Empowered Skills

Empowered Skills
Master-Thesis von Alexander Gabriel aus Weilburg
Januar 2017

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Empowered Skills
Empowered Skills

Vorgelegte Master-Thesis von Alexander Gabriel aus Weilburg

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Dr. Riad Akrour

Tag der Einreichung:

For everyone,
as science should be.

Erklärung zur Master-Thesis

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter und 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.
In der abgegebenen Thesis stimmen die schriftliche und elektronische Fassung überein.

Darmstadt, den 30. Januar 2017

(Alexander Gabriel)

Thesis Statement

I herewith formally declare that I have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.
In the submitted thesis the written copies and the electronic version are identical in content.

Darmstadt, January 30, 2017

(Alexander Gabriel)

Abstract

Robot Reinforcement Learning (RL) algorithms return a policy that maximizes a global cumulative reward signal but typically do not create diverse behaviors. Hence, the policy will typically only capture a single solution of a task. However, many motor tasks have a large variety of solutions and the knowledge about these solutions can have several advantages. For example, in an adversarial setting such as robot table tennis, the lack of diversity renders the behavior predictable and hence easy to counter for the opponent. In an interactive setting such as learning from human feedback, an emphasis on diversity gives the human more opportunity for guiding the robot and to avoid the latter to be stuck in local optima of the task. In order to increase diversity of the learned behaviors, we leverage prior work on intrinsic motivation and empowerment. We derive a new intrinsic motivation signal by enriching the description of a task with an outcome space, representing interesting aspects of a sensorimotor stream. For example, in table tennis, the outcome space could be given by the return position and return ball speed. The intrinsic motivation is now given by the diversity of future outcomes, a concept also known as empowerment. We derive a new policy search algorithm that maximizes a trade-off between the extrinsic reward and this intrinsic motivation criterion. Experiments on a planar reaching task and simulated robot table tennis demonstrate that our algorithm can learn a diverse set of behaviors within the area of interest of the tasks.

Acknowledgments

My thanks go out to Jacqueline for her hospitality and her support, for her thoughts and her input, for those many British evenings with scones and tea and elementary entertainment, for afternoons in the park that helped to keep the balance, and generally for being the awesome person she is.

My thanks go out to Chris for his company in the coffee breaks, his companionship in the dark ages of flatspace, and for generally being a pleasant presence – every day.

They also go out to Tobi for the many years of his presence in my life, for being a critical thinker, an inspiration and a wonderful brother.

They go out to my parents who let me walk my own path and always supported me along the way.

And they go out to Riad and Geri for their support on the scientific front and for the opportunities they offered me.

To all of you: Thank you so very much!

Contents

Figures and Tables

List of Figures

List of Tables

Abbreviations, Symbols and Operators

List of Abbreviations

Notation	Description
DMP	Dynamic Movement Primitive
DoF	Degrees of Freedom
DPS	Directed Policy Search
i.i.d.	independently and identically distributed
MORE	Model-Based Relative Entropy Stochastic Search [2]
RBF	Radial Basis Function
REPS	Relative Entropy Policy Search [1]
RL	Reinforcement Learning
WML	Weighted Maximum Likelihood

List of Symbols

Notation	Description
η	a Lagrangian multiplier
λ_1	a Lagrangian multiplier
λ_2	a Lagrangian multiplier
$\boldsymbol{\mu}$	a Lagrangian multiplier
β	the trade-off factor between intrinsic motivation and reward
ϵ	the regulation factor for similarity between old and new policy
$\boldsymbol{o}$	the vector of outcomes (interesting aspects of the sensorimotor stream)
$\boldsymbol{\theta}$	the vector of parameters from a probability distribution
$\mathcal{O}$	the outcome space
$\mathcal{X}$	the sensorimotor space

Θ	the parameter space
$\mathscr{T}$	the trajectory space

List of Operators

Notation	Description	Operator
π	the current policy	$\pi(\bullet)$
q	the last policy	$q(\bullet)$
E	the empowerment of a state	$\mathrm{E}(\bullet)$
$\mathscr{H}_{\mathscr{O}}$	the entropy	$\mathscr{H}_{\mathscr{O}}(\bullet)$
KL	the Kullback-Leibler divergence	$\mathrm{KL}(\bullet)$
MI	the mutual Information	$\mathrm{MI}(\bullet)$
$\boldsymbol{\phi}$	the feature function	$\boldsymbol{\phi}(\bullet)$
J	the policy return	$J(\bullet)$
$\mathscr{R}$	the reward function	$\mathscr{R}(\bullet)$
g	the dual of the Lagrangian function	$g(\bullet)$
$\mathscr{L}$	the Lagrangian function	$\mathscr{L}(\bullet)$
exp	the exponential function	$\exp(\bullet)$
log	the natural logarithm	$\log(\bullet)$

1 Introduction

Robotics is broadly concerned with the design, control and behavior of robotic systems. The application of Reinforcement Learning (RL) to this domain is widespread and has several success stories [3, 4, 5]. In this work, we apply RL to the task of learning diverse movement policies.
RL is the machine learning approach to the idea that an individual learns when interacting with its environment. Some actions turn out to be beneficial for the individual, others do not. RL captures this relation in a problem setting where an agent perceives a state (situational information about the environment) and takes actions that change the state according to a given state transition function. In each time step the agent gets a reward that depends on the last state transition. The goal of RL algorithms is to learn a policy of which action to take in which state, so that the global cumulative reward is maximized.

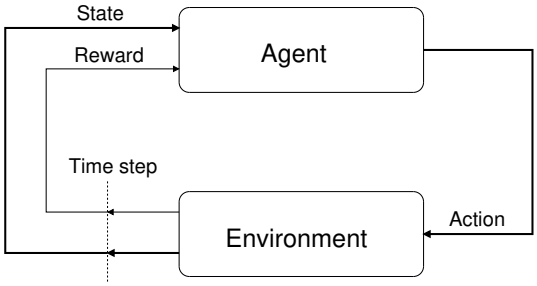

Figure 1.1: Abstract representation of RL. Taking an action jumps to the next time step and changes the state and reward signal.

Applying RL to robotic problems poses a few challenges, such as high-dimensional and continuous state-action spaces and real time constraints that conventional RL algorithms struggle with. Policy Search algorithms are an alternative that can handle these challenges. Instead of learning the value function which would require exploring the whole state-action space, Policy Search algorithms directly learn the parameters for an action policy.
This policy can either be deterministic or stochastic but even in the latter case, stochasticity has so far only been introduced for the sake of exploration [6, 7] and is typically reduced over time, yielding as a result a policy exhibiting a relative small amount of diversity.
However, diversity can be an advantage in many settings. In dynamic environments, a diverse policy is often easier to adapt to a changing environment or changing task constraints [8]. Moreover, diversity can help to avoid getting stuck in local optima. For example, in the cooperative setting of [9], a robot learns grocery checkout by interacting with a human. The human can add constraints such as not moving liquids over electronics by re-ranking the trajectories of the robot.
Unfortunately, the robot can get stuck in local optima requiring manual modification of the trajectory's waypoints [9]. Increasing the diversity in the robot's behavior can give more feedback opportunities to the human and avoids manual interventions.
Another example where diverse behaviors should be preferred are competitive settings. Here, an agent with a repetitive strategy becomes predictable and thus more easily exploitable. In robot table tennis, for example, a task can be to return incoming balls to the opponent's side of the table in a way that is hard to return for the opponent. A deterministic behavior would preclude any effect of surprise and reduce the chances of defeating the opponent. Therefore, we are interested in allowing the robot to constitute a diverse library of high reward motor skills.
In order to increase the diversity of the learned skills, we use intrinsic motivation [10] and empowerment [11]. We derive a new intrinsic motivation signal by enriching the description of a task with an outcome space. The outcome space represents interesting aspects of a sensorimotor stream, for example, in table tennis, the outcome space could be given by the return position of the ball or the return ball speed. Our intrinsic motivation signal is now given by the diversity or entropy of the outcomes of the policy.
Maximizing the entropy of the future is a concept also known as empowerment [11], therefore, we will call our approach *Empowered Skills*. Our new algorithm maximizes a trade-off between an extrinsic reward, e.g., returning the ball on the table in table tennis, and this intrinsic motivation criterion. Experiments on a planar reaching task and a simulated robot table tennis task demonstrate that our algorithm can learn a diverse set of behaviors within the area of interest of the tasks.

2 Related Work

In this chapter, we give an overview of the various concepts, algorithms and ideas that inspired this work and built its foundation.

2.1 Intrinsic Motivation

Intrinsic motivation is a term that roboticists adopted from developmental psychology where it is defined in contrast to extrinsic motivation which is at work whenever an activity is undertaken to affect a specific result whereas intrinsically motivated actions are undertaken simply for the enjoyment of the action itself.
In robotics the difference is a lot less clear as from a computational stand point both can be implemented by giving some reward for doing an action. In consequence there have been a number of different metrics described as intrinsic motivation. [10] finds that researchers commonly base their interpretation of intrinsic motivation on some properties of the sensorimotor stream and on its relation to the agent's knowledge. Another aspect common in the defined metrics is that they are independent of the specific types of sensorimotor channels involved.
Intrinsic motivation is interesting to roboticists because it allows us to build an agent with an inherent drive towards generally beneficial behavior, i.e., behavior that is overall positive regardless of the situations the robot encounters at any given time. Researchers have aimed to imbue robots with autonomy and life-like intelligence using similar systems. In recent years attempts have been made at using intrinsic motivation to implement exploration or to steer the learning process.
We can cluster intrinsic motivation into three clusters [10]: knowledge-based models, competence-based models and morphological models.

2.1.1 Knowledge-Based Models of Intrinsic Motivation

Agents of this category typically monitor and maximize a learning progress. This can be done in an information-theoretic way in which the robot builds probabilistic models of events (states, state transitions) and is rewarded for not encountering events that seem likely or for encountering events that seem unlikely or surprising.
[12, 13] for example monitor the learning progress of the transition model (predicting the next state given the current state and action). Its maximization leads to the sequencing of the learning process from the simplest to the most intricate parts of the state-action space, while ignoring the parts that have already been learned or are impossible to learn. Such an intrinsic motivation signal is also used in [14] and is mixed with an extrinsic, goal-oriented reward. However, unlike in our algorithm, the increased exploration due to the intrinsic motivation serves only the purpose of speeding-up the learning process. Moreover, its influence decreases with time and the returned policy is similar to standard RL algorithms.
An alternative approach to knowledge-based models uses predictors and monitors prediction errors to detect interesting, novel or surprising situations which are then rewarded [10].

2.1.2 Competence-Based Models of Intrinsic Motivation

Agents of this category would set challenges or goals for themselves which consist of a specific sensorimotor configuration with associated difficulty level. They would then plan their actions to achieve this goal and get a reward based on the difficulty of the challenge and their actual performance. [10] suggests three possible performance measures based on incompetence, competence and competence progress. Since then, [15] has examined using a competence-based model to support an agent's autonomous decision on what skills to learn. They measure the improvement rate of competence on the basis of the Temporal-Difference learning signal.

2.1.3 Morphological Models of Intrinsic Motivation

While the agents of the first two clusters use intrinsic motivation metrics that are based on the interaction between the learning cognitive system and the sensorimotor stream, the agents of this category use metrics that are solely based on the sensorimotor stream itself.

The cluster comprises algorithms that maximize mathematical properties of the sensorimotor space of the robot and as such is closely related to the diversity in behavioral space we are striving for.
In Evolutionary Robotics, [16] designed intrinsic online fitness functions that encourage a population of robots to seek out experiences that neither the current nor previous generations of robots had before. Thus, the algorithm grows a population of curious explorers that are motivated not by reward but by diverse actions and experiences alone.
[16] and [17] evolved neural network controllers where the typical goal-oriented fitness function is replaced by an intrinsically motivated term. Specifically, [17] introduced a distance metric in behavioral space and aimed to find behaviors that are maximally different to all the previously experienced behaviors. However, enumerating all such behaviors might not be sustainable in high-dimensional spaces.
The authors of [16] propose to search for a behavior exhibiting maximal entropy in its sensorimotor stream. For some settings, such behaviors can result in interesting solutions to a task such as navigating through a maze. Yet, high entropy behaviors do not necessarily coincide with high reward behaviors and as such a trade-off between the intrinsic and extrinsic reward is necessary.
Other properties that could be exploited include the stability of a chosen property of the sensorimotor stream as well the synchronicity of different sensorimotor channels [10].

2.2 Empowerment

The notion of empowerment, introduced in [18] and extended to the continuous case in [11], is an intrinsic motivation signal that leads an agent to seek out situations in which its perceived action potential is maximized. An empowered agent thus aims to move to states, in which its available actions appear to afford the most diverse effects.
The definition of empowerment is based on the communication problem from information theory:

2.2.1 The Communication Problem

In the communication problem there is a sender who transmits a signal to a receiver over a given channel. The sent and received signals are denoted by random variables X and Y.
The channel defines how the received signal $p(y)$ relates to sent signal $p(x)$. In the case of discrete signals it can be modeled by a conditional probability distribution $p(y|x)$. The mutual information $\mathrm{MI}(X;Y)$ is the amount of information, measured in bits, that $p(y)$ contains about $p(x)$ and is defined as

$$\mathrm{MI}(X;Y) = \sum_{X,Y} p(y|x)p(x)\log_2\left(\frac{p(y|x)}{\sum_X p(y|x)}\right).$$

The maximum information the received signal can contain about the sent signal given the channel is known as the channel capacity C with

$$C = \max_{p(x)} \mathrm{MI}(X;Y).$$

It depends solely on the channel $p(y|x)$.

2.2.2 The Definition of Empowerment

In the context of RL we can use these concepts to determine for every state s_t the maximum information the agent could inject into the environment through a series of n actions and receive again n steps later through its sensors. In other words, we can say for each state how much influence the agent's actions have on the environment as it is perceived by the agent.
To calculate this we interpret the environment as the channel $p(s_{t+n}|a_t^n)$ that connects the agent's action sequence a_t^n to its perceived state after n steps s_{t+n}. The empowerment $E(s_t)$ of a state s_t is then defined as the channel capacity

$$E(s_t) = C = \max_{p(a_t^n)} \sum_{A^n,S} p(s_{t+n}|a_t^n)p(a_t^n)\log_2\left(\frac{p(s_{t+n}|a_t^n)}{\sum_{A^n} p(s_{t+n}|a_t^n)}\right).$$

The empowerment of a state is proportional to the number of distinct states that can be reached from it. It was demonstrated in [11] that for a task such as pole balancing, the optimal swing up policy coincides with the search of the most empowered state as the upright position has the highest diversity of future states. However, similar to the discussion of [16], such behaviors do not always emerge for other tasks. As such, in our algorithm, the intrinsic motivation is not used

on its own but is combined with the extrinsic reward of the task such that we can restrict the diverse solution space to a subset of useful solutions.
Similar to [16], we measure diversity in our algorithm with an entropy function. However, as the sensorimotor stream can be excessively large, we restrict the domain of the entropy to the outcome space (Sec. 3.1) that only captures task-relevant aspects of the stream. We call our algorithm *Empowered Skills* and distinguish it from the notion of empowerment of states [18, 11] as we search for (motor) *skills* that result in diverse *outcomes*.

2.3 Policy Search

Policy Search, a subfield of RL that is especially successful in robot RL, is concerned with finding locally optimal parameterized policies. Learning the policy parameters directly without learning the value function circumvents substantial problems that arise in the context of robot RL as learning the value function requires exploration of the whole state-action space.
The popularity of policy search in robot RL does not only come from its ability to handle high-dimensional, continuous state-action spaces but also from the possibility to combine it with pre-structured policy representations, furthermore an initial estimate for the parameters can be obtained by imitation learning from expert demonstrations.

2.3.1 Model-Free and Model-Based Policy Search

Policy search algorithms can be split into model-free and model-based versions.
Model-free policy search algorithms generate sample trajectories from real robot interactions and use these to learn the policy parameters directly. This approach has a few drawbacks as the generation of each sample trajectory usually requires some level of human involvement, is time-consuming, expensive and causes wear and tear in the used robots.
To reduce these costs model-based approaches try to increase sample efficiency by first learning a forward model of the robot's dynamics and the environment from robot interaction. This model is subsequently used to generate sample trajectories through internal simulation, which is quick, cheap and can be automated. The sample trajectories are then used to learn the policy parameters.
In general, the learned model is not exact but rather an approximation of the real dynamics. Errors in the model can be exploited by policy search algorithms and can ultimately result in a policy of poor quality.
Despite the high cost associated with model-free methods, they are more widely used than model-based methods, as generating sample trajectories and learning a policy is often easier than creating an accurate forward model.

In the following two sections we will introduce the two algorithms that we use as a baseline for our experiments, as examples for these two classes of policy search methods.

2.3.2 Relative Entropy Policy Search (REPS)

Relative Entropy Policy Search [1] (REPS) is an algorithm that addresses a central limitation of policy search algorithms that are framed as optimal control problems: The risk of losing the connection to previously observed data.
In the optimal control problem a policy search algorithm finds the policy $\pi(a|s)$ which optimizes the expected return $J(\pi)$, given rewards $r(s,a)$, and the stationary state distribution $\mu^\pi(s)$ with state transition probability $p(s'|s,a)$, i.e.,

$$J(\pi) = \sum_{s,a} \mu^\pi(s)\pi(a|s)r(s,a), \tag{2.1}$$

$$\forall s' : \sum_{s,a} \mu^\pi(s)\pi(a|s)p(s'|s,a) = \mu^\pi(s'). \tag{2.2}$$

The optimization relies on μ^π and π being probability distributions as well as on the constraint of Equation 2.2.
The optimal control problem has no notion of data. Solution approaches that take bigger update steps are prone to move away from prior policies and hence lose previously gathered experience.
REPS introduces an additional constraint

$$\epsilon \geq \mathrm{KL}(p^\pi || q)$$

that limits the information loss per policy update step to a maximum of ϵ. The loss of information is measured here using the relative entropy (or Kullback-Leibler divergence) $\mathrm{KL}(p^\pi||q)$ between the observed data distribution $q(s,a)$ and the data distribution $p^\pi(s,a) = \mu^\pi(s)\pi(a|s)$ generated by the new policy π, i.e.,

$$\mathrm{KL}(p^\pi \,||\, q) = \sum_{s,a} p^\pi(s,a) \log\left(\frac{p^\pi(s,a)}{q(s,a)}\right) d\boldsymbol{\theta}.$$

Thanks to this additional constraint, the new and better policy will always stay close to the previous policy and the loss of information incurred through the optimization process will be a lot smaller.

In a more general view on REPS we have a reward $\mathscr{R}(\boldsymbol{\theta})$ and policies $\pi(\boldsymbol{\theta})$ and $q(\boldsymbol{\theta})$ all depending on a parameter vector $\boldsymbol{\theta}$. Given those we have the goal of finding a set of parameters $\boldsymbol{\theta}$ that maximize the average return, i.e.,

$$\begin{aligned} \max_{\pi} & \int \pi(\boldsymbol{\theta}) \mathscr{R}(\boldsymbol{\theta}) \, d\boldsymbol{\theta} \\ \text{s.t.} \quad & \epsilon \geq \mathrm{KL}(\pi(\boldsymbol{\theta}) \,||\, q(\boldsymbol{\theta}), \\ & 1 = \int \pi(\boldsymbol{\theta}) \, d\boldsymbol{\theta}. \end{aligned}$$

Like in the initial optimal control problem, π and q need to be probability distributions. β can be chosen freely as the task requires. A more detailed description of the algorithm can be found in [1], furthermore many related algorithms and variants are given in [6].

2.3.3 Model-Based Relative Entropy Search (MORE)

Model-Based Relative Entropy Stochastic Search [2] (MORE) enhances a stochastic search algorithm with an information-theoretic distance metric that keeps the new data-distribution close to the old one. The distance metric is not approximated from samples like in earlier work, but calculated in closed form.
A stochastic search algorithm is a black-box optimizer. The only way it has to get information about the function it aims to optimize, is to evaluate the function at chosen points. Evaluation of the function can be expensive though, for example if it involves interacting with the real world. For this reason stochastic search algorithms usually keep a multivariate Gaussian search distribution (aka policy) over the parameters of the function to represent the region in which the optimal solution is expected to be found. This policy is updated iteratively to ultimately find the optimal policy.
One subgroup of strategies to update the policy are information-theoretic approaches. These bound the relative entropy between subsequent policies. Prior to this work though, the relative entropy bound could only be applied in an approximate fashion, either by using the second-order Taylor expansion of the KL [19] or by approximating the KL using samples [1].
In contrast, MORE locally learns a quadratic model of the objective function which it can then use to analytically compute the new, relative entropy bounded policy. The model is only exploited locally and so the risk of the algorithm being misled by errors in the model is mitigated.
The benefits of using the quadratic model come at the price of a quadratic increase of the number of parameters that need to be estimated. MORE employs dimensionality reduction to alleviate this.
In addition to calculating the relative entropy analytically, MORE also employs a lower bound for new policy that aims to prevent the premature convergence of the search algorithm.
The problem that MORE solves is thus described as

$$\begin{aligned} \max_{\pi} & \int \pi(\boldsymbol{\theta}) \mathscr{R}(\boldsymbol{\theta}) \, d\boldsymbol{\theta} \\ \text{s.t.} \quad & \epsilon \geq \mathrm{KL}(\pi(\boldsymbol{\theta}) \,||\, q(\boldsymbol{\theta}), \\ & \beta \leq H(\boldsymbol{\theta}), \\ & 1 = \int \pi(\boldsymbol{\theta}) \, d\boldsymbol{\theta} \end{aligned}$$

where ϵ bounds the information loss and β is a lower bound for the entropy $H(\boldsymbol{\theta}) = -\int \pi(\boldsymbol{\theta}) \log \pi(\boldsymbol{\theta}) \, d\boldsymbol{\theta}$ of the new policy.
For a more detailed description of the algorithm and possible strategies of settings the parameters ϵ and β the interested reader is referred to [2].

3 Foundations

The Empowered Skills algorithm rests on the foundation of the Directed Policy Search (DPS) framework [6] and therefore reuses most of its terminology (Sec. 3.1). In the following sections, we describe our algorithm first for discrete and then for continuous outcome spaces.

3.1 Notation and Problem Statement

Given a reward function $\mathscr{R} : \mathscr{T} \mapsto \mathbb{R}$ mapping a robot trajectory $\tau \in \mathscr{T}$ to a real value $\mathscr{R}(\tau)$, the goal of DPS algorithms is to find a set of parameters $\boldsymbol{\theta} \in \Theta$ maximizing $\mathbb{E}_{p_\theta(\tau)}[\mathscr{R}(\tau)]$ that we denote with a slight abuse of notation as $\mathscr{R}(\boldsymbol{\theta})$. Specifically, DPS is a class of iterative algorithms maintaining search distributions over Θ that we refer to as policies[1]. The main objective of a DPS algorithm can be formulated as maximizing the policy return $J(\pi) = \mathbb{E}_{\boldsymbol{\theta}\sim\pi}[\mathscr{R}(\boldsymbol{\theta})]$.
In addition to the maximization of the reward, we introduce an intrinsic motivation term to the objective function of our algorithm with the goal to enforce diverse solutions. Similar to other intrinsically motivated algorithms [14, 16, 11, 20], diversity is measured by an entropy term. However, instead of computing the entropy over the whole sensorimotor stream $\mathscr{X}$ (like in [16]) which is typically very high-dimensional, we provide additional guidance to the algorithm by singling out relevant parts of the sensorimotor stream which we call *outcomes*.
Formally, the outcome space $\mathscr{O}$ is defined as the image of a non-injective mapping $f : \mathscr{X} \mapsto \mathscr{O}$ such that typically $\text{card}(\mathscr{O}) \ll \text{card}(\mathscr{X})$. For instance, a trajectory τ, that is part of $\mathscr{X}$ and comprised of all joint positions of the robot as well as the positions of the ball during the execution of a robot table tennis strike could retain as outcome $\boldsymbol{o} = f(\tau)$ only the 2D ball position when it hits the table after returning the ball.
Upon the definition of the outcome space, the intrinsic term is simply given[2] by the entropy $\mathscr{H}_\mathscr{O}(\pi) = -\sum_{\boldsymbol{o}} p_\pi(\boldsymbol{o}) \ln p_\pi(\boldsymbol{o})$ of the outcome probabilities $p_\pi(\boldsymbol{o}) = \int p(\boldsymbol{o}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})d\boldsymbol{\theta}$. Finally, the policy π^* returned by our algorithm is optimal with respect to a trade-off between the extrinsic reward and intrinsic motivation, i.e.,

$$\pi^* = \arg\max_\pi \left(J(\pi) + \beta \mathscr{H}_\mathscr{O}(\pi)\right),$$

with trade-off parameter β.

3.2 Empowered Skills with Finite Outcomes

Inspired by information-theoretic policy search algorithms [1, 21], we solve this optimization problem in an iterative scheme where at each iteration the new policy is updated by solving the following constrained optimization problem:

$$\begin{aligned}
&\arg\max_\pi (J(\pi) + \beta \mathscr{H}_\mathscr{O}(\pi)) \\
\text{s.t.} \quad & \epsilon \geq \text{KL}(\pi||q), && (3.1)\\
& 1 = \int \pi(\boldsymbol{\theta})\, d\boldsymbol{\theta}, && (3.2)\\
\forall \boldsymbol{o}: \quad & \hat{p}_\pi(\boldsymbol{o}) = \int p(\boldsymbol{o}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})\, d\boldsymbol{\theta}, && (3.3)\\
& 1 = \sum_{\boldsymbol{o}} \hat{p}_\pi(\boldsymbol{o}) && (3.4)
\end{aligned}$$

The Kullback-Leibler divergence $\text{KL}(\pi||q)$ is used to specify the step-size [1, 21] of the policy update and is given by

$$\text{KL}(\pi||q) = \int \pi(\boldsymbol{\theta}) \log\left(\frac{\pi(\boldsymbol{\theta})}{q(\boldsymbol{\theta})}\right) d\boldsymbol{\theta}.$$

Without the outcome entropy term $\mathscr{H}_\mathscr{O}(\pi)$ and conditions 3.3 and 3.4, this optimization problem would be a standard Policy Search problem where the information loss, given by the KL between the last and current policy, is bounded by ϵ [1, 21]. The use of KL-constraints is widespread in the robotic Reinforcement Learning (RL) community whether it is in the context of Policy Search [1], Policy Gradient [22] or Optimal Control [20].

[1] For simplicity of notation, only policies of the form $\pi(\boldsymbol{\theta})$ are considered throughout the paper. An extension to the contextual case $\pi(\boldsymbol{\theta}|\boldsymbol{c})$ for some initial i.i.d. task-dependent contexts $\boldsymbol{c} \in C$ is straightforward and similar to that of previous DPS algorithms (cf. chapter 2.4.3.2 in [6])

[2] $\mathscr{H}_\mathscr{O}(\pi)$ is given for a discrete outcome space; extension to continuous outcomes follows in Section 3.3

3.2.1 A Very Short Introduction to Lagrangian Optimization

The method of Lagrangian multipliers [23] is a strategy to find candidate solutions to constrained optimization problems, i.e., optimization problems subject to constraints given by additional formulas. The idea is to construct and then optimize the Lagrangian function $\mathscr{L}$.
Given a constrained optimization problem

$$\begin{aligned} &\arg\max_{\boldsymbol{x}} f(\boldsymbol{x}) \\ \text{s.t.} \quad &\forall i: \quad c_i(\boldsymbol{x}) = 0 \end{aligned}$$

with a target function $f(\boldsymbol{x})$ subject to constraints $c_i(\boldsymbol{x}) = 0$, one constructs the Lagrangian

$$\mathscr{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) - \sum_{i=1}^{n} \lambda_i c_i(\boldsymbol{x}) \tag{3.5}$$

by multiplying each constraint c_i by a factor λ_i known as Lagrangian multiplier, and adding it to (or subtracting it from) $f(\boldsymbol{x})$ as seen in Equation 3.5. The candidate solutions are then the saddle points of $\mathscr{L}(\boldsymbol{x}, \boldsymbol{\lambda})$ and can be found by setting its derivative equal to zero.

Example Problem with One Equality Constraint

The intuition behind Lagrangian optimization is to view $f(\boldsymbol{x})$ as returning the height over an n-dimensional space so that $f(\boldsymbol{x})$ defines a landscape the same way as an orthographic map does. Adding a constraint $c(\boldsymbol{x})$ to $f(\boldsymbol{x})$ can then be interpreted as drawing a path on the map. A hiker looking for the highest point (our optimizer) is forced to stay on the ground by $f(\boldsymbol{x})$ and is forced to stay on the path by $c(\boldsymbol{x})$. The highest point they can reach is thus a point where the path is parallel to the contour lines of the map. In that case the gradients of $f(\boldsymbol{x})$ and $c(\boldsymbol{x})$ are parallel as well and thus

$$\Delta f(\boldsymbol{x}) = \lambda \Delta c(\boldsymbol{x}).$$

The Lagrangian multiplier λ accounts for the possibly different magnitudes of the gradients.
To solve the problem we introduce the Lagrangian

$$\mathscr{L}(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) - \lambda c(\boldsymbol{x})$$

and set its derivative equal to zero

$$\Delta \mathscr{L}(\boldsymbol{x}, \lambda) = \Delta f(\boldsymbol{x}) - \lambda \Delta c(\boldsymbol{x}) = 0$$

which is equivalent to searching for points at which both gradients are parallel – the saddle points which are the candidate solutions to the constrained optimization problem.
Multiple constraints can be added straight-forwardly as seen in Equation 3.5.

Lagrange Duality

To find solution candidates while allowing inequality constraints, one can make use of the Lagrange Duality. For a given optimization problem

$$\begin{aligned} &\arg\min_{\boldsymbol{x}} f(\boldsymbol{x}) \\ \text{s.t.} \quad &\forall i: \quad c_i(\boldsymbol{x}) = 0, \\ &\forall j: \quad d_j(\boldsymbol{x}) \leq 0, \end{aligned}$$

the Lagrangian is

$$\mathscr{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \sum_{i=1}^{n} \lambda_i c_i(\boldsymbol{x}) + \sum_{j=1}^{m} \mu_j d_j(\boldsymbol{x}) \tag{3.6}$$

and its dual function is

$$g(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \min_{\boldsymbol{x}} \mathscr{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}). \tag{3.7}$$

The dual function is a lower bound of the Lagrangian. In case of hard duality, which is subject to several constraint qualifications, the maximum of the dual is equal to the minimum of the Lagrangian.

The duality of both functions motivates the formulation of the dual problem

$$\begin{aligned} &\arg\max_{\boldsymbol{\lambda},\boldsymbol{\mu}} g(\boldsymbol{\lambda},\boldsymbol{\mu}) \\ \text{s.t.} \quad &\forall j: \quad \mu_j \geq 0, \end{aligned}$$

which is defined as maximizing the dual function under the constraint that the Lagrangian multipliers belonging to the inequality constraints are positive.
Constructing and solving the dual problem is especially rewarding, as the dual function is always concave, which makes the dual problem convex. This guarantees global optimal solutions for algorithms like gradient ascent, that generally only guarantee local optimality.

The reader interested in a more detailed and in-depth discussion of this matter is kindly referred to [24], where the discussion of the Lagrangian methods starts with chapter four.

3.2.2 The Entropy Term adds Complications

The addition of the outcome entropy term $\mathcal{H}_{\mathcal{O}}(\pi)$ introduces an interdependency between $p_\pi(\boldsymbol{o})$ and π. This interdependency prohibits the derivation of a closed form policy update directly from the constrained optimization problem[3]. A set of auxiliary variables $\hat{p}_\pi(\boldsymbol{o})$ that we optimize for is therefore introduced to break this dependency yielding $\mathcal{H}_{\mathcal{O}}(\pi) = -\sum_{\boldsymbol{o}} \hat{p}_\pi(\boldsymbol{o}) \ln \hat{p}_\pi(\boldsymbol{o})$. For a finite (and relatively small outcome space) constraints 3.3 and 3.4 then allow us to enforce for all $\boldsymbol{o} \in \mathcal{O}$ the equality[4] $\hat{p}_\pi(\boldsymbol{o}) = \int p(\boldsymbol{o}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})\, d\boldsymbol{\theta}$, resulting in the closed form policy update

$$\pi(\boldsymbol{\theta}) \propto q(\boldsymbol{\theta}) \exp\left(\frac{1}{\eta}\left(\mathcal{R}(\boldsymbol{\theta}) + \sum_{\boldsymbol{o}} \mu_{\boldsymbol{o}}\, p(\boldsymbol{o}|\boldsymbol{\theta})\right)\right),$$

where μ_o are the Lagrangian multipliers for the constraints given in Equation 3.3. In comparison to the standard solution for DPS (cf. the episodic REPS algorithm given in [21]) we can identify the term $\sum_{\boldsymbol{o}} \mu_{\boldsymbol{o}}\, p(\boldsymbol{o}|\boldsymbol{\theta})$, which is added to the reward function $\mathcal{R}(\boldsymbol{\theta})$ as intrinsic motivation. This term depends on the Lagrangian multipliers $\mu_{\boldsymbol{o}}$ that are obtained by optimizing the dual function of the optimization problem (Eq. 3.28).

3.3 Empowered Skills with Continuous Outcomes

In the continuous case, adding a constraint for every possible outcome is impossible. As a consequence, we need to resort to matching feature expectations instead of single probability values. Hence, we replace the constraints 3.3 and 3.4 with

$$\begin{aligned} \int \boldsymbol{\phi}(\boldsymbol{o})\, \hat{p}_\pi(\boldsymbol{o})\, d\boldsymbol{o} &= \int \boldsymbol{\phi}(\boldsymbol{o}) \int p(\boldsymbol{o}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})\, d\boldsymbol{\theta}\, d\boldsymbol{o}, \\ 1 &= \int \hat{p}_\pi(\boldsymbol{o})\, d\boldsymbol{o}. \end{aligned}$$

The expression of $\mathcal{H}_{\mathcal{O}}(\pi)$ in the continuous case is simply obtained by replacing the sum over the domain $\mathcal{O}$ by an integral. The full optimization problem is then given by

$$\begin{aligned} &\arg\max_{\pi} (J(\pi) + \beta \mathcal{H}_{\mathcal{O}}(\pi)) \\ \text{s.t.} \quad &\epsilon \geq \mathrm{KL}(\pi||q), && (3.8) \\ &\int \boldsymbol{\phi}(\boldsymbol{o})\, \hat{p}_\pi(\boldsymbol{o})\, d\boldsymbol{o} = \int \boldsymbol{\phi}(\boldsymbol{o}) \int p(\boldsymbol{o}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})\, d\boldsymbol{\theta}\, d\boldsymbol{o}, && (3.9) \\ &1 = \int \pi(\boldsymbol{\theta})\, d\boldsymbol{\theta}, && (3.10) \\ &1 = \int \hat{p}_\pi(\boldsymbol{o})\, d\boldsymbol{o}. && (3.11) \end{aligned}$$

[3] The interdependency results in a log-sum expression in the Lagrangian that cannot be solved in closed form.

[4] Within the internal optimization routine of a policy update, a sample estimate of the r.h.s. $\int p(\boldsymbol{o}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})\, d\boldsymbol{\theta}$ of each equality constraint needs to be computed for different π. This can be done solely from data generated in previous iterations using importance sampling. However, we do not elaborate further as these constraints will not appear in the continuous outcome case.

3.4 Solving the Optimization Problem

The policy update can now again be obtained by Lagrangian optimization and is given by

$$\pi(\boldsymbol{\theta}) \propto q(\boldsymbol{\theta}) \exp\left(\frac{1}{\eta}\delta_{\mu}(\boldsymbol{\theta})\right).$$

where $\boldsymbol{\mu}$ is a vector of Lagrangian multipliers for the constraint given in Equation 3.9, η is the Lagrangian multiplier for the KL constraint given in Equation 3.8, and

$$\delta_{\mu}(\boldsymbol{\theta}) = \mathscr{R}(\boldsymbol{\theta}) + \boldsymbol{\mu} \int p(\boldsymbol{o}|\boldsymbol{\theta})\boldsymbol{\phi}(\boldsymbol{o})\, d\boldsymbol{o},$$

is an abbreviation we introduce for reasons of brevity and clarity in later parts of this work.

The Lagrangian multipliers η and $\boldsymbol{\mu}$ on which the policy update depends, can be determined by minimizing the dual function which will be derived in the next section.

3.4.1 Derivation of the Empowered Skills Algorithm

To get the dual function we first formulate the Lagrangian of the constrained optimization problem

$$\begin{aligned}\mathscr{L}(\boldsymbol{\theta}, \boldsymbol{\mu}, \eta, \lambda_1, \lambda_2) = &\int \pi(\boldsymbol{\theta})\mathscr{R}(\boldsymbol{\theta})\, d\boldsymbol{\theta} - \beta \int \hat{p}_{\pi}(\boldsymbol{o}) \log(\hat{p}_{\pi}(\boldsymbol{o}))\, d\boldsymbol{o} \\ &+ \boldsymbol{\mu}\left[\int \left(\boldsymbol{\phi}(\boldsymbol{o}) \int p(\boldsymbol{o}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})\, d\boldsymbol{\theta} - \boldsymbol{\phi}(\boldsymbol{o})\hat{p}_{\pi}(\boldsymbol{o})\right) d\boldsymbol{o}\right] \\ &+ \lambda_1\left(1 - \int \pi(\boldsymbol{\theta})\, d\boldsymbol{\theta}\right) + \lambda_2\left(1 - \int \hat{p}_{\pi}(\boldsymbol{o})\, d\boldsymbol{o}\right) + \eta\left(\epsilon - \int \pi(\boldsymbol{\theta}) \log\left(\frac{\pi(\boldsymbol{\theta})}{q(\boldsymbol{\theta})}\right) d\boldsymbol{\theta}\right),\end{aligned} \tag{3.12}$$

and its derivatives

$$\partial_{\pi(\boldsymbol{\theta})}\mathscr{L} = \mathscr{R}(\boldsymbol{\theta}) - \lambda_1 - \eta \log\left(\frac{\pi(\boldsymbol{\theta})}{q(\boldsymbol{\theta})}\right) - \eta + \boldsymbol{\mu} \int \boldsymbol{\phi}(\boldsymbol{o}) p(\boldsymbol{o}|\boldsymbol{\theta})\, d\boldsymbol{o}, \tag{3.13}$$

$$\partial_{\boldsymbol{\mu}}\mathscr{L} = \int \boldsymbol{\phi}(\boldsymbol{o}) \int p(\boldsymbol{o}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})\, d\boldsymbol{\theta}\, d\boldsymbol{o} - \int \boldsymbol{\phi}(\boldsymbol{o})\hat{p}_{\pi}(\boldsymbol{o})\, d\boldsymbol{o}, \tag{3.14}$$

$$\partial_{\hat{p}_{\pi}(\boldsymbol{o})}\mathscr{L} = -\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o}) - \beta\left(\log(\hat{p}_{\pi}(\boldsymbol{o})) + 1\right) - \lambda_2, \tag{3.15}$$

$$\partial_{\lambda_1}\mathscr{L} = 1 - \int \pi(\boldsymbol{\theta})\, d\boldsymbol{\theta}, \tag{3.16}$$

$$\partial_{\lambda_2}\mathscr{L} = 1 - \int \hat{p}_{\pi}(\boldsymbol{o})\, d\boldsymbol{o}, \tag{3.17}$$

$$\partial_{\eta}\mathscr{L} = \epsilon - \int \pi(\boldsymbol{\theta}) \log\left(\frac{\pi(\boldsymbol{\theta})}{q(\boldsymbol{\theta})}\right) d\boldsymbol{\theta}. \tag{3.18}$$

Formulating the Dual Function

We then rewrite the Lagrangian in the more convenient form

$$\begin{aligned}\mathscr{L}(\boldsymbol{\theta}, \boldsymbol{\mu}, \eta, \lambda_1, \lambda_2) = \eta\epsilon + \eta\left(1 - \left(1 - \int \pi(\boldsymbol{\theta})\, d\boldsymbol{\theta}\right)\right) + \lambda_1 - \boldsymbol{\mu} \int \boldsymbol{\phi}(\boldsymbol{o})\hat{p}_{\pi}(\boldsymbol{o})\, d\boldsymbol{o} - \beta \int \hat{p}_{\pi}(\boldsymbol{o}) \log(\hat{p}_{\pi}(\boldsymbol{o}))\, d\boldsymbol{o} \\ + \lambda_2\left(1 - \int \hat{p}_{\pi}(\boldsymbol{o})\, d\boldsymbol{o}\right) + \int \pi(\boldsymbol{\theta})\left[\mathscr{R}(\boldsymbol{\theta}) - \lambda_1 - \eta \log\left(\frac{\pi(\boldsymbol{\theta})}{q(\boldsymbol{\theta})}\right) - \eta + \boldsymbol{\mu} \int \boldsymbol{\phi}(\boldsymbol{o}) p(\boldsymbol{o}|\boldsymbol{\theta})\, d\boldsymbol{o}\right] d\boldsymbol{\theta},\end{aligned}$$

collecting all the various factors of $\pi(\boldsymbol{\theta})$ in one place. In this form it becomes apparent that $\partial_{\pi(\boldsymbol{\theta})}\mathscr{L}$, $\partial_{\lambda_1}\mathscr{L}$, and $\partial_{\lambda_2}\mathscr{L}$ (Eq. 3.13, 3.16, and 3.17) are parts of $\mathscr{L}(\boldsymbol{\theta},\boldsymbol{\mu},\eta,\lambda_1,\lambda_2)$. Since the derivatives are equal to zero at the maximum[5], we can drop the respective parts when constructing the dual

$$\begin{aligned} g(\boldsymbol{\mu},\eta,\lambda_1) &= \max_{\boldsymbol{\theta}} \mathscr{L}(\boldsymbol{\theta},\boldsymbol{\mu},\eta,\lambda_1,\lambda_2) \\ &= \eta\epsilon + \eta\,(1-(\mathbf{0})) + \lambda_1 \; -\boldsymbol{\mu}\int \boldsymbol{\phi}(\boldsymbol{o})\hat{p}_\pi(\boldsymbol{o})\,d\boldsymbol{o} - \beta\int \hat{p}_\pi(\boldsymbol{o})\log(\hat{p}_\pi(\boldsymbol{o}))\,d\boldsymbol{o} + \lambda_2\,(\mathbf{0}) + \int \pi(\boldsymbol{\theta})\,[\mathbf{0}]\,d\boldsymbol{\theta} \\ &= \eta\epsilon + \eta + \lambda_1 \qquad\quad -\boldsymbol{\mu}\int \boldsymbol{\phi}(\boldsymbol{o})\hat{p}_\pi(\boldsymbol{o})\,d\boldsymbol{o} - \beta\int \hat{p}_\pi(\boldsymbol{o})\log(\hat{p}_\pi(\boldsymbol{o}))\,d\boldsymbol{o}. \end{aligned} \tag{3.19}$$

This leaves us with the unknown factor $\hat{p}_\pi(\boldsymbol{o})$ which we can replace with the formula we get by reformulating $\partial_{\hat{p}_\pi(\boldsymbol{o})}\mathscr{L}$ (Eq. 3.15), i.e.,

$$\begin{aligned} & & \partial_{\hat{p}_\pi(\boldsymbol{o})}\mathscr{L} &= -\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o}) - \beta\,(\log(\hat{p}_\pi(\boldsymbol{o})) + 1) - \lambda_2 \\ \iff & & 0 &= -\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o}) - \beta\,(\log(\hat{p}_\pi(\boldsymbol{o})) + 1) - \lambda_2 \\ \iff & & \beta\,(\log(\hat{p}_\pi(\boldsymbol{o})) + 1) &= -\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o}) - \lambda_2 \\ \iff & & \log(\hat{p}_\pi(\boldsymbol{o})) &= \frac{-\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o}) - \lambda_2}{\beta} - 1 \\ \iff & & \hat{p}_\pi(\boldsymbol{o}) &= \exp\left(\frac{-\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o}) - \lambda_2}{\beta} - 1\right). \end{aligned} \tag{3.20}$$

We insert our new formula for $\hat{p}_\pi(\boldsymbol{o})$ into $g(\boldsymbol{\mu},\eta,\lambda_1)$ (Eq. 3.19) and reduce the result to get $g(\boldsymbol{\mu},\eta,\lambda_1,\lambda_2)$, i.e.,

$$\begin{aligned} g(\boldsymbol{\mu},\eta,\lambda_1,\lambda_2) &= \eta\epsilon + \eta + \lambda_1 - \boldsymbol{\mu}\int \boldsymbol{\phi}(\boldsymbol{o})\hat{p}_\pi(\boldsymbol{o})\,d\boldsymbol{o} - \beta\int \hat{p}_\pi(\boldsymbol{o})\left(\frac{-\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o}) - \lambda_2}{\beta} - 1\right)d\boldsymbol{o} \\ &= \eta\epsilon + \eta + \lambda_1 - \boldsymbol{\mu}\int \boldsymbol{\phi}(\boldsymbol{o})\hat{p}_\pi(\boldsymbol{o})\,d\boldsymbol{o} - \beta\int \hat{p}_\pi(\boldsymbol{o})\left(\frac{-\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o}) - \lambda_2 - \beta}{\beta}\right)d\boldsymbol{o} \\ &= \eta\epsilon + \eta + \lambda_1 - \boldsymbol{\mu}\int \boldsymbol{\phi}(\boldsymbol{o})\hat{p}_\pi(\boldsymbol{o})\,d\boldsymbol{o} + \int \hat{p}_\pi(\boldsymbol{o})\,(\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o}) + \lambda_2 + \beta)\,d\boldsymbol{o} \\ &= \eta\epsilon + \eta + \lambda_1 - \boldsymbol{\mu}\int \boldsymbol{\phi}(\boldsymbol{o})\hat{p}_\pi(\boldsymbol{o})\,d\boldsymbol{o} + \boldsymbol{\mu}\int \boldsymbol{\phi}(\boldsymbol{o})\hat{p}_\pi(\boldsymbol{o})\,d\boldsymbol{o} + \lambda_2\int \hat{p}_\pi(\boldsymbol{o})\,d\boldsymbol{o} + \beta\int \hat{p}_\pi(\boldsymbol{o})\,d\boldsymbol{o} \\ &= \eta\epsilon + \eta + \lambda_1 + \lambda_2\int \hat{p}_\pi(\boldsymbol{o})\,d\boldsymbol{o} + \beta\int \hat{p}_\pi(\boldsymbol{o})\,d\boldsymbol{o}. \end{aligned} \tag{3.21}$$

We can now replace the two integrals by 1 because of $\partial_{\lambda_2}\mathscr{L}$ (Eq. 3.17). This leaves us for a short while with a much more compact version of $g(\boldsymbol{\mu},\eta,\lambda_1,\lambda_2)$ that now has the disadvantage of relying on the additional Lagrangian multiplier λ_2, i.e.,

$$g(\boldsymbol{\mu},\eta,\lambda_1,\lambda_2) = \eta\epsilon + \eta + \lambda_1 + \lambda_2 + \beta. \tag{3.22}$$

We can further simplify the optimization problem by eliminating the two Lagrangian multipliers λ_1 and λ_2.

Losing the Dependency on λ_1

We start with λ_1 which we can eliminate via a little detour. First, we rewrite $g(\boldsymbol{\mu},\eta,\lambda_1,\lambda_2)$ into the more convenient form

$$g(\boldsymbol{\mu},\eta,\lambda_1,\lambda_2) = \eta\epsilon + \eta\log\left(\exp\left(\frac{\lambda_1+\eta}{\eta}\right)\right) + \lambda_2 + \beta. \tag{3.23}$$

[5] To align this formulation with the standard problem as seen in Section 3.2.1, convert it following $\max_{\boldsymbol{x}} f(\boldsymbol{x}) = min_{\boldsymbol{x}} -f(\boldsymbol{x})$.

This will allow us later to insert the modified $\partial_{\lambda_1}\mathscr{L}$, but first we have to reformulate $\partial_{\pi(\boldsymbol{\theta})}\mathscr{L}$ (Eq. 3.13)

$$
\begin{aligned}
& \partial_{\pi(\boldsymbol{\theta})}\mathscr{L} = \mathscr{R}(\boldsymbol{\theta}) - \lambda_1 - \eta \log\left(\frac{\pi(\boldsymbol{\theta})}{q(\boldsymbol{\theta})}\right) - \eta + \boldsymbol{\mu}\int \boldsymbol{\phi}(\boldsymbol{o}) p(\boldsymbol{o}|\boldsymbol{\theta})\, d\boldsymbol{o} \\
\iff \quad & 0 = \mathscr{R}(\boldsymbol{\theta}) - \lambda_1 - \eta \log\left(\frac{\pi(\boldsymbol{\theta})}{q(\boldsymbol{\theta})}\right) - \eta + \boldsymbol{\mu}\int \boldsymbol{\phi}(\boldsymbol{o}) p(\boldsymbol{o}|\boldsymbol{\theta})\, d\boldsymbol{o} \\
\iff \quad & \eta \log\left(\frac{\pi(\boldsymbol{\theta})}{q(\boldsymbol{\theta})}\right) = \mathscr{R}(\boldsymbol{\theta}) - \lambda_1 - \eta + \boldsymbol{\mu}\int \boldsymbol{\phi}(\boldsymbol{o}) p(\boldsymbol{o}|\boldsymbol{\theta})\, d\boldsymbol{o} \\
\iff \quad & \log\left(\frac{\pi(\boldsymbol{\theta})}{q(\boldsymbol{\theta})}\right) = \frac{\mathscr{R}(\boldsymbol{\theta}) + \boldsymbol{\mu}\int \boldsymbol{\phi}(\boldsymbol{o}) p(\boldsymbol{o}|\boldsymbol{\theta})\, d\boldsymbol{o}}{\eta} + \frac{-\lambda_1 - \eta}{\eta} \\
\iff \quad & \frac{\pi(\boldsymbol{\theta})}{q(\boldsymbol{\theta})} = \exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right)\exp\left(\frac{-\lambda_1 - \eta}{\eta}\right) \\
\iff \quad & \pi(\boldsymbol{\theta}) = q(\boldsymbol{\theta})\exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right)\exp\left(\frac{-\lambda_1 - \eta}{\eta}\right),
\end{aligned}
\tag{3.24}
$$

which results in the policy update formula (Eq. 3.24).

We can insert Equation 3.24 into $\partial_{\lambda_1}\mathscr{L}$ (Eq. 3.16) and restructure it to fit into Equation 3.23, i.e.,

$$
\begin{aligned}
& \partial_{\lambda_1}\mathscr{L} = 1 - \int \pi(\boldsymbol{\theta})\, d\boldsymbol{\theta} \\
\iff \quad & 0 = 1 - \int \pi(\boldsymbol{\theta})\, d\boldsymbol{\theta} \\
\iff \quad & 0 = 1 - \int q(\boldsymbol{\theta})\exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right)\exp\left(\frac{-\lambda_1 - \eta}{\eta}\right) d\boldsymbol{\theta} \\
\iff \quad & 1 = \int q(\boldsymbol{\theta})\exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right)\exp\left(\frac{-\lambda_1 - \eta}{\eta}\right) d\boldsymbol{\theta} \\
\iff \quad & \frac{1}{\exp\left(\frac{-\lambda_1 - \eta}{\eta}\right)} = \int q(\boldsymbol{\theta})\exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) d\boldsymbol{\theta} \\
\iff \quad & \exp\left(\frac{\lambda_1 + \eta}{\eta}\right) = \int q(\boldsymbol{\theta})\exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) d\boldsymbol{\theta}.
\end{aligned}
\tag{3.25}
$$

After inserting the result into Equation 3.23 we get

$$
g(\boldsymbol{\mu}, \eta, \lambda_2) = \eta\epsilon + \lambda_2 + \beta + \eta \log\left(\int q(\boldsymbol{\theta})\exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) d\boldsymbol{\theta}\right). \tag{3.26}
$$

The dual is now independent of λ_1.

Losing the Dependency on λ_2

To also eliminate λ_2 we start with our formula for $\partial_{\lambda_2}\mathscr{L}$ (Eq. 3.17), insert the formula for $\hat{p}_\pi(\boldsymbol{o})$ (Eq. 3.20) and restructure, i.e.,

$$\begin{aligned}
&\partial_{\lambda_2}\mathscr{L} = 1 - \int \hat{p}_\pi(\boldsymbol{o})\, d\boldsymbol{o} \\
\iff \quad & 0 = 1 - \int \hat{p}_\pi(\boldsymbol{o})\, d\boldsymbol{o} \\
\iff \quad & 1 = \int \exp\left(\frac{-\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o}) - \lambda_2}{\beta} - 1\right) d\boldsymbol{o} \\
\iff \quad & 1 = \int \exp\left(\frac{-\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o}) - \lambda_2 - \beta}{\beta}\right) d\boldsymbol{o} \\
\iff \quad & 1 = \exp\left(\frac{-\lambda_2 - \beta}{\beta}\right) \int \exp\left(\frac{-\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o})}{\beta}\right) d\boldsymbol{o} \\
\iff \quad & \exp\left(\frac{\lambda_2 + \beta}{\beta}\right) = \int \exp\left(\frac{-\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o})}{\beta}\right) d\boldsymbol{o} \\
\iff \quad & \lambda_2 + \beta = \beta \log\left(\int \exp\left(\frac{-\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o})}{\beta}\right) d\boldsymbol{o}\right).
\end{aligned} \tag{3.27}$$

After this reformulation we can plug the result neatly into our formula for $g(\boldsymbol{\mu}, \eta, \lambda_2)$ (Eq. 3.26), again removing the dependency on λ_2:

$$g(\eta, \boldsymbol{\mu}) = \eta\epsilon + \eta \log\left(\int q(\boldsymbol{\theta}) \exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) d\boldsymbol{\theta}\right) + \beta \log\left(\int \exp\left(\frac{-\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o})}{\beta}\right) d\boldsymbol{o}\right). \tag{3.28}$$

Now optimizing the dual only depends on two parameters, η and $\boldsymbol{\mu}$, which makes the optimization problem significantly easier as the space we have to search to find the optimum is now only 2-dimensional.

The Dual's Derivatives

To optimize the dual $g(\eta, \boldsymbol{\mu})$ we need its partial derivatives. We start with ∂_η:

$$\begin{aligned}
\partial_\eta g &= \epsilon + \log\left(\int q(\boldsymbol{\theta}) \exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) d\boldsymbol{\theta}\right) + \eta \partial_\eta \log\left(\int q(\boldsymbol{\theta}) \exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) d\boldsymbol{\theta}\right) \\
&= \epsilon + \log\left(\int q(\boldsymbol{\theta}) \exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) d\boldsymbol{\theta}\right) + \eta \frac{\partial_\eta \left(\int q(\boldsymbol{\theta}) \exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) d\boldsymbol{\theta}\right)}{\int q(\boldsymbol{\theta}) \exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) d\boldsymbol{\theta}} \\
&= \epsilon + \log\left(\int q(\boldsymbol{\theta}) \exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) d\boldsymbol{\theta}\right) + \eta \frac{\int q(\boldsymbol{\theta}) \exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) \partial_\eta \left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) d\boldsymbol{\theta}}{\int q(\boldsymbol{\theta}) \exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) d\boldsymbol{\theta}} \\
&= \epsilon + \log\left(\int q(\boldsymbol{\theta}) \exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) d\boldsymbol{\theta}\right) + \eta \frac{\int q(\boldsymbol{\theta}) \exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) \frac{-\delta_\mu(\boldsymbol{\theta})}{\eta^2} d\boldsymbol{\theta}}{\int q(\boldsymbol{\theta}) \exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) d\boldsymbol{\theta}} \\
&= \epsilon + \log\left(\int q(\boldsymbol{\theta}) \exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) d\boldsymbol{\theta}\right) - \eta \frac{\int \left(q(\boldsymbol{\theta}) \exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) \frac{\delta_\mu(\boldsymbol{\theta})}{\eta^2}\right) d\boldsymbol{\theta}}{\int q(\boldsymbol{\theta}) \exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) d\boldsymbol{\theta}}
\end{aligned} \tag{3.29}$$

and end with ∂_μ:

$$\begin{aligned}
\partial_\mu g &= \eta \partial_\mu \log\left(\int q(\boldsymbol{\theta}) \exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) d\boldsymbol{\theta}\right) && + \beta \partial_\mu \log\left(\int \exp\left(\frac{-\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o})}{\beta}\right) d\boldsymbol{o}\right) \\
&= \eta \frac{\partial_\mu\left(\int q(\boldsymbol{\theta}) \exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) d\boldsymbol{\theta}\right)}{\int q(\boldsymbol{\theta}) \exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) d\boldsymbol{\theta}} && + \beta \frac{\partial_\mu\left(\int \exp\left(\frac{-\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o})}{\beta}\right) d\boldsymbol{o}\right)}{\int \exp\left(\frac{-\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o})}{\beta}\right) d\boldsymbol{o}} \\
&= \eta \frac{\int q(\boldsymbol{\theta}) \exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) \partial_\mu\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) d\boldsymbol{\theta}}{\int q(\boldsymbol{\theta}) \exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) d\boldsymbol{\theta}} && + \beta \frac{\int \exp\left(\frac{-\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o})}{\beta}\right) \partial_\mu\left(\frac{-\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o})}{\beta}\right) d\boldsymbol{o}}{\int \exp\left(\frac{-\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o})}{\beta}\right) d\boldsymbol{o}} \\
&= \eta \frac{\int q(\boldsymbol{\theta}) \exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) \frac{\int p(\boldsymbol{o}|\boldsymbol{\theta}) \boldsymbol{\phi}(\boldsymbol{o}) d\boldsymbol{o}}{\eta} d\boldsymbol{\theta}}{\int q(\boldsymbol{\theta}) \exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) d\boldsymbol{\theta}} && + \beta \frac{\int \exp\left(\frac{-\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o})}{\beta}\right) \left(\frac{-\boldsymbol{\phi}(\boldsymbol{o})}{\beta}\right) d\boldsymbol{o}}{\int \exp\left(\frac{-\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o})}{\beta}\right) d\boldsymbol{o}} \\
&= \frac{\int \left(q(\boldsymbol{\theta}) \exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) \int p(\boldsymbol{o}|\boldsymbol{\theta}) \boldsymbol{\phi}(\boldsymbol{o}) d\boldsymbol{o}\right) d\boldsymbol{\theta}}{\int q(\boldsymbol{\theta}) \exp\left(\frac{\delta_\mu(\boldsymbol{\theta})}{\eta}\right) d\boldsymbol{\theta}} - \frac{\int \exp\left(\frac{-\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o})}{\beta}\right) \boldsymbol{\phi}(\boldsymbol{o}) d\boldsymbol{o}}{\int \exp\left(\frac{-\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o})}{\beta}\right) d\boldsymbol{o}}.
\end{aligned} \tag{3.30}$$

Now that we have the derivatives of the dual function, we can optimize the dual to get $\boldsymbol{\mu}$ and η, the Lagrangian multipliers for the feature expectation constraints and the KL constraint.

3.4.2 Calculating the Weights

Similar to other DPS algorithms such as episodic REPS [21], we can only solve this optimization problem for a finite set of samples. The result of this optimization is then given by a weighting

$$w_i = \exp\left(\frac{1}{\eta}(r_i + \boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o}_i))\right)$$

for each sample $\boldsymbol{\theta}_i$. These weights are subsequently used to estimate a new parametric policy using a Weighted Maximum Likelihood (WML) estimate [21].

3.5 Estimating the New Policy

Because of their simplicity, we use multivariate Gaussian distributions for our policies. This is also a common assumption in DPS. The mean and covariance matrix of the new policy π are given by the WML estimator, i.e.,

$$\begin{aligned}
\boldsymbol{\mu}_\pi &= \frac{\sum w_i \boldsymbol{\theta}_i}{\sum w_i}, \\
\boldsymbol{\Sigma}_\pi &= \frac{\sum w_i (\boldsymbol{\theta}_i - \boldsymbol{\mu}_\pi)(\boldsymbol{\theta}_i - \boldsymbol{\mu}_\pi)^T}{Z}, \\
\text{where} \qquad Z &= \frac{\left(\sum w_i\right)^2 - \sum w_i^2}{\sum w_i},
\end{aligned}$$

which uses samples θ_i that are drawn from the previous policy q.

3.6 The Resulting Algorithm

Table 3.1 gives an overview of the Empowered Skills algorithm. Before we can start the policy iteration, we have to select the initial parameters and policy. We select the maximal relative entropy between two consecutive policies by setting the parameter ϵ and choose a β to define the balance between the reward and the outcome entropy. Lastly, we need an initial policy, which we can create using imitation learning and samples from an expert.
With the initial parameters and policy set, we begin the policy iteration. Each iteration step starts with generating sample trajectories $\boldsymbol{\theta}_i$ and the corresponding outcomes $\boldsymbol{o}_i$ and rewards r_i.
Given these values, we can calculate the features $\boldsymbol{\phi}(\boldsymbol{o})$ and optimize the dual function of the policy update (Eq. (3.28)) to get η and $\boldsymbol{\mu}$.
As seen in Section 3.4.2, we need the dual parameters to calculate the weights w_i for the newly generated trajectory samples. With the weights we can then fit the new policy π by WML. The next iteration of the policy search can begin.
The number of iterations can be fixed in advance. Alternatively, the algorithm can be run until a high-enough reward or a minimum desired entropy are reached.

Empowered Skills

Input: Maximal information loss ϵ, entropy weight β, initial policy $q(\boldsymbol{\theta})$

for each policy update

Sampling: Obtain samples $(\boldsymbol{\theta}_i, \boldsymbol{o}_i, r_i)$.

Critic: Evaluate policy for θ.

Define Error Function:

$$\delta_\mu(\boldsymbol{\theta}) = R(\boldsymbol{\theta}) + \boldsymbol{\mu} \int p(\boldsymbol{o}|\boldsymbol{\theta}) \boldsymbol{\phi}(\boldsymbol{o})\, d\boldsymbol{o}$$

Compute Dual Function:

$$g(\eta, \boldsymbol{\mu}) = \eta\epsilon + \eta \log\left(\int q(\boldsymbol{\theta}) \exp\left(\frac{1}{\eta} \delta_\mu(\boldsymbol{\theta}) \right) d\boldsymbol{\theta} \right) + \beta \log\left(\int \exp\left(\frac{-\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o})}{\beta} \right) d\boldsymbol{o} \right)$$

Compute Dual Function's Derivative:

$$\partial_\eta g = \epsilon + \log\left(\int q(\boldsymbol{\theta}) \exp\left(\frac{1}{\eta} \delta_\mu(\boldsymbol{\theta}) \right) d\boldsymbol{\theta} \right) - \eta \frac{\int \left(q(\boldsymbol{\theta}) \exp\left(\frac{1}{\eta} \delta_\mu(\boldsymbol{\theta}) \right) \frac{\delta_\mu(\boldsymbol{\theta})}{\eta^2} \right) d\boldsymbol{\theta}}{\int q(\boldsymbol{\theta}) \exp\left(\frac{1}{\eta} \delta_\mu(\boldsymbol{\theta}) \right) d\boldsymbol{\theta}}$$

$$\partial_\mu g = \frac{\int \left(q(\boldsymbol{\theta}) \exp\left(\frac{1}{\eta} \delta_\mu(\boldsymbol{\theta}) \right) \int p(\boldsymbol{o}|\boldsymbol{\theta}) \boldsymbol{\phi}(\boldsymbol{o})\, d\boldsymbol{o} \right) d\boldsymbol{\theta}}{\int q(\boldsymbol{\theta}) \exp\left(\frac{1}{\eta} \delta_\mu(\boldsymbol{\theta}) \right) d\boldsymbol{\theta}} - \frac{\int \exp\left(\frac{-\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o})}{\beta} \right) \boldsymbol{\phi}(\boldsymbol{o})\, d\boldsymbol{o}}{\int \exp\left(\frac{-\boldsymbol{\mu}\boldsymbol{\phi}(\boldsymbol{o})}{\beta} \right) d\boldsymbol{o}}$$

Optimize: $(\eta^*, \boldsymbol{\mu}^*) = \texttt{fmin\_BFGS}(g, \partial g, \eta_0, \boldsymbol{\mu}_0)$

Actor: Compute new policy $\pi(\boldsymbol{\theta})$.

$$\pi(\boldsymbol{\theta}) = \frac{q(\boldsymbol{\theta}) \exp\left(\frac{1}{\eta^*} \delta_{\mu^*}(\boldsymbol{\theta}) \right)}{\int q(\boldsymbol{\theta}) \exp\left(\frac{1}{\eta^*} \delta_{\mu^*}(\boldsymbol{\theta}) \right) d\boldsymbol{\theta}}$$

Output: Policy $\pi(\boldsymbol{\theta})$

Table 3.1: Overview of the Empowered Skills algorithm.

4 Implementation

In this chapter, we will explain how we implemented the algorithm for our experimental setup.

4.1 Policy Representation

In our experiments the policy learned by the algorithm is represented by a multivariate Gaussian distribution over parameters of the Dynamic Movement Primitives (DMPs) that control the motion of the respective robot arm.

4.1.1 Multivariate Gaussians

A multivariate Gaussian distribution is the generalized form of the 1-dimensional normal distribution. For a k-dimensional random vector like our set of parameters $\boldsymbol{\theta} \in \Theta$ the multivariate Gaussian is given by

$$\boldsymbol{\theta} \sim \mathcal{N}_k(\boldsymbol{\mu}, \Sigma)$$

with

$$\begin{aligned}\boldsymbol{\mu} &= \mathbb{E}[\boldsymbol{\theta}] = [\mathbb{E}[\theta_1], \mathbb{E}[\theta_2], ..., \mathbb{E}[\theta_k]],\\ \Sigma &= \mathbb{E}[(\boldsymbol{\theta} - \boldsymbol{\mu})(\boldsymbol{\theta} - \boldsymbol{\mu})^T].\end{aligned}$$

The advantage of using multivariate Gaussian distributions for the policy parameters comes from the simplicity associated with integrating them into an algorithm. However, their simple form can also be a disadvantage in cases where the bell shape of the density function for each individual parameter does not fit the task at hand. Though for a first implementation such as ours, they present a decent enough choice while keeping the derivation simple.

4.1.2 Dynamic Movement Primitives

DMPs [25] are a versatile form of policy representation. They use a nonlinear dynamical system to represent the movement of a robot and allow for scaling of the execution speed.
At the core of a DMP is a linear spring-damper system that is modulated by a nonlinear forcing function f_t, i.e.,

$$\ddot{y}_t = \tau^2 \alpha_y(\beta_y(g - y_t) - \dot{y}_t) + \tau^2 f_t,$$

where the parameter τ is the time-scaling coefficient and the variables y_t and $\dot{y}_t$ give the desired joint position and velocity of the robot. α_y and β_y define the spring and damping constants. Lastly the goal parameter g is the unique-point attractor of the spring-damper system. It is changed by the forcing function

$$f_t = f(z_t) = \frac{\sum_{i=1}^{K} w_i \phi_i(z)}{\sum_{i=1}^{K} \phi_i(z)} z,$$

which is constructed as a weighted sum of K basis functions

$$\phi_i(z) = \exp\left(-\frac{1}{2\sigma_i^2}(z - c_i)^2\right).$$

The forcing function depends on the phase variable z_t which is initialized as $z = 1$ and exponentially converges to zero as $t \to \infty$ according to $\dot{z} = -\tau \alpha_z z$. α_z specifies the speed of that decline. The forcing function additionally depends on the weights w_i which are referred to as the shape-parameters of the DMP because they modulate the acceleration profile and thus influence the shape. For a more detailed look at DMPs the reader is referred to [25] and [6].

4.2 Outcome Representation

As the outcome space is continuous in all the experiments we use a weighted sum of Radial Basis Functions (RBFs) to approximate the distribution of outcomes $p_\pi(\boldsymbol{o})$.

4.2.1 Radial Basis Functions

An RBF is simply a real-valued function $\phi(\boldsymbol{x}, \boldsymbol{c})$ whose value depends solely on the distance between some point $\boldsymbol{x}$ and a center $\boldsymbol{c}$ so that

$$\phi(\boldsymbol{x}, \boldsymbol{c}) = \phi(\|\boldsymbol{c} - \boldsymbol{x}\|).$$

Two popular choices for $\phi(\boldsymbol{x}, \boldsymbol{c})$ are the Gaussian RBF

$$\phi(\boldsymbol{x}, \boldsymbol{c}) = \exp(-\|\boldsymbol{c} - \boldsymbol{x}\|_2^2)$$

and the multiquadratic RBF

$$\phi(\boldsymbol{x}, \boldsymbol{c}) = \sqrt{1 + \|\boldsymbol{c} - \boldsymbol{x}\|_2^2},$$

but depending on the problem, a different RBF might be preferable. A discussion of the various options is outside the scope of this work, so we kindly direct the interested reader towards [26] for more information.

Function Approximation with RBFs

After the choice for a specific RBF has been made, a sum of RBFs known as a trial function $u(\boldsymbol{x})$ can be used to approximate a target function $f(\boldsymbol{x})$, i.e.,

$$f(\boldsymbol{x}) \approx u(\boldsymbol{x}) = \sum_{i=1}^{N} \mu_i \phi(\boldsymbol{x} - \boldsymbol{c}_i),$$

where μ_i are a set of weights for the centers $\boldsymbol{c}_i$.

4.2.2 Approximation of $p_\pi(\boldsymbol{o})$

In our experiments, we decided on Gaussian RBFs as features. The feature weights $\boldsymbol{\mu} = [\mu_1, \mu_2, ..., \mu_n]$ we need for the approximation of $p_\pi(\boldsymbol{o})$ are determined by optimizing the dual function as described in the previous chapter.
The complete approximation is then given by

$$\begin{aligned} p_\pi(\boldsymbol{o}) \approx u_\mu(\boldsymbol{o}, \boldsymbol{c}) = \sum_{i=1}^{N} \mu_i \exp\left(-\frac{\boldsymbol{d}_i}{2w_i^2}\right), \\ \boldsymbol{d}_i = [d_{i1}, d_{i2}, ..., d_{im}], \\ d_{ij} = \|\boldsymbol{o}_j - \boldsymbol{c}_i\|_2^2, \\ w_i = \tilde{\boldsymbol{d}}_i, \end{aligned}$$

where $\boldsymbol{o} = [\boldsymbol{o}_1, \boldsymbol{o}_2, ..., \boldsymbol{o}_m]$ is vector of outcomes, $\boldsymbol{c} = [\boldsymbol{c}_1, \boldsymbol{c}_2, ...\boldsymbol{c}_n]$ is a vector of RBF centers and $\boldsymbol{w}_i$ is a center-specific weight.
We use twenty randomly chosen outcomes as RBF centers. For each center, we weigh the distance to each outcome by the squared median of distances to this center. We chose the median to have an outlier resistant scaling factor for every center.

5 Experiments

The objectives of our experiments are twofold. Firstly, we investigate the ability of our algorithm to learn policies showing high diversity in the outcome space. Secondly, we investigate how we can shape the learned policies by choosing different characteristics as outcomes.
We assessed the abilities and performance of our algorithm in two scenarios: a 2D reaching task involving a five Degrees of Freedom (DoF) robot arm and the table tennis task featuring a 9 DoF robot arm as shown in Figure 5.4.

5.1 Reaching Task

In the reaching task the robot's objective is to move the end effector from the top to a target position on the right. The robot performs this task in 100 time steps. The parameter space is 30-dimensional and consists of the weights of the Dynamic Movement Primitives (DMPs) (5 bases per joint and the target location in joint space). The reward $\mathscr{R}$ depends on the action cost $\boldsymbol{u}$ and the distance d between end effector and target in the last time step, i.e.,

$$\mathscr{R}(d, \boldsymbol{u}) = -10^5 d^2 - 0.7\boldsymbol{u}^T\boldsymbol{u}.$$

In this experiment scenario we define the positions of the middle joint at the end of the trajectory as the outcome. This results in a 2-dimensional, continuous outcome space. These outcomes are one possible way to hint at the diversity of the reaching strategy. A combination of joint positions in the last time step or the speed of joints at various time steps are other possible metrics.
To learn the policy, the algorithms runs 75 iterations of policy updates. In every iteration they each draw 50 sample trajectories from their current policy.

Comparison to State-of-the-Art RL Algorithms

The first set of experiments pits our algorithm against two state-of-the-art RL algorithms: Relative Entropy Policy Search [1] (REPS) and Model-Based Relative Entropy Stochastic Search [2] (MORE). We will evaluate the policy search process and the end positions for the two baseline algorithms as well as several runs of our algorithm with different settings of β.

The statistical results of the experiment runs can be seen in **Table 5.1**. While the policy entropy for our algorithms is only marginally increased in comparison to REPS and a lot lower than in the MORE results, the outcome entropy is distinctly higher. The average rewards of our algorithm and REPS are similar while MORE is slightly better. The results of the average reward come with a specifically high standard deviation.

β	Policy Entropy	Outcome Entropy	Average Reward
100	-302.47 ± 6.28	-10.980 ± 1.496	-4032 ± 3329
1000	-289.72 ± 10.04	-8.675 ± 1.533	-4316 ± 3363
10000	-264.36 ± 10.87	-5.677 ± 1.429	-4406 ± 2480
REPS	-311.48 ± 1.44	-17.394 ± 0.989	-3900 ± 3270
MORE	-49.56 ± 0.00	-14.291 ± 0.262	-2144 ± 1946

Table 5.1: The table shows the results of our reaching task experiments. The experiments were run with five trials of 75 iterations per setting. We include REPS [1] and MORE [2] results as baseline comparison. Of note are the similar policy entropy and average reward, the high standard deviation on the reward as well as the distinct differences in outcome entropy which show that our algorithm is able to find more diverse solutions.

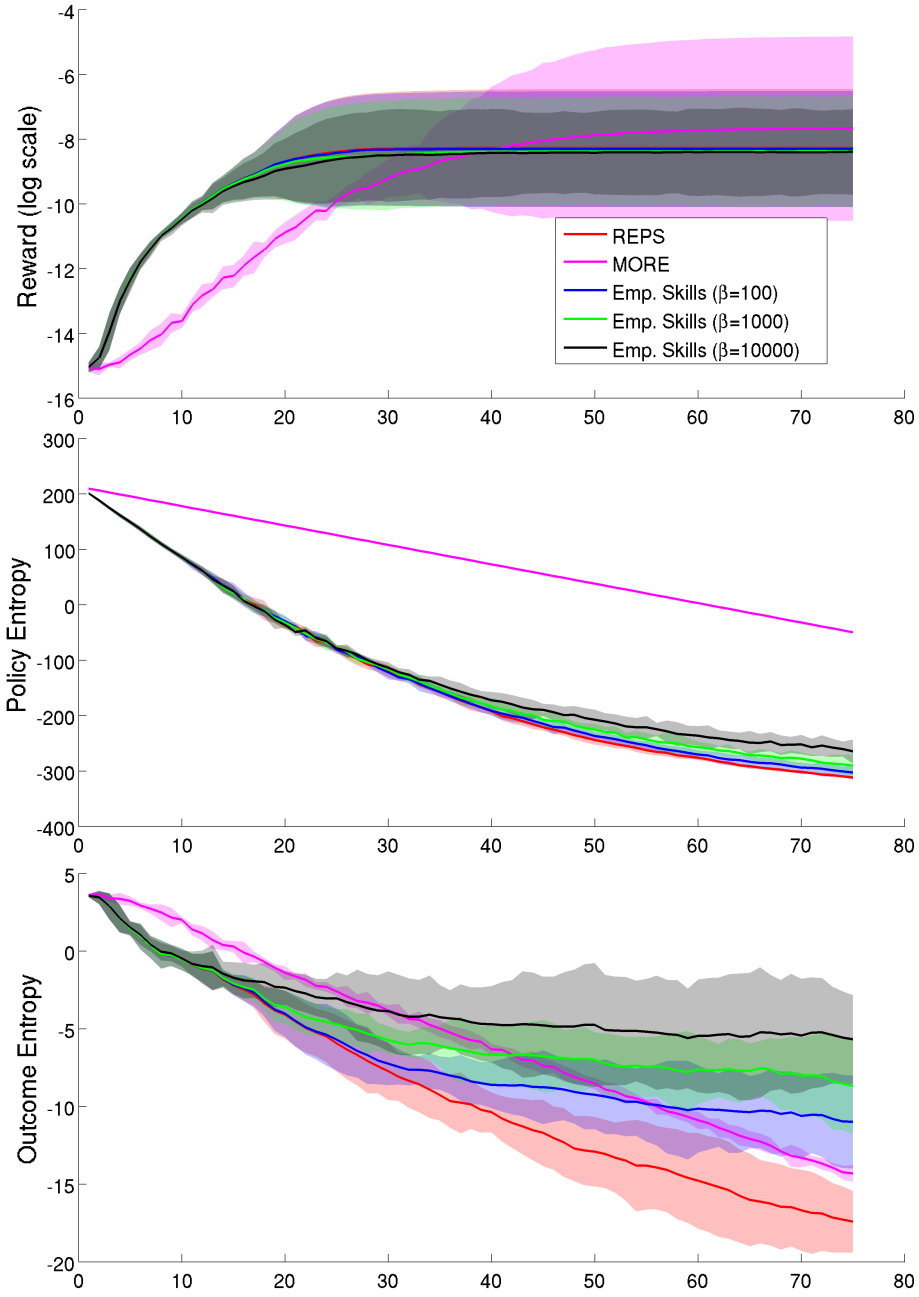

Figure 5.1: Evolution of outcome entropy, policy entropy and reward over 75 iterations in the reaching task experiments. The figure shows that our algorithm manages to find policies with higher outcome entropy at the expense of a slight reward decrease.

Figure 5.1 shows how outcome entropy, policy entropy and reward evolve over the 75 iterations for each algorithm. First we note that for all the runs except for MORE, the reward has stabilized by iteration 30 while the policy entropy continues to shrink for all the algorithms. The outcome entropy converges more slowly compared to the reward, stabilizing around iteration 60 for our algorithm, but continuing its descent for REPS and MORE. In the runs of our algorithm this stabilization occurs even earlier than that of the policy entropy. As expected, a higher setting of β, i.e., a higher emphasis on the outcome entropy, results in a policy with higher outcome entropy.
MORE [2] introduces an entropy constraint on the search distribution regulating the reduction of the parameter entropy at each iteration. The purpose of this constraint is to have a better control over the exploration, trading-off slower initial progress as apparent in **Figure 5.1** for better reward at convergence as can be seen in **Table 5.1**. The entropy constraint of MORE results in a slower and steadier decrease of the policy entropy. However, the higher entropy in parameter space does not translate into entropy in outcome space where our algorithm clearly outperforms the others.
The consequence of the returned policy of each algorithm can be seen in **Figure 5.2** which shows for different settings

of β, REPS and MORE, the robot arm configurations in the last time step as well as the target point, indicated by the red cross. For this we plotted the arm configuration for all the 50 sample trajectories on top of each other. The results make it clear that the solution found by our algorithm is similar to that found by REPS which it is based on. But where REPS finds a single solution, in which all the 50 trajectory samples produce postures that largely overlap, our algorithm exploits the robot's structure to locally increase diversity in the target joint. Choosing higher values for β leads to increased diversity in posture as was already indicated by the increase in outcome diversity in **Figure 5.1**.

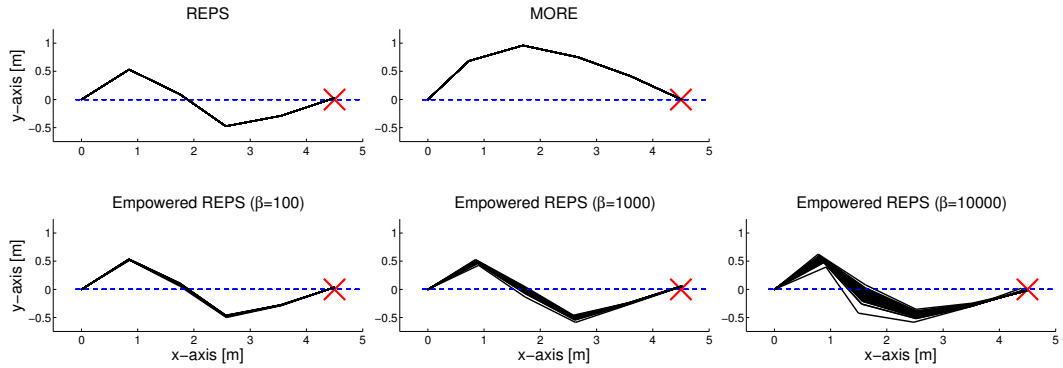

Figure 5.2: Final arm configuration of 50 trajectories sampled for different algorithms and values of β. Policies returned by standard RL algorithms exhibit no diversity (top row). For our algorithm (bottom row), increasing β increases the behavioral diversity while only slightly decreasing the reward.

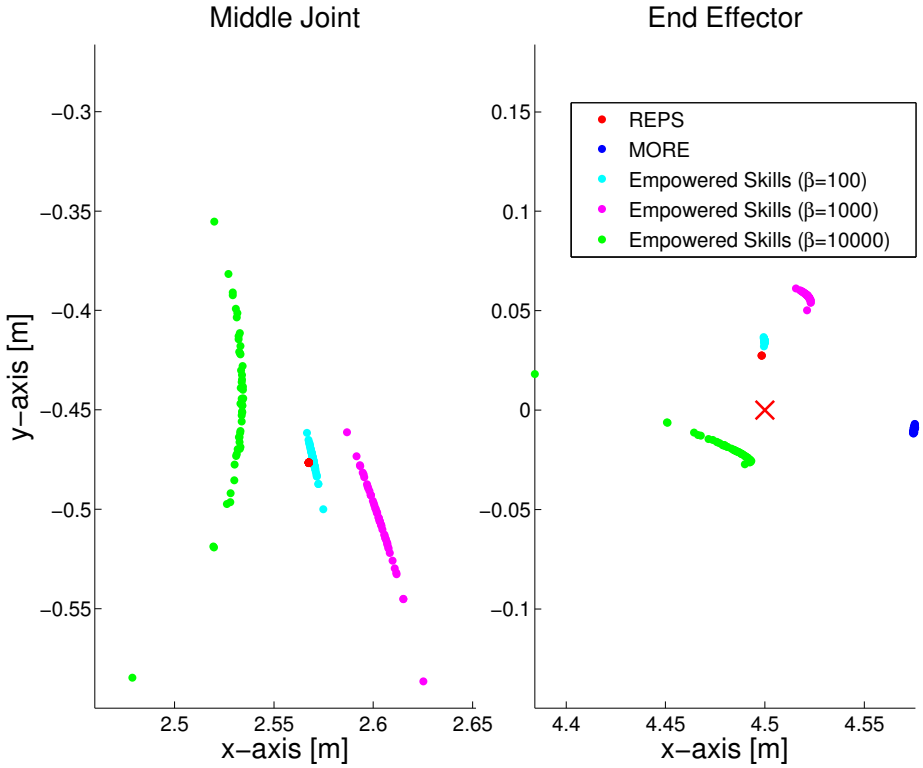

Figure 5.3: Middle joint position (left, outcome) and end effector (right, reward) of 50 trajectories sampled for different algorithms and values of β illustrating the trade-off between behavioral diversity and reward.

This can be confirmed with a look at **Figure 5.3**, which displays the end effector and middle joint positions in the last time step of the last iteration. The position distributions seem to be arc shaped in the runs of our algorithm while they are spots or blobs for REPS and MORE. This suggests that our algorithm exploits the configuration of the robot to locally increase diversity as it hints at variation in few, specific joints. Variation in many joints dissolves the arc-shape-constraint imposed by a single link rotating around a single joint.
We measured a sizable increase in the diversity of the outcomes. For REPS the middle joint positions cover an area smaller than $0.01mm^2$. For Empowered Skills ($\beta = 100$) this area is more than 220 times larger. The size of the markers in our plots exaggerates the area covered by REPS due to visibility constraints.

5.2 Robot Table Tennis

In the table tennis setting there is a simulated robot arm with a racket attached to its end effector. The robot's objective is to return a ball to the opponent's side of the table. The robot arm has six joints and is suspended over the table from a floating base which itself has three linear joints to allow small 3D movement. A trajectory is parameterized by the goal position and velocities of a DMP, yielding an 18-dimensional continuous action space (nine positions and nine velocities).

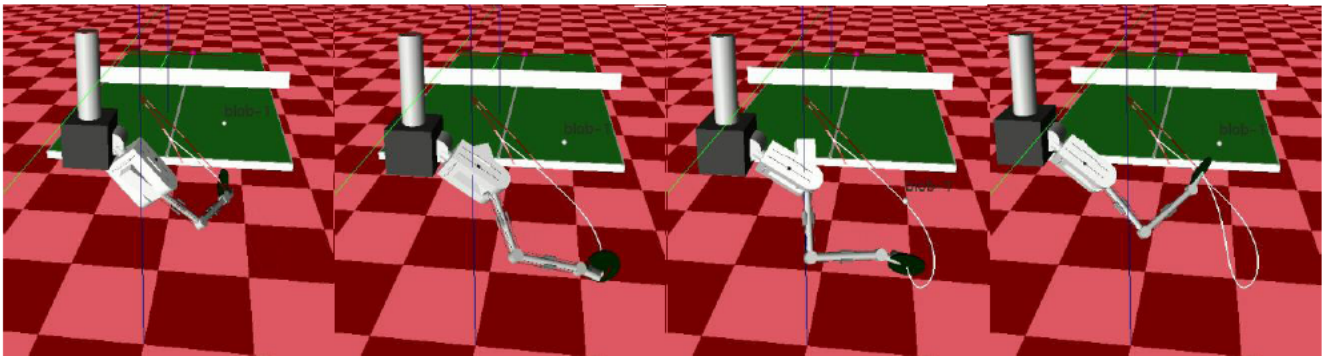

Figure 5.4: Images of the robot table tennis performing a forehand strike using a policy learned by our algorithm.

In this setting we conduct several experiments in which we choose different characteristics as outcomes. In the first experiment we use the location of the ball when landing on the table upon being returned by the robot. This gives us a 2-dimensional continuous outcome space.
In the second experiment we use the speed of the ball at the moment of impact instead, which gives a 1-dimensional outcome space. Since in these sets of experiments the outcomes are not properties of the robot, the relation between policy and outcomes depends on the environment (in this case the physics of ball movement).

We use imitation learning to initialize the weight parameters of the DMP and optimize for the goal position and velocity. The initial trajectory used for imitation learning was generated using a hand coded player following [27]. Once the policy is initialized, we run 100 iterations of each of the RL algorithms.
The reward optimized by these algorithms is inversely proportional to the distance between a target point[1] and the location where the ball first enters the table plane after being returned by the robot. If the robot does not hit the ball, the reward is the negative minimum distance between the racket and the ball throughout the trajectory.
In all the experiments we ran five trials of 100 iterations using 50 samples per iteration for each of the algorithms and settings of β.

Impact Location Experiment

The first set of experiments again pits our algorithm against a state-of-the-art RL algorithm and explores the influence of an increased β on the shape of the solutions. The outcomes in this case are the position of the ball in the table plane at the moment the ball enters this plane.

Figure 5.5 shows how outcome entropy, policy entropy and reward develop over 100 iterations of running the algorithms. The biggest differences are visible in the outcome entropy. While the policy entropy is comparable for all the runs over all iterations, the outcome entropy clearly decreases when a smaller β is chosen. The reward converges slower for higher β but does reach a similar score for all but the run with the highest β. Our algorithms seems to be able to keep outcome entropy and reward relatively high while decreasing the policy entropy.

[1] The target is marked by a red cross in our plots.

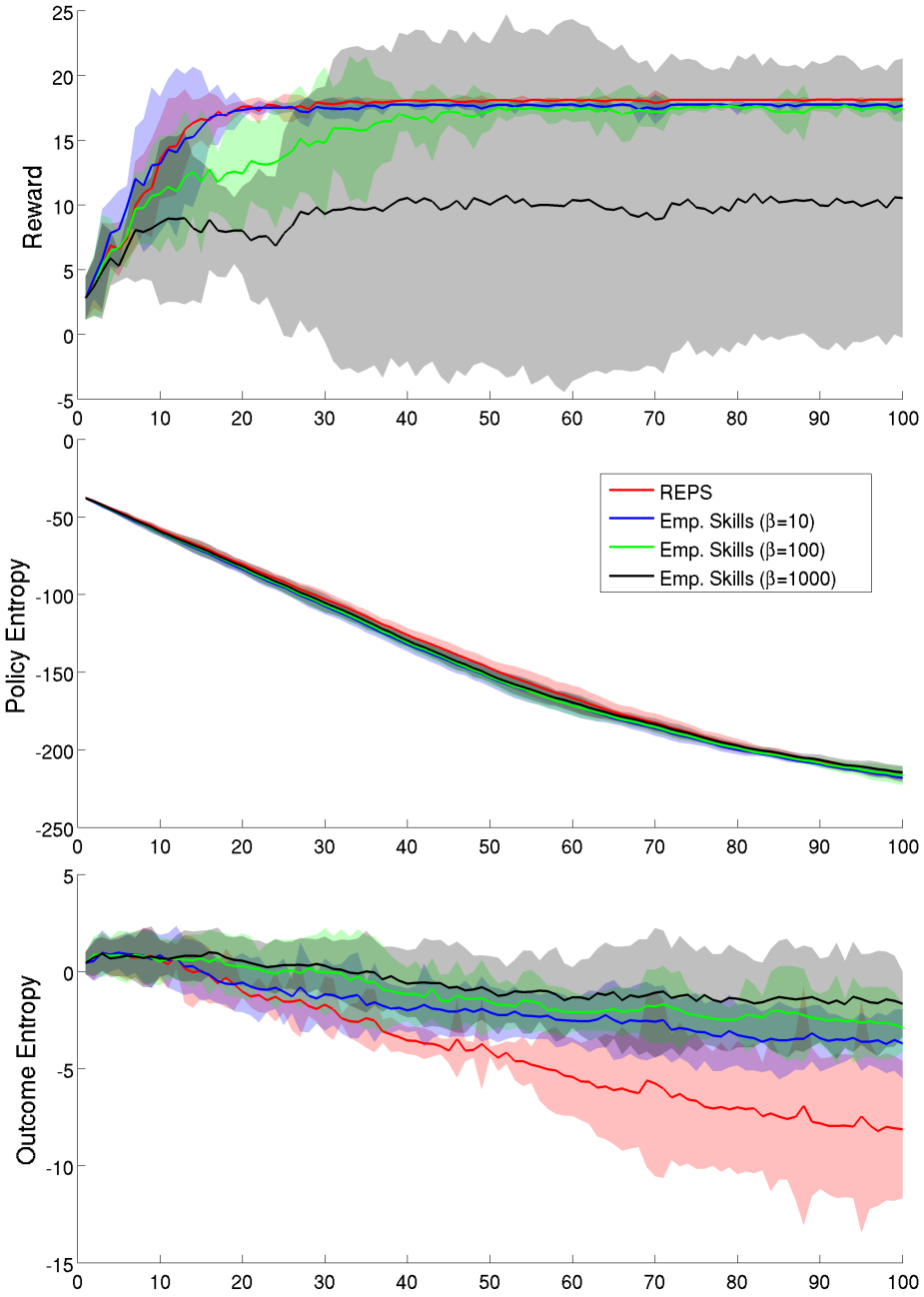

Figure 5.5: Evolution of outcome entropy, policy entropy, and reward over 100 iterations of the table tennis experiment. The figure shows the similarities in the policy entropy and reward while the outcome (bounce locations) curves are more distinct. For $\beta = 1000$, reward decreases due to balls hitting the net.

Figure 5.6 displays the outcomes of 50 sample trajectories drawn from the final policy for different settings of β. We can see that the impact area grows with β. For the second highest choice of β two shots miss the table and for the highest β some shots end in the net. Interestingly, with increasing outcome entropy the center of outcomes moves to the middle of the opponent's side of the table. Had it stayed in the same place (the target point marked by the red cross) approximately half of the outcomes would have missed the table and with it the objective. This shows that our algorithm finds a compromise between getting a high reward and producing outcomes with higher entropy.

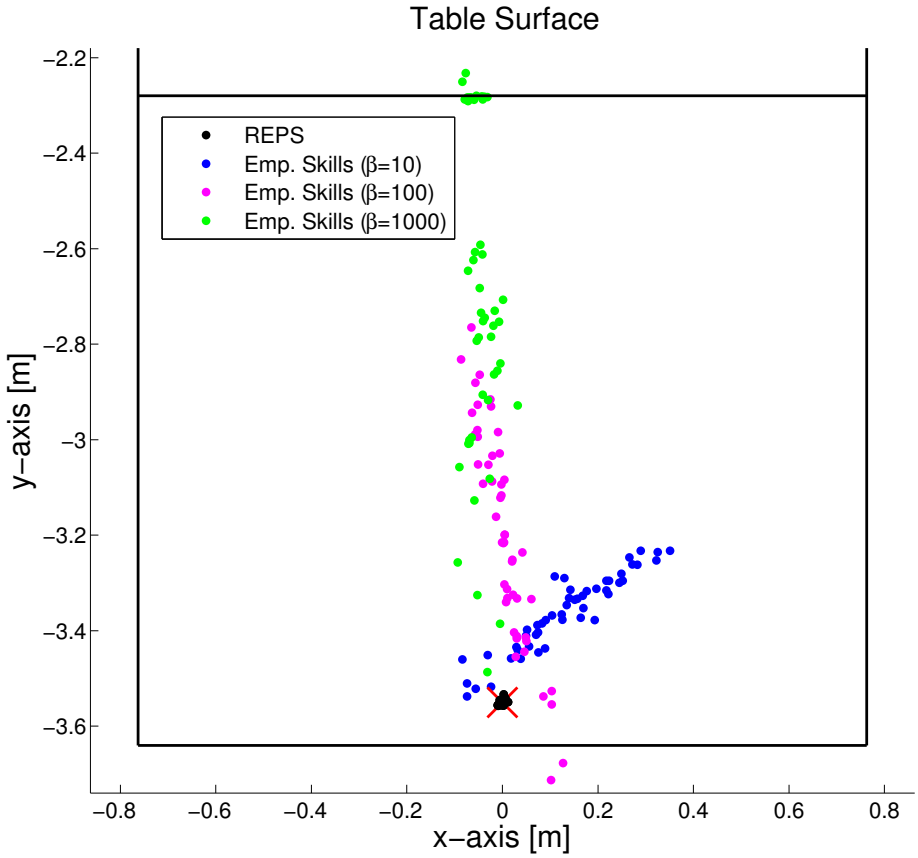

Figure 5.6: A top view of the table. Marked on it are the impact points of the ball for 50 sample trajectories per algorithm. Higher values of β lead to a larger spread of outcomes. The mean of the outcomes is different as well, and gets further away from the table's edge.

The results for the last iteration are shown again in **Table 5.2**. In the table tennis experiment, the standard deviation of the average return is a lot smaller than in the reaching task experiment for all runs except for that with the highest setting of β. In the runs of our algorithm the average reward is slightly decreased compared to REPS and decreases further for high values of β. The gain in outcome entropy is significantly more distinct while the entropy of the policy appears very similar in all runs.

β	Policy Entropy	Outcome Entropy	Average Reward
10	-217.805 ± 1.463	-3.705 ± 0.891	17.694 ± 0.190
100	-216.498 ± 2.951	-2.898 ± 0.616	17.412 ± 0.503
1000	-214.432 ± 2.184	-1.654 ± 0.830	10.536 ± 5.394
REPS	-216.863 ± 1.591	-8.119 ± 1.765	18.124 ± 0.017

Table 5.2: Results of the Table Tennis Experiments. These experiments were run in five trials with 100 iterations and 50 samples per iteration. Displayed are mean values over the five trials. The results show how higher settings of β lead to an increased outcome entropy while having a limited effect on policy entropy and reward except for $\beta = 1000$. In this experiment the outcomes are the 2D location where the ball impacts the table after being returned by the robot.

Figure 5.7 shows the final ball trajectories for this experiment. The figure shows 50 ball trajectories for each of the algorithms. While the trajectories for REPS are indistinguishable and those with $\beta = 10$ are still very similar, for higher values of β the trajectories cover a much wider area. The figure makes it clear that the Empowered Skills algorithm is able to learn a policy that represents multiple solutions to the posed problem while the policy learned by REPS just yields a single behavior.

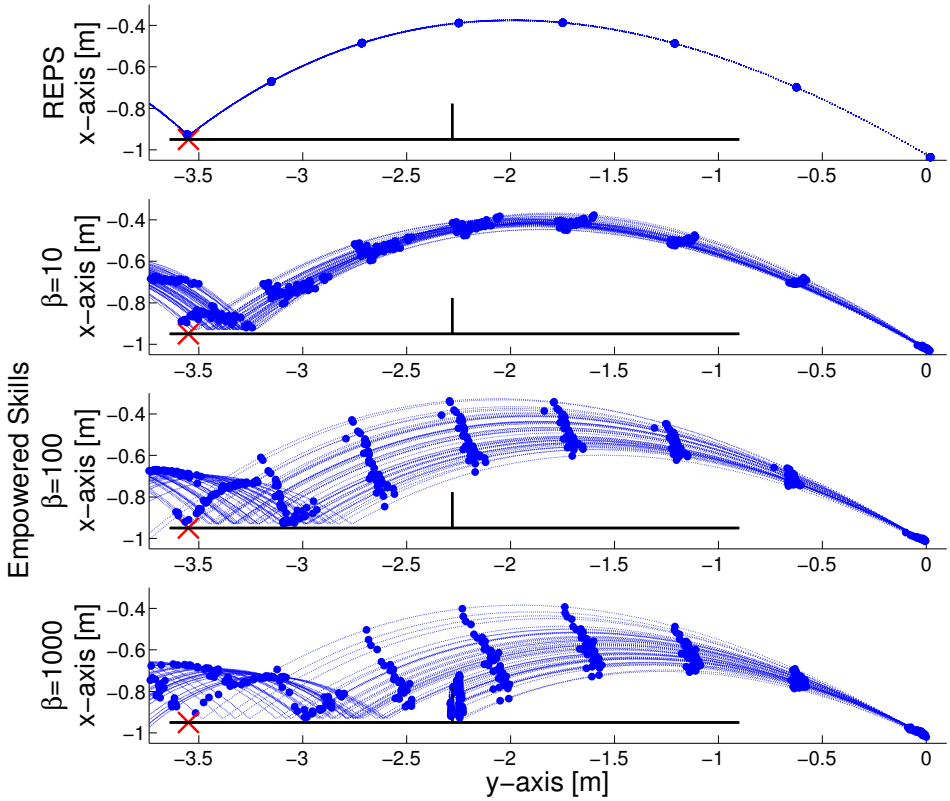

Figure 5.7: Sample trajectories from REPS and Empowered Skills for different values of β. REPS, although learning a probabilistic policy, always shoots to the target. Our algorithm is able to simultaneously learn to shoot to different areas of the table.

Impact Speed Experiment

The second set of experiments changes the selection of outcomes from the impact location of the ball on the table to the speed at which the ball impacts. This leads our algorithm to learn a different set of policies that result in a different solution again tailored to increase the diversity of the selected outcomes.

Figure 5.8 shows the distribution of outcomes for REPS and our algorithm at different settings of β. The diversity of impact speeds increases with growing β. Again the algorithm learns policies with different mean, shifting away from the table border with increasing β. This shift is a lot less pronounced than in the experiments with position outcomes in Section 5.2.

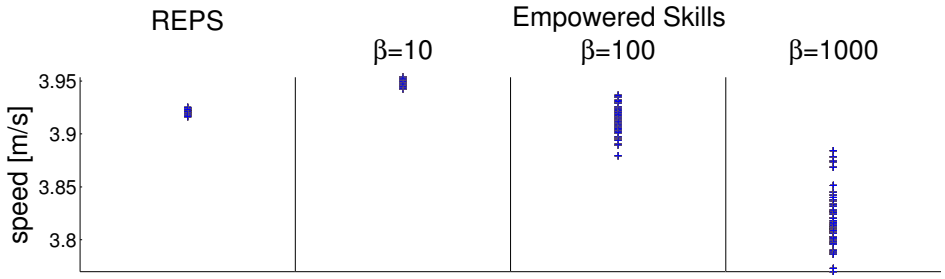

Figure 5.8: Speed outcomes of 50 trajectories sampled for different values of β and REPS. Here the outcome is the ball's impact speed on the table. The figure shows how increasing β leads to more diverse impact speeds that are distributed around different means.

With a look at **Figure 5.9** it becomes also clear that the diversity of impact locations is much lower than in the previous experiment. We can thus selectively increase the diversity of specific, task appropriate parts of the sensorimotor stream, by selecting these parts as outcomes, without unduly compromising the accuracy of other parts of the sensorimotor stream.

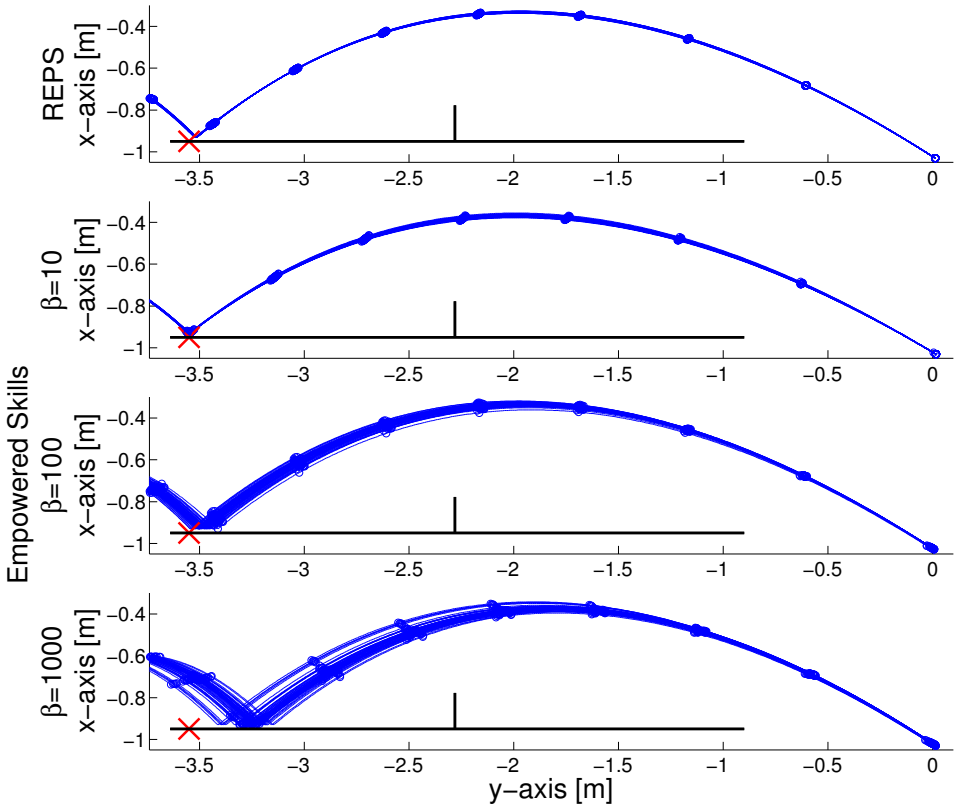

Figure 5.9: Speed outcomes of 50 trajectories sampled for different values of β and REPS. The solutions found by our algorithm for the speed experiment show a distinctly reduced spread on the table compared to those for the position experiment (Sec. 5.2).

6 Conclusion

In this work we gave a short overview of research on intrinsic motivation and its application in robotics. We specifically highlighted work on empowerment, the idea that the agent should move towards states that maximize its potential influence on the environment, and later introduce a flavor of empowerment does not seek out empowering states, but searches empowering action policies.
We continue to introduce policy search, a family of algorithms that optimizes parameterized action policies, including the differences between model-based and model-free approaches.
In the main part of this work we added an intrinsic motivation term to a state-of-the-art objective driven Policy Search algorithm. The intrinsic motivation term we use is based on the entropy of action outcomes, where an outcome is an interesting aspect of the sensorimotor stream.
Our algorithm is capable of solving problems with continuous action and outcome spaces and is easily extendable to the contextual case. We tested the algorithm in two simulated experimental settings, a reaching task and a robot table tennis task.
The experiments show how the Empowered Skills algorithm proposed in this work is able to learn policies which exhibit higher behavioral diversity in high reward areas of the task and how that diversity is specifically tied to those parts of the sensorimotor stream chosen as outcomes.
Future work might improve upon this by adding an inverse diversity metric. This might allow the improved algorithm to find solutions featuring increased diversity in one set of sensorimotor aspects and decreased diversity in another set.

The main limitation of our current setting is the simple form of our Gaussian policy. In order to further increase outcome diversity, our future perspective is to study the integration of this intrinsic motivation term in more complex distributions such as mixture models.
Another perspective is to learn a higher level policy to select the most appropriate skill in a cooperative or adversarial setting such as robot table tennis, from a library of skills discovered by intrinsic motivation.

Bibliography

[1] J. Peters, K. Mülling, and Y. Altün, "Relative Entropy Policy Search," in *AAAI Conference on Artificial Intelligence*, pp. 1607–1612, 2010.

[2] A. Abdolmaleki, R. Lioutikov, J. R. Peters, N. Lau, L. P. Reis, and G. Neumann, "Model-Based Relative Entropy Stochastic Search," in *Advances in Neural Information Processing Systems*, pp. 3523–3531, 2015.

[3] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement Learning in Robotics: A Survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[4] S. Levine, N. Wagener, and P. Abbeel, "Learning Contact-Rich Manipulation Skills with Guided Policy Search," in *IEEE International Conference on Robotics and Automation*, pp. 156–163, 2015.

[5] P. Kormushev, S. Calinon, and D. G. Caldwell, "Robot Motor Skill Coordination with EM-based Reinforcement Learning," in *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010*, pp. 3232–3237, 2010.

[6] M. P. Deisenroth, G. Neumann, and J. Peters, "A Survey on Policy Search for Robotics," *Foundations and Trends in Robotics*, vol. 2, no. 2011, pp. 1–142, 2011.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *arXiv preprint arXiv:1312.5602*, pp. 1–9, 2013.

[8] C. Daniel, G. Neumann, O. Kroemer, and J. Peters, "Hierarchical Relative Entropy Policy Search," *Journal of Machine Learning (JMLR)*, vol. 17, pp. 1–50, 2016.

[9] A. Jain, B. Wojcik, T. Joachims, and A. Saxena, "Learning Trajectory Preferences for Manipulators via Iterative Improvement," in *Advances in Neural Information Processing Systems*, pp. 575–583, 2013.

[10] P. Y. Oudeyer and F. Kaplan, "What is Intrinsic Motivation? A Typology of Computational Approaches," *Frontiers in Neurorobotics*, vol. 1, no. Nov, p. 6, 2009.

[11] T. Jung, D. Polani, and P. Stone, "Empowerment for Continuous Agent—Environment Systems," *Adaptive Behavior*, vol. 19, pp. 16–39, 2011.

[12] A. Baranes and P.-Y. Oudeyer, "R-IAC: Robust Intrinsically Motivated Exploration and Active Learning," *IEEE Transactions on Autonomous Mental Development*, vol. 1, no. 3, pp. 155–169, 2013.

[13] J. Schmidhuber, "A Possibility for Implementing Curiosity and Boredom in Model-Building Neural Controllers," in *From Animals to Animats: International Conference on Simulation of Adaptive Behavior*, vol. 1, pp. 222–227, 1991.

[14] M. Lopes, T. Lang, M. Toussaint, and P.-Y. Oudeyer, "Exploration in Model-Based Reinforcement Learning by Empirically Estimating Learning Progress," in *Advances in Neural Information Processing Systems*, pp. 206–214, 2012.

[15] G. Baldassarre and M. Mirolli, "Deciding Which Skill to Learn When: Temporal-Difference Competence-Based Intrinsic Motivation (TD-CB-IM)," in *Intrinsically Motivated Learning in Natural and Artificial Systems* (G. Baldassarre and M. Mirolli, eds.), pp. 257–278, Springer, 2013.

[16] P. Delarboulas, M. Schoenauer, and M. Sebag, "Open-Ended Evolutionary Robotics: An Information Theoretic Approach," in *International Conference on Parallel Problem Solving from Nature*, pp. 334–343, 2010.

[17] J. Lehman and K. O. Stanley, "Abandoning Objectives: Evolution Through the Search for Novelty Alone.," *Evolutionary computation*, vol. 19, no. 2, pp. 189–223, 2011.

[18] A. S. Klyubin, D. Polani, and C. L. Nehaniv, "All Else Being Equal Be Empowered," in *European Conference on Artificial Life*, pp. 744–753, 2005.

[19] Y. Sun, D. Wierstra, T. Schaul, and J. Schmidhuber, "Efficient Natural Evolution Strategies," in *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, pp. 539–546, 2009.

[20] V. Kumar, E. Todorov, and S. Levine, "Optimal Control with Learned Local Models : Application to Dexterous Manipulation," in *IEEE International Conference on Robotics and Automation*, pp. 378–383, 2016.

[21] A. Kupcsik, M. P. Deisenroth, J. Peters, A. P. Loh, P. Vadakkepat, and G. Neumann, "Model-based Contextual Policy Search for Data-Efficient Generalization of Robot Skills," *Artificial Intelligence*, 2015.

[22] J. Schulman, S. Levine, M. Jordan, and P. Abbeel, "Trust Region Policy Optimization," in *International Conference on Machine Learning*, pp. 1889–1897, 2015.

[23] J. L. Lagrange, "Méthode des Multiplicateurs," in *Mécanique Analytique vol 1*, ch. IV, pp. 74–104, Paris: Ve Courcier, 1811.

[24] D. P. Bertsekas, *Nonlinear Programming: 3rd Edition*. Athena Scientific, 2016.

[25] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert, "Learning Movement Primitives," *Robotics Research*, vol. 15, no. 2005, pp. 1–10, 2005.

[26] M. D. Buhmann, *Radial Basis Functions: Theory and Implementations*, vol. 12. Cambridge university press, 2003.

[27] K. Mülling, J. Kober, and J. Peters, "Simulating Human Table Tennis with a Biomimetic Robot Setup," in *International Conference on Simulation of Adaptive Behavior,*, pp. 273–282, 2010.