Learning Concurrent Motor Skills in Versatile Solution Spaces

Christian Daniel[1] and Gerhard Neumann[1] and Jan Peters[1,2]

***Abstract*— Future robots need to autonomously acquire motor skills in order to reduce their reliance on human programming. Many motor skill learning methods concentrate on learning a single solution for a given task. However, discarding information about additional solutions during learning unnecessarily limits autonomy. Such favoring of single solutions often requires re-learning of motor skills when the task, the environment or the robot's body changes in a way that renders the learned solution infeasible. Future robots need to be able to adapt to such changes and, ideally, have a large repertoire of movements to cope with such problems. In contrast to current methods, our approach simultaneously learns multiple distinct solutions for the same task, such that a partial degeneration of this solution space does not prevent the successful completion of the task. In this paper, we present a complete framework that is capable of learning different solution strategies for a real robot Tetherball task.**

I. Introduction

Robot learning of dynamic motor tasks has been an active field of research in recent years and many impressive applications have been demonstrated [1], [2], [3], [4], [5]. A large number of skills have been learned by reinforcement learning (RL), including the game 'Ball-in-the-Cup' [1], the Peg-in-Hole task [2], robot soccer [6], walking [3] and jumping [4]. In RL, the robot explores actions to learn motor skills and gets rewards as evaluative feedback on the quality of the performed movement.

The most prominent motor skill learning algorithms are Policy Search (PS) methods [7], [8], [4], [9], including Policy Gradient methods [2], [10], Expectation-Maximization (EM)-like approaches such as [1], [11] and policy improvements based on path integrals [4]. Policy search is a sub-field of reinforcement learning which directly tries to optimize the parameters of the policy. PS has been successful in learning single solutions for many different tasks. However, most interesting motor tasks can be accomplished in several distinct ways. For example, there are many different possibilities to return a ball in table tennis, such as forehand or backhand swings, volleys, loops, or drops. Similarly, multi-legged animals, e.g. horses, can have several gaits for the same walking speed, such as tolt, amble and trot. Having multiple solutions to accomplish a motor task may considerably improve the robustness of robot applications, rendering their movement-repertoire more human-like as well as increasing the autonomy of the robot.

[1]Technische Universitaet Darmstadt, Hochschulstrasse 10, 64289 Darmstadt Germany, FG Informatik, FB Institute for Intelligent Autonomous Systems. {daniel, neumann, peters}@ias.tu-darmstadt.de

[2]Max-Planck-Institut fuer Intelligente Systeme

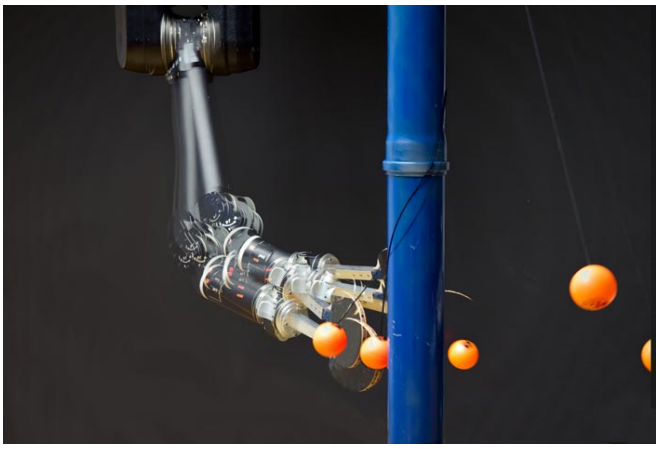

Fig. 1: The Robot-Tetherball task. We adapt the children's game Tetherball for a robot application. A ball is fixed to a string which is fixed to the ceiling. The robot has to hit the ball such that it winds around the pole. This requires two dynamic movements, one for pushing the ball out of his resting position and a second hitting movement.

For example, in uncontrolled environments such as households, hospitals or nursing homes, we cannot expect the environment to be static. We will frequently encounter situations where the task itself changes, e.g., we might be presented with a different racket in a tennis playing task. The environment or our robot itself might even change, e.g., if the robot is carrying a tray with different objects. In any of the above cases, representing just one motor skill for a given task would drastically limit the autonomy of the robot as it would require re-learning of the motor skill. However, if the robot is aware of multiple solutions and has multiple skills, it can employ alternative policies in situations where the standard policy may be inadequate. Many current methods can only represent a single solution [1], [4] since they model the policy as a uni-modal distribution such as a single Gaussian. Additionally, as Neumann [12] has shown, representing the policy by a single Gaussian can affect the performance of policy search methods due to averaging over several modes.

In this paper, we propose to concurrently model and learn multiple skills of versatile solution spaces for motor tasks by employing the Relative Entropy Policy Search (REPS) approach [8], [13]. We present a unifying framework for learning multiple solutions for one task, and show how to choose from these solutions in a given situation. Our approach learns on two levels of a hierarchy. The high-level gating policy selects the specific solution, which we will also call *option* [14], and the action-policy subsequently defines

the action or movement plan which is executed by the robot.

Using hierarchical policy representations, also called hierarchical RL, is a promising idea and has already been shown to accelerate learning in many situations [15], [16], [14]. However, most of these algorithms [15], [16] are formulated only for discrete domains and are, therefore, not task-appropriate for robotics.

Due to the hierarchical policy representation, we are able to learn several solutions at once. Our approach extends the Relative Entropy Policy Search [8] algorithm to the hierarchical policy case. In this paper, we apply our algorithm in combination with the commonly used dynamic movement primitives (DMP) [17]. We simultaneously learn a gating network, which selects between primitives given the current context, and the policies of the primitives, which specify the robot's actual actions.

We apply our method to a robot version of the children's game Tetherball. In robot Tetherball, a ball is hung from the ceiling by an elastic string and the robot has to hit the ball such that it winds around a pole. We are able to concurrently learn multiple solutions for this task, i.e., hitting the ball to the right side or to the left side of the pole.

II. Learning versatile motor skills

We start our discussion with the formal problem statement, and continue by briefly describing the learning setup. After reviewing information-theoretic policy search approaches we show an extension of one of this approaches, called Relative Entropy Policy Search (REPS) [8] to be applicable to hierarchical policy representations.

For our formal problem statement, we will use the standard Markov decision process (MDP) setup [18]. An agent in state $\boldsymbol{s} \in \mathcal{S}$ takes action $\boldsymbol{a} \in \mathcal{A}$ according to a policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ and receives a reward $r(\boldsymbol{s}, \boldsymbol{a})$. After executing action $\boldsymbol{a}$ in state $\boldsymbol{s}$, the agent ends up in a next state $\boldsymbol{s}'$ according to the transition probability $\mathcal{P}^{\boldsymbol{a}}_{\boldsymbol{s}\boldsymbol{s}'}$.

In addition to the standard MDP formulation, we use options $o \in \mathcal{O}$ [14] to represent our hierarchical policy by

$$\pi(\boldsymbol{a}|\boldsymbol{s}) = \sum_o \pi(\boldsymbol{a}|\boldsymbol{s}, o)\pi(o|\boldsymbol{s}).$$

Hence, our action selection policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ consists of a gating network $\pi(o|\boldsymbol{s})$ to select a specific option and the option policies $\pi(\boldsymbol{a}|\boldsymbol{s}, o)$ for selecting the action.

A. Episodic Reinforcement Learning with Motor Primitives

We concentrate on the episodic learning case for motor skills as this applies for many single-stroke motor skills such as hitting movements [1]. We represent each option by a *dynamic movement primitive* (DMP) [17]. A single DMP represents a movement plan for the whole episode. The gating network $\pi(o|\boldsymbol{s})$ only initiates sub-policies at the beginning of each episode. Subsequently, the policy of the movement primitive takes over and generates the movement. The action $\boldsymbol{a}$ denotes the parameters of the DMP in this case. As each option defines a different distribution $\pi(\boldsymbol{a}|\boldsymbol{s}, o)$ over the parameter-space of the DMPs, it also represents a distinct family of solutions which are applicable to the task.

B. Information-Theoretic Policy Search

Our policy search approach extends an existing policy search method, the Relative Entropy Policy Search (REPS) algorithm, in such a way that it can be applied to hierarchical policy representations [13]. REPS is an information theoretic policy search approach which is based on the insight that the information loss between two policy steps should be bounded [19], [20], [8]. If the policy update step is too greedy w.r.t to the reward function, the update might jump into a local minimum or even result in policies which are potentially dangerous to the real robot. This potential damage to the policy is avoided by bounding the Kullback-Leibler divergence, also called relative entropy, between the old state-action distribution $q(\boldsymbol{s}, \boldsymbol{a})$ and the state-action distribution $p(\boldsymbol{s}, \boldsymbol{a})$ resulting from the new policy.

In the episodic case of REPS, we only consider one action $\boldsymbol{a}$ per episode and the states are only used to describe the initial state of the episode. As the action defines the parameters of the DMP, the action already determines the whole movement plan for the episode. The states $\boldsymbol{s}$ are given by the initial state of the episode and the reward $\mathcal{R}_{\boldsymbol{s}\boldsymbol{a}}$ is given by the reward of the whole episode.

REPS maximizes the episodic reward while bounding the Kullback-Leibler divergence between the old model distribution $q(\boldsymbol{s}, \boldsymbol{a})$ and the new state action distribution $p(\boldsymbol{s}, \boldsymbol{a})$. In addition, as the initial state distribution $\mu_0(\boldsymbol{s})$ is given, the state distribution $\mu_\pi(\boldsymbol{s}) = \sum_{\boldsymbol{a}} p(\boldsymbol{s}, \boldsymbol{a})$ of the model has to match $\mu_0(\boldsymbol{s})$ in order to avoid inconsistencies. We use a relaxed version of this constraint, where we require that the feature averages of $p(\boldsymbol{s}, \boldsymbol{a})$ match the observed feature averages $\hat{\boldsymbol{\phi}}_0(\boldsymbol{s})$, i.e.,

$$\sum_{\boldsymbol{s},\boldsymbol{a}} p(\boldsymbol{s}, \boldsymbol{a})\boldsymbol{\phi}(\boldsymbol{s}) = \hat{\boldsymbol{\phi}}_0(\boldsymbol{s}). \tag{1}$$

The optimization problem solved by REPS is defined as

$$\begin{aligned} \max_p J(p) &= \sum_{\boldsymbol{s},\boldsymbol{a}} p(\boldsymbol{s}, \boldsymbol{a}) R_{\boldsymbol{s}\boldsymbol{a}}, \\ \text{s.t.} \quad \epsilon &\geq \sum_{\boldsymbol{s},\boldsymbol{a}} p(\boldsymbol{s}, \boldsymbol{a}) \log \frac{p(\boldsymbol{s}, \boldsymbol{a})}{q(\boldsymbol{s}, \boldsymbol{a})}, \end{aligned} \tag{2}$$

where we omitted the normalization constraint, i.e. $\sum_{\boldsymbol{s},\boldsymbol{a}} p(\boldsymbol{s}, \boldsymbol{a}) = 1$, and the initial state constraint given in Equation (1). This constrained optimization problem can be solved by maximizing the corresponding Lagrangian $\mathcal{L}$. The relative entropy formulation allows for a closed form solution of $p(\boldsymbol{s}, \boldsymbol{a})$, which can be used to determine the dual-function g of the original optimization problem. As the dual-function g is convex, it can be optimized efficiently.

C. Hierarchical Policy Search

We reformulate the problem of estimating a hierarchical policy as latent variable estimation problem. Thus, we treat the options o as unobserved variables. As in REPS, we bound the Kullback-Leibler divergence between $q(\boldsymbol{s}, \boldsymbol{a})$ and $p(\boldsymbol{s}, \boldsymbol{a}) = \sum_o \pi(\boldsymbol{a}|\boldsymbol{s}, o)\pi(o|\boldsymbol{s})\mu^\pi(\boldsymbol{s})$. However, using the

marginal $p(\boldsymbol{s}, \boldsymbol{a})$ in the bound no longer allows for a closed form solution of $p(\boldsymbol{s}, \boldsymbol{a}, o)$ any more. Thus, we need to use an iterative update rule, which is strongly inspired by the Expectation-Maximization algorithm [21]. This update results from the bound

$$\sum_{\boldsymbol{s},\boldsymbol{a}} \sum_{o} p(\boldsymbol{s}, \boldsymbol{a}, o) \log \frac{p(\boldsymbol{s}, \boldsymbol{a}, o)}{q(\boldsymbol{s}, \boldsymbol{a}) \tilde{p}(o|\boldsymbol{s}, \boldsymbol{a})} \leq \epsilon, \tag{3}$$

where $\tilde{p}$ is a proposal distribution, often called *responsibility* in EM-based algorithms. In the E-step of our algorithm, the responsibilities are determined by

$$\tilde{p}(o|\boldsymbol{s}, \boldsymbol{a}) = \frac{p(\boldsymbol{s}, \boldsymbol{a}, o)}{\sum_o p(\boldsymbol{s}, \boldsymbol{a}, o)}. \tag{4}$$

In the M-step, the responsibilities are kept fixed and we directly optimize for $p(\boldsymbol{s}, \boldsymbol{a}, o)$. Both iterations, the E- and the M-step, can be proven to increase a lower bound of the original optimization problem [13]. Thus, REPS with the additional latent variable estimation always converges to a local optimum of the optimization problem defined in Equation (2).

Since we are interested in versatile solutions, we want to avoid that several options are concentrating on the same solution. In order to do so, we additionally bound the expected entropy of the responsibilities $\pi(o|\boldsymbol{s}, \boldsymbol{a})$, i.e.,

$$-\sum_{\boldsymbol{s},\boldsymbol{a}} p(\boldsymbol{s}, \boldsymbol{a}) \sum_{o} \pi(o|\boldsymbol{s}, \boldsymbol{a}) \log \pi(o|\boldsymbol{s}, \boldsymbol{a}) \leq \kappa. \tag{5}$$

Low entropies of $\pi(o|\boldsymbol{s}, \boldsymbol{a})$ ensures that our options do not overlap and instead represent individual and clearly distinct solutions. We can again replace $\pi(o|\boldsymbol{s}, \boldsymbol{a})$ with the responsibilities $\tilde{p}(o|\boldsymbol{s}, \boldsymbol{a})$ in the term $\log \pi(o|\boldsymbol{s}, \boldsymbol{a})$.

By combining the constraints given in Equations (3) and (5) with the original optimization problem of REPS, given in Equation (2), we can determine a hierarchical version of REPS. By derivating the Lagrangian $\mathcal{L}$ w.r.t. to $p(\boldsymbol{s}, \boldsymbol{a}, o)$, we can determine a closed form solution for p, i.e.,

$$p(\boldsymbol{s}, \boldsymbol{a}, o) \propto q(\boldsymbol{s}, \boldsymbol{a}) \tilde{p}(o|\boldsymbol{s}, \boldsymbol{a})^{1+\xi/\eta} \exp\left(\frac{R_{\boldsymbol{sa}} - V(\boldsymbol{s})}{\eta}\right). \tag{6}$$

The parameter η denotes the Lagrange multiplier for the relative entropy bound from Equation (3) and ξ the Lagrange multiplier for the entropy bound of the responsibilities given in Equation (5). The function $V(\boldsymbol{s}) = \boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{s})$ can be seen as a value function like term [8]. The parameters $\boldsymbol{\theta}$ are again Lagrange multipliers of the initial distribution constraint which is given in Equation (1). All Lagrange parameters can be acquired by minimizing the dual function $g(\eta, \xi, \boldsymbol{\theta})$ of the optimization problem. Note that our approach reduces to REPS if we only use one option.

In the original formulation of REPS [8], we are not required to know the distribution $q(\boldsymbol{s}, \boldsymbol{a})$ in its parametric form. Instead, it is sufficient to have access to samples of $q(\boldsymbol{s}, \boldsymbol{a})$. Hence, $q(\boldsymbol{s}, \boldsymbol{a})$ does not necessarily need to be the old-model distribution.

Input: Information loss tolerance ϵ, Entropy tolerance κ, Number of options n
Initialize π using n Gaussians with random mean
for $k = 1 \ldots L$
Set sample policy: $q(\boldsymbol{a}\|\boldsymbol{s}) = \sum_o \pi_{\text{old}}(o\|\boldsymbol{s}) \pi_{\text{old}}(\boldsymbol{a}\|\boldsymbol{s}, o)$
Sample: collect new samples from the sample policy and add to dataset $\{\boldsymbol{s}_j \sim p(\boldsymbol{s}_0), \boldsymbol{a}_j \sim q(\boldsymbol{a}\|\boldsymbol{s}_j), R_j\} j \in \{1, .., N\}$ Calculate **importance weights** $v_i^k = \frac{q_k(\boldsymbol{s}_i, \boldsymbol{a}_i)}{\sum_{h=k-H}^{k} q_h(\boldsymbol{s}_i, \boldsymbol{a}_i)}$. for all i
Proposal distribution: $\tilde{p}(o\|\boldsymbol{s}_i, \boldsymbol{a}_i) = p_{\text{old}}(o\|\boldsymbol{s}_i, \boldsymbol{a}_i)$ for all i
Minimize the dual function $[\boldsymbol{\theta}, \eta, \xi] = \arg\min_{[\boldsymbol{\theta}, \eta, \xi]} g(\boldsymbol{\theta}, \eta, \xi)$
Policy update: *Calculate model distribution* $p(\boldsymbol{s}_i, \boldsymbol{a}_i, o) \propto v_i^k \ \tilde{p}(o\|\boldsymbol{s}_i, \boldsymbol{a}_i)^{1+\xi/\eta} \exp\left(\frac{R_i - \boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{s}_i)}{\eta}\right)$ *Estimate distributions* $\pi(o\|\boldsymbol{s})$ and $\pi(\boldsymbol{a}\|\boldsymbol{s}, o)$ by weighted ML estimates
Output: Policy $\pi(\boldsymbol{a}, o\|\boldsymbol{s})$

TABLE I: Hierarchical REPS. The algorithm collects new samples in each policy iteration. Due to the use of importance sampling, we can re-use old samples. The importance weights v_i^k are used for the calculation of the dual-function g as well as the model distribution p. The parameters η, ξ and $\boldsymbol{\theta}$ are determined by minimizing the dual-function g.

As a consequence of using samples, we can evaluate our model distribution $p(\boldsymbol{s}_i, \boldsymbol{a}_i, o)$ only at our sampled state-action pairs $\boldsymbol{s}_i$ and $\boldsymbol{a}_i$. Hence, we still need to fit a parametric model to $\pi(\boldsymbol{a}|\boldsymbol{s}, o)\pi(o|\boldsymbol{s})\mu^\pi(s)$ in order to draw new samples from this distribution. We estimate the parametric model by minimizing the Kullback-Leibler divergence $\text{KL}\left(p(\boldsymbol{s}, \boldsymbol{a}, o) || \pi(\boldsymbol{a}|\boldsymbol{s}, o)\pi(o|\boldsymbol{s})\mu^\pi(\boldsymbol{s})\right)$, which is given by

$$\begin{aligned} &\sum_{\boldsymbol{s},\boldsymbol{a}} p(\boldsymbol{s}, \boldsymbol{a}, o) \log \pi(\boldsymbol{a}|\boldsymbol{s}, o)\pi(o|\boldsymbol{s})\mu^\pi(\boldsymbol{s}) + \text{const} \\ = &\sum_{\boldsymbol{s}_i,\boldsymbol{a}_i \sim q(\boldsymbol{s}_i,\boldsymbol{a}_i)} w(\boldsymbol{s}_i, \boldsymbol{a}_i, o) \log \pi(\boldsymbol{a}_i|\boldsymbol{s}_i, o)\pi(o|\boldsymbol{s}_i)\mu^\pi(\boldsymbol{s}_i), \end{aligned} \tag{7}$$

where

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

This minimization defines a weighted maximum-likelihood estimation problem. For simplicity, we use a Gaussian gating network for $\pi(o|\boldsymbol{s})$ and a linear Gaussian model as action selection policy $\pi(\boldsymbol{a}|\boldsymbol{s}, o)$.

D. Sample Re-use by Importance Sampling

As we usually have knowledge of the sampling distribution $q_k(\boldsymbol{s}, \boldsymbol{a})$ at each policy iteration k, we can extend the hierarchical REPS framework by re-using samples from previous iterations using importance sampling [22]. At each iteration k of the policy search, we assume that our samples have been generated by a mixture of the last H sampling distributions

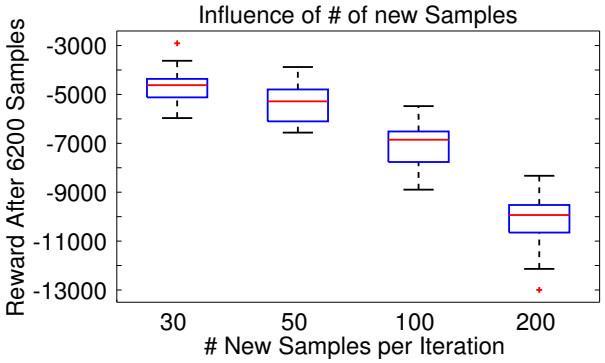

Fig. 2: Evaluation of the number of new samples per iteration. Collecting more than 30 samples per iteration requires a higher total number of samples to achieve similar reward. Averaged over 10 cross validations.

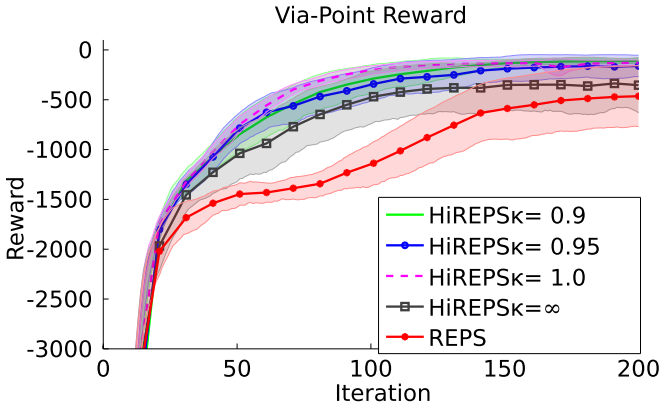

Fig. 3: Average rewards of our approach with different κ values and the standard REPS. Our approach learns faster and converges to a better solution. Averaged over 20 cross validations.

$q_h(\boldsymbol{s}, \boldsymbol{a}), h = k - H \ldots k$. Thus, the importance weights for the sample $\boldsymbol{s}_i$ and $\boldsymbol{a}_i$ is given by

$$v_i^k = \frac{q_k(\boldsymbol{s}_i, \boldsymbol{a}_i)}{\sum_{h=k-H}^{k} q_h(\boldsymbol{s}_i, \boldsymbol{a}_i)}.$$

Such importance weights are used in the calculation of the dual function (see appendix) as well as well as for the weighted maximum likelihood estimation, as defined in Equation (7). The algorithmic form of our approach is summarized in Table I.

We typically start the algorithm with too many options and delete options if an option has a very low prior $p(o) = \sum_{\boldsymbol{s},\boldsymbol{a}} p(\boldsymbol{s}, \boldsymbol{a}, o)$. To prevent options from prematurely getting deleted in the beginning of the learning process, we ensure that each option gets a minimum amount of samples at each policy search iteration and always keep at least four options.

III. Experiments

We evaluate the proposed method within the episodic motor skill learning setup with movement primitives. We will first describe the employed movement primitives. Subsequently, we demonstrate the basic characteristics of our

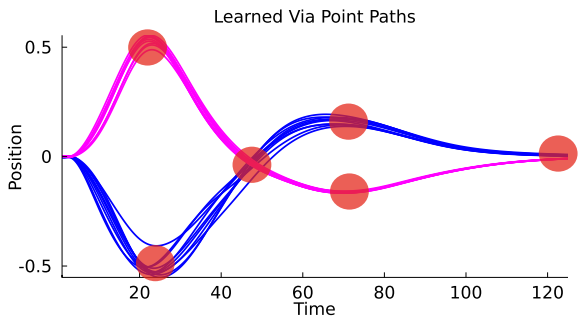

Fig. 5: Trajectories found by two distinct options (magenta and blue colors) for the via-point task. The red circles show the positions of the via points that have to be reached during the motion. Our approach simultaneously finds both solutions.

algorithm on a toy task wherein the agent has to learn a movement through several via-points.

In order to illustrate the algorithm on a more complex, real-robot task we present experiments of a robot learning Tetherball. Here, we first evaluated our algorithm on a real-physics simulation and, subsequently, learned the task on a real Barrett WAM arm as can be seen in Figure 4.

A. Dynamic Motor Primitives

To describe the motion of the robot arm, we use a recent adaption of dynamic motor primitives (DMPs) to hitting movements as presented in [23]. A DMP uses a second order linear dynamical system which is modulated by a learnable non-linear function $f(z; \mathbf{w})$, where z denotes a phase variable of the movement. The function $f(z; \mathbf{w}) = \boldsymbol{\Phi}(z)^T \mathbf{w}$ is non-linear in the phase variable z but linear in its parameters $\mathbf{w}$. The parameters $\mathbf{w}$ define the shape of the movement and can be learned efficiently from a demonstrator's trajectory in a imitation learning setup [17]. For each joint a different, learnable function f and, thus, different parameters $\mathbf{w}$ are used. In addition to the shape parameters $\mathbf{w}$, we can adapt meta-parameters of the DMP such as the desired goal position $\mathbf{y}_g$ and the corresponding desired velocity $\dot{\mathbf{y}}_g$ at the end of the movement. For a more detailed description of the used DMP approach we refer to [23].

In our simple via-point task, we learn the shape parameters $\mathbf{w}$ of multiple DMPs while keeping the meta-parameters fixed. For the robotic Tetherball experiments, we acquire $\mathbf{w}$ from one teacher's demonstration trajectories and adapt the final position and velocities of the hitting movement. The demonstration usually provides an adequate initialization for the movement. However, simply replaying the demonstration typically does not solve the motor skill task as the robot is unable to exactly reproduce the teacher's behavior, also known as correspondence problem [24], [25], or because the teacher himself could not optimally solve the task. Thus, based on the demonstration, we need to learn an adapted movement that can solve the task.

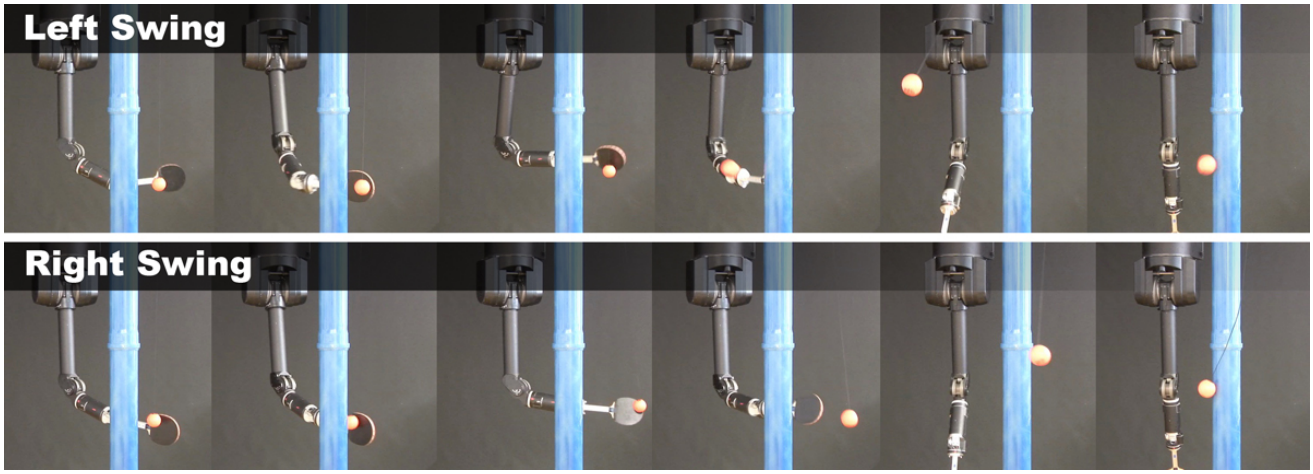

Fig. 4: Time series of a successful swing of the robot. The robot first has to swing the ball to the pole and can, subsequently, when the ball has swung backwards, arc the ball around the pole.

B. Illustrative Toy Task

To demonstrate the characteristics of our hierarchical learning framework, we choose a multi-modal via-point task [26]. We use a one dimensional linear dynamic system, the state $\boldsymbol{s}$ is given by both the position x and velocity $\dot{x}$ of the agent. The control variable is given by the acceleration $\ddot{x}$. The task is given by reaching a set of via-points, marked as red circles in Figure 5. As we want to model a motor skill task with multiple solutions, we use multiple via-points for some of the time points.

The reward function for this task is defined as the negative distance between the agent's position at the specified time-points and the closest via-point. In addition, we punish the summed squared accelerations to prefer energy efficient solutions[1]. Thus, this task has two optimal solutions, one passing the via-points $v_1 = 0.5$ and $v_3 = -0.2$ and the second solution with $v_1 = -0.5$ and $v_3 = 0.2$. The setup is also visualized in Figure 5

The agent starts each episode in the initial position $x_0 = 0$. Subsequently, the hierarchical policy chooses which DMP to execute and also chooses the exact parametrization of the DMP. For this task, we assume that the goal state at $t = 1.25$s is given, we use ten basis functions for the DMPs and learn the corresponding parameters $\mathbf{w}$.

We evaluate our approach with different bounding parameters κ for the responsibilities. We always use a value for κ which is proportional to the current entropy of the responsibilities. We also compare our approach to the standard unimodal REPS algorithm. At each iteration we use 200 samples. To illustrate the effect of different κ values we did not use importance sampling in this experiment. The resulting learning curves can be seen in Figure 3. As we can see, $\kappa = 1.0$ results in the highest rewards as well as fast learning in this setup. Using no bound for κ results in worse results as most options will concentrate on the same solution space, which slows down learning. The standard REPS approach needs more samples to find good solutions as it often gets stuck between the two modes in the beginning of the learning process. Two distinct options of the same learning trial are depicted in Figure 5. We can see that our approach is indeed able to discover both modes and, thus, be fail-safe in changing environments.

In a second experiment, we evaluate the influence of importance sampling. We evaluate our algorithm with $N = [30, 50, 100, 200]$ samples per iteration and re-evaluate previously collected samples with importance sampling, where we always re-use the last 200 samples. Each learning trial is performed until the algorithm has collected 6200 samples. The results are shown in Figure 2 and show the benefits of importance sampling. Just collecting 30 samples per iteration and using importance sampling improved the average reward from $-10,000$ to -5000. Without importance sampling and with $N = 30$ no good solution could be found.

C. Robot Tetherball

Tetherball is a common two player children's game. Each player tries to hit the ball such that it winds around the pole in one direction while the second player is trying to wind the ball in the opposing direction. Inspired by this game, we introduce the robot Tetherball task. As we are currently interested in learning to hit the ball, rather than learning a competitive two-player strategy, we modify the rules of the game to be suitable for a single player game. Instead of having an opponent, the robot has to wind the ball around the pole once and receives reward proportional to the speed of the ball winding around the pole. We mount a table-tennis paddle to the end-effector of the robot arm. The real-robot setup is depicted in Figure 1 and two successful hitting movements of the real robot are shown in Figure 4.

In the original Tetherball game, the ball is hung from the upper end of the pole and the players start the game by first moving the ball away from the pole and then hitting it towards their opponent. Displacing the ball from its resting

[1]For the distance to the via-points we used a factor of 10^4 and for the squared accelerations a factor of 10^{-3} in the reward function.

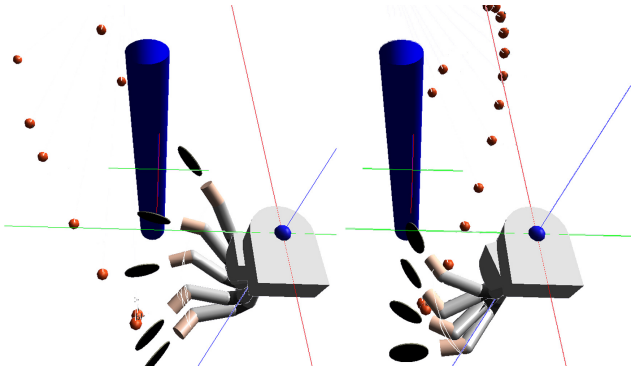

Fig. 6: Time series overlay of two swings in simulation, one left swing and one right swing. The two solutions are different options of the same policy. HiREPS can also keep more options representing additional solutions to a task.

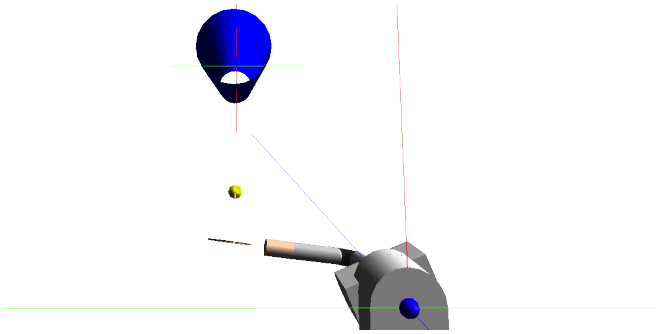

Fig. 7: Top view of the setup of robot tetherball. The ball is hung from the ceiling in front of the pole.

pose is necessary to achieve a circular motion of the ball, enabling the ball to actually wrap around the pole. In order to allow the robot to serve the ball using a single hand, we attach the string to the ceiling, between the robot and the pole, rather than to the pole directly. Hence, the robot can hit the ball once to displace it from its resting pose and, subsequently, hit it again to arc it around the pole.

Thus, we decompose our movement into a swing-in motion and a hitting motion. For both motions we extract the shape parameters $\mathbf{w}$ by kinesthetic teach-in [27]. For learning both motions in our episodic setup we represent the two motions by a single set of parameters and jointly learn the parameters for the two DMPs. For both movements, we learn the final positions and velocities of all seven joints. Additionally, we learn the waiting time between both movements. This task setup results in a 29-dimensional action space for our robot Tetherball task.

The reward is determined by the speed of the ball when the ball winds around the pole. We define winding around the pole as the ball passing the pole on the opposite side from the initial position.

In order to reliably test our algorithm, without harming the robot, we implemented an accurate physics simulation of the setup, as seen in Figure 7. We run our algorithm with 50 samples per iteration and always keep the last 400 samples.

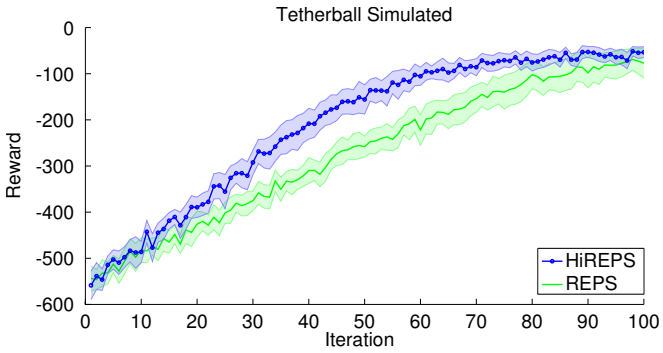

Fig. 8: The average learning curve of our algorithm on the simulated Robot Tetherball task. HiREPS converges faster while additionally keeping a minimum of five solutions for the presented task and can, thus, wind the ball around the left side of the pole as well as the right side of the pole. REPS will converge to similarly good results as HiREPS after around 100 iterations but does not keep information about more than one solution.

We initialize our algorithm with 30 options and stop deleting options if only 5 options are left. The resulting learning curve in the simulation can be seen in Figure 8.

The learning curve of our approach can be seen in Figure 8. After 200 iterations the robot has learned to wind the ball around the pole in 5/5 trials. In all trials, we were able to observe options for the left and for the right mode. The resulting movements are shown in Figure 6 and illustrate that the resulting movement of the two solutions are easily differentiated. Subsequently, we tested our algorithm on the real robot. We initialize our algorithm with 15 options and sample 15 trajectories per iteration. The learning curve is shown in Figure 9. The noisy reward signal is mostly due to the vision system (Microsoft Kinect) and partly also due to real world effects such as friction. Two resulting movements of the robot are shown in Figures 4 and 6.

IV. Conclusion and Future Work

Learning versatile motor skills is an important step towards autonomous robotic agents. In this paper, we presented the first real robot results of our hierarchical policy search method, which can learn concurrent solutions to a motor skill task. Many motor skill tasks can be realized by distinct solutions. Learning such distinct solutions is likely to simplify the generalization of motor skills, increase the adaptability of robots and render the movements more human-like.

In the presented hierarchical relative entropy policy search framework, we formulate the problem of learning a hierarchical policy as latent variable estimation problem. In this paper, we applied this approach to learn a hierarchical DMP policy, where the option selection policy chooses a single DMP and the action selection policy, subsequently, determines the exact parameters of the DMP. We evaluated our algorithm on a simulated robot tetherball setup and, subsequently, learned to play robot tetherball on a real robot. In the future, we also

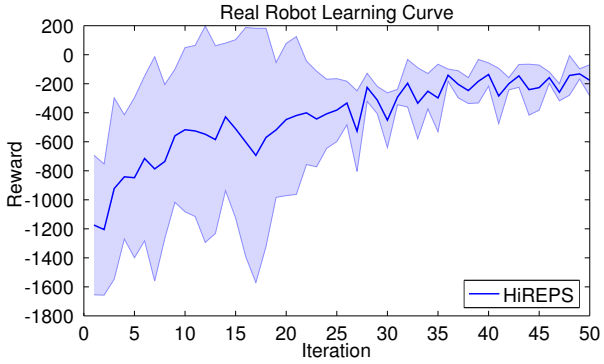

Fig. 9: Average rewards of our approach on the real robot setup. Mean and standard deviation of three cross validations. In all of the three trials, after 50 iterations the robot has found solutions to wind the ball around the pole on either side.

plan to use our approach to learn to select multiple DMPs in a sequence.

We also presented an extension of our policy search approach, that reuses old samples by importance sampling. Due to the increased sample efficiency of the algorithm, the approach is more suitable for real robots. We also plan to extend our task to the case of a two-player game as well as using our approach on a two-player table tennis setup.

APPENDIX

A. The Dual Function

The dual function for the our algorithm, with importance sampling, is given by

$$g(\boldsymbol{\theta}, \eta, \xi) = \epsilon\eta + \kappa\xi + \boldsymbol{\theta}^T \hat{\boldsymbol{\phi}}_0 + \eta \log \left(\frac{\sum_i v_i^k Z_i}{\sum_i v_i^k} \right),$$

$$Z_i = \sum_o \tilde{p}(o|\boldsymbol{s}_i, \boldsymbol{a}_i)^{1+\xi/\eta} \exp\left(\frac{R_i - \boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{s}_i)}{\eta} \right).$$

As the dual-function consists of the log of summed exponential terms it is convex in its parameters. The factors v_i^k denote the importance weights.

ACKNOWLEDGMENT

The authors want to thank for the support of the European Union projects # FP7-ICT-270327 (Complacs) and # 248 273 (GeRT).

REFERENCES

[1] J. Kober and J. Peters, "Policy Search for Motor Primitives in Robotics," *Machine Learning*, pp. 1–33, 2010.

[2] V. Gullapalli, F. J, and H. Benbrahim, "Acquiring Robot Skills via Reinforcement Learning," in *IEEE Control Systems Special Issue on Robotics: Capturing Natural Motion*, 1994.

[3] T. Matsubara, J. Morimoto, J. Nakanishi, M.-a. Sato, and K. Doya, "Learning Sensory Feedback to CPG with Policy Gradient for Biped Locomotion," in *Proceedings of the 2005 IEEEE International Conference on Robotics and Automation ICRA, Barcelona*, April 2005, pp. 4175–4180.

[4] 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, pp. 2397–2403.

[5] M. Rosenstein, "Robot Weightlifting by Direct Policy Search," *International Joint Conference on Artificial Intelligence (ICJAI)*, 2001.

[6] M. Riedmiller, T. Gabel, R. Hafner, and S. Lange, "Reinforcement Learning for Robot Soccer," *Auton. Robots*, vol. 27, pp. 55–73, July 2009. [Online]. Available: http://portal.acm.org/citation.cfm?id=1569248.1569254

[7] J. Peters and S. Schaal, "Reinforcement Learning of Motor Skills with Policy Gradients," *Neural Networks*, no. 4, pp. 682–97, 2008.

[8] J. Peters, K. Mülling, and Y. Altun, "Relative Entropy Policy Search," in *Proceedings of the 24th National Conference on Artificial Intelligence (AAAI2010)*. AAAI Press, 2010.

[9] J. A. Bagnell and J. C. Schneider, "Covariant Policy Search," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[10] M. P. Deisenroth, C. E. Rasmussen, and D. Fox, "Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning," in *Proceedings of R:SS*, 2011.

[11] N. Vlassis and M. Toussaint, "Model-Free Reinforcement Learning as Mixture Learning," in *International Conference on Machine Learning (ICML 2009)*, 2009, p. 136.

[12] G. Neumann, "Variational Inference for Policy Search in Changing Situations," in *Proceedings of the 28th International Conference on Machine Learning*, ser. (ICML 2011). New York, NY, USA: ACM, June 2011, pp. 817–824.

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

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

[15] M. Ghavamzadeh and S. Mahadevan, "Hierarchical Policy Gradient Algorithms," in *International Conference for Machine Learning (ICML)*. AAAI Press, 2003, pp. 226–233.

[16] T. G. Dietterich, "State abstraction in maxq hierarchical reinforcement learning," in *NIPS*, 1999, pp. 994–1000.

[17] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert, "Learning Movement Primitives," in *International Symposium on Robotics Research*, ser. (ISRR 2003), 2003, pp. 561–572.

[18] R. S. Sutton and A. G. Barto, *Reinforcement Learning*. Boston, MA: MIT Press, 1998.

[19] J. A. Bagnell and J. G. Schneider, "Autonomous Helicopter Control using Reinforcement Learning Policy Search Methods," in *Proceedings of the International Conference for Robotics and Automation (ICRA)*, 2001, pp. 1615–1620.

[20] S. Still and D. Precup, "An Information-theoretic Approach to Curiosity-driven Reinforcement Learning," *International Conference on Humanoid Robotics*, 2011.

[21] R. Neal and G. E. Hinton, "A View Of The EM Algorithm That Justifies Incremental, Sparse, And Other Variants," in *Learning in Graphical Models*. Kluwer Academic Publishers, 1998, pp. 355–368.

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

[23] J. Kober, K. Mülling, O. Kroemer, C. H. Lampert, B. Schölkopf, and J. Peters, "Movement Templates for Learning of Hitting and Batting," in *International Conference on Robotics and Automation (ICRA)*, 2010, pp. 853–858.

[24] C. L. Nehaniv and K. Dautenhahn, "The Correspondence Problem," in *Imitation in animals and artifacts*, K. Dautenhahn and C. L. Nehaniv, Eds. Cambridge, MA, USA: MIT Press, 2002, pp. 41–61. [Online]. Available: http://dl.acm.org/citation.cfm?id=762896.762899

[25] S. Schaal, "Is Imitation Learning the Route to Humanoid Robots?" *Trends in Cognitive Sciences*, vol. 3, no. 6, pp. 233–242, 1999. [Online]. Available: http://courses.media.mit.edu/2003spring/mas963/schaal-TICS1999.pdf

[26] J. Peters and S. Schaal, "Policy Gradient methods for Robotics," in *Proceedings of the IEEE International Conference on Intelligent Robotics Systems (IROS)*, Beijing, China, 2006.

[27] H. Ben Amor, E. Berger, D. Vogt, and B. Jung, "Kinesthetic bootstrapping: Teaching motor skills to humanoid robots through physical interaction," *KI 2009: Advances in Artificial Intelligence*, pp. 492–499, 2009.