Learning Hierarchical Policies from Human Ratings

Zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.)
genehmigte Dissertation von M.Sc. Christian Daniel aus Frankfurt am Main
Tag der Einreichung: 30. November 2015, Tag der Prüfung: 22. April 2016
Darmstadt (2016) — D 17

1. Gutachten: Prof. Jan Peters
2. Gutachten: Prof. Chris Watkins

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Informatik
Intelligente Autonome Systeme

Learning Hierarchical Policies from Human Ratings

Genehmigte Dissertation von M.Sc. Christian Daniel aus Frankfurt am Main

1. Gutachten: Prof. Jan Peters
2. Gutachten: Prof. Chris Watkins

Tag der Einreichung: 30. November 2015
Tag der Prüfung: 22. April 2016

Darmstadt (2016) — D 17

Erklärung zur Dissertation

Hiermit versichere ich, die vorliegende Dissertation ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den May 2, 2016

(C. Daniel)

Abstract

Robots are on the verge of becoming ubiquitous. In the form of affordable humanoid toy robots, autonomous cars, vacuum robots or quadrocopters, robots are becoming part of our everyday life. As of today, most of these robots still follow largely hard coded behavior routines. Constraining a robot's behavior to pre-programmed routines, however, limits its potential in several important ways. For example, programming even simple behavior patterns is a challenging task and programming behavior with human like performance by hand seems impossible. The goal of this thesis, thus, is to develop methods which allow robots to learn solutions to tasks through trial and error instead of relying on manual programming. These learned solutions should fulfill a range of desired properties. Foremost, the solutions should be learned on real world robots and not be constrained to simplified simulation environments. Furthermore, we would like the robot to learn versatile solutions which are able to cope with different variations of a task. Finally, we would like to be able to also solve more complicated tasks which require sequencing of multiple skills.

In the first part of this thesis, we propose an algorithm to learn such versatile solutions. The proposed algorithm aims to find a hierarchical policy, consisting of a gating policy and a set of sub-policies. The gating policy selects a sub-policy and the sub-policy decides which action to take. Each of the learned sub-policies may be able to solve the task or a part of the overall task. To learn multiple sub-policies from one sample set, we employ an expectation-maximization based learning algorithm, where the sub-policies are updated according to their responsibilities for individual samples. These responsibilities indicate how likely it is that a certain sub-policy generated a given state-action pair. By constraining our learning algorithm to solutions which increase the entropy of the responsibilities, the robot will learn a set of sub-policies that encode different solutions for the same task. This kind of concurrency is highly desirable as it allows the robot to learn back-up solutions which may still be valid even if the original solution fails due to changes in the robot or environment.

In the second part of this thesis, we tackle the challenge of learning skills directly from state-action trajectories. A common approach in robot learning is to assume access to some sort of parametrized skill to represent the robot's behavior over multiple time steps. These skills are usually either movement primitives (MPs) or parametrized feedback controllers. While especially MPs have been a cornerstone in advancing the state of the art in robot learning, it is not clear how to learn from the actions taken throughout the execution of a skill and MPs usually do not encode feedback. In the discrete state-action reinforcement learning (RL) setting, macro-actions, or options, have been introduced to address the problem of learning temporally correlated actions, which can be viewed as a form of skill. To connect robot learning with the advances in the field of discrete state-action RL, we propose a probabilistic framework to infer options from state-action trajectory observations. The inference is based on a hidden Markov model (HMM), where the options indices are modeled as latent variables and where inference can be performed by adapting well known expectation-maximization algorithms. Because this framework allows for the inference of parametric policies, it is also compatible with policy search (PS) methods, a class of RL algorithms which is at the core of many recent successes in robot learning.

Learning methods, such as the one we propose in the first chapter of this thesis, enable robots to learn from trial and error instead of having to program specific solutions. Unfortunately, the

quality of the learned solutions still depends heavily on a significant amount of expert knowledge and programming which has to be directed at designing reward functions. In RL, reward functions effectively serve as task description, guiding the robot to a good solution by encoding the desirability of states and actions. Designing these reward functions, however, is a difficult task even for experts and it is unlikely that non-experts will be able to train robots on new tasks by means of programming reward functions. Thus, in the last part of this thesis, we propose a method which allows the robot to model a teacher's implicit reward function during the RL process. This reward model is represented by a Gaussian process (GP), a class of probabilistic function approximators which allow us to take both the teacher's and the robot's uncertainty into account. Using the proposed method, the robot can ask the teacher to evaluate the quality of just a few select robot actions, such that the robot can improve its reward model. To minimize the amount of human-robot interactions, the robot uses its uncertainty estimate to request evaluations only for actions which seem to be promising but have a high uncertainty. In this setting, the teacher will not need to be a roboticist and no manual coding is required.

Altogether, the individual contributions of this thesis allow robots to learn versatile solutions to complicated tasks from trial and error. These solutions generalize over similar settings and are robust by representing multiple solutions to one task. The robot is able to learn these solutions by interacting with a human teacher, eliminating the need for task specific expert programming. The contributions of this thesis are evaluated on a wide array of both simulated and real world tasks and the results show the effectiveness of the proposed methods.

Zusammenfassung

Roboter sind kurz davor allgegenwärtig zu werden. Ob als preiswerte humanoide Spielzeugroboter, autonome Autos, Staubsaugroboter oder Quadrocopter werden Roboter immer mehr Teil unseres Alltags. Gegenwärtig folgen diese Roboter allerdings hauptsächlich händisch programmierten Verhaltensroutinen. Wenn man aber das Verhalten eines Roboters auf fest programmierte Routinen begrenzt, kann man das volle Potenzial dieses Roboters nicht nutzen. Selbst bei einfachen Aufgaben ist es oft schwierig, gute Lösungen von Hand zu programmieren. Menschenähnliche Leistungsstufen von Hand für komplizierte Aufgaben zu programmieren scheint fast unmöglich. Das Ziel dieser Arbeit ist deshalb Methoden zu entwickeln, die es Robotern ermöglichen Lösungen zu Aufgaben durch "trial and error" zu finden, anstatt auf händisch programmierte Lösungen für spezifische Aufgaben angewiesen zu sein. Diese gelernten Lösungen sollten verschiedene wünschenswerte Eigenschaften aufweisen. So sollten diese Lösungen auf realen Robotern gelernt werden können und nicht auf stark vereinfachte simulierte Umgebungen beschränkt sein. Zusätzlich sollte der Roboter vielfach anwendbare Lösungen lernen, die in vielen verschiedenen Variationen einer Aufgabe anwendbar sind. Schlussendlich sollte der Roboter auch komplizierte Aufgaben lösen können, die eine Aneinanderkettung mehrerer "Skills" benötigen, wobei der Ausdruck "Skill" ein gelerntes Verhalten oder eine Fertigkeit des Roboters beschreibt.

In dem ersten Teil dieser Arbeit stellen wir einen Algorithmus vor der solche vielseitigen Lösungen lernen kann. Dieser Algorithmus findet eine hierarchische Policy, die aus einer Gating-Policy und mehren Unter-Policies besteht. Dabei wählt die Gating-Policy zwischen den verschiedenen Unter-Policies und die Unter-Policies codieren die Aktionen, die der Roboter ausführt. Jede der gelernten Unter-Policies kann eine Aufgabe oder einen Teil einer komplizierteren Aufgabe lösen. Um mehrere Unter-Policies gleichzeitig von einem Datensatz zu trainieren, nutzen wir einen expectation-maximization Algorithmus, bei dem jeder Datenpunkt (state-action Paar) für das Update aller Unter-Policies genutzt wird. Der Einfluss jedes state-action Paares auf das Update einzelner Unter-Policies ist dabei davon abhängig, wie sehr der Datenpunkt von der Unter-Policy erklärt wird, also wie wahrscheinlich es ist, das die Unter-Policy diese state-action Paar selbst erzeugt hat. Dies wird auch als Verantwortung der Unter-Policy für einen Datenpunkt bezeichnet. Wenn wir unseren Algorithmus auf Lösungen begrenzen, die eine hohe Entropie dieser Verantwortungen haben, wird der Roboter Unter-Policies lernen die verschiedene Lösungen für die gleiche Aufgabe codieren. Diese Mehrfachlösungen sind sehr wünschenswert, da sie dem Roboter erlauben Ersatzlösungen zu lernen die eingesetzt werden können falls die bevorzugte Lösung nicht mehr valide ist. Dies kann zum Beispiel durch unvorhergesehene Änderungen in der Umgebung des Roboters erforderlich werden.

Im zweiten Teil dieser Arbeit untersuchen wir Lösungen um Skills direkt von beobachteten state-action Trajektorien zu lernen. Ein beliebter Ansatz im Bereich des Roboterlernens ist es parametrisierte Skills zu verwenden, um das Roboterverhalten über mehrere Zeitschritte hinweg zu beschreiben. Diese Skills sind häufig entweder sogenannte Movement Primitives (MPs) oder parametrisierte Regler. Während besonders MPs wesentlich zu Erfolgen im Feld des Roboterlernens beigetragen haben, lösen diese Ansätze nicht die Frage, wie alle state-action Paare einer Trajektorie genutzt werden können, um die Effektivität der Lernalgorithmen weiter zu steigern. Im Feld des diskreten state-action Reinforcement Learning (RL) wurden Makroaktionen, auch

Options genannt, eingeführt um von solchen zeitlich zusammenhängenden Aktionen Skills zu lernen. Um das Feld des Roboterlernens mit den Fortschritten des diskreten RLs zu verbinden, schlagen wir ein probabilistisches Modell vor womit Options durch Inferenz von kontinuierlichen state-action Paaren gelernt werden können. Diese Inferenz basiert auf einem Hidden Markov Model (HMM), in dem die Option-Indizes als latente Variablen modelliert werden und für welches bekannte expectation-maximization Algorithmen angewandt werden können. Da dieses Modell die Inferenz von parametrisierten Policies erlaubt, ist es kompatibel mit sogenannten Policy Search (PS) Methoden, einer Klasse von RL Algorithmen, die das Herzstück vieler derzeitiger Fortschritte im Roboterlernens ist.

Lernmethoden, wie diese die wir im ersten Kapitel dieser Arbeit beschreiben, befähigen Roboter von selbstständigen Versuchen und Fehlern "trial and error" zu lernen, anstatt spezifisch programmierte Lösungen zu erforden. Allerdings hängt die Qualität dieser gelernten Lösungen immer noch stark von einer beachtlichen Menge an Expertenwissen ab, das in die Programmierung sogenannter Rewardfunktionen fließt. In RL dienen diese Rewardfunktionen als Beschreibung einer Aufgabe, die den Roboter zu einer guten Lösung führt, indem die Qualität von verschiedenen states und actions codiert wird. Die Erstellung solcher Rewardfunktionen ist aber eine schwierige Aufgabe in sich selbst, an der selbst Experten oft scheitern und es ist daher unwahrscheinlich, dass Laien Robotern neue Aufgaben auf diesem Wege beibringen können. Daher beschreiben wir in dem letzten Teil dieser Arbeit eine neue Methode, die es dem Roboter erlaubt die implizite Rewardfunktion eines menschlichen Lehrers während des RL Prozesses selbst zu erlernen. Der Roboter lernt dazu ein Modell der Rewardfunktion, welches als Gauß- Prozess (GP) , eine Klasse probabilistischer Funktionsapproximatoren, repräsentiert wird. Dieses Modell erlaubt es dem Roboter sowohl seine eigene Unsicherheit bezüglich der Rewardfunktion als auch die Unsicherheit des Lehrers zu berechnen. Mit der vorgestellten Methode kann der Roboter den Lehrer um Evaluationen selektiver Aktionen bitten, damit der Roboter sein Modell der Rewardfunktion verbessern kann. Um die Anzahl der Mensch-Roboter Interaktionen zu minimieren, nutzt der Roboter seine Einschätzung der Unsicherheit um Evaluationen nur für vielversprechende Aktionen anzufordern, die aber mit einer hohen Unsicherheit belegt sind. In diesem Szenario muss der Lehrer kein Experte sein und ein manuelles Programmieren des Roboters ist nicht erforderlich.

Gemeinsam erlauben es die individuellen Beiträge dieser Arbeit, dem Roboter vielfältige Lösungen für komplizierte Aufgaben durch trial and error zu lernen. Diese Lösungen generalisieren über ähnliche Aufgaben hinweg und sind robust, da mehrere Lösungen für eine Aufgabe gefunden werden. Der Roboter kann diese Lösungen durch die Interaktion mit einem menschlichen Lehrer lernen, was die aufgabenspezifische Programmierung des Roboters hinfällig macht. Die Beiträge dieser Arbeit werden im Folgenden im Detail dargestellt und auf verschiedenen simulierten und realen Aufgaben evaluiert.

Acknowledgments

Throughout my Ph.D. I have received support from many different sources without which this thesis would not have been possible. I would like to thank, therefore,

- my thesis advisers Jan Peters and Gerhard Neumann. Jan was an inspiring adviser who introduced me into the world of research, convinced me to pursue a Ph.D. and always pushed me to be my best. Gerhard Neumann was instrumental in the success of my Ph.D. Gerhard was always eager to help with every problem and usually already thinking three steps ahead.
- my thesis referees Prof. Jan Peters and Prof. Chris Watkins for evaluating this thesis.
- Prof. Gerhard Neumann for heading the thesis committee and Prof. Stefan Roth and Prof. Constantin Rothkopf for participating in the defence.
- the colleagues that shared an office and a large part of my life with me over the last four years. Especially I would like to thank Herke van Hoof, who always took the time to understand every problem I threw at him and almost always was able to come up with an ingenious solution. I would like to thank Oliver Kroemer, who showed me the ropes in what it means to be a graduate student and who improved all of my publications by proof reading them in unprecedented detail. Gregor Gebhard and Daniel Tanneberg who always had an open ear for all of my large and little troubles.
- all other colleagues and collaborators whom I shared many treasured experiences with over the last four years.
- my parents and sister for steering me through life and always being there for me when I need help.
- Ilka for making life as exciting as possible.
- Deniz, Helge and Florian for giving me support whenever I needed it.

Contents

Notation

The following notation and symbols are used throughout this thesis:

Notation	Description
$\{x_1, x_2, \ldots, x_N\}$	set of N elements x_1 through x_N.
$\boldsymbol{x} = [x_1, x_2, \ldots, x_N]$	vector
x_i	i$^{\text{th}}$ component of a vector
$\boldsymbol{x}^T$	transpose of a vector
$\boldsymbol{x}_{1:N}$	series of N vectors $\boldsymbol{x}_1$ through $\boldsymbol{x}_N$
$\boldsymbol{A}$	matrix
$\boldsymbol{A}^T$	transpose of a matrix
$p(\boldsymbol{x})$	probability density
$\frac{\mathrm{d}f}{\mathrm{d}q}$	partial derivative
$\mathbb{E}[\boldsymbol{x}]$	expectation of $\boldsymbol{x}$

Symbol	Description
$\boldsymbol{s}$	state
$\boldsymbol{a}$	action
r	reward
t	time step
o	option index
$\boldsymbol{\phi}$	features
$\pi(\boldsymbol{a}\|\boldsymbol{s})$	policy
$J(\pi)$	expected reward of policy π
D_{KL}	Kullback-Leibler divergence
$H(p(\boldsymbol{x}))$	entropy of distribution $p(\boldsymbol{x})$
$\mathscr{R}_{\boldsymbol{sa}}$	reward $r(\boldsymbol{s}, \boldsymbol{a})$
$\mathscr{P}_{\boldsymbol{sa}}^{\boldsymbol{s}'}$	transition probability $p(\boldsymbol{s}'\|\boldsymbol{s}, \boldsymbol{a})$
$\eta, \boldsymbol{\theta}, \xi$	Lagrangian parameters
$\mathcal{N}(\boldsymbol{x}\|\boldsymbol{\mu}, \boldsymbol{\Sigma})$	normal distribution over $\boldsymbol{x}$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$
$\mathscr{GP}$	Gaussian process

1 Introduction

Enabling robots to go beyond their current capabilities will require new algorithms, allowing robots to learn from their experience and from human guidance. In this thesis, we tackle four major challenges of real robot reinforcement learning: learning hierarchical policies, sequencing robotic skills, exploiting temporal correlation and learning from human ratings. In the following, we give a high level overview of the current state of the art of artificially intelligent robots and motivate the need for the methods proposed in this thesis. To present a more technical and in-depth overview of the state of the art, each individual chapter discusses the respective related work in a dedicated section.

1.1 State of the Art & Open Challenges

To illustrate some of the challenges we address in this thesis, we can consider the example application of a robot nurse. In our example, we can assume that the robot leaves the factory not as a nurse, but as a blank sheet, multi-purpose robot. Becoming a nurse requires the robot to train and learn the skills expected of a (robot) nurse. To do so, the robot may try different actions and learn which actions lead to desired outcomes.

1.1.1 Reward Learning Challenge

To assess the quality of these outcomes, the robot needs access to a reward model. Unfortunately, designing these reward models is hard even for robotic experts and, thus, beyond what can be expected from the hospital staff. However, the staff and even the hospital's patients can be relied on to indicate the quality of specific outcomes without having to design a full reward model. Thus, the robot can learn directly from these human ratings.

While this approach is technically feasible, it ignores two problems. First, humans will always give noisy ratings that are hard to learn from directly. Second, while humans may be willing to rate a few outcomes, the robot will typically require much more learning episodes than the human is willing to rate. To overcome these challenges, we propose a novel approach to learn a reward model from only a few human ratings. We model the reward function using a Gaussian process and present a novel acquisition function that decides when the robot needs to ask for additional ratings. This novel acquisition function considers the change in the resulting policy instead of the change in the reward model to be highly sample efficient. Thus, the robot can employ its learned reward model to rate the majority of outcomes and only needs to ask for human ratings occasionally.

1.1.2 Robustness Challenge

Apart from the possibility of learning from human teachers, a further challenge is to learn robust skills that offer fall-back solutions. Typically learning algorithms only represent one solution and are rendered useless when this solution becomes unavailable. For example, we may consider

the simple task of handing a patient his medicine using either the right or the left hand. During learning, both options may be equally feasible, but using traditional learning methods the robot would have to learn one of the solutions and discard all data referring to the alternative. If the robot now encounters a situation where it cannot use its preferred hand, either because of a malfunction or possibly because it is carrying something else already, it could no longer assist the patient, even though a simple solution would exist. Instead, we propose a learning method which represents and learns multiple solutions at the same time. Thus, even if one solution becomes unavailable, the robot can choose from a repertoire of learned alternatives. To represent multiple solutions, we develop a hierarchical policy by splitting the policy in a gating network and sub-policies. The gating network chooses which sub-policy to activate and the sub-policy encodes a skill. To learn multiple sub-policies from the same experience, we frame the learning problem as a latent variable estimation problem, where the index of the generating sub-policy is unknown.

1.1.3 Skill Chaining Challenge

The ability to represent a complex policy made up of simpler sub-policies will help our robot to deal with complicated tasks. For example, many tasks in the real world are strongly structured and require a combination of different skills. Getting a cup requires going to the cupboard, opening it, grabbing the cup and bringing it to the patient. Adapting and sequencing these skills such that they interleave and successfully solve the task is an important step towards a robot which works in human environments. To learn how to sequence skills, we phrase the learning problem as a a learning problem with a fixed number of decision steps, i.e., a finite horizon RL problem. In each decision step, the robot can choose from a repertoire of skills and it learns how to select between these skills and how to adapt these skills to maximize the overall reward.

1.1.4 Temporally Correlated Actions Challenge

Finally, the last challenge we tackle in this thesis is the exploitation of temporal correlation. Since the search space of robotic tasks is usually very large, exploring this space by taking random actions can lead to poor performance. For example, we can imagine the robot nurse trying to locate a missing patient. By performing random actions, the search will concentrate on an area very close to the starting point of the search and reaching areas far away from the starting point will take a long time. Instead, it would be beneficial to take actions which are correlated and lead, for example, to a different wing of the hospital before exploring locally. Temporally correlated sub-policies are a form of skills or motor primitives. However, these correlated sub-policies are represented by stochastic distributions which can encode feedback controllers and that do not depend on a manual specification of the execution duration. These correlated sub-policies will enable our robot to learn skills that are much more robust to small deviations, which may be induced due to factors such as noise in the system or the environment.

To encode temporal correlation into the policy, we further extend the proposed hierarchical policy representation by a termination policy, which can decide to keep executing an active skill, or else choose to start a new skill. As with the gating network, we phrase the problem as a latent variable estimation problem and show how we can use the expectation-maximization algorithm to find termination policies that accelerate initial exploration but still find optimal solutions upon convergence.

1.2 Contributions

This thesis proposes several methods to address key challenges in robot learning. These contributions represent important steps towards the goal of autonomous robots and while some of these contributions can be applied on their own, they interact with and benefit from each other.

Learning Hierarchical Policies

Real world tasks are often strongly structured. To take advantage of these structures we propose a hierarchical policy representation, which is able to mirror these structures and which allows learning solutions to complex tasks through a combination of simpler sub-policies. To improve multiple sub-policies from a single data set, we treat the index of the generating option of a state-action sample as a latent variable and employ an expectation-maximization based approach to assign responsibilities for each data point to each option.

1.2.1 Concurrent Learning of Multiple Solutions

Using the hierarchical policy representation, we are also able to learn concurrent solutions to a given task, where individual sub-policies encode alternative approaches to solving a task. To ensure that different sub-policies do not concentrate on the same solution during learning, we propose an additional constraint on the learning process, which guarantees sufficient entropy of the responsibilities. Where a low entropy of the responsibilities is equivalent to all options being responsible for the whole state-action space, a higher entropy is achieved when the different sub-policies are active for different regions of the state-action space.

1.2.2 Sequencing of Skills

Many real world tasks require a sequence of possibly quite different skills. While each skill execution might be rewarded individually, the overall outcome depends on the correct selection and adaptation of skills for each step of this sequence. We show how our hierarchical policy search algorithm can be used in a finite horizon setting, where in each state a different hierarchical policy can be learned. This algorithm will adapt the individual policies in such a way that they will generate compatible behavior which solves multi-step problems.

1.2.3 Learning with Temporal Correlation

Given a hierarchical policy representation, one important problem is how to learn individual skills directly from state-action trajectories. We propose a Hidden Markov Model, where both the option indices as well as the termination events are treated as latent variables. Based on the resulting graphical model, we can perform inference on the state-action trajectories observed during learning in order to model the distributions over the latent variables. This result allows us to represent a stochastic hierarchical policy in the Semi Markov Decision Process (SMDP) setting as a combination of parametrized distributions. The stochastic policy encodes multiple temporally

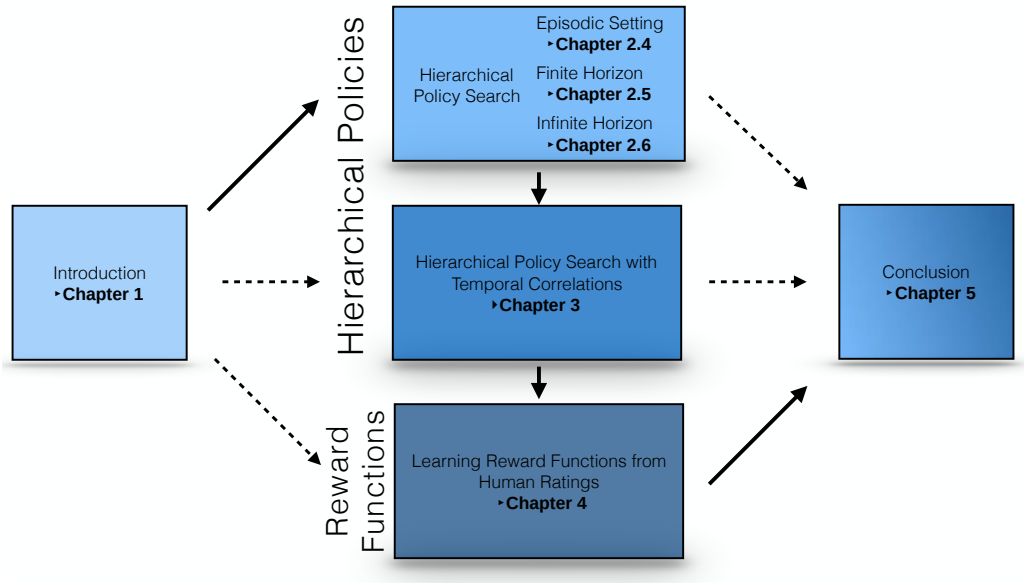


Figure 1.1.: Outline of the presented thesis. While the chapters of this thesis can be read independently, we recommend reading them in order. Chapter 2 details the HiREPS algorithm for the three RL settings, i.e., the episodic setting, the finite horizon setting and the infinite horizon setting. The algorithms of Chapter 2 depend on good parametrizations of temporally extended actions. Chapter 3 proposes a framework to infer such options from data in the continuous state-action case. Finally, both Chapter two and three assume access to a reward function, as is standard in RL. However, designing these reward functions is non-trivial and Chapter 4 discusses how such reward functions can be learned from a human teacher.

correlated skills and is applicable both in the discrete as well as in the continuous state-action setting. Because the option components are represented as parametrized distributions, we can employ state-of-the-art policy search methods to train the policy.

1.2.4 Learning from Human Ratings

Training policies using RL methods generally requires good reward functions, which are often hard to design by hand. Instead of requiring an expert to manually design a reward function, we propose a framework wherein the robot models the implicit reward function of a teacher. In this setting, the robot aims to solve an active learning problem in between iterations of the RL process. After each RL iteration, the robot can select actions to be rated by the teacher in order to refine the reward model. This setting potentially allows non-experts to train robots on new tasks.

1.2.5 A Novel Acquisition Function for Bayesian Optimization

Because we use a probabilistic function approximator to model the reward function, the robot can estimate both its own as well as the teachers uncertainty to select which actions should be rated by the teacher. Traditional Bayesian Optimization (BO) methods allow the robot to select samples which minimize the overall uncertainty of the reward model. However, these traditional

acquisition functions do not take the robot's policy into account which can make them sample inefficient. We propose a novel acquisition which does not only aim to reduce uncertainty, but instead aims to maximize the expected impact of additional ratings on the policy.

1.3 Outline

While we recommend reading the chapters of this thesis in order, they are written such that they can also be read independently. Fig. 1.1 shows a diagramm of the presented thesis. In Chapter 2, we discuss the HiREPS algorithm, which enables us to efficiently learn multiple sub-policies simultaneously. The Sections 2.5 to 2.7 discuss implementations for the three RL scenarios, i.e., the episodic, finite horizon and infinite horizon setting. Chapter 3 discusses how to learn all elements of an option, i.e., the global activation policy and the option-specific termination and sub-policies from data. Chapter 4 discusses the difficulty of designing reward functions. These reward functions serve as task-descriptors in RL and are usually assumed to be given. However, designing these reward functions is often hard even for experts. Thus, we propose a method to learn reward functions from a human non-expert in RL. In the final chapter of this thesis, we summarize our findings and discuss the results of the experimental evaluations presented in this thesis. In Chapter 5.2, we propose several topics for future research. Some of these research ideas are very directly linked to extending the methods presented in this thesis, while others have a more general scope.

2 Hierachical Reinforcement Learning

Many reinforcement learning (RL) tasks, especially in robotics, consist of multiple sub-tasks that are strongly structured. Such task structures can be exploited by incorporating hierarchical policies that consist of gating networks and sub-policies. However, this concept has only been partially explored for real world settings and complete methods, derived from first principles, are needed. Real world settings are challenging due to large and continuous state-action spaces that are prohibitive for exhaustive sampling methods. We define the problem of learning sub-policies in continuous state action spaces as finding a hierarchical policy that is composed of a high-level gating policy to select the low-level sub-policies for execution by the agent. In order to efficiently share experience with all sub-policies, also called inter-policy learning, we treat these sub-policies as latent variables which allows for distribution of the update information between the sub-policies. We present three different variants of our algorithm, designed to be suitable for a wide variety of real world robot learning tasks and evaluate our algorithms in two real robot learning scenarios as well as several simulations and comparisons.

2.1 Introduction

Employing robots in unpredictable environments, such as in hospitals, disaster sites or households, requires robotic agents to autonomously learn new tasks and adapt to new environments. Such robots will need to acquire new skills through trial and error, also known as reinforcement learning [Sutton and Barto, 1998], as well as being able to generalize their skills to solve a large variety of tasks. However, successful implementation of reinforcement learning (RL) methods on real robot tasks is challenging for multiple reasons. Most importantly, real robots have high dimensional and continuous state-action spaces which is difficult to deal with for many RL methods. Furthermore, evaluations on real robots are resource intensive and, hence, the methods need to be sample efficient. Methods should also not fully explore the state-action space due to the risk of damaging the robot or its environment. Finally, real robot RL does not allow resetting of the state to arbitrary initial conditions as is required for many RL methods.

While presenting additional challenges, robot tasks also offer some advantages over traditional RL settings as many real world motor tasks are heavily structured. Exploiting the environment's structure can drastically simplify the learning problem. First, the solutions of many tasks lie within a tube of the solution space [Peters and Schaal, 2006], such that learning can be given a head start by demonstrating a sub-optimal solution. Local RL methods can, subsequently, be used to improve upon this demonstration. Second, the motor commands for many motor tasks exhibit strong temporal correlations. Such correlations allow for a hierarchical decomposition of the task into a sequence of elemental movements, often also referred to movement primitives [Schaal et al., 2003, Ijspeert et al., 2003], options [Sutton et al., 1999b] or motor templates [Neumann and Peters, 2009]. For example, a tennis game can be decomposed into a sequence of single strokes, e.g., a backhand strokes, forehand strokes, lobs, volleys and a serve. Finally, many motor tasks can be solved in multiple, often incompatible, ways. Identifying and representing such solutions as separate sub-policies to be used by the agent increases the versatility as well as the robustness of the learned policy. Additionally, it simplifies the use of local learning methods.

We base our learning algorithm on the Relative Entropy Policy Search (REPS) algorithm [Peters et al., 2010]. In REPS, the exploitation-exploration trade-off is balanced by bounding the loss of information between policy updates. Such a bound results in a smoother and more stable learning process that avoids wild exploration of the state action space, as required by the robotics domain. We extend the REPS algorithm such that we can use sub-policies as building blocks of a hierarchical policy to exploit the temporal correlations inherent to many tasks. We formulate the problem of inferring such a hierarchical policy as a latent variable estimation problem, where the index of the sub-policy that has generated a given action is modeled as a latent variable. The policy update is now performed in two steps. In the Expectation step, we compute the responsibilities of the latent variables, i.e., the probabilities that the individual sub-policies have generated the observed state action pairs. The sub-policies are kept fixed in the E-step. Subsequently, the responsibilities are used to update the hierarchical policy containing the sub-policies with the REPS algorithm in the Maximization step. We show that such an Expectation-Maximization (EM) algorithm [Dempster et al., 1977] improves a lower bound of the original REPS optimization problem and that the lower bound is tight after each expectation step.

Our algorithm also allows for using prior knowledge to constrain the structure of the estimated hierarchical policy. For example, we often want the sub-policies to represent individual, distinct solutions. In many cases, these solutions may be incompatible, i.e., the solution space may not be convex. Therefore, the set of sub-policies should be separable in the solution space. Adding a constraint that avoids such an overlap ensures that the policy search algorithm does not average over multiple modes of the solution space, which may result in a poor performance of the resulting policy [Neumann, 2011].

We will discuss different variants of the Hierarchical REPS (HiREPS) approach with increasing complexity. We start our discussion with the contextual, continuous-armed bandit case where the episode ends after executing all actions belonging to one sub-policy. sub-policies are selected according to a gating policy $\pi(o|\boldsymbol{s})$ and define a distribution over the action given the state for a predetermined time horizon. Subsequently, we show how to extend this framework to the finite horizon setting, where the agent sequences multiple sub-policies to achieve his goal. Finally, we present the infinite horizon formulation of HiREPS, where the agent keeps executing sub-policies until either the task is fulfilled or a random reset occurs.

2.2 Related Work

Policy search (PS) is a class of RL methods which have been shown to fulfill the requirements of robot RL and is widely used in real robot learning. In applications, PS methods are often preferred over other traditional RL methods such as value-based methods, since they do not require the explicit estimation of a value function. Estimating a value function often requires the agent to fill the state and action space with samples, which is infeasible in robot RL. Therefore, most PS methods are commonly local methods and improve only locally on an initial, sub-optimal solution.

Particular properties of PS methods have been highlighted by several papers. Bagnell and Schneider [2003] show their strong convergence guarantees, Sutton et al. [1999a] discuss their application in combination with function approximators and Stone [2001] improved the possibilities of incorporating domain knowledge. Backing up the theoretical strength of policy search methods, several authors have demonstrated impressive application results. Bagnell and Schneider [2001] use policy search methods to autonomously control helicopters, Rosenstein [2001] learns

robot weight-lifting, Kohl and Stone [2003] demonstrates learning of a quadrupedal walking behavior and Kober et al. [2008] successfully learn the ball in a cup (also known as Kendama) game. Endo et al. [2008] show the application of PS methods on a biped locomotion and Kormushev et al. [2010] PS to learn how to flip a pan-cake with a robot arm.

Important advances in the area of policy search methods have included pair-wise policy comparisons [Strens, 2003], policy gradient methods [Baxter and Bartlett, 2001, Sutton et al., 1999a] and natural policy gradient methods [Bagnell and Schneider, 2003, Peters and Schaal, 2006]. More recently, Kober et al. [2008] presented probabilistic policy search approaches based on expectation maximization. Using EM like methods for reinforcement learning was initially pioneered by Dayan and Hinton [1997]. Toussaint et al. [2006] showed how an EM like method can be used not only for MDPs but also partially observed MDPs. Deisenroth [2010] developed a model-based policy search method based on probabilistic modeling. Theodorou et al. [2010] showed a policy improvement algorithm inspired by path integrals and Peters et al. [2010] proposed a PS method that limits the loss of information between policy updates. Recently, Levine and Abbeel [2014] and Schulman et al. [2015] have shown how policy search methods can be combined with neural networks to learn complicated tasks in high dimensional domains. As these results show, current methods are well suited to learn single tasks in isolation. Nevertheless, they frequently fail to scale up to more complex motor tasks as they cannot exploit the hierarchical structure inherent to many tasks.

A common way of representing a hierarchical task structure is to use the concept of options. Options are temporally extended actions and were first introduced by Sutton et al. [1999a] as a way of reducing task complexity. They consist of three components, an action-selection policy, an initiation set that defines in which states an option can be initiated and a termination condition which defines the probability of terminating an option in a given state. Typically, options extend the action space of the agent, i.e., the agent can still take the primitive actions which are active for one time step, and, in addition, the agent can choose the options as temporally extended actions. The use of options can significantly improve the learning speed as the options can be used to connect parts of the state space that are otherwise only reachable with a long action sequence [Sutton et al., 1999a]. Consequently, the state space becomes easier to explore and, hence, the learning problem gets simplified. Formally, learning with options can be formulated as learning a Semi-Markov Decision Process (SMDP). SMDPs are not fully Markovian, as the selected option does not only depend on the current state, but also on the active option in the previous time step.

Unlike in the simplest setup where the set of options is given and fixed during learning, it is often also desirable to learn useful options for a given task. In this case, the data efficiency for learning the options can be significantly improved by leveraging all information collected during the execution of one option, also denoted as intra-option learning [Sutton et al., 1998]. The state transitions generated by a certain option can also be used to update other options in an off-policy learning setup. Levy and Shimkin [2012] proposed a different view of intra-option learning by augmenting the state space with the option-termination probability. In that augmented state space, the option selection policy and termination condition may be represented by orthogonal basis functions and is optimized individually by standard policy gradient methods.

Options are also used in many hierarchical RL approaches where they are often extended to sub-tasks. A sub-task also contains an individual reward function such that we can learn the policy of the subtask. Dietterich [2000] proposed the MAXQ framework that uses several layers of subtasks. In contrast to the traditional option framework, the policies of the subtasks can also

choose to execute a new subtask instead of just a primitive action. The subtasks are given by an individual reward function per subtask, a termination condition and a set of actions or other subtasks that can be executed. The policy of the subtasks is learned by the MAXQ-Q learning algorithm that learns the optimal value function for each subtask. However, the structure of the subtasks, i.e., the individual reward functions, must be specified by the user. How to define such subtasks for complex robot motor tasks or even learn the structure from data remains an open question. Additionally, value function methods are less applicable in continuous state action domains and not well suited for robotic tasks. Barto et al. [2004] used an 'intrinsic motivation' mechanism to define the reward signal of each subtask. A naturally curious agent is exploring its environment and whenever it discovers an unexpected change in its environment, it creates a new subtask in its internal representation that tries to reproduce this unexpected event. Hence, the agent incrementally builds up a set of skills that help the agent to explore its environment more efficiently. After this exploration phase, new tasks can be learned more efficiently by using the learned set of skills.

The notion of subtasks has also been used in continuous environments. Morimoto and Doya [2001] proposed a hierarchical RL setup where the subtask is defined by reaching a specific joint configuration of the robot. In this work, the set of subtasks, i.e., desired joint configurations, is given and the robot learns to reach the individual joint configurations as well as to choose a reachable next desired joint configuration. Ghavamzadeh and Mahadevan [2003] used a policy gradient approach to learn the policies for the individual subtasks while a value based method is used to select the next subtask. Policy gradient approaches work well in continuous environments but often converge slowly. In both approaches, the subtasks have been specified by the user and could not be modified by the learning algorithm. Hence, both approaches are limited to simpler robots where appropriate subtasks can be hand-tuned. In Konidaris and Barto [2009], the structure of the subtasks is also learned by discovering the initiation set, the termination set and the option policy for new options. The options are then be chained together to solve the overall task. However, the option discovery algorithm is to date still limited to rather simple agents, such as a ball navigating in a maze.

The approach which is probably most closely related to ours is Bayesian policy search with hierarchical policy priors [Wingate et al., 2011]. In this approach, the probabilistic formulation of policy search [Kober et al., 2008] is used and a hierarchical Bayesian prior is used for the policy parameters. Similarly to our approach, the activations of options are modeled as latent variables in their model that are estimated by MCMC sampling methods. While the use of hierarchical Bayesian priors is a promising idea, the approach is restricted to directly learn on the trajectory data instead of state actions pairs, which might lead to inefficient policy updates. In addition, the approach cannot be extended straightforwardly to complex high dimensional robots as the structure of the options is limited.

Another approach to simplifying the hierarchical RL problem in continuous action spaces is to restrict the space of possible trajectories by using parametrized policies, often also called movement primitives [Schaal et al., 2003], motion templates [Neumann and Peters, 2009] or parametrized skills [Da Silva et al., 2012]. Learning the option policy now reduces to learning an appropriate parameter vector for the option. In Neumann and Peters [2009], the agent learned to sequence such parametrized policies, i.e., it learned the correct order of the parametrized policy as well as the single parameter vectors of the policy. In total, the agent has to learn fewer decisions as opposed to directly learning with primitive actions. However, learning is often also more

difficult as the effect of an inaccurate decision can be much more costly than for primitive actions. While Neumann and Peters [2009] used value function approximation to evaluate the quality of the chosen parameter vectors of the options, Stulp and Schaal [2012] directly used the reward to come as evaluation. This reward to come was subsequently used by the Policy Improvement by Path Integrals (PI^2) algorithm [Theodorou et al., 2010] to learn the option policy.

Parametrized options or skills were also used for skill transfer, i.e., generalizing the parameters of the options to new tasks [Kupcsik et al., 2014]. Thomas and Barto [2012] showed how the structure of motor primitives for continuous state action tasks can be learned and later be used to accelerate learning of a similar task. Da Silva et al. [2012] as well as Kober et al. [2010b] generalized parametrized skills for reasonably similar tasks drawn from a task distribution. They do not only use the concept of an option for a single policy, but instead allow each option to adapt to a subset of the task space by smoothly changing the parameter vector of the option. Hence, individual options are responsible for areas of the task space that are locally smooth. Multiple options together span discontinuous areas of the task space. The proposed framework consists of a pipeline of machine learning tools to identify the individual smooth lower dimensional manifolds as well as to generalize and improve the policy parameters. Alternative approaches for task generalization of parametrized options include learning a mixture of experts [Mülling et al., 2013] and Gaussian processes [Ude et al., 2010]. The method presented in this chapter is applicable to both primitive actions as well as movement primitives.

In the remainder of the chapter we will explain how to obtain a hierarchical formulation of the relative entropy policy search (REPS) method. We treat the problem of learning a hierarchical policy as a latent variable estimation problem. For the policy update, we assume that we can only observe the resulting actions of the old policy. The underlying hierarchy is unknown and, therefore, unobserved. The resulting algorithm is closely related to expectation maximization (EM) for latent variable models. We prove that such EM mechanisms can be incorporated in the information theoretic regularization of the REPS algorithm in order to get a lower bound of the original optimization problem which can, subsequently, be optimized in closed form. Furthermore, we introduce an additional constraint into the optimization problem that bounds the uncertainty of identifying a sub-policy given an action. As a consequence, the sub-policies are separated in the action space, which results in finding more versatile solutions and also alleviates the problem of averaging over several modes in the solution space, which is present in many current policy search algorithms as shown in Neumann [2011]. In the evaluation section, we compare our algorithm to other state-of-the-art methods on benchmark problems and evaluate our algorithm on real world robotic applications.

2.3 Background on Information Theoretic Policy Search

As our algorithm is based on information theoretic policy search [Peters et al., 2010, Daniel et al., 2012a], we will quickly review the most relevant concepts of information theoretic policy search in the non-hierarchical learning setup. In policy search, an agent tries to maximize the expected return by adapting a parametrized policy. Formally, in a Markov decision process setting, the agent is in a state $\boldsymbol{s} \in \mathbb{S}$ and chooses an action $\boldsymbol{a} \in \mathbb{A}$ to execute. Given state $\boldsymbol{s}$ and action $\boldsymbol{a}$, the agent transfers to a next state $\boldsymbol{s}'$ in accordance with a transition probability distribution $p(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a}) = \mathscr{P}^{\boldsymbol{a}}_{\boldsymbol{s}\boldsymbol{s}'}$. For each transition, the agent also receives a reward $r \in \mathbb{R}$ that depends on $\boldsymbol{s}$ and $\boldsymbol{a}$, which we also write as $\mathscr{R}_{\boldsymbol{s}\boldsymbol{a}}$. The agent chooses actions $\boldsymbol{a}$ in the current state $\boldsymbol{s}$ according

to a policy $\pi(\boldsymbol{a}|\boldsymbol{s})$. We will consider an average reward setting where the goal of the agent is to find an optimal policy that will maximize the average reward

$$J(\pi) = \mathbb{E}[\mathscr{R}_{\boldsymbol{sa}}] = \iint \mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})\mathscr{R}_{\boldsymbol{sa}} \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a}, \tag{2.1}$$

where $\mu^{\pi}(\boldsymbol{s})$ is the state visit distribution of policy π.

Information-theoretic policy search was introduced with the relative entropy policy search (REPS) algorithm [Peters et al., 2010]. We will start by defining a constrained optimization problem for solving the discussed average reward reinforcement learning setting and, subsequently, add an information-theoretic constraint to make the optimization problem feasible.

Constraints for the State Distribution.

The objective of our optimization problem is given in Equation (3.17) which is supposed to be maximized with respect to $\mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})$. The agent can not freely choose the state distribution $\mu^{\pi}(\boldsymbol{s})$ of the policy, since $\mu^{\pi}(\boldsymbol{s})$ needs to comply with the policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ and the system dynamics $\mathscr{P}^{\boldsymbol{a}}_{\boldsymbol{ss}'}$, i.e.,

$$\forall \boldsymbol{s}' : \mu^{\pi}(\boldsymbol{s}') = \int_{\boldsymbol{s},\boldsymbol{a}} \mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})\mathscr{P}^{\boldsymbol{a}}_{\boldsymbol{ss}'} \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a}.$$

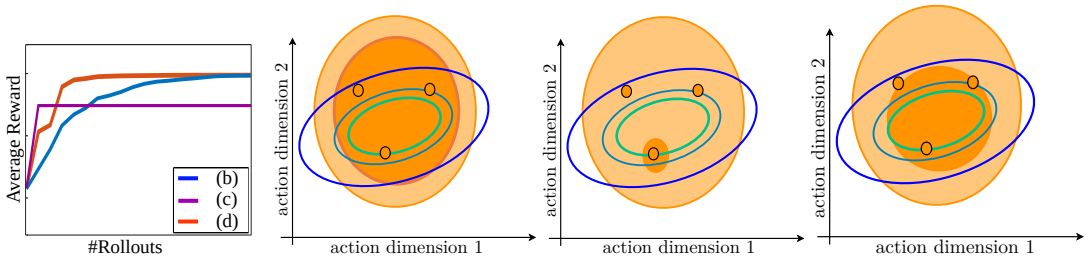
However, direct matching of the state probabilities is not feasible in continuous state spaces. Instead, we introduce state features $\boldsymbol{\phi}(\boldsymbol{s})$ and only match the feature averages

$$\int \boldsymbol{\phi}(\boldsymbol{s}')\mu^{\pi}(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s}' = \iiint \mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})\mathscr{P}^{\boldsymbol{a}}_{\boldsymbol{ss}'}\boldsymbol{\phi}(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a} \, \mathrm{d}\boldsymbol{s}'. \tag{2.2}$$

For example, if we use all linear and squared terms of $\boldsymbol{s}$ in our feature vector $\boldsymbol{\phi}(\boldsymbol{s})$, we match the mean and the variance under both distributions. Additionally, we require the joint probability $p(\boldsymbol{s},\boldsymbol{a}) = \mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})$ to define a probability distribution, i.e., its integral has to evaluate to

$$1 = \iint \mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a}. \tag{2.3}$$

While the optimization criterion given in Equation (3.17) with constraints in Equations (2.2,2.3) describes the general average reward reinforcement learning problem, it does not consider that we have typically estimated $\mathscr{R}_{\boldsymbol{sa}}$ and $\mathscr{P}^{\boldsymbol{a}}_{\boldsymbol{ss}'}$ from a limited amount of data. Hence, we do not want the agent to 'jump' to the optimum of this problem but instead balance exploitation versus exploration. In REPS [Peters et al., 2010], this exploitation-exploration trade off is balanced by the insight that the loss of information in the policy updates should be limited. Independently, such regularization was also suggested and motivated from different perspectives by other authors. Still and Precup [2011] showed that the policy update with maximum information gain results in a similar solution and Azar et al. [2012] motivated a similar update as punishing the distance between the controlled system and the uncontrolled one. Finally, Rawlik et al. [2012] showed that even previous probabilistic policy search approaches are closely related. All these arguments have lead to similar solutions despite their different motivations and the resulting algorithms work well on benchmark problems.



(a) Learning Curves **(b)** Not greedy enough **(c)** Too greedy **(d)** Balanced Exploration

Figure 2.1.: Schematic sketch of the behavior of policy updates. Solid lines show the contours of a quadratic reward function. The orange shaded area delimited by the dashed lines illustrates the area within two times the standard deviation of the sampling policy and the orange dots represent samples from the original policy. The darker shaded area delimited by the dotted line indicates the the possible policy update. The policy update needs to be balanced, such that it converges quickly to the local optimum, while not converging too fast to miss it. Such a balance is chosen by the user by specifying the relative entropy bound. Due to the relative entropy bound, the step size of the update is invariant to the task, the reward function or the parametrization of the policy. a) Learning curves corresponding to illustrations b-c. b) The update is not greedy enough, the policy will take too long to converge. c) The update is too greedy, the policy will not find the optimal solution. d) The update balances exploration and exploitation such that the policy quickly converges to a local optimum.

Staying Close to the Data by Information Theoretic Constraints.

As motivated by Peters et al. [2010], one key feature of effective PS methods is to limit the loss of information between policy updates, i.e., while we want to maximize the average reward of the new policy, we also want to stay close to the 'data', i.e., the state action distribution $q(\boldsymbol{s},\boldsymbol{a})$ of the old policy. Staying close to the data can be achieved by limiting the Kullback-Leibler (KL) divergence between the observed data $q(\boldsymbol{s},\boldsymbol{a})$ and the next policy, i.e.,

$$\epsilon \geq D_{\mathrm{KL}}\left(\mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s}) \,||\, q(\boldsymbol{s},\boldsymbol{a})\right) = \iint \mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s}) \log \frac{\mu(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})}{q(\boldsymbol{s},\boldsymbol{a})} \,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{a}. \tag{2.4}$$

Maximizing Equation (3.17) under the constraints of Equations (2.2, 2.3, 2.4) yields the optimization problem that defines the REPS method [Peters et al., 2010] which is the basis for our hierarchical policy search algorithm. The REPS optimization problem allows for a closed form solution for $\mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})$ that can be derived by the method of Lagrangian multipliers. It is given by

$$\mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s}) \propto q(\boldsymbol{s},\boldsymbol{a}) \exp\left(\frac{\mathscr{R}_{\boldsymbol{sa}} + \mathbb{E}\left[\boldsymbol{\theta}^{T}\boldsymbol{\phi}(\boldsymbol{s}')|\boldsymbol{s},\boldsymbol{a}\right] - \boldsymbol{\theta}^{T}\boldsymbol{\phi}(\boldsymbol{s})}{\eta}\right), \tag{2.5}$$

where η and $\boldsymbol{\theta}$ are Lagrangian parameters that are obtained by optimizing the dual function

$$g(\eta,\boldsymbol{\theta}) = \eta \log \iint q(\boldsymbol{s},\boldsymbol{a}) \exp\left(\frac{\mathscr{R}_{\boldsymbol{sa}} + \mathbb{E}[V(\boldsymbol{s}')] - V(\boldsymbol{s})}{\eta}\right) \mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{a} + \eta\epsilon, \tag{2.6}$$

of the original optimization problem. If we interpret the term $V(\boldsymbol{s}) = \boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{s})$ as value function linear in the parameters $\boldsymbol{\theta}$, the term $\mathscr{R}_{\boldsymbol{sa}} + \mathbb{E}[V(\boldsymbol{s}')] - V(\boldsymbol{s})$ can be viewed as advantage function. The Lagrangian parameter η is a scaling factor for the advantage. It becomes the temperature of the soft-max distribution defined in Equation (2.5) such that the KL-bound from Equation (2.4) is met.

Sample-Based REPS.

As the reward function and the old distribution $q(\boldsymbol{s}, \boldsymbol{a})$ can have an arbitrary structure, the integral contained in the dual function can typically not be obtained in closed form. However, the reward function can typically be evaluated on samples from $q(\boldsymbol{s}, \boldsymbol{a})$ and these samples can be used to approximate the dual function g. As a result, we can only evaluate the distribution $\mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})$ on a finite set of samples $\boldsymbol{s}_i$ and $\boldsymbol{a}_i$ with $i = 1 \ldots N$.

In order to sample new actions $\boldsymbol{a}$ in the next iteration of the algorithm, we need to find a parametric representation $\tilde{\pi}(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{\beta})$ of the policy that approximates the distribution $\pi(\boldsymbol{a}_i|\boldsymbol{s}_i)$, where $\boldsymbol{\beta}$ denotes the parameter vector of the policy. To do so, we aim to minimize the expected Kullback-Leibler divergence between $\pi(\boldsymbol{a}_i|\boldsymbol{s}_i)$ and the next parametric policy $\tilde{\pi}(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{\beta})$, i.e.,

$$
\begin{aligned}
&\arg\min_{\boldsymbol{\beta}} \int p(\boldsymbol{s}) D_{\mathrm{KL}}\left(\pi(\boldsymbol{a}|\boldsymbol{s}) \,||\tilde{\pi}(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{\beta})\right) \mathrm{d}\boldsymbol{s} \\
&= \arg\min_{\boldsymbol{\beta}} \int p(\boldsymbol{s}) \int \pi(\boldsymbol{a}|\boldsymbol{s}) \log \frac{\pi(\boldsymbol{a}|\boldsymbol{s})}{\tilde{\pi}(\boldsymbol{a}|\boldsymbol{s}; \boldsymbol{\beta})} \,\mathrm{d}\boldsymbol{a} \,\mathrm{d}\boldsymbol{s} \\
&\approx \arg\max_{\boldsymbol{\beta}} \sum_i \frac{p(\boldsymbol{s}_i, \boldsymbol{a}_i)}{q(\boldsymbol{s}_i, \boldsymbol{a}_i)} \log \tilde{\pi}(\boldsymbol{a}_i|\boldsymbol{s}_i; \boldsymbol{\beta}) + \mathrm{const} \\
&= \arg\max_{\boldsymbol{\beta}} \sum_i w_i \log \tilde{\pi}(\boldsymbol{a}_i|\boldsymbol{s}_i; \boldsymbol{\beta}),
\end{aligned} \tag{2.7}
$$

with

$$
w_i = \exp\left(\frac{r(\boldsymbol{s}_i, \boldsymbol{a}_i) + \mathbb{E}\left[\boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{s}') \middle| \boldsymbol{s}_i, \boldsymbol{a}_i\right] - \boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{s}_i)}{\eta}\right). \tag{2.8}
$$

Since the available samples are drawn from the distribution $q(\boldsymbol{s}, \boldsymbol{a})$, and not $p(\boldsymbol{s}, \boldsymbol{a})$, we need to perform importance sampling and divide by the sampling distribution $q(\boldsymbol{s}, \boldsymbol{a})$. Equation (2.7) is equivalent to computing the weighted maximum likelihood estimator for $\boldsymbol{\beta}$. In Section 2.8.3 we discuss how to compute the expectation over next states $\mathbb{E}\left[\boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{s}') \middle| \boldsymbol{s}_i, \boldsymbol{a}_i\right]$.

Illustration of Information Theoretic Policy Search.

We illustrate the concept of information theoretic policy search on a two-dimensional toy problem with a quadratic reward function in Figure 2.1(a). Here, the algorithm typically starts with a broad explorative policy. Given a set of samples generated using this policy, we update the policy such that the exploitation-exploration trade-off is well balanced. This trade-off is chosen by the relative entropy bound of the REPS algorithm. Due to the relative entropy bound, the step-size of the policy update is largely independent of the task, the reward function, or the representation of the policy. Changing the policy only by a small step leads to further exploration as we stay close to the old exploration policy as illustrated in Figure 2.1(b). In contrast, a large KL divergence for

the policy update greedily jumps to the best observed samples with very little further exploration, see Figure 2.1(c). This trade-off has to be chosen by the user by specifying the bound ϵ for the KL-divergence. Typically, we want to achieve a moderate exploration-exploitation trade-off as illustrated in Figure 2.1(d).

2.4 Learning Hierarchical Policies for Real Robot Reinforcement Learning

In this section, we will extend the REPS optimization problem to the hierarchical case, where the agent learns a hierarchical policy that is based on both a gating network and sub-policies. In order to obtain an efficient update rule for the hierarchical policy, we will rely on several insights detailed below. We will use these insights to derive three different learning algorithms that can be employed in different scenarios.

Hierarchical Policies.

More complex tasks often require multiple sub-policies, such as a forehand stroke, backhand stroke, smash or lob in tennis. Different sub-policies are appropriate in different states of the environment. In order to model a policy consisting of several sub-policies, we use a hierarchical policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ which consists of a set of sub-policies $\pi(\boldsymbol{a}|\boldsymbol{s},o)$ and a gating network $\pi(o|\boldsymbol{s})$ that selects the currently active sub-policy. Thus, the hierarchical policy can be represented as

$$\pi(\boldsymbol{a}|\boldsymbol{s}) = \sum_{o\in\mathcal{O}} \pi(o|\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o). \tag{2.9}$$

In order to determine the action $\boldsymbol{a}$ in state $\boldsymbol{s}$, we first sample a sub-policy from the gating network $\pi(o|\boldsymbol{s})$ and, subsequently, sample the action $\boldsymbol{a}$ from the specific sub-policy $\pi(\boldsymbol{a}|\boldsymbol{s},o)$. The benefit of representing multiple solutions has already been made evident in recent research results [Calinon et al., 2013, Daniel et al., 2012a].

Options as Latent Variables.

Sharing of experience between sub-policies, known as inter-option learning, is an important property of a hierarchical learning algorithm for data-efficient learning. To realize inter-option learning, we treat the problem of learning sub-policies as a latent variable problem, i.e., we assume to observe only state-action samples $\{\boldsymbol{s},\boldsymbol{a}\}$, but not the index of the generating sub-policy o. Instead, we infer the latent structure of the hierarchical policy that has generated the re-weighted samples. Expectation-maximization based methods can be used to iteratively estimate the responsibilities $p(o|\boldsymbol{s},\boldsymbol{a})$ of the sub-policies for each sample $\{\boldsymbol{s},\boldsymbol{a}\}$ and, subsequently, update the sub-policies where the influence of each sample $\{\boldsymbol{s},\boldsymbol{a}\}$ for the update of the sub-policy $\pi(\boldsymbol{a}|\boldsymbol{s},o)$ is weighted by the responsibility $p(o_j|\boldsymbol{s},\boldsymbol{a})$. Hence, the generated experience can be shared between the sub-policies. In Section 2.5, we integrate such an expectation-maximization mechanism into the REPS algorithm.

Learning Versatile Solutions.

Being able to represent multiple solutions does not force the learning algorithm to find different solutions. Thus, without further constraints, we are likely to find multiple sub-policies that concentrate on the same mode of the solution space. Versatility of sub-policies can be achieved if the

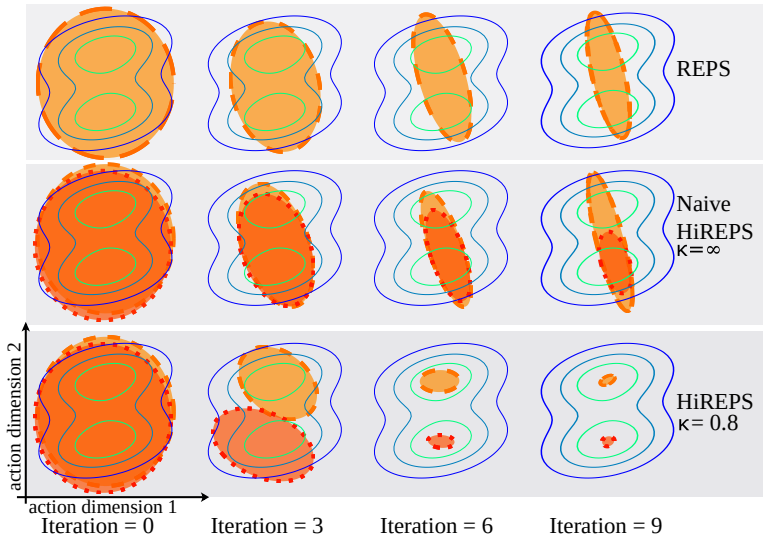


Figure 2.2.: Schematic sketch of the behavior of REPS, and HiREPS with ($\tilde{\kappa} = 0.8$) and without ($\tilde{\kappa} = \infty$) bounding of the gating's entropy. A two-dimensional, bi-modal reward function is shown in the contour plot lines. Two sub-policies are shown in dashed and dotted outlines respectively (outlines show 95% confidence boundaries). REPS and naive HiREPS average over both modes. Constrained HiREPS separates the sub-policies and is therefore able to find both modes.

sub-policies are clearly separated in the state-action space. To enforce this separation of the sub-policies, we limit the expected change in the entropy H of the responsibilities of the sub-policies, i.e.,

$$\kappa \geq \frac{\mathbb{E}_{\boldsymbol{s},\boldsymbol{a}}[H\big(p(o|\boldsymbol{s},\boldsymbol{a})\big)]}{\mathbb{E}_{q(\boldsymbol{s},\boldsymbol{a})}[H\big(q(o|\boldsymbol{s},\boldsymbol{a})\big)]} = \frac{\int\int \mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s}) \sum_{o\in\mathcal{O}} p(o|\boldsymbol{s},\boldsymbol{a}) \log p(o|\boldsymbol{s},\boldsymbol{a}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a}}{\int\int q(\boldsymbol{s},\boldsymbol{a}) \sum_{o\in\mathcal{O}} q(o|\boldsymbol{s},\boldsymbol{a}) \log q(o|\boldsymbol{s},\boldsymbol{a}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a}}, \quad (2.10)$$

where $\mathbb{E}_{q(\boldsymbol{s},\boldsymbol{a})}[H\big(q(o|\boldsymbol{s},\boldsymbol{a})\big)]$ is a constant and we write $\tilde{\kappa} = \mathbb{E}_{q(\boldsymbol{s},\boldsymbol{a})}[H\big(q(o|\boldsymbol{s},\boldsymbol{a})\big)]\kappa$ to simplify the notation, such that the constraint reads

$$\tilde{\kappa} \geq \mathbb{E}_{\boldsymbol{s},\boldsymbol{a}}[H\big(p(o|\boldsymbol{s},\boldsymbol{a})\big)] = -\int\int \mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s}) \sum_{o\in\mathcal{O}} p(o|\boldsymbol{s},\boldsymbol{a}) \log p(o|\boldsymbol{s},\boldsymbol{a}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a}. \quad (2.11)$$

A high entropy of the responsibility $p(o|\boldsymbol{s},\boldsymbol{a})$ indicates a high uncertainty in deciding which sub-policy has generated the state action pair, which implies that several sub-policies do overlap in the parameter space. By limiting the entropy, such overlapping is avoided and we ensure that different sub-policies concentrate on different and separate solutions.

We will discuss three settings for Hierarchical REPS (HiREPS). In the episodic setup, a single sub-policy is executed per episode, but the algorithm can choose between multiple sub-policies and adapt these to the current context. Subsequently, we will show how to sequence a fixed number of sub-policies with a finite horizon MDP formulation. Finally, we will use a infinite horizon MDP formulation to learn how to sequence a possibly infinite number of skills.

Illustration of Hierarchical Learning.

We illustrate the advantage of learning with multiple sub-policies on a toy task with a two dimensional action space and a bi-modal reward function (see Figure 2.2). In this task, the reward function consists of two attractors $\boldsymbol{g}_1, \boldsymbol{g}_2$ and a reward scaling matrix Σ_r, such that the reward function is given by

$$r(\boldsymbol{a}) = -\min_i \left((\boldsymbol{a}-\boldsymbol{g}_i)\Sigma_r(\boldsymbol{a}-\boldsymbol{g}_i)^T\right),$$

and has two local optima at $\boldsymbol{g}_1$ and $\boldsymbol{g}_2$, respectively. The illustration in Figure 2.2 demonstrates two key insights into PS methods. First, many standard policy search methods such as Policy Improvement with Path Integrals [Theodorou et al., 2010] or EM-based policy search methods [Kober et al., 2008] will be attracted to multiple optima in a multi-modal solution space and, therefore, converge slowly due to averaging over several modes [Neumann, 2011]. Second, we would like to represent a versatile solution space by learning all modes of the reward function. Figure 2.2 shows a qualitative comparison of HiREPS to the standard REPS algorithm which can only use one sub-policy and to the naive implementation of HiREPS that does not bound the sub-policies' entropy (i.e., $\tilde{\kappa} = \infty$). The single sub-policy algorithm tries to average over both modes and, takes a long time to converge. The naive HiREPS exhibits similar behavior. Both sub-policies are attracted by both modes. In most cases, both sub-policies will find the same mode and convergence will be slower. Thus, only introducing hierarchical policies without additional constraints cannot take full advantage of the increased flexibility. When limiting the entropy, however, the sub-policies quickly separate and concentrate on the two individual modes, allowing for a fast improvement of the policy without getting stuck between two modes.

2.5 Episodic Selection of Sub-Policies

We start our discussion of HiREPS with the continuous multi-armed contextual bandit setting. In this setting, the agent is presented with an initial state according to an initial state distribution $p^0(\boldsymbol{s})$ and has to select a sub-policy o as well as an action $\boldsymbol{a}$ according to this state. In this setting, an episode consists of executing exactly one sub-policy, afterwards the episode is terminated and the environment is reset. Thus, no state transition is modeled, as the whole episode consists of only one step, i.e., executing one sub-policy until it terminates.

2.5.1 Contextual Optimization of the Policy.

While we do not need to use the state distribution constraint from REPS, as we do not have to incorporate state transitions, the agent can still not freely choose its state action distribution $p(\boldsymbol{s},\boldsymbol{a}) = \mu^\pi(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})$ as the initial state distribution $p^0(\boldsymbol{s})$ is specified by the learning task. Hence, the estimated state distribution $\mu^\pi(\boldsymbol{s})$ has to satisfy $\mu^\pi(\boldsymbol{s}) = p^0(\boldsymbol{s})$ for all $\boldsymbol{s}$. As this requirement would result in an infinite number of constraints, we need to resort to matching feature averages, i.e.,

$$\hat{\boldsymbol{\phi}} = \sum_{o\in\mathcal{O}} \iint p(\boldsymbol{s},\boldsymbol{a},o)\boldsymbol{\phi}(\boldsymbol{s})\, \mathrm{d}\boldsymbol{s}\, \mathrm{d}\boldsymbol{a} \tag{2.12}$$

where $\hat{\boldsymbol{\phi}}$ denotes the average observed feature vector for the initial state

$$\hat{\boldsymbol{\phi}} = \iint q(\boldsymbol{s}, \boldsymbol{a}) \boldsymbol{\phi}(\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a}. \tag{2.13}$$

This constraint enforces that the learned state action distribution does not concentrate only on contexts where the task is easy to achieve, but considers all contexts sampled from the initial state distribution.

2.5.2 Resulting Optimization Problem.

For the resulting optimization problem, we combine the insights from the previous section on hierarchical policies with the contextual episodic learning constraint in Eq. (2.12). The episodic learning problem of multiple sub-policies is then given as

$$\begin{aligned}
\max_{\pi,\mu} J(\pi) = \max_{\pi,\mu} \sum \iint \mu^{\pi}(\boldsymbol{s}) \pi(o|\boldsymbol{s}) \pi(\boldsymbol{a}|\boldsymbol{s}, o) \mathscr{R}_{\boldsymbol{sa}} \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a}, \\
\text{s.t.} \quad \epsilon \geq D_{\mathrm{KL}}\left(p(\boldsymbol{s}, \boldsymbol{a}, o) || \, q(\boldsymbol{s}, \boldsymbol{a}) p(o|\boldsymbol{s}, \boldsymbol{a})\right), \\
\tilde{\kappa} \geq \mathbb{E}_{\boldsymbol{s},\boldsymbol{a}}\left[H\big(p(o|\boldsymbol{s}, \boldsymbol{a})\big)\right], \\
\hat{\boldsymbol{\phi}} = \sum_{o \in \mathscr{O}} \iint \mu^{\pi}(\boldsymbol{s}) \pi(\boldsymbol{a}|\boldsymbol{s}, o) \pi(o|\boldsymbol{s}) \boldsymbol{\phi}(\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a}, \\
1 = \sum \iint \mu^{\pi}(\boldsymbol{s}) \pi(o|\boldsymbol{s}) \pi(\boldsymbol{a}|\boldsymbol{s}, o) d\boldsymbol{s} d\boldsymbol{a}.
\end{aligned} \tag{2.14}$$

In this optimization problem, the responsibilities

$$p(o|\boldsymbol{s}, \boldsymbol{a}) = \frac{p(\boldsymbol{s}, \boldsymbol{a}, o)}{\sum_{o \in \mathscr{O}} p(\boldsymbol{s}, \boldsymbol{a}, o)},$$

occur inside the log-term of the KL-divergence. As the responsibilities contain a sum in the denominator, we also obtain the sum inside the log-term, preventing us from solving for $p(\boldsymbol{s}, \boldsymbol{a}, o)$ in closed form. However, we can resort to an iterative, expectation maximization (EM) update strategy shown in Daniel et al. [2012a]. In the expectation step, we first fix the sub-policies and compute the responsibilities $\tilde{p}(o|\boldsymbol{s}, \boldsymbol{a})$. In the maximization step, we replace $p(o|\boldsymbol{s}, \boldsymbol{a})$ in our optimization problem with the pre-computed responsibilities $\tilde{p}(o|\boldsymbol{s}, \boldsymbol{a})$ and, therefore, neglect that the responsibilities will change once we change the sub-policies. In Appendix A.1 we show that

the EM-step maximizes a lower bound of the original optimization problem. We use this iterative update strategy in all further algorithms. The adapted optimization problem reads as

$$\begin{aligned}
&\max_{\pi,\mu} J(\pi) = \max_{\pi,\mu} \mathbb{E}_{\boldsymbol{s},\boldsymbol{a},o}\left[\mathscr{R}_{\boldsymbol{s}\boldsymbol{a}}\right], \\
&\text{s.t.} \quad \epsilon \geq D_{\mathrm{KL}}\left(p(\boldsymbol{s},\boldsymbol{a},o) \,||\, q(\boldsymbol{s},\boldsymbol{a})\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})\right), \\
&\qquad \tilde{\kappa} \geq -\sum_{o\in\mathscr{O}} \iint p(\boldsymbol{s},\boldsymbol{a},o) \log \tilde{p}(o|\boldsymbol{s},\boldsymbol{a}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a}, \\
&\qquad \hat{\boldsymbol{\phi}} = \sum_{o\in\mathscr{O}} \iint p(\boldsymbol{s},\boldsymbol{a},o) \boldsymbol{\phi}(\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a}, \\
&\qquad 1 = \sum_{o\in\mathscr{O}} \iint p(\boldsymbol{s},\boldsymbol{a},o) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a}.
\end{aligned} \tag{2.15}$$

The above optimization problem can be solved by the method of Lagrange multipliers. The Lagrangian also allows for a closed form of $p(\boldsymbol{s},\boldsymbol{a},o)$ which is given as

$$p(\boldsymbol{s},\boldsymbol{a},o) \propto q(\boldsymbol{s},\boldsymbol{a})\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})^{1+\frac{\xi}{\eta}} \exp\left(\frac{\mathscr{R}_{\boldsymbol{s}\boldsymbol{a}} - V(\boldsymbol{s})}{\eta}\right), \tag{2.16}$$

which depends on the Lagrangian parameters ξ, η and $\boldsymbol{\theta}$ with $V(\boldsymbol{s}) = \boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{s})$. The Lagrangian parameter ξ is associated to the overlapping constraint in Eq. (2.11). In this update equation, the parameter ξ controls the spreading of the sub-policies. A higher value of ξ will force the sub-policies to spread further apart and reduce overlapping of sub-policies as the influence of state-action pairs with a high entropy of $p(o|\boldsymbol{s},\boldsymbol{a})$ is weakened. The derivation of the dual formulation is given in the appendix.

In Table 2.3, we give the algorithmic form of episodic HiREPS. The new parametric policy $\tilde{\pi}(\boldsymbol{a}|\boldsymbol{s},o;\boldsymbol{\beta}_o)$ has to be computed from the weighted samples by performing a weighted maximum likelihood estimate for the single sub-policy parameters $\boldsymbol{\beta}_o$ as well as the parameter vector for the gating policy. In HiREPS, we obtain a weight

$$w_{i,o} = \tilde{p}(o|\boldsymbol{s}_i,\boldsymbol{a}_i)^{1+\frac{\xi}{\eta}} \exp\left(\frac{\mathscr{R}_{\boldsymbol{s}\boldsymbol{a}} - V(\boldsymbol{s}_i)}{\eta}\right),$$

for each sample and each sub-policy. These weights can be used to update the policy. HiREPS does not make any assumptions on the form of the policy and many different representations could be considered. In Section 2.8, we show the details of the policy model we chose to implement for the evaluations, i.e., a soft-max gating policy and linear Gaussian sub-policies.

Since HiREPS does not assume knowledge of which sub-policy generated a sample, it can use all samples to update all sub-policies, also known as inter-option learning. The weights $w_{i,o}$ are then used to update the sub-policies $\pi(\boldsymbol{a}|\boldsymbol{s},o)$ and the gating policy $\pi(o|\boldsymbol{s})$.

2.6 Sequencing of Skills

Many real world tasks require not only one, but multiple sequential interactions. For example, in a game of tennis, we need to perform several tennis strokes in sequence in order to win the

Input: Information loss tolerance ϵ, Entropy tolerance $\tilde{\kappa}$, Number of sub-policies O, number of Iterations L, number of episodes per iteration M.
Initialize all $\pi(\boldsymbol{a}|\boldsymbol{s}, o)$ and $\pi(o|\boldsymbol{s})$.

for $l = 1$ to L ... # iterations

Collect samples

for $i = 1, \ldots, M$ (# episodes)

Sample initial state $\boldsymbol{s}_{1,i}$ from environment.

Sample action: $\boldsymbol{a}_i \sim q(\boldsymbol{a}|\boldsymbol{s}_i) = \sum_{o \in \mathcal{O}} \pi_{\text{old}}(o|\boldsymbol{s}_i)\pi_{\text{old}}(\boldsymbol{a}|\boldsymbol{s}_i, o)$.

Execute action $\boldsymbol{a}_i$ and observe reward $r(\boldsymbol{a}_i, \boldsymbol{s}_i)$.

Compute Responsibilities:

$$\tilde{p}(o|\boldsymbol{s}_i, \boldsymbol{a}_i) = p_{\text{old}}(o|\boldsymbol{s}_i, \boldsymbol{a}_i) = \frac{\mu^{\pi}_{\text{old}}(\boldsymbol{s}_i)\pi_{\text{old}}(\boldsymbol{a}_i|\boldsymbol{s}_i o_i)\pi_{\text{old}}(o|\boldsymbol{s}_i)}{\sum_{o \in \mathcal{O}} \mu^{\pi}_{\text{old}}(\boldsymbol{s}_i)\pi_{\text{old}}(\boldsymbol{a}_i|\boldsymbol{s}_i o_i)\pi_{\text{old}}(o|\boldsymbol{s}_i)} \text{ for all } i.$$

Minimize the dual function

$[\boldsymbol{\theta}^*, \eta^*, \xi^*] = \arg\min_{[\boldsymbol{\theta}, \eta, \xi]} g(\boldsymbol{\theta}, \eta, \xi)$.

Policy update:

Compute model distribution

$$\mu^{\pi}(\boldsymbol{s}_i)\pi(\boldsymbol{a}_i|\boldsymbol{s}_i o_i)\pi(o|\boldsymbol{s}_i) \propto \tilde{p}(o|\boldsymbol{s}_i, \boldsymbol{a}_i)^{1+\xi^*/\eta^*} \exp\left(\frac{R_i - V^*(\boldsymbol{s}_i)}{\eta^*}\right).$$

Estimate policies

$\pi(o|\boldsymbol{s})$ and $\pi(\boldsymbol{a}|\boldsymbol{s}, o)$ for all $o = 1 \ldots O$ by weighted ML estimates.

Output: Policies $\pi(\boldsymbol{a}, o|\boldsymbol{s})$

Table 2.1.: Episodic HiREPS. In each iteration the algorithm starts by sampling an sub-policy o from the gating policy $\pi(a|\boldsymbol{s})$ given the initial state $\boldsymbol{s}$ and an action $\boldsymbol{a}$ from $\pi(\boldsymbol{a}|\boldsymbol{s}, o)$ from the sub-policy. Subsequently, the action is executed to generate the reward $r(\boldsymbol{s}, \boldsymbol{a})$. The parameters η, ξ and $\boldsymbol{\theta}$ are determined by minimizing the dual-function g.

game. Just learning to return a ball is insufficient to win a game. Instead, the ball has to returned in such a way that future strokes can lead to situations where the opponent is unable to return the ball. Hence, tasks like tennis can be learned more efficiently if we learn a sequence of tennis strokes against an opponent. We will model the skill sequencing case with a limited number K of sequential decisions. We will denote the individual action selection problems as stages of the decision problem. Our goal is to maximize the sum of the expected reward over all stages, i.e.,

$$\begin{aligned} J &= \mathbb{E}_{\boldsymbol{a}_{1:K},\boldsymbol{s}_{1:K+1}}\left[r_{K+1}(\boldsymbol{s}_{K+1}) + \sum\nolimits_{k=1}^{K} r_k(\boldsymbol{s}_k,\boldsymbol{a}_k)\right] \\ &= \int \mu_{K+1}^{\pi}(\boldsymbol{s}) r_{K+1}(\boldsymbol{s}) d\boldsymbol{s} + \iint \sum_{k=1}^{K} \mu_k^{\pi}(\boldsymbol{s})\pi_k(\boldsymbol{a}|\boldsymbol{s}) r_k(\boldsymbol{s},\boldsymbol{a})\, \mathrm{d}\boldsymbol{s}\, \mathrm{d}\boldsymbol{a}, \end{aligned} \tag{2.17}$$

where $\mu_k^{\pi}(\boldsymbol{s})$ are the state distributions at for each decision step. The function $r_{K+1}(\boldsymbol{s}_{K+1})$ denotes the final reward for reaching state $\boldsymbol{s}_{K+1}$ and $r_k(\boldsymbol{s}_k,\boldsymbol{a}_k)$ denotes the reward for executing an action $\boldsymbol{a}_k$ in state $\boldsymbol{s}_k$. In the tennis example, the individual rewards could, for example, describe the energy efficiency of the movements, while the overall reward signal carries information about winning or losing the point. The sub-policy is given by

$$\pi_k(\boldsymbol{a}|\boldsymbol{s}) = \sum_{o\in\mathcal{O}} \pi_k(o|\boldsymbol{s})\pi_k(\boldsymbol{a}|\boldsymbol{s},o). \tag{2.18}$$

Alternatively, we can also share the sub-policies for all decision stages and only make the gating policy time dependent, i.e.,

$$\pi_k(\boldsymbol{a}|\boldsymbol{s}) = \sum_{o\in\mathcal{O}} \pi_k(o|\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s},o).$$

This formulation allows for reusing experience between the stages. Whether this property is useful depends on the task at hand. For example in tennis, the strokes at the decision stages can taken from the same skill library[1]. In the case of skill sequencing, we have one latent variable $o_{i,k}$ per rollout i and per decision stage k.

2.6.1 Connecting the Decision Stages.

The state distributions $\mu_k^{\pi}(\boldsymbol{s})$ are now connected through the transition dynamics $\mathscr{P}_{\boldsymbol{s}\boldsymbol{s}'}^{\boldsymbol{a}}$ yielding

$$\int \boldsymbol{\phi}(\boldsymbol{s}')\mu_{k+1}^{\pi}(\boldsymbol{s}')\, \mathrm{d}\boldsymbol{s}' = \sum_{o\in\mathcal{O}} \iiint \mathscr{P}_{\boldsymbol{s}\boldsymbol{s}'}^{\boldsymbol{a}} p_k(\boldsymbol{s},\boldsymbol{a},o)\boldsymbol{\phi}(\boldsymbol{s}')\, \mathrm{d}\boldsymbol{s}\, \mathrm{d}\boldsymbol{s}'\, \mathrm{d}\boldsymbol{a}. \tag{2.19}$$

In addition, the agent cannot freely choose the initial state distribution $\mu_1^{\pi}(\boldsymbol{s})$, but needs to match the given initial state distribution $p_1(\boldsymbol{s})$, which is implemented in the same way as for the episodic case. Typically the dynamics $\mathscr{P}_{\boldsymbol{s}\boldsymbol{s}'}^{\boldsymbol{a}}$ of a task are not known and need to be estimated from data.

[1] Except for the first stroke, as it is supposed to be the serve.

2.6.2 Skill Sequencing Optimization Problem.

After including the overlapping constraints from the previous section, the optimization problem reads

$$\max_{\pi_k, \mu_k^\pi} \mathbb{E}_{\boldsymbol{a}_{1:K}, \boldsymbol{s}_{1:K+1}} \left[r_{K+1}(\boldsymbol{s}_{K+1}) + \sum\nolimits_{k=1}^{K} r_k(\boldsymbol{s}_k, \boldsymbol{a}_k) \right],$$

$$\text{s.t.} \quad \epsilon \geq D_{\mathrm{KL}} \left(\mu_{K+1}^\pi(\boldsymbol{s}) || \, q_{K+1}(\boldsymbol{s}) \right),$$

$$\hat{\boldsymbol{\phi}}_1 = \int \boldsymbol{\phi}(\boldsymbol{s}') \mu_1^\pi(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s}',$$

$$\forall k \leq K : \epsilon \geq D_{\mathrm{KL}} \left(p_k(\boldsymbol{s}, \boldsymbol{a}, o) || \, q_k(\boldsymbol{s}, \boldsymbol{a}) \tilde{p}_k(o|\boldsymbol{s}, \boldsymbol{a}) \right),$$

$$\tilde{\kappa} \geq -\sum_{o \in \mathcal{O}} \iint p_k(\boldsymbol{s}, \boldsymbol{a}, o) \log \tilde{p}_k(o|\boldsymbol{s}, \boldsymbol{a}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a},$$

$$\int \boldsymbol{\phi}(\boldsymbol{s}') \mu_{k+1}^\pi(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s}' = \sum_{o \in \mathcal{O}} \iiint \mathscr{P}_{\boldsymbol{s}\boldsymbol{s}'}^{\boldsymbol{a}} p_k(\boldsymbol{s}, \boldsymbol{a}, o) \boldsymbol{\phi}(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{s}' \, \mathrm{d}\boldsymbol{a},$$

$$1 = \sum\nolimits_{\boldsymbol{s}, \boldsymbol{a}, o} p_k(\boldsymbol{s}, \boldsymbol{a}, o). \tag{2.20}$$

The resulting policy update rules are given by

$$p_k(\boldsymbol{s}, \boldsymbol{a}, o) \propto$$
$$q_k(\boldsymbol{s}, \boldsymbol{a}) \tilde{p}_k(o|\boldsymbol{s}, \boldsymbol{a})^{1 + \frac{\xi_k}{\eta_k}} \exp\left(\frac{r_k(\boldsymbol{s}, \boldsymbol{a}) + \mathbb{E}[V_{k+1}(\boldsymbol{s}')|\boldsymbol{s}, \boldsymbol{a}] - V_k(\boldsymbol{s})}{\eta_k} \right), \tag{2.21}$$

where one set of Lagrangian parameters $\xi_k, \eta_t, \boldsymbol{\theta}_k$ is computed for each stage. As we observe, we now obtain an individual value function $V_k(\boldsymbol{s}_k)$ for each decision stage. As in the standard REPS algorithm, the value functions are connected by the advantage function term that occurs inside the exponent in the policy. The skill sequencing algorithm is given in Table 2.2. The derivations of the dual function and the update rules are given in Appendix A.3. In Section 2.8.3, we discuss how to compute the expectation over next states $\mathbb{E}[V_{k+1}(\boldsymbol{s}')|\boldsymbol{s}, \boldsymbol{a}]$.

2.7 Infinite Horizon Skill Sequencing

An infinite horizon setting is needed for all repetitive tasks where we sequence an unknown, possibly infinite amount of sub-policies to fulfill the task. For example, when bouncing a ball on a paddle, the repetitive paddling movements induce a clear structure in the motion that suggests the use of sub-policies. In addition, we want to bounce the ball on the paddle for an infinite amount of time. In this setting, the agent needs to consider the state of the environment before each stroke.

Input: Information loss tolerance ϵ, entropy tolerance $\tilde{\kappa}$, number of sub-policies n, number of time steps K, number of iterations L.
Initialize all $\pi_k(\boldsymbol{a}|\boldsymbol{s}, o)$ and $\pi_k(o|\boldsymbol{s})$.

for $l = 1$ to L ... # iterations
  Collect samples
  for $i = 1, \ldots, M$ (# episodes)
    Sample initial state $\boldsymbol{s}_{1,i}$ from environment.
    for $k = 1, \ldots, K$ (# motor primitives)
      Sample action: $\boldsymbol{a}_{k,i} \sim q_k(\boldsymbol{a}|\boldsymbol{s}_{k,i}) = \sum_{o \in \mathcal{O}} \pi_{k,\text{old}}(o|\boldsymbol{s}_{k,i}) \pi_{k,\text{old}}(\boldsymbol{a}|\boldsymbol{s}_{k,i}, o)$.
      Execute action $\boldsymbol{a}_{k,i}$, observe next state $\boldsymbol{s}_{k+1,i}$ and reward $r(\boldsymbol{a}_{k,i}, \boldsymbol{s}_{k,i})$.
    Observe Final Reward: $r(\boldsymbol{s}_{K+1,i})$.
  Compute Responsibilities:
  $\tilde{p}_k(o|\boldsymbol{s}_{k,i}, \boldsymbol{a}_{k,i}) = p_{k,\text{old}}(o|\boldsymbol{s}_{k,i}, \boldsymbol{a}_{k,i})$ for all k and i.
  Minimize the dual function:
  $[\boldsymbol{\theta}_{1:K+1}{}^*, \eta_{1:K+1}{}^*, \xi_{1:K+1}{}^*] = \arg\min_{[\boldsymbol{\theta}_{1:K+1}, \eta_{1:K+1}, \xi_{1:K+1}]} g(\boldsymbol{\theta}_{1:K+1}, \eta_{1:K+1}, \xi_{1:K+1})$.
  Policy update:
  for $k = 1, \ldots, K$
    Compute model distribution
    $\mu_k^{\pi}(\boldsymbol{s}_{k,i}) \pi_k(\boldsymbol{a}_{k,i}|\boldsymbol{s}_{k,i}, o) \pi_k(o|\boldsymbol{s}_{k,i}) \propto$
    $\tilde{p}_k(o|\boldsymbol{s}_{k,i}, \boldsymbol{a}_{k,i})^{1+\xi_k{}^*/\eta_k{}^*} \exp\left(\frac{R_{k,i} + \mathbb{E}[V_{k+1}^*(\boldsymbol{s}')] - V_k^*(\boldsymbol{s}_{k,i})}{\eta_k{}^*}\right)$.
    Estimate policies
    $\pi_k(o|\boldsymbol{s})$ and $\pi_k(\boldsymbol{a}|\boldsymbol{s}, o)$ by weighted ML estimates.

Output: Policies $\pi_k(\boldsymbol{a}, o|\boldsymbol{s})$ for all $k = 1, \ldots, K$

Table 2.2.: Time-indexed HiREPS. In each iteration the algorithm starts by sampling from the policy π_1 given the initial state $\boldsymbol{s}_1$ and executes the sampled action to generate the next state $\boldsymbol{s}_2$. From this state, the next action is sampled with policy π_2. This procedure is repeated until the final time-step is reached. The algorithm observes state transitions and rewards for each step k and the final reward signal $r(\boldsymbol{s})$. The parameters $\eta_{1:K+1}$, $\xi_{1:K+1}$ and $\boldsymbol{\theta}_{1:K+1}$ are determined by minimizing the dual-function g, where $\eta_{1:K+1}$ and $\xi_{1:K+1}$ are vectors containing the Lagrangian parameters η_k and ξ_k for each decision step.

The traditional objective in an infinite horizon MDP is to use the discounted accumulated future rewards, i.e.,

$$R = \sum_{t=0}^{\infty} \gamma^t r_t,$$

where $\gamma < 1$ is a discount factor. Such an objective can be easily transferred to the average reward setting by introducing a reset probability γ, where the agent jumps to a state sampled from the initial state distribution $\mu^0(\boldsymbol{s})$ with probability γ and transitions to the next state with probability $(1-\gamma)$ [van Hoof et al., 2015]. Thus, we obtain a transformed transition probability distribution given by

$$\tilde{p}(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a}) = \gamma p(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a}) + (1-\gamma)\mu^0(\boldsymbol{s}), \tag{2.22}$$

where γ is the reset probability. Thus, the discount factor is considered as a termination probability. The constraint ensuring the state transition probabilities are satisfied reads as

$$\int \boldsymbol{\phi}(\boldsymbol{s}')\mu^{\pi}(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}' = \sum_{o\in\mathcal{O}} \iiint \tilde{\mathscr{P}}^{\boldsymbol{a}}_{\boldsymbol{s}\boldsymbol{s}'} p(\boldsymbol{s},\boldsymbol{a},o)\boldsymbol{\phi}(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{s}'\,\mathrm{d}\boldsymbol{a}. \tag{2.23}$$

The optimization problem for the infinite horizon case reads as

$$\begin{aligned}
&\max_{\pi,\mu^{\pi}} J(\pi) = \max_{\pi,\mu^{\pi}} \mathbb{E}_{\boldsymbol{s},\boldsymbol{a},o}[\mathscr{R}_{\boldsymbol{s}\boldsymbol{a}}],\\
&\quad \text{s.t.} \quad \epsilon \geq D_{\mathrm{KL}}\left(p(\boldsymbol{s},\boldsymbol{a},o)||\, q(\boldsymbol{s},\boldsymbol{a})\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})\right),\\
&\qquad\qquad \tilde{\kappa} \geq -\sum_{o\in\mathcal{O}} \iint p(\boldsymbol{s},\boldsymbol{a},o)\log\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{a},\\
&\int \boldsymbol{\phi}(\boldsymbol{s}')\mu^{\pi}(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}' = \sum_{o\in\mathcal{O}} \iiint \gamma\mathscr{P}^{\boldsymbol{a}}_{\boldsymbol{s}\boldsymbol{s}'} p(\boldsymbol{s},\boldsymbol{a},o)\boldsymbol{\phi}(\boldsymbol{s}') + (1-\gamma)\mu^0(\boldsymbol{s})\boldsymbol{\phi}(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{s}'\,\mathrm{d}\boldsymbol{a},\\
&\qquad\qquad 1 = \sum_{o\in\mathcal{O}} \iint p(\boldsymbol{s},\boldsymbol{a},o)\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{a},
\end{aligned} \tag{2.24}$$

with the resulting policy update

$$p(\boldsymbol{s},\boldsymbol{a},o) \propto q(\boldsymbol{s},\boldsymbol{a})\tilde{p}(o|\boldsymbol{s},\boldsymbol{a})^{1+\xi/\eta} \exp\left(\frac{\mathscr{R}_{\boldsymbol{s}\boldsymbol{a}} + \mathbb{E}[V(\boldsymbol{s}')|\boldsymbol{s},\boldsymbol{a}] - V(\boldsymbol{s})}{\eta}\right), \tag{2.25}$$

where

$$\mathbb{E}[V(\boldsymbol{s}')|\boldsymbol{s},\boldsymbol{a}] = \boldsymbol{\theta}^T\left[\int \gamma\mathscr{P}^{\boldsymbol{a}}_{\boldsymbol{s}\boldsymbol{s}'}\boldsymbol{\phi}(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}' + \int (1-\gamma)\mu^0(\boldsymbol{s})\boldsymbol{\phi}(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}'\right].$$

In contrast to the finite horizon case, the value function, the policy and the state distributions are now stationary instead of time dependent.

Input: Information loss tolerance ϵ, Entropy tolerance $\tilde{\kappa}$, Number of sub-policies O, number of Iterations L, number of rollouts per iteration M, reset probability γ.
Initialize all $\pi(\boldsymbol{a}|\boldsymbol{s}, o)$ and $\pi(o|\boldsymbol{s})$.

for $l = 1$ to L ... # iterations

Collect samples

for $i = 1, \ldots, M$ (# rollouts)

t = 1; reset = 0;

Sample initial state $\boldsymbol{s}_{i,t}$ from environment.

while reset $< \gamma$

Sample action $\boldsymbol{a}_{i,t} \sim q(\boldsymbol{a}|\boldsymbol{s}_{i,t}) = \sum_{o \in \mathcal{O}} \pi_{\text{old}}(o|\boldsymbol{s}_{i,t}) \pi_{\text{old}}(\boldsymbol{a}|\boldsymbol{s}_{i,t}, o)$.

Execute action $\boldsymbol{a}_{i,t}$ and observe reward $r_{i,t}(\boldsymbol{a}_{i,t}, \boldsymbol{s}_{i,t})$.

Observe reward $r_{i,t}(\boldsymbol{a}_{i,t}, \boldsymbol{s}_{i,t})$ and next state $\boldsymbol{s}_{i,t+1}$.

Sample reset value.

Compute Responsibilities:

$\tilde{p}(o|\boldsymbol{s}_{i,t}, \boldsymbol{a}_{i,t}) = p_{\text{old}}(o|\boldsymbol{s}_{i,t}, \boldsymbol{a}_{i,t}) = \frac{p_{\text{old}}(\boldsymbol{s}_{i,t}, \boldsymbol{a}_{i,t}, o)}{\sum_{o \in \mathcal{O}} p_{\text{old}}(\boldsymbol{s}_{i,t}, \boldsymbol{a}_{i,t}, o)}$ for all i, t.

Minimize the dual function

$[\boldsymbol{\theta}^*, \eta^*, \xi^*] = \arg\min_{[\boldsymbol{\theta}, \eta, \xi]} g(\boldsymbol{\theta}, \eta, \xi)$.

Policy update:

Compute model distribution

$p(\boldsymbol{s}_{i,t}, \boldsymbol{a}_{i,t}, o) \propto \tilde{p}(o|\boldsymbol{s}_{i,t}, \boldsymbol{a}_{i,t})^{1+\xi^*/\eta^*} \exp\left(\frac{R_{i,t} + \mathbb{E}[V(\boldsymbol{s}')] - V^*(\boldsymbol{s}_{i,t})}{\eta^*}\right)$.

Estimate policies

$\pi(o|\boldsymbol{s})$ and $\pi(\boldsymbol{a}|\boldsymbol{s}, o)$ for all $o = 1 \ldots O$ by weighted ML estimates.

Output: Policies $\pi(\boldsymbol{a}, o|\boldsymbol{s})$

Table 2.3.: Infinite Horizon HiREPS. The algorithm follows the form of the episodic implementation, however one iteration produces state, action, reward triples for each time step in each rollout of an iteration. In the infinite horizon case, a model $\mathscr{P}^{\boldsymbol{a}}_{\boldsymbol{s}\boldsymbol{s}'}$ is required to compute $\mathbb{E}[V(\boldsymbol{s}')]$.

2.8 Algorithmic Design Choices

Having shown the theoretical foundation of HiREPS, we use this Section to explain implementation details concerning the policy representation and parametrizations as well as feature representations and model learning. The choices detailed in this Section represent only some of the possible implementations and are design choices. The HiREPS framework as specified in the previous Section is independent of these design choices and can be used with arbitrary representations for the gating, the sub-policies as well as arbitrary feature representations. HiREPS only requires that the policy representation can be updated using weighted samples.

2.8.1 Policy Representation

We implemented the hierarchical policy using a soft-max gating network $\pi(o|\boldsymbol{s})$ and linear Gaussian sub-policies $\pi(\boldsymbol{a}|\boldsymbol{s},o)$. The gating network $\pi(o|\boldsymbol{s})$ chooses which option to activate according to the model

$$\pi(o=k|\boldsymbol{s}) = \frac{\exp\left(\boldsymbol{\phi}_G(\boldsymbol{s})^T\boldsymbol{\beta}_{G,k}\right)}{\sum_i^O \exp\left(\boldsymbol{\phi}_G(\boldsymbol{s})^T\boldsymbol{\beta}_{G,i}\right)},$$

where $\boldsymbol{\beta}_{G,1\ldots O}$ are the parameter vectors defining the influence of the sub-policies and $\boldsymbol{\phi}_G$ are the feature representations of the state used for the gating network. While this model itself is relatively simple, the complexity and expressiveness of the resulting policy depends largely on the feature transformations employed for the individual components of the hierarchical policy. In HiREPS, we are free to choose different feature transformations for the sub-policies, the gating and the value function, where the value function features are the features in the primal optimization problem. In the presented experiments, the sub-policies themselves are represented by linear Gaussian policies, i.e.,

$$\pi(\boldsymbol{a}|\boldsymbol{s},o;\boldsymbol{\beta}_o) = \mathcal{N}\left(\boldsymbol{a}|\boldsymbol{\mu}_o + \boldsymbol{F}_o\boldsymbol{s},\boldsymbol{\Sigma}_o\right),$$

where $\boldsymbol{\beta}_o = \left[\boldsymbol{\mu}_o, \boldsymbol{F}_o, \boldsymbol{\Sigma}_o\right]$ is the parameter tuple describing sub-policy o. The sub-policies could also be defined on a feature transformation of the state. Equally well, non-linear sub-policies instead of the linear Gaussians could be used. However, since HiREPS performs a piecewise linear approximation in the state space using multiple sub-policies, linear sub-policies are usually sufficient. Both, the gating policy as well as the sub-policies can easily be updated using the sample weights computed by HiREPS. The update equations, as shown in the appendix, yield a weight matrix with one row per sample and cone column per sub-policy. The gating requires the full weight matrix as well as the responsibilities to be updated whereas each sub-policy is updated independently using the respective column of the weight matrix to weigh all state-action samples.

Generally, HiREPS can be initialized with many more sub-policies than solutions are expected to be available for the problem at hand. The gating network $\pi(o|\boldsymbol{s})$ will reduce the probability of selecting options that are not concentrated on good solutions to zero and effectively ignore these unnecessary sub-policies. Options can also be pruned throughout the learning process as in the first series of experiments described in Section 2.9.1. There, we report the number of actual solutions found. In this case, whenever the prior $p(o) = \int \pi(o|\boldsymbol{s})\mu(\boldsymbol{s})d\boldsymbol{s}$ of sub-policy activation gets too small (i.e., $p(o) < 10^{-4}$) we delete the sub-policy. However, in order to avoid

sub-policies getting deleted too quickly, we assure that each sub-policy gets a minimum amount of samples in the sampling process. We also bound the minimum variance of our Gaussian sub-policies to small values in order to avoid singularities. After the sampling process, we evaluate the quality of the exploration-free policy (i.e., without variance) found so far.

2.8.2 Feature Based State Representation

We use feature transformations for representing the value function as well as achieving a higher flexibility in the gating policy. Different feature representations can be used for the gating and the value function. In the presented experiments, the gating network $\pi(o|\boldsymbol{s})$ is constructed on squared expansion of the state space. Thus, the squared features are constructed as the vector of all linear and squared combinations of the state dimensions. For example, for a two-dimensional state space the feature vector would be given by

$$\boldsymbol{\phi}([s_1, s_2]) = [1, s_1, s_2, s_1^2, s_2^2, s_1 s_2].$$

While the squared expansion of the state space can also be used as feature representation for the value function, more complex tasks often benefit from a more flexible representation, such as a kernel based feature transformation. In our experiments we used either kernel based features or features based on a Fourier transformation as shown by [Konidaris et al., 2011]. The kernel based features used a squared exponential kernel, i.e.,

$$\left[\boldsymbol{\phi}_V\right]_i(\mathbf{s}) = \exp\left(-\frac{1}{2}(\mathbf{s}-\mathbf{s}_i)^T \boldsymbol{\Lambda}^{-1}(\mathbf{s}-\mathbf{s}_i)\right),$$

where $\boldsymbol{\Lambda}$ is a diagonal matrix denoting the bandwidth of the kernel and the index i iterates over the reference set of observed states.

Choosing sufficiently expressive features for the representation of the value function is crucial to the success of the learning algorithm. As an example, we can consider a contextual bandit task, where the robot has to select the number of steps to take to walk to a goal position. The context in this task is given by the robot's initial distance to the goal. If the robot has access to insufficient features, e.g., only a constant feature, it cannot differentiate between the different starting positions, and, hence, cannot learn to adapt its policy to the initial position. This effect is compounded in the infinite horizon setting, were successful policies often depend on reaching intermediate states with potentially low rewards on the path to the goal states.

2.8.3 Model Learning

For the finite horizon, as well as the for the infinite horizon formulation, HiREPS requires a transition model $\mathscr{P}_{ss'}^{\boldsymbol{a}}$. In this chapter, we use a sample based transition model, which reproduces previously observed state transitions and does not necessarily generalize. Using a sample based transition model effectively results in computing $V(\boldsymbol{s}')$ directly based on one single observed state transition instead of computing the expectation $\mathbb{E}[V(\boldsymbol{s}')|\boldsymbol{s}, \boldsymbol{a}]$ in the update, Equation (2.25), as well as in the corresponding dual function in Eq. (A.29) as given in Appendix A.4. Replacing the expectation in such a manner is only feasible if the controlled system has limited stochasticity. However, it is indeed still sufficiently robust to solve the real robot tasks as presented in the experimental Section. For systems with more stochasticity, van Hoof et al. [2015] show how to learn a better model that is able to generalize if sufficient data is available.

2.8.4 Sample Efficiency

One of the important trade-offs when using iterative policy update methods is the number of rollouts performed per iteration. More rollouts per iteration result in more stable policy updates but also increase the number of overall rollouts required to learn a task. When using sample based policy update techniques in high dimensional action spaces, the number of samples is crucial. Using fewer samples than action dimensions will lead to an underestimate of the variance or the variance might even collapse, as the new policy is just based on the available samples. In HiREPS, we are not restricted to using only samples from the previous iteration of rollouts when computing the sample weights. By considering samples from M multiple past iterations, we can stabilize the policy update while retaining a fast learning speed. As a general rule of thumb, keeping three times as many samples as used per rollout can stabilize the algorithm against aggressive choices in the number of rollouts per iteration (lower than number of parameters) or high ϵ (≥ 1.5). When keeping samples from previous iterations, we usually define our current state-action distribution $q(\boldsymbol{s}, \boldsymbol{a})$ to be given by the collection of these samples. Alternatively, importance weighting schemes can be employed to incorporate samples from previous iterations. However, in our experience, these importance weighting schemes are often prone to destabilizing the learning algorithm.

2.8.5 Parametrized Trajectories

For the results of this chapter, we often rely on movement primitives that inherently encode temporal correlations, such that each sub-policy $\pi(\boldsymbol{a}|\boldsymbol{s}, o)$ describes the parametrization of one movement primitives. We are using the extended version of DMPs by Kober et al. [2010a], that parametrizes trajectories using a combination of weighted basis functions as well as the end point g and final velocity $\dot{g}$ of the trajectory. DMPs are based on a simulated spring damper system $\ddot{x} = \alpha(\dot{g} - \dot{x}) + \beta(g - x)$ with damping factor α and spring stiffness β. The spring damper system will result in a trajectory that reaches the desired goal and goal velocity in a direct manner. However, the spring damper system alone is insufficient to encode arbitrary shapes along the trajectory. To represent arbitrary shapes, DMPs modulate the dynamical system with a forcing function $f(z, v) = \boldsymbol{\Phi}(z)^T v$ that depends on the phase z of the movement, the basis functions $\boldsymbol{\Phi}(z)$ and the basis functions weights v. Using the forcing function to deviate from the direct path of the spring damper system allows DMPs to generate complicated trajectory shapes. To ensure that the desired final goal position and velocity constraints are met, the forcing function depends on the exponentially decaying phase variable z, such that the forcing function does not influence the system towards the end of the trajectory.

If a demonstration trajectory, for example from kinesthetic teach-in, exists, basis function weights v that reproduce the shape of that motion as well as the goal position and velocity can be computed to initialize the learning process. After initialization, the basis function weights as well as the desired goal position and velocity are the parameters that the reinforcement learner optimizes over. In the presented robot learning experiments we learn one DMP per degree of freedom. It is important to note that the output of the DMPs only represents the desired trajectories, which are then usually tracked by an internal controller of the robot itself. This distinction is important because even if the robot is unable to track trajectories, e.g., due to gain or torque limitations, the

RL agent can adapt the DMP such that the resulting trajectory still solves the task. When using DMPs, each sub-policy lasts until the DMP has finished execution.

2.8.6 Computational Complexity

The presented algorithm depends on the three key computations: solving the dual function for HiREPS, fitting a gating policy and fitting the individual sub-policies. The optimization of the dual problem can be performed using existing optimizers such as, for example, the Broyden-Fletcher-Goldfarb-Shannon algorithm. The optimization can often be accelerated by providing the first and second derivative of the dual function. While the actual complexity depends on which algorithm is used, we can analyze the problem itself. The number of open parameters in this optimization problem stays constant with the number of options. However, the number of open parameters depends directly on the dimensionality of the features $\boldsymbol{\phi}_V(\boldsymbol{s})$ for the value function. Thus, the complexity of the optimization problem depends on the feature representation rather than on the number of options.

Fitting a gating policy based on the weights computed through the HiREPS optimization problem yields a multiclass classification problem. For example, multiclass logistic regression can be used. As before, a solution can be found through the use of third party optimization software. However, the number of open parameters scales multiplicative with the dimensionality of the gating feature representation and the number of options. Thus, for high dimensional gating features and a large number of options, the gating optimization problem can become computationally expensive.

Finally, the sub-policies have to be fitted. Given the gating, the sub-policies are independent and each sub-policy can be fitted individually using a weighted maximum-likelihood estimate. In the case of linear Gaussians, the complexity of this operation is governed by a linear regression operation which depends on the dimensionality of the sub-policy feature representation. However, since these optimizations are performed individually per option, the computational effort for this last step is usually negligible.

2.8.7 Hyperparameter Tuning

The presented method exposes three main hyperparameters to be set by the practitioner. These are the number of options O, the entropy bound κ as well as the relative entropy bound ϵ. The number of options can usually be chosen generously, i.e., around 20 seems to be reasonable for a wide range of problems. The algorithm will prioritize more promising options such that even if too many options are initialized, only options which yield high rewards will be sampled from after the first few iterations. The entropy bound κ is probably the most important parameter to consider since it does not have a clear equivalent in existing approaches. However, our experiments showed that a value of 0.9 seems to work well in almost all cases and no major tuning was necessary. The parameter ϵ is probably the parameter that is the most tuning-intensive in the proposed method, especially if the total number of episodes is crucial, e.g., in real robot experiments. In our experience values for ϵ between 0.5 and 1.5 are reasonable and, most often, we would start a new task with $\epsilon = 1$. Changing these parameters certainly influences the learning speed of the proposed method. However, while sub-optimal settings may lead to slower convergence, they usually do

not prevent successful learning. Thus, in our experience, the algorithm is generally robust in that even sub-optimal settings will lead to convergence.

2.9 Experimental Evaluations

We validated the different presented formulations of HiREPS on both simulated and real robot tasks. We compare our algorithm to the non-hierarchical counterpart, REPS [Peters et al., 2010]. Evaluating against REPS is interesting, as the two algorithms share the same basis. In effect, REPS is equivalent to our algorithm with just one sub-policy. This similarity allows us to directly investigate the influence of adding a hierarchical policy representation without additional confounders. The experimental section is structured analogously to the structure of Section 2.4, i.e., we first report results on the contextual episodic settings, subsequently the skill sequencing problem and finally the infinite horizon setting. For the contextual setting as well as for the skill sequencing, we report real robot results. In both cases, we start by presenting results on related simulated experiments to better investigate the properties of the presented algorithms as well as to compare to the REPS algorithm. For all presented results, we have optimized the parameters of the algorithms to deliver the best performance and both algorithms receive the same number of samples per iteration. For all experiments, if not noted otherwise, the experiments were repeated ten times to produce the errorbars reported in the results.

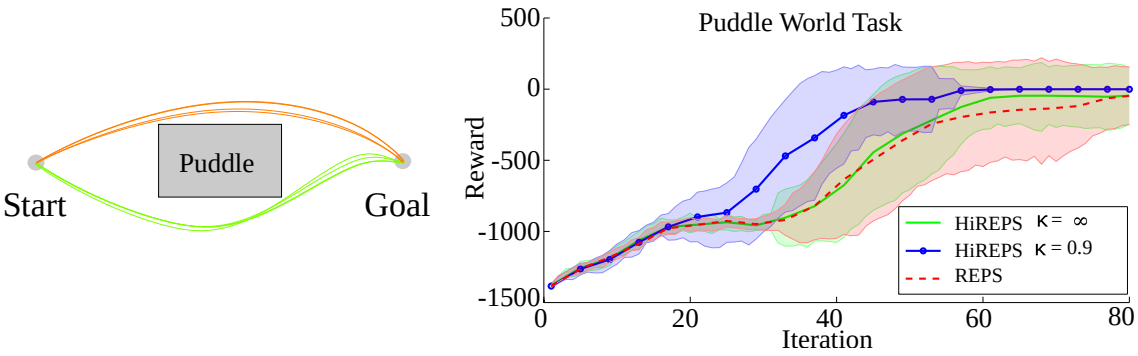
2.9.1 Episodic Skill Based Tasks

We start by presenting results for the episodic formulation of HiREPS. In the contextual setting, as well as for the skill sequencing setting in the next part, we start by presenting results of simpler experiments that allow a more in-depth analysis of the algorithms and then proceed to more complex tasks.

Puddle World Experiment

In a first toy task, we test HiREPS on a variation of the puddle world [Sutton, 1996]. While this task is of limited difficulty it is interesting as it is a well known setting which exhibits the averaging problem of interest to us. Additionally, the simplicity of the problem allows us to thoroughly assess the quality of the solutions found by the RL agent, which is often difficult in real robot tasks. Our version differs from the standard version by having a continuous action space instead of a discrete one. While the agent proceeds with a constant velocity along the x-dimension of the environment, it has to learn the DMPs shape parameters such that the puddle is avoided. Thus, the actions $\mathbf{a}$ of our sub-policies are five-dimensional vectors. The reward of the task is given by the negative length of the line segments, which encourages shorter solutions. An additional punishment occurs for passing through the puddles. The arrangement of the puddles can be seen in Figure 2.3a. The presented puddle world has two solutions which are located close to each other. Still, the mean of both solutions leads through a puddle and, therefore, yields lower rewards.

In Figure 2.3b, we evaluate the performance of REPS and HiREPS with and without bounding the sub-policies' entropy. For HiREPS, just two sub-policies were used. REPS takes a longer



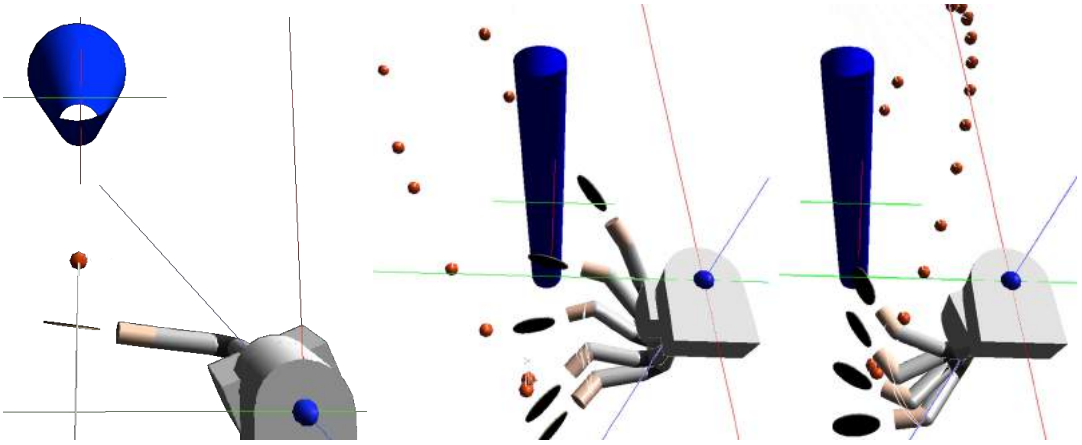
(a) Puddle World **(b)** Evaluation

Figure 2.3.: (a) The puddle world task. Sub-policies are represented as two-dimensional DMPs with fixed end points. The DMPs have five basis functions per dimension and we learn the weights of the basis functions for the y dimension while leaving the weights for the x dimension fixed. The plot shows trajectories sampled from two sub-policies found after 80 iterations (using 10 samples per iteration) of HiREPS. Start and end points of the trajectory are pre-defined, the agent may choose the path in between those points. The puddle world has two solutions. The plot shows samples from both modes acquired by HiREPS after 30 iterations. (b) Performance of HiREPS and REPS on the puddle world task. As HiREPS can represent both solutions, it does not get stuck averaging over both modes. Note, that this effect is much more pronounced if we bound the entropy of the sub-policies.

time to reliably find good solutions, as the algorithm averages over both modes. HiREPS without bounded entropy performs slightly better than REPS. However, the advantage of HiREPS is much more pronounced when also bounding the expected entropy of the responsibilities. Furthermore, if we limit the entropy, the algorithm is able to reliably find both modes. The results also show that even after 80 iterations (using ten samples per iteration), the REPS learning curve still exhibits some variance, showing that the algorithm has not fully converged.

Tetherball

Our aim is to adapt the game 'Tetherball' for a robotic player, as shown in Figure 2.4b. Specifically, the task consists of a ball, a rope, an obstacle (i.e., a pole) and a target. The ball is hung in front of the pole, in a line with the robot, the pole and the target. This setup requires the robot to induce a circular trajectory to be able to wind the ball around the pole. In the planar simulation, the robot can induce this angular velocity through the elasticity of the rope. In the physically accurate simulation as well as on the real robot, a pre-strike is necessary to achieve similar results. This task presents a versatile solution space, as many different strategies successfully hit the target. Our goal is to model this versatile solution space with HiREPS. In order to thoroughly assess the proposed method, we split our investigation of the tetherball task into three parts of ascending difficulty. In a first evaluation, we implement a planar simulation of the task in which we abstract the robot into a force, i.e., we do not simulate a robotic arm but instead allow the agent to directly exert a force onto the ball. We use this simplified setup to compare HiREPS to its non-hierarchical



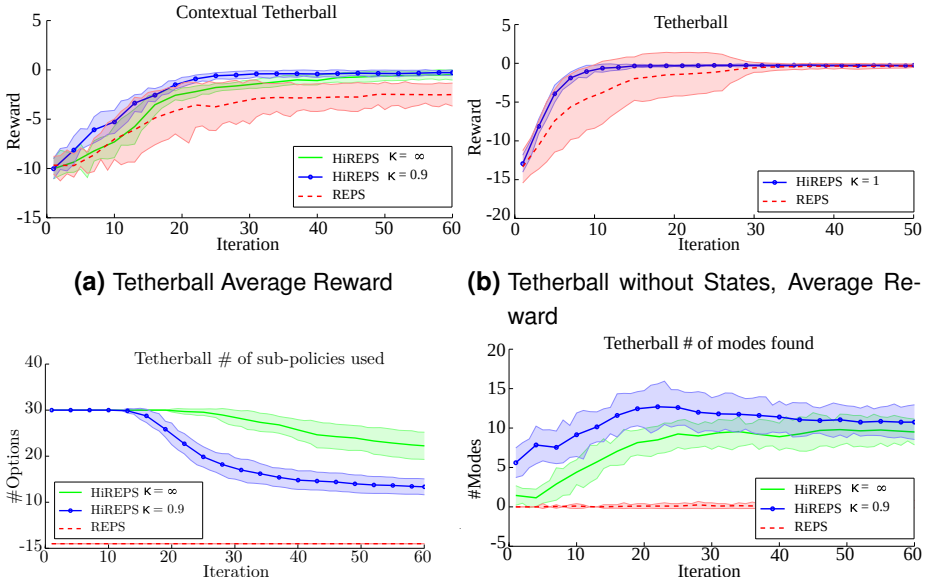
(a) Top View **(b)** Left and right swings

Figure 2.4.: (a) Top view of the setup of robot tetherball. The ball is hung on a string from the ceiling in front of the pole. (b) Time series overlay of two swings in simulation, one left swing and one right swing. The two solutions use different sub-policies of the same hierarchical policy. HiREPS can also keep more sub-policies representing additional solutions to a task.

counterpart as well as to investigate the effect of different settings. In a second evaluation, we use a detailed physical simulation of the task including the robot arm, in which we evaluate the benefits of a hierarchical policy in a more realistic setting. Finally, we present the results of the actual robotic task which show that HiREPS finds multiple solutions within one learning scenario.

Planar Simulation.

For the purpose of testing the HiREPS method, we use a strongly simplified setup where the tetherball task is implemented in a planar simulation. We initialize HiREPS with 30 randomly located sub-policies and use ten samples per iteration. The agent can accelerate the ball with a two dimensional impulse $\{F_x, F_y\}$. The reward is given by the negative minimum squared distance of the ball to the target throughout the ball's trajectory. The initial state of the agent is given by the initial position of the ball before hitting it. We only vary the x-position of the ball and learn different solutions to hit the target. As before, we compare HiREPS with and without bounding the overlap of the sub-policies to the standard REPS approach. In Figure 2.5a, we evaluate the average reward of all three approaches. The results show that HiREPS with the bound on the overlap outperforms the two other methods. The HiREPS approach already without the additional bound is better than the REPS approach, as REPS only uses one sub-policy to cover the whole state space. Thus, REPS needs to approximate the optimal policy using a single linear model. Therefore, in order to have a fair comparison to REPS, we also compare HiREPS and REPS on the tetherball task without states, i.e., we always start from the same initial state in the middle. This comparison can be found in Figure 2.5b. We can observe that REPS is impaired by the multi-modality of the solution space as it tries to average over several modes, but finally learns a solution that is equally good as the solutions found by HiREPS. However, while REPS only learns one solutions, Figure 2.5d shows that HiREPS learns multiple solutions at the same time. In order to compute the number of modes, we divide each dimension of the state action



(a) Tetherball Average Reward

(b) Tetherball without States, Average Reward

(c) Tetherball Number of Sub-Policies Used

(d) Tetherball Number of Modes Found

Figure 2.5.: (a) Average reward gathered by the REPS and the HiREPS with and without bounding the entropy of the sub-policies in the tetherball task including states. As REPS only uses a single sub-policy and thus a linear model as policy, it cannot represent the complicated structure of the solution. The HiREPS approach benefits from bounding the entropy of the sub-policies. (b) Average reward in the tetherball task without states. While REPS also finds one of the optimal solutions, HiREPS benefits from its structured policy representation and outperforms REPS in learning speed. At the same time, HiREPS finds both solutions. (c) Number of sub-policies used by the HiREPS approach with and without bounding the entropy. If the prior $p(o)$ of a sub-policy becomes too small it gets deleted. By bounding the overlap of the sub-policies, less sub-policies are used while the performance of the algorithm is increased. (d) Number of modes found by HiREPS with and without bounding the overlap of the sub-policies. We can see that, despite that HiREPS with bounding the overlap uses less sub-policies, it can find more modes. Without bounding the overlap of the sub-policies, many sub-policies concentrate on the same mode, which attenuates the advantages of the structured policy representation.

space into 5 partitions and count the number of partitions which contain at least one sub-policy with an average reward larger than -1. The plot shows that, as the sub-policies distribute more uniformly in the state-action space due to the bounding, we can find more modes. Thus, the bound of the overlap also helps us to find more versatile solutions as it avoids situations where multiple sub-policies concentrate on the same solution. In Figure 2.5c, we show the number of sub-policies used for different bounds of the overlap $\tilde{\kappa}$. By bounding the overlap, the gating network can learn to select individual sub-policies more decisively, explaining the faster learning speed in the experiments when using the bound.

Physically Accurate Simulation.

Having compared the properties of (Hi)REPS on the simpler simulation, we proceed to presenting the results of the physically accurate simulation as shown in Figure 2.4a, which also serves as a stepping stone to the real robot results. As described, the pole is placed on a line between

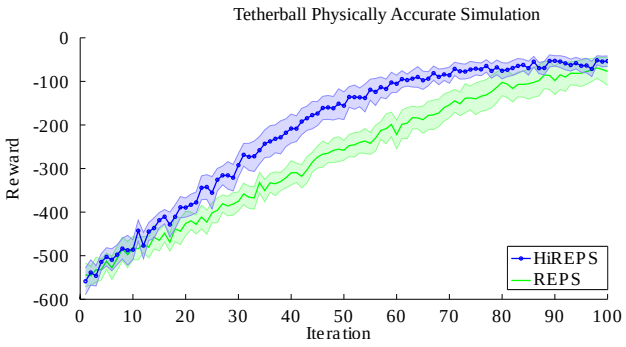


Figure 2.6.: Results of the physically accurate tetherball simulation. Both REPS and HiREPS reach the same asymptotic reward. However, since HiREPS can differentiate between the multiple local optima and assign different options to these optima, it exhibits faster learning speed.

the ball's resting position and the target location, such that it is impossible to hit the target with a single strike of the ball in direction of the target. In order to evoke a circular trajectory that arcs the ball around the ball, displacing the ball from its resting pose is necessary. Thus, we decompose our movement into a swing-in motion and a hitting motion. However, the parameters for both motions are combined into one parameter vector that is learned jointly. A more powerful approach would be to respect the natural separation of the successful trajectories and use the skill sequencing approach as presented in Section 2.6. The reward is determined by the speed of the ball when the ball winds around the pole. We define winding around the pole as the ball passing the pole on the opposite side of the pole. We run the algorithms with 50 samples per iteration and always keep the last 400 samples. We initialize our algorithm with 30 sub-policies and stop deleting sub-policies if only 5 sub-policies are left. The resulting learning curve in the simulation can is shown in Figure 2.6. After 100 iterations the robot has learned to wind the ball around the pole in 5/5 trials. In all trials, we were able to observe sub-policies for the left and for the right mode. The resulting movements are shown in Figure 2.4b and illustrate that the resulting movement of the two solutions are easily differentiated. The results show, that albeit HiREPS uses the same amount of total samples per iteration as REPS, it can use those samples to learn multiple solutions while being faster than REPS can learn a single solution.

Real Robot Tetherball.

After presenting the results on the two simulated tetherball settings, we proceed to show the results of the real robot experiment. For the robot experiment, we mounted a table-tennis paddle to the end-effector of the robot arm. The real-robot setup is depicted in Figure 2.8a and two successful hitting movements of the real robot are shown in Figure 2.7. In order to track the ball, a Kinect RGBD camera was setup to look at the robot from the opposite side of the pole. The vision data was used to compute the reward signal. As in the physically accurate simulation, the robot needed to perform a pre-swing as well as the actual swing. However, the real robot can easily be bootstrapped through imitation learning. Thus, we extract the shape parameters v by kinesthetic teach-in [Ijspeert et al., 2003] for both motions. Subsequently, the robot learns the final positions and velocities of all seven joints through the presented approach. Additionally, we learn the waiting time between both movements. This task setup results in a $2\times2\times7+1=29$-dimensional

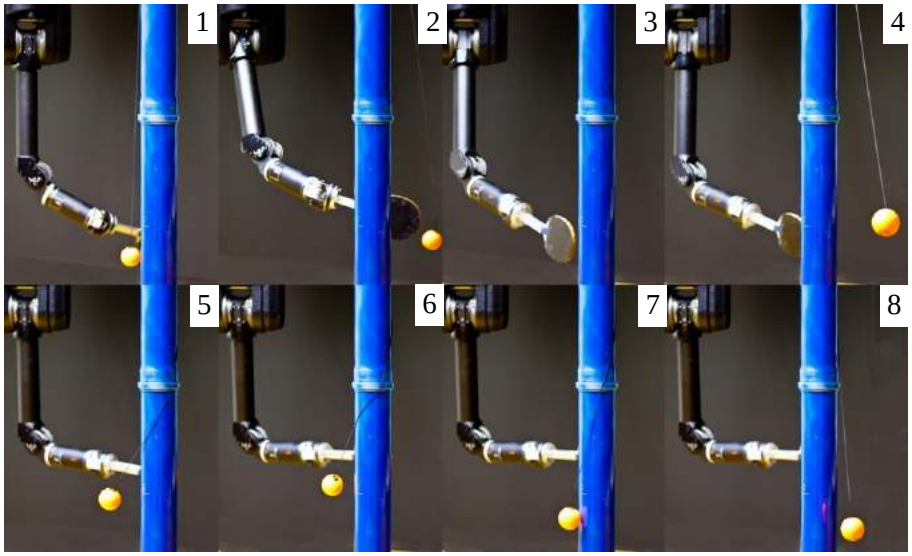


Figure 2.7.: Time series of a successful swing of the robot. The robot first has to swing the ball to the pole and can, once the ball has swung backwards, arc the ball around the pole.

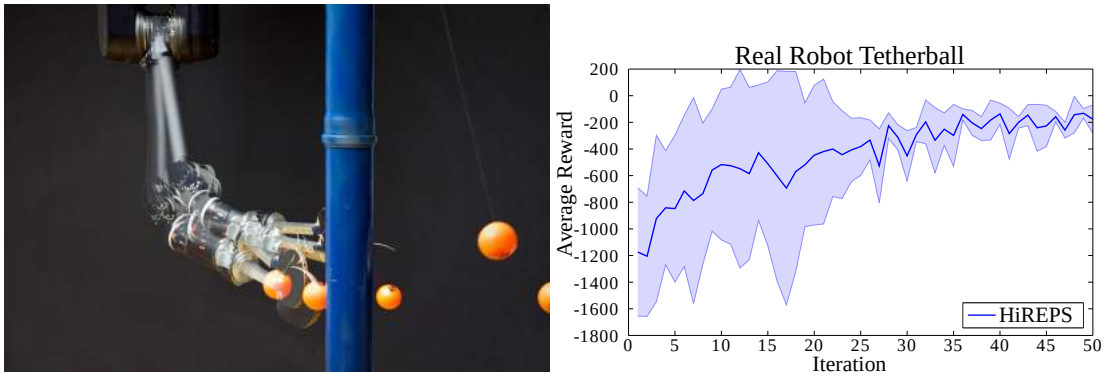
action space. We initialized the algorithm with 15 sub-policies and sampled 15 trajectories per iteration. While this scheme amounts to considerably fewer samples than in simulation, it is sufficient to learn multiple solutions, given the initial demonstrations. We performed three trials to produce the errorbars reported in the results. The learning curve is shown in Figure 2.8b. The noisy reward signal is mostly due to the vision system and partly also due to real world effects such as friction which lead to a non-repeatability of rollouts. Two resulting movements of the robot are shown in Figures 2.7 and 2.4b.

2.9.2 Planar Reaching Task

To further evaluate the properties of the proposed algorithm, we present a series of experiments on a planar reaching task. In this simulated task, the agent controls the joint trajectories of a three-link robotic arm using DMPs with two basis functions per joint. The robot starts from a fixed initial position and executes a trajectory for 100 time steps. At time steps 40 and 100 the robot is rewarded for passing through via points. While the robot is controlled in joint space, these via points are given in task space. Furthermore, for both time steps two possible via points exist.

Comparison to Baseline Methods.

Most of the presented experiments compare the performance of HiREPS to its natural competitor, REPS. To better analyze the performance levels of HiREPS, we also performed an experiment comparing HiREPS to other state of the art methods. Specifically, we compared to the PI^{BB} algorithm [Stulp and Sigaud, 2013], the CMA-ES algorithm [Hansen et al., 2003] as well as the NES algorithm [Wierstra et al., 2014]. To compare HiREPS against these base-lines we performed an empirical optimization of the open parameters these methods perform. For example, Fig. 2.9b shows the effects of the initial variance of CMA-ES. The results in Fig. 2.9a show that HiREPS converges faster than the alternative algorithms we compared to. Furthermore, the alternative al-



(a) Tetherball **(b)** Real robot results

Figure 2.8.: (a) The real robot tetherball setup. (b) Average rewards of HiREPS on the real robot Tetherball setup. Mean and standard deviation of three trials are shown. In all three trials, the robot has found solutions to wind the ball around the pole on either side after 50 iterations.

gorithms are designed to represent a single solution. In this experiment, all algorithms used 10 samples per iteration. Both HiREPS and PI^{BB} used a higher initial variance than CMA-ES and NES in this experiment since the latter methods' performance decreases using higher initial variances as shown in Fig. 2.9b. Due to this higher initial variance, PI^{BB} and HiREPS start with a higher initial reward.

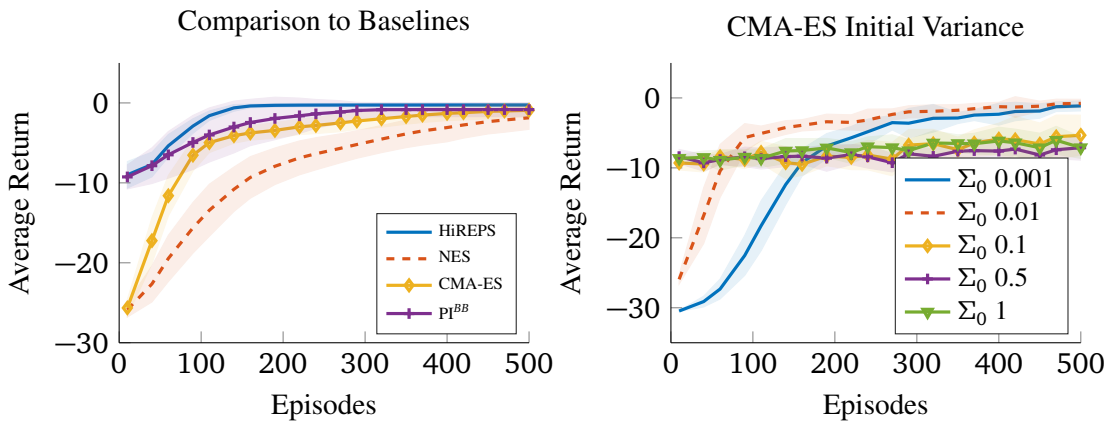
Initialization of the Algorithm.

Since the presented algorithm is based on the availability of multiple sub-policies, these options have to be initialized to sensible values. In the presented experiments linear Gaussians were usually used as sub-policies, i.e., $\pi(\boldsymbol{a}|\boldsymbol{s}, o) = \mathcal{N}(\boldsymbol{a}|\boldsymbol{\mu}_o + \boldsymbol{F}_o\boldsymbol{s}, \Sigma_o)$. To initialize each linear Gaussian, we can keep $\boldsymbol{F}_o = \boldsymbol{0}$ and $\Sigma_o = c\boldsymbol{I}$, where the constant c is a task specific variable set by the experimenter. While F_o and Σ_o are initially identical for all sub-policies, the means $\boldsymbol{\mu}_o$ have to be different to allow for later separation of the sub-policies based on the responsibilities. To that effect, we usually sampled the individual sub-policy means $\boldsymbol{\mu}_o$ from a normal distribution, i.e., $\boldsymbol{\mu}_o \sim \mathcal{N}(\cdot|\boldsymbol{0}, \Sigma_\mu)$.

Figure 2.10a shows the results of varying Σ_μ relative to the admissible range of parameters. The results show that while a larger initial separation seems to help bootstrapping the learning process, the overall effect of changing this parameter is negligible. In this experiment, a total of 20 options were available to the algorithm. While the comparison to the base-line methods was performed using ten samples per iteration, for the following experiments 20 samples per iteration were used. Using more samples per iteration allowed us to observe the effects under investigation more clearly, while the comparison to other baselines was optimized for the CMA-ES and NES algorithms.

The Influence of Probabilistic Option Assignments.

The presented algorithm allows for inter-option learning, i.e., the sharing of information between options during learning which is expected to improve learning. To test this hypothesis, we performed an experiment comparing the proposed formulation of the algorithm to an alter-



(a) Comparison To Baselines

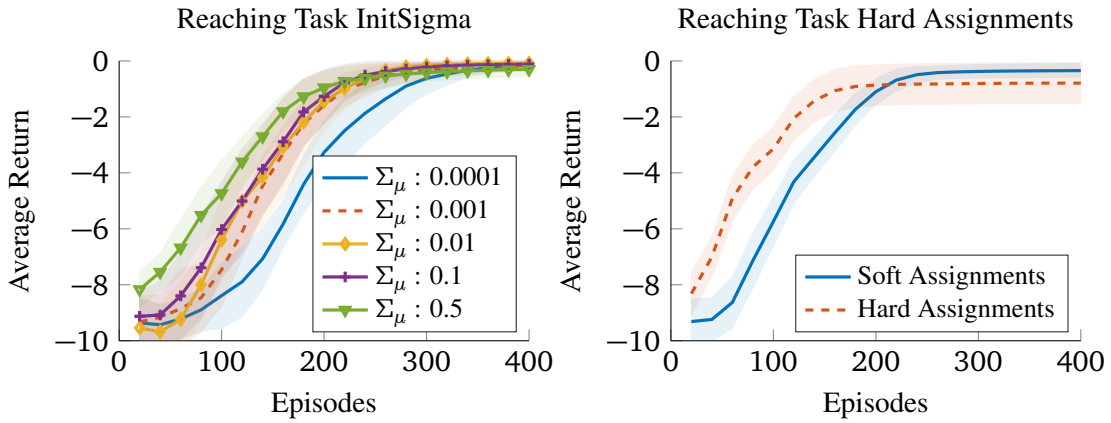
(b) Influence of Initial Variance in CMA-ES

Figure 2.9.: (a) Comparison of HiREPS to popular baselines. HiREPS and PI^{BB} start at higher initial rewards because CMA-ES as well as NES have to be initiated with a much lower variance. While HiREPS and Pi^{BB} start with a high-variance initial policy (or sub-policies) and keep contracting it until convergence. CMA-ES and NES on the other hand work best with a smaller initial variance that is kept for longer. (b) Effects of different initial variances on CMA-ES. We chose the best initial variance for the comparative experiments.

native formulation based on hard assignments. In the alternative formulation, every option was effectively limited to using it's own samples to update its policy. This behavior was achieved by assigning a responsibility of 1 to the option that generated a sample while all other options had zero responsibility for the same sample. Using soft assignments allows the gating to shift responsibilities such that options can specialize on distinct sub-tasks. The results in Figure 2.10b show that using soft assignments did indeed improve asymptotic learning performance but was slower in the early stages. However, in our experiments we observed that the importance of this effect would diminish with an increasing number of options. As the number of options increases, HiREPS can implement a more effective divide and conquer strategy. Sharing of information can becomes less important after the division of the state-action space has been successfully performed.

Evaluation of the Entropy Bound.

One of the main parameters to be considered in the proposed algorithm is the desired bound on the entropy of the responsibilities, $\tilde{\kappa}$. The lower the value for $\tilde{\kappa}$, the more aggressively the algorithm will separate the individual options. Figure 2.11a shows the results of evaluating a wide range of possible parameter values for $\tilde{\kappa}$ in the reaching task. The results show that values for $\tilde{\kappa} \geq 1$ yield slower convergence speeds while the options still overlap and aim to explain multiple solutions. As in most other experiments presented in this chapter, a value of $\tilde{\kappa} = 0.9$ seems to yield the best learning speeds. While even lower values of $\tilde{\kappa}$ do not noticeably decrease the learning speed in the presented experiment, our experience shows that lower values may prevent successfully learning multiple solutions. Since lower values of $\tilde{\kappa}$ force options apart, only few options may actually be fully developed while the probability of sampling from most other options will diminish.



(a) Effect of Different Initializations

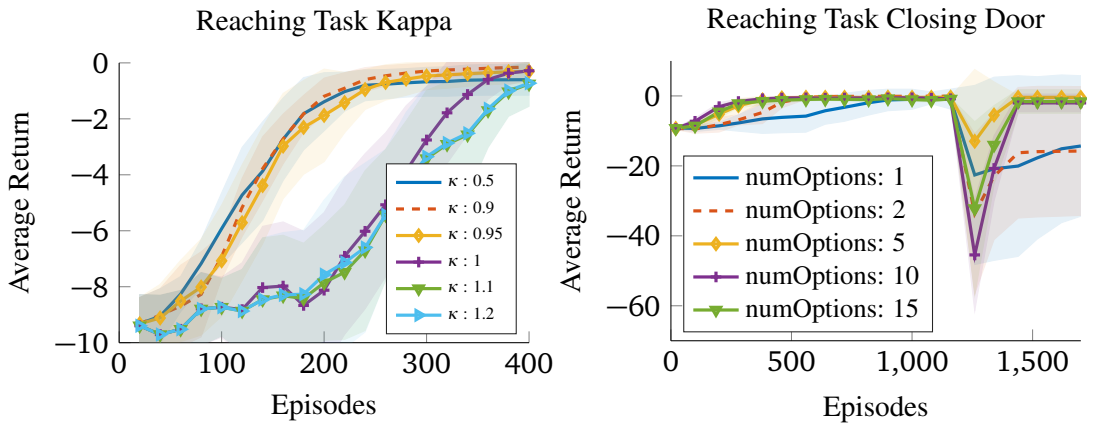
(b) Effect of Probabilistic Responsibilities

Figure 2.10.: (a) Investigation of different initial distributions of sub-policies. Using a low initial variance of the distribution over option means, all options share similar responsibilities for all samples and the entropy bound forces a separation of the options. Using a higher initial variance for this distribution, the individual sub-policies will be responsible for different regions of the action space from the beginning. While the results show that such an initial separation can improve learning speed initially, it can also lead to sub-optimal asymptotic performance. (b) Using hard assignments, i.e., not sharing information between options, leads to high initial learning speed. However, using the hard assignments, the algorithm did not always find the optimal solution. In our experiments, the non-probabilistic (hard assignments) version of the algorithm would usually require more samples per iteration to consistently find equally good solutions as the proposed probabilistic approach. However, our experience shows that when using more samples per iteration, the learning speed of the hard assignment approach will actually decrease relative to the speed of the probabilistic approach. This effect can be explained by the fact that each option is drawn to all local optima more uniformly if more samples are used.

The results reported for this experiment and the puddle world experiment regarding the entropy bound mirror our experience with the remaining experiments reported in this chapter. In all experiments a bound of $\kappa = 0.9$ seems to give good results and much higher (> 0.95) or much lower (< 0.8) values usually deteriorate performance. Thus, for the remaining experiments, we chose $\kappa = 0.9$ and do not present further evaluations.

Robustness to Changing Environments.

One main contribution of the proposed algorithm is to learn multiple solutions for the same task. Learning multiple solutions can be interesting if, for example, the environment changes in a way that makes some solutions inaccessible. To evaluate the behavior of the proposed algorithm in such a scenario, we let the algorithm learn a solution for the reaching task as described above. After 30 iterations HiREPS typically converged to a good solution. At that point, the via points on either the lower or upper path were randomly disabled, which simulated blocking one of the paths. Figure 2.11b show the effects of learning multiple solutions. With only one option, the agent cannot recover in about half the trials. Using more options increases the likelihood that the agent has learned sufficiently versatile solutions to successfully perform the task even after cutting off of the paths.



(a) Influence of Entropy Bound

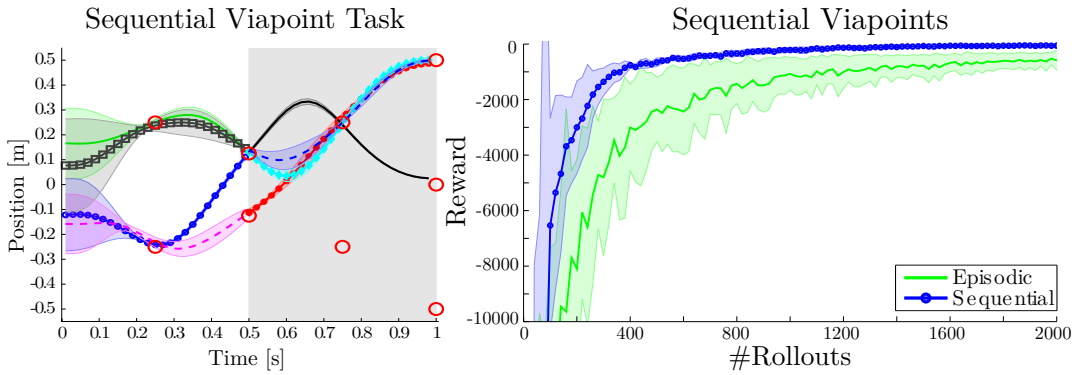
(b) Changing Environments

Figure 2.11.: (a) Comparison of different relative values κ for the entropy bound. The results show that higher values for the entropy bound slow down learning, as the options are not forced to separate. Thus, multiple options will compete for the same local optimum for longer. Very low values for κ will also slow down learning and can potentially lead to sub-optimal asymptotic performance when the entropy bound 'overpowers' the reward maximization criterion. (b) Investigation of a task which requires multi-modal solutions. After learning a solution, some of the via points were randomly disabled and the agent had to rely on the availability of alternative solutions. The results show that increasing the number of options allowed for a robust policy which could recover from the change in the environment because the agent had previously learned multiple solutions.

2.9.3 Sequencing of Skills

As evident from the tetherball experiment, many real world tasks require multiple steps to be solved. To evaluate the skill sequencing implementation of HiREPS presented in Section 2.6, we present a second set of experiments. Before testing skill sequencing on a real robot task, we evaluated the time-indexed HiREPS algorithm on a via-point task in order to illustrate the properties of the approach. In this task, we modeled a second-order dynamical system. The state of the agent is given by its position x and velocity $\dot{x}$. The actions u control the accelerations $\ddot{x}$. The task of the agent is to reach specified via-points at four different points in time. For each of these time points, different via-points exist. The reward at the time points is given by the negative squared distance to the closest via-point. In addition to the deviation to the via-points at the four specified time points, the reward function contains a squared punishment term for taking high accelerations. As we defined multiple via-points for each of the four time-points, this task has multiple solutions per construction. The agent used 20 samples per iteration for this task. The exact setting of the task including its via-points is depicted in Figure 2.12a.

In order to demonstrate sequencing of skills, we decomposed the task into two DMPs which were executed sequentially. We used five shape parameters for both DMPs. In addition, we also learn the goal-parameter of the DMP, resulting in 6 parameters per movement primitive. To compare our sequencing method to the commonly used episodic policy search setup, we also solve this task with the episodic version of HiREPS. In this case, we only used one DMP with ten shape



(a) Task Setup. Colors indicate active option.

(b) Average Reward

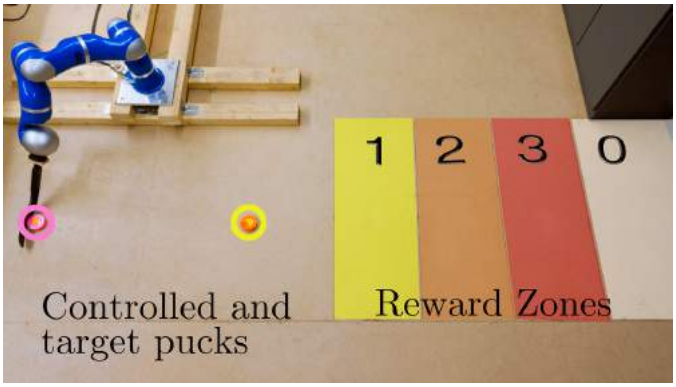
Figure 2.12.: (a) We use this via-point task to illustrate our algorithm.The agent has to reach one of the via-points (denoted by red circles) at each of four specified times $[0.25, 0.5, 0.75, 1.0]$s. The reward is given by the negative squared distance to the closest via-point. The last via-point has to be reached with zero velocity. The initial positions and velocities are sampled from a Gaussian distribution with zero mean and a standard deviation of 0.25 for the position and 0.1 for the velocity. In this task, we learn to sequence two motor primitives, with the second primitive starting at $t = 0.5$s (shaded region in which the line colors change). This task is per construction multi-modal and illustrates how our algorithm learns distinct motor primitives. The mean and variance are indicated by shaded error bars. The agent learned several but not all possible solutions to solve the task. (b) The multi-modal via-point task learned with episodic and sequential motor primitive learning where the movement was decomposed into two primitives, see Figure 2.12a for a more detailed description. As we can see, our algorithm could exploit this decomposition resulting in increased learning speed and higher quality final policies.

parameters and the additional goal-parameter. For both scenarios, the agent could choose between four distinct sub-policies o_i at each decision time-point. As we can see from Figure 2.12a, the agent learned to select these primitives according to the state at the decision time-points as well as to adapt the primitives such the task can be solved. Our approach was able to learn multiple solutions for the task as can be seen from Figure 2.12a. However, only a subset of all possible solutions was found.

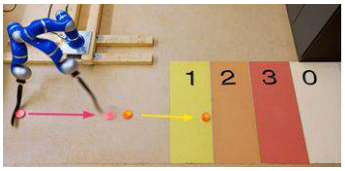
The comparison of the episodic and the sequential learning methods can be seen in Figure 2.12b and shows the advantage of the sequencing approach in learning speed as well as in the quality of the learned solution. Additionally, the episodic formulation will also require more options in total in order to find all possible solutions than the sequential formulation. However, in the presented experiment we only rewarded the agent for finding at least one solution such that this effect did not alter the resulting performance analysis.

Evaluation on the Robot Hockey Task

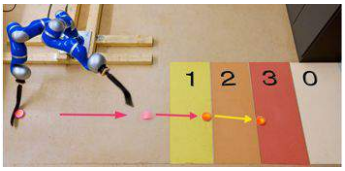
As before, after using a simpler task to investigate the properties of the proposed algorithm, we now present the results of a more sophisticated task. Similar to the structure of the tetherball experiments, we first present a comparative analysis on a physically accurate simulation and then



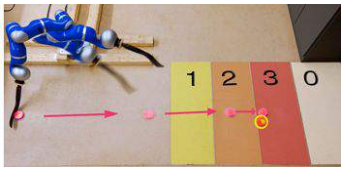
(a) Hockey Setup



(b) First strike



(c) Second strike



(d) Final strike

Figure 2.13.: (a) The robot hockey task. The robot has two pucks, the pink control puck and the yellow target puck. The task is to shoot the yellow target puck into one of the colored reward zones. Since the best reward zone is too far away from the robot to be reached with only one shot, each episode consists of three strikes. After each strike the control puck is returned to the robot, but the target puck is only reset after one episode is concluded. Concluding an episode with the target puck in one of the reward zones yields rewards from one to three as indicated in the picture. However, if the robot shoots the target puck too far, the reward is zero. (b-d) One episode of the Hockey task, consisting of three strikes. Each picture shows the initial and final position of control and target puck. The movement of the pucks is indicated by arrows. The robot can shoot the pink control puck to move the target puck and tries to place the yellow target puck in one of the marked target zones while not overshooting. In the depicted episode the robot only needed two strikes to place the target puck into the highest reward zone. With the last strike the robot taps the target puck only slightly without actually moving it to avoid negative reward for missing it.

proceed to show the results of the real robot task. In the robot hockey task, the robot has to move a target puck into one of three target areas. This target puck can only be moved by shooting a control puck at it. The target areas are defined by a specified distance to the robot. The first zone is defined as distance from 1.4 to 1.8m, the second zone from 1.8 to 2.2m and the last zone from 2.2 to 2.6m. The robot gets rewards of 1, 2, and 3 for reaching zone 1, 2 or 3, respectively, with the target puck. If the robot overshoots the last target zone, the reward drops to zero. The reward is only given after each episode which consists of three shots of the control puck. After each shot, the control puck is returned to the robot. The target puck, however, is only reset after each episode. The setup of the robot hockey task is shown in Figure 2.13a.

The 2-dimensional position of the target puck defines the state of the environment as perceived by the agent. After performing one shot, the agent observes the new position of the target puck to plan the subsequent shot. In order to give the agent an incentive to shoot at the target puck, we punished the agent with the negative minimum distance of the control puck to the target puck

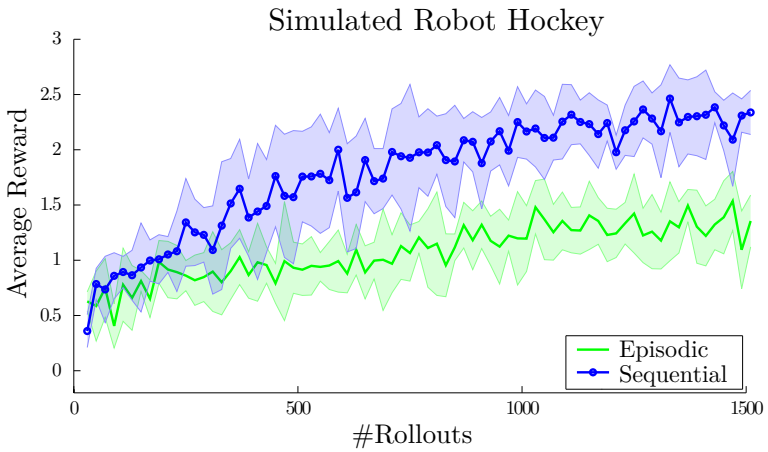


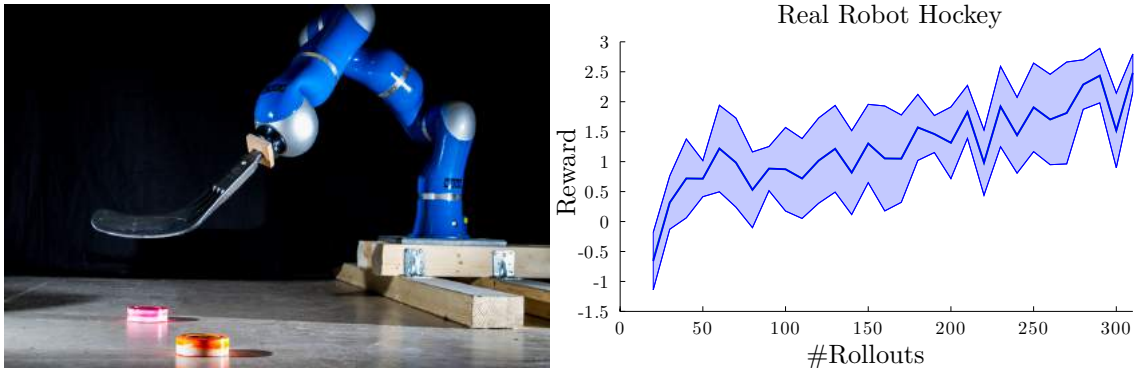
Figure 2.14.: Comparison of sequential motor primitive learning to the episodic learning setup on the simulated robot hockey task. The sequential motor primitive learning framework was able to find a good strategy to place the puck in the third reward zone in most of the cases while the episodic learning scenario failed to learn such a strategy.

after each shot. While this reward was given after every step, the zone reward was only given at the end of the episode (every third step) as $r(\boldsymbol{s}_{K+1})$.

We used a DLR-Kuka lightweight arm with 7 degrees of freedom as depicted in Figure 2.15a. We used DMPs to represent single motor primitives where we only adapted the goal positions and velocities of the primitives. This setup resulted in 14 parameters per primitive per shot. Thus, the episodic version of HiREPS has to optimize one 42-dimensional parameter vector while the time indexed version of HiREPS that we use for skill sequencing has to optimize three policies with 14-dimensional parameter spaces each. Both, the episodic as well as the time-indexed version of HiREPS had access to five options. The shape parameters v of the single primitives were learned from imitation by collecting trajectory data via kinesthetic teach-in.

Simulation Results.

We first implemented a realistic simulation of the robot hockey task. In simulation, we varied the initial position of the puck by sampling the position from a normal distribution with standard deviation of 10cm. The agent used 30 samples per iteration. We compared our sequential motor primitive learning method with its episodic variant. For the episodic variant, we encoded the policy-parameters of all three shots into one policy, resulting in 42 parameters. The episodic variant cannot use state-feedback except for the information of the initial position of the puck. To make the comparison as fair as possible we did not use any noise in our simulation and, hence, the initial position of the puck is sufficient to solve the task. The comparison of both methods can be seen in Figure 2.14. The episodic learning setup failed to learn a proper policy while our sequential motor primitive learning framework could steadily increase the average reward. Our method reached an average reward of 2.3 after learning for 1500 episodes. Note that an optimal strategy would have reached a reward value of 3, however, this is still a clear improvement in comparison to the episodic setup, which reached a final reward value of 1.4.



(a) Hockey Setup **(b)** Real Robot Hockey Rewards

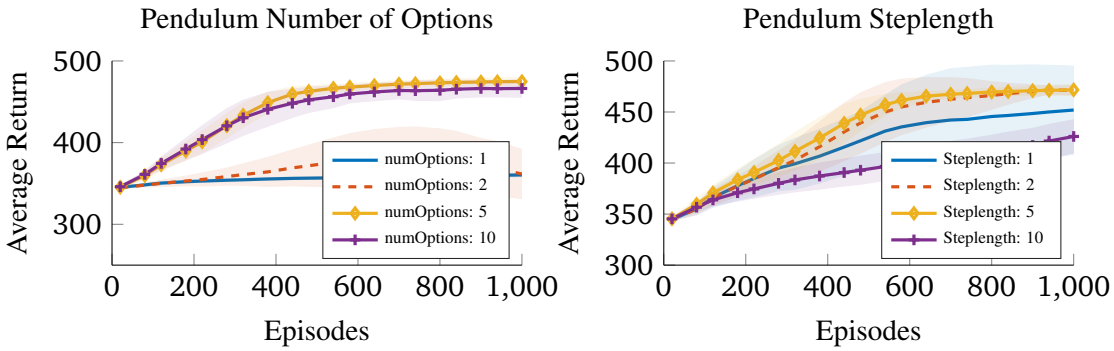
Figure 2.15.: (a) The real robot Hockey setup. The robot first has to shoot the pink puck on the orange puck, to move the orange puck into a target zone. (b) One trial of of the real robot hockey tasks. The robot starts with a negative initial reward and learns to achieve an average reward of 2.5 after 300 episodes. The optimal theoretical reward of the presented task is 3.0. However, learning has not yet converged and had to be stopped prematurely due to time constraints.

Real Robot Results

We used a Kinect RGB-D camera to observe the state of target puck. For the real robot hockey task, the initial position was not varied. On the real robot, we could reproduce the simulation results. The robot learned a strategy which could move the target puck to the highest reward zone in most of the cases after 300 episodes, where the robot used ten samples per iteration. One episode of robot hockey is depicted in Figure 2.13. In the final trials, the robot tended to prefer using a soft hit in the first shot and to shoot the target puck to the last reward zone with the remaining two shots. This behavior yielded higher average reward, since it is easier to only tap the target puck without moving it too much while it is still closer to the robot. During learning the robot steadily adapted his strategy when it mastered the necessary motor skills to achieve higher rewards by placing the target puck in the highest reward zones.

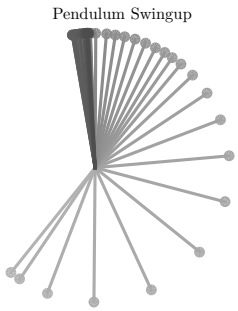
2.9.4 Infinite Horizon Formulation for Sequencing Skills

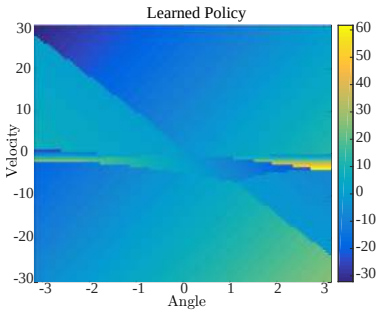
We evaluated the infinite horizon formulation of HiREPS on the pendulum swing-up task in simulation. In this task, the pendulum starts hanging down with a random perturbation. The goal of the robot is to find a solution that first swings up the pendulum and then stabilizes the pendulum at the top. Instead of directly choosing motor commands in each time step, the robot chooses a desired joint value which is tracked with a PD-controller that is active over multiple time steps d (in the standard setting $d = 5$). While using direct motor commands is generally feasible as well, using a PD-control scheme is often beneficial from a robotics point of view. Real systems are often controlled at very high frequencies internally, however, policy signals are usually only necessary at lower frequencies. Reducing the control frequency of the policy will increase the signal to noise ratio, especially in real systems. Furthermore, even in simulated systems, the temporal extension of the actions can benefit the learning process. Since this task is highly non-linear in nature, it cannot be solved with a single linear policy. However, since HiREPS is able to repre-



(a) Pendulum #Sub-Policies

(b) Pendulum #Steps per Decision



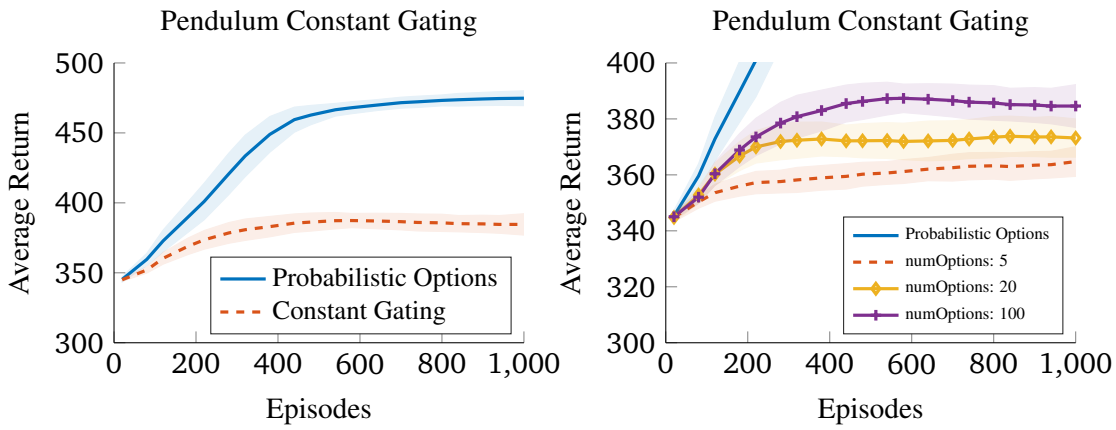


(c) Task Setup

(d) Policy

Figure 2.16.: Evaluations of the pendulum swingup task. The task starts with the pendulum hanging down and the robot has to swing up and stabilize the pendulum. (a) The results show that one linear sub-policy is insufficient to learn this task. With five sub-policies the robot is able to learn the task, with ten sub-policies the asymptotic performance does not improve anymore. (b) The number of time steps d each sub-policy stays active. The results show that activating a sub-policy for multiple time steps increases the speed of learning. If a sub-policy stays active for more than five time steps, the performance decreases. (c) One rollout of the pendulum swingup task. The pendulum starts at the bottom and requires a pre-swing to be brought to the upright position. (d) Pendulum Swingup-Policy learned by HiREPS. Colors indicate the mean value of the policy.

sent a piecewise linear policy, it can be used to solve the pendulum swing-up task with multiple linear sub-policies. In this task, the pendulum has a mass of 10kg, a length of 0.5m and a friction coefficient of 0.2. The robot can exert at most 30nm of torque and uses 20 samples per learning iteration. The internal robot control runs at 100Hz and the restart probability in the base setting is given as $(1-\gamma) = 0.02/d$, where d determines the control frequency of the learned policy. The primitive actions wich are chosen by the sub-policies are the desired joint angles of the pendulum. The robot can typically perform a swing-up within 1.5s. To represent the value function, we used the Fourier transform based feature transformation of the states using five Fourier bases, see Section 2.8. The gating uses a squared expansion of the states as feature representation. The reward function punishes deviation from the desired upright position with a factor of 500 and punishes velocities with a factor of 10. These punishments are subtracted from a base value of 500. The results of the experiments show that a single linear policy is insufficient for swinging up and stabilizing, however, HiREPS successfully combines multiple linear policies to solve the task, a resulting policy is shown in Fig 2.16.



(a) Comparison to Constant Gating

(b) Number of Options in Constant Gating

Figure 2.17.: (a) Evaluation of a pre-defined constant gating vs. a learned gating. In the constant gating approach, the gating was pre-determined by performing a K-Means clustering of the state space. The results show that the pre-defined constant gating does not allow the agent to successfully learn the task. (b) Effect of the number of options when using a constant gating. The results show that the constant gating depends strongly on the number of options used, which related directly to the resolution with which the state-action space is parceled. Even with 100 options the constant gating approach is outperformed by the learned approach which uses only five options.

Evaluation of a Constant Gating.

In the pendulum swing-up task, a good gating policy is essential if the task is to be solved using simple sub-policies. To evaluate the effect of a learned gating versus a constant pre-determined gating, we compared the proposed approach to an alternative formulation where the individual action policies can be learned, but the gating stays constant. To find a such a constant gating, we divided the state space using K-Means clustering and kept the resulting gating constant afterwards. To perform K-Means clustering we sampled 1000 data points uniformly within $\pm[\pi, 50]$, where π in this case is the numerical value. The results show that the performance of such a constant gating strongly depended on the number of options available. However, even with 100 options the constant gating did not allow to learn this task successfully.

Comparison to a Non-linear Flat Policy.

While the proposed approach can learn the pendulum swing-up task using a combination of linear sub-policies, the task can also be solved using a single non-linear policy. To evaluate the effects of the proposed approach, we compared to a non-linear flat policy which was based on the same features as the gating in HiREPS. The results in Figure 2.18 show that while the non-linear flat policy initially learned faster than the HiREPS, the average asymptotic performance of the HiREPS was higher. Initially, HiREPS requires some iterations to learn a good gating policy. However, once a good gating policy was available, having different options that specialize on sub-tasks such as high accelerations vs. stabilization yielded better policies.

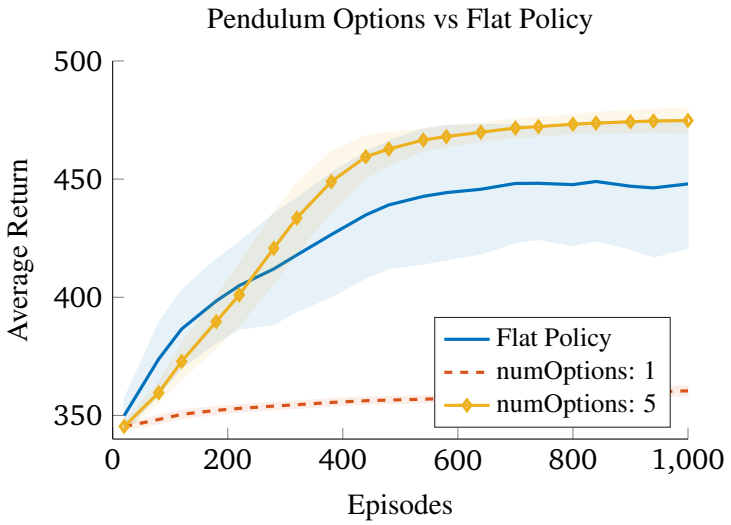


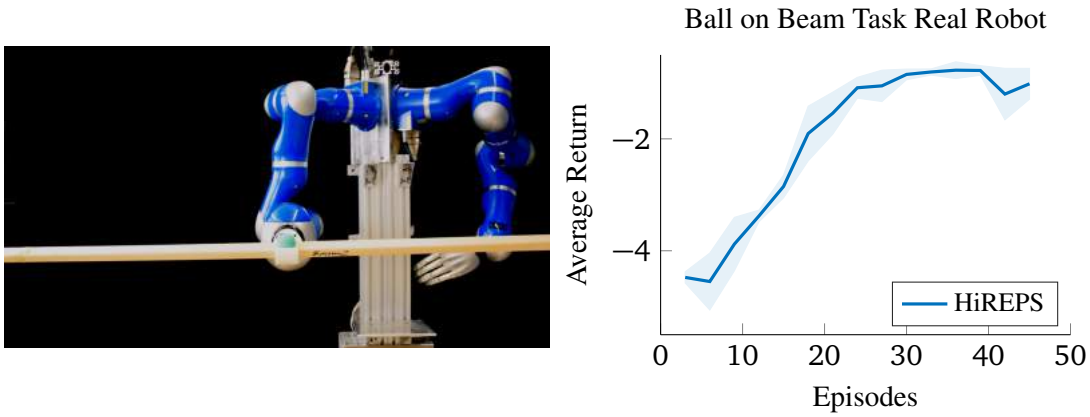
Figure 2.18.: Comparison to flat but non-linear policy. The non-linear policy is given by a linear Gaussian based on a non-linear feature transformation of the states while the hierarchical policy uses only linear sub-policies. The results show that the non-linear flat policy is outperformed by the hierarchical policy. While the flat policy is able to learn the solution, the asymptotic performance varies over multiple runs and would require more samples per iteration to exhibit a robust learning performance.

2.9.5 Infinite Horizon Real Robot Experiment

To evaluate the practical applicability of the proposed infinite horizon approach, we performed a real robot evaluation on the 'ball on a beam' task as shown in Fig. 2.19a. In this task, the robot has to guide a ball to a prescribed position on a beam and balance it at this position. In our experiments, the robot had to choose between two possible goal positions, 10cm to the left or right of the beam center. To learn the task, the robot learned a policy which set a new desired end effector angle every 0.5s. Every policy was evaluated for 20s and in each iteration the agent collected three episodes before updating the policy. In this task, the state space was given by the ball's position and velocity as well as the current beam angle. The sub-policies were linear Gaussian policies. The gating operated on squared features of the state space and the value function was based on a Fourier feature transformation of the state space using five basis functions. The reward function in this task was given as the summed negative squared distance and velocity of the ball (to the closest goal position) in every control step. The ball's position and velocity were determined using a Microsoft Kinect which was mounted above the robot. The results in Fig. 2.19b show that the robot was able to learn this task within 30 episodes in all three trials performed.

2.10 Conclusion & Future Work

In this chapter, we present a novel method, derived from first principles, to represent multiple solutions to a task. The representation of multiple versatile solutions is achieved though a hierarchical policy, which consists of a gating network and multiple sub-policies. We show that a naive implementation of a hierarchical policy is insufficient as it does not find distinct solutions.



(a) Ball On Beam Setup

(b) Ball On Beam Resyults

Figure 2.19.: (a) The experimental setup for the 'ball on a beam' task. A beam of 1m length is attached as the robot's end effector. The robot has to balance a ball in one of two goal positions, ±10cm from the center of the beam. The ball is initialized alternating on the right and left end of the beam. (b) Results of three trials on the real robot. In all trials the robot was able to learn the task within 30 episodes. In the second trial performance dropped slightly at the end of the trial before recovering again. This drop was most likely due to noise in the vision system.

To address this problem, we introduce an entropy-based constraint which ensures that the agent finds distinct solutions with different sub-policies. Having a hierarchical policy based on multiple sub-policies additionally allows us to solve tasks that are non-linear using multiple linear sub-policies.

We evaluated the proposed method on two real robot tasks and several simulated tasks. The evaluations showed that our framework can be used to learn temporally extended sub-policies, also called options, and to sequence these sub-policies to learn complicated tasks on real systems.

The results show, that HiREPS is able to learn multiple solutions for complicated tasks and that HiREPS is able to learn piecewise linear solutions for non-linear tasks. The comparison to the non-hierarchical REPS method shows that HiREPS additionally often learns faster than REPS. This effect is especially intriguing as HiREPS does not only aim to learn one solution but multiple solutions at the same time and effectively uses fewer samples per sub-policy.

While the presented approach is able to learn a multi-modal policy using weights generated by solving one optimization problem, alternative solutions are possible. In the presented approach, computing the lower bound and, subsequently, solving the optimization problem and fitting the sub-policies is equivalent to an EM approach with only one iteration. While EM approaches typically benefit from multiple iterations, we observed no further benefit from applying multiple passes in our experiments. The observation that one EM iteration is sufficient can be explained by the fact that we start with a distribution which is highly similar to the target distribution. Thus, using HiREPS with $\kappa \gg 1$ results in a version of the REPS algorithm which works on a mixture model. Furthermore, alternative mixture model fitting approaches could be used to incorporate the entropy bound which is integrated into the optimization problem in the proposed approach. For

example, Graça et al. [2007, 2009] show how EM can be performed with additional constraints such as, for example, the entropy constraint presented in this chapter.

In this chapter we focus on learning multiple sub-policies and do not solve the temporal extension aspect of the options framework directly. Temporal extension is inherently achieved in many of the presented experiments through the use of movement primitives. Using movement primitives, the time horizon is usually pre-determined. However, this time horizon could be made adaptive by including the duration of the movement primitive into the set of parameters that is learned by the agent. Learning both when to terminate options as well as learning how to construct options from atomic state-action pairs in a unified framework is an important aspect for future work. Equally, extending the framework to not only allow pruning of sub-policies, but also generating new sub-policies during the learning process could be an important addition to the versatility of the presented framework. In the presented version, options might be separated in the state-action space in the beginning of learning when actually multi-modalities exist on a smaller scale than can initially be detected. In such cases, it would be helpful to split one sub-policy into two sub-policies that can adapt to the individual modes.

3 Learning with Temporal Correlations

Tasks that require many sequential decisions or complex solutions are hard to solve using conventional reinforcement learning algorithms. In the previous chapter, we took advantage of parametrized movement generators, such as DMPs to abstract from this difficulty. In this chapter, based on the semi Markov decision process setting (SMDP) and the option framework, we propose a model which aims to alleviate these concerns. As before, instead of learning a single monolithic policy, the agent learns a set of simpler sub-policies. However, now we also aim to learn the initiation and termination probabilities for each of those sub-policies. While existing option learning algorithms frequently require manual specification of components such as the sub-policies, we present a first step towards an algorithm which performs a data-driven segmentation of continuous state-action spaces as well as learning the individual sub-policies from data. Furthermore, the proposed approach works well in combination with current policy search methods, which are particularly well suited for continuous real-world tasks. We present results on SMDPs with discrete as well as continuous state-action spaces. The results show that the presented algorithm can combine simple sub-policies to solve complex tasks and can improve learning performance on simpler tasks.

3.1 Introduction

Solving tasks which require long decision sequences or complex policies is an important challenge in reinforcement learning (RL). The semi Markov decision process (SMDP) setting [Sutton et al., 1999b] is a promising approach to simplify the complexity of such tasks. The SMDP setting extends the MDP setting by introducing macro-actions, which are carried out over multiple time steps [Parr and Russell, 1998, Sutton et al., 1999b]. Using these macro-actions, the agent has to make less decisions to solve a task. Furthermore, even if macro-actions are based on simple policies, the combination of multiple macro-actions can represent more complex solutions than the simple policies would allow for on their own. Such a decomposition of complex solutions can simplify the learning problem in many domains.

The option framework [Sutton et al., 1998] incorporates such macro actions. An option consists of a sub-policy, an activation policy and a termination probability. After an option is activated, actions are generated by the sub-policy until the option is terminated and a new option is activated. While the option-framework has received considerable attention [Parr and Russell, 1998, Sutton et al., 1999b, Dietterich, 2000], to date most algorithms either require the manual specification of the activation policies or sub-policies. Algorithms for autonomous option discovery typically depend on discrete state-action spaces [McGovern and Barto, 2001, Mann and Mannor, 2014, Simsek et al., 2005] with exceptions such as the work of Konidaris et al. [Konidaris and Barto, 2009]. Furthermore, many existing algorithms first explore a given MDP and learn suitable options afterwards [Menache et al., 2002, Simsek and Barto, 2008]. Hence, they are not aimed at leveraging the efficiency of options in the initial stages of learning but rather aim at transferring the options to new tasks. These approaches are powerful in scenarios where options can be transferred to similar tasks in the future. In contrast, the approach suggested in this chapter

aims at directly learning suitable options for the problem at hand while also being applicable in continuous state-action spaces.

In continuous state-action spaces, policy search (PS) methods which optimize parametrized policies have been shown to learn efficiently in simulated and real world tasks [Ng and Coates, 1998, Kober and Peters, 2010]. Thus, the compatibility of the proposed option discovery framework with PS methods such as PoWER [Kober and Peters, 2010] and REPS [Peters et al., 2010] is an important goal of this chapter. In the discrete setting, the framework can equally be combined with a wide range of methods such as as Q-Learning [Watkins and Dayan, 1992] and LSPI [Lagoudakis and Parr, 2003].

Furthermore, many complex tasks can be solved through combinations of simple behavior patterns. The proposed framework can combine multiple simple sub-policies to achieve complex behavior when a single one of these sub-policies would be insufficient to solve the task. These simpler sub-policies are easier to learn and help improving the overall learning speed.

To achieve a generalizing framework which works in discrete and continuous settings and requires minimal prior knowledge, we propose a probabilistic formulation of the option framework where all components are represented by distributions. To learn these options from data, we propose a probabilistic inference algorithm based on the expectation maximization algorithm [Baum, 1972] to determine the options' components. The resulting algorithm offers three key benefits

1. It infers a data-driven segmentation of the state-space to learn the initialization and termination probability for each option.

2. It is applicable to discrete as well as continuous state-action spaces.

3. It outperforms monolithic algorithms on discrete tasks and can solve complex continuous tasks by combining simpler sub-policies.

Together, these contributions allow for learning of options from data with minimal prior knowledge.

3.1.1 Problem Statement

We consider the SMDP setting with states $\boldsymbol{s} \in \mathscr{S}$, actions $\boldsymbol{a} \in \mathscr{A}$, option indices $o \in \mathscr{O}$, and termination events $b \in \mathscr{B}$. Every option consists of a sub-policy $\pi(\boldsymbol{a}|\boldsymbol{s}, o)$ and a termination policy $\pi(b|\boldsymbol{s}, \bar{o})$, where b encodes a binary termination event. Furthermore, one global activation policy $\pi(o|\boldsymbol{s})$ governs the initialization of options following a termination event. A new option can only be initialized if the previously active option $\bar{o}$ is terminated. Thus, the agent uses the same option for multiple time steps if it is not terminated. The option transition model is given as

$$p(o|\boldsymbol{s}, b, \bar{o}) = \begin{cases} \pi(o|\boldsymbol{s}) & \text{if } b = 1, \\ \delta_{o=\bar{o}} & \text{if } b = 0, \end{cases} \tag{3.1}$$

and ensures that option $\bar{o}$ remains active until a termination event occurs. After each termination a new option will be sampled according to $\pi(o|\boldsymbol{s})$. In Section 3.2, we show how to learn a hierarchical policy given a set of demonstrated trajectories, i.e., how to solve the imitation learning problem. We formulate the option framework as a graphical model where the index of the executed option is treated as latent variable. Subsequently, we show how an EM algorithm can be used

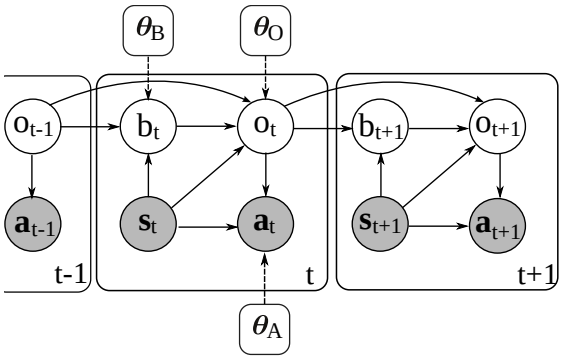


Figure 3.1.: The graphical model of the proposed framework. The termination policy $\pi(b|\boldsymbol{s}, o)$ decides whether to keep executing the previously active option or to terminate its execution. After a termination event, the activation policy $\pi(o|\boldsymbol{s})$ samples a new option and the sub-policy $\pi(\boldsymbol{a}|\boldsymbol{s}, o)$ samples an action. Arrows from $\boldsymbol{s}_t, \boldsymbol{a}_t$ to $\boldsymbol{s}_{t+1}$ are not shown to reduce clutter as we do not aim to model the transition probability distribution. Shaded nodes indicate observed variables, while clear nodes indicate the hidden variables. The model parameters $\boldsymbol{\theta}_B, \boldsymbol{\theta}_O, \boldsymbol{\theta}_A$ are shown in rounded square boxes.

to infer the parameters of the option components. In Section 3.3 we extend the imitation learning solution to allow for reinforcement learning, i.e., for iteratively improving the hierarchical policy such that it maximizes a reward function.

3.2 Learning Options From Data

The goal of this chapter is to determine sub-policies, termination policies and the activation policy from data with minimal prior knowledge. All option components are represented by parametrized distributions, governed by the parameter vector $\boldsymbol{\theta} = \{\boldsymbol{\theta}_A, \boldsymbol{\theta}_O, \boldsymbol{\theta}_B\}$. The individual components are given as binary classifiers for the termination policies for each option $p(b|\boldsymbol{s}, o = i; \boldsymbol{\theta}_B^i)$, one global multi-class classifier for the activation policies $\pi(o|\boldsymbol{s}; \boldsymbol{\theta}_O)$ and the individual sub-policies $\pi(\boldsymbol{a}|\boldsymbol{s}, o = i; \boldsymbol{\theta}_A^i)$. Our goal is to estimate the set of parameters $\boldsymbol{\theta} = \{\boldsymbol{\theta}_A, \boldsymbol{\theta}_O, \boldsymbol{\theta}_B\}$, which explain one or more demonstrated trajectories $\tau = \{\tau_1, \ldots, \tau_T\}$ with $\tau_t = \{\boldsymbol{s}_t, \boldsymbol{a}_t\}$. Crucially, the observations τ do not contain the option indices o nor the termination events b but only states $\boldsymbol{s}$ and actions $\boldsymbol{a}$. Both, the option index and the termination events are latent variables.

For this Section, we ignore the optimality of the resulting trajectories and focus on the imitation learning problem, i.e., how to recover hierarchical policies from trajectory observations. We propose a probabilistic option framework, where all option components are represented as distributions. We can then recover distributions over the latent variables using probabilistic inference techniques.

3.2.1 The Graphical Model for Options

To apply these probabilistic inference techniques, we need to specify how the variables in our model interact with each other. Figure 3.1 shows a graphical model of the proposed setting and the resulting hierarchical policy can be given as

$$\pi(\boldsymbol{a}|\boldsymbol{s},\bar{o};\boldsymbol{\theta}) = \sum_{o\in\mathcal{O}}\sum_{b\in\mathcal{B}} \pi(b|\boldsymbol{s},\bar{o};\boldsymbol{\theta}_B)p(o|\boldsymbol{s},b,\bar{o};\boldsymbol{\theta}_O)\pi(\boldsymbol{a}|\boldsymbol{s},o;\boldsymbol{\theta}_A). \tag{3.2}$$

The graphical model in Fig 3.1 shows that the following operations occur in every time step. First, the termination policy $\pi(b|\boldsymbol{s},\bar{o};\boldsymbol{\theta}_B)$ samples a termination event b. According to Equation (3.1) the previously active option $\bar{o}$ remains active if no termination occurs. Otherwise, the activation policy $\pi(o|\boldsymbol{s};\boldsymbol{\theta}_O)$ samples a new option index o based on the current state $\boldsymbol{s}$. Finally, the sub-policy $\pi(\boldsymbol{a}|\boldsymbol{s},o;\boldsymbol{\theta}_A)$ samples an action $\boldsymbol{a}$ based on the state $\boldsymbol{s}$.

3.2.2 Expectation Maximization for Options

The graphical model for the option framework is a special case of a hidden Markov model (HMM). The Baum-Welch algorithm [Baum, 1972] is an EM algorithm for estimating the parameters of a HMM. We will now state the Baum-Welch algorithm for our special case of the option model.

In our graphical model, the latent variables $\{o_{1:T}, b_{1:T}\}$ are given by trajectories of the option index and the termination event. EM algorithms optimize a lower-bound of the marginal log-likelihood

$$\log p(\tau|\boldsymbol{\theta}) \geq \sum_{o_{1:T},b_{1:T}} p(o_{1:T},b_{1:T}|\tau,\theta)\log p(o_{1:T},b_{1:T},\tau|\boldsymbol{\theta}) = Q(\boldsymbol{\theta},\theta). \tag{3.3}$$

The lower-bound is maximized iteratively. In the expectation (E-) Step, we compute the posterior probabilities of the latent variables using our current estimate of the parameters θ. In the maximization (M-) Step, we maximize $Q(\boldsymbol{\theta},\theta)$ w.r.t $\boldsymbol{\theta}$ to obtain a new estimate of the parameter vector. As the lower-bound is tight after every E-Step, the EM algorithm is guaranteed to converge to a local maximum of the marginal log-likelihood.

As we are dealing with time series, we can leverage the structure of the trajectory distribution

$$\begin{aligned}\log p(o_{1:T},b_{2:T},\tau|\boldsymbol{\theta}) \;=\; & p(o_1|\boldsymbol{s}_1)p(a_1|o_1,\boldsymbol{s}_1) \\ & \prod_{t=1}^{T-1} p(b_{t+1}|\boldsymbol{s}_t,o_t;\boldsymbol{\theta}_B)p(o_{t+1}|\boldsymbol{s}_{t+1},b_{t+1},\boldsymbol{\theta}_O)p(\boldsymbol{a}_{t+1}|\boldsymbol{s}_{t+1},o_{t+1};\boldsymbol{\theta}_A)p(\boldsymbol{s}_{t+1}|\boldsymbol{a}_t,\boldsymbol{s}_t).\end{aligned} \tag{3.4}$$

After setting Eq. (3.4) in Eq. (3.3) and rearranging terms, the lower bound decomposes in a sum over the time steps, i.e.,

$$\begin{aligned} Q(\boldsymbol{\theta}, \theta) = &\sum_{t=1}^{T-1} \sum_{o_t} \sum_{b_{t+1}} p(o_t, b_{t+1}|\tau; \theta) \log \pi(b_{t+1}|\boldsymbol{s}_{t+1}, o_t; \boldsymbol{\theta}_B) \\ &+ \sum_{t=1}^{T} \sum_{o_t} p(o_t|\tau; \theta) \log \pi(\boldsymbol{a}_t|\boldsymbol{s}_t, o_t; \boldsymbol{\theta}_A), \\ &+ \sum_{t=1}^{T} \sum_{o_t} p(o_t, b_t = 1|\tau; \theta) \log \pi(o_t|\boldsymbol{s}_t; \boldsymbol{\theta}_O). \end{aligned} \tag{3.5}$$

The posterior probabilities $p(o_t, b_{t+1}|\tau; \theta), p(o_t|\tau; \theta)$ and $p(o_t, b_t = 1|\tau; \theta)$ can be recovered from two posterior belief distributions

$$\xi_t(o_{t:t+1}, b_{t:t+1}) = p(o_{t:t+1}, b_{t:t+1}|\tau; \theta), \tag{3.6}$$

and

$$\gamma_t(o_t, b_t) = p(o_t, b_t|\tau; \theta). \tag{3.7}$$

Consequently, we need to infer only the posteriors γ_t and ξ_t in the E-Step, and not the probability $p(o_{1:T}, b_{1:T}|\tau)$ of a whole trajectory.

Expectation Step.

During the expectation step, the responsibilities

$$\gamma_t(o_t, b_t) = z_t^{-1} \alpha_t(o_t, b_t) \beta_t(o_t, b_t), \tag{3.8}$$

and

$$\xi_t(o_{t:t+1}, b_{t:t+1}) = z_t^{-1} \alpha_t(o_t, b_t) \beta_{t+1}(o_{t+1}, b_{t+1}) p(\boldsymbol{a}_{t+1}, o_{t+1}, b_{t+1}|\boldsymbol{s}_{t+1}, o_t; \theta), \tag{3.9}$$

can be determined using forward messages $\alpha_t(o_t, b_t)$ and backward messages $\beta_t(o_t, b_t)$, where z_t is the normalizing constant of $\gamma_t(o_t, b_t)$. Given the option model, the forward messages

$$\alpha_t(o_t, b_t) = p(\boldsymbol{a}_{1:t}, o_t, b_t|\boldsymbol{s}_{1:t}; \theta)$$

can be computed recursively by

$$\alpha_t(o_t, b_t) = \sum_{o_{t-1}} \sum_{b_{t-1}} \alpha_{t-1}(o_{t-1}, b_{t-1}) p(\boldsymbol{a}_t, b_t, o_t|\boldsymbol{s}_t, o_{t-1}; \theta), \tag{3.10}$$

with $\alpha_1(o_1, b_1) = p(\boldsymbol{a}_1, b_1, o_1|\boldsymbol{s}_1; \theta)$. Based on these forward messages, the backward messages

$$\beta_t(o_t, b_t) = p(\boldsymbol{a}_{t+1:T}, |\boldsymbol{s}_{t+1:T}, b_t, o_t; \theta) \tag{3.11}$$

are computed recursively by

$$\beta_{t-1}(o_{t-1}, b_{t-1}) = \sum_{o_t} \sum_{b_t} \beta_t(o_t, b_t) p(\boldsymbol{a}_t, b_t, o_t|\boldsymbol{s}_t, o_{t-1}, b_{t-1}; \theta), \tag{3.12}$$

with $\beta_T(o_T, b_T) = \mathbf{1}$.

Maximization Step.

Given the distributions over latent variables and the observed state-action samples, the parameters $\boldsymbol{\theta}$ can be determined by maximizing Equation (3.5). Since $Q(\boldsymbol{\theta}, \theta)$ is decoupled, independent optimization can be performed for the sub-policies, termination policies and the activation policy.

Termination Policies.

For example, for optimizing the termination models, we have to consider the first term of the lower bound which is given by

$$Q_B^i(\boldsymbol{\theta}_B, \theta) = \sum_t \sum_{b_{t+1}} p(o_t = i, b_{t+1}|\tau; \theta) \log \pi(b_{t+1}|\boldsymbol{s}_{t+1}, o_t; \boldsymbol{\theta}_B) \tag{3.13}$$

for each time step t and a single option $o_t = i$. This term can be rewritten as

$$\begin{aligned} Q_B^i(\boldsymbol{\theta}_B, \theta) = \sum_t w_{B,i}(t) \Big(& t_{B,i}(t) \log p(b_{t+1}=1|o_t=i; \boldsymbol{\theta}_B) \\ & + \big(1 - t_{B,i}(t)\big) \log \big(1 - p(b_{t+1}=1|o_t=i; \boldsymbol{\theta}_B)\big) \Big), \end{aligned} \tag{3.14}$$

where $w_{B,i}(t)$ is a weight and $t_{B,i}(t)$ defines the target probability of the termination event. Equation (3.14) resembles a weighted cost function for logistic regression [Bishop, 2006] where the weights are given by

$$w_{B,i}(t) = p(o_t = i|\tau; \theta) = \sum_{b_t} \gamma(o_t = i, b_t).$$

The target probability of the termination event is given by

$$t_{B,i}(t) = p(b_{t+1} = 1|o_t = i, \tau; \theta) = \frac{\sum_{o_{t+1}, b_t} \xi_t(o_t = i, o_{t+1}, b_t, b_{t+1} = 1)}{p(o_t = i|\tau; \theta)}.$$

Using these target probabilities and weights, standard techniques can be used to fit, for example, a sigmoidal classifier for each termination policy [Bishop, 2006].

Activation Policy.

The case of the activation policy $\pi(o|\boldsymbol{s}; \boldsymbol{\theta}_O)$ follows the argument of the termination policies. Similarly to the termination policies, we consider the relevant term of the lower bound

$$Q_O(\boldsymbol{\theta}_O, \theta) = \sum_{t=1}^{T} \sum_{o_t} p(o_t, b_t = 1|\tau; \theta) \log \pi(o_t|\boldsymbol{s}_t; \boldsymbol{\theta}_O), \tag{3.15}$$

to extract weights

$$w_O(t) = p(b_t = 1|\tau; \theta) = \sum_o \gamma_t(o_t, b_t),$$

and target probabilities

$$t_O(t) = p(o_t|\tau, b_t = 1; \theta) = \frac{\gamma_t(o_t, b_t)}{\sum_o \gamma_t(o_t, b_t)}.$$

While there exists one termination policy for each option, only a single global activation policy governs the initialization of all options. Thus, only one multi-class classifier has to be learned. Given the weighted target probabilities of the activation policy, standard methods can be used to fit a multi-class classifier [Bishop, 2006].

Sub-Policies.

Finally, the sub-policies have to be fit. The relevant terms of the lower bound are given by

$$Q_A(\boldsymbol{\theta}_A, \theta) = \sum_{t=1}^{T} \sum_{o_t} p(o_t|\tau;\theta) \log \pi(\boldsymbol{a}_t|\boldsymbol{s}_t, o_t; \boldsymbol{\theta}_A), \tag{3.16}$$

such that the weights are given by

$$w_{A,j}(t) = \; p(o_t = j|\tau;\theta) = \sum_{b_t} \gamma_t(o_{t,j}, b_t).$$

Since the target values are the observed state-action pairs, i.e., $t_A(t) = \{\boldsymbol{s}_t, \boldsymbol{a}_t\}$, these do not have to be computed. Given the weighted state-action pairs, stochastic policies, such as linear Gaussian policies, can be fit to the data.

Feature Representations of the State.

The above derivations are given in their most general form, where each option component depends directly on the state $\boldsymbol{s}$. In practice, it may often be beneficial to train the individual components on a feature transformation $\boldsymbol{\phi}(\boldsymbol{s})$ of the state. Such features might, for example, be polynomial expansion of the state variable or a kernelized representation. When using feature representations, different representations can be chosen for the individual option components.

3.3 Probabilistic Reinforcement Learning for Option Discovery

The results in the previous section allow us to recover a hierarchical policy from state-action observations, i.e., to perform imitation learning with hierarchical policies. In this section, we extend these results to allow for reinforcement learning. Using the hierarchical policy, the agent aims to maximize the expected reward

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t) \right], \tag{3.17}$$

where γ is the discount factor.

3.3.1 Probabilistic Reinforcement Learning Algorithms

There exist several algorithms which use probabilistic inference techniques for computing the policy update in reinforcement learning [Dayan and Hinton, 1993, Theodorou et al., 2010, Kober and Peters, 2010, Peters et al., 2010]. More formally, they either re-weight state-action trajectories or state-action pairs according to the estimated quality of the state-action pair and, subsequently, use a weighted maximum likelihood estimate to obtain the parameters of a new policy π^*.

A common approach is to to use an exponential transformation of the advantage function $A(\boldsymbol{s}, \boldsymbol{a}) = Q(\boldsymbol{s}, \boldsymbol{a}) - V(\boldsymbol{s})$, where $Q(\boldsymbol{s}_i, \boldsymbol{a}_i)$ is the Q-function and $V(\boldsymbol{s}_i)$ is the value function,

to reweight the state-action distribution [Peters et al., 2010, Daniel et al., 2012a]. The resulting desired state-action distribution $p(\boldsymbol{s}, \boldsymbol{a})$ is then given by

$$p(\boldsymbol{s}, \boldsymbol{a}) \propto q(\boldsymbol{s}, \boldsymbol{a}) \exp\left(\frac{Q(\boldsymbol{s}, \boldsymbol{a}) - V(\boldsymbol{s})}{\eta}\right),$$

where $q(\boldsymbol{s}, \boldsymbol{a})$ is the sampling distribution which is typically obtained by sampling from the old policy $\tilde{\pi}(\boldsymbol{a}|\boldsymbol{s})$. The parameter η is a temperature parameter that is either optimized by the algorithm [Peters et al., 2010, Daniel et al., 2012a] or manually set [Theodorou et al., 2010, Kober and Peters, 2010]. A new parametrized policy π^* can then be obtained by minimizing the expected Kulback-Leibler divergence between the re-weighted policy update $p(\boldsymbol{a}|\boldsymbol{s})$ and the new parametric policy π^* [van Hoof et al., 2015], i.e.,

$$\begin{aligned}
\pi^* &= \text{argmin}_\pi \mathbb{E}_{p(\boldsymbol{s})}[\text{KL}(p(\boldsymbol{a}|\boldsymbol{s})||\pi(\boldsymbol{a}|\boldsymbol{s}))] \\
&= \text{argmax}_\pi \int p(\boldsymbol{s}, \boldsymbol{a}) \log \pi(\boldsymbol{a}|\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a} + \text{const} \\
&= \text{argmax}_\pi \int q(\boldsymbol{s}, \boldsymbol{a}) \exp\left(\frac{Q(\boldsymbol{s}, \boldsymbol{a}) - V(\boldsymbol{s})}{\eta}\right) \log \pi(\boldsymbol{a}|\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{a} + \text{const} \\
&\approx \text{argmax}_\pi \sum_{(\boldsymbol{s}_i, \boldsymbol{a}_i) \sim q(\boldsymbol{s}, \boldsymbol{a})} w(\boldsymbol{s}_i, \boldsymbol{a}_i) \log \pi(\boldsymbol{a}_i|\boldsymbol{s}_i),
\end{aligned} \tag{3.18}$$

where

$$w(\boldsymbol{s}_i, \boldsymbol{a}_i) = \exp\left(\frac{Q(\boldsymbol{s}_i, \boldsymbol{a}_i) - V(\boldsymbol{s}_i)}{\eta}\right)$$

defines a weighting for each state action pair. It can now be easily seen that Equation 3.18 is equivalent to a weighted maximium likelihood estimate for the policy π^*.

We will employ different RL algorithms in the continuous and discrete setting to compute these weightings. For the continuous experiments, we will use the HiREPs [Daniel et al., 2012a] algorithm. While other algorithms are equally feasible [Theodorou et al., 2010, Kober and Peters, 2010, Peters et al., 2010], HiREPS is able to actively separate sub-policies if options are available by ensuring that different sub-policies generate distinct behaviors in similar states. Ensuring such versatility of the sub-policies is achieved by constraining the entropy of responsibilities for each option, i.e.,

$$H\big(p(\boldsymbol{a}|\boldsymbol{s}, o)\big) > \kappa, \tag{3.19}$$

where κ is manually set. Furthermore, HiREPS limits the KL divergence of state-action distributions induced by policy updates such that

$$\text{KL}\big(p(\boldsymbol{s}, \boldsymbol{a})||q(\boldsymbol{s}, \boldsymbol{a})\big) < \epsilon, \tag{3.20}$$

where ϵ is equally set manually. Values for κ and ϵ are usually set close to 1.

For discrete environments, we can equally employ standard reinforcement learning techniques to obtain the Q-function $Q(\boldsymbol{s}, \boldsymbol{a})$ and value function $V(\boldsymbol{s})$. In our experiments, we employed standard Q-learning [Watkins and Dayan, 1992] and LSPI [Lagoudakis and Parr, 2003] to obtain those quantities. In the discrete case, the temperature parameter η was set to 1.

3.3.2 Combining EM and Probabilistic Reinforcement Learning

As we have seen, the only difference between imitation learning and probabilistic reinforcement learning algorithms is the use of a weighted maximum likelihood (ML) estimate instead of a standard ML estimate. We can now combine the expectation maximization algorithm for discovering parametrized options with probabilistic reinforcement learning algorithms by weighting each time step in the maximization step of the EM algorithm.

Since the reinforcement learning weights $w_t = w(\boldsymbol{s}_t, \boldsymbol{a}_t)$ are independent of all latent variables, the derivations in Section 3.2 remain largely unaffected. The integration of the reinforcement learning weights affects only the maximization step. In the M-Step, all sample weights for the individual option components obtained by the EM algorithm are multiplied by the RL weights. Thus, we obtain the following combined weights of each sample

$$\begin{aligned}\tilde{w}_{B,j}(t) &= w_t p(o_t = j|\tau;\theta),\\ \tilde{w}_G(t) &= w_t p(b_t = 1|\tau;\theta),\\ \tilde{w}_{A,j}(t) &= w_t p(o_t = j|\tau;\theta).\end{aligned}$$

Since the RL weightings only depend on the observed variables, we fortunately do not have to devise a new RL algorithm but can rely on a wide range of existing methods to provide the RL weights w_t. Thus, the proposed framework acts as an interface between existing RL algorithms and the policy updates. While traditional methods use the RL weights w_t to perform, for example, a maximum likelihood update of a monolithic policy $\pi(\boldsymbol{a}|\boldsymbol{s})$, the proposed method estimates all elements of the option framwork by fitting the hierarchical policy to the weighted state-action samples.

The information flow of the proposed algorithm is shown in Table 3.1.

3.4 Related Work

Options as temporally extended macro-actions were introduced by Sutton et al. [1999b]. While previous research leveraged the power of temporal abstraction [Kaelbling, 1993, Parr and Russell, 1998, Sutton et al., 1999b], such efforts did not improve the sub-policies themselves. Improving the sub-policies based on the observed state-action-reward sequences is known as intra-option learning. Sutton et al. [1998] showed that making use of intra-option learning can drastically improve the overall learning speed. Yet, the algorithms presented by Sutton et al. [1998] relied on hand coded options and were presented in the discrete setting.

Options are also used in many hierarchical RL approaches to either extend the action space or options are directly extended to sub-tasks, where the overall problem is broken up into potentially simpler sub-problems. Dietterich [2000] proposed the MAXQ framework which uses several layers of such sub-tasks. However, the structure of these sub-tasks needs to be either specified by the user [Dietterich, 2000], or they rely on the availability of a successful trajectory [Mehta et al., 2008]. Barto et al. [2004] rely on artificial curiosity to define the reward signal of individual sub-tasks, where the agent aims to maximize its knowledge of the environment to solve new tasks quicker. This approach relies on salient events which effectively define the sub-tasks.

Stolle and Precup [2002] first learn a flat solution to the task at hand and, subsequently, use state visitation statistics to build the option's initiation and termination sets. Mann and Mannor [2014] apply the options framework to value iteration and show that it can speed up convergence.

Input: Number of sub-policies O, number of iterations L, number of episodes per iteration M, reinforcement learner $f_{\mathrm{RL}}(\boldsymbol{s}, \boldsymbol{a}, r)$.
Initialize $\pi(\boldsymbol{a}|\boldsymbol{s}, o; \theta_A)$,$\pi(o|\boldsymbol{s}; \theta_O)$ and $\pi(b|\boldsymbol{s}, \bar{o}; \theta_B)$.

for $l \leftarrow 1$ to L (# iterations)

Collect samples

for $i \leftarrow 1$ to M (# episodes)

Sample initial state $\boldsymbol{s}_{i,t}$ from environment.

Sample $b \sim \pi(b|\boldsymbol{s}, \bar{o}; \theta_B)$

if $b = 1$

Sample option $o \sim \pi(o|\boldsymbol{s}; \theta_O)$,

Sample action $\boldsymbol{a} \sim \pi(\boldsymbol{a}|\boldsymbol{s}, o; \theta_A)$.

Observe reward $r_{i,t}(\boldsymbol{a}_{i,t}, \boldsymbol{s}_{i,t})$ and next state $\boldsymbol{s}_{i,t+1}$.

Compute Sample Weights:

$\tilde{w}(\boldsymbol{s}, \boldsymbol{a}) \leftarrow f_{\mathrm{RL}}(\boldsymbol{s}, \boldsymbol{a}, r)$.

Calculate Lower Bound:

Compute Messages:

$\alpha_t(o_t, b_t) \leftarrow p(\boldsymbol{a}_{1:t}, o_t, b_t|\boldsymbol{s}_{1:t}; \theta)$

$\beta_t(o_t, b_t) \leftarrow p(\boldsymbol{a}_{t+1:T}, |\boldsymbol{s}_{t+1:T}, b_t, o_t; \theta)$

Compute responsibilities:

$\gamma_t(o_t, b_t) \leftarrow \alpha_t(o_t, b_t)\beta_t(o_t, b_t)z_t^{-1}$

$\xi_t(o_{t:t+1}, b_{t:t+1}) \leftarrow \alpha_t(o_t, b_t)\beta_{t+1}(o_{t+1}, b_{t+1})$
$p(\boldsymbol{a}_{t+1}, o_{t+1}, b_{t+1}|\boldsymbol{s}_{t+1}, o_t; \theta)z_t^{-1}$

Compute weights $w_{B,o}$, $w_{A,o}$, and w_G and target values $t_{B,o}$ and t_G .

Update Policy:

with weights, target values and state-action pairs.

Output: Policies $\pi^*(\boldsymbol{a}|\boldsymbol{s}, o)$, $\pi^*(o|\boldsymbol{s})$ and $\pi^*(b|\boldsymbol{s}, o)$.

Table 3.1.: Learning options from experience. Termination events, options and actions are sampled from from the current policies. Subsequently, the distribution over latent variables is computed and weights f_{RL} are proposed by the RL algorithm The next policies are determined according to the update equations in the method section.

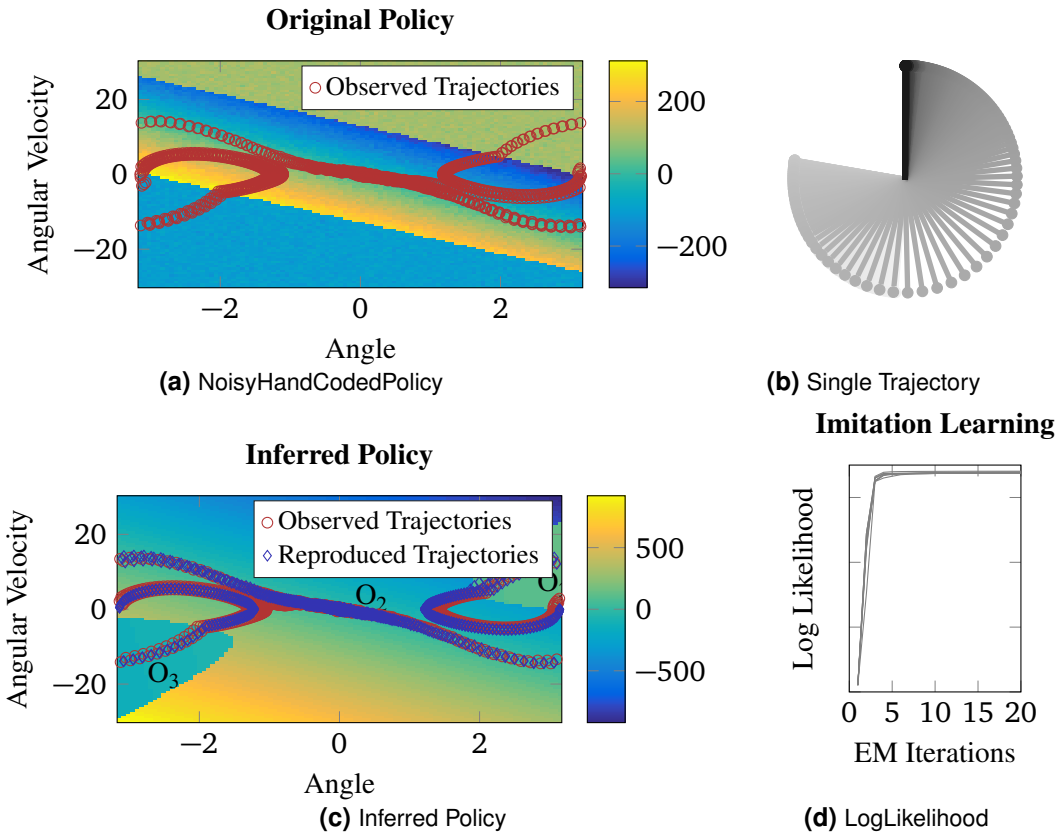
Option discovery approaches often aim to identify so called bottleneck states, i.e., states the agent has to pass pass through on its way from start to goal. McGovern and Barto proposed to formulate this intuition as a multiple-instance learning problem and solve it using a diverse density method [McGovern and Barto, 2001]. Other approaches aim to find such bottleneck states using graph theoretic algorithms. The Q-Cut [Menache et al., 2002] and L-Cut [Simsek and Barto, 2008] build transition graphs of the MDP and solve a min cut problem to find bottleneck states. Silver and Ciosek [2012] assume a known MDP model to propose an option model composition framework, which can be used for planning while discovering new options. Niekum and Barto [2011] present a method to cluster subgoals discovered by existing subgoal discovery methods to find skills that generalize across tasks. In the presented chapter, we do not assume knowledge of the underlying MDP and, further, present a framework which is also suitable for the continuous setting.

In continuous state-action settings, several sub-task based approaches have been proposed. Ghavamzadeh and Mahadevan [2003] proposed the use of a policy gradient method to learn sub-tasks while the selection of the sub-tasks is realized through Q-Learning. Morimoto and Doya [2001] proposed to learn how to reach specific joint configurations of a robot as sub-tasks, such that these options can later be combined to learn more complicated tasks. In both approaches, the sub-tasks have to be pre-specified by the user. Wingate et al. [2011] use policy priors to encode desired effects like temporal correlation. Konidaris and Barto [2009] use the option framework to learn chaining of skills. This approach requires that the agent can reach the goal state before constructing the series of options leading to the goal state.

A concept similar to the options framework has been widely adapted in the field of robot learning. There, temporal abstraction is achieved through the use of movement primitives [Paraschos et al., 2013, Da Silva et al., 2012]. Instead of learning policies over state-torque mappings to control robots, the agent learns parameters of a trajectory generator [Kober and Peters, 2010, Kajita et al., 2003]. Based on a Beta-Process Autoregressive HMM proposed by Fox et al. [2009], Niekum et al. [2012] proposed a method to segment demonstrated trajectories into a sequence of primitives, addressing the imitation learning problem. Sequencing multiple primitives can be viewed as an approximation to the options framework for robot learning and allows more challenging tasks to be solved [Stulp and Schaal, 2012, Daniel et al., 2013]. Robot learning approaches often benefit from substantial task knowledge in the form of task demonstrations. Furthermore, a key benefit of these methods is the ability to offer a simple parametrization of complex behaviors, which also inspired the proposed approach. However, existing robot learning methods usually do not allow for intra-option learning from the complete trajectory data [Daniel et al., 2012a] and the individual primitives are generally of fixed duration [Kober and Peters, 2010].

3.5 Experimental Evaluations

The evaluation of the proposed framework is separated in two parts. We first evaluated the imitation learning capabilities and, subsequently, proceeded to evaluate different reinforcement learning tasks as well as comparing the proposed methods to other option learning frameworks.



(a) NoisyHandCodedPolicy **(b)** Single Trajectory

(c) Inferred Policy **(d)** LogLikelihood

Figure 3.2.: a) The hand coded policy consists of three bands, where the center band realizes a stabilizing controller and the outer bands accelerate the pendulum to enable a swing-up. b) One trajectory sample generated with the hand coded policy. c) The policy learned through imitation learning. Red circles show the observations and blue diamonds show trajectories generated using this learned policy. The plot shows that the learned policy uses only two of three available options. Option 2 realizes the stabilizing policy and Option 3 realizes the accelerating policy. d) A qualitative plot showing the log likelihood of the observed data under the learned policy for ten trials. The results show that convergence is usually reached after about five EM iterations.

3.5.1 Imitation Learning

We started our evaluations with an imitation learning task. The evaluation of the imitation learning capabilities allowed us to ensure that the foundation of the proposed framework performs as expected.

For the imitation learning task, we evaluated the underpowered pendulum swing-up task. In this task, the agent exerted torques on the rotational joint of a pendulum and had to swing up the pendulum into the upright position. However, the torques were limited such that the agent was not strong enough to perform a swing up directly. Instead, the agent had to first perform a pre-swing to build up sufficient kinetic energy. The pendulum had a length of 0.5m, a mass of 10kg and a friction coefficient of 0.2Ns/m. The pendulum was internally simulated with a frequency of 10kHz. The agent controlled the pendulum at a frequency of 33Hz. The agent could exert at most 30Nm of torque and each episode had 100 time steps. This continuous task had two state

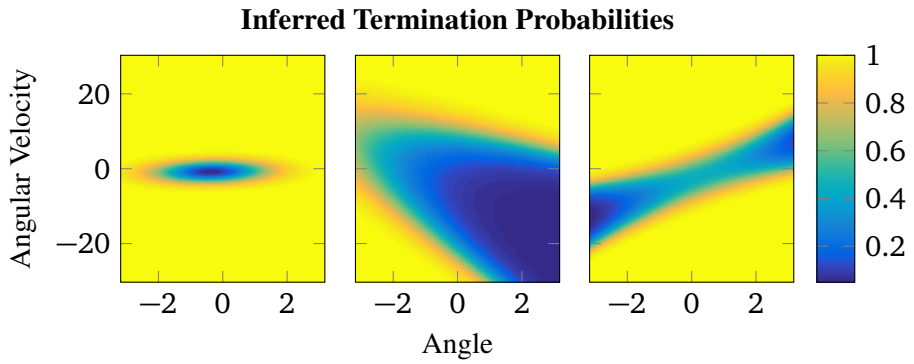


Figure 3.3.: Visualization of the Inferred Termination Policies for options one (left) through three (right). As shown in Fig. 3.2c, the first option is not used in the final policy and the agent learned to almost always terminate the first option. The third option learned a symmetric termination policy which terminates once the pendulum has sufficient energy to be 'pulled up' to the upright position. Interestingly, the second option learned a non-symmetric termination policy, possibly due to the relatively small number (five) of noisy demonstrations.

variables, the angle and the velocity of the pendulum, where we used a periodic representation of the angle.

To generate observations, we provided a hand-coded policy which is shown in Fig. 3.2a. This policy was designed to generate noisy actions within ±300Nm. However, the torques on the system were capped to the torque limit of 30Nm. The much larger range of desired torques was chosen to simplify the programming of the controller. This policy could successfully perform a pendulum swingup if the pendulum was initially hanging down with zero velocity. An example trajectory generated by this hand coded policy is shown in Fig. 3.2b.

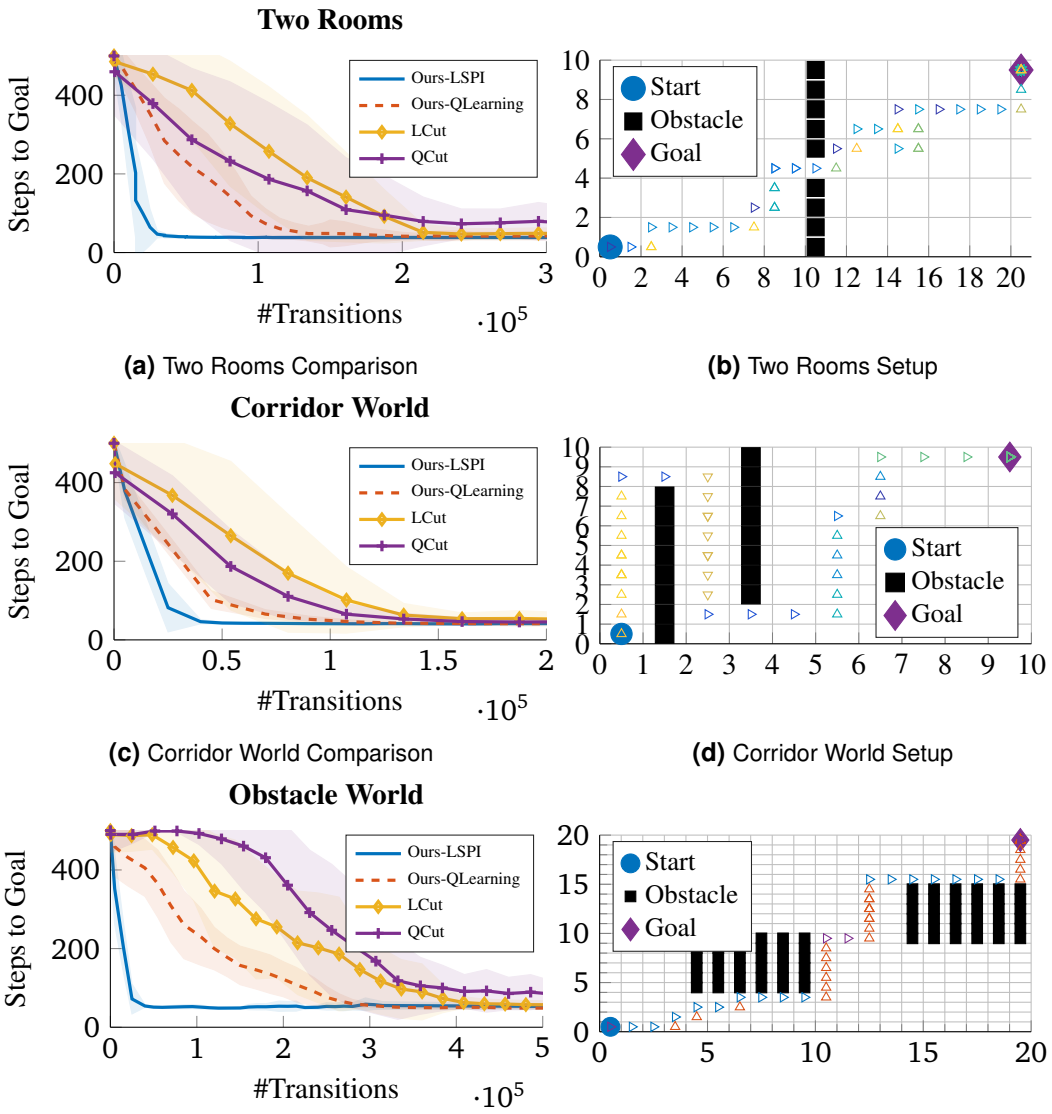
Based on five observed trajectories, we used the proposed framework to learn a policy with three options. However, the resulting policy, shown in Fig. 3.2c, learned to reproduce an effective policy using only two of the three available policies. Fig. 3.2c shows that the trajectories generated with this imitated policy closely resemble the observed trajectories. Fig. 3.3 shows the inferred termination policies of the different options. The results show that options will only be terminated once they are outside of their region of 'expertise'. Option one has a high termination probability in most regions, however, Fig. 3.2c shows that the final policy is actually not using this option.

Fig. 3.2d shows the development of the log likelihood of the observed data under the imitated policy. The results show that convergence typically has been reached after about five iterations of the EM algorithm.

In this task, the activation policy was a soft-max distribution based on a squared expansion of the state features. The termination policies were represented by sigmoidal functions based on the same features as the activation policy. The sub-policies, however, were represented as linear Gaussian policies based directly on the state features.

3.5.2 Reinforcement Learning with Temporal Correlations

We present results on three discrete state-action tasks as well as one continuous state-action task. In the discrete state action task settings, we compare to two existing option learning algorithms,

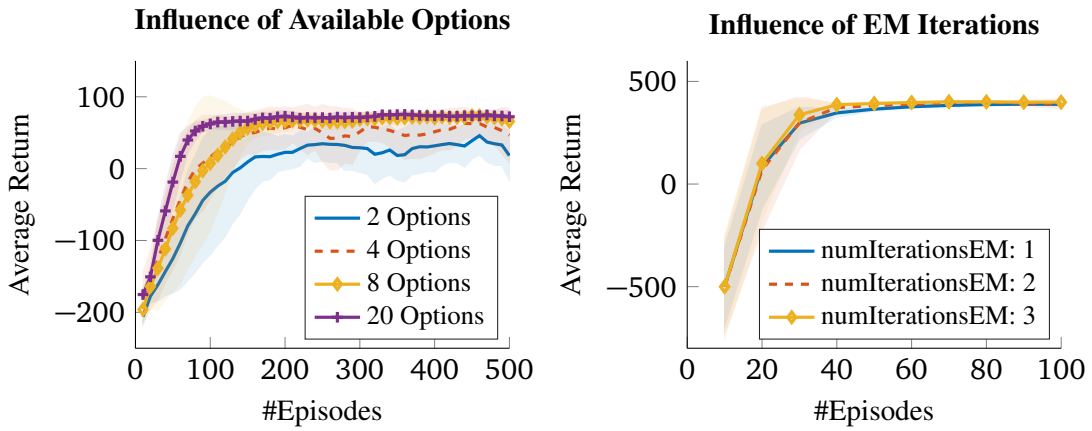


(a) Two Rooms Comparison **(b)** Two Rooms Setup

(c) Corridor World Comparison **(d)** Corridor World Setup

(e) Obstacle World Comparison **(f)** Obstacle World Setup

Figure 3.4.: b,d,f) Setups of the discrete world task. The agent has to cross fields of varying sizes while avoiding the obstacles on its way to the goal. Outlined triangles show trajectories learned using the proposed approach in combination with LSPI. The colors differentiate between active options. Since the transition dynamics were stochastic, the transitions did not always follow the selected actions. a,c,e) Learning performances of the different algorithms on the discrete world tasks. The results show that the L-Cut and Q-Cut generally had similar performance, where Q-Cut outperformed L-Cut in the first two worlds. However, in the relatively open obstacle world, L-Cut outperformed Q-Cut since Q-Cut was not able to find suitable bottleneck states. In all worlds the proposed approach outperformed both L-Cut and Q-Cut when using Q-Learning. Changing the RL algorithm to LSPI further increased the learning performance.



(a) Obstacle World #Options

(b) Obstacle World #EM Iterations

Figure 3.5.: a) Evaluation of the proposed algorithm on a gridworld task. With an increased number of options the performance of the algorithm also increases. With eight options, optimal asymptotic performance is reached, however adding even more options further improves the learning speed. b)The number of EM iterations in the RL case does not have a large effect on the performance. More EM iterations improve the performance only marginally.

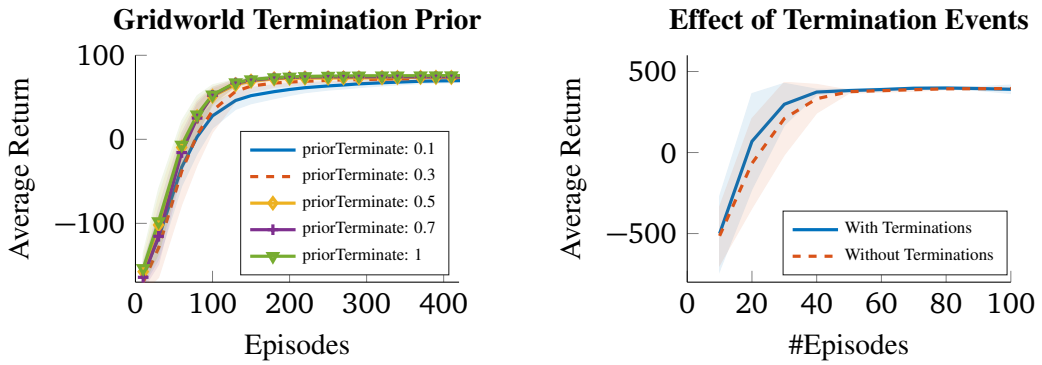
namely the Q-Cut algorithm [Menache et al., 2002] as well as the L-Cut algorithm [Simsek et al., 2005]. Both algorithms aim to find bottleneck states by solving a max flow/ min cut problem on the transition graph. After identifying such bottleneck states, options are generated which lead to these states. The main difference between these two algorithms is that Q-Cut aims to solve a global graph partitioning problem, while L-Cut aims to solve a local problem, based on transitions observed in one episode. The main difference to the proposed algorithm is that we do not aim to identify bottleneck states, but instead aim to break up the problem into simpler sub-problems.

For all evaluations, we tested each setting ten times and report mean and standard deviation of the results.

Discrete Grid World Tasks

The discrete environments were given by three different gridworlds as shown in Figures 3.4b, 3.4d and 3.4f. The first world shown in Fig 3.4b represents the two rooms scenario [McGovern and Barto, 2001], where the agent has to find a doorway to travel between two rooms. In the second world shown in Fig 3.4b, the agent has to traverse two elongated corridors before entering a big room in which the target is located. Finally, in the third world show in Fig. 3.4f no traditional bottleneck states appear, but the agent has to navigate around two obstacles. Furthermore, in this world optimal paths can lead around either side of the first obstacle.

In all experiments, the agent started in the lower level corner of the respective grid and had to traverse the grid while avoiding two obstacles to reach the goal at the opposite end. The actions available to the agent were going up, left, right and down. Transitions were successful with a probability of 0.8 and randomly sampled otherwise. The transition to each accessible field but the goal field generated a reward signal of -1. After reaching the goal, the agent received a reward of



(a) Obstacle World Termination Prior

(b) Obstacle World Terminations

Figure 3.6.: a) The effect of the termination prior (initialization of termination policies) using eight options. Changing the initial value of the termination probabilities within a reasonable range around 0.5 has only a small effect on the learning process. Very low values will lead to decreased performance. b) The effect of completely disabling terminations. Without terminations, the performance of the algorithm decreases.

$+1$ for every remaining step of the episode, where each episode had a length of 500 time steps. If the agent tried to walk into an obstacle or to leave the field, it remained in the current position.

For the discrete tasks we used a tabular feature encoding for the flat policy. For the hierarchical policy, we used the tabular features for the activation and termination policies. The sub-policies were state-independent multinomial distributions over the four actions.

In the discrete setting, we present results of two different RL learning methods, Q-Learning [Watkins and Dayan, 1992] and LSPI [Lagoudakis and Parr, 2003], in combination with the proposed framework and converted the resulting Q functions into RL weights as described in Section 3.3. However, especially when using Q-Learning, the resulting policy updates can be very different and may de-stabilize the learning process. To stabilize the learning process, we use a learning rate α for our policy updates such that

$$\tilde{\pi}(\boldsymbol{a}|\boldsymbol{s},\bar{o}) = \alpha\pi(\boldsymbol{a}|\boldsymbol{s},\bar{o}) + (1-\alpha)\hat{\pi}(\boldsymbol{a}|\boldsymbol{s},\bar{o}),$$

where $\pi(\boldsymbol{a}|\boldsymbol{s},\bar{o})$ is the resulting policy of the EM update and $\hat{\pi}(\boldsymbol{a}|\boldsymbol{s},\bar{o})$ is the previous policy. Alpha was set to 0.1 in the experiments.

Comparison To Existing Methods.

In the comparative results to related work we follow the therein established method of reporting 'steps to goal' as qualitative measure. In the remaining evaluations of our algorithm we report the average return, which may in some cases be more informative since our reward functions also punish 'falling off' the board. The results in Fig. 3.4 show that in all experiments the proposed framework learned solutions faster than both the Q-Cut as well as the L-Cut methods. Comparing the use of Q-Learning and LSPI in the proposed framework, the results show that LSPI leads to convergence considerably faster than Q-Learning.

Influence of Available Options.

After comparing to existing methods, we further evaluated the properties of the proposed framework. All remaining evaluations were performed using LSPI in the obstacle world. In our experience, these results were representative of both using Q-Learning as well as performing them in different tasks. Fig. 3.5a shows the influence of available options. In theory this task ca be solved optimally with only two options, where one will always go right and one will always go down. However, the results show that making more options available to the algorithm improved both asymptotic performance as well as speed of convergence. Adding more than 20 options did not further increase the performance.

Influence of EM Iterations.

The proposed EM style of computing policy updates can be costly and, generally, requires several iterations as also seen in the imitation learning case in Fig. 3.2d. However, in the RL setting, subsequent policies are expected to be similar and, thus, a small number of EM Iterations should be sufficient. The results in Fig. 3.5b show that this intuition holds true in our evaluations. In our experience, performing even more EM iterations did not change the result. However, when using very few rollouts per iteration, the effect of performing multiple EM iterations can be more pronounced.

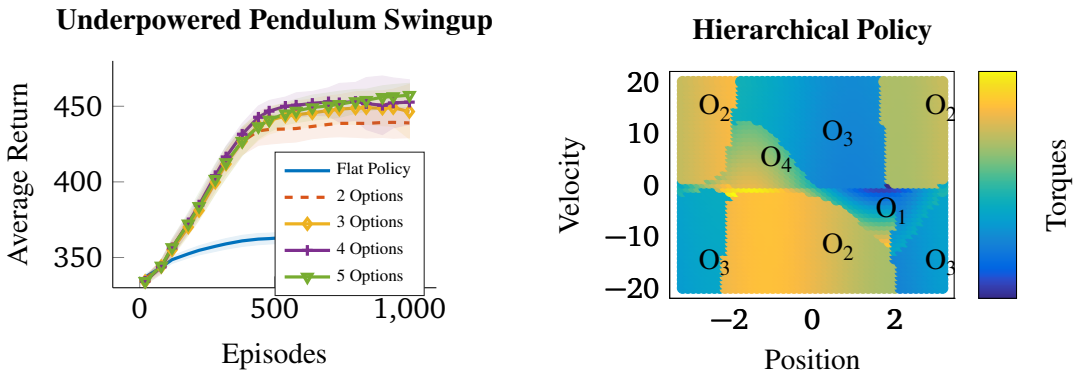
Influence of Termination Events.

Finally, we evaluated the influence of the probabilistic terminations. In the proposed framework, the sub-policies have to be initialized and, thus, a prior for the termination policies has to be set. Fig. 3.6a shows the effect of changing this prior. The results show that the proposed framework is robust to wide range of these initializations.

We also evaluated the effect of disabling the probabilistic termination sub-policies. In this case, the algorithm could still learn multiple options but no termination polices, Thus, each option could not be active for more than one time step but terminated after every step. The results in Fig. 3.6b show that learning without terminations slowed down the convergence speed. In our experience, this effect was strongly linked to the stochasticity of the transitions. The higher this stochasticity was, the stronger the benefit of the termination policies became.

Continuous Pendulum Swing-up Task

After evaluating our algorithm on several discrete tasks, we returned to the continuous pendulum-swingup task described in the imitation learning section of the results. As before, the task is to swing up an underpowered pendulum. To solve this task the agent has to learn to first perform a pre-swing to build up kinetic energy and then use the momentum to fulfill the task. Furthermore, in the presented task two solutions are possible, performing the swing-up clock-wise or counter-clock-wise. To provide a reinforcement learning signal for the continuous task, we employed the Hierarchical Relative Entropy Policy Search (HiREPS) algorithm [Daniel et al., 2012a], where we used Fourier basis features as described by Konidaris et al. [Konidaris et al., 2011] to represent the value function. For the value function a feature expansion of the fifth order was used. For the activation policy as well as the termination policies, a squared feature expansion was used. The individual sub-policies worked directly on the state observations, using only an additional constant offset. Thus, the sub-policies were linear controllers. In the pendulum swing-up task, a



(a) Pendulum Number of Options

(b) Hierarchical Policy

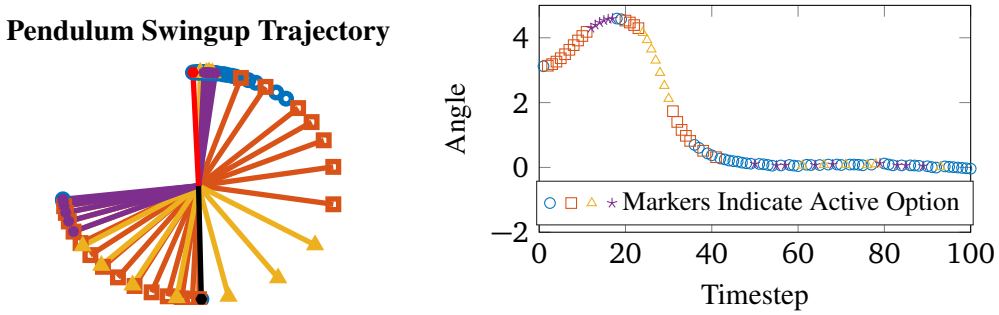
Figure 3.7.: (a) Evaluation of the proposed algorithm on the underpowered pendulum swingup task. A flat linear policy is insufficient to solve the task, only the combination of multiple linear policies is sufficiently expressive. Using two or three options, the task can be solved but stabilization depends on bang-bang control. With four or more options, the policy is sufficiently expressive to incorporate high torque signals for the swing-up and finely tuned sub-policies for stabilization. (b) Visualization of the hierarchical policy composed of four options. Options 2 and 3 learned a bang-bang control scheme and options 1 and 4 learned to stabilize the pendulum around the upright position. (Best viewed in color).

single linear controller is insufficient and, thus, the proposed approach had to combine multiple sub-policies to achieve the necessary non-linear behavior required for a swing-up.

The results in Figure 3.7a show that while a single linear policy was insufficient to solve this task, it could be solved using two options. Adding more options further improved the resulting hierarchical policy. The visualization of the resulting policy in Figure 3.7b shows that with more options, the algorithm learned a control scheme where options two and three were used to swing up the pendulum, and options one and four incorporated a linear stabilization scheme around the upright position of the pendulum. Figure 3.8a shows a trajectory generated by the resulting policy. Starting from the bottom, the pendulum was first accelerated by options two and three. The plot shows that in-between options two and three, option four was active for a few time steps. However, the kinetic energy at that point was insufficient to fully swing up the pendulum. After the pendulum almost reached the upright position around time step 40, the stabilizing options took over control. Since all option components are stochastic distributions, some option switches still occur even after the pendulum is stabilized. Since the effect of switching into a different option in the stable position for a single time step could easily be balanced by activating the stabilizing option in the next time step, the agent did not have a strong incentive to learn to avoid such behavior. Letting the algorithm run for more iterations might further improve this behavior.

3.6 Conclusion & Future Work

In this chapter, we presented a method to estimate the components of the option framework from data. The results show, that the proposed method is able to learn options in the discrete and continuous setting. In the discrete setting, the algorithm performs better than two related option-discovery algorithms which are based on exploiting bottle-neck states. Instead of relying on



(a) Trajectory **(b)** Trajectory in state space

Figure 3.8.: (a) Visualization of swingup trajectory color coded by the currently active option. The pendulum is initially hanging down with a small rotation. First option 2 performs a pre-swing and, subsequently, option 3 accelerates the pendulum until options 1 and 4 start the stabilization process. Around timestep 15, option 4 is active for a short time, but the kinetic energy at that point is insufficient for a direct swingup. (Best viewed in color).

bottle-neck states, the proposed algorithm achieves its performance by combining options with simpler sub-policies.

In the continuous setting, the results show that the algorithm is able to solve a non-linear task using a combination of options with only linear sub-policies. In this setting, a single linear policy is insufficient for solving the task. Furthermore, the framework allows for parametrized policies and, thus, state-of-the-art policy search methods developed for flat policies can be used to learn hierarchical policies.

The presented approach infers the option's structure, such as the activation policy and termination policies, from data. However, the number of options still has to be set a-priori by the practitioner. While the results show that setting a relatively large number of options typically yields good performance, learning the number of options is an important aspect of future work. Finally, while the presented framework estimates the most likely termination policies, finding a way of enforcing fewer terminations might further improve learning performance.

4 Learning from Human Ratings

In the previous chapters, we proposed methods to efficiently learn policies which solve a given task, where these tasks are described in the form of reward functions. Defining such reward functions, however, remains hard in many practical applications. For tasks such as grasping, there are no reliable success measures available. Defining reward functions by hand requires extensive task knowledge and often leads to undesired emergent behavior. We introduce a framework, wherein the robot simultaneously learns an action policy and a model of the reward function by actively querying a human expert for ratings. We represent the reward model using a Gaussian process and evaluate several classical acquisition functions from the Bayesian optimization literature in this context. Furthermore, we present a novel acquisition function, expected policy divergence. We demonstrate results of our method for a robot grasping task and show that the learned reward function generalizes to a similar task. Additionally, we evaluate the proposed novel acquisition function on a real robot pendulum swing-up task.

4.1 Introduction

An important goal of Reinforcement Learning (RL) is to yield more autonomous robots. However, RL methods require reward functions to specify desired behavior. While such reward functions are usually hand coded, defining them for real robot tasks is challenging even for well-studied problems such as grasping. Thus, hand coding reward functions is shifting the problem of requiring an expert to design a hard-coded controller to requiring a hard-coded reward function.

Despite the variety of grasp stability measures that have been developed [Suárez et al., 2012], it has been shown that the resulting grasps are outperformed by grasps learned from kinesthetic teach-in [Balasubramanian et al., 2012]. This example shows that even experts will often design reward functions that are not effective in practice, or lead to undesired emergent behavior [Ng et al., 1999], while even non-experts can indicate good grasps. While it may be difficult to analytically design a reward function or to give demonstrations, it is often easy for an expert to rate an agent's executions of a task. Thus, a promising alternative is to use the human not as an expert in performing the task, but as an expert in evaluating task executions. Unfortunately, as already shown by Thomaz and Breazeal [2008] and by Cakmak and Thomaz [2012], humans have considerable noise in their ratings of actions. To deal with imprecise human ratings, we propose to learn a probabilistic model of the reward function and use it to guide the learning process of a RL agent. However, while learning a complicated task will typically require many rollouts, a good estimate of the reward model can often by inferred from less samples. Thus, our goal is to learn a model of the human's implicit reward function that generalizes to similar observations such that the agent can learn the task from few human-robot interactions.

To avoid specifying reward functions, Inverse RL (IRL) extracts a reward function from demonstrations [Ratliff et al., 2006, 2009, Ziebart et al., 2008]. For many tasks, such demonstrations can be obtained by kinesthetic teach-in or tele-operation. Unfortunately some skills – such as throwing a basket ball – are hard to demonstrate. Moreover, both methods require sufficient proficiency of the demonstrator which requires being good at a task and being good at performing the task

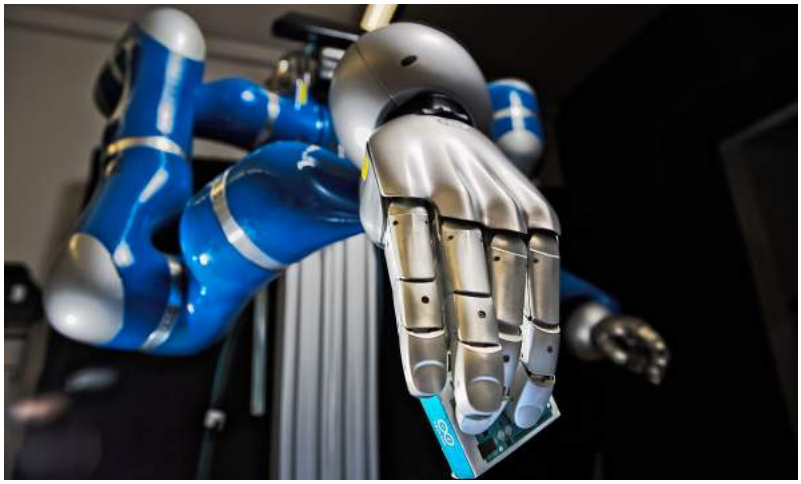


Figure 4.1.: The Robot-Grasping Task. While grasping is one of the most researched robotic tasks, finding a good reward function still proves difficult.

on a robot. Following this intuition, preference based algorithms allow the expert to rank executions and learn a controller based on these rankings [Akrour et al., 2013, Cheng et al., 2011]. The commonly used approach in ranking is to let the expert rank the previously best sample against the current sample. This approach presents an intriguing idea, as humans are better at giving relative judgments than absolute judgments. However, this approach only provides a single bit of information. The technical term describing how much information human subjects can transmit for a given input stimuli is called channel capacity [Miller, 1956]. The *channel capacity* is a measure for how many input stimuli subjects can distinguish between. In a review of several experiments, Miller [1956] concludes that humans have a general channel capacity between 1.6 and 3.9 bits for unidimensional stimuli. Adding more dimensions to the stimuli can further increase this value. Experiments have also been performed to find out whether humans can transmit more information when labeling stimuli according to predescribed categories or when being allowed to rate on a scale [Hake and Garner, 1951]. While there was no significant difference, subjects performed better when rating on a scale. Based on these insights, we design our framework such that the human expert can assign numerical values to observed demonstrations. Furthermore, numerical rewards allow the human to indicate strong preferences over demonstrations.

In this chapter, we take advantage of the Gaussian Process (GP) regression framework to represent the reward model as well as the Bayesian Optimization (BO) literature to optimize this model. We show that we can use standard BO methods to efficiently minimize the number of expert interactions, which is essential when designing methods for 'autonomous' agents. Furthermore, we introduce a novel acquisition function (AF) that outperforms traditional AFs in the proposed setting. We evaluate the proposed method on a series of simulated and real robot tasks. We also show that a reward function which is learned for one task (grasping a box) can be reused to learn a similar task (grasping a pestle).

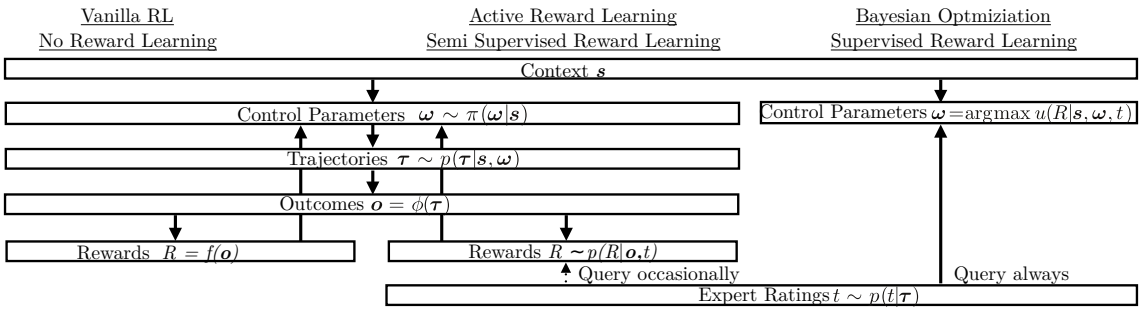


Figure 4.2.: Sketch of the elements of different possible approaches. The left column shows the 'vanilla' RL where a reward function is assumed. The middle column shows the proposed active reward learning approach, which shares the policy learning component with vanilla RL but models a probabilistic reward model that gets updated by asking for expert ratings sometimes. The right column shows the BO approach, where actions are chosen that maximize the utility function (for example one of the acquisition functions presented in 4.3 and requires an expert rating for each of the chosen actions).

4.2 Reward Learning Framework

In this chapter, we present an approach that complements the Reinforcement Learning (RL) framework by replacing the hard-coded reward function by a learned one. As most RL methods do not make assumptions about the reward function, the proposed approach does not replace existing RL methods but can be used to extend them. We base our discussion on the Policy Search (PS) framework. While there are many approaches to RL, PS is especially well suited for real robot tasks. It does not rely on exhaustive sampling of the state-action space and, as a result, many recent advances in robot learning are based on PS [Kormushev et al., 2010, Kober et al., 2008, Ng and Coates, 1998]. We consider the contextual episodic PS case with continuous contexts $\boldsymbol{s} \in \mathcal{S}$ and continuous control parameters $\boldsymbol{\omega} \in \Omega$. We use the term 'context' to describe the subset of information contained in the initial state that is necessary to describe the task at hand. Examples could be the position of an object or desired target locations. The distribution over contexts is denoted by $\mu^{\pi}(\cdot)$. Whereas the traditional RL notation uses actions $\boldsymbol{a}$, we instead write control parameters $\boldsymbol{\omega}$ to clarify that we directly optimize for the parameters of a controller which can, for example, be a Movement Primitive as discussed in Section 4.5.2. Executing a rollout with control parameters $\boldsymbol{\omega}$ in context $\boldsymbol{s}$ results in a trajectory $\boldsymbol{\tau} \sim p(\boldsymbol{\tau}|\boldsymbol{s},\boldsymbol{\omega})$, where the trajectory encodes both, the robot's internal transitions as well as relevant environment transitions (e.g. joint angles and a ball's position and speed). The agent starts with an initial control policy $\pi_0(\boldsymbol{\omega}|\boldsymbol{s})$ in iteration 0 and performs a predetermined number of rollouts. After each iteration, the agent updates its policy to maximize the expected reward

$$\mathbb{E}_{\boldsymbol{s},\boldsymbol{\omega},\boldsymbol{\tau}}[R(\boldsymbol{\tau})] = \iiint R(\boldsymbol{\tau})\, p(\boldsymbol{s},\boldsymbol{\omega},\boldsymbol{\tau})\, \mathrm{d}\boldsymbol{\tau}\, \mathrm{d}\boldsymbol{s}\, \mathrm{d}\boldsymbol{\omega}, \tag{4.1}$$

with $p(\boldsymbol{s},\boldsymbol{\omega},\boldsymbol{\tau}) = p(\boldsymbol{\tau}|\boldsymbol{s},\boldsymbol{\omega})\pi(\boldsymbol{\omega}|\boldsymbol{s})\mu^{\pi}(\boldsymbol{s})$. Parameter-based PS has been shown to be well suited for complex robot tasks [Daniel et al., 2013], as it abstracts complexity while retaining sufficient flexibility in combination with suitable movement primitive representations.

In the original RL setting, a reward function for evaluating the agent's behavior is assumed to be known. While not always explicitly stated, the reward function for real robot tasks often depends

on features $\boldsymbol{\phi}(\boldsymbol{\tau})$ of the trajectory. We refer to the result of the feature extraction from trajectories as the *outcomes* $\boldsymbol{o} = \boldsymbol{\phi}(\boldsymbol{\tau})$. We assume that the reward depends only on the features of the trajectories. Such outcomes could, for example, describe the minimal distance to goal positions or the accumulative energy consumption.

As the problem of specifying good features $\boldsymbol{\phi}(\boldsymbol{\tau})$ is beyond the scope of this chapter, we assume the features to be known, as is usually assumed when designing reward functions. The results in Section 4.6 show that the proposed method learns complicated tasks using simple features. This is possible because we base the estimation of our probabilistic reward model on kernel functions with automatic relevance determination, such that the importance of individual features can be estimated from data [Rasmussen and Williams, 2006].

As outcomes are of much lower dimensionality than trajectories, we can use outcomes to efficiently model the reward function $\hat{R}(\boldsymbol{o})$ from a training set $\mathscr{D} = \{\boldsymbol{o}_{1:n}, \boldsymbol{R}_{1:n}\}$ using regression. Using the feature representation, the term $R(\boldsymbol{\tau})$ in Eq. (4.1) is replaced by $R(\boldsymbol{o} = \boldsymbol{\phi}(\boldsymbol{\tau}))$ to become

$$\mathbb{E}_{\boldsymbol{s},\boldsymbol{\omega},\boldsymbol{\tau}}[R(\boldsymbol{o})] = \iiint R(\boldsymbol{o} = \boldsymbol{\phi}(\boldsymbol{\tau}))\, p(\boldsymbol{s}, \boldsymbol{\omega}, \boldsymbol{\tau})\, \mathrm{d}\boldsymbol{\tau}\, \mathrm{d}\boldsymbol{s}\, \mathrm{d}\boldsymbol{\omega}. \tag{4.2}$$

Obviously, it is not sufficient to build the reward function model solely on samples from the initial policy $\pi_0(\boldsymbol{\omega}|\boldsymbol{s})$, as the agent is likely to exhibit poor performance in the early stages of learning and the reward learner would not observe good samples. Therefore, we need to couple the process of learning a good policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$ with learning a good reward function $\hat{R}(\boldsymbol{o})$, such that they are developed interdependently. However, in such a coupled active learning process, we need to know which samples to query from our expert to minimize expert interaction.

Modeling a probabilistic reward model $p(R|\boldsymbol{o}, \mathscr{D})$, instead of a deterministic reward function $R(\boldsymbol{o})$, allows us to leverage the information about the certainty of our estimate to control the amount of human interaction required, as we only need to interact with the human expert if our model of the reward is not sufficiently certain. We describe the details of this process in Section 4.2.1.

When using a probabilistic model of the reward, we have to replace the reward term in Eq. (4.1) with

$$R(\boldsymbol{o}) = \mathbb{E}_{p(R|\boldsymbol{o})}[R] = \int p(R|\boldsymbol{o})\, R\, \mathrm{d}R,$$

as most PS methods work on the expectation of the reward. Using the probabilistic model of the reward function, the PS method continuously learns how to achieve high reward outcomes, while the reward model learner adapts the accuracy of its model.

4.2.1 A Probabilistic Reward Model

The proposed approach is independent of and complementary to the RL method used. The RL method aims to find the optimal policy $\pi^*(\boldsymbol{\omega}|\boldsymbol{s})$ and is agnostic as to how the rewards are computed. Similarly, Active Reward Learning (ARL) is agnostic as to how the set of available samples are created. Hence, the method for actively learning the reward can be combined with any RL method. It follows that while RL methods are inherently active, and often aim to represent a reward-related function such as the value function internally, the proposed approach solves a

Input: Information loss tolerance ϵ, improvement threshold λ, acquisition function u

Initialize π using a single Gaussian with random mean. GP with zero mean prior.

while not converged

Set sample policy: $q(\boldsymbol{\omega}|\boldsymbol{s}) = \pi_{\text{old}}(\boldsymbol{\omega}|\boldsymbol{s})$

Sample: collect samples from the sample policy
$\{\boldsymbol{s}_i \sim p(\boldsymbol{s}), \boldsymbol{\omega}_i \sim q(\boldsymbol{\omega}|\boldsymbol{s}_i), \boldsymbol{o}_i\}, i \in \{1, \ldots, N\}$

Define Bellman Error Function: $\delta(\boldsymbol{s}, \boldsymbol{\omega}) = R(\boldsymbol{o}) - V(\boldsymbol{s})$

Minimize the dual function: $[\boldsymbol{\alpha}^*, \eta^*] = \arg\min_{[\boldsymbol{\alpha},\eta]} g(\boldsymbol{\alpha}, \eta)$

Determine base line: $V(\boldsymbol{s}) = \boldsymbol{\alpha}^{T*}\boldsymbol{\phi}(\boldsymbol{s})$

Policy update:

Calculate weighting $p(\boldsymbol{s}_i, \boldsymbol{\omega}_i) \propto q(\boldsymbol{s}_i, \boldsymbol{\omega}_i) \exp\left(\frac{1}{\eta^*}\delta^*(\boldsymbol{s}_i, \boldsymbol{\omega}_i)\right)$

Estimate $\pi(\boldsymbol{\omega}|\boldsymbol{s})$ by weighted maximum likelihood estimates.

Reward model update:

while FindNominee

Nominate outcome:

$\boldsymbol{o}^+ = \arg\max u(\boldsymbol{o})$

if $(\boldsymbol{o}^+ \notin \mathscr{D}) \wedge (\sigma(\boldsymbol{o}^+)/\beta > \lambda)$

Demonstrate corresponding trajectory $\boldsymbol{\tau}^+$

Query Expert Reward R^+

$\mathscr{D} = D \cup \{\boldsymbol{o}^+, R^+\}$

else

FindNominee = false

Update reward model $p(R|\boldsymbol{o}, \mathscr{D})$

Optimize GP-hyper parameters $\boldsymbol{\theta}$

Output: Policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$, reward model $p(R|\boldsymbol{o}, \mathscr{D})$

Table 4.1.: We show the algorithmic form of active reward learning with REPS. We specify the information loss tolerance ϵ as well as an initial sampling policy and an improvement threshold λ. In each iteration, the algorithm first samples from the sampling policy, minimizes the dual function to find values for $\boldsymbol{\alpha}^{T*}$ and η^* and then computes the next policy. After each policy search iteration, the reward function learner chooses whether to demonstrate samples to the expert according to the acquisition function. The parameters $\boldsymbol{\alpha}$ and η are parameters of the dual function problem of REPS and can be optimized through standard optimization algorithms [Daniel et al., 2013].

distinct learning problem and has to be categorized separately. ARL aims to maximize its relevant information about the human's implicit reward model in a sample efficient manner. Thus, rather than annotating all samples, we aim to find a method that actively selects informative samples to query. Hence, learning the human's implicit reward function is an active learning process.

Our goal is to find a model $p(R|\boldsymbol{o}, \mathscr{D})$ that predicts the reward R given an observed outcome $\boldsymbol{o}$ and training data $\mathscr{D}$, obtained from an expert. When modeling the reward, we have to take into account that the expert can only give noisy samples of their implicit reward function and we also have to model this observation noise. Thus, we need to solve the regression problem

$$\tilde{R}(\boldsymbol{o}) = R(\boldsymbol{o}) + \eta,\ \eta \sim \mathcal{N}(0, \beta),$$

where we assume zero mean Gaussian noise. Such a regression problem can, for example, be solved with Gaussian Process (GP) regression

$$R(\boldsymbol{o}) \sim \mathcal{GP}\left(m(\boldsymbol{o}), k(\boldsymbol{o}, \boldsymbol{o}')\right),$$

where $m(\boldsymbol{o})$ is the mean function and $k(\boldsymbol{o}, \boldsymbol{o}')$ is the covariance function of the GP. For the remainder of this chapter, we use the standard squared exponential covariance function

$$k(\boldsymbol{o}, \boldsymbol{o}') = \boldsymbol{\theta}_0^2 \exp\left(-\frac{||\boldsymbol{o} - \boldsymbol{o}'||^2}{2\boldsymbol{\theta}_1^2}\right).$$

The hyper parameters $\boldsymbol{\theta} = \{\boldsymbol{\theta}_0, \boldsymbol{\theta}_1, \beta\}$ are found through optimization of the model's log likelihood [Rasmussen and Williams, 2006]. Given a training set $\mathscr{D} = \{\boldsymbol{o}_{1:n}, \boldsymbol{R}_{1:n}\}$, we can write down the covariance matrix between previously observed rewards and outcomes

$$\boldsymbol{K} = \begin{bmatrix} k(\boldsymbol{o}_1, \boldsymbol{o}_1) & \dots & k(\boldsymbol{o}_1, \boldsymbol{o}_n) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{o}_n, \boldsymbol{o}_1) & \dots & k(\boldsymbol{o}_n, \boldsymbol{o}_n) \end{bmatrix} + \beta \boldsymbol{I}.$$

Assuming a mean zero prior, the joint Gaussian probability of the training samples in $\mathscr{D}$ and the reward prediction R^+ of a new unrated observation is given by

$$\begin{bmatrix} \boldsymbol{R}_{1:n} \\ R^+ \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} \boldsymbol{K} & \hat{\boldsymbol{k}} \\ \bar{\boldsymbol{k}} & k(\boldsymbol{o}^+, \boldsymbol{o}^+) \end{bmatrix}\right),$$

with $\hat{\boldsymbol{k}} = [k(\boldsymbol{o}_1, \boldsymbol{o}^+) \dots k(\boldsymbol{o}_n, \boldsymbol{o}^+)]^T$ and $\bar{\boldsymbol{k}} = [k(\boldsymbol{o}^+, \boldsymbol{o}_1) \dots k(\boldsymbol{o}^+, \boldsymbol{o}_n)]$. The predictive posterior reward $p(R^+|\boldsymbol{o}, \mathscr{D})$ of a new outcome $\boldsymbol{o}^+$ is then given by a Gaussian

$$p(R^+|\boldsymbol{o}, \mathscr{D}) \sim \mathcal{N}\left(\mu(\boldsymbol{o}^+), \sigma^2(\boldsymbol{o}^+)\right),$$

with mean and variance

$$\mu(\boldsymbol{o}^+) = \boldsymbol{k}^T \boldsymbol{K}^{-1} \boldsymbol{R}_{1:n},$$
$$\sigma^2(\boldsymbol{o}^+) = k(\boldsymbol{o}^+, \boldsymbol{o}^+) - \boldsymbol{k}^T \boldsymbol{K}^{-1} \boldsymbol{k} + \beta,$$

by conditioning the GP on the observed outcome $\boldsymbol{o}^+$. Using the GP, we can represent both our expected reward $\mu(\boldsymbol{o}^+)$, which is provided to the policy learner, and the variance of the reward $\sigma^2(\boldsymbol{o}^+)$ which is essential to the reward learner. The reward variance $\sigma^2(\boldsymbol{o}^+)$ depends on the distance of the outcome $\boldsymbol{o}^+$ to all outcomes in the training set $\mathscr{D}$ and the observation noise β, which is a hyper parameter that we optimize for. Using the predictive variance, we can employ one of many readily available BO methods to find the maximum of the reward function.

4.3 Bayesian Optimization for Active Reward Learning

In this section, we first describe how to adapt arbitrary acquisition to the presented framework. Subsequently, we present a novel acquisition function which is optimized for active reward learning. The goal of BO is to optimize a function under uncertainty. Acquisition functions (AFs) are utility functions whose function value is maximized at locations of the input space which are likely to be maxima of the original problem. AFs usually encode an exploration-exploitation trade-off, such that they not only query samples in known high value regions but also in regions that have not been sufficiently explored before. While using GPs to model the reward function allows us to employ BO, we deviate from the general BO approach in two points.

First, as our GP models a relationship of outcomes to rewards instead of context-actions to rewards, we cannot sample arbitrary outcomes $\hat{\boldsymbol{o}}$ to improve our estimate, as the robot does not know how to produce these actions. To do so, we would require access to $p(\boldsymbol{\tau}|\boldsymbol{s},\boldsymbol{\omega})$ such that we can request the agent to perform actions that result in trajectories $\hat{\boldsymbol{\tau}}$ which yield the outcome $\hat{\boldsymbol{o}} = \boldsymbol{\phi}(\hat{\boldsymbol{\tau}})$. Additionally, we would need to guarantee that the outcomes requested by the AF are physically possible. However, this transition model is unknown and, thus, we are restricted to the set of previously observed outcomes that have been generated during the agent's learning process so far. Alternatively we could aim to learn $p(\boldsymbol{\tau}|\boldsymbol{s},\boldsymbol{\omega})$. However, learning the transition model would usually require far more samples in the considered setting. Second, we need to balance the improvement of our current estimate of the reward function and the number of queries that we request from the expert, i.e., we want to find a trade-off between finding a good reward function and learning the task with minimal human input.

While restricting the set of outcomes that can be queried to previously generated outcomes is straightforward, balancing the number of queries requires additional constraints in the case of arbitrary AFs. In the following section we will introduce several techniques to optimize this trade-off.

4.3.1 Sample Efficient Reward Model Learning

The traditional BO framework aims to find a global maximum. Sampling of the objective function can be terminated when the improvement of the function value is marginal, for example when the predicted variance around the optimum is very low. In the proposed scenario where a policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$ and a reward model $p(R|\boldsymbol{o})$ are learned simultaneously and obtaining training samples of the reward model is very expensive, the problem of deciding *when* to improve the reward model becomes crucial.

The reward model relies on the policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$ to provide outcomes in interesting, i.e., high reward regions, and the policy relies on the reward model $p(R|\boldsymbol{o})$ to guide its exploration towards such regions of interest. Thus, the reward model needs sufficient training data to approximately predict the reward function in early stages and a higher density of training points once the agents policy starts to converge to a solution. At the same time, we want to minimize the number of queries to the expert over the learning process.

In order to balance this trade-off, we propose an acquisition algorithm which, in accordance with the selected acquisition function $u(\boldsymbol{o})$, first samples the best observed sample outcome from the history of the agent's outcomes $\boldsymbol{o}_{n+1} = \arg\max_{\boldsymbol{o}} u(\boldsymbol{o}|\mathscr{D})$. In this sample based search, we

additionally include all outcomes that have been rated by the expert. If the acquisition function is maximized by a previously rated outcome, we stop and do not query any samples in this iteration.

Maximizing the AF by an already observed outcome is unique to the sample based case. As for the continuous case, a point around an observed outcome would usually have higher variance and, thus, maximize the AF. With our approach, we achieve a sparse sampling behavior that requires fewer expert interactions. If, however, the outcome that maximizes u has not yet been rated by the expert, we need to decide whether querying the outcome is beneficial.

For example, we may have already converged to a good estimate of the reward function, and new outcomes improve on the mean reward solely due to the observation noise. In this case we do not want to query the expert. Thus, we decide whether to query an outcome by thresholding the ratio of predictive variance and estimated observation noise $\sigma(\boldsymbol{o})/\sqrt{\beta} > \lambda$, where λ is a tuning parameter which allows us to trade off the accuracy of the final solution with the query frequency by adapting the available AFs to explicitly take the estimated observation noise into account. While this adaptation of the AFs introduces a new parameter to be tuned, our experiments show that the use of this technique results in fewer human interactions while maintaining a high performance and tuning of the parameter becomes straightforward.

If the robot decides to query the sample's reward value, it updates the GP and searches for the new maximum of the acquisition function. Otherwise it stops and does not update the GP further in this iteration. Additionally, we can limit the history of samples the AF can access. Limiting the history can be useful when using local search methods (e.g., policy search methods). When outcomes that were produced earlier in the learning process no longer have relevance under the current policy (the policy does not have probability mass in these regions). Adding information about these regions cannot affect the learning process. Since general AFs are not designed to evaluate the effect on the resulting policy they might propose samples that are informative for the reward model but not the policy learner.

4.4 Expected Policy Divergence

The tuning strategies (sampling threshold, sparse sampling, limiting the history of samples and not re-querying known samples) detailed above allow us to use arbitrary acquisition functions in our framework. In order to follow a more principled approach without these heuristics, we introduce a novel acquisition function which is specialized to our use case. The proposed AF, Expected Policy Divergence (EPD) is based on the insight that, as opposed to the traditional BO case, we are not mainly interested in the modeled reward function itself, but rather in the policy which the robot learns through the reward model. Thus, we should not evaluate the improvement in the reward model but instead quantify the potential change of the policy given additional information in the form of expert ratings.

Generally, starting from the current policy $\pi(\boldsymbol{\omega})$ the robot can update this policy under the current reward model $p(R|\boldsymbol{o}, \mathscr{D})$ to get a new policy $\tilde{\pi}(\boldsymbol{\omega})$. Alternatively, the robot can first update its reward model to become $p^*(R|\boldsymbol{o}, \mathscr{D}^*)$ and then update the policy to get the policy $\pi^*(\boldsymbol{\omega}|\mathscr{D}^*)$ under the updated reward model. In this setting the difference between $\pi(\boldsymbol{\omega})$ and any successive policy (e.g., $\tilde{\pi}(\boldsymbol{\omega})$ or $\pi^*(\boldsymbol{\omega}|\mathscr{D}^*)$) will often be controlled through some form of step size. However, the difference between $\tilde{\pi}(\boldsymbol{\omega})$ and $\pi^*(\boldsymbol{\omega}|\mathscr{D}^*)$ is only explained by the additional information gained by updating the reward model. In EPD, we aim to maximize the difference

between $\tilde{\pi}(\boldsymbol{\omega})$ and $\pi^*(\boldsymbol{\omega}|\mathscr{D}^*)$. Thus, if an additional sample would improve the reward model, but not affect the policy update, the human will not be queried.

Similarly to traditional AFs, we evaluate the set of possible sample locations $\boldsymbol{o} \in \mathscr{O}$ to find the most informative sample location. Given a sample $\boldsymbol{o}$, we assign a reward R according to our current reward model $R \sim p(R|\boldsymbol{o}, \mathscr{D})$ and add this pair to our data set $D^* = \mathscr{D} \cup \{(\boldsymbol{o}, R)\}$. This additional information will affect the reward model to become $p^*(R|\boldsymbol{o}, \mathscr{D}^*)$, such that the expected reward of all other outcomes is changed. Hence, we have to use the new reward model $p^*(R|\boldsymbol{o}, \mathscr{D}^*)$ to evaluate the policy update

$$\pi^*(\boldsymbol{\omega}|\mathscr{D}^*) = f(\pi(\boldsymbol{\omega}), p^*(R|\boldsymbol{o}, \mathscr{D}^*)),$$

where $f(\cdot)$ is any policy update function and $\pi(\boldsymbol{\omega})$ is the stochastic policy.

Finally, we can compare the new policy $\pi^*(\boldsymbol{\omega}|\mathscr{D}^*)$ to the baseline policy $\tilde{\pi}(\boldsymbol{\omega})$ to quantify how much querying $\boldsymbol{o}$ influenced the policy update. The KL divergence,

$$\mathrm{KL}(\pi^*(\boldsymbol{\omega}|\mathscr{D}^*) \,||\, \tilde{\pi}(\boldsymbol{\omega})) = \int \pi^*(\boldsymbol{\omega}|\mathscr{D}^*) \log \frac{\pi^*(\boldsymbol{\omega}|\mathscr{D}^*)}{\tilde{\pi}(\boldsymbol{\omega})} \mathrm{d}\boldsymbol{\omega},$$

is a natural choice to quantify the difference between two policies. When working with a deterministic policy, the KL can be computed on the policy's mean. The proposed AF maximizes the expected KL divergence between the new policy $\pi^*(\boldsymbol{\omega}|\mathscr{D}^*)$ and the baseline $\tilde{\pi}(\boldsymbol{\omega})$,

$$\begin{aligned} \mathrm{EPD}(\boldsymbol{o}) &= \mathbb{E}_{p(R|\boldsymbol{o},\mathscr{D})}[\mathrm{KL}(\pi^*(\boldsymbol{\omega}|\mathscr{D}^*) \,||\, \tilde{\pi}(\boldsymbol{\omega}))] \\ &= \iint \pi^*(\boldsymbol{\omega}|\mathscr{D}^*) \log \frac{\pi^*(\boldsymbol{\omega}|\mathscr{D}^*)}{\tilde{\pi}(\boldsymbol{\omega})} p(R|\boldsymbol{o}, \mathscr{D}) \mathrm{d}\boldsymbol{\omega} \mathrm{d}R, \end{aligned} \tag{4.3}$$

where the baseline $\tilde{\pi}(\boldsymbol{\omega})$ is defined to be the policy obtained through the policy update $\tilde{\pi}(\boldsymbol{\omega}) = f(\pi(\boldsymbol{\omega}), p(R|\boldsymbol{o}, \mathscr{D}))$. Similar to the the sampling trade-off variable λ that we introduced for the traditional AFs, EPD also needs a sampling threshold. EPD will query a sample if

$$\mathbb{E}_{p(R|\boldsymbol{o},\mathscr{D})}[\mathrm{KL}(\pi^*(\boldsymbol{\omega}|\mathscr{D}^*) \,||\, \tilde{\pi}(\boldsymbol{\omega}))] \geq \kappa, \tag{4.4}$$

where κ is user specified. However, EPD does not rely on the additional tuning strategies used for generic AFs.

4.4.1 Practical Considerations

After presenting the theoretical foundations of EPD, we will now investigate some of the implementation details.

Expected Policy.

Unfortunately, the EPD depends on the possibly nonlinear policy update and cannot generally be solved in closed form. Instead, we have to resort to sample-based methods. However, both the update of the reward model $p(R|\boldsymbol{o}, \mathscr{D})$ as well as the policy update are expensive operations and Monte-Carlo sampling should be avoided. Instead, we propose alternatives that are evaluated in the experimental section.

A promising alternative to extensive sampling is the unscented transform [Julier and Uhlmann, 1997]. The unscented transform is a method of selecting samples from an original distribution $\boldsymbol{s} \sim p(\cdot)$ such that the target distribution $\tilde{p}(\cdot)$ obtained through a nonlinear transformation $\tilde{p}(\cdot) = f\,(p(\cdot))$ can be approximated by fitting a distribution to the transformed samples $\tilde{\boldsymbol{s}} = f(\boldsymbol{s})$. The samples $\boldsymbol{s}$ are selected according to sigma points. In the presented case, the distribution $p(R|\boldsymbol{o}, \mathscr{D})$ is only one dimensional and we obtain the sigma points $[\mu_{\boldsymbol{o}}, \mu_{\boldsymbol{o}} + \sigma_{\boldsymbol{o}}, \mu_{\boldsymbol{o}} - \sigma_{\boldsymbol{o}}]$. However, the policy update only depends on the expected value of the reward model, which is independent from additional data points on the current predictive mean. Thus, we can remove $\mu_{\boldsymbol{o}}$ from the set of sigma points without introducing additional error to the estimate of $\mathbb{E}_{p(R|\boldsymbol{o},\mathscr{D})}[\pi^*(\boldsymbol{\omega}|\mathscr{D}^*)]$.

Even more sample efficient than using sigma points would be to emulate the Upper Confidence Bound AF, which considers an optimistic estimate of the reward. To follow this intuition, we always assign $R = \mu_{\boldsymbol{o}} + \sigma_{\boldsymbol{o}}$ and, thus, require only one evaluation per sample location. We also evaluate the analogous Lower Confidence Bound.

Numerical Stability.

The computation of the KL between $\pi^*(\boldsymbol{\omega}|\mathscr{D}^*)$ and $\tilde{\pi}(\boldsymbol{\omega})$ can be done either in closed form in case of Gaussian policies, or numerically in the general case. Computing the sample based approximation of the KL requires importance sampling with respect to the policy that the samples $\boldsymbol{\omega}_i$ were drawn from, i.e.,

$$\mathrm{KL}\left(\pi^*(\boldsymbol{\omega}|\mathscr{D}^*) \,||\, \tilde{\pi}(\boldsymbol{\omega})\right) \approx \frac{1}{N} \sum_{i=1}^{N} \frac{\pi^*(\boldsymbol{\omega}_i|D^*)}{\pi(\boldsymbol{\omega}_i)} \log \frac{\pi^*(\boldsymbol{\omega}_i|D^*)}{\tilde{\pi}(\boldsymbol{\omega}_i)}. \tag{4.5}$$

Unfortunately, both the sample based approximation as well as the closed form solution can become numerically unstable for high dimensional parameter spaces (e.g. greater than 20).

For high dimensional policies, we can employ a third alternative, if the policy learning method represents the policy update by assigning weights $\boldsymbol{\gamma}_i$ to the policy samples $\boldsymbol{\omega}_i$. In that case numerical instabilities can be alleviated by computing the KL directly in the one dimensional weight space.

The weights $\boldsymbol{\gamma}_i$ are determined using the policy update function. For a Gaussian policy for example, the updated policy $\tilde{\pi}(\boldsymbol{\omega})$ can then be computed through a weighted Maximum-Likelihood estimate. However, we can avoid first fitting $\tilde{\pi}(\boldsymbol{\omega})$ to $\tilde{\boldsymbol{\gamma}}$ and $\pi^*(\boldsymbol{\omega}|\mathscr{D}^*)$ to $\boldsymbol{\gamma}^*$ by computing the KL divergence of the weight vectors them self.

If we denote the weight vector that would generate $\tilde{\pi}(\boldsymbol{\omega})$ as $\tilde{\boldsymbol{\gamma}}_i = f(\boldsymbol{\omega}_i, p(R|\boldsymbol{o}, \mathscr{D}))$ and the weight vector obtained through the updated reward model $\boldsymbol{\gamma^*}_i = f(\boldsymbol{\omega}_i, p^*(R|\boldsymbol{o}, \mathscr{D}^*))$, then we can compute

$$\mathrm{KL}\left(\pi^*(\boldsymbol{\omega}|\mathscr{D}^*) \,||\, \tilde{\pi}(\boldsymbol{\omega})\right) \approx \sum_{i=1}^{N} \boldsymbol{\gamma^*}_i \log \frac{\boldsymbol{\gamma^*}_i}{\tilde{\boldsymbol{\gamma}}_i}, \tag{4.6}$$

and avoid numerical instabilities. When computing the KL in weight space, the importance sampling is implicit as $\boldsymbol{\gamma}_i \equiv 1$.

Incorporating Human Error.

In Section 4.2.1 we introduced the additionally hyper parameter β, which models the human's imprecision. We can take advantage of the estimate of the human uncertainty by not including it

in our prediction of the reward for EPD. The sample rewards for example in the Sigma point case then become $[\mu_o + (\sigma_o - \sqrt{\beta}), \mu_o - (\sigma_o - \sqrt{\beta})]$.

4.4.2 Information Flow

The general information flow of our method is as follows. We start with an uninformed, i.e., zero mean GP for $p(R|\boldsymbol{o}, \mathscr{D})$ and we initialize the PS method with an initial policy $\pi_0(\boldsymbol{\omega}|\boldsymbol{s})$. The PS learner then starts performing one iteration of rollouts. After each iteration of rollouts, rewards for the outcomes of the resulting trajectories $\boldsymbol{o} = \boldsymbol{\phi}(\boldsymbol{\tau})$ are requested from the reward learner $R \sim p(R|\boldsymbol{o}, \mathscr{D})$. The reward learner then decides whether to ask for expert ratings for any of the outcomes to update its model. In that case, the agent repeats the corresponding episode to present the outcome to the expert. Finally, the reward learner returns the mean estimate of the reward to the PS learner, which uses the rewards to update its policy and start the next iteration.

4.5 Existing Foundations

In this section we provide compact background information on components of the proposed algorithms that are necessary to give a complete picture of the proposed approach.

4.5.1 Relative Entropy Policy Search

We pair our reward learning algorithm with the recently proposed Relative Entropy Policy Search (REPS) [Peters et al., 2010]. REPS is a natural choice for a PS method to be combined with an active learning component as it has been shown to work well on real robot problems [Daniel et al., 2013] and is designed to 'stay close to the data'. Thus, previous expert queries will remain informative. A distinctive feature of Relative Entropy Policy Search (REPS), is that its successive policies vary smoothly and do not jump in the parameter space or the context space. This behavior is a beneficial characteristic to increase compliance with our proposed reward learning approach, as we are only able to predict correct rewards within observed regions of the parameter space. To constrain the change in the policy, REPS limits the Kullback-Leibler divergence between a sample distribution $q(\boldsymbol{s}, \boldsymbol{\omega})$ and the next distribution $\pi(\boldsymbol{\omega}|\boldsymbol{s})\mu^{\pi}(\boldsymbol{s})$

$$\epsilon \geq \sum_{\boldsymbol{s},\boldsymbol{\omega}} \mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{\omega}|\boldsymbol{s}) \log \frac{\mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{\omega}|\boldsymbol{s})}{q(\boldsymbol{s}, \boldsymbol{\omega})}. \tag{4.7}$$

For the complete optimization problem and its solution we refer to the original work [Peters et al., 2010].

4.5.2 Dynamic Movement Primitives

A popular use case for PS methods is to learn parameters of trajectory generators such as Dynamic Movement Primitives (DMPs) [Ijspeert et al., 2003]. The resulting desired trajectories can then be tracked by a linear feedback controller. DMPs model trajectories using an exponentially

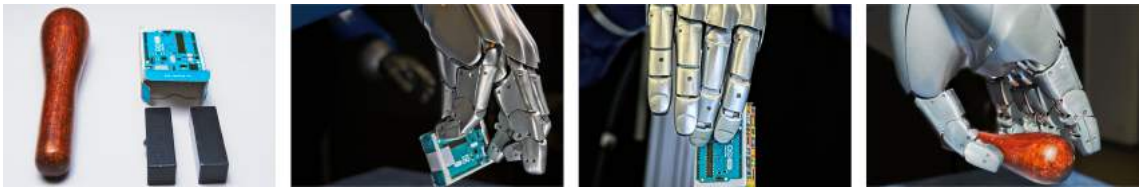


Figure 4.3.: Examples of different grasps and their categorization. Grasps count as failed if the object is either not picked up at all or if small perturbations would make the object drop. Grasps that are stable but do not keep the original orientation of the object count as OK but not successful grasps. Grasps that are both stable and keep the original orientation count as successful grasps. From Left to right: Pestle and paper box (filled with metal bars); Failed grasp, not robust against perturbations; Mediocre grasp, stable but incorrect orientation; Good grasp, stable with intended orientation.

decreasing phase function and a nonlinear forcing function. The forcing function excites a spring damper system that depends on the phase and is guaranteed to reach a desired goal position, which is one of the parameters of a DMP. The forcing function is modeled through a set of weighted basis functions $\boldsymbol{\omega}\Psi$. Using the weights $\boldsymbol{\omega}$ of the basis functions Ψ as parameters, we can learn parametrized joint trajectories, and an increasing number of basis functions results in an increased flexibility of the trajectory. We use DMPs for our simulated robot experiments.

4.5.3 Bayesian Optimization for Reinforcement Learning

An alternative approach to learning control parameters $\boldsymbol{\omega}$ using PS methods is to model $p(R|\boldsymbol{s}, \boldsymbol{\omega})$ directly and to use Bayesian Optimization (BO) instead of the PS learner. However, the approach proposed in this chapter introduces a layer of abstraction which allows us to learn a mapping of only a low-dimensional input space to the reward, as we only have to map from outcomes to reward as opposed to mapping state-action to reward. As BO methods are global methods, they are more susceptible to the curse of dimensionality than PS methods which are local methods. As a result, the mapping $p(R|\boldsymbol{s}, \boldsymbol{\omega})$ which the standard BO solution uses is considerably more difficult to learn than the mapping of the modular approach that we are proposing.

The BO approach would also require expert ratings for every sample, while the proposed approach requires only occasional human feedback.

4.5.4 Acquisition Functions

In the following we present four Acquisition Function (AF) schemes taken from Hoffman et al. [2011] that we used to optimize the model of the reward function.

Probability of Improvement

The Probability of Improvement (PI) [Kushner, 1964] in its original formulation greedily searches for the optimal value of the input parameter that maximizes the function. An adapted ver-

sion of PI balances the greedy optimization with an exploration-exploitation trade-off parameter ξ. The adapted version is given by

$$\mathrm{PI}(\boldsymbol{o}) = \boldsymbol{\Phi}\left(\frac{\mu(\boldsymbol{o}) - f(\boldsymbol{o}^*) - \xi}{\sigma(\boldsymbol{o})}\right),$$

where $\boldsymbol{o}^*$ is the best sample in the training set $\mathscr{D}$ and $\boldsymbol{\Phi}(\cdot)$ is the normal cumulative density function. The exploration-exploitation trade-off parameter ξ has to be chosen manually.

Expected Improvement

Instead of finding a point that maximizes the PI, the Expected Improvement (EI) Mockus et al. [1978], tries to find a point that maximizes the magnitude of improvement. Thus, it does not only try to improve local maxima but also considers maxima in different regions and is less greedy in the search of an optimal reward R.

$$\mathrm{EI}(\boldsymbol{o}) = (\mu(\boldsymbol{o}) - f(\boldsymbol{o}^*) - \xi)\,\boldsymbol{\Phi}(M) + \sigma(\boldsymbol{o})\rho(M),$$

if $\sigma(\boldsymbol{o}) > 0$ and zero otherwise, where $\rho(\cdot)$ is the normal probability density function. M is given by

$$M = \frac{\mu(\boldsymbol{o}) - f(\boldsymbol{o}^*) - \xi}{\sigma(\boldsymbol{o})}.$$

The EI acquisition function shares the tuning factor ξ for the exploitation-exploration trade-off with the PI, where a suggested value is $\xi = 0.01$ [Hoffman et al., 2011].

Upper Confidence Bound

The Upper Confidence Bound directly uses the mean and standard deviation of the reward function at the sample location to define the acquisition function. An adapted version of the UCB function [Srinivas et al., 2010] is given by

$$\text{GP-UCB}(\boldsymbol{o}) = \mu(\boldsymbol{o}) + \sqrt{\nu\gamma_n}\sigma(\boldsymbol{o}),$$

where recommended values $\nu = 1$ and $\gamma_n = 2\log(n^{d/2+2}\pi^2/3\delta)$ with $d = \dim(\boldsymbol{o})$ and $\delta \in (0,1)$ are given by Srinivas et al. [2010].

GP Hedge

As each of the above acquisition functions lead to a characteristic and distinct sampling behavior, it is often not clear which acquisition function should be used. Portfolio methods, such as the GP-Hedge [Hoffman et al., 2011], evaluate several acquisition functions before deciding for a sample location. Given a portfolio with J different acquisition functions, the probability of selecting acquisition function j for the sample $n+1$ is given by the softmax

$$p(j) = \frac{\exp(\eta g_n^j)}{\sum_{i=1}^{J}\exp(\eta g_n^j)},$$

where $\eta > 0$ is the temperature of the soft-max distribution. The gains vector $\boldsymbol{g}$ is initialized to zero, $g_0^{1:J} = 0$, before taking the first sample, and is then updated with the cumulative reward gained by the selected acquisition function, i.e., $g_{n+1}^j = g_n^j + \mu(\boldsymbol{o}_{n+1}^j)$, where $\boldsymbol{o}_{n+1}^j$ is the sample point nominated by acquisition function j. For all other gains the value does not change, i.e., $g_{n+1}^{i\neq j} = g_n^i$.

4.6 Experimental Evaluations

In this section we show evaluations of the proposed active reward learning approach. For all simulation experiments, unless stated otherwise, we tested each setting ten times. We first evaluate a set of traditional AFs against each other and then proceed to compare the best traditional AF to EPD. While on some of the simulated tasks we use a noisy oracle as stand-in for a human expert, all real robot experiments where performed with a human expert. The human experts were the author of the thesis as well as collaborators.

4.6.1 Five Link Reaching Task

To allow extensive evaluation of the proposed methods, we implemented a simulated reaching task. On this task an analytical reward function is readily available such that we can compare a learned reward function against the hard coded one. Additionally, this allows us to use a noisy version of the hard coded reward function as a stand-in for human experimenters and, thus, perform far more repetitions of the experiments. A planar robot consisting of five links connected by rotary joints was controlled in joint space using Dynamic Movement Primitives (DMP) [Ijspeert et al., 2003], as described in Section 4.5.2. If not stated otherwise, we used 20 basis functions per joint, resulting in a total of 100 parameters that had to be learned. The hand coded reward function was given by $R(\boldsymbol{p}_r) = 1000 - 100||\boldsymbol{p}_r - \boldsymbol{p}_g||$, where $\boldsymbol{p}_r$ was the position of the robot's end effector and $\boldsymbol{p}_g$ was the desired target position. However, we increased the difficulty of the task for the reward learning component by not supplying the outcome features in task space, but rather in joint space, i.e., the GP had to model the forward kinematics to predict the reward, making the problem both non-convex and high-dimensional (five dimensional mapping).

To allow extensive and consistent evaluation of all parameters of the presented approach, we programmed a noisy expert, which returned the reward with additive white noise (standard deviation was 20). In Fig. 4.7 we present results that compare the coded noisy expert approach to actually querying a human expert and show that the behavior is comparable. The human expert could give rewards on a vertical bar through a graphical interface.

Evaluation of Acquisition Functions

The evaluation of the available AFs given in Fig. 4.4 shows that even though there was only limited difference in the asymptotic performance, there was considerable difference in the sample efficiency. Especially the GP-UCB AF asks for many user queries. While PI had the lowest asymptotic performance, as it is the most greedy of the presented AFs, it also required the lowest number of user queries. This behavior makes it an interesting candidate when trying to minimize human interactions.

Evaluation of Uncertainty Threshold

To optimally trade off the number of queries and the agents performance we need to set the uncertainty threshold trade-off parameter $\lambda < \sigma(\boldsymbol{o})/\sqrt{\beta}$. This parameter expresses how certain we require our algorithm to be that a proposed query is not explained by the estimated observation

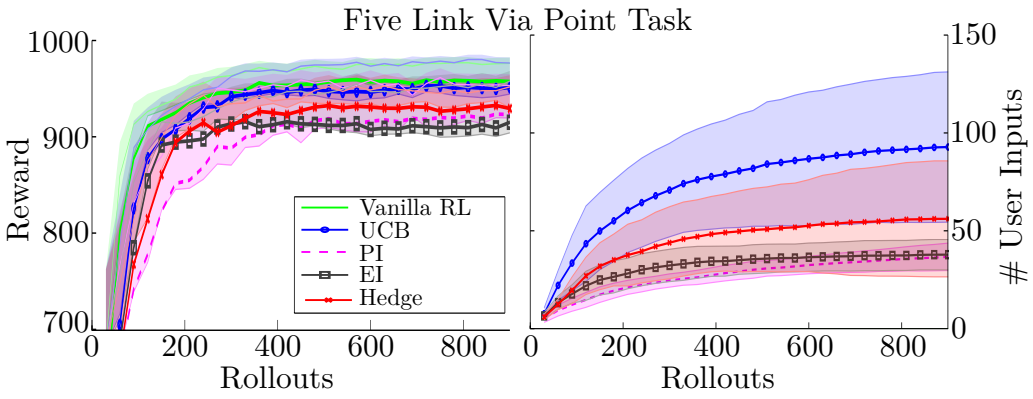


Figure 4.4.: We evaluated our approach on a programmed, but noisy expert to emulate human expert input. Vanilla RL REPS queries the programmed expert for every sample, while our approach builds a model of the reward function and only queries the expert occasionally.

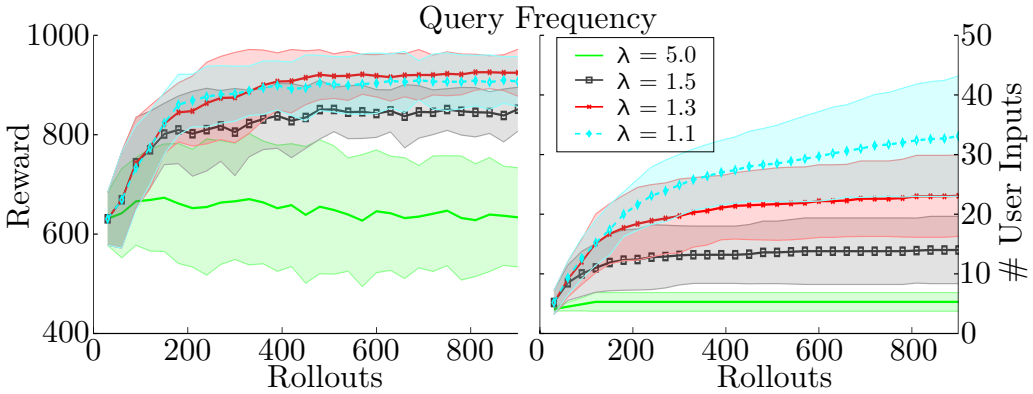


Figure 4.5.: We evaluated the impact of the threshold factor λ that influences when and how often we sample. Lower values of λ led to better converged performance but required more user interaction.

noise. If we choose λ to be greater than 1, we require our estimate to improve on the observation noise. Fig. 4.5 show the effects of adjusting λ. The performance remained stable up to $\lambda = 1.3$ and started degrading with $\lambda = 2$. Changing the order of magnitude of λ resulted in a failure to learn the task. While the effects of setting $\lambda = 1.5$ on the performance were moderate, the number of queries were reduced by about 50% when compared to $\lambda = 1.3$.

Evaluation of Sparse Sampling

In order to minimize human interaction, we stop improving the GP in each iteration if the outcome that maximizes the AF has already been queried before, instead of selecting the next best outcome according to the AF. The results in Fig. 4.6 show that sparse sampling leads to equally good asymptotic performance but requires considerably less expert interactions.

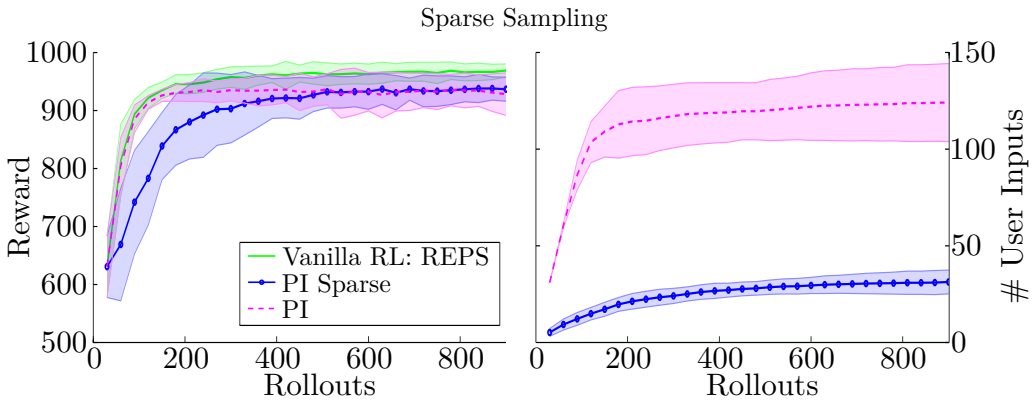


Figure 4.6.: Results of comparing our acquisition algorithm to the standard acquisition algorithm. Our algorithm collects sparse user queries and does not ask for any user queries after a PS iteration if the outcome that maximizes the AF has already been queried before.

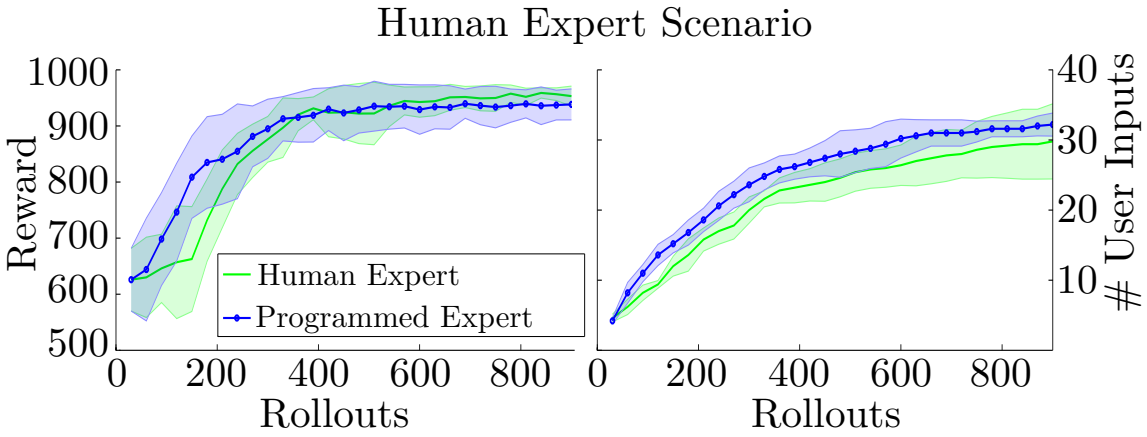


Figure 4.7.: We validated our approach of using a noisy programmed expert as substitute for a human expert on the simulated tasks. The results show that both experts yielded similar learning behavior.

Evaluation of Direct Learning

The premise of this chapter is that while BO can be used to efficiently learn the mapping of outcome to reward, it would be too sample intensive to directly learn the mapping from parameter space to outcome. We validated this premise by comparing our joint approach to directly learning the reward from the control parameters (i.e., from the basis function weights $\boldsymbol{\omega}$). The results in Fig. 4.8 show that BO did not converge to a good solution within 50 expert queries. Both methods used PI.

Comparison to Inverse Reinforcement Learning

We compared our algorithm to the Maximum Entropy IRL approach [Ziebart et al., 2008] on a via point task in two different scenarios. Trajectories generated by DMPs had to pass one or two via points, respectively (20 dimensional action space for each). For this comparison only, we relaxed our assumption that no demonstrations are available and provided the IRL approach with (imper-

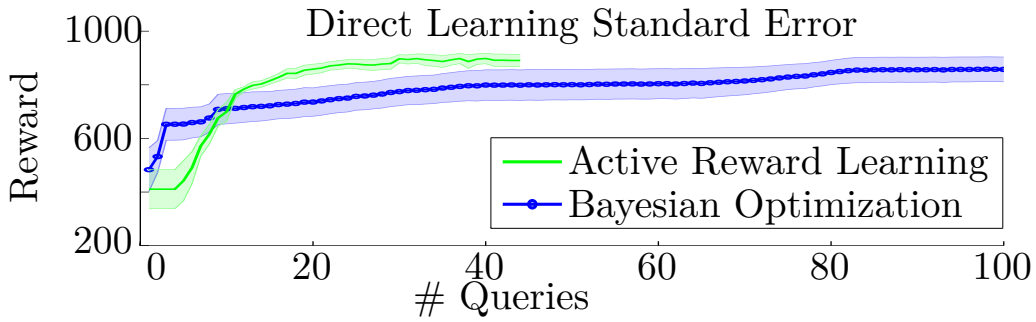


Figure 4.8.: Comparison of ARL and directly learning the reward from the control parameter $\boldsymbol{\omega}$ using BO. In this figure we plot the standard error as opposed to the standard deviation. ARL performs significantly better after 50 queries ($p < 0.05$).

		#UI = 5	#UI = 10	#UI = 20
ARL	1VP	$\mathbf{995 \pm 2.48}$	$\mathbf{999 \pm 0.25}$	$\mathbf{999 \pm 0.25}$
IRL	1VP	983 ± 3.31	977 ± 6.33	984 ± 3.04
ARL	2VPs	970 ± 13.1	$\mathbf{998 \pm 1.13}$	$\mathbf{999 \pm 0.17}$
IRL	2VPs	$\mathbf{994 \pm 1.88}$	996 ± 0.476	996 ± 0.56

Table 4.2.: Comparison of IRL and ARL. We compare the methods on two via point (VP) tasks with one and two VPs. The columns show the achieved performance (mean and standard deviation) after five, ten or 20 user interactions (UIs).

fect) demonstrations that passed close to the via point (at most 0.1 units away). In Table 4.2, we show the mean and standard deviation of the best policy reached for either five, ten or 20 user interactions (UI). In our approach, UIs are ratings while in IRL UIs are demonstrations. The results show that our approach yields competitive results while not requiring access to demonstrations.

Alternative Policy Learners

While we used REPS for most of our experiments, the proposed framework is not limited to a specific RL method. To investigate the compatibility with other methods, we compared our framework using REPS with our framework in combination with a Bayesian Policy Gradient (BPG) method [Ghavamzadeh and Engel, 2007] on a via point task with one via point (20 dimensional action space). BPG models the gradient of the policy using a GP and uses the natural gradient in the policy update. The results in Fig. 4.9 show that both methods were able to learn the task in combination with our approach.

4.6.2 Evaluation of Expected Policy Divergence

The previous results demonstrate the compatibility of the presented framework with traditional AFs and establish PI as baseline. In this section, we evaluate and compare the proposed EPD to the baseline PI. We repeat the original experiments on the simulated five link task and provide additional evaluations on a simulated pendulum swing-up task as well as a real robot pendulum swing-up task.

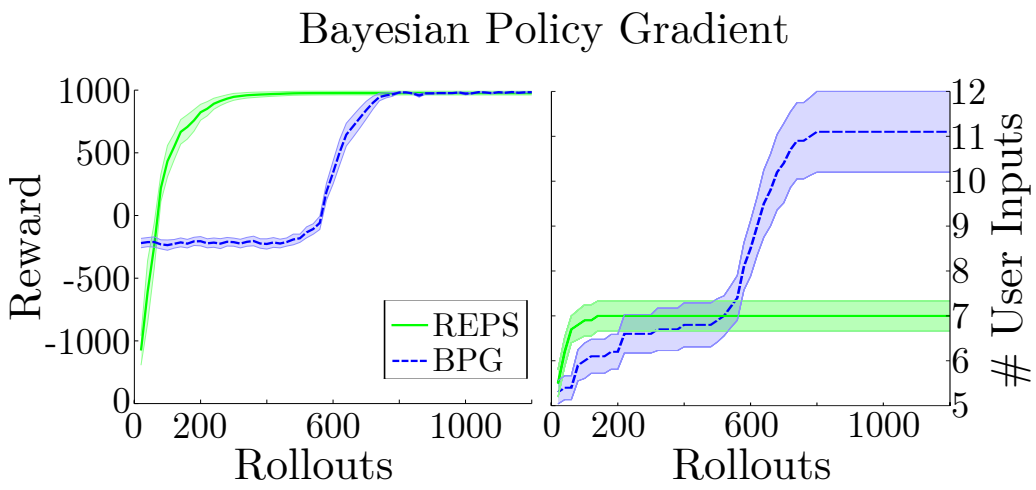


Figure 4.9.: Performance of REPS and BPG on a via point task.

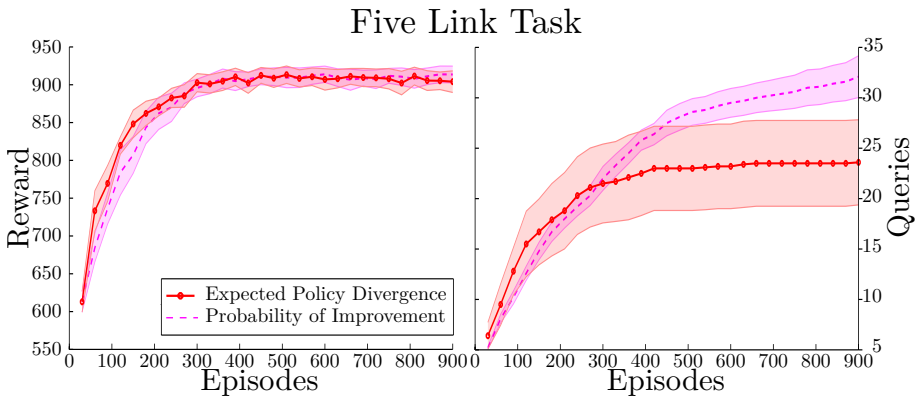


Figure 4.10.: Performance of EPD ($\kappa = 0.02$) and PI ($\lambda = 1.1$) on the five link task. EPD and PI show similar reward while EPD requires less total user inputs. Since PI does not have a notion of a policy, it keeps optimizing the reward model after the policy has converged.

Five Link Task EPD vs PI

To compare Expected Policy Divergence to PI on the five link task, we repeated the experiment reported above. We chose the tuning parameter $\lambda = 1.1$ for PI and the KL-threshold $\kappa = 0.02$ for EPD such that the algorithms achieve similar performance and we can perform a fair comparison of their querying behavior. Fig. 4.10 shows that EPD and PI perform equally well in terms of achieved reward but EPD requires less user queries. The user query plot also shows that PI keeps requesting user input after the policy has reached asymptotic performance, while EPD requests more user inputs in early iterations and stops querying after the policy has reached convergence (around episode 400). This behavior is due to the fact that EPD is not trying to optimize the reward model, but instead tries to maximize the information gain for the policy.

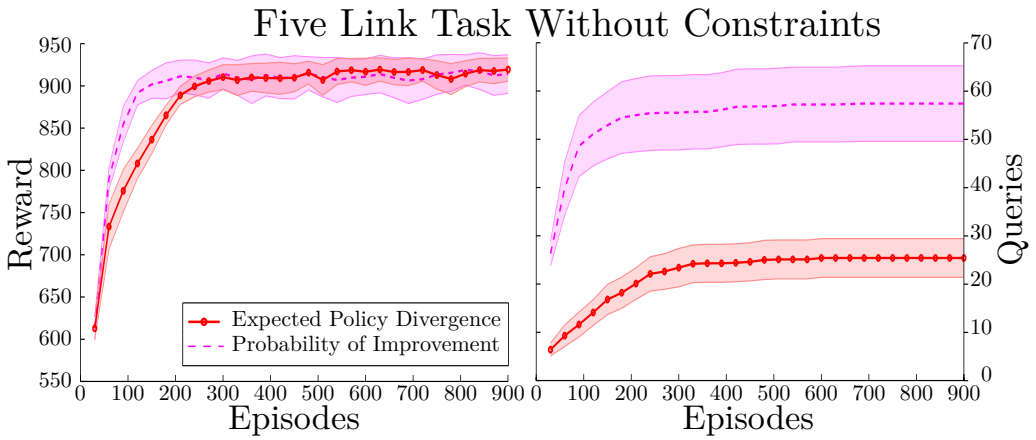


Figure 4.11.: Performance of EPD ($\kappa = 0.02$) and PI ($\lambda = 1.5$) on the five link tasks without additional constraints. Traditional AFs require additional constraints to be sample efficient in combination with an underlying learning method. EPD's behavior remains almost unaffected from lifting these constraints.

Five Link Task Without Constraints

In the previous five link task experiments, we employed the sparse sampling strategy described in Section 4.3.1, i.e., the AF would stop requesting queries if the currently best candidate had already been sampled before. Additionally, we limited the history of rollouts that can be queried, to the last iteration. Limiting the history is beneficial for all AFs that try to optimize the reward model, since they cannot differentiate between regions of the outcome space that are relevant to the policy and regions of the outcome space where the policy has negligible probability mass. However, improving the reward model in an area of the outcome space without policy probability mass cannot influence the learning process and results in inefficient query strategies. EPD, on the other hand, aims to only improve the reward model where it has influence on the policy learning process. The results in Fig. 4.11 show that lifting the constraints helps improve the performance for both EPD and PI. Both methods achieve similar asymptotic performance with PI reaching the optimum earlier. However, PI requires almost twice as many samples as before while the querying strategy of EPD remains mostly unaffected.

EPD Sampling Strategies

We evaluated the different sampling strategies discussed in Section 4.4. Using a sigma point sampling scheme (without the mean) gives the best results in terms of the required user queries while performing well. In this experiment, LCB performs similarly well while UCB requires more user queries to achieve the same result. The results show that random sampling with ten samples also performs well. However, it is computationally more expensive than the other strategies.

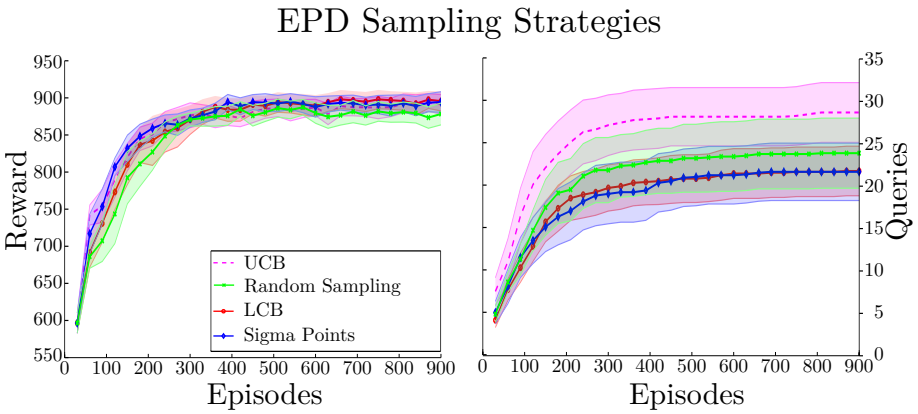


Figure 4.12.: Performance of different EPD sampling strategies. Sigma point sampling and LCB sampling require the least amount of user queries while delivering competitive performance.

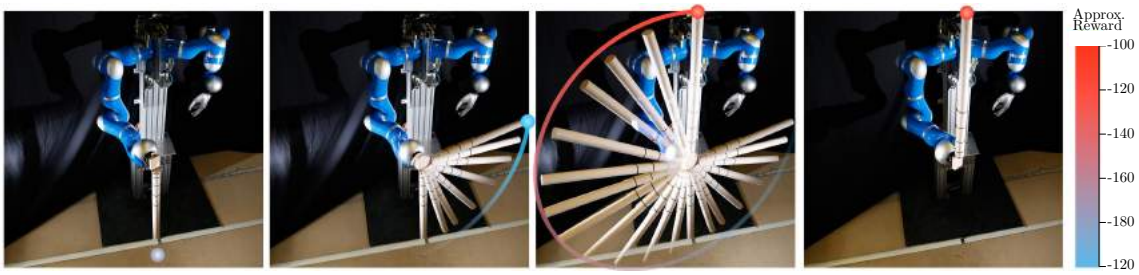


Figure 4.13.: Picture of the robot performing the swing-up task. The initially suspended pendulum is first counter-clockwise to build momentum and then swung clockwise to the upright position. The heat map gives an indication of the approximate trajectory reward if the swing-up would reach up to the indicated position.

Simulated Pendulum Swing-up

We further evaluated the relative performance of the proposed EPD to the performance of PI on a simulated pendulum swing-up task. In this simulated task, a pole of length 0.5m and mass 10kg, distributed equally along the pendulum length, is initially hanging down, mounted to a rotational joint. The joint is actuated and has a friction coefficient of 0.3. The controlled joint can exert a maximum torque of 30Nm, which is insufficient to directly swing up the pendulum to an upright position. Instead, the robot has to first swing the pendulum in the opposing direction to build up enough momentum to swing it up completely. The simulated pendulum swing-up task uses a one dimensional feature, which is the negative sum of the pole tip position to the upright position,

$$\boldsymbol{\phi}_1(\boldsymbol{\tau}) = -\sum_{t=1}^{T} 1 - \cos(\boldsymbol{\theta}_t + \pi),$$

where $\boldsymbol{\theta}$ is the angle of the controlled joint ($\boldsymbol{\theta} = 0$ in the suspended position) and $T = 250$. The programmed reward expert also used this feature as a reward signal, with additional white noise.

Fig. 4.14 shows the comparison of EPD and PI. The results show that EPD has slightly better performance and requires less than half the number of user inputs that PI requires to find the

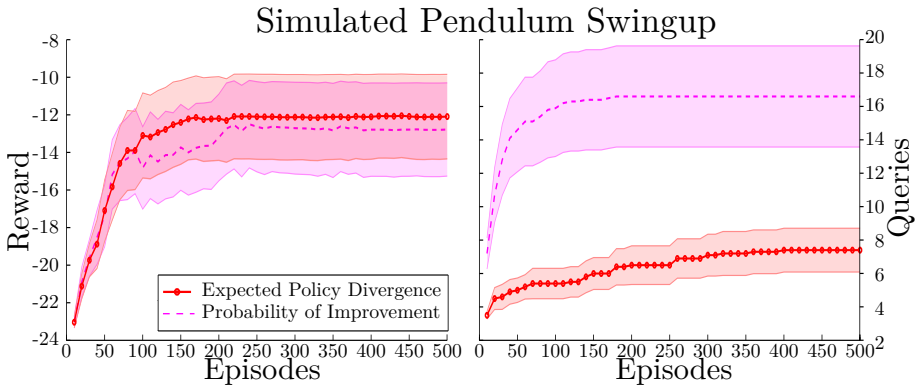


Figure 4.14.: Performance of EPD ($\kappa = 0.001$) and PI ($\lambda = 1.1$) on the simulated pendulum swing-up task. EPD achieves slightly better performance while requiring less than half the number of user queries.

solution. The large variance in the asymptotic performance is due to the fact that some solutions overshoot the goal position initially before achieving the upright position. In the real robot experiment, we add a second feature to the reward computation which helps mitigating this problem. For this experiment the KL threshold for EPD was $\kappa = 0.001$ and the sampling threshold for PI was $\lambda = 1.1$. In each iteration, ten new rollouts were performed. As the pendulum swing-up task is multi-modal, it was solved using HiREPS [Daniel et al., 2013], starting with 100 options. Using HiREPS allowed the agent to learn both swing-ups, clockwise and counter-clockwise.

Real Robot Pendulum Swing-up

After comparing EPD and PI on the simulated pendulum swing-up task, we evaluated EPD on a real robot pendulum swing-up task. To learn the reward model we provided the same feature as in the simulated task, with an additional feature representing the sum of the kinetic and potential energy at each time point

$$\boldsymbol{\phi}_2(\boldsymbol{\tau}) = \sum_{t=1}^{T} \frac{1}{2} m v_t^2 + m\,(l - cos(\boldsymbol{\theta}_t + \pi))\,.$$

such that the joined feature vector was $\boldsymbol{\phi}(\boldsymbol{\tau}) = [\boldsymbol{\phi}_1(\boldsymbol{\tau}), \boldsymbol{\phi}_2(\boldsymbol{\tau})]^T$. Adding the energy as feature helps to differentiate trajectories that exhibit the desired behavior of performing a pre-swing from those that try to directly swing up the pendulum in the early stages of learning. While it is difficult to provide a hard-coded target value for this feature, it is easy to learn the desired value from human ratings. The reported reward for the real robot experiment is computed only from the $\boldsymbol{\phi}_1$, analogously to the simulated swing-up task. In the real robot experiments $T = 10e4$s, the length of the pole was $l = 1$m, with a mass of $m = 10kg$.

Fig. 4.15 shows the results of three trials on the real robot task. The results show that the robot learned to swing up the pendulum in all three trials. In the last trial, the robot learned a double pre-swing before swinging up the pendulum. For the real robot experiment the KL threshold for EPD was $\kappa = 0.005$. As in simulation, ten new rollouts were performed per iteration.

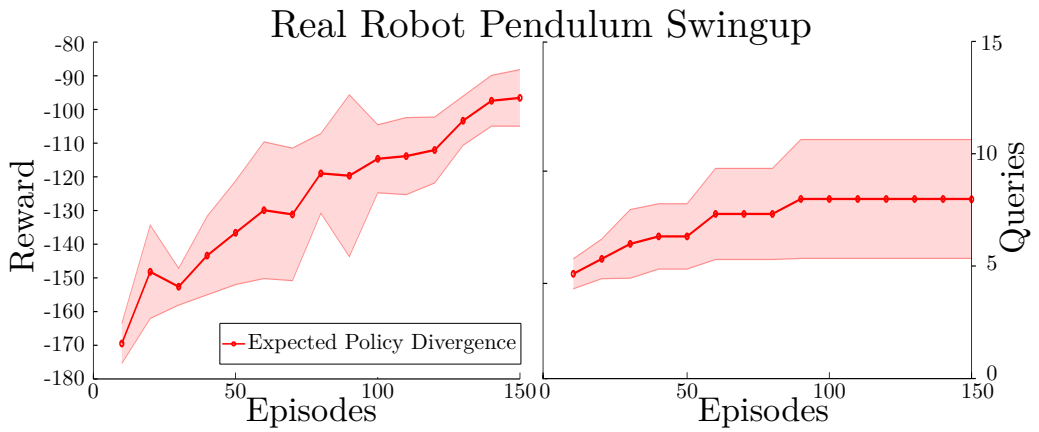


Figure 4.15.: Performance of Expected Policy Divergence on the real robot pendulum swing-up task. EPD learns to successfully swing up the pendulum on the real robot with an average of about eight user queries. Results averaged over three trials.

4.6.3 Robot Grasping

We used the results from Section 4.6.1 to set the parameters for the real robot experiment (we use PI and set $\lambda = 3$). We learned the policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$ with 15 samples per iteration for a total of 10 iterations and we repeated the experiment three times. The control parameters of the policy were the 15 joints of the five finger DLR hand, which is mounted to a 7 DOFs KUKA lightweight arm as shown in Fig. 4.1. We considered the task of blind grasping as described by Dang and Allen [2012], where no object information was available and we did not have visual feedback, i.e., we did not have information about the contact points. Instead, we calculated the forces in the finger tips through the joint torques and and the hand kinematics. We used the finger tip force magnitudes as outcome features which were used by the GP to model the reward function.

Evaluating the real robot experiments presented the problem of a success metric. As we did not have a 'correct' reward function that we could evaluate the learned reward function against, we resorted to introducing three label categories which were only used for the evaluation after finishing the experiments. The scheme presented in Fig. 4.3 labels grasps as failures with a reward of -1, if the object was not lifted at all or slipped when slightly perturbed. Grasps that were stable but did not keep the intended orientation were given a reward of 0. Finally, grasps that lifted the object and kept the orientation of the object were assigned a reward of 1. These labels and reward values were not used during the learning of either the policy or the reward model but only used to visualize the robot's learning progress. During the learning phase, the human expert assigned grasp ratings in the range of ± 1000.

Learn to Grasp Unknown Object

The object to be grasped was a cardboard box filled with metal weights such that the robot cannot grasp the object with very unstable grasps or by deforming the paper box. The box, shown in Fig. 4.3, was of size 7.5cm x 5.5cm x 2cm and filled with two metal bars with a combined weight of 350g. The results of three trials presented in Fig. 4.16 show that the robot learned to perform a

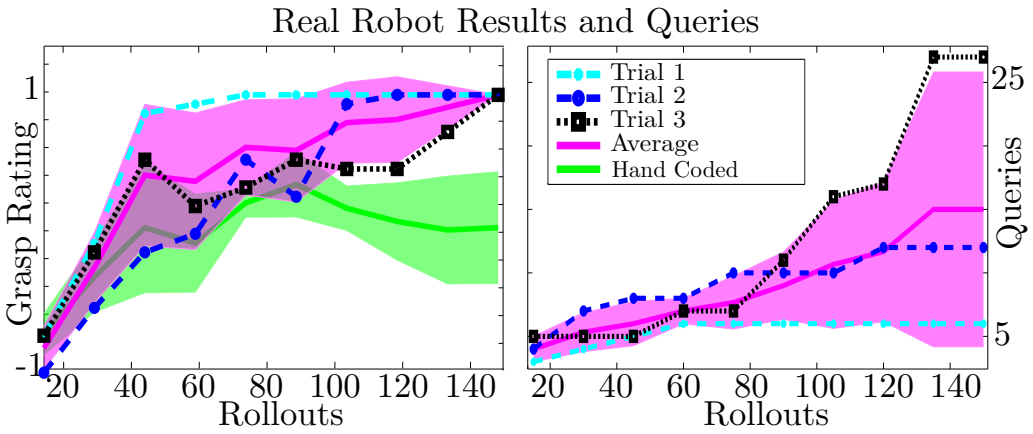


Figure 4.16.: The real robot successfully learned good grasps in all of the three trials. The grasp rating scheme is described in Fig. 4.3. In the last trial, one joint failed, represented by a spike in expert queries. The robot successfully adapted the reward function to recover from the hardware failure. We also show the average performance of a naive hand coded reward function.

successful grasp of the object in all three trials, while only requesting six queries in the first and twelve queries in the second trial. In the last trial, the robot's performance first increased quickly but dropped after 80 episodes (or rollouts), coinciding with a sudden increase of user queries, such that the final number of queries in the last trials was 27. The reason for this unusual behavior was a malfunction of the distal joint of the thumb, which rendered the grasping scheme dysfunctional. As the learner could not reproduce outcomes that led to good rewards, it resorted to finding a different grasp strategy. At the same time, since the GP was presented with new outcome samples in previously unobserved regions of the outcome space, it requested new user queries to model the reward function in the new region of interest.

We compared our learned reward function to a (naive) hand coded reward function based on the same features. The hand coded reward function aimed to reach a total force magnitude over all fingers. Using the programmed reward, the robot was able to reliably pick up the object after the first two trials and in ten out of 15 grasps at the end of the third trial (We expect the robot would have also learned to pick it up reliably in the last trial with more iterations). However, the robot did not pick up the object in a way that kept the original orientation of the object. Encoding such behavior through only force features by hand is challenging. The performance curve of the hand coded reward function shows a slight dip, which is possible as we are not plotting the internal reward but the reward assigned according to the grading scheme introduced in Fig. 4.3.

Transfer Learned Reward Function to New Object

As we based our reward model only on the finger tip forces, it is modular and can be used for other similar objects. To test these generalization capabilities, we started a new trial with a different object (a pestle as shown in Fig. 4.3) and initialized the GP with the training data from the first trial of the previous task. For this experiment we only used ratings from the previous trial, and did not ask for additional expert ratings. The pestle that we used for this task was similar in dimensions but different in shape, such that the agent had to learn different joint configurations to achieve similar finger forces that optimized the reward function. The pestle had a length of 18cm,

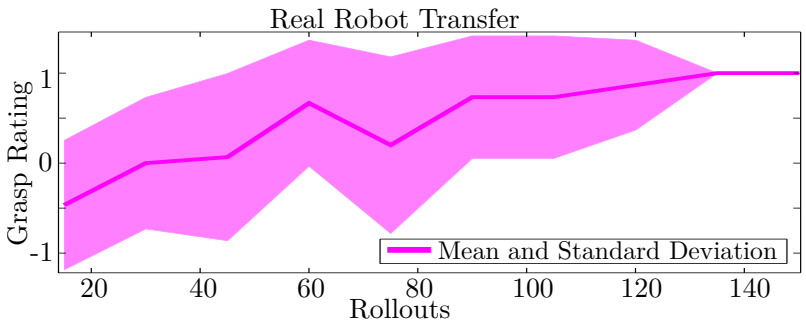


Figure 4.17.: Performance of one trial on the real robot system. The robot uses the reward function learned in trial one of the original task to learn to grasp a new, unknown object (a wooden pestle as shown in Fig. 4.3). For this trial, we did not allow any expert queries. The robot successfully learned to grasp the object.

with a 1.5cm radius at the thin end and a 2.25cm radius at the thick end. The results in Fig. 4.17 show that the agent was able to learn to reliably perform a robust grasp on the pestle, reusing the reward function learned for the box.

4.7 Related Work

The term reward shaping describes efforts to adapt reward to increase learning speed while not affecting the policy [Ng et al., 1999, Devlin and Kudenko, 2012, Bratman et al., 2012, Singh et al., 2010] or to accelerate learning of a new task [Konidaris and Barto, 2006]. Aside from reward shaping, Dorigo and Colombetti [1994] have used the term robot shaping to describe efforts of teaching different tasks to a robotic agent. Derived from this terminology the TAMER framework [Knox and Stone, 2009] uses the term interactive shaping. There, the agent receives reinforcements from a human, guiding the learning process of the agent without active component. The Advise framework [Griffith et al., 2013] uses the term shaping to describe the process of modeling an oracle and actively requesting binary labels for an agent's action (good or bad). This approach is tailored to discrete settings and the feedback frequency is pre-defined.

In preference learning algorithms, the expert is usually requested to rank the current best against a new execution and the reward function is inferred from these rankings [Akrour et al., 2013, Cheng et al., 2011, Wilson et al., 2012]. The method of Akrour et al. [2011] can also deal with noisy rankings. Chu and Ghahramani [2005] introduced the use of GPs for preference rankings. However, the expert is limited to transmit one bit of information and cannot express strong preferences. Our contribution is to show how a rating based approach with explicit noise model can be used in real robot continuous state-action space problems taking advantage of the stronger guidance through strong preferences (large differences in assigned rewards).

Schoenauer et al. [2014] show how a preference based learning algorithm can be used in the cart-pole balancing task with continuous states and discrete actions. However, they require to first build a model of the system dynamics through extensive sampling. In our approach the core idea is that the control of the system and the reward function should be learned simultaneously, which minimizes the total number of rollouts required on the system. If, on the other hand, the number of rollouts on the system to build a model is of no importance, different approaches to learning the reward function are better suited.

The problem of expert noise has also been addressed in Bayesian RL and IRL. In Bayesian RL, Engel et al. [2005] have proposed to model the value function through a GP and Ghavamzadeh and Engel [2007] have proposed to model the policy gradient through a GP. Especially the policy gradient method is also well suited to work in combination with our approached framework.

Kroemer et al. [2010] have used Gaussian Process Regression with a UCB policy to directly predict where to best grasp an object. If demonstrations are available, IRL is a viable alternative. Ziebart et al. [2008] have relaxed the assumptions on the optimality of demonstrations such that a reward function can be extracted from noisy demonstrations.

In a combination of IRL and preference learning, Jain et al. [2013] have proposed the iterative improvement of trajectories. In their approach, the expert can choose to rank trajectories or to demonstrate a new trajectory that does not have to be optimal but only to improve on the current trajectory. This approach cannot directly be used on learning tasks such as grasping, as a forward model is required. Alternatively, Lopes et al. [2009] propose a framework where IRL is combined with active learning such that the agent can decide when and where to ask for demonstrations.

4.8 Conclusion & Future Work

We presented a general framework for actively learning the reward function from human experts while simultaneously learning the agent's policy. We propose a novel acquisition function which evaluates the change of the policy induced through altering the reward model. Our experiments showed that the learned reward function can outperform hand-coded reward functions, generalizes to similar tasks and that the approach is robust to sudden changes in the environment, for example a mechanical failure. The results also show that the proposed acquisition function, EPD, outperforms traditional AFs and works well on simulated and real robot experiments. In the future we plan to investigate how different kernel functions affect the learning process and study how well non-expert users can 'program' reward functions with the presented framework.

5 Conclusion

In this thesis, we have considered several important challenges in the field of reinforcement learning. We have investigated the question of how to learn hierarchical policies that can learn multiple solutions, which we called sub-policies, to one task concurrently in one learning process. We have also investigated the question of how to sequence such sub-policies to solve more complex tasks in both the finite horizon setting and the infinite horizon setting. Further, we examined the problem of integrating the semi markov decision process setting with continuous state-action reinforcement learning algorithms. Here we put a focus on policy search methods, wich have shown to work well for many real world problems. Finally, we explored possibilities of making reinforcement learning a more autonomous process by learning a teacher's implicit reward function instead of requiring a hand-coded reward function. In the following we will discuss the results of this thesis in detail.

5.1 Contributions of this Thesis

In this thesis, we have presented several steps towards the goal of more autonomous robots. In the first chapter, we have presented approaches for learning a set of versatile skills which can be combined to solve complicated problems. The results of our real robot experiments show that the proposed approach works well on a range of robotic experiments. The tetherball experiments show that the proposed method can learn multiple solutions to a single problem during one training run using the episodic formulation of the proposed algorithm. This result allows robots to learn more efficiently because they can avoid modeling multiple solutions with one policy and it makes these robots more robust because back-up solutions are available. The results on the hockey playing robot show that the proposed approach can be extended to finite horizon tasks, which further increases the scope of tasks we can solve using our learning method. Finally, the results on the ball balancing task show that the infinite horizon formulation of the proposed algorithm is equally able to solve real world tasks. Together, these three formulations, namely the episodic, finite horizon and infinite horizon formulation, cover all settings a robot might encounter in the real world.

In the second part of this thesis, we looked at temporally correlated macro-actions, or options, which generate related actions over multiple time steps. Many real robot experiments rely on such correlated actions in the form of movement primitives, which generate desired continuous action trajectories based on a set of learned parameters. However, this approach usually requires domain knowledge, for example in the form of hyper-parameters describing the smoothness of desired trajectories or their duration. Furthermore, these MPs often generate desired open loop trajectories which do not incorporate feedback. To close the gap between existing work on discrete option discovery algorithms and our need for continuous versions thereof, we proposed a framework which is based on a graphical model where option indices and termination events are treated as latent variables. Based on this model, we could apply inference techniques to reconstruct a hierarchical policy which consists of an activation policy, termination policies and sub-policies. The results show that this hierarchical policy was able to generate temporally correlated actions

and was able to outperform state-of-the-art option discovery methods even on discrete tasks. Furthermore, the results show that the proposed option discovery framework is not constrained to discrete state-action spaces, but also works well in continuous state-action spaces.

Finally, in the last part of this thesis, we tackled the problem of designing reward functions, which is often difficult even for experts. We proposed an approach where the robot learns a model of a teacher's implicit reward function during the RL process. The reward function is modeled using a Gaussian process, which allows the robot to also model the uncertainty about the reward function. Using this probabilistic model, we could use existing optimization methods to select query points which would allow the teacher to give informative feedback. The results show that this approach worked well on a difficult grasping task with very limited amount of human advice. Furthermore, the reward function learned on the task of grasping a box transferred to a different object without additional human feedback.

To reduce the amount of human interaction even further when learning a new reward function, we proposed a novel acquisition function, which takes the RL pipeline into account. Based on the fact that the predictive distribution over reward functions will influence the policy updates, the proposed acquisition function can propose query points which have the largest influence on the policy update. The comparative results show that this novel acquisition function requires even less human-robot interactions compared to employing existing acquisition functions and real robot experiments show that this new acquisition function works well on real world tasks.

In conclusion, we have proposed approaches for three highly interwoven challenges in robot learning. Learning hierarchical policies allows us to tackle more challenging problems by combining and sequencing versatile skills. Learning temporally correlated sub-policies allows us to learn these skills from scratch and frees us of the limitations of motor primitives. Finally, to learn these hierarchical, temporally correlated skills, we need good reward functions which cannot be readily supplied by non-experts. To ultimately make these advances useful, we narrowed the gap between RL and end users by providing an intuitive and highly effective way of guiding robot learning processes.

5.2 Open Problems

While this thesis has proposed several steps towards more autonomous robots, many open challenges remain on the way to this goal. Here, we want to discuss some of the key open problems pertinent especially to this thesis.

5.2.1 Integration of Planning and RL

While RL approaches have proven their suitability in learning new tasks, it does not seem efficient to learn one integrated policy to solve all tasks. Instead, a more efficient solution might use planning algorithms to select between tasks and invoke RL algorithms to learn in new situations. To motivate such an integrated approach, we can consider the example task of getting coffee. The planner can break up getting coffee into many sub tasks which constitute the overall coffee task. Some of these sub-tasks would be going to the kitchen, opening a cabinet, grasping the coffee jar, opening the coffee jar, etc., These general steps will be similar for a wide range of users, and if the apartment layout is known and the kitchen uses standard cabinets, the planner can compute a path

to the kitchen and select a standard cabinet opening skill. However, many users will have different coffee jars, cups and coffee machines. Thus, some of the sub-tasks like grasping a specific coffee jar might need to be learned in an unknown kitchen.

To integrate learning and planning, two key elements have to be available. First, the planner has to decide when to invoke a learning routine and second, a reward function for the training routine has to be generated. One solution for deciding when to switch to the RL mode might be to learn a model of outcomes for each stored policy. This model would not predict the dynamics of the task but only the final configuration given a parametrized context. This parametrized context could, for example, be the shape of the coffee jar. Because the model would aim to only predict the outcomes of applying a policy, it would be much easier to learn than a full dynamics model. Given such a model, the planner could choose the policy which has the highest likelihood of ending in a state which is compatible with the requirements of the next state. If no stored policy meets these requirements, a learning routine is invoked. Once the planner switches to the RL mode, a reward function is required to guide the learning process. For general tasks such as grasping, stored reward functions such as presented in Chapter 4 might be applicable. However, the planner is also assumed to know the required configuration for the next sub-task. Thus, this configuration could be used as a target state to base an automatically designed reward function on. An important research question is what features such automated reward functions require to be effective in guiding the learning process.

Finally, learning individual sub-tasks should be boosted by available solutions from all other robots, i.e., use transfer of existing knowledge. This approach would also help not wasting individual experiments. If one robot comes up with a solution to a new problem it would be added to a networked library. This proposed approach is still dependent on a known task structure, i.e., the planner requires knowledge of the individual sub-tasks that make up an overall task. This knowledge is instrumental in guiding the RL learning process towards the intermediate goals. Learning and iteratively refining such task structures would also be an interesting research question. One could easily imagine using algorithms such as HiREPS which already solves tasks using learned sub-policies to define the boundaries between sub-tasks.

5.2.2 Data Driven Generation of New Options

In Chapters 2 and 3 we proposed algorithms which are able to learn multiple options concurrently. One limitation of the proposed methods is the necessity of defining the number of available options a priori. A more general solution would start with one single option and automatically generate additional options during the learning process. Generating new options could be achieved by simply using concepts known from Gaussian mixture models, such as the Bayesian Information Criterion (BIC) or the Akaike Information Criterion. The BIC could be used to find the best number of clusters to explain the weighted samples that are returned by the HiREPS algorithm. A more elegant solution could be based on Dirichlet Processes (DPs), which represent a distribution over mixture models. In this case, the number of options would be sampled directly from the DP.

5.2.3 Exploration of the State Space

The RL algorithms presented in this thesis start from an initial policy with a large variance and continuously shrink this variance during the learning process to arrive at the optimal solution.

One of the drawbacks of this design becomes apparent in cases where important regions of the state space are difficult to reach. In such cases, the algorithm will first have to sufficiently optimize the policy to reach these states by reducing the policy's variance. However, when using, for example, a single Gaussian to represent the policy, this variance can only be reduced globally. Thus, there will be insufficient exploration in these new states. One solution to ensure sufficient exploration also in novel states could be the use of other policy representations such as Gaussian Processes [van Hoof et al., 2015], which may represent a state-dependent variance. However, using more complex policy representations often also results in a large amount of hyper-parameters and sometimes less stable learning behavior. Alternatively, we could build on the idea presented in the previous paragraph and instantiate additional options for regions of the state space which are insufficiently explored. In this case, a simple criterion like the BIC alone would be insufficient to guarantee exploration. Instead, one might want to compare the current policy to the initial policy to verify that the state space has actually been explored. Alternatively, one could imagine a combination of an entropy maximizing bound such as proposed by Abdolmaleki et al. [2015] could be combined with, for example, the BIC to find mixture models which simultaneously maximize the reward and the policy's entropy.

5.2.4 Automatic Feature Detection in REPS

In Section 2.8 of this thesis, we discussed feature representations of the state that have been used in this thesis. Possibilities include polynomial expansions of the state space such as a squared feature expansion, features based on the combination of basis functions such as the Fourier features, or kernel based features. All but the kernel based features are parametric features, where the practitioner has to decide on the level of complexity required to solve the task at hand. For a polynomial representation, for example, this complexity would define the order of expansion while for basis function features it would describe the number of basis functions per dimension. Unfortunately, choosing an adequate feature representation as well as a reasonable complexity is often a time intensive and unpredictable process when attempting to solve novel tasks. While kernel based features such as used by van Hoof et al. [2015] can be used instead, these representations usually have both a high computational as well as a high memory complexity. In cases where the intrinsic dimensionality of the problem at hand is low, these kernel based approaches can be used successfully by restricting the overall amount of data points used. This, for example, is the case in van Hoof et al. [2015], where the state space is a camera image of high dimensionality, but the task is only of intrinsic dimensionality two, i.e., the pendulum position and velocity.

Instead, an alternative approach for higher dimensional problems might attempt to automatically select the type of feature or, at least, the required complexity of the feature. When using REPS type learning methods, one could take advantage of the fact that these feature representations do not have to be consistent across learning iterations. Thus, in the beginning of learning, a lower complexity might be sufficient and this complexity could be increased in later iterations of the learning process. The question of how to select the appropriate complexity might be tackled by looking at the temporal difference (TD) error.

In REPS, we constrain our solutions such that under the resulting policy, the expected features of the next state are explained by applying the policy and the transition dynamics to the steady state distribution, i.e.,

$$\int \boldsymbol{\phi}(\boldsymbol{s}')\mu^{\pi}(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}' = \iiint \mathscr{P}^{\boldsymbol{\omega}}_{\boldsymbol{s}\boldsymbol{s}'}\mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{\omega}|\boldsymbol{s})\boldsymbol{\phi}(\boldsymbol{s}')\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{s}'\,\mathrm{d}\boldsymbol{\omega}.$$

When formulating the Lagrangian, this equation is associated with the Lagrangian multiplier $\boldsymbol{\theta}$. In the dual formulation, this $\boldsymbol{\theta}$ appears in the term

$$r(\boldsymbol{s}) + \mathbb{E}_{\boldsymbol{s}'}[\boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{s}')|\boldsymbol{s},\boldsymbol{\omega}] - \boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{s}),$$

which we recognize as the TD error where $\boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{s})$ represents the value function $V^{\pi}(\boldsymbol{s})$ under the proposed policy. Given the TD error, a solution to finding the best feature complexity might be based on minimizing this error over all states. However, the quality of the value function approximation will also impact the quality of the resulting policy update, i.e., the computation of optimal thetas $\boldsymbol{\theta}^*$ depends on the current η and vice versa. Thus, instead of trying to optimize the TD error in isolation, a more effective alternative solution would aim to minimize the function value of the dual. This solution, however, would require solving the full dual problem for each evaluation. A comparative evaluation would be required to evaluate the effective difference of both approaches.

Finally, in both approaches, cross validation should be employed to avoid overfitting to individual data points. Since cross validation can also be computationally expensive, empirical studies are required to evaluate which degree of cross validation would be required.

While determining the optimal feature complexity will likely have considerable computational requirements in both approaches, one could bootstrap the solution by using previous solutions. Additionally, the optimal feature complexity may not have to be computed in every single iteration if the computational overhead would become excessive.

5.2.5 Individual Value Functions Per Option

In this thesis, we have shown methods to learn multiple sub-policies to to solve tasks which can be broken down into sub-tasks more efficiently. However, the efficiency of the proposed approach might be further improved by not only allowing for individual sub-policies, but also a decomposition of the value function into option specific sub-value functions. In HiREPS the value function appears in the policy update equation as a product of the features $\boldsymbol{\phi}$ and the Lagrangian parameters $\boldsymbol{\theta}$, i.e., the Value function in HiREPS is incorporated through the feature matching constraint. To allow for individual value functions per option, this constraint would have to be made option specific. In that case, the overall value function should become a super-position of the option specific value functions. One of the main advantages of such a representation would be a simplified representation of the value function. Currently, for problems such as the pendulum swing-up, complex feature expansions such as the Fourier features have to be used to represent a sufficiently accurate value function. Using less expressive features will result in a failure to learn the optimal policy. At the same time, because of the modular nature of the options, the sub-policies can work efficiently on much simpler feature representations such as squared expansions of the state space. A decomposed value function would, hopefully, also be able to deliver good results with such simpler representations while increasing the overall data efficiency of the algorithm.

5.2.6 Individual Policy Representations per Option

The results presented in Chapter 2 of this thesis show how HiREPS can be used to learn multiple solutions to one task and how HiREPS can be used to sequence blocks of a strategy. However, it might often also be desirable to select between skills with different parametrizations. In the proposed framework, skills with individual parametrizations cannot be integrated naively. HiREPS is based on the premise of latent option indices to allow improvement of all options from all skill executions. This approach, however, is not compatible with individual skill parametrizations, where the same parameter vector can describe very different behavior. In that case, the same parameter value can have different meanings for the different parametrizations and, thus, cannot be shared between options. Furthermore, different skills might not have the same dimensionality of the parameter space. Thus, to allow for individual skill parametrizations, an additional layer of hierarchy would have to be introduced into HiREPS. In this extended version, a skill selection policy would first have to select between parametrizations, before the activation policy selects a specific sub-policy for that skill. One interesting research question would be to use such a setup for a version of transfer learning. One could imagine the skill selection policy to choose the number of basis functions for a movement primitive (MP). In that case, parameters learned for a MP with ten basis functions would still retain some of their meaning for a MP with eleven or twelve basis functions. However, the additional basis functions might add flexibility required to solve a more complex version of a similar task. Thus, the open question would be how information could be shared between different but similar versions of the same skill to either boot-strap creation of new skills or to accelerate learning of all skills at the same time.

5.2.7 Biased Exploration

In Chapter 3 of this thesis, we showed how temporal correlations can be learned from data. Having temporally correlated actions allows us to learn more efficiently and can be especially useful to transfer skills to new settings. In the proposed framework, the agent samples an option once and, subsequently, keeps sampling actions from the according sub-policy until a termination event occurs. However, learning of complex behaviors, especially in noisy systems, could be accelerated further if the agent could decide to test the effect of an action for multiple time steps. For example, we can imagine a pendulum swing-up scenario with limited torques and a very fine time discretization, which is often the case on real robots where control frequencies are commonly exceeding 1 kHz. In that case, sampling torques from a zero mean normal distribution may lead to sensor values which are dominated by noise and it will be hard for the agent to discern the effect of its actions. Instead, if the agent could first sample an exploration bias, e.g., 10 Nm, and, subsequently, sample torques around that bias point until a termination event occurs, it would be much easier to discern the effect of positive torque values on the system. This bias would also work well with any state-dependent policy and would only move the mean of the policy but not remove or negate any state-dependent effects. In the proposed framework, this exploration bias could be sampled directly after the option index and would stay active until a termination event occurs. The exploration bias could either be an observed variable or, alternatively, an additional latent variable. Since the exploration bias would be a continuous variable for continuous action spaces, a latent variable representation would significantly complicate the inference process. However, EM techniques similar to those used in Chapter 3 can be applied.

5.2.8 Infinite Horizon vs. Episodic Setting

In this thesis, we have presented variants of HiREPS for the three different cases of RL, the episodic setting, the finite horizon setting and the infinite horizon setting. In our experiments, we have shown how similar tasks can be solved either using parametrized action generators such as movement primitives in the episodic setting or using learned control policies in the infinite horizon setting. For example, in Chapter 3 we have shown how the pendulum swingup task can be solved using a combination of linear controllers, while in Chapter 4 we have used DMPs to solve a similar swingup problem. Thus, the question for the practitioner may be which scenario to use for a given task. Generally, one can argue that the infinite horizon setting takes advantage of more data generated throughout one rollout and is able to perform more exploration in one rollout. On the other hand, the infinite horizon setting requires some form of forward model which requires the selection of good features in a sensible complexity. Furthermore, the infinite horizon setting is more susceptible to noise in the sensors and actuators, especially when using fine time discretizations.

This difficulty of selecting the appropriate learning framework is present in many tasks and motivates the need for a survey paper. In this survey paper, contenders of both frameworks, the episodic and the infinite horizon framework should be discussed and evaluated on a range of real world problems and simulations. The evaluation should concentrate on both the ease of use, i.e., how much time one has to spend on, for example, the correct feature configuration as well as the learning time on the real system and the re-usability of setups for different tasks. The goal of this survey paper should be to give an overview as well as recommendations for task categories on which learning framework to try and wich setups of that learning framework to use as a starting point.

6 Publications

Excerpts of the research presented in this thesis have led to the following publications.

6.1 Articles in Scientific Journals

C. Daniel, H. van Hoof, G. Neumann and J. Peters. Probabilistic Inference for Determining Options in Reinforcement Learning. *European Conference On Machine Learning Journal Track (ECMLPKKD).* Submitted.

C. Daniel, G. Neumann, O. Kroemer and J. Peters. Hierarchical Relative Entropy Policy Search. *Journal of Machine Learning (JMLR).* In press.

C. Daniel, O. Kroemer, M. Viering, J. Metz and J. Peters (2015). Active Reward Learning with a Novel Acquisition Function. *Autonomous Robots (AuRo)*, 39(3):389–405, Springer

A. Paraschos, **C. Daniel**, J. Peters and G. Neumann. Probabilistic Movement Primitives. *Transactions on Robotics.* Submitted.

G. Neumann, **C. Daniel**, A. Paraschos, A. Kupcsik and J. Peters (2014). Learning Modular Policies for Robotics. *Frontiers in Computational Neuroscience.* 8(62)

6.2 Articles in Conference Proceedings

C. Daniel, J. Taylor, and S. Nowozin (2016). Learning Step Size Controllers for Robust Neural Network Training. *National Conference of the American Association for Artificial Intelligence (AAAI).*

O. Kroemer, **C. Daniel**, G. Neumann, H. van Hoof, J. Peters (2015). Towards Learning Hierarchical Skills for Multi-Phase Manipulation Tasks, *Proceedings of the International Conference on Robotics and Automation (ICRA)* . **Best Paper Award Finalist, Best Student Paper Award Finalist**

C. Daniel, M. Viering, J. Metz, O. Kroemer and J. Peters (2014). Active Reward Learning. *Robots: Science and Systems (R:SS). (Acceptance rate* $\approx 30\%$*)*

C. Daniel, G. Neumann, O. Kroemer and J. Peters (2013). Learning Sequential Motor Tasks. *International Conferene on Robotics and Automation (ICRA).*

C. Daniel, G. Neumann and J. Peters (2013). Learning Versatile Solutions. *Machine Learning and Cognitive Science (MLCOGS)*

A. Paraschos, **C. Daniel**, J. Peters and G. Neumann (2013). Probabilistic Movement Primitives. *Advances in Neural Information Processing Systems (NIPS). (Acceptance rate* $\approx 25\%$*)*

C. Daniel, G. Neumann and J. Peters (2012). Learning Concurrent Motor Skills in Versatile Solution Spaces. *International Conferene on Intelligent Robots and Systems (IROS)* **IROS 2012 CoTeSys Best Paper Award, IROS 2012 Best Paper Award Finalist, IROS 2012 Best Student Paper Award Finalist**

C. Daniel, G. Neumann and J. Peters (2012). Hierarchical Relative Entropy Policy Search. *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS). (Acceptance rate* $\approx 34\%$*)*

Bibliography

A. Abdolmaleki, R. Lioutikov, J Peters, N. Lau, L. Reis, and G. Neumann. Model-based relative entropy stochastic search. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

R. Akrour, M. Schoenauer, and M. Sebag. Preference-based policy learning. In *Machine Learning and Knowledge Discovery in Databases*, pages 12–27. Springer, 2011.

R. Akrour, M. Schoenauer, and M. Sebag. Interactive robot education. In *European Conference on Machine Learning Workshop*, 2013.

M. Gheshlaghi Azar, V. Gómez, and H. J. Kappen. Dynamic Policy Programming. *Journal of Machine Learning Research*, 13(Nov):3207–3245, 2012.

J. A. Bagnell and J. C. Schneider. Covariant Policy Search. In *Proceedings of the IEEE International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

J. A. Bagnell and J. G. Schneider. Autonomous Helicopter Control using Reinforcement Learning Policy Search Methods. In *Proceedings of the IEEE International Conference for Robotics and Automation (ICRA)*, pages 1615–1620, 2001.

R. Balasubramanian, L. Xu, P. D. Brook, J. R. Smith, and Y. Matsuoka. Physical human interactive guidance: Identifying grasping principles from human-planned grasps. *IEEE Transactions on Robotics*, 2012.

A.G. Barto, S. Singh, and N. Chentanez. Intrinsically motivated learning of hierarchical collections of skills. *Proceedings of the International Conference on Developmental Learning (ICDL),*, 2004.

L. E. Baum. An equality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, 3:1–8, 1972.

J. Baxter and P. L. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, 2006.

J. Bratman, S. Singh, J. Sorg, and R. Lewis. Strong mitigation: Nesting search for good policies within search for good reward. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 407–414. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

M. Cakmak and A. L. Thomaz. Designing robot learners that ask good questions. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pages 17–24. ACM, 2012.

S. Calinon, P. Kormushev, and D. Caldwell. Compliant Skills Acquisition and Multi-Optima Policy Search with EM-based Reinforcement Learning. *Robotics and Autonomous Systems (RAS)*, 61(4):369 – 379, 2013.

W. Cheng, J. Fürnkranz, E. Hüllermeier, and S.-H. Park. Preference-based policy iteration: Leveraging preference learning for reinforcement learning. In *Machine Learning and Knowledge Discovery in Databases*, pages 312–327. Springer, 2011.

W. Chu and Z. Ghahramani. Preference learning with gaussian processes. In *Proceedings of the 22nd international conference on Machine learning*, pages 137–144. ACM, 2005.

B. Da Silva, G. Konidaris, and A.G. Barto. Learning parameterized skills. *In Proceedings of the International Conference on Machine Learning (ICML)*, 2012.

H. Dang and P. K. Allen. Learning grasp stability. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2392–2397. IEEE, 2012.

C. Daniel, G. Neumann, and J. Peters. Hierarchical Relative Entropy Policy Search. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2012a.

C. Daniel, G. Neumann, and J. Peters. Learning Concurrent Motor Skills in Versatile Solution Spaces. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012b.

C. Daniel, G. Neumann, O. Kroemer, and J. Peters. Learning Sequential Motor Tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

P. Dayan and G. E. Hinton. Feudal reinforcement learning. In *Advances in neural information processing systems*, pages 271–271. Morgan Kaufmann Publishers, 1993.

P. Dayan and G. E. Hinton. Using expectation-maximization for reinforcement learning. *Neural Computation*, 9(2):271–278, 1997. ISSN 0899-7667.

M. P. Deisenroth. *Efficient Reinforcement Learning using Gaussian Processes*. KIT Scientific Publishing, 2010. ISBN 978-3-86644-569-7.

A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

S. Devlin and D. Kudenko. Dynamic potential-based reward shaping. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 433–440. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research (JAIR)*, 2000.

M. Dorigo and M. Colombetti. Robot shaping: Developing autonomous agents through learning. *Artificial intelligence*, 71(2):321–370, 1994.

G. Endo, J. Morimoto, T. Matsubara, J. Nakanishi, and G. Cheng. Learning CPG-based Biped Locomotion with a Policy Gradient Method: Application to a Humanoid Robot. *International Journal of Robotics Research*, 2008.

Y. Engel, S. Mannor, and R. Meir. Reinforcement learning with Gaussian processes. In *International Conference on Machine Learning*, 2005.

E. B. Fox, M. I. Jordan, E. B. Sudderth, and A. S. Willsky. Sharing features among dynamical systems with beta processes. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.

M. Ghavamzadeh and Y. Engel. Bayesian policy gradient algorithms. *Advances in Neural Information Processing Systems*, 2007.

M. Ghavamzadeh and S. Mahadevan. Hierarchical Policy Gradient Algorithms. In *International Conference for Machine Learning (ICML)*, 2003.

J. V. Graça, K. Ganchev, and B. Taskar. Expectation maximization and posterior constraints. In *Neural Information and Processing Systems (NIPS)*, 2007.

J. V. Graça, K. Ganchev, B. Taskar, and F. Pereira. Posterior vs parameter sparsity in latent variable models. In Y. Bengio, D. Schuurmans, J.D. Lafferty, C.K.I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems (NIPS)*, 2009.

S. Griffith, K. Subramanian, J. Scholz, C. Isbell, and A. L. Thomaz. Policy shaping: Integrating human feedback with reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

H. W. Hake and W. R. Garner. The effect of presenting various numbers of discrete steps on scale reading accuracy. *Journal of Experimental Psychology*, 42(5):358, 1951.

N. Hansen, S.D. Muller, and P. Koumoutsakos. Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18, 2003.

M. D. Hoffman, E. Brochu, and N. de Freitas. Portfolio allocation for Bayesian optimization. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2011.

A. Ijspeert, J. Nakanishi, and S. Schaal. Learning Attractor Landscapes for Learning Motor Primitives. In *Advances in Neural Information Processing Systems (NIPS)*, 2003.

A. Jain, B. Wojcik, T. Joachims, and A. Saxena. Learning trajectory preferences for manipulators via iterative improvement. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

S. J. Julier and J. K. Uhlmann. A new extension of the kalman filter to nonlinear systems. In *International Symposium on Aerospace/Defense Sensing, Simulation and Controls*, 1997.

L. P. Kaelbling. Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the International Conference on Machine Learning (ICML)*, 1993.

S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, H. Yokoi, and H. Hirukawa. Biped Walking Pattern Generation by using Preview Control of Zero-Moment Point. In *Internation Conference on Robotic and Automation (ICRA)*, 2003.

W. B. Knox and P. Stone. Interactively shaping agents via human reinforcement: The TAMER framework. In *International Conference on Knowledge Capture*, 2009.

J. Kober and J. Peters. Policy Search for Motor Primitives in Robotics. *Machine Learning*, (1-2): 171–203, 2010.

J. Kober, B. J. Mohler, and J Peters. Learning perceptual coupling for motor primitives. In *Intelligent Robots and Systems (IROS)*, 2008.

J. Kober, K. Mülling, O. Kroemer, C. H. Lampert, B. Schölkopf, and J. Peters. Movement Templates for Learning of Hitting and Batting. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2010a.

J. Kober, E. Oztop, and J. Peters. Reinforcement Learning to adjust Robot Movements to New Situations. In *Proceedings of the Robotics: Science and Systems Conference (RSS)*, 2010b.

N. Kohl and Peter Stone. Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion. In *Proceedings of the IEEE International Conference for Robotics and Automation (ICRA)*, 2003.

G. Konidaris and A. Barto. Autonomous shaping: Knowledge transfer in reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2006.

G. Konidaris and A. Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.

G. Konidaris, S. Osentoski, and P. Thomas. Value function approximation in reinforcement learning using the fourier basis. *AAAI Conference on Artificial Intelligence (AAAI)*, 2011.

P. Kormushev, S. Calinon, and D. Caldwell. Robot Motor Skill Coordination with EM-based Reinforcement Learning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

O. Kroemer, R. Detry, J. Piater, and J. Peters. Combining active learning and reactive control for robot grasping. *Robotics and Autonomous Systems (RAS)*, 58(9):1105–1116, 2010.

A. Kupcsik, M. P. Deisenroth, J. Peters, and G. Neumann. Data-Efficient Contextual Policy Search for Robot Movement Skills. *Journal of Artificial Intelligence*, 2014.

H. J. Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Fluids Engineering*, 86(1):96–106, 1964.

M. G. Lagoudakis and R. Parr. Least-Squares Policy Iteration. *Journal of Machine Learning Research*, 4:1107–1149, December 2003.

S. Levine and P. Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

K. Y. Levy and N. Shimkin. Unified inter and intra options learning using policy gradient methods. In *Recent Advances in Reinforcement Learning*, pages 153–164. Springer, New York City, 2012.

M. Lopes, F. Melo, and L. Montesano. Active learning for reward estimation in inverse reinforcement learning. In *Machine Learning and Knowledge Discovery in Databases*, pages 31–46. Springer, 2009.

T. A. Mann and S. Mannor. Scaling up approximate value iteration with options: Better policies with fewer iterations. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2014.

A. McGovern and A. G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. *International Conference on Machine Learning (ICML)*, page 8, 2001.

N. Mehta, S. Ray, P. Tadepalli, and T. G. Dietterich. Automatic discovery and transfer of MAXQ hierarchies. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2008.

I. Menache, S. Mannor, and N. Shimkin. Q-cut - dynamic discovery of sub-goals in reinforcement learning. In *Proceedings of the European Conference on Machine Learning (ECML)*, 2002.

G. A. Miller. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological Review*, 63(2):81, 1956.

J. Mockus, V. Tiesis, and A. Zilinskas. The application of bayesian methods for seeking the extremum. *Towards Global Optimization*, 1978.

J. Morimoto and K. Doya. Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning. *Robotics and Autonomous Systems*, 36(1):37–51, 2001.

K. Mülling, J. Kober, O. Kroemer, and J. Peters. Learning to select and generalize striking movements in robot table tennis. *International Journal of Robotics Research (IJRR)*, 32(3):263–279, 2013.

G. Neumann. Variational Inference for Policy Search in Changing Situations. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, 2011.

G. Neumann and J. Peters. Fitted Q-Iteration by Advantage Weighted Regression. In *Advances in Neural Information Processing Systems (NIPS)*, 2009.

A. Ng and A. Coates. Autonomous Inverted Helicopter Flight via Reinforcement Learning. *Experimental Robotics IX*, 1998.

A. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Internation Conference on Machine Learning (ICML)*, 1999.

S. Niekum and A. G. Barto. Clustering via dirichlet process mixture models for portable skill discovery. In *Advances in Neural Information Processing Systems (NIPS)*, 2011.

S. Niekum, S. Osentoski, G.D. Konidaris, and A.G. Barto. Learning and generalization of complex tasks from unstructured demonstrations. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.

A. Paraschos, C. Daniel, J. Peters, and G Neumann. Probabilistic movement primitives. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. *Advances in Neural Information Processing Systems (NIPS)*, 1998.

J. Peters and S. Schaal. Policy Gradient methods for Robotics. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robotics Systems (IROS)*, 2006.

J. Peters, K. Mülling, and Y. Altun. Relative Entropy Policy Search. In *Proceedings of the 24th National Conference on Artificial Intelligence (AAAI)*, 2010.

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

N. Ratliff, A. Bagnell, and M. Zinkevich. Maximum Margin Planning. In *In Proceedings of the 23rd International Conference on Machine Learning (ICML)*, 2006.

Nathan D Ratliff, David Silver, and J Andrew Bagnell. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 27(1):25–53, 2009.

K. Rawlik, M. Toussaint, and S. Vijayakumar. On Stochastic Optimal Control and Reinforcement Learning by Approximate Inference. In *Proceedings of Robotics: Science and Systems*, 2012.

M. T. Rosenstein. Robot Weightlifting by Direct Policy Search. *Proceedings of the International Joint Conference on Artificial Intelligence (ICJAI)*, 2001.

S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert. Learning Movement Primitives. In *Proceedings of the International Symposium on Robotics Research*, (ISRR), 2003.

M. Schoenauer, R. Akrour, M. Sebag, and J.-C. Souplet. Programming by feedback. In *International Conference on Machine Learning (ICML)*, 2014.

J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel. Trust region policy optimization. *arXiv preprint arXiv:1502.05477*, 2015.

D. Silver and K. Ciosek. Compositional Planning Using Optimal Option Models. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2012.

Ö. Simsek and A. G. Barto. Skill characterization based on betweenness. In *Advances in Neural Information Processing Systems (NIPS)*, 2008.

Ö. Simsek, A. P. Wolfe, and A. G. Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2005.

S. Singh, R. L. Lewis, A. G. Barto, and J. Sorg. Intrinsically motivated reinforcement learning: An evolutionary perspective. *Autonomous Mental Development*, 2(2):70–82, 2010.

N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *Internation Conference on Machine Learning (ICML)*, 2010.

S. Still and D. Precup. An Information-theoretic Approach to Curiosity-driven Reinforcement Learning. *Proceedings of the International Conference on Humanoid Robotics*, 2011.

M. Stolle and D. Precup. Learning options in reinforcement learning. In *Abstraction, Reformulation, and Approximation*, pages 212–223. Springer, New York City, 2002.

P. Stone. Scaling reinforcement learning toward RoboCup soccer. *Proceedings of the Conference on Machine Learning (ICML)*, 2001.

M. J. A. Strens. Policy Search using Paired Comparisons. *Journal of Machine Learning Research (JMLR)*, 3:921–950, 2003.

F. Stulp and S. Schaal. Hierarchical Reinforcement Learning with Movement Primitives. In *IEEE International Conference on Humanoid Robots (HUMANOIDS)*, 2012.

F. Stulp and O. Sigaud. Policy improvement methods: Between black-box optimization and episodic reinforcement learning. *Journées Francophones Planification, Décision, et Apprentissage pour la conduite de systèmes*, 2013.

F. R. Suárez, M. J. Cornellà, and M. A. Roa. Grasp quality measures. *Autonomous Robots (AuRo)*, 38:65–88, 2012.

R. S. Sutton, D. Precup, and S. Singh. Intra-option learning about temporally abstract actions. In *Proccedings of the International Conference on Machine Learning (ICML)*, 1998.

R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Advances in Neural Information Processing Systems (NIPS)*, 1999a.

R. S. Sutton, D. Precup, and S. Singh. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, 112:181–211, 1999b.

R.S. Sutton. Generalization in Reinforcement Learning: Successful Examples using Sparse Coarse Coding. In *Advances in Neural Information Processing Systems (NIPS)*, 1996.

R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Boston, MA, 1998.

E.. Theodorou, J. Buchli, and S. Schaal. A generalized path integral control approach to reinforcement learning. *Journal of Machine Learning Research*, 11:3137–3181, 2010.

P. S Thomas and A. G. Barto. Motor primitive discovery. In *IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL)*, 2012.

A. L. Thomaz and C. Breazeal. Teachable robots: Understanding human teaching behavior to build more effective robot learners. *Artificial Intelligence (AI)*, 172(6):716–737, 2008.

M. Toussaint, S. Harmeling, and A. Storkey. Probabilistic inference for solving (po) mdps. *University of Edinburgh, School of Informatics Research Report EDI-INF-RR-0934*, 2006.

A. Ude, A. Gams, T. Asfour, and J. Morimoto. Task-Specific Generalization of Discrete and Periodic Dynamic Movement Primitives. *IEEE Transactions on Robotics*, 5, October 2010. ISSN 1552-3098.

H. van Hoof, J. Peters, and G. Neumann. Learning of non-parametric control policies with high-dimensional state features. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2015.

C. H. Watkins and P. Dayan. Q-Learning. *Machine Learning*, 8(3-4):279–292, 1992. doi: 10.1023/A:1022676722315.

D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber. Natural evolution strategies. *The Journal of Machine Learning Research*, 15(1):949–980, 2014.

A. Wilson, A. Fern, and P. Tadepalli. A bayesian approach for policy learning from trajectory preference queries. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

D. Wingate, N. D. Goodman, D. M. Roy, L. P. Kaelbling, and J. B. Tenenbaum. Bayesian policy search with policy priors. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

B. Ziebart, A. Maas, A. Bagnell, and A. Dey. Maximum entropy inverse reinforcement learning. In *Conference on Artificial Intelligence (AAAI)*, 2008.

A Appendix

A.1 Derivation of the Lower Bound

Consider the optimization problem in Equation (2.14) with the real conditional $p(o|s,a)$ instead of the responsibilities $\tilde{p}(o|s,a)$. For simplicity, we do not introduce the steady-state distribution and the normalization constraint as, our results are not affected by these constraints. The Lagrangian of this problem is then given by

$$\begin{aligned} F(\pi,\eta,\xi) = &\sum_{o\in\mathcal{O}} \iint p(\boldsymbol{s},\boldsymbol{\omega},\boldsymbol{o}) R_{\text{sa}} \,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{\omega} \\ &+ \eta\left(\epsilon - \sum_{o\in\mathcal{O}} \iint p(\boldsymbol{s},\boldsymbol{\omega},\boldsymbol{o}) \log \frac{p(\boldsymbol{s},\boldsymbol{\omega},\boldsymbol{o})}{q(\boldsymbol{s},\boldsymbol{\omega}) p(o|\boldsymbol{s},\boldsymbol{\omega})} \,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{\omega}\right) \\ &+ \xi\left(\tilde{\kappa} + \iint p(\boldsymbol{s},\boldsymbol{\omega}) \sum_{\boldsymbol{o}\in\mathcal{O}} p(o|\boldsymbol{s},\boldsymbol{\omega}) \log p(o|\boldsymbol{s},\boldsymbol{\omega}) \,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{\omega}\right). \end{aligned} \tag{A.1}$$

Simplifying the terms, we obtain

$$F(\pi,\eta,\xi) = \sum_{o\in\mathcal{O}} \iint p(\boldsymbol{s},\boldsymbol{\omega},\boldsymbol{o})\Big(R_{\text{sa}} - \eta \log \frac{p(\boldsymbol{s},\boldsymbol{\omega},\boldsymbol{o})}{q(\boldsymbol{s},\boldsymbol{\omega}) p(\boldsymbol{o}|\boldsymbol{s},\boldsymbol{\omega})^{1+\xi/\eta}}\Big) \,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{\omega} + \eta\epsilon + \xi\tilde{\kappa}. \tag{A.2}$$

However, determining a closed form solution for $p(\boldsymbol{s},\boldsymbol{\omega},\boldsymbol{o})$ is infeasible as the conditional $p(\boldsymbol{o}|\boldsymbol{s},\boldsymbol{\omega})$ is inside the log. Yet, we can determine a lower bound $L(p,\eta,\xi,\tilde{p})$ by using a proposal distribution $\tilde{p}(\boldsymbol{o}|\boldsymbol{s},\boldsymbol{\omega})$ for $p(\boldsymbol{o}|\boldsymbol{s},\boldsymbol{\omega})$ which we can iteratively maximize in an Expectation-Maximization like manner. We need to verify that L is a lower bound on F and that maximizing L w.r.t $\tilde{p}$ is equivalent to setting $\tilde{p}(\boldsymbol{o}|\boldsymbol{s},\boldsymbol{\omega}) = p(\boldsymbol{o}|\boldsymbol{s},\boldsymbol{\omega})$, both of which follows from the relation

$$L = F - (\eta+\xi) \iint p(\boldsymbol{s},\boldsymbol{\omega}) \underbrace{\sum_{\boldsymbol{o}\in\mathcal{O}} p(\boldsymbol{o}|\boldsymbol{s},\boldsymbol{\omega}) \log \frac{p(\boldsymbol{o}|\boldsymbol{s},\boldsymbol{\omega})}{\tilde{p}(\boldsymbol{o}|\boldsymbol{s},\boldsymbol{\omega})}}_{D_{KL}\left(p(\boldsymbol{o}|\boldsymbol{s},\boldsymbol{\omega})\|\tilde{p}(\boldsymbol{o}|\boldsymbol{s},\boldsymbol{\omega})\right)\geq 0} \,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{\omega}.$$

After the Expectation step, the lower bound is tight, i.e., $\max_{\tilde{p}} L(p,\eta,\xi,\tilde{p}) = F(p,\eta,\xi)$. In the Maximization step, we fix $\tilde{p}$ and maximize L w.r.t p, η and ξ. This combination defines our constraint optimization problem.

A.2 Derivation of Contextual HiREPS

We first reiterate the complete optimization problem, i.e.,

$$\begin{aligned}
\max_{\pi,\mu^\pi} J(\pi) &= \max_{\pi,\mu^\pi} \sum_{o\in\mathcal{O}} \iint \mu^\pi(s)\pi(\omega|s,o)\pi(o|s)\mathcal{R}_{s\omega}\, \mathrm{d}s\, \mathrm{d}\omega, \\
\text{s.t.} \quad \epsilon &\geq \sum_{o\in\mathcal{O}} \iint p(s,\omega,o) \log \frac{p(s,\omega,o)}{q(s,\omega)\tilde{p}(o|s,\omega)}\, \mathrm{d}s\, \mathrm{d}\omega, \\
\tilde{\kappa} &\geq -\sum_{o\in\mathcal{O}} \iint \mu^\pi(s)\pi(\omega|s,o)p(o|s,\omega)\log\tilde{p}(o|s,\omega)\, \mathrm{d}s\, \mathrm{d}\omega, \\
\hat{\boldsymbol{\phi}} &= \sum_{o\in\mathcal{O}} \iint \mu^\pi(s)\pi(\omega|s,o)\pi(o|s)\boldsymbol{\phi}(s)\, \mathrm{d}s\, \mathrm{d}\omega, \\
1 &= \sum_{o\in\mathcal{O}} \iint \mu^\pi(s)\pi(\omega|s,o)\pi(o|s)\, \mathrm{d}s\, \mathrm{d}\omega,
\end{aligned} \tag{A.3}$$

which includes an additional constraint to represent prior knowledge about the desired behavior of sub-policies, i.e., that sub-policies should spread out and not overlap. In the entropy constraint, we replace only the responsibility term inside the log with the approximation $\tilde{p}(o|s,\omega)$, as we can combine the expectation over the entropy into $\mathbb{E}_{p(s,\omega,o)} \log\tilde{p}(o|s,\omega)$.

The Lagrangian formulation reads

$$\begin{aligned}
L = \sum_{o\in\mathcal{O}} \iint p(s,\omega,o)\Big[\mathcal{R}_{s\omega} - \eta\log\frac{p(s,\omega,o)}{q(s,\omega)\tilde{p}(o|s,\omega)} \\
+\xi\log\tilde{p}(o|s,\omega) - \boldsymbol{\theta}^T\boldsymbol{\phi}(s) - \lambda\Big]\, \mathrm{d}s\, \mathrm{d}\omega + \eta\epsilon + \lambda + \boldsymbol{\theta}^T\hat{\boldsymbol{\phi}} + \xi\tilde{\kappa},
\end{aligned} \tag{A.4}$$

where η , $\boldsymbol{\theta}$, ξ and λ are Lagrangian parameters. To arrive at a closed form update rule for $p(s,\omega,o)$, we differentiate L

$$\begin{aligned}
\frac{\mathrm{d}L}{\mathrm{d}p(s,\omega,o)} &= \mathcal{R}_{s\omega} - \eta\log\frac{p(s,\omega,o)}{q(s,\omega)\tilde{p}(o|s,\omega)} + \xi\log\tilde{p}(o|s,\omega) - \boldsymbol{\theta}^T\boldsymbol{\phi}(s) - \lambda - \eta, \\
&= \mathcal{R}_{s\omega} - \eta\log\frac{p(s,\omega,o)}{q(s,\omega)\tilde{p}(o|s,\omega)^{1+\xi/\eta}} - \boldsymbol{\theta}^T\boldsymbol{\phi}(s) - \lambda - \eta,
\end{aligned} \tag{A.5}$$

with

$$\xi\log\tilde{p}(o|s,\omega) = \eta\frac{\xi}{\eta}\log\tilde{p}(o|s,\omega) = \eta\log\tilde{p}(o|s,\omega)^{\frac{\xi}{\eta}} = -\eta\log\frac{1}{\tilde{p}(o|s,\omega)^{\xi/\eta}}.$$

We set the derivative to zero and write $V(s) = \boldsymbol{\theta}^T\boldsymbol{\phi}(s)$ to solve for the update rule

$$p(s,\omega,o) = q(s,\omega)\tilde{p}(o|s,\omega)^{1+\xi/\eta}\exp\left(\frac{\mathcal{R}_{s\omega} - V(s)}{\eta}\right)\exp\left(-1-\frac{\lambda}{\eta}\right). \tag{A.6}$$

Here, we use the fact that

$$\sum_{\boldsymbol{o}\in\mathcal{O}}\iint \mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{\omega}|\boldsymbol{s},\boldsymbol{o})\pi(\boldsymbol{o}|\boldsymbol{s})\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{\omega}=1, \tag{A.7}$$

and, thus,

$$\exp\left(-1-\frac{\lambda}{\eta}\right)^{-1}=\sum_{\boldsymbol{o}\in\mathcal{O}}\iint q(\boldsymbol{s},\boldsymbol{\omega})\tilde{p}(\boldsymbol{o}|\boldsymbol{s},\boldsymbol{\omega})^{1+\xi/\eta}\exp\left(\frac{\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}-V(\boldsymbol{s})}{\eta}\right)\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{\omega}. \tag{A.8}$$

Hence, the term $\exp(-1-\lambda/\eta)$ acts as a normalization constant and $p(\boldsymbol{s},\boldsymbol{\omega},\boldsymbol{o})$ can be written as

$$p(\boldsymbol{s},\boldsymbol{\omega},\boldsymbol{o})=\frac{q(\boldsymbol{s},\boldsymbol{\omega})\tilde{p}(\boldsymbol{o}|\boldsymbol{s},\boldsymbol{\omega})^{1+\xi/\eta}\exp\left(\frac{\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}-V(\boldsymbol{s})}{\eta}\right)}{\sum_{\boldsymbol{o}\in\mathcal{O}}\iint q(\boldsymbol{s},\boldsymbol{\omega})\tilde{p}(\boldsymbol{o}|\boldsymbol{s},\boldsymbol{\omega})^{1+\xi/\eta}\exp\left(\frac{\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}-V(\boldsymbol{s})}{\eta}\right)\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{\omega}}.$$

However, for improved clarity, we resort to writing the update equation in the proportional formulation

$$p(\boldsymbol{s},\boldsymbol{\omega},\boldsymbol{o})\propto q(\boldsymbol{s},\boldsymbol{\omega})\tilde{p}(\boldsymbol{o}|\boldsymbol{s},\boldsymbol{\omega})^{1+\xi/\eta}\exp\left(\frac{\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}-V(\boldsymbol{s})}{\eta}\right), \tag{A.9}$$

which depends only on the Lagrangian parameters ξ, η and $\boldsymbol{\theta}$ with $V(\boldsymbol{s})=\boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{s})$, but not on the parameter λ. The values for these Lagrangian parameters can be calculated by optimizing the dual problem. The dual formulation can be obtained by inserting Eq. (A.6) into the original Lagrangian formulation, i.e.,

$$\begin{aligned}g(\eta,\boldsymbol{\theta},\xi,\lambda)=\sum_{\boldsymbol{o}\in\mathcal{O}}\iint p(\boldsymbol{s},\boldsymbol{\omega},\boldsymbol{o})\Big[&\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}\\ &-\eta\log\frac{q(\boldsymbol{s},\boldsymbol{\omega})\tilde{p}(\boldsymbol{o}|\boldsymbol{s},\boldsymbol{\omega})^{1+\xi/\eta}\exp\left(\frac{\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}-V(\boldsymbol{s})}{\eta}\right)\exp\left(-1-\frac{\lambda}{\eta}\right)}{q(\boldsymbol{s},\boldsymbol{\omega})\tilde{p}(\boldsymbol{o}|\boldsymbol{s},\boldsymbol{\omega})^{1+\xi/\eta}}\\ &-\boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{s})-\lambda\Big]\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{\omega}+\eta\epsilon+\lambda+\boldsymbol{\theta}^T\hat{\boldsymbol{\phi}}+\xi\tilde{\kappa},\end{aligned} \tag{A.10}$$

which can easily be simplified to

$$g(\eta,\boldsymbol{\theta},\xi,\lambda)=\eta\epsilon+\lambda+\boldsymbol{\theta}^T\hat{\boldsymbol{\phi}}+\xi\tilde{\kappa}+\sum_{\boldsymbol{o}\in\mathcal{O}}\iint p(\boldsymbol{s},\boldsymbol{\omega},\boldsymbol{o})\eta\,\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{\omega}, \tag{A.11}$$

where we again use Eq. (A.7) to simplify further to

$$g(\eta,\boldsymbol{\theta},\xi,\lambda)=\eta\epsilon+\lambda+\boldsymbol{\theta}^T\hat{\boldsymbol{\phi}}+\xi\tilde{\kappa}+\eta. \tag{A.12}$$

At this point we rewrite Eq. (A.8) as

$$\eta+\lambda=\eta\log\sum_{\boldsymbol{o}\in\mathcal{O}}\iint q(\boldsymbol{s},\boldsymbol{\omega})\tilde{p}(\boldsymbol{o}|\boldsymbol{s},\boldsymbol{\omega})^{1+\xi/\eta}\exp\left(\frac{\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}-V(\boldsymbol{s})}{\eta}\right)\mathrm{d}\boldsymbol{s}\,\mathrm{d}\boldsymbol{\omega}, \tag{A.13}$$

where we used $\log(a^{-1}) = -\log(a)$ and multiplied by η. We insert Eq. (A.13) back into Eq. (A.11) and get

$$g(\eta, \boldsymbol{\theta}, \xi) = \eta \log \sum_{\boldsymbol{o} \in \mathcal{O}} \iint q(\boldsymbol{s}, \boldsymbol{\omega}) \tilde{p}(\boldsymbol{o}|\boldsymbol{s}, \boldsymbol{\omega})^{1+\xi/\eta} \exp\left(\frac{\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}} - V(\boldsymbol{s})}{\eta}\right) + \eta\epsilon + \boldsymbol{\theta}^T \hat{\boldsymbol{\phi}} + \xi\tilde{\kappa}. \tag{A.14}$$

The dual function $g(\eta, \boldsymbol{\theta}, \xi)$ is formulated in such a way that it does not depend on the Lagrangian parameter λ, which acts as a normalization factor.

If we work on samples $\{\boldsymbol{s}, \boldsymbol{\omega}\}_i$ with $i = 1, 2, \ldots N$, we have to divide by the generating distribution $q(\boldsymbol{s}, \boldsymbol{\omega})$ and obtain

$$g(\eta, \boldsymbol{\theta}, \xi) = \eta \log \frac{1}{N} \sum_{\boldsymbol{o} \in \mathcal{O}} \sum_{i=1}^{N} \tilde{p}(\boldsymbol{o}|\boldsymbol{s}_i, \boldsymbol{\omega}_i)^{1+\xi/\eta} \exp\left(\frac{R_i - V(\boldsymbol{s}_i)}{\eta}\right) + \eta\epsilon + \boldsymbol{\theta}^T \hat{\boldsymbol{\phi}} + \xi\tilde{\kappa}, \tag{A.15}$$

which we can optimize using standard optimization libraries.

A.3 Derivation of Skill Sequencing

As above, we first reiterate the complete optimization problem, i.e.,

$$\begin{aligned}
\max_{\pi_k, \mu_k^\pi} J(\pi_{1:K}) &= \max_{\pi, \mu_k^\pi} \sum_{k=1}^{K} \sum_{\boldsymbol{o} \in \mathcal{O}} \iint \mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}^k \pi_k(\boldsymbol{\omega}|\boldsymbol{s}) \pi_k(\boldsymbol{o}|\boldsymbol{s}) \mu_k^\pi(\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{\omega} \\
&\quad + \int \mu_{K+1}^\pi(\boldsymbol{s}) \mathcal{R}_{\boldsymbol{s}}^{K+1} \, \mathrm{d}\boldsymbol{s}, \\
\text{s.t.} : \epsilon &\geq \int \mu_{K+1}^\pi(\boldsymbol{s}) \log \frac{\mu_{K+1}^\pi(\boldsymbol{s})}{q_{K+1}(\boldsymbol{s})} \, \mathrm{d}\boldsymbol{s}, \\
\hat{\boldsymbol{\phi}}_0 &= \int \boldsymbol{\phi}(\boldsymbol{s}') \mu_1^\pi(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s}', \\
\forall k \leq K : \epsilon &\geq \sum_{\boldsymbol{o} \in \mathcal{O}} \iint p_k(\boldsymbol{s}, \boldsymbol{\omega}, \boldsymbol{o}) \log \frac{p_k(\boldsymbol{s}, \boldsymbol{\omega}, \boldsymbol{o})}{q_k(\boldsymbol{s}, \boldsymbol{\omega}) \tilde{p}_k(\boldsymbol{o}|\boldsymbol{s}, \boldsymbol{\omega})} \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{\omega}, \\
\tilde{\kappa} &\geq -\sum_{\boldsymbol{o} \in \mathcal{O}} \iint \mu_k^\pi(\boldsymbol{s}) \pi_k(\boldsymbol{\omega}|\boldsymbol{s}, o) p_k(\boldsymbol{s}, \boldsymbol{\omega}) \pi_k(\boldsymbol{o}|\boldsymbol{s}) \log \tilde{p}_k(\boldsymbol{o}|\boldsymbol{s}, \boldsymbol{\omega}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{\omega}, \\
\int \boldsymbol{\phi}(\boldsymbol{s}') \mu_{k+1}^\pi(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s}' &= \sum_{\boldsymbol{o} \in \mathcal{O}} \iiint \mathcal{P}_{\boldsymbol{s}\boldsymbol{s}'}^{\boldsymbol{\omega}} p_k(\boldsymbol{s}, \boldsymbol{\omega}, \boldsymbol{o}) \boldsymbol{\phi}(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{s}' \, \mathrm{d}\boldsymbol{\omega}, \\
\forall k : 1 &= \sum\nolimits_{\boldsymbol{o} \in \mathcal{O}} \iint \mu_k^\pi(\boldsymbol{s}) \pi_k(\boldsymbol{\omega}, \boldsymbol{o}|\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{\omega}.
\end{aligned} \tag{A.16}$$

The Lagrangian formulation reads

$$
\begin{aligned}
L =& \boldsymbol{\theta}_1^T \hat{\boldsymbol{\phi}}_0 + \sum_{k=1}^{K} \eta_k \epsilon + \lambda_k + \int \boldsymbol{\theta}_{k+1}^T \boldsymbol{\phi}(\boldsymbol{s}) \mu_k^\pi(\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s} + \xi_k \tilde{\kappa} + \sum_{\boldsymbol{o} \in \mathcal{O}} \int\int p_k(\boldsymbol{s}, \boldsymbol{\omega}, \boldsymbol{o}) \\
& \Big[\mathscr{R}_{\boldsymbol{s}\boldsymbol{\omega}}^k - \eta_k \log \frac{p_k(\boldsymbol{s}, \boldsymbol{\omega}, \boldsymbol{o})}{q_k(\boldsymbol{s}, \boldsymbol{\omega}) \tilde{p}_k(\boldsymbol{o}|\boldsymbol{s}, \boldsymbol{\omega})} + \xi_k \log \tilde{p}_k(\boldsymbol{o}|\boldsymbol{s}, \boldsymbol{\omega}) \\
& - \int \mathscr{P}_{\boldsymbol{s}\boldsymbol{s}'}^{\boldsymbol{\omega}} \boldsymbol{\theta}_{k+1}^T \boldsymbol{\phi}(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s}' - \lambda_k \Big] \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{\omega} \\
& + \int \mu_{K+1}^\pi(\boldsymbol{s}) \Big[\mathscr{R}_{\boldsymbol{s}}^{K+1} - \eta_{K+1} \log \frac{\mu_{K+1}^\pi(\boldsymbol{s})}{q_{K+1}(\boldsymbol{s})} - \lambda_{K+1} \Big] \mathrm{d}\boldsymbol{s} + \eta_{K+1} \epsilon + \lambda_{K+1}, \qquad \text{(A.17)}
\end{aligned}
$$

where η_k , $\boldsymbol{\theta}_k$, ξ_k and λ_k are Lagrangian parameters. To arrive at a closed form update rule for $p_k(\boldsymbol{s}, \boldsymbol{\omega}, \boldsymbol{o})$, we differentiate L

$$
\begin{aligned}
\frac{\mathrm{d}L}{\mathrm{d}p_k(\boldsymbol{s}, \boldsymbol{\omega}, \boldsymbol{o})} =& \mathscr{R}_{\boldsymbol{s}\boldsymbol{\omega}}^k - \eta_k \log \frac{p_k(\boldsymbol{s}, \boldsymbol{\omega}, \boldsymbol{o})}{q_k(\boldsymbol{s}, \boldsymbol{\omega}) \tilde{p}_k(\boldsymbol{o}|\boldsymbol{s}, \boldsymbol{\omega})} + \xi_k \log \tilde{p}_k(\boldsymbol{o}|\boldsymbol{s}, \boldsymbol{\omega}) \\
& + \int \mathscr{P}_{\boldsymbol{s}\boldsymbol{s}'}^{\boldsymbol{\omega}} \boldsymbol{\theta}_{k+1}^T \boldsymbol{\phi}(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s}' - \boldsymbol{\theta}_k^T \boldsymbol{\phi}(\boldsymbol{s}) - \lambda_k - \eta_k, \\
=& \mathscr{R}_{\boldsymbol{s}\boldsymbol{\omega}}^k - \eta_k \log \frac{p_k(\boldsymbol{s}, \boldsymbol{\omega}, \boldsymbol{o})}{q_k(\boldsymbol{s}, \boldsymbol{\omega}) \tilde{p}(\boldsymbol{o}|\boldsymbol{s}, \boldsymbol{\omega})^{1+\xi_k/\eta_k}} \\
& + \int \mathscr{P}_{\boldsymbol{s}\boldsymbol{s}'}^{\boldsymbol{\omega}} \boldsymbol{\theta}_{k+1}^T \boldsymbol{\phi}(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s}' - \boldsymbol{\theta}_k^T \boldsymbol{\phi}(\boldsymbol{s}) - \lambda_k - \eta_k, \qquad \text{(A.18)}
\end{aligned}
$$

with

$$
\begin{aligned}
\xi_k \log \tilde{p}_k(\boldsymbol{o}|\boldsymbol{s}, \boldsymbol{\omega}) &= \eta_k \frac{\xi_k}{\eta_k} \log \tilde{p}_k(\boldsymbol{o}|\boldsymbol{s}, \boldsymbol{\omega}) && \text{(A.19)} \\
&= \eta_k \log \tilde{p}_k(\boldsymbol{o}|\boldsymbol{s}, \boldsymbol{\omega})^{\xi_k/\eta_k} && \text{(A.20)} \\
&= -\eta_k \log \frac{1}{\tilde{p}_k(\boldsymbol{o}|\boldsymbol{s}, \boldsymbol{\omega})^{\xi_k/\eta_k}}
\end{aligned}
$$

and set the derivative to zero to solve for the update rule

$$
\begin{aligned}
p_k(\boldsymbol{s}, \boldsymbol{\omega}, \boldsymbol{o}) =& q_k(\boldsymbol{s}, \boldsymbol{\omega}) \tilde{p}_k(\boldsymbol{o}|\boldsymbol{s}, \boldsymbol{\omega})^{1+\xi_k/\eta_k} \\
& \exp\left(\frac{\mathscr{R}_{\boldsymbol{s}\boldsymbol{\omega}}^k + \int \mathscr{P}_{\boldsymbol{s}\boldsymbol{s}'}^{\boldsymbol{\omega}} \boldsymbol{\theta}_{k+1}^T \boldsymbol{\phi}(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s}' - \boldsymbol{\theta}_k^T \boldsymbol{\phi}(\boldsymbol{s})}{\eta_k} \right) \exp\left(-1 - \frac{\lambda_k}{\eta_k} \right). \qquad \text{(A.21)}
\end{aligned}
$$

For improved readability we write $\mathbb{E}[V_{k+1}(\boldsymbol{s}')] = \int \mathscr{P}_{\boldsymbol{s}\boldsymbol{s}'}^{\boldsymbol{\omega}} \boldsymbol{\theta}_{k+1}^T \boldsymbol{\phi}(\boldsymbol{s}') \, \mathrm{d}\boldsymbol{s}'$ and $V_k(\boldsymbol{s}) = \boldsymbol{\theta}_k^T \boldsymbol{\phi}(\boldsymbol{s})$. As before, we use

$$
\sum_{\boldsymbol{o} \in \mathcal{O}} \int\int \mu_k^\pi(\boldsymbol{s}) \pi_k(\boldsymbol{\omega}, \boldsymbol{o}|\boldsymbol{s}) \pi_k(\boldsymbol{o}|\boldsymbol{s}) \, \mathrm{d}\boldsymbol{s} \, \mathrm{d}\boldsymbol{\omega} = 1, \qquad \text{(A.22)}
$$

and, thus, obtain

$$\exp\left(-1-\frac{\lambda_k}{\eta_k}\right)^{-1} = \sum_{o\in\mathscr{O}} \iint q_k(s,\omega)\tilde{p}_k(o|s,\omega)^{1+\xi_k/\eta_k} \exp\left(\frac{\mathscr{R}^k_{s\omega} + \mathbb{E}[V_{k+1}(s')] - V_k(s)}{\eta_k}\right) \mathrm{d}s\ \mathrm{d}\omega. \tag{A.23}$$

Hence, the term $\exp\left(-1-\frac{\lambda_k}{\eta_k}\right)$ acts as a normalization constant and $p_k(s,\omega,o)$ can be written as

$$p_k(s,\omega,o) = \frac{q_k(s,\omega)\tilde{p}_k(o|s,\omega)^{1+\xi_k/\eta_k} \exp\left(\frac{\mathscr{R}^k_{s\omega}+\mathbb{E}[V_{k+1}(s')]-V_k(s)}{\eta_k}\right)}{\sum_{o\in\mathscr{O}}\iint q_k(s,\omega)\tilde{p}_k(o|s,\omega)^{1+\xi_k/\eta_k} \exp\left(\frac{\mathscr{R}^k_{s\omega}+\mathbb{E}[V_{k+1}(s')]-V_k(s)}{\eta_k}\right) \mathrm{d}s\ \mathrm{d}\omega}.$$

For improved clarity, we resort to writing the update equation in the proportional formulation

$$p_k(s,\omega,o) \propto q_k(s,\omega)\tilde{p}_k(o|s,\omega)^{1+\xi_k/\eta_k} \exp\left(\frac{\mathscr{R}^k_{s\omega} + \mathbb{E}[V_{k+1}(s')] - V_k(s)}{\eta}\right), \tag{A.24}$$

which depends only on the Lagrangian parameters ξ, η and $\boldsymbol{\theta}$ with $V(s) = \boldsymbol{\theta}^T\boldsymbol{\phi}(s)$, but not on the parameter λ. The values for these Lagrangian parameters can be calculated by optimizing the dual problem.

As before, we set the solution for $p_k(s,\omega,o)$ back into the Lagrangian and simplify to get a dual function that does not depend on the Lagrangian parameters λ_k, i.e.,

$$g(\eta_{1:K+1},\boldsymbol{\theta}_{1:K+1},\xi_{1:K+1}) = \tag{A.25}$$

$$\boldsymbol{\theta}_1^T\hat{\boldsymbol{\phi}}_0 + \sum_{k=1}^{K+1}\eta_k\epsilon + \xi_k\tilde{\kappa} + \eta_k\log\sum_{o\in\mathscr{O}}\iint q_k(s,\omega)\tilde{p}_k(o|s,\omega)^{1+\xi_k/\eta_k} \exp\left(\frac{\mathscr{R}^k_{s\omega} + \mathbb{E}[V_{k+1}(s')] - V_k(s)}{\eta_k}\right) \mathrm{d}s\ \mathrm{d}\omega. \tag{A.26}$$

Now, the dual function $g(\eta_{1:K+1},\boldsymbol{\theta}_{1:K+1},\xi_{1:K+1})$ is formulated in such a way that it does not depend on the Lagrangian parameters λ_k, which acts as a normalization factor.

A.4 Derivation of Infinite Horizon HiREPS

The derivation of the infinite horizon case follows the derivations given above, where the Lagrangian is given by

$$L = \sum_{o\in\mathscr{O}}\iint p(s,\omega,o)\Big[\mathscr{R}_{s\omega} - \eta\log\frac{p(s,\omega,o)}{q(s,\omega)\tilde{p}(o|s,\omega)} + \xi\log\tilde{p}(o|s,\omega) - \boldsymbol{\theta}^T\boldsymbol{\phi}(s) - \lambda\Big]\mathrm{d}s\ \mathrm{d}\omega + \eta\epsilon + \lambda + \int\boldsymbol{\theta}^T\boldsymbol{\phi}(s')\,\mathrm{d}s' + \xi\tilde{\kappa}, \tag{A.27}$$

and the resulting update rule and dual function are given as

$$p(\boldsymbol{s},\boldsymbol{\omega},\boldsymbol{o}) \propto q(\boldsymbol{s},\boldsymbol{\omega})\tilde{p}(\boldsymbol{o}|\boldsymbol{s},\boldsymbol{\omega})^{1+\xi/\eta} \exp\left(\frac{\mathscr{R}_{\boldsymbol{s}\boldsymbol{\omega}} + \mathbb{E}[V(\boldsymbol{s}')] - V(\boldsymbol{s})}{\eta}\right), \tag{A.28}$$

and

$$\begin{aligned} g(\eta,\boldsymbol{\theta},\xi) =& \eta \log \sum_{\boldsymbol{o}\in\mathscr{O}} \int\int q(\boldsymbol{s},\boldsymbol{\omega})\tilde{p}(\boldsymbol{o}|\boldsymbol{s},\boldsymbol{\omega})^{1+\xi/\eta} \exp\left(\frac{\mathscr{R}_{\boldsymbol{s}\boldsymbol{\omega}} + \mathbb{E}[V(\boldsymbol{s}')] - V(\boldsymbol{s})}{\eta}\right) \\ &+ \eta\epsilon + \xi\tilde{\kappa}. \end{aligned} \tag{A.29}$$

As for the other cases we can find the minimum of the dual function using standard optimization libraries.

B Curriculum Vitae

Education

Technische Universität Darmstadt, Germany (2012 - today)

Position	Ph.D. Student
Thesis Title	Learning Hierarchical Policies from Human Ratings
Adviser	Professor Jan Peters

Technische Universität Darmstadt, Germany (2010 - 2012)

Position	M.Sc. Student Computational Engineering (second year)
Thesis Title	Hierarchical Relative Entropy Policy Search
Adviser	Professor Jan Peters

École Polytechnique Fédérale de Lausanne, Switzerland (2009 - 2010)

Position	M.Sc. Student Computational Engineering (first year)

Technische Universität Darmstadt, Germany (2006 - 2009)

Position	B.Sc. Student Computational Engineering
Thesis Title	On the Acceleration of CFD Computationsusing a GPU Approach
Adviser	Professor Michael Schaefer

Liebigschule Frankfurt am Main, Germany (1992 - 2005)

Position	Abitur Student (university-entrance diploma)
Majors	Math, English

Work Experience

Microsoft Research Cambridge, United Kingdom (2015)

Position	Research Intern
Research Project	Learning Step Size Controllers for Robust Neural Network Training
Supervisor	Sebastian Nowozin

Fraunhofer Institut für Graphische Datenverarbeitung, (2010 - 2011)
Germany

Position	Student Assistant
Research Project	Development of a Realtime Computational Fluid Dynamics System.
Supervisor	Daniel Weber

Learning Algorithms and Systems Laboratory, EPFL, (2010)
Switzerland

Position	Research Assistant
Research Project	Reinforcement Learning from Negative Demonstrations
Supervisor	Daniel Grollman, Professor Aude Billard

Civil Service in After-School Care Club, Germany (2005 - 2006)

Awards

Datenlotsen Award for the Master's Thesis in Computer Science at TU Darmstadt
IROS 2012 CoTeSys Best Paper Award
IROS 2012 Best Paper Award Finalist
IROS 2012 Best Student Paper Award Finalist
ICRA 2015 Best Paper Award Finalist
ICRA 2015 Best Student Paper Award Finalist

Invited Talks

Invited Talk at Universität Freiburg. Learning Step Size Controllers for Robust Neural Network Training. *Hosted by Prof. Frank Hutter.* (2016).
Invited Talk at Bosch Research Stuttgart. Autonomous Learning for Real Robots. *Hosted by Dr. D. Nguyen-Tuong.* (2015)
Invited Talk at Technical R:SS Session, AAAI 2015. Active Reward Learning. *Hosted by B. Knox and G. Konidaris.* (2015)
Invited Talk at Imperial College London. Learning with Human Ratings. *Hosted by M. Deisenroth.* (2014)

Program Committee

International Joint Conference on Artificial Intelligence (IJCAI) (2013)
European Workshop on Reinforcement Learning (EWRL) (2012)

Reviewing

International Conference on Robotics and Automation (ICRA) (2015)
Neural Information and Processing Systems (NIPS) (2014)
International Conference on Robotics and Automation (ICRA) (2014)
International Conference on Intelligent Robots and Systems (IROS) (2014)
Neural Information and Processing Systems (NIPS) (2013)
International Conference on Robotics and Automation (ICRA) (2011)

Teaching

Technical Foundations of Computing, Guest Lecturer (2014)
Robot Learning Seminar, Guest Lecturer (2012 - 2014)
Technical Foundations of Computing, Teaching Assistant (2014)
Computational Engineering, Teaching Assistant (2013)
Machine Learning, Teaching Assistant (2013)

Student Co-Supervision

O. Arenz. Feature Extraction for Inverse Reinforcement Learning. *Master Thesis*	(2014)
K. Berninger. Hierarchical Policy Search Algorithms. *Bachelor Thesis*	(2015)
C. Mayer. Learning to Sequence Movement Primitives for Rhythmic Tasks. *Bachelor Thesis*	(2015)
B. Loew and D. Wilberts. Learning Ball on a Beam on the KUKA Lightweight Arm. *Integrated Project*	(2014)
J. Gast and B. Wild. Ball Balancing Along a Trajectory. *Integrated Project*	(2013)
J. Metz and M. Viering. Learning Reward Functions. *Integrated Project*	(2013)

EU Project Member

Composing Learning for Artificial Cognitive Systems (CompLACS)	(2013-2015)
Generalizing Robot Manipulation Tasks (GeRT)	(2013)