Learning Sequential Motor Tasks

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

***Abstract*—Many real robot applications require the sequential use of multiple distinct motor primitives. This requirement implies the need to learn the individual primitives as well as a strategy to select the primitives sequentially. Such hierarchical learning problems are commonly either treated as one complex monolithic problem which is hard to learn, or as separate tasks learned in isolation. However, there exists a strong link between the robots strategy and its motor primitives. Consequently, a consistent framework is needed that can learn jointly on the level of the individual primitives and the robots strategy. We present a hierarchical learning method which improves individual motor primitives and, simultaneously, learns how to combine these motor primitives sequentially to solve complex motor tasks. We evaluate our method on the game of robot hockey, which is both difficult to learn in terms of the required motor primitives as well as its strategic elements.**

I. Introduction

In recent years, policy search methods for robotics [1]–[4] have yielded encouraging results on tasks which are infeasible to encode by hand or to teach via demonstration. For example, Ng and Coates [3] have used policy search to develop autonomous stunt helicopters, Kober et al. [2] have shown how to learn the game 'ball in a cup', and Kormushev et al. [4] learned how to flip pancakes.

In this paper we focus on learning to sequence motor primitives with policy search. A motor primitive encodes an elemental movement and is typically represented as parametrized policy. Policy search methods directly search for parameters of the primitives that yield high rewards. While, the use of policy search methods is in most cases limited to learning a single motor primitive, many complex tasks require the sequential combination of motor primitives. For example, playing a game of tennis does not only require a single hitting movement but a sequence of distinct hitting movements that finally result in scoring a point. Such a behavior requires strategic decisions on what type of motor primitive to use according to the current situation. Such primitives can also be sequenced in multiple ways to achieve a given task. Simultaneously representing a versatile strategic solution in the learned policy of the agent is desirable, as such knowledge typically improves the adaptability of the learned policy [5]. We present a policy search method that efficiently learns both the individual motor primitives and a versatile strategy to combine these primitives to achieve the long-term objective of the task.

Most policy search approaches are tailored for episodic policy search—the robot searches for a single parameter vector of the policy which is used throughout the whole episode. Hence, this setup allows the abstraction of a whole episode as a single decision: choosing the parameter vector of the policy at the start of the episode. Subsequently, the policy is executed with the specified parameters and the accumulated reward is observed by the agent. However, in order to multiple primitives sequentially, we need to use multiple parameter vectors sequentially throughout the episode. Learning such sequencing of motor primitives is mostly unsolved due to the high dimensionality of the problem.

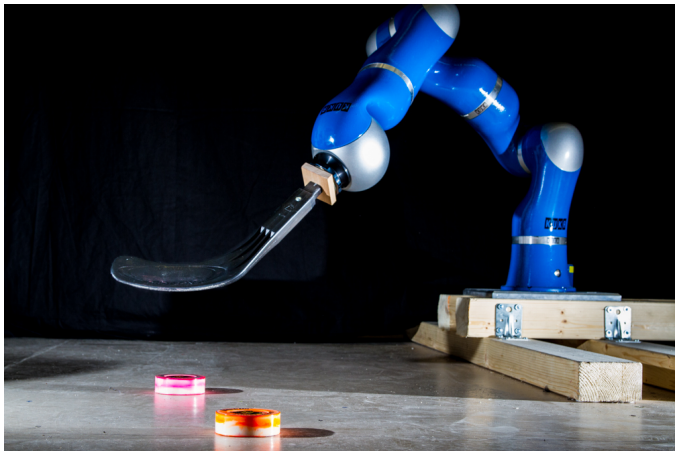

Fig. 1: The robot hockey task. The robot has to hit a puck into one of three target zones which yield different rewards. The puck can only be moved by shooting another puck at it. The robot has three shots to move the puck—overshooting the third zone yields zero reward.

In some approaches, the two problems of learning the motor primitives and sequencing the primitives are learned in isolation [6], [7]. However, such a strategy does not allow interaction between the two learning problems which might slow down learning or lead to an interference of both learning processes. Other approaches sequence primitives by using a combined policy parameter vector that contains the parameters of all primitives in sequence [8], [9]. Such strategy slows down learning as the parameter vector becomes unnecessarily high-dimensional. One approach for learning to sequence motor primitives in a single framework is given by Neumann et. al. [10]. However, this approach requires too many evaluations to be applied on a real robot.

In this paper, we extend an existing episodic policy search method to learn the sequencing of multiple motor primitives while simultaneously improving the individual primitives. We base our algorithm on the Hierarchical Relative Entropy Policy Search (HiREPS) method [5]. HiREPS has two desirable properties which we can utilize for learning sequential motor primitives: it can adapt the movement primitives to the current

[1]Technische Universität Darmstadt, Germany

[2]Max Planck Institute for Intelligent Systems, Tübingen, Germany

{daniel, neumann, kroemer, peters}@ias.tu-darmstadt.de

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

situation and it allows the robot to learn versatile solutions for a single motor task. In order to use HiREPS for learning a sequence of motor primitives, we extend the optimization problem defined by HiREPS to the finite-horizon case where each episode is composed of K motor primitives. The finite horizon formulation results in a time-indexed version of HiREPS, which allows us to learn individual policies for each decision time point. As we will show, we are able to efficiently solve the temporal credit assignment problem by the use of additional constraints imposed by our finite horizon formulation. The policies at the single decision time points are connected by these constraints such that they jointly try to maximize the accumulated reward of the whole episode.

We evaluate our algorithm on a new robot skill, robot hockey. This game requires the robot to learn both a good strategy as well as the individual motor primitives needed to execute the strategy. Our integrated method can efficiently learn both elements of the problem simultaneously and outperforms the episodic counterpart of the algorithm.

II. Episodic Policy Learning

Most policy-search algorithms are formulated in the episodic learning scenario. In the episodic case, a parametrized policy or movement primitive π with a given parameter vector $\boldsymbol{\omega}$ is executed for a whole episode to collect the accumulated rewards $\mathcal{R}_{\boldsymbol{\omega}}$ during the execution of the motor primitive. The parameter vector $\boldsymbol{\omega}$ of the policy can be chosen at the beginning of the episode [8], [11], [12], or, also, a noisy variant of $\boldsymbol{\omega}$ at each time step [2], [13]. All of the above algorithms assume that the policy parameter vector $\boldsymbol{\omega}$ is fixed for the whole episode — they cannot learn to execute several policies π_i, representing individual motor primitives, with different parameter vectors $\boldsymbol{\omega}_i$ in sequence. We can abstract the episodic learning scenario as a single decision task, such that the goal is to learn a high-level policy $\pi(\boldsymbol{\omega})$ which selects the parameters $\boldsymbol{\omega}$ of a motor primitive. This policy π can also be conditioned on the initial state $\boldsymbol{s}_0$ of the episode which allows the algorithm to adapt the motor primitive to the initial state of an episode. We will denote the accumulated reward of executing a movement primitive with parameters $\boldsymbol{\omega}$ and starting from state $\boldsymbol{s}_0$ as $\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}$.

A. Episodic Relative-Entropy Policy Search

We now briefly discuss the Relative Entropy Policy Search (REPS) algorithm [14] on which we base our learning framework. An episode consists of one action of the agent, e.g., one stroke of the robot arm. The main insight of REPS is that the difference between two subsequent policies during policy search should be bounded in order to avoid 'damaging' the policy by losing too much information about the old policy. This information loss can result in premature convergence or jumps in the policy parameter-space which can harm the robot.

REPS is formulated as a constrained optimization problem. The most important constraint results from limiting the change of the policy, which is achieved by bounding the Kullback-Leibler divergence (KL), or relative entropy, between two policies and their resulting state distributions

$$\epsilon \geq \sum\nolimits_{\boldsymbol{s},\boldsymbol{\omega}} \mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{\omega}|\boldsymbol{s}) \log \frac{\mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{\omega}|\boldsymbol{s})}{q(\boldsymbol{s},\boldsymbol{\omega})}, \tag{1}$$

where $\mu^{\pi}(\boldsymbol{s})$ denotes the state distribution of policy π and $q(\boldsymbol{s},\boldsymbol{\omega})$ the state-action distribution of the old policy. At the same time, REPS tries to optimize the expected reward

$$\max_{\mu,\pi} \sum\nolimits_{\boldsymbol{s},\boldsymbol{\omega}} \mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{\omega}|\boldsymbol{s})\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}.$$

In the episodic case, the parameter vector $\boldsymbol{\omega}$ only gets chosen in the beginning of the episode. Consequently, the state distribution $\mu^{\pi}(\boldsymbol{s})$ is defined to be the initial state distribution $\mu^0(\boldsymbol{s})$ of the episodes. Hence, the state distribution $\mu^{\pi}(\boldsymbol{s})$ is constrained by

$$\forall \boldsymbol{s} : \mu^{\pi}(\boldsymbol{s}) = \mu^0(\boldsymbol{s}). \tag{2}$$

Even though $\mu^{\pi}(\boldsymbol{s})$ is already specified, REPS still optimizes over $\mu^{\pi}(\boldsymbol{s})$ as the optimization is done over the joint state-action distribution $p(\boldsymbol{s},\boldsymbol{\omega}) = \mu^{\pi}(\boldsymbol{s})\pi(\boldsymbol{\omega}|\boldsymbol{s})$.

In order to satisfy this constraint for continuous state-spaces, REPS introduces feature averages — only the feature averages of $\mu^{\pi}(\boldsymbol{s})$ have to match the observed feature averages $\hat{\boldsymbol{\phi}}_0$ of the initial state distribution $\mu^0(\boldsymbol{s})$, i.e.,

$$\sum\nolimits_{\boldsymbol{s}} \mu^{\pi}(\boldsymbol{s})\boldsymbol{\phi}(\boldsymbol{s}) = \hat{\boldsymbol{\phi}}_0. \tag{3}$$

For example, if we define the features $\boldsymbol{\phi}$ to be all first and second order terms of the state $\boldsymbol{s}$, this formulation is equivalent to matching the first and second moments of both state distributions.

The resulting constrained optimization problem can be solved by using the method of Lagrangian multipliers and optimizing the resulting dual function g of the optimization problem. This dual function is known to be convex and easy to optimize [14]. Additionally, the REPS formulation can be used to derive a closed form solution for the joint state-action probabilities

$$p(\boldsymbol{s},\boldsymbol{\omega}) \propto q(\boldsymbol{s},\boldsymbol{\omega}) \exp\left(\frac{\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}} - V(\boldsymbol{s})}{\eta}\right), \tag{4}$$

where $V(\boldsymbol{s})$ is given by $\boldsymbol{\theta}^T\boldsymbol{\phi}(\boldsymbol{s})$. The vector $\boldsymbol{\theta}$ and the scaling parameter η are Lagrangian parameters which are obtained by optimizing the dual-function $g(\eta,\boldsymbol{\theta})$. The function $V(\boldsymbol{s})$ serves as a baseline. The scaling parameter η is computed by the algorithm such that the KL-bound in Eq. (1) is fulfilled.

The probabilities $p(\boldsymbol{s},\boldsymbol{\omega})$ are only calculated for a finite set of samples $\boldsymbol{s}_i$ and $\boldsymbol{\omega}_i$ and, subsequently, used to obtain a new policy π by performing a weighted maximum likelihood estimate on these data-points [5]. As these samples have been generated from the distribution $q(\boldsymbol{s},\boldsymbol{\omega})$, the term $q(\boldsymbol{s},\boldsymbol{\omega})$ can be cancelled out in Eq. (4) by the resulting importance weights. For this reason, the term $q(\boldsymbol{s},\boldsymbol{\omega})$ will be omitted in the discussion of the remaining algorithms.

Input: Information loss tolerance ϵ, Entropy tolerance κ, Number of options n, Number of time steps K, number of Iterations L.
Initialize π_k using n Gaussians with random mean for $k = 1, .., K$.

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

Collect samples

for $i = 1, \ldots, M$ ... # episodes

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

for $k = 1, \ldots, K$ ... # motor primitives

Sample action: $\boldsymbol{\omega}_{k,i} \sim q_k(\boldsymbol{\omega}|\boldsymbol{s}_{k,i})$
$= \sum_o \pi_{k,\text{old}}(o|\boldsymbol{s}_{k,i})\pi_{k,\text{old}}(\boldsymbol{\omega}|\boldsymbol{s}_{k,i}, o)$.

Execute action $\boldsymbol{\omega}_{k,i}$, observe next state $\boldsymbol{s}_{k+1,i}$ and reward $r(\boldsymbol{\omega}_{k,i}, \boldsymbol{s}_{k,i})$.

Observe Final Reward: $r(\boldsymbol{s}_{K+1,i})$.

Compute Responsibilities:
$\tilde{p}_k(o|\boldsymbol{s}_{k,i}, \boldsymbol{\omega}_{k,i}) = p_{k,\text{old}}(o|\boldsymbol{s}_{k,i}, \boldsymbol{\omega}_{k,i})$ for all k and i.

Minimize the dual function
$[\boldsymbol{\theta}_{1:K+1}, \boldsymbol{\eta}, \boldsymbol{\xi}] = \arg\min_{[\boldsymbol{\theta}_{1:K+1}, \boldsymbol{\eta}, \boldsymbol{\xi}]} g\left(\boldsymbol{\theta}_{1:K+1}, \boldsymbol{\eta}, \boldsymbol{\xi}\right)$.

Policy update:

for $k = 1, \ldots, K$

Compute model distribution
$p_k(\boldsymbol{s}_{k,i}, \boldsymbol{\omega}_{k,i}, o) \propto \tilde{p}_k(o|\boldsymbol{s}_{k,i}, \boldsymbol{\omega}_{k,i})^{1+\xi_k/\eta_k}$
$\exp\left(\frac{R_{k,i}+\mathbb{E}[V_{k+1}(\boldsymbol{s}')]-V_k(\boldsymbol{s}_{k,i})}{\eta_k}\right)$.

Estimate policies
$\pi_k(o|\boldsymbol{s})$ and $\pi_k(\boldsymbol{\omega}|\boldsymbol{s}, o)$ by weighted ML estimates.

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

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

B. Hierarchical Relative Entropy Policy Search

HiREPS [5] extends the REPS framework by introducing a hierarchical version of the policy

$$\pi(\boldsymbol{\omega}|\boldsymbol{s}) = \sum\nolimits_o \pi(o|\boldsymbol{s})\pi(\boldsymbol{\omega}|\boldsymbol{s}, o).$$

The policy can now choose from multiple motor primitives which we will also refer to as *options* [15]. The gating-policy $\pi(o|\boldsymbol{s})$ selects the motor primitive in a given state $\boldsymbol{s}$ and $\pi(\boldsymbol{\omega}|\boldsymbol{s}, o)$ is the parameter selection policy for the individual primitives.

This basic hierarchy allows the algorithm to represent multiple solutions of a motor task in one policy. In order to simplify the problem of learning such a hierarchical policy, the estimation of the policy is formulated as a latent variable estimation problem [5]. First, the responsibilities $\tilde{p}(o|\boldsymbol{s}, \boldsymbol{\omega})$ (see Table I) for each option and each sample $\boldsymbol{s}_i$, $\boldsymbol{\omega}_i$ are calculated. Subsequently, these responsibilities are used to update the probabilities $p(\boldsymbol{s}, \boldsymbol{\omega}, o)$ for each sample. These probabilities can then be used to estimate the new gating policy $\pi(o|\boldsymbol{s})$ and the parameter-selection policies $\pi(\boldsymbol{\omega}|\boldsymbol{s}, o)$.

In addition to the standard episodic REPS constraints, HiREPS uses an additional constraint to avoid overlapping of the learned options in the action space. This constraint forces HiREPS to concentrate its options on different modes of the reward-function and, hence, enables HiREPS to learn multiple solutions of a motor task in parallel. The non-overlapping constraint can be formulated efficiently by bounding the expected entropy of the responsibilities $\tilde{p}(o|\boldsymbol{s}, \boldsymbol{\omega})$ of the options

$$\kappa \geq \mathbb{E}_{\boldsymbol{\omega},\boldsymbol{s}}\left[-\sum\nolimits_o \tilde{p}(o|\boldsymbol{s}, \boldsymbol{\omega}) \log \tilde{p}(o|\boldsymbol{s}, \boldsymbol{\omega})\right]. \quad (5)$$

The bound in Eq. (5) results in the following closed-form solution for $p(\boldsymbol{s}, \boldsymbol{\omega}, o)$

$$p(\boldsymbol{s}, \boldsymbol{\omega}, o) \propto \tilde{p}(o|\boldsymbol{s}, \boldsymbol{\omega})^{1+\xi/\eta} \exp\left(\frac{\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}} - V(\boldsymbol{s})}{\eta}\right), \quad (6)$$

where ξ defines a Lagrangian parameter. For the exact derivations of the dual-functions of the episodic REPS and HiREPS we refer to [5].

III. Policy Learning for Sequential Motor Tasks

In this section, we reformulate the HiREPS framework to be applicable to learning a sequence of motor primitives. In order to do so, we formulate sequential motor primitive learning as a Markov Decision Process (MDP). We use parametrized motor primitives, with parameters $\boldsymbol{\omega}$, as actions of the robot. Each motor primitive o_i is executed for d_i seconds. This duration can also be parametrized by $\boldsymbol{\omega}$ and, hence, be part of the action of the agent. However, in this paper we will always use a pre-specified duration of the motor primitives.

The execution of a motor primitive in state $\boldsymbol{s}$ results in a transition into state $\boldsymbol{s}'$ with probability $\mathcal{P}^{\boldsymbol{\omega}}_{\boldsymbol{s}\boldsymbol{s}'} = p(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{\omega})$ and an expected reward of $\mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}} = r(\boldsymbol{s}, \boldsymbol{\omega})$ of the primitive. Both quantities will be estimated by samples. We use this adapted MDP notation to highlight the fact that we are not learning individual control signals, but rather a sequence of motor primitives.

We concentrate on the finite-horizon case, i.e., each episode consists of K steps where each step is defined as the execution of a whole motor primitive. In a finite horizon scenario the optimal primitive selection policy $\pi_k(\boldsymbol{\omega}|\boldsymbol{s}_k)$ and its state distribution $\mu^\pi_k(\boldsymbol{s})$ depends on the decision time step k. Our task is to learn a policy $\pi(\boldsymbol{\omega}|\boldsymbol{s})$ which maximizes the expected total reward

$$J = \mathbb{E}_{\boldsymbol{\omega}_{1:K},\boldsymbol{s}_{1:K+1}}\left[r(\boldsymbol{s}_{K+1}) + \sum\nolimits_{k=1}^{K} r(\boldsymbol{s}_k, \boldsymbol{\omega}_k)\right], \quad (7)$$

where the expectation is performed over the states $\boldsymbol{s}_{1:K+1}$ and the selected primitive parameters $\boldsymbol{\omega}_k \sim \pi_k(\boldsymbol{\omega}|\boldsymbol{s}_k)$. The term $r(\boldsymbol{s}_{K+1})$ denotes the final reward for ending up in the state $\boldsymbol{s}_{K+1}$ after executing the last motor skill.

A. Time-Indexed HiREPS

In order to optimize a sequence of motor primitives instead of a single primitive, we will use a finite horizon formulation of HiREPS. We have to find a policy which maximizes

$$J = \sum_{\boldsymbol{s}} \mu^\pi_{K+1}(\boldsymbol{s}) r(\boldsymbol{s}) + \sum_{k=1}^{K} \sum_{\boldsymbol{s},\boldsymbol{\omega}} \mu^\pi_k(\boldsymbol{s}) \pi_k(\boldsymbol{\omega}|\boldsymbol{s}) \mathcal{R}_{\boldsymbol{s}\boldsymbol{\omega}}, \quad (8)$$

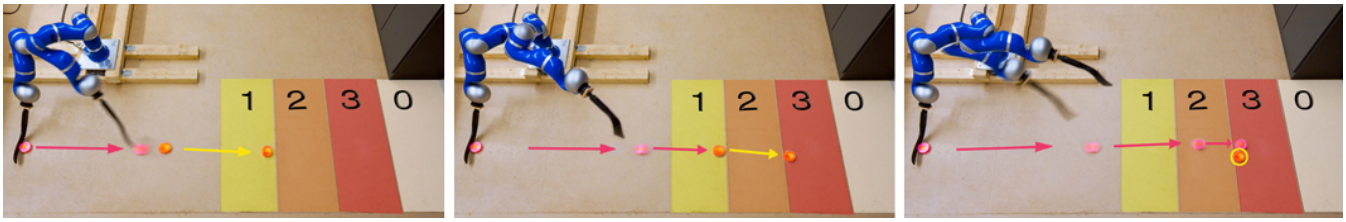

Fig. 2: One episode of the Hockey task, consisting of three strikes. Each picture shows the initial and final position of control and target puck. The movement of the pucks is indicated by arrows. The robot can shoot the pink control puck to move the target puck and tries to place the yellow target puck in one of the marked target zones while not overshooting. In the depicted episode the robot only needed two strikes to place the target puck into the highest reward zone. With the last strike the robot taps the target puck only slightly without actually moving it to avoid negative reward for missing it.

where K denotes the number of motor primitives to apply in each episode and $r(\boldsymbol{s})$ is the reward for the final state. Analogous to the episodic HiREPS, we decompose the policy $\pi_k(\boldsymbol{\omega}|\boldsymbol{s}) = \pi_k(\boldsymbol{\omega}|\boldsymbol{s},o)\pi_k(o|\boldsymbol{s})$ into a gating and a parameter selection policy for each time-step k. We also bound the distance between $p_k(\boldsymbol{s},\boldsymbol{\omega})$ and the observed distribution of the old policy $q_k(\boldsymbol{s},\boldsymbol{\omega})$, as well as the entropy of the responsibilities $\tilde{p}_k(o|\boldsymbol{s},\boldsymbol{\omega})$. Thus, the main principle remains unchanged from the episodic case. However, these bounds are now applied to each decision step k.

The most important difference to the episodic case lies in the application of the state transition constraints to the state distribution $\mu_k^\pi(\boldsymbol{s})$. In our finite horizon MDP, the state-distributions $\mu_k^\pi(\boldsymbol{s})$ have to be consistent with the policy $\pi_k(\boldsymbol{\omega}|\boldsymbol{s})$ and the transition model $\mathcal{P}^{\boldsymbol{\omega}}_{\boldsymbol{s}\boldsymbol{s}'}$, i.e.,

$$\forall \boldsymbol{s}', k : \mu_k^\pi(\boldsymbol{s}') = \sum\nolimits_{\boldsymbol{s},\boldsymbol{\omega}} \mu_{k-1}^\pi(\boldsymbol{s})\pi_{k-1}(\boldsymbol{\omega}|\boldsymbol{s})\mathcal{P}^{\boldsymbol{\omega}}_{\boldsymbol{s}\boldsymbol{s}'}, \quad (9)$$

for each step of the episode. These constraints connect the policies for the individual time-steps and result in a policy $\pi_k(\boldsymbol{\omega}|\boldsymbol{s})$ that optimizes the long-term reward instead of the immediate ones. Following the REPS formulation, these constraints are implemented by matching feature averages, i.e.,

$$\sum_{\boldsymbol{s}'} \mu_{k+1}^\pi(\boldsymbol{s}')\boldsymbol{\phi}(\boldsymbol{s}') = \sum_{\boldsymbol{s}'}\sum_{\boldsymbol{s},\boldsymbol{\omega}} \mu_k^\pi(\boldsymbol{s})\pi_k(\boldsymbol{\omega}|\boldsymbol{s})\mathcal{P}^{\boldsymbol{\omega}}_{\boldsymbol{s}\boldsymbol{s}'}\boldsymbol{\phi}(\boldsymbol{s}'), \quad (10)$$

for all k. This formulation requires the use of a model $\mathcal{P}^{\boldsymbol{\omega}}_{\boldsymbol{s}\boldsymbol{s}'}$ which needs to be learned from data. However, for simplicity, we will approximate this model by the single-sample outcome $\boldsymbol{s}'$ of executing the parameters $\boldsymbol{\omega}$ in state $\boldsymbol{s}$. This approximation is valid as long as we deal with almost deterministic systems. Learning the model $\mathcal{P}^{\boldsymbol{\omega}}_{\boldsymbol{s}\boldsymbol{s}'}$ is part of future work.

The complete optimization problem for the time-indexed HiREPS algorithm and its corresponding dual-function $g\left(\boldsymbol{\theta}_{1:K+1},\boldsymbol{\eta},\boldsymbol{\xi}\right)$ are given in the appendix. Due to our time-indexed formulation, we now have one $\boldsymbol{\theta}_k$ vector for each time step. The resulting function $V_k(\boldsymbol{s}) = \boldsymbol{\phi}(\boldsymbol{s})^T\boldsymbol{\theta}_k$ is also time-dependent. In addition, we have one scaling factor η_k and one ξ_k factor for each time step.

The joint distribution $p_k(\boldsymbol{s},\boldsymbol{\omega},o)$ can be determined from samples generated from the distribution $q_k(\boldsymbol{s},\boldsymbol{\omega})$,

$$p_k(\boldsymbol{s}_i,\boldsymbol{\omega}_i,o) \propto \tilde{p}_k(o|\boldsymbol{s}_i,\boldsymbol{\omega}_i)^{1+\xi_k/\eta_k} \exp\left(\frac{A_{k,i}}{\eta_k}\right), \quad (11)$$

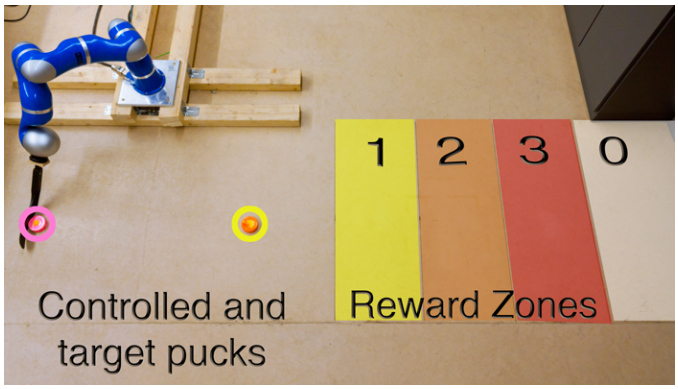

Fig. 3: The robot hockey task. The robot has two pucks, the pink control puck and the yellow target puck. The task is to shoot the yellow target puck into one of the colored reward zones. Since the best reward zone is too far away from the robot to be reached with only one shot, each episode consists of three strikes. After each strike the control puck is returned to the robot, but the target puck is only reset after one episode is concluded. Concluding an episode with the target puck in one of the reward zones yields rewards from one to three as indicated in the picture. However, if the robot shoots the target puck too far, the reward is zero.

where $A_{k,i}$ is given by the term

$$A_{k,i} = R_{k,i} + \mathbb{E}[V_{k+1}(\boldsymbol{s}')] - V_k(\boldsymbol{s}_i).$$

Similar to the episodic case, the reward $R_{k,i}$ is transformed into an advantage function. However, the advantage function now also depends on the expected value of the next state. This is not surprising as the advantage function has to solve the temporal credit assignment problem. The advantage function emerged naturally from the formulation of the constraint in Eq. (10).

For the state features, we will use a squared feature representation $\boldsymbol{\phi}(\boldsymbol{s})$ consisting of all linear and squared terms of the state in all our experiments. In order to increase the flexibility of the feature representation, we will use one individual feature representation for each option o_i by multiplying the squared feature representation with the responsibility $\tilde{p}(o_i|\boldsymbol{s})$ of the option for the current state. The overall feature vector is then defined as the concatenation of all option features. We use a Gaussian gating for the gating policy $\pi_k(o|\boldsymbol{s})$ and a Gaussian linear model for the parameter selection policy $\pi_k(\boldsymbol{\omega}|\boldsymbol{s},o)$. The parameters of both models can be easily obtained by weighted maximum likelihood estimates from the sample points.

IV. Evaluations

We first illustrate our hierarchical learning method on a basic via-point task, where the agent has to learn to sequence two motor primitives. We evaluated our learning algorithm on a variant of the game hockey. Here, a seven degree-of-freedom robot arm has to move a puck into one of three different target areas. The puck can only be moved by shooting another puck at it with a hockey stick.

We compared the time-indexed version of HiREPS with its episodic version, and show its superior performance on this complex learning task. The experiments are done in a physically-realistic robot simulator as well as with the real robot. In all our experiments, we used a KL-bound of 0.5 and a κ-value of 1.0. These values have proven to converge quickly and reliably in a wide range of experiments, and did not need to be adapted to individual tasks.

A. Policy Representation by Dynamical Motor Primitives

As parametrization of a single motor primitive we use Dynamic Movement Primitives [16], where we employ the refined version of the DMPs as presented by Kober et. al. [17]. A DMP describes a spring-damper system which is modulated by a forcing function $f(z;\boldsymbol{v}) = \vec{\psi}(z)^T\boldsymbol{v}$, where z is the phase of the movement. The parameters $\boldsymbol{v}$ define the shape of the movement and can be learned by imitation from a teacher's demonstration. In addition to the shape parameters $\boldsymbol{v}$, the trajectory can be influenced by setting the final position y_f as well as the final velocity $\dot{y}_f$ of the trajectory. For the exact description of the DMP framework we refer to [17].

After obtaining a desired trajectory through the DMPs, this trajectory is followed by the internal feedback-controller of the robot.

For our experiments, we record one demonstration of the approximate movement by kinaesthetic teach-in and extract the shape parameters $\boldsymbol{v}$ from that demonstration. In order to learn the task, we adapt the final positions and velocities of all joints, i.e., $\boldsymbol{\omega} = \{y_{f1}, ..., y_{f7}, \dot{y}_{f1}, ..., \dot{y}_{f7}\}$. In the presented tasks, the motor primitive is always executed for a predefined amount of time.

B. Feasibility Study on a Toy Task

Before testing on the real robot task, we evaluated the time-indexed HiREPS algorithm on a via-point task in order to illustrate the properties of our approach. In this task, we modelled a second-order dynamical system. The state of the agent is given by its position x and velocity $\dot{x}$. The actions u control the accelerations $\ddot{x}$. The task of the agent is to reach specified via-points at four different points in time. For each of these time points, different via-points exists. The reward at the time points is given by the negative squared distance to the closest via-point. In addition to the deviation to the via-points at the four specified time points, the reward function contains a squared punishment term for taking high accelerations. As we defined multiple via-points for each of the four time-points, this task has multiple solutions per construction. The exact setting of the task including its via-points is depicted in Figure 4.

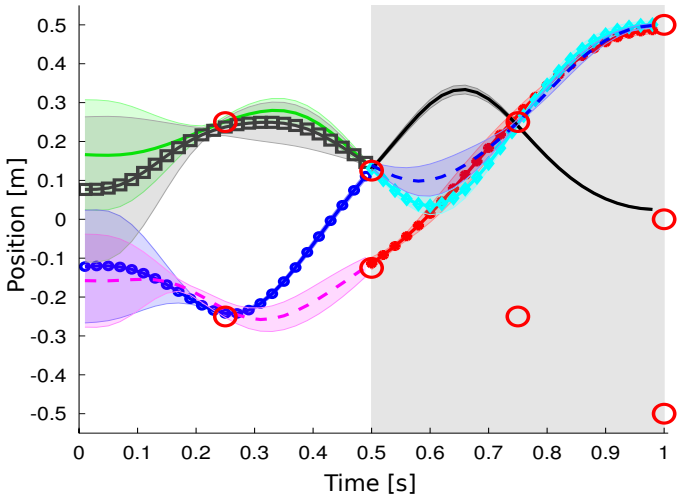

Fig. 4: We use this via-point task to illustrate our algorithm.The agent has to reach one of the via-points (denoted by red circles) at each of four specified times $[0.25, 0.5, 0.75, 1.0]$s. The reward is given by the negative squared distance to the closest via-point. The last via-point has to be reached with zero velocity. The initial position and velocity is sampled from a Gaussian distribution with zero mean and a standard deviation of 0.25 for the position and 0.1 for the velocity. In this task we learn to sequence two motor primitives, with the second primitve starting at $t = 0.5$s (shaded region in which the line colors change). This task is per construction multi-modal and illustrates how our algorithm learns distinct motor primitives. The mean and variance are indicate by shared error bars. The agent learned several, but not all possible solutions to solve the task.

In order to demonstrate the sequencing of motor primitives we decomposed the task into two DMPs which were executed sequentially. We used 5 shape parameters for both DMPs. In addition, we also learn the goal-parameter of the DMP, resulting in 6 parameters per movement primitive. To compare our sequencing method to the commonly used episodic policy search setup we also solve this task with the episodic version of HiREPS. In this case, we only used one DMP with ten shape parameters and the additional goal-parameter. For both scenarios the agent could choose between four distinct options o_i at each decision time-point. As we can see from Figure 4, the agent learned to select these primitives according to the state at the decision time-points as well as to adapt the primitives such the task can be solved. Our approach was able to learn multiple solutions for the task as can be seen from Figure 4. However, only a subset of all possible solutions was found.

The comparison of the episodic and the sequential learning methods can be seen in Figure 5 and clearly shows the advantage of our approach in learning speed as well as in the quality of the learned solution.

C. Evaluation on the Robot Hockey Task

In the robot hockey task, the robot has to move a target puck into one of three target areas. This target puck can only be moved by shooting a control puck at it. The target areas

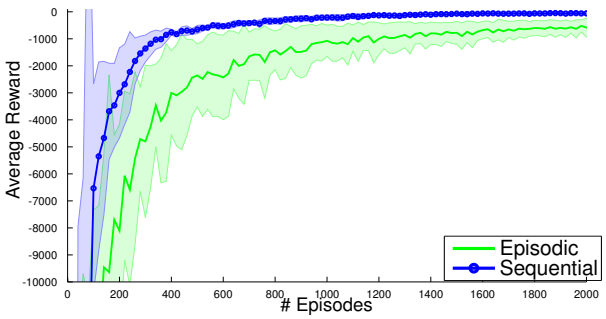

Fig. 5: The multi-modal via-point task learned with episodic and sequential motor primitive learning where the movement was decomposed into two primitives, see Figure 4 for a more detailed description. As we can see, our algorithm could exploit this decomposition resulting in increased learning speed and higher quality policies.

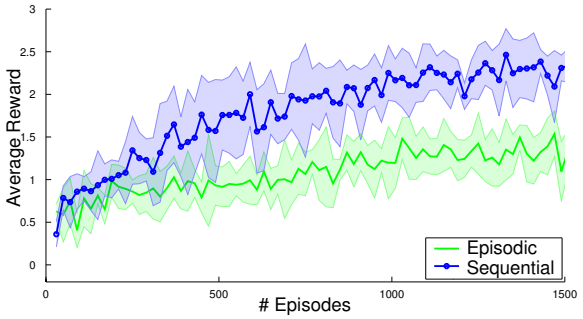

Fig. 6: Comparison of sequential motor primitive learning to the episodic learning setup on the simulated robot hockey task. The sequential motor primitive learning framework was able to find a good strategy to place the puck in the third reward zone in most of the cases while the episodic learning scenario failed to learn such a strategy.

are defined by a specified distance to the robot. The first zone is defined as distance from 1.4 to 1.8m, the second zone from 1.8 to 2.2m and the last zone from 2.2 to 2.6m. The robot gets rewards of 1, 2, and 3 for reaching zone 1, 2 or 3, respectively, with the target puck. The reward is only given after each episode which consists of three shots of the control puck. After each shot, the control puck is returned to the robot. The target puck, however, is only reset after each episode. The setup of the robot hockey task is shown in Figure 3.

The 2-dimensional position of the target puck defines the state of the agent. After performing one shot, the agent observes the new position of the target puck to plan the subsequent shot. In order to give the agent an incentive to shoot at the target puck, we punished the agent with the negative minimum distance of the control puck to the target puck after each shot. While this reward was given after every step, the zone reward was only given at the end of the episode (every third step) as $r(\boldsymbol{s}_{K+1})$.

We used a DLR-Kuka lightweight arm with 7 degrees of freedom as depicted in Figure 1. We used DMPs to represent single motor primitives where we only adapted the goal positions and velocities of the primitives. This resulted in 14 parameters per primitive per shot. Thus, the episodic HiREPS has to optimize one 42-dimensional parameter vector while the time indexed HiREPS has to optimize three policies with one 14-dimensional parameter space each. The shape parameters $\boldsymbol{v}$ of the single primitives were learned from imitation by collecting trajectory data via kinaesthetic teach-in.

Simulation Results: We first implemented a realistic simulation of the robot hockey task. In simulation, we varied the initial position of the puck by sampling the position from a normal distribution with standard deviation of 10cm. We compared our sequential motor primitive learning method with its episodic variant. For the episodic variant, we encoded the policy-parameters of all three shots into one policy, resulting in 42 parameters. The episodic variant cannot use state-feedback except for the information of the initial position of the puck. Using this feedback is not straightforward. To make the comparison as fair as possible we did not use any noise in our simulation, and hence, in theory, the initial position of the puck is sufficient to solve the task. The comparison of both methods can be seen in Figure 6. The episodic learning setup failed to learn a proper policy while our sequential motor primitive learning framework could steadily increase the average reward. Our method reached an average reward of 2.3 after learning for 1500 episodes. Note that an optimal strategy would have reached a reward value of 3, however, this is still a clear improvement in comparison to the episodic setup, which reached a final reward value of 1.4.

Real Robot Results: We used a Kinect RGB-D camera to observe the state of target puck . For the real robot hockey task, the initial position was not varied. On the real robot, we could reproduce the simulation results. The robot learned a strategy which could move the target puck to the highest reward zone in most of the cases after 300 episodes. Due to time constraints, the learning process had to be stopped before it converged to a final policy. In the final version of the paper, we will show more validation runs until convergence. One episode of robot hockey is depicted in Figure 2.

In the final trials, the robot tended to prefer using a soft hit in the first shot and to shoot the target puck to the last reward zone with the remaining two shots. This behaviour yielded higher average reward, since it is easier to only tap the target puck without moving it too much while it is still closer to the robot. During learning the robot steadily adapted his strategy when it mastered the necessary motor skills to achieve higher rewards by placing the target puck in the highest reward zones.

V. Conclusion & Future Work

In this paper, we introduced a new framework for learning sequential motor tasks. The algorithm learns how to sequence multiple movement primitives while simultaneously improving the individual primitives. Such learning capabilities are essential for learning many complex motor tasks.

We extended the HiREPS algorithm from the episodic to the finite horizon case which allows us to sequentially use K primitives in an episode. Such a sequence of primitives provides the agent with more flexibility and speeds up learning as the

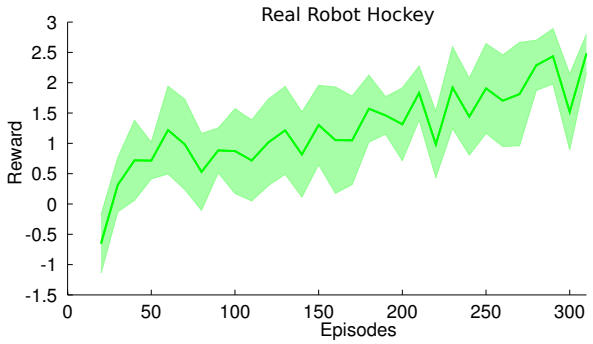

Fig. 7: One trial of of the real robot hockey tasks. The robot starts with a negative initial reward and learns to achieve an average reward of 2.5 after 300 episodes. The optimal theoretical reward of the presented task is 3.0. However, learning has not yet converged and had to be stopped prematurely due to time constraints.

overall task is divided into more natural and simpler learning problems. An important problem for learning sequential motor primitives, which is solved by our algorithm, is the temporal credit assignment problem. Our algorithm can evaluate the quality of each executed motor primitive individually instead of evaluating the outcome of an episode. As a consequence, our algorithm can use the collected data more efficiently which also results in an improved learning speed.

In future work we will investigate how to create new options, merge options if possible, and delete options if they are not needed any more. We will also investigate learning the transition models for the primitives in order to speed up learning.

APPENDIX

A. Time-Indexed HiREPS Optimization problem

The time-indexed HiREPS optimization problem is given by maximizing Eq. (8) under the constraints

$$\begin{aligned}
\text{s.t.} : \epsilon &\geq \sum_{\boldsymbol{s}} \mu^{\pi}_{K+1}(\boldsymbol{s}) \log \frac{\mu^{\pi}_{K+1}(\boldsymbol{s})}{q_{K+1}(\boldsymbol{s})}, \\
\hat{\boldsymbol{\phi}}_0 &= \sum_{\boldsymbol{s}'} \boldsymbol{\phi}(\boldsymbol{s}') \mu^{\pi}_1(\boldsymbol{s}'), \\
\forall k \leq K : \epsilon &\geq \sum_{\boldsymbol{s},\boldsymbol{\omega},o} p_k(\boldsymbol{s},\boldsymbol{\omega},o) \log \frac{p_k(\boldsymbol{s},\boldsymbol{\omega},o)}{q_k(\boldsymbol{s},\boldsymbol{\omega}) \tilde{p}_k(o|\boldsymbol{s},\boldsymbol{\omega})}, \\
\kappa &\geq \sum_{\boldsymbol{s},\boldsymbol{\omega},o} p_k(\boldsymbol{s},\boldsymbol{\omega}) \tilde{p}_k(o|\boldsymbol{s},\boldsymbol{\omega}) \log \tilde{p}_k(o|\boldsymbol{s},\boldsymbol{\omega}), \\
\sum_{\boldsymbol{s}'} \boldsymbol{\phi}(\boldsymbol{s}') \mu^{\pi}_{k+1}(\boldsymbol{s}') &= \sum_{\boldsymbol{s}'} \sum_{\boldsymbol{s},\boldsymbol{\omega},o} \mathcal{P}^{\boldsymbol{\omega}}_{\boldsymbol{s}\boldsymbol{s}'} p_k(\boldsymbol{s},\boldsymbol{\omega},o) \boldsymbol{\phi}(\boldsymbol{s}'),
\end{aligned} \tag{12}$$

where we skipped the normalization constraint for p and $p_k(\boldsymbol{s},\boldsymbol{\omega},o)$ is defined as

$$p_k(\boldsymbol{s},\boldsymbol{\omega},o) = \mu^{\pi}_k(\boldsymbol{s}) \pi_k(o|\boldsymbol{s}) \pi_k(\boldsymbol{\omega}|\boldsymbol{s},o).$$

B. Resulting Dual Function

The dual function of the time indexed HiREPS problem is obtained by deriving the Lagrangian of the optimization defined in 12 with respect to p and setting this derivative to zero. Substituting this solution back into the Lagrangian results in the dual.

$$\begin{aligned}
g(\boldsymbol{\theta}_{1:K+1}, \boldsymbol{\eta}, \boldsymbol{\xi}) &= \boldsymbol{\theta}_1^T \hat{\boldsymbol{\phi}}_0 + \sum\nolimits_k \epsilon\eta_k + \kappa\xi_k \\
&\quad + \sum_{k=1}^{K+1} \eta_k \log \left(\sum_i Z_{k,i} \right),
\end{aligned}$$

where $Z_{k,i}$ is defined as

$$\begin{aligned}
Z_{k,i} &= \sum_o \tilde{p}(o|\boldsymbol{s}_i,\boldsymbol{\omega}_i)^{1+\xi_k/\eta_k} \\
&\quad \exp\left(\frac{R_{k,i} + \boldsymbol{\theta}_{k+1} \mathbb{E}[\boldsymbol{\phi}(\boldsymbol{s}')] - \boldsymbol{\theta}_k^T \boldsymbol{\phi}(\boldsymbol{s}_{k,i})}{\eta_k} \right).
\end{aligned}$$

REFERENCES

[1] J. Peters and S. Schaal, "Natural Actor-Critic," *Neurocomputation*, vol. 71, no. 7-9, pp. 1180–1190, 2008.

[2] J. Kober, B. J. Mohler, and J. Peters, "Learning Perceptual Coupling for Motor Primitives," in *Intelligent Robots and Systems (IROS)*, 2008, pp. 834–839.

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

[4] P. Kormushev, S. Calinon, and D. G. Caldwell, "IEEE/RSJ International Conference on Intelligent Robots and Systems," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

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

[6] J. Morimoto and K. Doya, "Reinforcement Learning of a Dynamic Motor Sequence: Learning to Stand-Up," in *International Conference on Intelligent Robots and Systems (IROS)*, 1998.

[7] J. Kober and J. Peters, "Learning Elementary Movements jointly with a Higher Level Task," in *Intelligent Robots and Systems (IROS)*, 2011, pp. 338–343.

[8] C. Daniel, G. Neumann, and J. Peters, "Learning Concurrent Motor Skills in Versatile Solution Spaces," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[9] F. Stulp and S. Schaal, "Hierarchical Reinforcement Learning with Movement Primitives," in *2012 IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2011, pp. 231–238.

[10] G. Neumann, W. Maass, and J. Peters, "Learning Complex Motions by Sequencing Simpler Motion Templates," in *Proceedings of the 26th International Conference on Machine Learning*, ser. (ICML 2009), Montreal, Canada, 2009, pp. 753–760.

[11] F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, and J. Schmidhuber, "Parameter-Exploring Policy Gradients," *Neural Networks*, vol. 23, no. 4, pp. 551–559, 2010.

[12] V. Heidrich-Meisner and C. Igel, "Neuroevolution Strategies for Episodic Reinforcement Learning," *Journal of Algorithms*, vol. 64, no. 4, pp. 152–168, oct 2009.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.