis, I. Mordatch, and S. Levine (2021). 'Model-Based Reinforcement Learning via Latent-Space Collocation'. In: *Proceedings of the International Conference on Machine Learning (ICML)* (cited on page 27).

Sæmundsson, S., K. Hofmann, and M. P. Deisenroth (2018). 'Meta Reinforcement Learning with Latent Variable Gaussian Processes'. In: *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)* (cited on pages 9, 31, 41, 45, 46, 125).

Särkkä, S. (2013). *Bayesian Filtering and Smoothing*. Cambridge University Press (cited on pages 83, 85, 88).

Schulman, J., S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz (2015). 'Trust Region Policy Optimization'. In: *Proceedings of the International Conference on Machine Learning (ICML)* (cited on page 6).

Schulman, J., F. Wolski, P. Dhariwal, A. Radford, and O. Klimov (2017). 'Proximal Policy Optimization Algorithms'. In: *CoRR* abs/1707.06347 (cited on pages 6, 23).

Schwenzer, M., M. Ay, T. Bergs, and D. Abel (2021). 'Review on model predictive control: an engineering perspective'. In: *The International Journal of Advanced Manufacturing Technology* 117.5, pp. 1327–1349 (cited on page 20).

Sekar, R., O. Rybkin, K. Daniilidis, P. Abbeel, D. Hafner, and D. Pathak (2020). 'Planning to Explore via Self-Supervised World Models'. In: *Proceedings of the International Conference on Machine Learning (ICML)* (cited on pages 9, 11, 46).

Shaj, V., P. Becker, D. Büchler, H. Pandya, N. van Duijkeren, C. J. Taylor, M. Hanheide, and G. Neumann (2020). 'Action-Conditional Recurrent Kalman Networks For Forward and Inverse Dynamics Learning'. In: *Proceedings of the Conference on Robot Learning (CoRL)* (cited on page 68).

Sharma, A., S. Gu, S. Levine, V. Kumar, and K. Hausman (2020). 'Dynamics-Aware Unsupervised Discovery of Skills'. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cited on pages 11, 99, 103).

Al-Shedivat, M., T. Bansal, Y. Burda, I. Sutskever, I. Mordatch, and P. Abbeel (2018). 'Continuous Adaptation via Meta-Learning in Nonstationary and Competitive Environments'. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cited on page 31).

Shyam, P., W. Jaskowski, and F. Gomez (2019). 'Model-Based Active Exploration'. In: *Proceedings of the International Conference on Machine Learning (ICML)* (cited on page 46).

Sikand, K. S., S. Rabiee, A. Uccello, X. Xiao, G. Warnell, and J. Biswas (2022). 'Visual Representation Learning for Preference-Aware Path Planning'. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (cited on page 67).

Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis (2016). 'Mastering the game of Go with deep neural networks and tree search'. In: *Nature* 529.7587, pp. 484–489 (cited on pages 2, 6, 7, 10, 97).

Silver, D., G. Lever, N. Heess, T. Degris, D. Wierstra, and M. A. Riedmiller (2014). 'Deterministic Policy Gradient Algorithms'. In: *Proceedings of the International Conference on Machine Learning (ICML)* (cited on page 23).

Silver, T., A. Athalye, J. B. Tenenbaum, T. Lozano-Perez, and L. P. Kaelbling (2022). 'Learning Neuro-Symbolic Skills for Bilevel Planning'. In: *Proceedings of the Conference on Robot Learning (CoRL)* (cited on page 100).

Singh, G., J. Yoon, Y. Son, and S. Ahn (2019). 'Sequential Neural Processes'. In: *Advances in Neural Information Processing Systems (NeurIPS)* (cited on page 45).

Siva, S., M. B. Wigness, J. G. Rogers, and H. Zhang (2021). 'Enhancing Consistent Ground Maneuverability by Robot Adaptation to Complex Off-Road Terrains'. In: *Proceedings of the Conference on Robot Learning (CoRL)* (cited on page 67).

Skogestad, S. and I. Postlethwaite (2005). *Multivariable Feedback Control: Analysis and Design*. John Wiley & Sons, Inc. (cited on page 5).

Snelson, E. L. and Z. Ghahramani (2005). 'Sparse Gaussian Processes using Pseudo-inputs'. In: *Advances in Neural Information Processing Systems (NeurIPS)* (cited on pages 40, 125).

Sonker, R. and A. Dutta (2021). 'Adding Terrain Height to Improve Model Learning for Path Tracking on Uneven Terrain by a Four Wheel Robot'. In: *IEEE Robotics and Automation Letters* (cited on pages 10, 66).

Spaan, M. T. J. (2012). 'Partially Observable Markov Decision Processes'. In: *Reinforcement Learning: State-of-the-Art*. Ed. by M. Wiering and M. van Otterlo. Springer, pp. 387–414 (cited on page 19).

Stuhlmüller, A., J. Taylor, and N. D. Goodman (2013). 'Learning Stochastic Inverses'. In: *Advances in Neural Information Processing Systems (NeurIPS)* (cited on page 26).

Sutton, R. S. (1991). 'Dyna, an Integrated Architecture for Learning, Planning, and Reacting'. In: *SIGART Bull.* 2.4, pp. 160–163 (cited on pages 1, 7, 41).

Sutton, R. S. and A. G. Barto (2018). *Reinforcement learning: An introduction*. 2nd ed. The MIT Press (cited on pages 1–3, 9, 18, 21, 22).

Sutton, R. S., D. Precup, and S. P. Singh (1999). 'Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning'. In: *Artificial Intelligence* 112.1-2, pp. 181–211 (cited on pages 11, 99, 101).

Tandon, P. (2021). *pytorch-soft-actor-critic*. `https://github.com/pranz24/pytorch-soft-actor-critic`. commit `3985 95e0 d9dc a98b 7db7 8c7f 2f93 9c96 9431 871a` (cited on pages 140, 144).

Tassa, Y., Y. Doron, A. Muldal, T. Erez, Y. Li, D. de Las Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, T. P. Lillicrap, and M. A. Riedmiller (2018). 'DeepMind Control Suite'. In: *CoRR* abs/1801.00690 (cited on pages 40, 97, 109).

Tebbe, J., Y. Gao, M. Sastre-Rienietz, and A. Zell (2018). 'A Table Tennis Robot System Using an Industrial KUKA Robot Arm'. In: *Proceedings of the German Conference on Pattern Recognition (GCPR)* (cited on page 81).

Tedrake, R. (2023). *Underactuated Robotics. Algorithms for Walking, Running, Swimming, Flying, and Manipulation*. Course Notes for MIT 6.832 (cited on pages 27, 28).

Thrun, S. and L. Pratt (1998). 'Learning to Learn: Introduction and Overview'. In: *Learning to Learn*. Ed. by S. Thrun and L. Pratt. Springer US, pp. 3–17 (cited on pages 8, 44).

Toro Icarte, R., E. Waldie, T. Klassen, R. Valenzano, M. Castro, and S. McIlraith (2019). 'Learning Reward Machines for Partially Observable Reinforcement Learning'. In: *Advances in Neural Information Processing Systems (NeurIPS)* (cited on page 100).

Tschantz, A., B. Millidge, A. K. Seth, and C. L. Buckley (2020). 'Reinforcement Learning through Active Inference'. In: *CoRR* abs/2002.12636 (cited on page 46).

Turing, A. M. (1950). 'Computing Machinery and Intelligence'. In: *Mind* 59.236, pp. 433–460 (cited on page 3).

uArm-Developer (2021). *uArm-Python-SDK*. `https://github.com/uArm-Developer/uArm-Python-SDK`. commit `f553 3d11 c12a 2966 b7af 462f 1c09 7aac e0e9 0598` (cited on page 143).

Ugur, E. and J. Piater (2015). 'Bottom-up learning of object categories, action effects and logical rules: From continuous manipulative exploration to symbolic planning'. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (cited on page 100).

Valada, A., J. Vertens, A. Dhall, and W. Burgard (2017). 'AdapNet: Adaptive semantic segmentation in adverse environmental conditions'. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (cited on page 67).

Vapnik, V. (1991). 'Principles of Risk Minimization for Learning Theory'. In: *Advances in Neural Information Processing Systems (NeurIPS)* (cited on page 24).

Vinyals, O., I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, Ç. Gülçehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. P. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver (2019). 'Grandmaster level in StarCraft II using multi-agent reinforcement learning'. In: *Nature* 575.7782, pp. 350–354 (cited on pages 2, 97).

Volpp, M., F. Flürenbrock, L. Großberger, C. Daniel, and G. Neumann (2021). 'Bayesian Context Aggregation for Neural Processes'. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cited on pages 49, 53).

Wang, J. M., D. J. Fleet, and A. Hertzmann (2005). 'Gaussian Process Dynamical Models'. In: *Advances in Neural Information Processing Systems (NeurIPS)* (cited on page 31).

Wang, Q., K. Zhang, and D. Wang (2014). 'The trajectory prediction and analysis of spinning ball for a table tennis robot application'. In: *Proceedings of the IEEE International Conference on Cyber Technology in Automation, Control and Intelligent Systems (CYBER)* (cited on page 81).

Wang, T. and J. Ba (2020). 'Exploring Model-based Planning with Policy Networks'. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cited on page 28).

Warde-Farley, D., T. V. de Wiele, T. D. Kulkarni, C. Ionescu, S. Hansen, and V. Mnih (2019). 'Unsupervised Control Through Non-Parametric Discriminative Rewards'. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cited on page 99).

Watkins, C. J. C. H. and P. Dayan (1992). 'Q-learning'. In: *Machine Learning* 8.3, pp. 279–292 (cited on page 21).

Watkins, C. J. C. H. (1989). 'Learning from Delayed Rewards'. PhD thesis. King's College, Cambridge (cited on pages 6, 11, 21).

Watter, M., J. T. Springenberg, J. Boedecker, and M. A. Riedmiller (2015). 'Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images'. In: *Advances in Neural Information Processing Systems (NeurIPS)* (cited on pages 7, 31, 45).

Williams, R. J. (1992). 'Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning'. In: *Machine Learning* 8, pp. 229–256 (cited on page 23).

Xiao, X., J. Biswas, and P. Stone (2021). 'Learning Inverse Kinodynamics for Accurate High-Speed Off-Road Navigation on Unstructured Terrain'. In: *IEEE Robotics and Automation Letters* (cited on page 67).

Yang, K., L. M. Bergasa, E. Romera, R. Cheng, T. Chen, and K. Wang (2018). 'Unifying terrain awareness through real-time semantic segmentation'. In: *IEEE Intelligent Vehicles Symposium* (cited on page 67).

Yarats, D., I. Kostrikov, and R. Fergus (2021). 'Image Augmentation Is All You Need: Regularizing Deep Reinforcement Learning from Pixels'. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cited on page 123).

Yoon, S. W., A. Fern, and R. Givan (2007). 'FF-Replan: A Baseline for Probabilistic Planning'. In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)* (cited on page 108).

Yu, C., J. Liu, S. Nemati, and G. Yin (2021). 'Reinforcement Learning in Healthcare: A Survey'. In: *ACM Computing Surveys* 55.1 (cited on page 2).

Yu, T., D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine (2019). 'Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning'. In: *Proceedings of the Conference on Robot Learning (CoRL)* (cited on pages 4, 8).

Zaheer, M., S. Kottur, S. Ravanbakhsh, B. Póczos, R. Salakhutdinov, and A. J. Smola (2017). 'Deep Sets'. In: *Advances in Neural Information Processing Systems (NeurIPS)* (cited on pages 50, 52).

Zhang, J., H. Yu, and W. Xu (2021). 'Hierarchical Reinforcement Learning By Discovering Intrinsic Options'. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cited on page 99).

Zhang, M., S. Vikram, L. M. Smith, P. Abbeel, M. J. Johnson, and S. Levine (2019). 'SOLAR: Deep Structured Representations for Model-Based Reinforcement Learning'. In: *Proceedings of the International Conference on Machine Learning (ICML)* (cited on page 31).

Zhang, Y., R. Xiong, Y. Zhao, and J. J. Wang (2015). 'Real-Time Spin Estimation of Ping-Pong Ball Using Its Natural Brand'. In: *IEEE Transactions on Instrumentation and Measurement (TIM)* 64.8, pp. 2280–2290 (cited on pages 81, 95, 126).

Zhang, Z., D. Xu, and M. Tan (2010). 'Visual Measurement and Prediction of Ball Trajectory for Table Tennis Robot'. In: *IEEE Transactions on Instrumentation and Measurement (TIM)* 59.12, pp. 3195–3205 (cited on page 81).

Zhao, Q., Y. Lu, K. J. Jaquess, and C. Zhou (2018). 'Utilization of cues in action anticipation in table tennis players'. In: *Journal of Sports Sciences* 36.23, pp. 2699–2705 (cited on pages 95, 125).

Zhu, B., S. Wang, and J. Zhang (2020). 'Neural Physicist: Learning Physical Dynamics from Image Sequences'. In: *CoRR* abs/2006.05044 (cited on pages 44, 45).

Zhu, Z., N. Li, R. Sun, H. Zhao, and D. Xu (2019). 'Off-road Autonomous Vehicles Traversability Analysis and Trajectory Planning Based on Deep Inverse Reinforcement Learning'. In: *CoRR* abs/1909.06953 (cited on page 67).