Generalizing Movements with Information Theoretic Stochastic Optimal Control

Rudolf Lioutikov[1], Alexandros Paraschos[1], Jan Peters[1,2] and Gerhard Neumann[1]

1: Technische Universität Darmstadt, Intelligent Autonomous Systems Group, Darmstadt, Germany

2: Max Planck Institute for Intelligent Systems, Department Empirical Inference, Tübingen, Germany

Abstract

Stochastic Optimal Control (SOC) is typically used to plan a movement for a specific situation. While most SOC methods fail to generalize this movement plan to a new situation without re-planning, we present a SOC method that allows us to reuse the obtained policy in a new situation as the policy is more robust to slight deviations from the initial movement plan. In order to improve the robustness of the policy, we employ information-theoretic policy updates that explicitly operate on trajectory distributions instead of single trajectories. To ensure a stable and smooth policy update, we limit the 'distance' between the trajectory distributions of the old and the new control policy. The introduced bound offers a closed form solution for the resulting policy and extends results from recent developments in SOC. In difference to many standard SOC algorithms, our approach can directly infer the system dynamics from data points, and, hence, can also be used for model-based reinforcement learning. This paper represents an extension of [16]. In addition to revisiting the content, we provide an extensive theoretical comparison our approach with related work, discuss additional aspects of the implementation and introduce further evaluations.

1 Introduction

The goal of Stochastic Optimal Control (SOC), see [26, 12, 31, 23], is to find the optimal control for a finite horizon of time steps. A common approach to solve the SOC problem is dynamic programming, i.e. the value function is computed iteratively backwards in time. The value function estimates the expected future reward for a given state and time step. Unfortunately, only few cases offer analytical solutions, e.g. discrete systems and linear system with quadratic cost functions and Gaussian noise (LQG). For more complex systems, many approaches rely on a linearisation of the underlying system [30, 31]. However, while these approaches work for planning a single movement, the resulting controller is hard to generalize to a new situation, e.g., a movement task starting from a slightly different initial position. In this case, re-planning is often required as the used linearizations are not valid for the new initial position. The second short coming of state of the art SOC methods is that the policy update might by unstable. The used approximations of either the system dynamics [30, 31], or of the value function [11, 19] cause these unstable updates. Due to such unstable policy update, the resulting state distribution of the new policy might jump, leading the agent into unexplored areas of the state space where the estimated policy might have poor quality.

In this paper, we tackle both problems by introducing an information theoretic policy update for SOC that directly operates on trajectory distributions instead of single nominal trajectories. We will use an information theoretic constraint in our optimization to ensure that the new policy will stay close to our previously generated data. Such 'closeness to the data' constraint can be formalized by limiting the Kullback-Leibler (KL) $\mathrm{KL}(p||q)$ divergence between the distribution q that generated the data and the resulting distribution p that is generated by the new policy. Such a constraint avoids that the resulting state distribution of the new policy to jumps away from the data. Too large update steps lead the agent into unexplored areas of the state space where we have

no guarantee about the estimated. Our method is inspired by policy search methods, where similar constraints have already been used to find high quality parameters for parametric policies [21, 7, 8]. However, SOC is a more challenging application as the resulting policy has typically one to two orders of magnitude more parameters. We address the complexity of the resulting optimization problem by dividing the problem into two separate problems that can be solved independently and, thus, are easier to solve.

As for most SOC methods, we focus on estimating locally optimal control policies for a single type of movement. Hence, we do not aim to estimate a globally optimal controller, but the controller should be robust to deviations from the nominal trajectory. For example, if we want to estimate a back-hand swing in tennis, the resulting controller should be applicable to many configurations of the incoming ball without re-estimating the controller. Yet, we only aim for using the same controller for the same type of movement, i.e., we do not want to generalize from backhand to forehand strikes.

As in the LQG case, our goal is to estimate a time-dependent linear feedback controller. Our approach is also model based. We learn a time dependent linear model representation, which can be easily obtained from samples. Our approach is easily applicable to model-based reinforcement learning, where the system dynamics are not known beforehand but learned from data. Existing SOC methods differ from our approach in two important points. The first difference is the policy update; we bound the change of the resulting, time-dependent, state distributions of the new policy to the state distributions of the old policy for each time step. This bound assures that we stay close to the data-generating distribution. As the data-generating distribution incorporates the exploration of the previous policy, the new policy will continue to explore in a similar area of the state space as the old policy. Recent SOC approaches [23, 2] penalize the deviation of the new policy from the previous policy, instead of penalizing

the deviation of the resulting state distributions. While such an approach continues to explore in a similar action space as the previous policy, it does not limit the change of the state distributions. Hence, the policy update might lead the agent into unseen areas of the state space where the estimated policy typically performs poorly. Other SOC approaches do not penalize any deviation from the data generating distribution, i.e, the policy update is greedy. Such greedy policy updates are often unstable and do not continue to explore the state-action space of the agent, resulting in premature convergence. The second important difference is that we can avoid value function approximation. We replace it by matching average features of the resulting state distributions. Both differences, namely the bound on the change of the state distributions and the average feature matching, allow to control which states are explored by the new policy. In this paper, we will explain in more detail that many of the limitations of existing SOC approaches are a consequence of the absence of such mechanism.

This paper is an extension of our results presented in [16]. We provide an extensive theoretical comparison of our approach with related work, discuss additional aspects of the implementation and introduce further evaluations. In the next section, we will discuss the related work. Subsequently, we will present an alternative formulation of SOC based on constrained optimization that is the basis for our algorithm, called information theoretic SOC (ITSOC). After introducing the information-theoretic constraints, we will state our specific algorithm including the representation of the policy and the state features. Finally, we will discuss the relation to existing approaches and present experiments of our method on a simulated 4-link non-linear planar arm.

1.1 Preliminaries

In this paper, we consider systems with continuous state and action spaces. We will denote the state of the robot as $\boldsymbol{x}$ and the control action as $\boldsymbol{u}$. The robot follows its stochastic system dynamics and transfers to state $\boldsymbol{x}_{t+1}$ from state $\boldsymbol{x}_t$ with probability

$p_t(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)$ when taking action $\boldsymbol{u}_t$. The goal of the robot is to find the optimal policy $\pi_t^*(\boldsymbol{u}|\boldsymbol{x})$ which maximizes the expected accumulative reward for a given time horizon H

$$\pi^* = \arg\max_{\pi} J = \arg\max_{\pi} \mathbb{E}_{p^\pi(\boldsymbol{\tau})} \left[\sum_{t=1}^{H-1} r_t(\boldsymbol{x}_t, \boldsymbol{u}_t) + r_H(\boldsymbol{x}_H) \right], \tag{1}$$

where the expectation is taken with respect to the trajectories $\boldsymbol{\tau} = (\boldsymbol{x}_{1:H}, \boldsymbol{u}_{1:H-1})$, $r_t(\boldsymbol{x}_t, \boldsymbol{u}_t)$ denotes the immediate reward signal for time step t and $r_H(\boldsymbol{x}_H)$ the final reward for reaching state $\boldsymbol{x}_H$. Note that the optimal policy is typically not stationary but depends on the time step t as we deal with a finite time horizon. The actions $\boldsymbol{u}_t$ of the agent are chosen by a policy $\pi_t(\boldsymbol{u}_t|\boldsymbol{x}_t)$ that in the finite horizon formulation is also time-dependent. The trajectory distribution of policy π is given by

$$p^\pi(\boldsymbol{\tau}) = p_1(\boldsymbol{x}_1) \prod_{t=1}^{H-1} p_t(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)\pi_t(\boldsymbol{u}_t|\boldsymbol{x}_t),$$

where $p(\boldsymbol{x}_1)$ is the initial state distribution. We will further denote $\mu_t^\pi(\boldsymbol{x}_t)$ as the state distribution of policy π. It represents the probability of being in state $\boldsymbol{x}_t$ at time step t when following policy π. Formally, the state distribution $\mu_t^\pi(\boldsymbol{x}_t)$ can be defined as

$$\mu_t^\pi(\boldsymbol{x}_t) = \int \dots \int p^\pi(\boldsymbol{\tau}) d\boldsymbol{x}_1 \dots d\boldsymbol{x}_{t-1} d\boldsymbol{x}_{t+1} \dots d\boldsymbol{x}_H d\boldsymbol{u}_1 \dots d\boldsymbol{u}_{H-1} \tag{2}$$

as marginalization over the state-action space of all remaining time steps. Alternatively, the state distributions $\mu_t^\pi(\boldsymbol{x}_t)$ can also be expressed recursively in terms of the state distribution from the previous time step, i.e.,

$$\mu_t^\pi(\boldsymbol{x}_t) = \iint \mu_{t-1}^\pi(\boldsymbol{x}_{t-1})\pi_{t-1}(\boldsymbol{u}_{t-1}|\boldsymbol{x}_{t-1})p_{t-1}(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}) d\boldsymbol{x}_{t-1} d\boldsymbol{u}_{t-1}. \tag{3}$$

2 Related Work

Current SOC methods either use an approximate system model, e.g., by linearizing the system, or they approximate the value function – also called approximate dynamic programming (ADP). We will first review these two approaches and point out important differences to our approach. Subsequently, we will discuss SOC control algorithms that also use Kullback-Leibler divergence terms to determine the policy, e.g., dynamic policy programming [2], SOC by approximate inference [23] and path integral approaches [27, 24]. We will also discuss the relation to approximate dynamic programming with soft-max operators [19] and existing policy search algorithms [8].

2.1 Dynamic Programming

Dynamic Programming (DP) is a common approach to solve a SOC problem by iteratively estimating the value function [18, 5, 11, 3]. The value function $V_t^\pi(\boldsymbol{x}_t)$ computes the expected future reward when being in state $\boldsymbol{x}_t$ at time step t and following policy π. In its recursive form, the definition of the value function is given as

$$V_t^\pi(\boldsymbol{x}_t) = \int \pi_t(\boldsymbol{u}_t|\boldsymbol{x}_t) \left(r_t(\boldsymbol{x}_t, \boldsymbol{u}_t) + \int p_t(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t) V_{t+1}^\pi(\boldsymbol{x}_{t+1}) d\boldsymbol{x}_{t+1} \right) d\boldsymbol{u}_t, \tag{4}$$

for $t \leq H$ and $V_H^\pi(\boldsymbol{x}) = r_H(\boldsymbol{x})$. The optimal value function $V_t^*(\boldsymbol{x}_t)$ can be obtained by iterating

$$V_t^*(\boldsymbol{x}_t) = \max_{\boldsymbol{u}} Q^*(\boldsymbol{x}_t, \boldsymbol{u}), \tag{5}$$

$$Q^*(\boldsymbol{x}_t, \boldsymbol{u}) = r_t(\boldsymbol{x}_t, \boldsymbol{u}) + \int p_t(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}) V_{t+1}^*(\boldsymbol{x}_{t+1}) d\boldsymbol{x}_{t+1} \tag{6}$$

backwards in time. Similarly, the optimal policy can be obtained by

$$\boldsymbol{u}_t^* = \pi_t^*(\boldsymbol{x}_t) = \operatorname{argmax}_{\boldsymbol{u}} \left(Q^*(\boldsymbol{x}_t, \boldsymbol{u}) \right) \tag{7}$$

Unfortunately, analytic solutions only exists in the case of discrete state and action spaces or for linear systems with a quadratic reward function and Gaussian noise, which we will denote as LQG system. It is well known that the optimal value function is in this case given as quadratic function, i.e., $V_t^*(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{V}_t \boldsymbol{x} + \boldsymbol{x}^T \boldsymbol{v}_t + v_t$ and the optimal policy is given as time varying linear feedback controller, i.e., $\pi^*(\boldsymbol{x}) = \boldsymbol{s}_t + \boldsymbol{S}_t \boldsymbol{x}$. The parameters of the reward function as well as for the controller can be obtained analytically [26].

For non-LQG systems we have to resort to some type of approximations. Here, we can distinguish between approaches based on approximating the system dynamics and approaches approximating the value function.

Approximating the System Dynamics. Approaches based on approximating the system dynamics, such as the iLQG [30] or the AICO [31] algorithm, linearise the system at the current estimate of the trajectory. For each time step, a different linearisation is used. The optimal controller is computed analytically by using the solution for the LQG system and, subsequently, a new trajectory is estimated by simulating this controller. This trajectory is used as a new linearisation point. Due to the used LQG solution, the obtained policy is greedy with respect to the linearised system dynamics. However, the linearisation of the model might change heavily in each iteration causing jumps in the resulting update of the trajectory. These problems can be alleviated by introducing heuristics such as regularizing the LQG solution [30] or using a learning rate on the estimated trajectory [31], but the policy update remains difficult to control. Moreover, in the case of small deviations from the nominal trajectory, the linearisation of the model becomes less accurate. Hence, the robustness of the resulting policy typically degrades and re-planning is required if, for example, we slightly deviate from the planned initial state. The advantage of linearisation approaches is their computational efficiency.

Approximating the Value Function. Other approximate dynamic programming approaches approximate the value function. These approaches are sample-based and iteratively collect data with the currently estimated policy. The data is subsequently used to improve the estimate of the value function that is approximated by supervised learning methods, such as linear function approximators [5, 23], locally-weighted regression [19] and regression trees [11]. The most common value function approximation method is a linear value function approximation. Here, a common method is to use the Least Squares Temporal Difference (LSTD) Algorithm [5, 9] to estimate the value function of a given policy. LSTD is iteratively used by the Least-Squares Policy Iteration (LSPI) algorithm [14] to obtain an estimate of the optimal policy. While there has been an exhaustive theoretical analysis of this algorithm [15], LSPI is limited by the linear function approximation which requires the knowledge of proper features for representing the value function. If the value function of the policies evaluated during learning is not representable, we might get poor estimates of the resulting policy. An alternative algorithm that does not require a linear representation of the value function is fitted Q-iteration [11, 19]. While convergence can only be proofed for a subset of supervised regression algorithms, i.e., k-nearest neighbor or local averagers, weaker theoretical results exists for more powerful regression algorithms such as regression trees. In this case, at least no divergence to infinity can be proofed [11]. Due to the improved flexibility of the value function representation the use of such regression algorithms often leads to improved results in comparison to LSPI.

One fundamental problem of value function approximation is that the approximation may 'damage' the resulting trajectory distribution of the policy, causing jumps and oscillations between subsequent iterations. Such behavior is in general not desirable as uncontrolled jumps in the trajectory distribution might lead the agent in unexplored areas of the state space. Consequently, the policy is likely to have a poor performance in these areas.

Advantage Weighted Regression. One problem for value function approximation approaches in continuous action spaces is the $\max$-operator over the action space. This operator is computationally costly to solve for continuous action spaces. Advantage Weighted Regression [19] is a method to efficiently approximate the $\max$-operator by replacing it with a soft-max operator [19]. The soft-max operator is implemented by using a stochastic policy

$$\pi_t(\boldsymbol{u}_t|\boldsymbol{x}_t) \propto \exp\left(\frac{Q_t^\pi(\boldsymbol{x}_t, \boldsymbol{u}_t) - V_t^\pi(\boldsymbol{x}_t)}{\eta}\right),$$

where $Q_t^\pi(\boldsymbol{x}_t, \boldsymbol{u}_t)$ is the state-action value function of the current policy. These probabilities are only evaluated on a finite set of samples and are subsequently used as weighting term in a weighted regression algorithm to determine the new value function. As the term inside the exponential function denotes the advantage function, the weighted regression is denoted as *advantage weighted regression* (AWR). From the weighted samples, we can also obtain a new parametric policy with a similar advantage weighted regression.

2.2 SOC based on Approximate Inference

An alternative view on SOC has been presented in [23] which is based on approximate inference. The main idea is to transform the reward into probabilities by introducing a binary reward event with probability $p(R_t = 1|\boldsymbol{x}_t, \boldsymbol{u}_t) \propto \exp(r_t(\boldsymbol{x}, \boldsymbol{u})/\eta)$. The parameter η_t has to be chosen by the user and denotes the temperature of the reward distribution. Given a prior policy π_0, the goal is to compute a policy π that produces a trajectory distribution $p^\pi(\boldsymbol{\tau})$ which is most similar to posterior trajectory distribution $p^{\pi_0}(\boldsymbol{\tau}|R_{1:H} = 1)$ after conditioning on seeing the reward event in every time step. More formally, we need to find a policy π which minimizes the KL-divergence between the posterior $p^{\pi_0}(\boldsymbol{\tau}|r_{1:H} = 1)$ and the trajectory distribution $p^\pi(\boldsymbol{\tau})$ induced by policy

π, i.e.,

$$\pi^* = \text{argmin}_\pi \text{KL}(p^\pi(\boldsymbol{\tau})||p^{\pi_0}(\boldsymbol{\tau}|r_{1:H} = 1)).$$

This minimization is performed iteratively. The policy exhibits a similar soft-max structure as the one presented in this paper. However, the exponential transformation of the reward is an assumption that is hard to justify in this approach. As we will see, the most important difference to our approach is that the KL-divergence is directly used on the trajectory distributions $p^\pi(\boldsymbol{\tau})$. Such a KL-formulation can be decomposed into the sum of the Kullback Leibler divergences $\sum_t \text{KL}(\pi_t(\boldsymbol{u}|\boldsymbol{x})||\pi_{0,t}(\boldsymbol{u}|\boldsymbol{x}))$ of the policies for each time step. These KL-terms can be seen as additional cost terms in the immediate reward function which punish deviations of the estimated policy from the prior policy π_0 [23]. Such a punishment term is also used in the dynamic policy programming (DPP) algorithm [2] that is a special case of the approximate inference framework introduced in [23]. Similar ideas of using the KL between the estimated policy and a prior policy have also been introduced within the field of linear solvable MDPs [28, 29]. To solve these modified SOC problems, typically value function approximation methods are used. Some approaches approximate the desirability function [2, 23], but such approaches suffer from similar problems as standard value function approximation as the approximation errors can propagate backwards in time. Consequently, SOC based on approximate inference suffers from the same deficits as ADP methods as the approximation of the value function might damage the resulting policy update. In contrast, our formulation of the KL-divergence acts on the marginals $p_t^\pi(\boldsymbol{x}, \boldsymbol{u})$ that also includes the state distribution $\pi^\mu(\boldsymbol{x})$ at each time step. Consequently, we can directly control the change of the state distributions $\pi^\mu(\boldsymbol{x})$, which results in a more stable policy update.

2.3 SOC based on Path Integrals

The path integral (PI) formulation of SOC [27] has recently gained a lot of attention as it allows for computing the optimal policy without the use of dynamic program-

ming. The PI approach exponentially transforms the continuous-time value function and computes the optimal value function and policy using the Feynman-Kac theorem [27]. The optimal value function for time step t is then given by

$$V_t^{\pi}(\boldsymbol{x}) = \eta \log \int p^{\pi_0}(\boldsymbol{\tau}|\boldsymbol{x}_t = \boldsymbol{x}) \exp\left(\frac{\sum_{t=1}^{H} r_t(\boldsymbol{x}_t)}{\eta}\right) d\boldsymbol{\tau}, \tag{8}$$

where $p_t^{\pi_0}(\boldsymbol{\tau}|\boldsymbol{x}_t = \boldsymbol{x})$ is the trajectory distribution of the *uncontrolled* process[1] starting at time step t in state $\boldsymbol{x}$. The parameter η defines the temperature of the exponential transformation of the value function and is chosen heuristically. The original PI approach is based on Monte-Carlo roll-outs and therefore requires a lot of samples, however, recently more efficient approaches based on value function approximation [24] have been introduced that are essentially similar to the SOC by approximate inference approaches. The PI approach suffers from two severe limitations. Firstly, it assumes that the control costs are quadratic and the quadratic control cost matrix coincides with the covariance matrix of the system noise. Moreover, it makes the assumption that all the noise in the system only acts on state variables that can be controlled by the agent. Both assumptions are a quite limiting.

2.4 Policy Search Methods

Our algorithm is inspired by policy search methods, where related information-theoretic bounds have already been introduced to learn to select and improve movement primitives for playing robot table-tennis [21], robot tether-ball [7] or robot hockey [8, 13]. The KL-bound has been originally introduced by the Relative Entropy Policy Search (REPS) algorithm [21], which has been extended in [7] to hierarchical policies and in [8] to learning sequential motor tasks. The most similar approach to our approach is the time-indexed formulation of REPS that has been used in [8]. The algorithm was designed for learning with movement primitives and can not be directly applied to the

[1] Or a process using a prior policy π_0.

SOC formulation as it introduces a bias if the system is stochastic and the proposed optimization algorithm is infeasible for the SOC domain. We will show in our experiments that this algorithm quickly degrades in the presence of noise in the system dynamics.

There has also been theoretical evidence that the information theoretic policy update has beneficial properties. In the case of a so called adversarial Markov decision process (MDP), the information theoretic policy update used in this paper achieves optimal regret bounds [33]. In an adversarial MDP, the reward function is chosen by an adversary at each episode. Hence, the reward function can change slightly from trial to trial. Such interpretation also fits well to our algorithm. While our reward function is fixed for all episodes, the continuous state and action samples change in each episode, and hence, the MDP defined on the discrete set of samples can be seen as adversary MDP.

Another policy search approach that is highly related to our approach is the model-based PILCO algorithm [10]. PILCO learns a Gaussian process (GP) model [22] of the system dynamics and uses deterministic approximate inference methods for predicting the trajectory distribution of the current policy with the learned models. The predicted trajectory distribution is used to obtain the policy gradient. PILCO performs a greedy policy update with the currently learned forward model, i.e., it searches for a local optimum of the policy parameters. However, as the currently estimated model is likely to be inaccurate, such greedy update might be dangerous as it might cause jumps in the trajectory distribution. Hence, the new policy might visit areas of the state space where we have little data, and hence, the learned forward models are of low quality. PILCO also neglects the problem of exploration as it relies on a deterministic policy. It is therefore likely to get stuck in a local minimum of the policy parameters. Such problems can be alleviated by building an exploration mechanism in the optimization criterion, such as an optimistic exploration with upper-confidence bounds [10]. In

difference to PILCO, we learn a much simpler model of the system dynamics, i.e., linear models. However, we learn an individual linear model for each time step and hence. our models are of similar expressiveness. While such time dependent models are less data efficient as time-independent GP models, they require considerably less computation time and offer good generalization properties.

In our comparisons, we will evaluate a special version of PILCO that uses the same time-dependent linear models as our approach. Instead of the gradient-based policy update of PILCO, we will use the AICO optimal controller [31] for the linear models. The difference to our approach is that PILCO computes the greedy policy with respect to the reward function using linear models, while we stay close to the data by the information theoretic policy update. As we will show, such greedy policy update performs poorly for the time-depend linear models, due to the instabilities of the policy update.

2.5 Inverse Reinforcement Learning

Inverse reinforcement learning (IRL) algorithms [32, 1, 20] try to estimate a reward function such that the optimal policy with respect to that reward function matches the features of the demonstrations on expectation. That is to say, they try to find a policy that stays close to the demonstrations. In our approach we also want to find a policy that stays close to the demonstrations while we use a given reward function that pulls the policy away from the demonstrations. We also use average feature matching. However, average feature matching is not used to reproduce the demonstrations but to ensure that we estimate a valid trajectory distribution. Our SOC approach also shares another important similarity with a recent IRL method called Maximum-Entropy (Max-Ent) IRL [32]. Max-Ent IRL estimates the maximum entropy policy such that the expected feature vector of this policy matches the average demonstrated features. As the entropy is highly connected to the relative entropy, which is used by our approach, the solution for

the optimal policy looks very similar in both approaches. However, the parameters of the policies are estimated differently as both algorithms try to solve different problems.

3 Information-Theoretic Stochastic Optimal Control

We will start our discussion by reformulating the SOC problem as constrained optimization problem. This reformulation has the advantage that we can easily add more constraints such as the information theoretic KL-bounds. Our approach exhibits two important differences to existing SOC approaches. First, we bound the change of the state-action distributions $\mu_t^\pi(\boldsymbol{x})\pi_t(\boldsymbol{u}|\boldsymbol{x})$ that is induced by the policy update instead of punishing solely the change of the policy $\pi_t(\boldsymbol{u}|\boldsymbol{x})$ [2, 23]. Second, instead of using value function approximation, we approximate the state distributions $\mu_t^\pi(\boldsymbol{x})$ by matching expected state features. Both differences allow an accurate control of the change in the state distribution, given we choose appropriate state features.

We want to maximize the expected reward, which we will now write in terms of the state distribution $\mu_t^\pi(\boldsymbol{x})$ as

$$J_{\pi\mu} = \sum_{t=1}^{H-1} \iint \mu_t^\pi(\boldsymbol{x})\pi_t(\boldsymbol{u}|\boldsymbol{x}) r_t(\boldsymbol{x},\boldsymbol{u}) d\boldsymbol{x} d\boldsymbol{u} + \int \mu_H^\pi(\boldsymbol{x}) r_H(\boldsymbol{x}) d\boldsymbol{x}. \tag{9}$$

For the first constraint, we require that π and μ jointly define a distribution, i.e., $\int \mu_t^\pi(\boldsymbol{x})\pi_t(\boldsymbol{u}|\boldsymbol{x}) d\boldsymbol{x} d\boldsymbol{u} = 1, \forall t$. Moreover, we can not freely choose our state distribution $\mu_t^\pi(\boldsymbol{x})$, but $\mu_t^\pi(\boldsymbol{x})$ has to comply with the policy and the system dynamics, i.e.,

$$\mu_t^\pi(\boldsymbol{x}') = \iint \mu_{t-1}^\pi(\boldsymbol{x})\pi_{t-1}(\boldsymbol{u}|\boldsymbol{x}) p_{t-1}(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u}) d\boldsymbol{x} d\boldsymbol{u}, \forall t > 1 \wedge \forall \boldsymbol{x}'. \tag{10}$$

Additionally, the state distribution $\mu_1^\pi(\boldsymbol{x})$ for the first time step has to reproduce the given initial state distribution of the problem, i.e., $\mu_1^\pi(\boldsymbol{x}) = p_1(\boldsymbol{x}), \forall \boldsymbol{x}$. The optimiza-

tion problem defined in Equation (9), under the constraints of Equations (10) and the initial state constraint is equivalent to the original SOC problem.

3.1 Approximate Constraints by Expected Feature Matching

The state distribution constraints given in Eq. (10) are infeasible in continuous state spaces as we would have an infinite amount of constraints. Therefore, we resort to matching expected feature averages [21], i.e.,

$$\int \mu_t^\pi(\boldsymbol{x}')\boldsymbol{\phi}(\boldsymbol{x}')d\boldsymbol{x}' = \iiint \mu_{t-1}(\boldsymbol{x})\pi_{t-1}(\boldsymbol{u}|\boldsymbol{x})p_{t-1}(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u})\boldsymbol{\phi}(\boldsymbol{x}')d\boldsymbol{x}d\boldsymbol{u}d\boldsymbol{x}' \quad (11)$$

for $t > 1$ and

$$\int \mu_1^\pi(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x})d\boldsymbol{x} = \int p_1(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x})d\boldsymbol{x} = \hat{\boldsymbol{\phi}}_1 \quad (12)$$

for $t = 1$, where $\boldsymbol{\phi}$ is a feature function and $\hat{\boldsymbol{\phi}}_1$ is the mean observed feature vector of the initial state. A potential problem with such a feature matching is, that this approximation does not guarantee a valid state-action distribution, as it introduces a bias in the optimization if the features are poorly chosen. However, a similar bias is introduced by the approximation error if we use value function approximation. As we only aim for optimizing local policies where, due to the locality, the state distribution is approximately Gaussian, all linear and squared terms of the state are a natural feature representation. This choice of the feature representation corresponds to matching the mean and the covariance of both distributions, i.e., the constraints would be accurate if both distributions are Gaussians. Our experiments show that the bias induced by the approximate feature constraints is considerably less severe compared to the bias introduced by value function approximation.

3.2 Staying Close to the Data with Information-Theoretic Bounds

It is crucial for SOC methods to stay close to the data in order to achieve a stable policy update. The data is given as state-action pairs $(\boldsymbol{x}_t, \boldsymbol{u}_t)$ generated from the old state-action distribution $q_t(\boldsymbol{x}_t, \boldsymbol{u}_t)$. As we want our new trajectory distribution $p^\pi(\tau)$ to stay close to this old data distribution, we bound the Kullback-Leibler divergence between the marginal distributions of the single time steps, i.e., $p_t^\pi(\boldsymbol{x}_t, \boldsymbol{u}_t) = \mu_t^\pi(\boldsymbol{x}_t)\pi_t(\boldsymbol{u}_t|\boldsymbol{x}_t)$ and $q_t(\boldsymbol{x}_t, \boldsymbol{u}_t)$. For $t < H$ the bound is given by

$$\epsilon \geq \iint \mu_t^\pi(\boldsymbol{x})\pi_t(\boldsymbol{u}|\boldsymbol{x}) \log \frac{\mu_t^\pi(\boldsymbol{x})\pi_t(\boldsymbol{u}|\boldsymbol{x})}{q_t(\boldsymbol{x}, \boldsymbol{u})} d\boldsymbol{x} d\boldsymbol{u}, \tag{13}$$

and for $t = H$ by

$$\epsilon \geq \int \mu_H^\pi(\boldsymbol{x}) \log \frac{\mu_H^\pi(\boldsymbol{x})}{q_H(\boldsymbol{x})} d\boldsymbol{x}. \tag{14}$$

We have now stated all ingredients for the full optimization problem that defines the information theoretic SOC algorithm

$$\begin{aligned}
\text{argmax}_{\pi,\mu^\pi} \quad & \sum_{t=1}^{H-1} \iint \mu_t^\pi(\boldsymbol{x})\pi_t(\boldsymbol{u}|\boldsymbol{x}) r_t(\boldsymbol{x}, \boldsymbol{u}) d\boldsymbol{x} d\boldsymbol{u} + \int \mu_H^\pi(\boldsymbol{x}) r_H(\boldsymbol{x}) d\boldsymbol{x}, \\
\text{s.t:} \quad & \iint \mu_t^\pi(\boldsymbol{x})\pi_t(\boldsymbol{u}|\boldsymbol{x}) d\boldsymbol{x} d\boldsymbol{u} = 1 \text{ for } t < H \text{ and } \int \mu_H^\pi(\boldsymbol{x}) d\boldsymbol{x} = 1, \\
1 < t < H: \quad & \int \mu_{t+1}^\pi(\boldsymbol{x}')\boldsymbol{\phi}(\boldsymbol{x}') d\boldsymbol{x}' = \iiint \mu_t(\boldsymbol{x})\pi_t(\boldsymbol{u}|\boldsymbol{x}) p(\boldsymbol{x}'|\boldsymbol{x}, \boldsymbol{u})\boldsymbol{\phi}(\boldsymbol{x}') d\boldsymbol{x} d\boldsymbol{u} d\boldsymbol{x}', \\
t = 1: \quad & \int \mu_1^\pi(\boldsymbol{x}')\boldsymbol{\phi}(\boldsymbol{x}') d\boldsymbol{x}' = \hat{\boldsymbol{\phi}}_1, \\
1 \leq t < H: \quad & \int \mu_t^\pi(\boldsymbol{x})\pi_t(\boldsymbol{u}|\boldsymbol{x}) \log \frac{\mu_t^\pi(\boldsymbol{x})\pi_t(\boldsymbol{u}|\boldsymbol{x})}{q_t(\boldsymbol{x}, \boldsymbol{u})} d\boldsymbol{x} d\boldsymbol{u} \leq \epsilon, \\
t = H: \quad & \int \mu_H^\pi(\boldsymbol{x}_H) \log \frac{\mu_H^\pi(\boldsymbol{x}_t)}{q_H(\boldsymbol{x}_H)} d\boldsymbol{x} \leq \epsilon.
\end{aligned} \tag{15}$$

Note that the constraints are always satisfiable as $q_t(\boldsymbol{x}, \boldsymbol{u})$ has been generated from roll-outs on our real system, and, hence, $\mu_t^\pi(\boldsymbol{x})\pi_t(\boldsymbol{u}|\boldsymbol{x}) = q_t(\boldsymbol{x}, \boldsymbol{u})$ is always a valid solution, although not the optimal one.

The stated optimization problem can be solved by the method of Lagrangian multipliers. The state-action probability $p_t^\pi(\boldsymbol{x}, \boldsymbol{u}) = \mu_t^\pi(\boldsymbol{x})\pi_t(\boldsymbol{u}|\boldsymbol{x})$ can be obtained for each sample in closed form and is given by

$$p_t^\pi(\boldsymbol{x}, \boldsymbol{u}) \quad \propto \quad q_t(\boldsymbol{x}, \boldsymbol{u}) \exp\left(\frac{A_t(\boldsymbol{x}, \boldsymbol{u})}{\eta_t}\right), \tag{16}$$

$$A_t(\boldsymbol{x}, \boldsymbol{u}) \quad = \quad r_t(\boldsymbol{x}, \boldsymbol{u}) + \mathbb{E}_{p_t(\boldsymbol{x}'|\boldsymbol{x}, \boldsymbol{u})}[v_{t+1}(\boldsymbol{x}')] - v_t(\boldsymbol{x}) \tag{17}$$

where $v_t(\boldsymbol{x}) = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\theta}_t$. The parameters η_t and $\boldsymbol{\theta}_t$ denote the Lagrangian multipliers for the KL-constraints and the constraints for $\mu_t^\pi(\boldsymbol{x})$, respectively. The Lagrangian multipliers can be found by minimizing the dual function $g(\eta_{1:H}, \boldsymbol{\theta}_{1:H})$, i.e.,

$$\{\eta_{1:H}^*, \boldsymbol{\theta}_{1:H}^*\} = \text{argmin}_{\eta_{1:H}, \boldsymbol{\theta}_{1:H}} g(\eta_{1:H}, \boldsymbol{\theta}_{1:H}), \quad s.t : \eta_t > 0, \forall t. \tag{18}$$

The derivation of the dual function is given in the appendix. We can see that our old data distribution $q_t(\boldsymbol{x}, \boldsymbol{u})$ is weighted by an exponential function. The term $A_t(\boldsymbol{x}, \boldsymbol{u})$ in the nominator strongly resembles the structure of an advantage function, if we would assume that $v_{t+1}(\boldsymbol{x}')$ denotes a value function. However, such a relationship can not be established. To improve our understanding of the meaning of the function v_t, we examine the structure of the dual function in more detail. The dual function $g(\eta_{1:H}, \boldsymbol{\theta}_{1:H})$ is given by

$$g(\eta_{1:H}, \boldsymbol{\theta}_{1:H}) = \hat{\boldsymbol{\phi}}_1^T \boldsymbol{\theta} + \sum_{t=1}^{H} \eta_t \epsilon + \sum_{t=1}^{H} \eta_t \log Z_t, \text{ with} \tag{19}$$

$$Z_{t<H} = \iint q_t(\boldsymbol{x}, \boldsymbol{u}) \exp\left(\frac{A_t(\boldsymbol{x}, \boldsymbol{u})}{\eta_t}\right) d\boldsymbol{x} d\boldsymbol{u}, \tag{20}$$

$$Z_H = \int q_H(\boldsymbol{x}) \exp\left(\frac{r_H(\boldsymbol{x}) - v_H(\boldsymbol{x})}{\eta_H}\right) d\boldsymbol{x}. \tag{21}$$

We can see that, due to the function $A_t(\boldsymbol{x}, \boldsymbol{u})$, each $v_t(\boldsymbol{x})$ is coupled with the function $v_{t-1}(\boldsymbol{x})$ from the previous as well as with $v_{t+1}(\boldsymbol{x})$ from the next time step. Hence,

all functions $v_t(\boldsymbol{x})$ are connected and need to be optimized at once. While the optimization of the dual function is more expensive than the traditional backwards iteration for obtaining the value function, it is also the biggest strength of the information theoretic policy update. The vectors $\boldsymbol{\theta}_t$ need to be optimized simultaneously as changing the policy at time step t changes both, the state distributions $\mu^{\pi}_{k>t}(\boldsymbol{x})$ for future time steps and the expected reward for time steps in the past. In contrast to approximate dynamic programming, the computation of the value function as intermediate step is not necessary. In the ITSOC formulation, the function $v_t(\boldsymbol{x})$ and the policy are always connected and we cannot compute one term without the other.

3.3 Sample-based ITSOC

While ITSOC offers an analytical solution for the policy, the dual function can in general not be evaluated analytically due to the integrals over the state action space where the analytical form of the reward function is unknown. However, the dual function can be straightforwardly approximated by samples obtained from the distributions $q_t(\boldsymbol{x}, \boldsymbol{u})$. The sample-based dual function is given in the Appendix A. As a consequence, we can also only obtain the probabilities $\mu^{\pi}_t(\boldsymbol{x})\pi_t(\boldsymbol{u}|\boldsymbol{x})$ for a discrete set of samples.

In order to generalize these sample-based policy, we need to resort to a parametric policy $\tilde{\pi}_t(\boldsymbol{u}|\boldsymbol{x};\boldsymbol{\omega}_t)$, where $\boldsymbol{\omega}_t$ denotes the parameters of the policy. The parameters $\boldsymbol{\omega}_t$ can be found by approximating the sample-based distribution $\pi_t(\boldsymbol{u}|\boldsymbol{x})$. We implement this approximation by minimizing the Kullback-Leibler divergence $\mathrm{KL}(\pi_t(\boldsymbol{u}|\boldsymbol{x})||\tilde{\pi}_t(\boldsymbol{u}|\boldsymbol{x};\boldsymbol{\omega}_t))$ between $\pi_t(\boldsymbol{u}|\boldsymbol{x})$ and $\tilde{\pi}_t(\boldsymbol{u}|\boldsymbol{x};\boldsymbol{\omega}_t)$ on the given set of samples [8], i.e.,

$$\begin{aligned} \boldsymbol{\omega}^*_t &= \mathrm{argmin}_{\boldsymbol{\omega}_t} \mathrm{KL}\left(\pi_t(\boldsymbol{u}|\boldsymbol{x})||\tilde{\pi}_t(\boldsymbol{u}|\boldsymbol{x};\boldsymbol{\omega}_t)\right), & (22) \\ &= \mathrm{argmax}_{\boldsymbol{\omega}_t} \sum_{\boldsymbol{x},\boldsymbol{u}} \exp\left(\frac{A_t(\boldsymbol{x},\boldsymbol{u})}{\eta_t}\right) \log \tilde{\pi}_t(\boldsymbol{u}|\boldsymbol{x};\boldsymbol{\omega}_t) + \text{const.} & (23) \end{aligned}$$

This optimization yields a weighted maximum likelihood estimate of $\boldsymbol{\omega}_t$ with the weightings

$$d_{i,t} = \exp(A_t(\boldsymbol{x}_i, \boldsymbol{u}_i)/\eta_t). \tag{24}$$

Note that we dropped the distribution $q_t(\boldsymbol{x}, \boldsymbol{u})$ from the weighting as the samples have already been generated by q_t. With the same trick, the distribution $q_t(\boldsymbol{x}, \boldsymbol{u})$ can also be dropped from the dual function $g(\eta_{1:H}, \boldsymbol{\theta}_{1:H})$, and, hence, the distribution $q_t(\boldsymbol{x}, \boldsymbol{u})$ does not need to be known in its analytic form.

4 Finding Locally Optimal Policies with Information Theoretic Stochastic Optimal Control

In this section, we present our resulting algorithm for estimating robust, locally optimal policies with ITSOC. We will also extend our algorithm with a local model-learning algorithm. Using the learned models, we can use our approach for model-based reinforcement learning where the system dynamics are unknown. Additionally, we can use the learned models as efficient simulator to create virtual roll-outs. We will first introduce the algorithm and subsequently explain the sub-parts of the algorithm in the following sub-section.

4.1 Algorithm

In each iteration, we use the currently estimated policy $\tilde{\pi}_t(\boldsymbol{u}|\boldsymbol{x}; \boldsymbol{\omega}_t)$ to create N_{real} roll-outs. Since we assume the sampling to be an expensive process, we construct the current set of samples $\mathcal{D}$ from the last L iterations. The samples from the real system are now used to train the linear time-varying models, see Section 4.3. These models are again used to generate a large amount of virtual samples $\tilde{\mathcal{D}}$ that are finally used to

Input: KL-bound ϵ, number of iterations K, samples N and virtual samples M, L last iterations to reuse
Initialize $\tilde{\pi}_t^0$ using Gaussians with zero mean and high variance.

for $k = 1$ to K ... # iterations

- **Collect data on the real system following** $\tilde{\pi}_t^{k-1}$:
  $\mathcal{D}_k = \{\boldsymbol{x}_{i,t}, \boldsymbol{u}_{i,t}\}_{i=1\ldots N, t=1\ldots H}$
- **Re-use last** L **iterations**: $\mathcal{D} = \{\mathcal{D}_l\}_{l=\max(1,k-L)\ldots k}$
- **Estimate time-varying linear models using** $\mathcal{D}$
- **Collect data on the learned system following** $\tilde{\pi}_t^{k-1}$:
  $\tilde{\mathcal{D}}_k = \{\tilde{\boldsymbol{x}}_{i,t}, \tilde{\boldsymbol{u}}_{i,t}\}_{i=1\ldots M, t=1\ldots H}$
- **Minimize dual function on** $\tilde{\mathcal{D}}_k$, see Sections 3.2 and Algorithm 2.
- **Estimate new policy** $\tilde{\pi}_t^k$ **for each** t:
  - Compute weighting $d_{t,i}$, see Eq. (24)
  - Compute policy parameters $\boldsymbol{k}_t, \boldsymbol{K}_t, \boldsymbol{\Sigma}_t$ using $\tilde{\mathcal{D}}_k$ and weightings $d_{t,i}$, see Eq. (26)

Output: Policies $\tilde{\pi}_t^K(\boldsymbol{u}|\boldsymbol{x})$ for all $t = 1, \ldots, H$

Table 1: The information-theoretic SOC algorithm. We collect data on the real system which we use to estimate individual linear models for each time step. These models are used to generate virtual samples by simulating whole roll-outs. We minimize the dual-function defined on these virtual samples and, subsequently, compute a weighting for each data point. This weighting is used to obtain new policy parameters by using a weighted maximum likelihood estimate.

update the policy. We then use these virtual samples $\tilde{\mathcal{D}}$ to solve the ITSOC optimization problem, see Eq. (15), and obtain a new sample-based policy, see Section 3.2 and 4.5. The feature representation we use is introduced in Section 4.4. In Section 4.2, we generalize the sample-based policy to a new parametric policy by a weighted maximum likelihood estimate. We start the algorithm with a rather large variance for the initial policy π_t^0. At each iteration, the policy improves towards the (locally) optimal policy. The ITSOC algorithm decreases the exploration of the estimated policy automatically in each iteration until it finally collapses to a deterministic policy. The amount of reduction of the exploration is determined by the KL-bound ϵ. The resulting algorithm is summarized in Algorithm 1.

4.2 Representation of the Policy

Inspired by optimal controllers in the LQG case [26, 30], we will use a linear representation for the policy, i.e.,

$$\pi_t(\boldsymbol{u}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{u}|\boldsymbol{k}_t + \boldsymbol{K}_t\boldsymbol{x}, \boldsymbol{\Sigma}_t).$$

The parameters $\boldsymbol{k}_t$, $\boldsymbol{K}_t$ and $\boldsymbol{\Sigma}_t$ of the policy can be efficiently estimated by a weighted linear regression. Assuming a set of samples $\mathcal{D} = \{\boldsymbol{x}_{i,t}, \boldsymbol{u}_{i,t}\}_{i=1...M,t=1...H}$ and the weightings $d_{i,t}$, the parameters $\boldsymbol{k}_t$, $\boldsymbol{K}_t$ and $\boldsymbol{\Sigma}_t$ can be computed by

$$\begin{bmatrix} \boldsymbol{k}_t^T \\ \boldsymbol{K}_t^T \end{bmatrix} = (\boldsymbol{X}_t^T \boldsymbol{D}_t \boldsymbol{X}_t)^{-1} \boldsymbol{X}_t^T \boldsymbol{D}_t \boldsymbol{U}_t, \tag{25}$$

$$\boldsymbol{\Sigma}_t = \frac{\sum_i d_{i,t}(\boldsymbol{\mu}_{i,t} - \tilde{\boldsymbol{u}}_{i,t})(\boldsymbol{\mu}_{i,t} - \tilde{\boldsymbol{u}}_{i,t})^T}{\sum_i d_{i,t}}, \tag{26}$$

where

$$\boldsymbol{X}_t = \begin{bmatrix} 1 & \boldsymbol{x}_{i,t} \\ \vdots & \\ 1 & \boldsymbol{x}_{M,t} \end{bmatrix}, \quad \boldsymbol{U}_t = \begin{bmatrix} \boldsymbol{u}_{i,t} \\ \vdots \\ \boldsymbol{u}_{M,t} \end{bmatrix}, \quad \boldsymbol{D}_t = \text{diag}([d_{i,t}]_{i=1...M}) \tag{27}$$

and

$$\boldsymbol{\mu}_{i,t} = \boldsymbol{k}_t + \boldsymbol{K}_t \boldsymbol{x}_{i,t}. \tag{28}$$

Note that we explicitly estimate a stochastic policy. The stochasticity of the policy is used for exploration. The covariance matrix $\boldsymbol{\Sigma}_t$ is based on our weighted samples, and, hence, the 'exploration direction' of our policy also adapts to our weighted samples. The exploration of the policy can be controlled by the KL-bound parameter ϵ. With a larger ϵ the policy will get more greedy and reduce exploration more quickly. With a

small ϵ, the new policy will continue to a explore a very similar state-action space as the old policy.

4.3 Learning Local Models for Reinforcement Learning

Since we also want to use our approach for RL, we will learn the system dynamics model $p_t(\boldsymbol{x}'|\boldsymbol{x}, \boldsymbol{u})$ from our generated samples. Following up our assumption of locality, we will use simple linear Gaussian models for each time step, i.e., $p_t(\boldsymbol{x}'|\boldsymbol{x}, \boldsymbol{u}) = \mathcal{N}(\boldsymbol{x}'|\boldsymbol{a}_t + \boldsymbol{A}_t\boldsymbol{x} + \boldsymbol{B}_t\boldsymbol{u}, \boldsymbol{C}_t)$. We obtain the parameters $\boldsymbol{a}_t, \boldsymbol{A}_t, \boldsymbol{B}_t$ and $\boldsymbol{C}_t$ of the models from our sampled data points $(\boldsymbol{x}_{i,t}, \boldsymbol{u}_{i,t})$ by a standard maximum likelihood estimate. As our experiments show in Section 5, the representation as time-dependent linear models also allows for a more efficient optimization of the policies, resulting in policies of higher quality. In the RL formulation of our algorithm, we assume that the reward function r_t is known as prior knowledge and only the system dynamics need to be learned.

The main difference to SOC approaches that are based on linearisations of the model, is that we use stochastic policies and explicitly control the exploration of the policy by estimating the covariance matrix $\boldsymbol{\Sigma}_t$ used for exploration. Hence, we also do not work with a single trajectory as linearisation point, but we use a whole distribution of trajectories to obtain the linear models. As we learn our models from data, our algorithm is equally suited for model-based reinforcement learning and SOC. For model-based RL, we only generate few samples on the real system and use the samples of the last L iterations to estimate the models. The stochastic policy estimated by ITSOC provides an efficient exploration strategy while the KL-bounds ensure that the ITSOC planner stays close to the data.

4.4 Local Feature Representation and Computation of Expected Features

We will use a quadratic representation for the functions $v_t(\boldsymbol{x})$, i.e., $v_t(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{V}_t \boldsymbol{x} + \boldsymbol{v}_t^T \boldsymbol{x}$, where $\boldsymbol{V}_t$ is an upper triangular matrix. The parameters $\boldsymbol{\theta}_t = \{\boldsymbol{v}_t, \boldsymbol{V}_t\}$ of v_t are obtained by optimizing the dual function which is explained in more detail in Section 4.5.

The information-theoretic formulation requires estimating the expected feature vector $\mathbb{E}_{p_t(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u})}[v_{t+1}(\boldsymbol{x}')]$ for each sample. Due to the representation of the system dynamics as linear time-varying Gaussian models and the quadratic feature representation, this operation can be performed analytically by

$$\begin{aligned}\mathbb{E}_{p_t(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u})}[v_{t+1}(\boldsymbol{x}')] &= \int \mathcal{N}(\boldsymbol{x}'|\boldsymbol{a}_t + \boldsymbol{A}_t\boldsymbol{x} + \boldsymbol{B}_t\boldsymbol{u}, \boldsymbol{C}_t)\left(\boldsymbol{x}^T\boldsymbol{V}_t\boldsymbol{x} + \boldsymbol{v}_t^T\boldsymbol{x}\right) d\boldsymbol{x}' \\ &= \boldsymbol{\mu}_t(\boldsymbol{x},\boldsymbol{u})^T \boldsymbol{V}_t \boldsymbol{\mu}_t(\boldsymbol{x},\boldsymbol{u}) + \text{trace}(\boldsymbol{C}_t\boldsymbol{V}_t) + \boldsymbol{v}_t^T\boldsymbol{\mu}_t(\boldsymbol{x},\boldsymbol{u}) \quad (29)\end{aligned}$$

with $\boldsymbol{\mu}_t(\boldsymbol{x},\boldsymbol{u}) = \boldsymbol{a}_t + \boldsymbol{A}_t\boldsymbol{x} + \boldsymbol{B}_t\boldsymbol{u}$. Equation (29) can be rewritten in the feature vector representation $\mathbb{E}_{p_t(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u})}[\boldsymbol{\phi}(\boldsymbol{x}')^T]\boldsymbol{\theta}_{t+1}$ which we omit due to space constraints. The computation of the expected features in closed form is an important extensions of the policy search algorithm presented in [8]. In [8], a single sample estimate is used to approximate the expectation, i.e., $\mathbb{E}_{p_t(\boldsymbol{x}'|\boldsymbol{x}_{i,t},\boldsymbol{u}_{i,t})}[v_t(\boldsymbol{x}')] \approx v_{t+1}(\boldsymbol{x}_{i,t+1})$. This approximation induces a bias in the resulting policy as the expectation appears in the exponential function, see Equation (16), and due to the fact that $\exp\big(\mathbb{E}[f(\boldsymbol{x})]\big) \neq \mathbb{E}\big[\exp(f(\boldsymbol{x}))\big]$. As we will show in our experiments, the bias grows with the stochasticity in the system, making the policy search method proposed in [8] inapplicable in the SOC setup.

4.5 Optimizing the Dual

In our ITSOC setup, we want to cope with time horizons as large as $H = 50$ or $H = 100$ steps. Additionally, the quadratic feature representation results in 40 to 60 parameters per time step for a moderate number of state dimensions, e.g. ten state variables. Hence, we easily end up with up to several thousands of parameters for the minimization of the dual function. In comparison to the formulation in [8], which has been used to optimize high-level policies, the number of parameters is increased by a factor of 10 to 100. With such a high number of parameters, the constraint optimization problem of the dual function runs into numerical problems and we could not use the algorithm proposed in [8]. Hence, an efficient and numerically stable optimization procedure is required.

In this section we will elaborate on the different techniques that we approached. In all cases, we provided the algorithms with the analytic gradients and Hessians.

Constrained optimization Since the dual is constrained by the lower bound for $\eta_{1:T} > 0$, a constrained optimizer is the most intuitive approach. In our case we used a trust-region-reflective algorithm [6, 17]. Unfortunately the optimizer was not able to find reliable solutions in a suitable time, due to numerical instabilities and the large amount of parameters.

Unconstrained optimization A common approach to circumvent lower bound constraints is to apply the $\exp$-trick [25], in our case resulting in an unconstrained problem. We substituted η by a non-negative function $\eta = f(\zeta) = \exp(\zeta)$ and optimize for ζ instead of η. Again, the optimizer did not achieve satisfactory results. We suspect the cause to be the introduction of additional non-linearities in combination with numerical instability. We also tried different sigmoidal substitution functions, e.g. arctangent, logistic or algebraic functions. None of these methods showed any significant improvements.

Iterative optimization A closer look at the dual reveals, that the Lagrangian parameters η and $\boldsymbol{\theta}$ can be optimized separately using a coordinate descent [4] like method. This separation is also desirable since the original dual function is convex in $\boldsymbol{\theta}$ and non-convex in η. It is convex in $\boldsymbol{\theta}_{1:H}$ as the dual function has a *log-sum-exp* structure, where the parameters $\boldsymbol{\theta}_t$ show up linearly in the $\exp$ terms. First, we optimize for each η_t individually while fixing the $\boldsymbol{\theta}_t$ parameters. Next, we fix the η parameters and optimize for all $\boldsymbol{\theta}_t$. We iterate over these two steps until we reach a satisfactory solution. The optimization for $\boldsymbol{\theta}_t$ is convex and unconstrained. Therefore, we used a large-scale method [17] for this optimization, whereas we applied a trust-region-reflective optimization [17] for each of the η_t. Both algorithms only run for a small number of iterations, e.g, 10 iterations. Each optimization is initialized with the result from the previous optimization, which increases the performance significantly. Using this approach, we were able to find good solutions in a suitable amount of time.

We experienced that using the progress of the dual value as a termination criteria for the optimization is rather ineffective as both constraints can be already satisfied with quite high accuracy while the dual value is still far from converging. Therefore, we introduced a new termination criteria which relaxes the KL and the feature constraints. At each iteration, we compute the maximum absolute error MAE_ϵ between the current KL divergence of each time step and the desired KL divergence ϵ. If the error is smaller then a certain threshold $\Delta\epsilon$, we assume the constraint as approximately satisfied. Additionally, we compute the maximum absolute error $\text{MAE}_{\boldsymbol{\theta}}$ for the average state features, where we normalize the errors with the standard deviation of the corresponding state feature. We again regard the feature constraint as sufficiently satisfied if $\text{MAE}_{\boldsymbol{\theta}}$ is beneath the threshold $\Delta\boldsymbol{\theta}$. If both constraints are satisfied, we stop the iterations. Applying these termination criteria, we experienced only small penalties in the performance of the algorithm while benefiting greatly in terms of computation time. The resulting algorithm for optimizing the dual is given in Algorithm 2.

Input: Initial estimate for all $\boldsymbol{\theta}$, initial estimates for all η, dataset $\mathcal{D}$ **Compute initial** MAE(ϵ) and MAE($\boldsymbol{\theta}$)
while MAE(ϵ) $> \Delta\epsilon$ ‖ MAE($\boldsymbol{\theta}$) $> \Delta\boldsymbol{\theta}$ **Optimize** g **for each** η_t: $\eta_t = \arg\min_{\eta_t} g(\eta_{1:H}, \boldsymbol{\theta}_{1:H}; \mathcal{D}), \forall t$ (Eq. 38) **Optimize** g **for all** $\boldsymbol{\theta}_t$: $\boldsymbol{\theta}_{1:H} = \arg\min_{\boldsymbol{\theta}_{1:H}} g(\eta_{1:H}, \boldsymbol{\theta}_{1:H}; \mathcal{D})$ (Eq. 38) **Compute** MAE(ϵ) and MAE($\boldsymbol{\theta}$)
Output: optimized $\boldsymbol{\theta}_{1:H}$ and $\eta_{1:H}$

Table 2: Iterative optimization of the dual function. We decompose the optimization problem in finding the single η_t values while keeping the $\boldsymbol{\theta}$ value fixed. Subsequently, we optimize for the $\boldsymbol{\theta}_{1:H}$ values, which is an unconstrained optimization problem. Both algorithms are only run for a small number of internal optimization steps to avoid oscillations in the parameter updates. As termination criteria for our iterative optimization procedure, we check whether our constraints are approximately met. If this is the case, the optimization is stopped.

4.6 Discussion of different KL-bounds

There are tight connections but also important differences to SOC algorithms that use KL-terms [23, 2]. The KL-terms in these approaches act on the trajectory distribution $p(\boldsymbol{\tau})$ instead of on the marginal distributions $p_t(\boldsymbol{x}, \boldsymbol{u})$. The KL on the trajectory distribution results in a sum over the KL's of the policies $\pi_t(\boldsymbol{u}|\boldsymbol{x})$ for each time step as the transition model cancels out in the log terms. A consequence of the KL acting on the policy is that these approaches have to rely on value function approximation or related approximations. Hence, they suffer from the drawbacks of an unstable policy update that comes with the value function approximation. [23, 2] use the KL as punishment term that is traded-off with the traditional cost function. As a consequence, the temperature parameter η has to be chosen by the user or set by heuristics [27] while in ITSOC it is given from the optimization problem. In ITSOC, the user has to choose the ϵ parameter, that is typically much easier to choose and can stay constant during the iterations.

The path integral (PI) formulation of SOC [27] also shares a lot of similarities with the ITSOC formulation. We can clearly see that the value function given for the path integrals in Eq. 8 and the dual function for ITSOC share the same structure. However, in ITSOC, the integral is performed over the state-action space while in the PI

formulation, the integral is only performed over the future trajectory space conditioned on the starting state $\boldsymbol{x}_0$. In theory, such formulation requires that we restart the process for many times at state $\boldsymbol{x}_t$ to obtain the optimal controls for state $\boldsymbol{x}_t$. This requirement is typically neglected in common PI implementations [27], which is, however, only a heuristic that can be used for learning open-loop controllers but not for feedback controllers. Furthermore, the PI approach also does not control the damage on the state distributions by the policy update. It is based on Monte-Carlo roll-outs and therefore requires a lot of samples. The temperature parameters η_t are also chosen by heuristics in the PI approach.

Our algorithm was inspired by the policy search community, where related information-theoretic bounds have already been introduced to learn high-level policies [8]. The algorithm given [8] did not use a model for estimating the expected next state features and, therefore, produced biased solutions. Moreover, the optimization problem in [8] could be solved straightforwardly, as the number of parameters was much lower. In order to apply the information-theoretic policy updates for SOC, we needed to develop our more efficient and numerically stable optimization strategy.

In order to evaluate the benefits of the KL term acting on the state action distribution we implemented two more algorithms that are variants of existing algorithms, but use different KL-based penalty terms. The first approach directly optimizes the reward on the learned linear models, i.e., it jumps to the greedy solution without penalizing any deviation from the KL-bound. The second approach uses a KL-bound on the policy, i.e., $\mathrm{KL}\big(\pi(\boldsymbol{u}|\boldsymbol{x})||q(\boldsymbol{u}|\boldsymbol{x})\big) \leq \epsilon$. Here, we adapt an existing dynamic programming based method to work with similar model assumptions as we use, i.e., a quadratic, time-dependent value function, and time-dependent linear policies.

Removing the KL-bound This approach is strongly related to the model-based PILCO algorithm where we use the same time-dependent linear models in order to obtain a fair comparison. At each iteration, we obtain the learned models in a similar fashion as for

the ITSOC algorithm. The objective of PILCO can be summarized by

$$\pi_{1:H} = \text{argmax}_{\pi_{1:H}} \mathbb{E}_{\tilde{p},\pi}\left[\sum_{t=1}^{H-1} r_t(\boldsymbol{x},\boldsymbol{u}) + r_H(\boldsymbol{x})\right],$$

i.e, it greedily optimizes the expected reward where the expectation is performed with respect to the *learned* system dynamics $\tilde{p}(\boldsymbol{x}'|\boldsymbol{x},\boldsymbol{u})$. In difference to the standard PILCO algorithm, we do not use a gradient-based optimizer to obtain the optimal policies but an optimal control algorithm called AICO [31], that uses second order expansions of the reward function and the learned linear models to obtain the new policy. Note that this variant of PILCO can also be seen as a variant of AICO, where the linearizations are not obtained from a known system model along a given trajectory, but learned from data. The resulting policy is optimal with respect to the learned model and is therefore a *deterministic* policy. Hence, we added a fixed noise to the policy to mimic exploration. We optimized for the noise parameter by cross validation.

KL-bound on the trajectory distribution For comparisons, we will use a variant of the advantage weighted regression (AWR) algorithm [19] to compare the effects of the KL-bound on the trajectory distribution versus the KL-bounds on the state action distribution marginals. As discussed in Section 2.2, a KL penalizing term on the trajectory distribution results in the expected KL between the policies for each time step. Hence, The objective of the algorithm can be summarized as

$$\pi_{1:H} = \text{argmax}_{\pi_{1:H}} \mathbb{E}_{\tilde{p},\pi}\left[\sum_{t=1}^{H-1} r_t(\boldsymbol{x},\boldsymbol{u}) + r_H(\boldsymbol{x})\right], \tag{30}$$

$$\text{s.t: } \mathbb{E}_{\mu_t(\boldsymbol{x})}[\text{KL}\big(\pi_t(\boldsymbol{u}|\boldsymbol{x})||q_t(\boldsymbol{u}|\boldsymbol{x})\big)] \leq \epsilon, \quad \forall t \tag{31}$$

In our variant of AWR, we also use a time-dependent quadratic value function and time-dependent linear feedback controllers as policy representation. We reformulate AWR such that we can optimize the temperature of the soft-max distribution with a

similar information-theoretic bound. Therefore, we formulate a similar optimization problem for ITSOC, which can, however, be solved independently for each time step. In time step t, we want to maximize the expected advantage-function while we keep the KL between the new and old policy bounded, i.e.,

$$\max_{\pi_t} \int_{\boldsymbol{x},\boldsymbol{u}} q_t(\boldsymbol{x})\pi_t(\boldsymbol{u}|\boldsymbol{x})A_t(\boldsymbol{x},\boldsymbol{u}), \quad \text{s.t:} \int_{\boldsymbol{x}} q_t(\boldsymbol{x})\mathrm{KL}(\pi_t(\boldsymbol{u}|\boldsymbol{x})||q_t(\boldsymbol{u}|\boldsymbol{x}))d\boldsymbol{x} \leq \epsilon. \tag{32}$$

The solution for $\pi_t(\boldsymbol{u}|\boldsymbol{x})$ can be obtained similarly by the method of Lagrangian multipliers and is given by

$$\pi_t(\boldsymbol{u}|\boldsymbol{x}) = q_t(\boldsymbol{u}|\boldsymbol{x}) \exp\left(\frac{A_t(\boldsymbol{x},\boldsymbol{u})}{\eta_t}\right). \tag{33}$$

We compute the advantage of the given samples $\boldsymbol{x}_{i,t}$ and $\boldsymbol{u}_{i,t}$ by

$$A_t(\boldsymbol{x},\boldsymbol{u}) = r_t(\boldsymbol{x},\boldsymbol{u}) + \mathbb{E}_{\boldsymbol{x}'}[V_{t+1}(\boldsymbol{x}')] - V_{t,\text{old}}(\boldsymbol{x})), \tag{34}$$

where $V_t(\boldsymbol{x})$ is obtained by an advantage weighted regression. For the advantage weighted regression, we use the states $\boldsymbol{x}_t$ as inputs and the Q-values $Q_t(\boldsymbol{x},\boldsymbol{u}) = r_t(\boldsymbol{x},\boldsymbol{u}) + \mathbb{E}_{\boldsymbol{x}'}(V_t(\boldsymbol{x}')$ as target values. The value function $V_{t,\text{old}}(\boldsymbol{x})$ for the current time step is approximated by using the samples without weighting, i.e., we compute the value function when using the old policy for the current time step while following the new policy for the future time steps. Since our KL-bound does not act on the state distributions, the optimization problem becomes much simpler, and it can be solved in a dynamic programming fashion where we compute V- and Q-functions. However, the policy update also becomes less stable as the state distributions are now allowed to jump in the policy update and the policy is additionally distorted by the value function approximation errors.

4.7 Contribution

By Reformulating the SOC framework as a constrained optimization, we are able to introduce additional bounds into the SOC framework which lead to more efficient policy updates. The used bounds on the state action distributions $p_t(\boldsymbol{x}, \boldsymbol{u})$ allow to learn high quality policies while ensuring a reliable and stable learning progress.

In this paper, we extended information-theoretic policy search in several ways such that it becomes applicable to the SOC scenario. To mimic the Linear Quadratic Regulator case, we use locally valid features for feature matching and time-dependent linear controllers. We learn the models in terms of time-dependent linear models and use these models to compute the expected next state features as well as to generate a vast amount of virtual samples, which also considerably speed up the convergence of the algorithm. All these steps are essential ingredients for SOC when we have to deal with stochastic dynamics. Finally, we extended the optimization procedure of the dual function in order to make it solvable with the large amount of parameters that come with the application of the method for SOC. Without this improved optimization, the algorithm could not finish in a reasonable amount of time.

Furthermore, we compare in the experiment section the different KL-bounds that can be used for the policy update, i.e., no KL-bound, KL-bound on the policy and KL-bound on the state-action distribution. We believe that this comparison provides important insights on how to construct stable and computationally efficient policy updates in the future.

5 Experiments

We evaluate our information-theoretic SOC algorithm on a 2-link planar arm and on a 4-link non-linear planar arm in different scenarios. The links of the arm have a length of one meter and a mass of one kilogram. The maximum torque is set to 25Nm and

we also implement a simple friction model for the joints. We use 50 time steps and the duration of the time step is dt $= 0.066$s. In addition to the control noise used in some experiments, we always use a Gaussian distribution for the initial state distribution $p_1(\boldsymbol{x})$. Hence, we do not want to estimate a single nominal trajectory but a controller which works well in a broader area for the initial state.

We compare our algorithm against our variants of advantage weighted regression (AWR), see [19] and PILCO [10], which we both adapted to use the same learned models, features and policies. A description of these algorithms can be found in Section 4.6. We also compare against the AICO algorithm [31] that uses linearisation of the underlying system dynamics. We evaluate the robustness of these algorithms to system noise and non-linearities in the dynamics. Moreover, we evaluate our approach in a model-based RL setup where we want to minimize the amount of real-robot interactions. Here, we compare our time-varying linear model to a constant linear model for which we use the data from all time steps for estimating the parameters.

5.1 Scenarios

In order to show the advantages and characteristics of ITSOC we evaluated the algorithm on various experiments, where we used a total of five scenarios. If not stated differently, we used for all scenarios an initial state distribution with a standard deviation of 0.1 for each joint position.

2-link reaching. The goal of this scenario is to reach a desired target position with the end-effector of a 2-Link planar arm at certain via-point at a certain time step. The reward function is given as a squared punishment term for the torques at each time step and a via-point reward $r_{\boldsymbol{v}}(\boldsymbol{x})$ at time steps $t_1 = 25$ and $t_2 = 50$. The via-point reward $r_v(\boldsymbol{x})$ is proportional to the negative squared distance in task space $r_{\boldsymbol{v}}(\boldsymbol{x}) = -(\boldsymbol{y} - \boldsymbol{v})^T \boldsymbol{H} (\boldsymbol{y} - \boldsymbol{v})$, where $\boldsymbol{y} = f(\boldsymbol{x})$ is the end-effector position for state $\boldsymbol{x}$ and $\boldsymbol{H}$ is set to $10^4 \boldsymbol{I}$. The state vector of the robot is 4 dimensional, resulting in a 14

dimensional feature vector.

2-link swing-up. In the second scenario for the 2-link arm, we learn a swing-up movement to evaluate the approaches on highly non-linear tasks. The goal is to swing-up and balance the pendulum such that it reaches the upright position with zero velocity at $t = 50$. In order to simplify the learning problem, we also punish the agent if the joints leave a pre-defined area of $q_1 \in [2/3\pi, 3\pi]$ and $q_2 \in [-\pi, \pi]$. This punishment term allows us to penalize overturning of the pendulum and to pre-select one of the two possible swing-up solutions. Hence, we simplify the learning problem by avoiding multi-modalities that are inherent in the solution space.

4-link non-gravity. In the first 4-link scenario, the arm has to reach two via-points in task space while we disable gravity. We use this task as a baseline to see the performance of the algorithms in case of a almost linear system dynamics. The state vector $\boldsymbol{x}$ of the robot is 8 dimensional containing all joint positions and velocities. A $d = 8$ dimensional state vector results in a 44 dimensional feature vector $\boldsymbol{\phi}(\boldsymbol{x})$, consisting of 8 linear and $d(d+1)/2 = 36$ squared terms.

4-link reaching. In this scenario, we test the robustness against non-linearities by enabling gravity. However, in order to simplify the search problem, we additionally add a high punishment term if the joint angles leave a pre-specified area. Hence, we limit the search space and avoid multi-modal solutions due to overturning of the arm. The larger the pre-specified area is, the more difficult the problem gets.

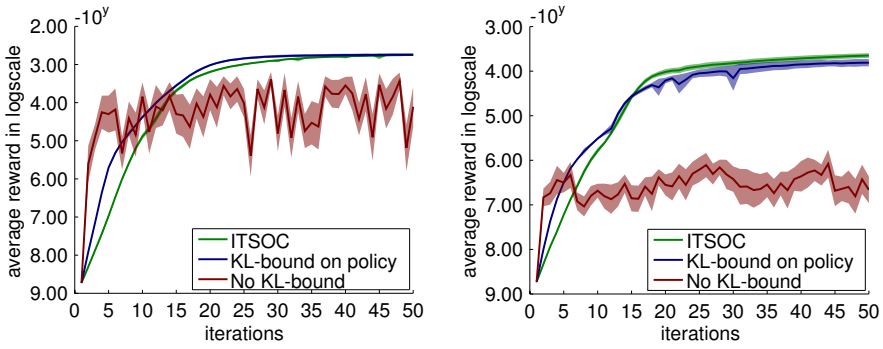
Table 3: Experiments and scenarios

Evaluation	Scenarios
KL-bounds	4-link non-gravity
	4-link reaching
	4-link tennis
State-distributions	2-link reaching
Non-linearities	2-link swing-up
Robustness	4-link reaching
Model-based RL	4-link reaching
Feature Constraints	4-link reaching

4-link tennis. Finally, we extend the 4-link arm scenario to the task of playing robot tennis. We add the position and velocity of a ball into our state space. The ball starts with random initial x-position and velocity. The ball moves with constant velocity, however, at time step 25 it bounces against the floor, introducing a perturbation. At the bounce, the velocities are perturbed by a significant amount of multiplicative noise, changing the incoming position of the ball. The agent has to hit the ball at time step 50. To do so, it has to reach a vicinity of 20cm of the ball position at $t = 50$. If it succeeds, it gets a positive reward proportional to the velocity of the end-effector in y-direction. Otherwise, the reward is proportional to the negative squared distance to the ball location. We add the position and the velocity of the ball in our state space, resulting in a 12 dimensional state vector and 90 dimensional feature vector.

Comparison of the different types KL-bounds.

In order to illustrate the effects of the different KL-bounds we compared our approach (KL-bound on the state-action marginals) to the AWR approach (KL-bound on trajectory distribution) and PILCO (no KL-bounds) on this scenario. The results are shown in Figure 1. While ITSOC and AWR found good solutions, PILCO showed even in the non-gravity task a highly unstable learning process. We optimized the fixed exploration rate for PILCO. If the exploration rate was too low, the policy update became very unstable. While the algorithm found good solutions in most trials, the average reward was poor as even the trials with good solutions tended to jump back again to bad solutions for one or two iterations. For high exploration rates, the quality of the final policy also degraded. With the additional non-linearity of gravity, PILCO performed even worse and could not find good solutions. When the KL bound was increased, we experience significant instabilities in the learning progress of AWR, while ITSOC showed a smooth learning curve with an improved convergence speed, as shown in Figure 2.



(a) 4-link reaching.No Gravity. $\epsilon = 0.3$ (b) 4-link reaching.With Gravity. $\epsilon = 0.3$

Figure 1: Evaluation of the different Algorithms on the 4-link reaching task without gravity and $\epsilon = 0.3$ (a), with gravity and $\epsilon = 0.3$ (b). Our ITSOC algorithm uses the KL-bound on the state-action marginals, while AWR uses the KL-bound on the policies and linear PILCO does not use any KL-bound. While linear PILCO suffers from instabilities in the policy update, AWR and ITSOC perform well. In the task without gravity, AWR and ITSOC converge to solutions of equal quality. However, in the case with gravity, AWR produces policies of less quality due to the additional non-linearities.

Evolution of the State-Distributions To illustrate the behavior of different algorithms, we show subsequent trajectory distributions during the iterations of the algorithms in Figure 3 for the two-link reaching task. We show the distributions of our ITSOC algorithm, the approximate dynamic programming based AWR algorithm and the linear PILCO method. We can see that the distributions change smoothly for our approach, allowing the algorithm to efficiently find an optimal solution. In contrast, the distributions jump for AWR and it quickly converges to a deterministic, but sub-optimal policy. Linear PILCO jumps to good solutions already after a few iterations, however, as it exploits the models greedily, already small inaccuracies in the model can cause jumps in the trajectory distribution.

Evaluation on a highly Non-linear Task To show that learning time-dependent linear models also works for highly non-linear tasks, we learn to swing-up a two-link pendulum. As reward function, we punish the squared distance of the end-effector to the up-right position in the last 10 time steps. The two-link pendulum is able to directly go to the upright position, however, due to the torque punishment factor, such behavior is

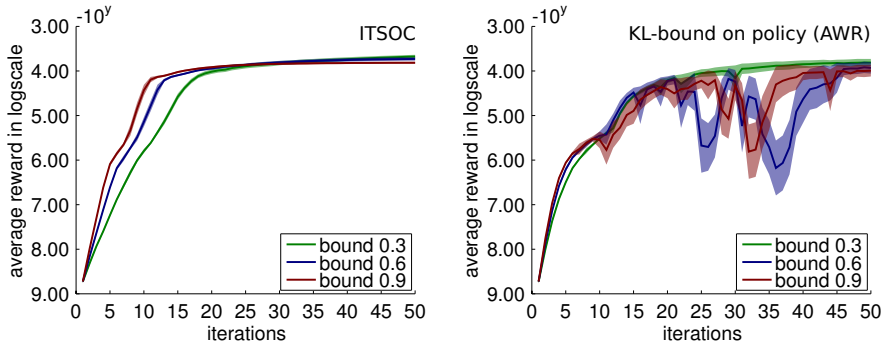


Figure 2: Evaluation of different KL-bounds ϵ for ITSOC and the modified AWR approach. We can see that while AWR only works well for a very small ϵ, ITSOC allows for the use of much higher ϵ values as it also ensures that the distance in the state-distributions stays bounded. As a consequence, ITSOC needs less iterations to converge to the optimal solution.

suboptimal. The optimal reward can only be reached by performing a swinging movement. An illustration of the learned movement can be seen in Figure 4. The agent was able to swing up the pendulum while smoothly controlling its torques. The resulting torque trajectories are shown for the ITSOC algorithm as well as for the value-based AWR method. AWR only found a suboptimal solution that exhibits considerably more jerk in the torque profile.

Robustness

We evaluated the robustness of our algorithm in terms of system noise as well as the variance of the initial state distribution. We use the 4-link reaching task and compare our algorithm to the model-free variant of our algorithm. The model-free version, originally proposed in [8] for policy search applications, does not estimate the expected features of the next state correctly[2], see Section 2.4. Hence, it can only be applied for deterministic systems.

We used additive control noise with a standard deviation of 0% and 60% of the

[2]A single sample $\boldsymbol{x}_{i,t+1}$ is used to estimate the expectation $\mathbb{E}[\boldsymbol{\phi}(\boldsymbol{x}_{t+1})|\boldsymbol{x}_{i,t}, \boldsymbol{u}_{i,t}]$

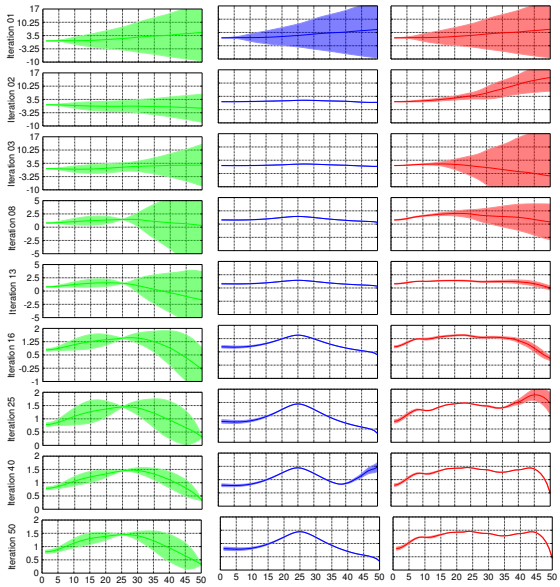
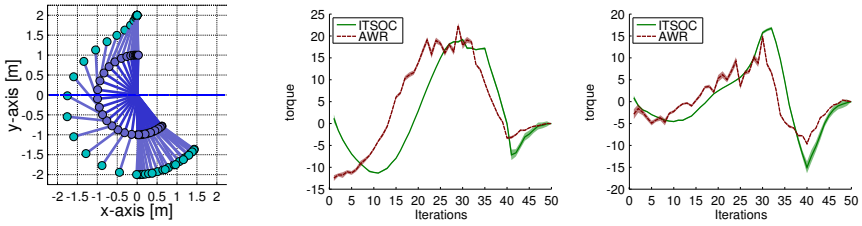


Figure 3: The plot shows the trajectory distributions for the last joint q_2 of a two link planar arm with ITSOC (left), linear PILCO (middle) and AWR (right). The x-axis depicts the time steps while in the rows we can see the distribution for subsequent iterations. While the AWR approach shows a jumping behavior and a resulting limited learning progress, the information-theoretic approach smoothly transforms the initial exploratory distribution into a goal-directed movement. PILCO quickly jumps to a good solution, but fails to further improve the policy. The average reward of ITSOC is -11.2, while for AWR it is -52.4 and PILCO oscillates between -3 and -2000.

maximum torques which can be applied by the robot. We use $N = 500$ samples per iteration and do not keep samples from old iterations to avoid effects from the sampling process. The results are shown in Figure 5. For the evaluation without noise, both, the model-free and model-based version of our algorithm estimated good policies. With an increasing level of the control noise, the performance of the model-free method quickly degraded due to the bias.

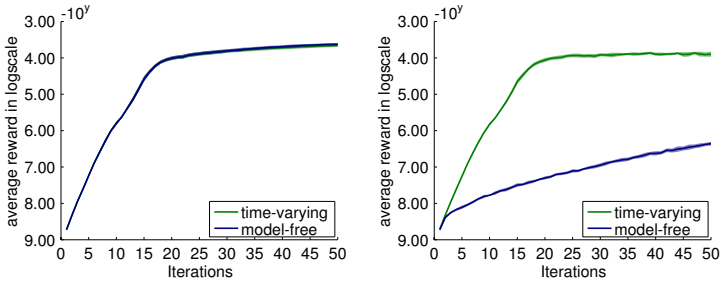
We also illustrate the resulting postures for different time points in Figure 6 for the setting with 60% noise. The robot manages to reach the via-points (illustrated by pink circles) in task space while it still exhibits a large variance in joint space. In between the via-points, also the variance in task space grows.

Finally, we evaluated the robustness of the ITSOC and the AWR algorithm with



(a) (b) (c)

Figure 4: (a)Two-Link Pendulum Swing-Up solution found by ITSOC. (b,c) The resulting torque trajectories are shown for the ITSOC algorithm as well as for the value-based AWR method. AWR only found a suboptimal solution that exhibits considerably more jerk in the torque profile.



(a) Noise = 0% (b) Noise = 60%

Figure 5: Comparison of ITSOC with learned models and without learned models where we use a single sample estimate for the expectation of the next features, which results in a bias in our optimization. The evaluations are done for different noise levels of additive control noise. Model-based ITSOC shows the best performance while the biased version of ITSOC quickly degrades with an increasing noise level. Note that all plots are in log-scale for the y-axis.

respect to the standard deviation of the initial joint configuration. The results are shown in Figure 7. As we can see, the ITSOC algorithm only slightly degraded with the increased variance in the initial state while the AWR algorithm was not able to learn high quality policies for the setting with the highest variance.

Accuracy of the Feature Constraints and Computation Time

Next, we demonstrate the influence of the threshold $\text{MAE}(\boldsymbol{\theta})$ for the state feature constraints as described in Algorithm 2. Figure 8(a) shows how the average reward decreases with a higher threshold, since the state feature constraint is further relaxed. Interestingly, the average reward is hardly affected until $\text{MAE}(\boldsymbol{\theta}) = 1.5$, but then

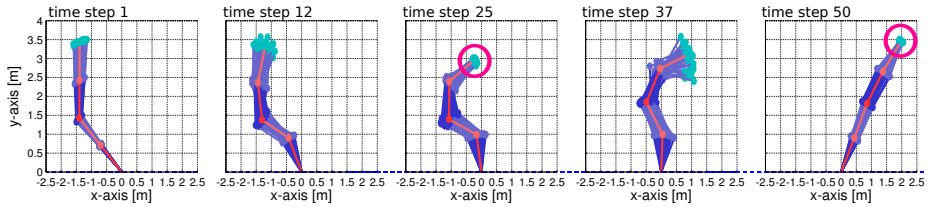
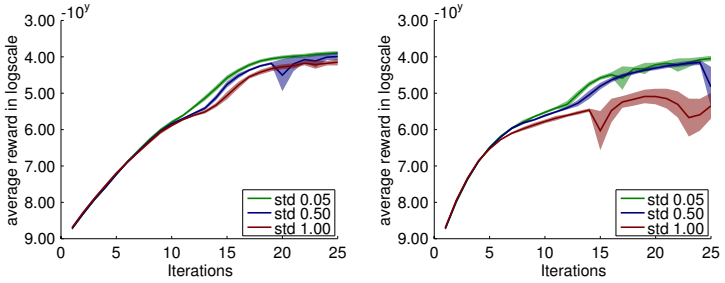


Figure 6: Illustration of the resulting postures for the via-point task in task space. The non-linear planar arm has to reach the via-points at $t = 25$ and $t = 50$ illustrated by pink circles. The plot shows for each time step samples from the resulting distribution of postures. The mean of the postures is shown in red. The robot managed to reach the via-point while exhibiting a significant amount of variance in joint space.



(a) ITSOC (b) AWR

Figure 7: Evaluation of the robustness of ITSOC and AWR to deviations of the initial state. We varied the standard deviation of the initial joint configuration from 0.05 to 1.0. While the learned policies of ITSOC only slightly degrade, the AWR algorithm could not handle the situations with a large deviations in the initial state.

quickly drops with an increasing value of MAE($\boldsymbol{\theta}$). As we can see in Figure 8(b), the time needed to compute the optimization decreases exponentially with the increasing relaxation of the feature constraints. We conclude that a value of MAE($\boldsymbol{\theta}$) between 0.5 and 1.0 is a reasonable choice, at least for this example.

We also compared the computation time of our algorithm to AWR and Linear PILCO and also evaluated the computation time of our algorithm with an increasing number of time steps. The results can be seen in Figure 9. While the other two algorithms clearly outperform our algorithm in terms of computation time, our algorithm runs in several hours which is still acceptable. We can also see a moderate increase of computation time if we increase the number of time steps. Improving the scalability of the approach in terms of computation time is part of future research.

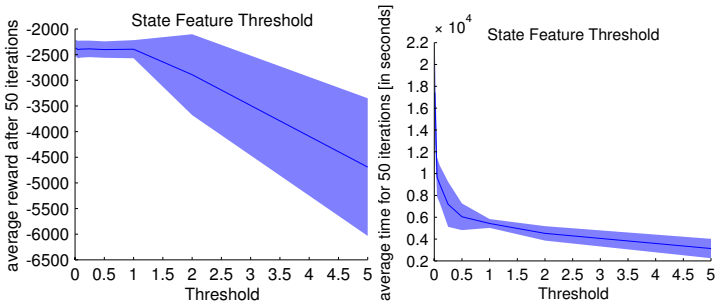


Figure 8: (a) The effect of increasing relaxation of the feature constraints MAE($\boldsymbol{\theta}$) on the average reward. (b) Computation time for different MAE($\boldsymbol{\theta}$). While the performance of ITSOC is almost unaffected for MAE($\boldsymbol{\theta}$) ≤ 1.0, the computation time depends exponentially on MAE($\boldsymbol{\theta}$). Hence, a reasonable choice for MAE($\boldsymbol{\theta}$) ≤ 1.0 in this example is 1.0.

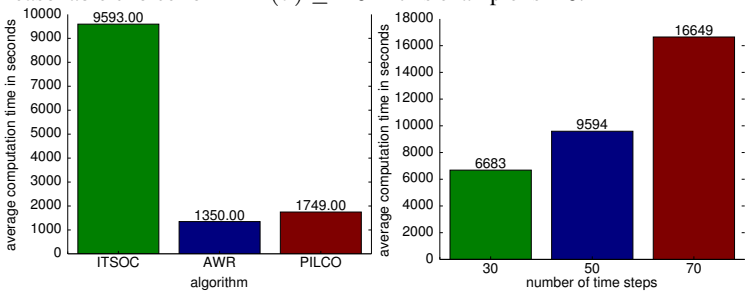


Figure 9: (a) Comparison of the computation time of the different algorithms. The bars show the average time needed for 50 iterations on the quad-link reaching task. (b) Computation time of ITSOC for an increasing number of time steps.

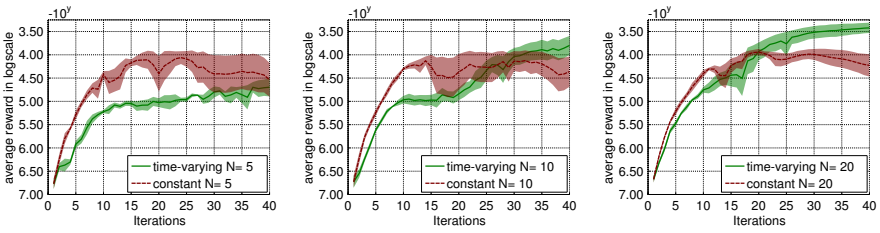
Because of the computational complexity of our method, AWR can perform many more iterations than ITSOC in the same amount of computation time. Therefore, if the amount of computation time is equal for both methods, AWR will produce policies of much higher quality. However, Figure 1(b) shows that ITSOC is able to outperform AWR after a certain number of iterations and converges to a better solution than AWR. Consequently, if the limiting factor is the data collection and not the computation time, as it is often the case for real robot experiments, ITSOC will find a better solution than AWR in a shorter amount of time.

Model-Based Reinforcement Learning

In this experiment, we evaluate the ability of our approach for model-based reinforcement leaning. We use the 4-link pendulum task with gravity and two via-points in joint space. For our evaluation, we use a different number of new samples $N = 5$ and $N = 10$ and we always keep the data of the last $L = 10$ iterations. Since we stay close to the data we can discard older iterations without the risk of sampling from completely unknown or badly rewarded areas. At the same time the KL bound ensures a certain exploration while narrowing down to the optimal solution. With the collected data we learn either a time-varying or a constant linear model. The constant linear model has the advantage that it can use more data points, however, it can not capture the system dynamics as well as the time-varying linear models. The comparison is shown in Figure 10. We can see that for a small number of samples, the time-varying models have too little data-points and consequently the constant model outperforms the time-varying model. However, for an increasing number of new samples, the time-varying model is still slower in the beginning, but outperforms the constant model in the end. It converges to a summed reward of -300 in comparison to a summed reward of -10000 for the constant model. In comparison to other reinforcement learning approaches the results are promising and the movement could be learned within 200 to 300 episodes, however, we can not keep up with state of the art model-based RL approaches [10] as it learns time-independent GP models, which is more data-efficient. Yet, the performance of our approach could be drastically improved by using more sophisticated model learning methods, for example, learning an hierarchical prior which connects the models of the single time steps.

Robot Tennis

In the robot tennis example, we used 100 samples per iteration and learned the time-varying models with the last 500 samples. The robot could reliably hit the incoming



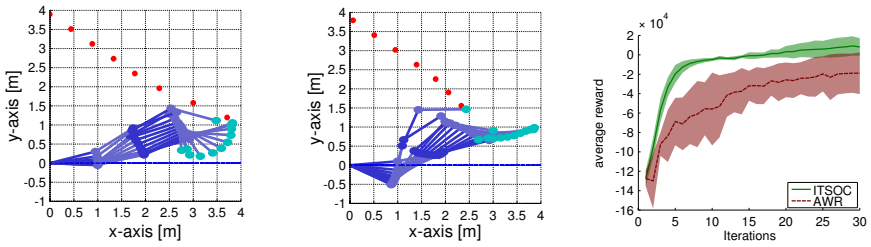
(a) $N = 5$ (b) $N = 10$ (c) $N = 20$

Figure 10: Comparison of learning constant linear models against learning time-varying linear models with a different number of collected roll-outs $N = 5, 10, 20$. For a small number of roll-outs, the constant model works better as it can use much more data-points. However, the constant model can not capture the non-linearity in the system and therefore converges to a worse solution (-10000) than the solution found by the time-varying model (-300). This evaluation clearly shows that our approach can be used for data-efficient reinforcement learning.

ball at different positions with a high velocity in the y-direction. An illustration of the resulting posture time-series for two different hitting movement is shown in Figure (11). The range of the incoming balls is approximately $1.5m$ in the x-direction and $0.5m$ in the y-direction. We can see that the robot already adapts its movement in the beginning to the predicted incoming position of the ball, but it quickly adapts its movement when the ball changes its velocity when bouncing at the floor at time step $t = 40$. The performance of the policy with an increasing number of iterations is shown in Figure (11)(c). We also compared our approach to AWR on this more complex task. The results show that AWR again can not cope with the more complex reward function and converges slowly.

6 Conclusion and Future Work

In this paper, we presented a novel information theoretic stochastic optimal control algorithm. The key idea of our approach is that we want to stay close to the generated data such that we can ensure a stable learning progress. To our knowledge, this notion of closeness to the data is missing in all other stochastic optimal control algorithms. However, we believe it is a key ingredient for save approximation of the value function



(a) (b) (c)

Figure 11: (a,b) Illustration of the robot playing tennis for two different configurations of the incoming ball. The ball has been simulated with constant velocity but bounces off the floor in a stochastic way, such that the robot needs to sensory feedback to adapt its movement. The robot can react to this perturbation and reliably hits the ball. (c) Comparison of ITSOC and AWR on the tennis task. While ITSOC learns shows a smooth learning process and converges to good solutions, the value function approximation of AWR causes jumps in the state distributions which results in a worse performance.

or the corresponding function $v_t(\boldsymbol{x})$, respectively. We show that our method can significantly outperform traditional approximate dynamic programming methods in terms of sample efficiency as well as quality of the final policy.

The information theoretic formulation provides several advantages over traditional approaches. We can get a closed-form solution for the estimated policy, at least on a finite set of samples. Furthermore, we can control the exploration of the policy in a principled manner without heuristics or fine-tuning. Moreover, we can use the roll-outs to learn simple local models which allow us to also use our algorithm for reinforcement learning. We use time-varying linear models to approximate the real system dynamics, however, the linear models are not computed at a single point of linearisation but on our current distribution of samples. Consequently, the estimated models are more robust then linear models obtained from linearisation.

The biggest disadvantage of our approach is the computation time. While methods based on linearisation can be computed in several seconds, and approximate dynamic programming methods need several minutes, our approach needs several hours to optimize the dual-function for 40 iterations. The main problem is the dimensionality of the dual function, as we have one $\boldsymbol{\theta}_t$ per time step and $\boldsymbol{\theta}_t$ can have easily up to 80

parameters. For future work, we will investigate dimensionality reduction techniques to reduce the dimensionality of the feature space. We will also investigate the combination of our locally linear policies with learned inverse dynamics controllers and learn more complex system dynamics models.

Acknowledgements

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7-ICT-2013-10) under grant agreement 610878 (3rdHand), and (FP7-ICT-2009-6) under grant agreement 270327 (ComPLACS).

References

[1] P. Abbeel and A. Ng. Apprenticeship learning via Inverse Reinforcement Learning. In *Proceedings of the 21st International Conference on Machine Learning (ICML)*, 2004.

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

[3] D. Bertsekas and J. Tsitsiklis. *Neuro Dynamic Programming*. Athena Scientific, 1998.

[4] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, MA, 1999.

[5] J. Boyan. Least-Squares Temporal Difference Learning. In *In Proceedings of the Sixteenth International Conference on Machine Learning*, pages 49–56, 1999.

[6] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[7] C. Daniel, G. Neumann, and J. Peters. Hierarchical Relative Entropy Policy Search. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2012.

[8] C. Daniel, G. Neumann, and J. Peters. Learning Sequential Motor Tasks. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[9] C. Dann, G. Neumann, and J. Peters. Policy Evaluation with Temporal Differences: A Survey and Comparison. *Journal of Machine Learning Research (JMLR)*, 2014.

[10] M. Deisenroth and C. Rasmussen. PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In *28th International Conference on Machine Learning (ICML)*, pages 465–472, 2011.

[11] D. Ernst, P. Geurts, and L. Wehenkel. Tree-Based Batch Mode Reinforcement Learning. *Journal of Machine Learning Resource*, 6:503–556, 2005.

[12] H. J. Kappen. An Introduction to Stochastic Control Theory, Path Integrals and Reinforcement Learning. In *Cooperative Behavior in Neural Systems*, volume 887 of *American Institute of Physics Conference Series*, pages 149–181, February 2007.

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

[14] M. Lagoudakis and R. Parr. Least-Squares Policy Iteration. *Journal of Machine Learning Research (JMLR)*, 4:1107–1149, December 2003.

[15] A. Lazaric and M. Ghavamzadeh. Bayesian Multi-Task Reinforcement Learning. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 2010.

[16] R. Lioutikov, A. Paraschos, G. Neumann, and J. Peters. Sample-based information-theoretic stochastic optimal control. In *Proceedings of 2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[17] The Mathworks. Matlab Optimization Toolbox User's Guide.

[18] J. Morimoto and C. Atkeson. Minimax differential dynamic programming: An application to robust bipedwalking. *Neural Information Processing Systems (NIPS)*, 2002.

[19] Gerhard Neumann and Jan Peters. Fitted q-iteration by advantage weighted regression. In Daphne Koller, Dale Schuurmans, Yoshua Bengio, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*, pages 1177–1184. Curran Associates, Inc., 2008.

[20] A. Ng and M. Jordan. PEGASUS: A Policy Search Method for large MDPs and POMDPs. In *Proceedings of the International Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 406–415, Palo Alto, CA, 2000.

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

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

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

[24] K. Rawlik, M. Toussaint, and S. Vijayakumar. Path Integral Control by Reproducing Kernel Hilbert Space Embedding. In *IJCAI*, 2013.

[25] F. Sisser. Elimination of bounds in Optimization Problems by Transforming Variables. *Mathematical Programming*, 20(1):110–121, 1981.

[26] R. Stengel. *Stochastic Optimal Control: Theory and Application*. John Wiley & Sons, Inc., 1986.

[27] E. Theodorou, J. Buchli, and S. Schaal. Reinforcement Learning of Motor Skills in High Dimensions: a Path Integral Approach. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 2010.

[28] E. Todorov. Linearly-solvable markov decision problems. In *Neural Information Processing Systems*, pages 1369–1376, 2006.

[29] E. Todorov. Efficient Computation of Optimal Actions. *Proceedings of the National Academy of Sciences*, 106(28):11478–11483, 2009.

[30] E. Todorov and Weiwei L. A generalized Iterative LQG Method for Locally-Optimal Feedback Control of Constrained Nonlinear Stochastic Systems. In *Proceedings of the 24th American Control Conference*, volume 1 of *(ACC 2005)*, 2005.

[31] M. Toussaint. Robot Trajectory Optimization using Approximate Inference. In *Proceedings of the 26th International Conference on Machine Learning*, (ICML), 2009.

[32] B. Ziebart, A. Bagnell, and A. Dey. Modeling Interaction via the Principle of Maximum Causal Entropy. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 2010.

[33] A. Zimin and G. Neu. Online learning in episodic Markovian decision processes by relative entropy policy search. In *Neural Information Processing Systems (NIPS)*, 2013.

A Derivation of the Dual-Function

We start the derivation of the dual with the Lagrangian of the optimization problem. For the sake of clarity, we will treat all time steps the same and neglect the initial state distribution constraint as well as the final KL-bound. The Lagrangian is given as

$$L = \sum_{t=1}^{T} \iint p_{\boldsymbol{xu}}^t \left(r_{\boldsymbol{xu}}^t - \eta_t \log \frac{p_{\boldsymbol{xu}}^t}{q_{\boldsymbol{xu}}^t} + \mathbb{E}[\boldsymbol{\phi}_{\boldsymbol{x}'}^T]\boldsymbol{\theta}_{t+1} - \boldsymbol{\phi}_{\boldsymbol{x}}^T \boldsymbol{\theta}_t - \lambda_t \right) d\boldsymbol{x} d\boldsymbol{u} + \eta_t \epsilon + \lambda_t,$$

where $p_{\boldsymbol{xu}}^t = p_t^\pi(\boldsymbol{x}, \boldsymbol{u})$ and we chose a similar subscript notation for q_t and $\boldsymbol{\phi}$. Differentiating the Lagrangian w.r.t. $p_{\boldsymbol{xu}}^t$

$$\frac{\partial L}{\partial p_{\boldsymbol{xu}}^t} = \qquad r_{\boldsymbol{xu}}^t - \eta_t \left(\log \frac{p_{\boldsymbol{xu}}^t}{q_{\boldsymbol{xu}}^t} + 1 \right) + \mathbb{E}[\boldsymbol{\phi}_{\boldsymbol{x}'}^T]\boldsymbol{\theta}_{t+1} - \boldsymbol{\phi}_{\boldsymbol{x}}^T \boldsymbol{\theta}_t - \lambda_t$$

and setting the result to zero yields the closed form solution for $p_{\boldsymbol{xu}}^t$,

$$p_{\boldsymbol{xu}}^t = q_{\boldsymbol{xu}}^t \exp\left(\frac{r_{\boldsymbol{xu}}^t + \mathbb{E}[\boldsymbol{\phi}_{\boldsymbol{x}'}^T]\boldsymbol{\theta}_{t+1} - \boldsymbol{\phi}_{\boldsymbol{x}}^T \boldsymbol{\theta}_t}{\eta_t} \right) \exp\left(1 - \frac{\lambda_t}{\eta_t}. \right) \tag{35}$$

Out of our normalization constraint we follow that

$$\exp\left(1 - \frac{\lambda_t}{\eta_t} \right) = \left(\iint q_{\boldsymbol{xu}}^t \exp\left(\frac{r_{\boldsymbol{xu}}^t + \mathbb{E}[\boldsymbol{\phi}_{\boldsymbol{x}'}^T]\boldsymbol{\theta}_{t+1} - \boldsymbol{\phi}_{\boldsymbol{x}}^T \boldsymbol{\theta}_t}{\eta_t} \right) d\boldsymbol{x} d\boldsymbol{u} \right)^{-1}. \tag{36}$$

Setting Equation (35) back into the Lagrangian, results after some transformations in the following equation

$$g(\eta_{1:H}, \lambda_{1:H}) = \sum_{t=1}^{T} \eta_t \epsilon - \eta_t + \lambda_t = \sum_{t=1}^{T} \eta_t \epsilon - \eta_t \log\left(\exp(1 - \lambda_t / \eta_t) \right).$$

We can now use Equation (36) to determine the dual function in terms of $\eta_{1:H}$ and $\boldsymbol{\theta}_{1:H}$,

$$g(\eta_{1:H}, \boldsymbol{\theta}_{1:H}) = \sum_{t=1}^{T} \eta_t \epsilon + \eta_t \log \left(\int q^t_{\boldsymbol{xu}} \exp \left(\frac{r^t_{\boldsymbol{xu}} + \mathbb{E}[\boldsymbol{\phi}^T_{\boldsymbol{x}'}] \boldsymbol{\theta}_{t+1} - \boldsymbol{\phi}^T_{\boldsymbol{x}} \boldsymbol{\theta}_t}{\eta_t} \right) d\boldsymbol{x} d\boldsymbol{u} \right). \tag{37}$$

Including the KL-bound of the last time step as well as the initial state constraint, the dual function is given as

$$\begin{aligned} g(\eta_{1:H}, \boldsymbol{\theta}_{1:H}; \mathcal{D}) \quad &= \sum_{t=1}^{H-1} \eta_t \log \left(\frac{1}{N} \sum_{\boldsymbol{x}, \boldsymbol{u} \in \mathcal{D}_t} \exp \left(\frac{r^t_{\boldsymbol{xu}} + \mathbb{E}[\boldsymbol{\phi}^T_{\boldsymbol{x}'}] \boldsymbol{\theta}_{t+1} - \boldsymbol{\phi}^T_{\boldsymbol{x}} \boldsymbol{\theta}_t}{\eta_t} \right) \right) \\ &\quad + \sum_{t=1}^{T} \eta_t \epsilon + \eta_H \log \left(\frac{1}{N} \sum_{\boldsymbol{x} \in \mathcal{D}_H} \exp \left(\frac{r^T_{\boldsymbol{x}} - \boldsymbol{\phi}^T_{\boldsymbol{x}} \boldsymbol{\theta}_H}{\eta_H} \right) \right) + \hat{\boldsymbol{\phi}}_1^T \boldsymbol{\theta}_1, \end{aligned} \tag{38}$$

where we replaced the distribution q_t by $1/N$ as we now use samples $\mathcal{D}$ which have been generated from q_t. The set $\mathcal{D}_t$ denotes all samples for time step t.