Hierarchical Reinforcement Learning with Movement Primitives

Freek Stulp, Stefan Schaal
Computational Learning and Motor Control Lab
University of Southern California, Los Angeles, CA 90089, USA
stulp@clmc.usc.edu, sschaal@usc.edu

***Abstract*— Temporal abstraction and task decomposition drastically reduce the search space for planning and control, and are fundamental to making complex tasks amenable to learning. In the context of reinforcement learning, temporal abstractions are studied within the paradigm of *hierarchical* reinforcement learning.**

We propose a hierarchical reinforcement learning approach by applying our algorithm PI^2 to *sequences* of Dynamic Movement Primitives. For robots, this representation has some important advantages over discrete representations in terms of scalability and convergence speed. The parameters of the Dynamic Movement Primitives are learned simultaneously at different levels of temporal abstraction. The shape of a movement primitive is optimized w.r.t. the costs *up to the next* primitive in the sequence, and the subgoals between two movement primitives w.r.t. the costs *up to the end* of the entire movement primitive sequence.

We implement our approach on an 11-DOF arm and hand, and evaluate it in a pick-and-place task in which the robot transports an object between different shelves in a cupboard.

This paper is accompanied by a video:

http://www-clmc.usc.edu/~stulp/humanoids2011.mp4

I. Introduction

Reinforcement learning has the potential to substantially increase the autonomy, flexibility and adaptivity of robots when acquiring skills for everyday tasks. An essential component of such learning robots will be to exploit temporal abstractions, i.e. to treat complex tasks of extended duration (e.g. doing the dishes) not as a single skill, but rather as a sequential combination of skills (e.g. grasping the plate, washing the plate, rinsing the plate, putting it the rack, etc.) Such task decompositions drastically reduce the search space for planning and control, and are fundamental to making complex tasks amenable to learning.

Within the reinforcement learning paradigm, task decompositions are studied in the field of *hierarchical* reinforcement learning (HRL). Barto and Mahadevan provide an overview of HRL [1], and define some of the open research challenges – compact representations, large applications, dynamic abstractions, and learning task hierarchies. Now almost a decade later, these challenges are still far from solved, especially when HRL is to be implemented on physical robot systems.

In HRL, the learning problem is most commonly formalized as a discrete semi-Markov decision process [1]. In this paper, we consider an alternative representation based on dynamical systems theory. In particular, we represent options as Dynamic Movement Primitives [6]. This *compact representation* has some very desirable properties for robotics, in terms of scalability to high-dimensional tasks, applicability to continuous state and action spaces, compactness, and control. As a single motion primitive only has limited duration and applicability, temporally extended tasks require sequences of motion primitives. An example is shown in Fig. 1, where the task of reaching for an object, grasping it, and transporting it to another shelve is represented as a sequence of two Dynamic Movement Primitives.

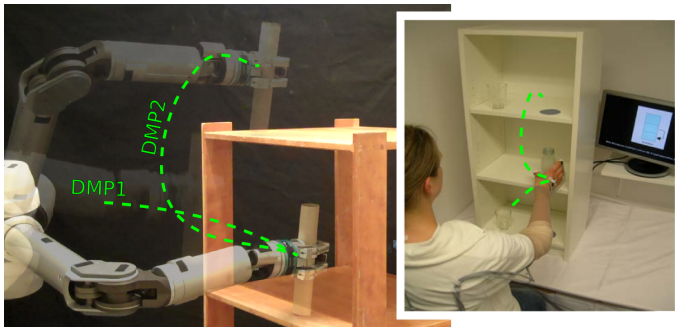

Fig. 1. The pick-and-place task that is used to evaluate hierarchical reinforcement learning. This task was inspired by a similar experimental set-up to analyze human pick-and-place behavior [14]. (Inset image taken from [14]).

The Policy Improvement through Path Integrals algorithm [17] (PI^2), which is derived from first principles of stochastic optimal control, exploits the advantages of Dynamic Movement Primitives. As PI^2 searches directly in the policy parameter space $\boldsymbol{\theta}$ – the parameters that determine the shape of the motion – it does not require a model of the robot or the environment. Combining movement primitives and model-free policy improvement has enabled us to apply reinforcement learning to very high-dimensional robotic tasks in *large applications* [16], [17]. In this paper, we extend PI^2 so that it can be applied to sequences of motion primitive.

In particular, the main contributions of this paper in apply-

This research was supported in part by National Science Foundation grants ECS-0325383, IIS-0312802, IIS-0082995, IIS-9988642, ECS-0326095, ANI-0224419, DARPA program on Advanced Robotic Manipulation, and the ATR Computational Neuroscience Laboratories. F.S. was supported by a Research Fellowship from the German Research Foundation (DFG). We thank the members of the Brain Body Dynamics Lab for emptying their cookie cupboard for our experimental set-up.

ing hierarchical reinforcement learning to optimize sequences of movement primitives are: • Deriving an update rule that allows PI^2 to also learn the optimal end-point of the motion represented by the goal parameters g. • Simultaneously learning shape *and* goal parameters in sequences of motion primitives at different levels of temporal abstraction, as depicted in Fig. 2. • Demonstrating how hierarchical learning leads to lower overall costs than optimizing motion primitives with respect only to local costs. • Applying these methods to simulated and real robots on a via-point and pick-and-place manipulation task.

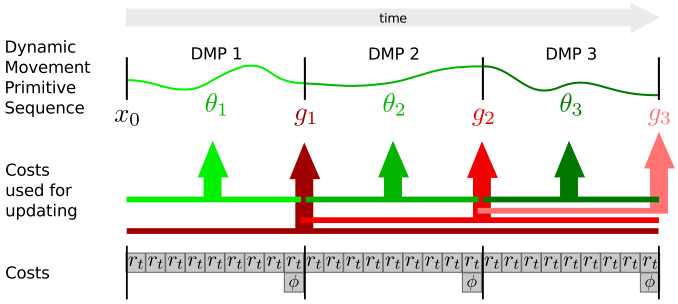

Fig. 2. Hierarchical Reinforcement Learning with Dynamic Movement Primitives. Learning happens at two levels of temporal abstraction. On the lower level, shape parameters $\boldsymbol{\theta}$ are optimized with respect to the cost-to-go within a motion primitive (green arrows). On the higher level, goal parameters g are optimized with respect to the total cost of the current *and* subsequent motion primitives in the sequence (red arrows). For each motion primitive, costs consist of instantaneous costs r_t, and terminal cost ϕ.

The rest of this paper is structured as follows. After discussing related work, we describe the movement representation and reinforcement learning algorithm that form the basis of our work in Section III. The main contributions, goal learning and hierarchical reinforcement learning with PI^2, are presented in Section IV. An evaluation of our methods on the main application task, a realistic everyday pick-and-place task, is presented in Section V. We conclude with Section VI.

II. Related Work

The term "hierarchical" in hierarchical reinforcement learning may refer to a number of hierarchical concepts, such as • Applying RL to the highest level of a control hierarchy, as in [5], [15], [11]. Technically, these approaches are not HRL (nor do they claim to be), because although RL is indeed *applied* to a hierarchical control system, learning *itself* only takes place one level of abstraction – the top level. • Using RL to generate skill hierarchies over time in a developmental approach [3]. • In this paper, as in [1], we specifically interpret HRL as the simultaneous learning of policy parameters in sequences of policies at multiple levels of temporal abstraction.

The most common formalization in HRL approaches in this last category is the semi-Markov decision process [1], where there is not one monolithic 'master' policy (as in Markov decision processes), but rather a collection of policies (also called options, macros, machines, skills, behaviors, etc.) which sequentially call lower-level policies depending on applicability and termination conditions. The lowest level of the hierarchy consists of a finite set of primitive actions, which last one discrete time step each. Exploiting temporal abstraction makes HRL much more efficient than 'monolithic' RL. Nevertheless, discrete action and state spaces still suffer from the curse of dimensionality [1], [9], and are in general not that well suited for the requirements of robot planning and control. Robotic tasks typically involve high-dimensional, continuous action spaces (humanoid robots with >30 degrees-of-freedom [16]), high-dimensional state spaces which are impossible to completely explore, on-the-fly adaptation of policies (for instance due to moving goals or to avoid obstacles [4]), and high costs for exploration trials on physical systems, in terms of time and maintenance.

HRL approaches that have been applied to real robots include [9], where RL is simultaneously applied to a high-level discretized state space and a low-level continuous state space to solve a pole balancing task. As in the work we present here, the overall task is optimized w.r.t. the cost of the entire sequence, whereas the low-level controllers are optimized w.r.t. cost for achieving their particular subgoal. An advantage of our work is that it does not rely on learning a state-value function, but searches directly in the space of the policy parameters. This reduces the number of roll-outs required to learn the task by an order of magnitude, and scales better to high-dimensional systems [16].

In [10], [7], the (moving) goal of the movement is determined such that a pre-specified velocity vector is achieved when coming into contact with a table tennis ball. In the pick-and-place task we consider, physical manipulation leads to discrete contact events that naturally define subgoals and transitions between controllers [2]. Given these discrete transitions and the fact that we want to grasp objects rather than dynamically hit them, having zero-velocity boundary conditions between the motion primitives is an advantage.

There are several other methods that could be used for learning optimal motion primitive goals, such as cross-entropy methods [12] or reward regression [8]. However, learning the goal of the movement is tightly coupled to learning its shape, and these methods do not readily apply to learning shape parameters, which have a temporally extended effect. Therefore, we propose a method that simultaneously learns shape and goal using the same cost function and update rule.

III. Model-free Reinforcement Learning with Parameterized Policies

In this section, we briefly introduce Dynamic Movement Primitives and the PI^2 reinforcement learning algorithm, which are presented in more detail in [6] and [17] respectively.

Dynamic Movement Primitives (DMPs) are a flexible representation for motion primitives [6], which consist of a set of dynamic system equations, listed and explained in Fig. 3. For the purposes of this paper, the most important aspect of DMPs are that they generate a motion (determined by the **shape parameters** $\boldsymbol{\theta}$) towards a goal end-point (determined by the **goal parameters** g).

Dynamic Movement Primitives

$$\frac{1}{\tau}\ddot{x}_t = \alpha(\beta(g - x_t) - \dot{x}_t) + \mathbf{g}_t^T\boldsymbol{\theta} \quad \text{Transform. system} \quad (1)$$

$$[\mathbf{g}_t]_j = \frac{w_j(s_t)\cdot s_t}{\sum_{k=1}^{p} w_k(s_t)}(g - x_0) \quad \text{Basis functions} \quad (2)$$

$$w_j = \exp\left(-0.5h_j(s_t - c_j)^2\right) \quad \text{Gaussian kernel} \quad (3)$$

$$\frac{1}{\tau}\dot{s}_t = -\alpha s_t \quad \text{Canonical. system} \quad (4)$$

Fig. 3. The core idea behind DMPs is to perturb a simple linear dynamical system (the first part of Eq. 1) with a non-linear component ($\mathbf{g}_t^T\boldsymbol{\theta}$) to acquire smooth movements of arbitrary shape. The non-linear component consists of basis functions $\mathbf{g}_t$, multiplied with a parameter vector $\boldsymbol{\theta}$.
We leave the details of DMPs to [6], [17]. For this paper, the important features of DMPs are: • When integrated over time, DMPs generate trajectories $[x_{d,t}\ \dot{x}_{d,t}\ \ddot{x}_{d,t}]$, which are used as for instance desired joint angles or desired end-effector positions. • DMPs converge from the initial value x_0 towards the goal parameter g. So at the end of the movement, $x_t = g$. • The general shape of the movement (i.e. the values of x_t between x_0 and g) is determined by the shape parameters $\boldsymbol{\theta}$.
Eq. 1 describes a 1-dimensional system. Multi-dimensional DMP are represented by coupling several dynamical systems equations as in Eq. 1 with one shared phase variable s. For an n-DOF arm for instance, an n-dimensional DMP can be used to generate desired joint angle trajectories. In multi-dimensional DMPs, each dimension has its own goal (g) and shape ($\boldsymbol{\theta}$) parameters.

The shape parameters $\boldsymbol{\theta}$ are commonly acquired through imitation learning, i.e. a DMP is trained with an observed trajectory through supervised learning [6]. The aim of policy improvement methods is to tune the policy parameters $\boldsymbol{\theta}$ such that they minimize a cost function. The imitated trajectory is thus not the end result, but rather an initialization for further improvement through learning. In this paper, we consider the generic cost function

$$J(\boldsymbol{\tau}_i) = \phi_{t_N} + \int_{t_i}^{t_N} (r_t + \frac{1}{2}\boldsymbol{\theta}_t^T\mathbf{R}\boldsymbol{\theta}_t)\ dt \quad \text{Traj. cost} \quad (5)$$

where J is the finite horizon cost over a trajectory $\boldsymbol{\tau}_i$ starting at time t_i and ending at time t_N. This cost consists of a terminal cost ϕ_{t_N}, an immediate cost r_t, and an immediate control cost $\frac{1}{2}\boldsymbol{\theta}_t^T\mathbf{R}\boldsymbol{\theta}_t$. The cost function J is task-dependent, and is provided by the user.

Policy improvement methods minimize cost functions through an iterative process of exploration and parameter updating, which we explain using Fig. 4. Exploration is done by executing a DMP K times, each time with slightly different parameters $\boldsymbol{\theta} + \boldsymbol{\epsilon}^{\boldsymbol{\theta}}{}_{t,k}$, where $\boldsymbol{\epsilon}^{\boldsymbol{\theta}}{}_{t,k}$ is noise which is added to explore the parameter space. This noise is sampled from a Gaussian distribution with variance $\boldsymbol{\Sigma}^{\boldsymbol{\theta}}$.

$$\frac{1}{\tau}\ddot{x}_t = \alpha(\beta(g - x_t) - \dot{x}_t) + \mathbf{g}_t^T(\underbrace{\boldsymbol{\theta} + \boldsymbol{\epsilon}^{\boldsymbol{\theta}}{}_{t,k}}_{\text{Shape exploration}}) \quad \text{DMP} \quad (6)$$

These 'noisy' DMP parameters generate slightly different movements $[\ddot{x}_{t,k}, \dot{x}_{t,k}, x_{t,k}]$, which each lead to different costs. Given the costs and noisy parameters of the K DMP executions, called *roll-outs*, policy improvement methods then update the parameter vector $\boldsymbol{\theta}$ such that it is expected to generate movements that lead to lower costs in the future. The process then continues with the new $\boldsymbol{\theta}$ as the basis for exploration.

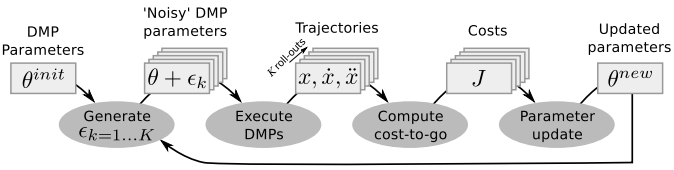

Fig. 4. Generic loop of policy improvement algorithms.

The most crucial part of the policy improvement loop in Fig. 4 is the parameter update; it is here that the key differences between PI^2 and other policy improvement methods lie. Rather than focussing on its derivation from first principles of stochastic optimal control, which is presented extensively in [17], we provide a post-hoc interpretation of the resulting update rule in Fig. 5. As demonstrated in [17], PI^2 often outperforms previous RL algorithms for parameterized policy learning by at least one order of magnitude in learning speed and also lower final cost performance.

IV. Hierarchical Reinforcement Learning

Our approach to hierarchical reinforcement learning hinges on two contributions: Section IV-A) learning goals of movement primitives; Section IV-B) simultaneous learning of shape and goal at different levels of temporal abstraction in a sequence of movement primitives.

A. Reinforcement Learning of Goals

To apply PI^2 to goal learning, the overall policy improvement loop for learning shape in Fig. 4 remains the same. Before executing a roll-out, goal exploration noise $\epsilon^g$and shape exploration noise $\boldsymbol{\epsilon}^{\boldsymbol{\theta}}$are generated by sampling from a Gaussian with variance $\boldsymbol{\Sigma}^g$ and $\boldsymbol{\Sigma}^{\boldsymbol{\theta}}$ respectively, and the DMP is executed with these noisy parameters:

$$\frac{1}{\tau}\ddot{x}_t = \alpha(\beta((\underbrace{g + \epsilon^g{}_k}_{\text{Goal exploration}}) - x_t) - \dot{x}_t) + \mathbf{g}_t^T(\underbrace{\boldsymbol{\theta} + \boldsymbol{\epsilon}^{\boldsymbol{\theta}}{}_{t,k}}_{\text{Shape exploration}}) \quad (13)$$

The goal is then updated through probability weighted averaging in Eq. 14–16, analogously to the updating of the shape parameters. The main difference is that only the cost-to-go at $t = 0$ is used to compute the probability. This means that we are using the total cost of the trajectory. The motivation behind this is that as the effect of g remains constant during execution, there is no temporal dependency of g on the cost. Note that $P(\boldsymbol{\tau}_{0,k})$ in Eq. 14 is equivalent to Eq. 9, with $t = 0$. Thus if shape parameters $\boldsymbol{\theta}$ are updated first, $P(\boldsymbol{\tau}_{0,k})$ is shared with the shape parameter update, and this probability need not be computed again.

PI^2 Shape Parameter Update Rule

$$S(\boldsymbol{\tau}_{i,k}) = \phi_{t_N,k} + \sum_{j=i}^{N-1} r_{t_j,k} + \frac{1}{2}\sum_{j=i+1}^{N-1} (\boldsymbol{\theta} + \mathbf{M}_{t_j,k}\boldsymbol{\epsilon}^{\boldsymbol{\theta}}{}_{t_j,k})^T \mathbf{R} (\boldsymbol{\theta} + \mathbf{M}_{t_j,k}\boldsymbol{\epsilon}^{\boldsymbol{\theta}}{}_{t_j,k}) \quad (7)$$

$$\mathbf{M}_{t_j,k} = \frac{\mathbf{R}^{-1}\mathbf{g}_{t_j}\ \mathbf{g}_{t_j}^T}{\mathbf{g}_{t_j}^T \mathbf{R}^{-1} \mathbf{g}_{t_j}} \quad (8)$$

$$P\left(\boldsymbol{\tau}_{i,k}\right) = \frac{e^{-\frac{1}{\lambda}S(\boldsymbol{\tau}_{i,k})}}{\sum_{l=1}^{K}[e^{-\frac{1}{\lambda}S(\boldsymbol{\tau}_{i,l})}]} \quad (9)$$

$$\delta\boldsymbol{\theta}_{t_i} = \sum_{k=1}^{K} \left[P\left(\boldsymbol{\tau}_{i,k}\right)\ \mathbf{M}_{t_i,k}\ \boldsymbol{\epsilon}^{\boldsymbol{\theta}}{}_{t_i,k}\right] \quad (10)$$

$$[\delta\boldsymbol{\theta}]_j = \frac{\sum_{i=0}^{N-1}(N-i)\ w_{j,t_i}\ [\delta\boldsymbol{\theta}_{t_i}]_j}{\sum_{i=0}^{N-1} w_{j,t_i}(N-i)} \quad (11)$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \delta\boldsymbol{\theta} \quad (12)$$

Fig. 5. The PI^2 parameter update consists of the following steps:
Eq. 7 – Determine cost-to-go of each roll-out. Compute the cost-to-go $S(\boldsymbol{\tau}_{i,k})$ at each time step i and for each roll-out k. This is an evaluation of the cost function $J(\boldsymbol{\tau}_i)$ in Equation 5 , which is task-dependent and provided by the user. The matrix $\mathbf{M}_{t_j,k}$ (Eq. 8) is needed to project the exploration noise onto the parameter space.
Eq. 9 – Compute probability of each roll-out. Compute the probability $P\left(\boldsymbol{\tau}_{i,k}\right)$ of each roll-out k at each time step i by exponentiating the cost-to-go. The intuition behind this step is that trajectories of lower cost should have higher probabilities.
Eq. 10 – Average over roll-outs. Compute the parameter update $\delta\boldsymbol{\theta}$ for each time step i through probability weighted averaging over the exploration $\boldsymbol{\epsilon}^{\boldsymbol{\theta}}$ of all K roll-outs. Trajectories with higher probability, and thus lower cost, therefore contribute more to the parameter update. Again, $\mathbf{M}_{t_j,k}$ is needed to project the exploration noise onto the parameter space.
Eq. 11 – Average over time-steps. In the final step, we average the parameter update $\delta\boldsymbol{\theta}_{t_i}$ per time step i over all time steps. Each parameter update is weighted according to the number of steps left in the trajectory. This is to give earlier points in the trajectory higher weights, as they influence a larger part of the trajectory. They are also weighted with the activation of the corresponding basis function w_j at time t_i, as the influence of parameter θ_j is highest when w_j is highest. Finally, the actual parameter update is performed with Eq. 12.
Eq. 12 – Update parameters. Add the parameter update to the current parameters to acquire the new parameters.

PI^2 Goal Parameter Update Rule

$$P\left(\boldsymbol{\tau}_{0,k}\right) = \frac{e^{-\frac{1}{\lambda}S(\boldsymbol{\tau}_{0,k})}}{\sum_{l=1}^{K}[e^{-\frac{1}{\lambda}S(\boldsymbol{\tau}_{0,l})}]} \quad \text{Probability} \quad (14)$$

$$\delta g = \sum_{k=1}^{K} \left[P\left(\boldsymbol{\tau}_{0,k}\right)\ \epsilon^{g}{}_{k}\right] \quad \text{Weighted averaging} \quad (15)$$

$$g \leftarrow g + \delta g \quad \text{Update} \quad (16)$$

By updating g in a similar fashion to updating $\boldsymbol{\theta}$, several important advantages are inherited from the PI^2 shape update rule: • Discontinuous and noisy cost functions are not a problem, as probability weighted averaging does not rely on computing a gradient. • Due to the averaging, $g + \delta g$ always lies within the convex hull of $g = g + \epsilon_k$. Thus, if the exploration is safe (joint limits are respected, the robot does not collide with itself or the environment), the new g after updating will also be safe. • Since g and $\boldsymbol{\theta}$ are updated simultaneously using the exact same costs and thus the same probability weights, there is no negative interference between learning g and $\boldsymbol{\theta}$.

Via-point task. We now introduce a simple via-point task. The goal of this tasks is not an extensive statistical evaluation (which is given in Section V), but rather to illustrate the differences between shape and goal learning, and show how the distribution of costs over motion primitives affects the resulting trajectories. In the via-point task, the real robot (system details given in the appendix) executes two subsequent DMPs, which have been initialized to generate straight motions with a bell-shaped velocity profile. The motions are represented in 3-D end-effector position space. The robot's task is to pass through two via-points (one for each motion) whilst minimizing the velocity at the end-effector. This task is expressed by the following cost function:

$$J(\boldsymbol{\tau}_i) = \int_{t_i}^{t_N} (10^3 C(t) + (\dot{\mathbf{x}}_t)^2 + \frac{1}{2}\boldsymbol{\theta}_t^T \mathbf{R}\boldsymbol{\theta}_t)\ dt \quad (17)$$

$$C^{DMP1}(t) = \delta(t-0.7)|\ \mathbf{x} - [0.65\ 0.75]^T\ | \quad (18)$$

$$C^{DMP2}(t) = \delta(t-0.3)|\ \mathbf{x} - [0.55\ 0.85]^T\ | \quad (19)$$

Here, $C^{DMP1}(t)$ is the distance to the via-point $[0.65\ 0.75]$ at $t = 0.7$ (similar for the second DMP); the z-coordinate is not relevant to the via-point. Furthermore, the end-effector velocity is penalized at each time step with $(\dot{\mathbf{x}}_t)^2$ to avoid high velocity movements. The control cost matrix is $\mathbf{R} = 10^{-6}\mathbf{I}$. The number off roll-outs per update is $K = 8$. When learning shape, the exploration noise for $\boldsymbol{\theta}$ is $\boldsymbol{\Sigma}^{\boldsymbol{\theta}} = 1.0\gamma^u$. When learning the goal, the exploration noise for g is $\boldsymbol{\Sigma}^g = 0.02\gamma^u$. In both cases, exploration decays as learning progresses, with u being the number of updates so far, and decay factor $\gamma = 0.95$. After 40 updates, the costs for all via-point learning sessions had converged, and learning was stopped.

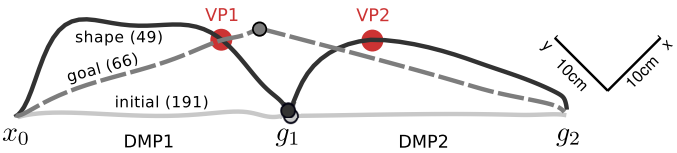

Fig. 6. End-effector path before (solid, light) and after learning of shape (solid, dark) *or* goal (dashed). Numbers in brackets are the costs of the entire trajectory.

Fig. 6 depicts the trajectories of the end-effector (projected onto the x, y-plane for clarity) before and after learning. Similar results for learning shape have been presented in [16]. The overall costs, which are discussed in more detail in Section IV-B, decreases from 191 to 49. For goal learning, the cost does not decrease as much (from 191 to 66), as g_1 cannot be placed such that both straight trajectories pass through the respective via-points. In this case, g_1 is learned such that the first trajectory passes through the via-point, but

the second trajectory doesn't. This leads to a higher cost then when learning shape (66>49), as discussed in more detail in Section IV-B.

B. Simultaneous Learning of Shape and Goal at Different Levels of Temporal Abstraction

The shape parameter update $\delta\boldsymbol{\theta}_{t_i}$ is computed per time step, by considering the cost-to-go starting at t_i, see Eq. 7-10. This is because a change in $\boldsymbol{\theta}$ at time t_i potentially influences the cost throughout the rest of the movement $S(\tau_t)$ for $t > t_i$. Similarly, in a sequence of D motion primitives, the goal of the d^{th} motion primitive g_d potentially influences not only the cost of the current trajectory $S(\tau_d)$, but also of the trajectories generated by subsequent motion primitives $S(\tau_{[d+1]\ldots D})$. For example, as we saw in the via-point task, a goal placed such that the cost of the first DMP is low might lead to high costs in the second DMP.

Therefore, we propose a hierarchical reinforcement learning approach in which 1) the shape parameters $\boldsymbol{\theta}$ are updated according to the cost-to-go *within* a motion primitive $S(\boldsymbol{\tau}_{t_i})$, as in Section III. 2) the goal parameters g_d of the d^{th} movement primitive are updated according to the total cost of the current *and all subsequent* motion primitives in the sequence. This approach is summarized in Eq. 20-23. In these equation, $S(\boldsymbol{\tau}_{0,d,k})$ denotes to the cost-to-go at $t = 0$ (i.e. the cost of the entire trajectory) of the d^{th} motion primitive in the sequence, for the k^{th} roll-out in the set of K exploration roll-outs.

PI^2 Subgoal Parameter Update Rule

$$S_{d,k} = \sum_{m=d}^{D} S(\boldsymbol{\tau}_{0,m,k}) \qquad \text{Subsequent sequence cost} \qquad (20)$$

$$P_{d,k} = \frac{e^{-\frac{1}{\lambda}S_{d,k}}}{\sum_{l=1}^{K}[e^{-\frac{1}{\lambda}S_{d,l}}]} \qquad \text{Probability} \qquad (21)$$

$$\delta g_d = \sum_{k=1}^{K} [P_{d,k}\ \epsilon^{g_d}{}_k] \qquad \text{Weighted averaging} \qquad (22)$$

$$g_d \leftarrow g_d + \delta g_d \qquad \text{Update} \qquad (23)$$

Via-point task. We apply Hierarchical Reinforcement Learning to the via-point task. The resulting path is depicted in Fig. 7. The resulting cost is substantially lower than when learn the shape or goal individually (40<49 and 40<66, see Fig. 6).

We have also implemented a greedy variant of hierarchical reinforcement learning, in which shape and goal are both optimized w.r.t. the cost *within* the motion primitive, but not subsequent primitives. These different costs for learning are depicted at the bottom of Fig. 7. The greedy approach leads to a slightly higher cost (43>40).

The costs (split up into the different cost components) for the via-point task for the different learning strategies are depicted in Fig. 8. From these results we draw the following

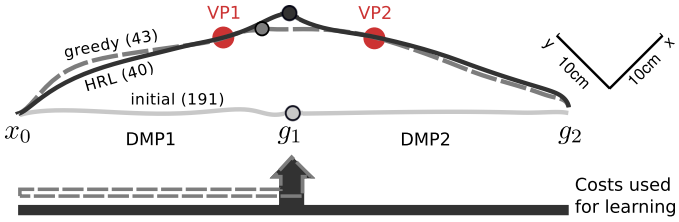

Fig. 7. End-effector path before (solid, light) and after simultaneous learning of shape *and* goal. Arrows below the trajectories depict the costs used to update the intermediate goal g_1. This difference leads to a greedy approach (dashed) and true hierarchical reinforcement learning (solid, dark). Numbers in brackets are the costs of the entire trajectory.

conclusions. *Learning shape*: the robot learns to go through both via-points, and the costs due to the distance error to the via-points decreases. However, the 'detours' that need to be made to do so lead to higher velocity costs than in the initial trajectory before learning (marked 'A'). *Learning goal*: Because it is not possible to put the intermediate goal such that both via-points are traversed, there is still a substantial cost due to the distance error to the second via-point when learning the goal alone (marked 'B'), and overall costs are higher than when learning shape. *Greedy approach*: PI^2 greedily optimizes the shape and the goal of the first DMP at the expense of higher costs for the second DMP. This can be seen in the large difference between the velocity costs for DMP 1 and 2 (marked 'C') *HRL*: The overall costs of the trajectory is minimized, which leads to an equal distribution of costs over the two DMPs (marked 'D'), and, more importantly, lower overall cost.

V. Application Domain: Pick-and-Place

In this section, we apply hierarchical reinforcement learning to a realistic everyday manipulation task. We consider a pick-and-place task, where the robot has to reach for grasp an object from a shelve, and place it on another shelve in the same cupboard. Exactly this task was recently used in experimental psychology [14], where humans are asked to grasp a bottle, and then place it on a shelve, as depicted in Fig. 1. Interestingly enough, humans grasp the bottle significantly lower if they are required to place it on a higher shelve, and vice versa [14]. Grasping the bottle in this way makes the second motion – transporting the bottle to the next shelve and placing it there – easier to perform, because it leads to a greater distance between the object and the shelve for the same general movement. This task clearly shows that motion parameters (where do I grasp the object?) are influenced by subsequent motions (transporting it to another shelve), and that there is a need to optimize intermediate goals of motions with respect to the cost of the *entire motion sequence*.

In our set-up, the robot is placed in front of an off-the-shelve cupboard with four shelves, the upper three of which are within the workspace of the robot. A cylindrical tube with 30cm height and 4cm radius is placed on the center shelve, and the robot is required to reach for the object, grasp it, transport it to a lower or higher shelve, and release the object. This

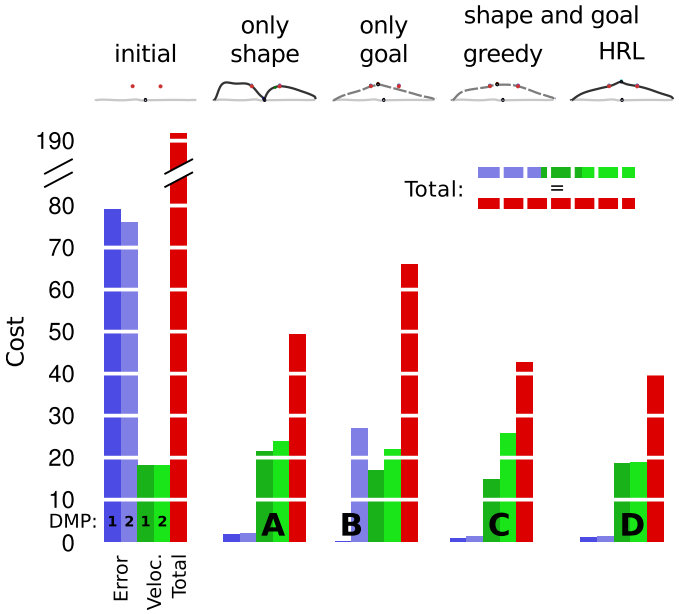

Fig. 8. Costs before and after learning the shape and/or goal. For each learning strategy, the different cost components are depicted, i.e. the costs due to deviations from the via-point (blue) and costs due to velocities of the end-effector (green). Immediate control costs ($\frac{1}{2}\boldsymbol{\theta}_t^T \mathbf{R}\boldsymbol{\theta}_t$) are very low, and not depicted for clarity. Furthermore, the costs have been split into the costs for the first and second motion. The total costs of the entire sequence are depicted in red. Note the 'broken' y-axis due to the high total cost (>190) before learning.

object was chosen as it is quite easy to grasp; grasp planning for complex objects is not the focus of this paper.

The task is solved by a sequence of two 7-DOF motion primitives, representing the position of the end-effector (3-DOF) and the posture of the hand (4-DOF). The orientation of the hand remains fixed throughout the entire motion, directly pointing at the shelves. The first motion reaches from the initial rest position x_0 to the object position g_1. The second motion primitive transports the object from g_1 to a position on the other shelve g_2. The parameters $\boldsymbol{\theta}$ of both these motions are trained through supervised learning, where the example movement consists of a reach-close-transport-release sequence as depicted in Fig. 9. It is acquired through kinesthetic teaching. The initial motion is shown in the video attachment.

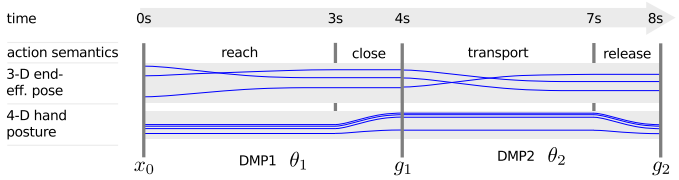

Fig. 9. Procedure used for grasping

To avoid collisions between the dynamic object and the static shelves, we implement a collision avoidance controller based on a potential field. We first determine the vector $\mathbf{p}$ from the point on the shelve that is closest to a point on the object[1]. If $|\mathbf{p}| < 0.15m$ then the there is no obstacle avoidance. Otherwise an acceleration away from the shelve is computed with $\ddot{x}_{avoid} = F\mathbf{p}/|\mathbf{p}|$, where the strength of the field is $F = 30.0(0.15 - |\mathbf{p}|)$. This field is visualized Fig. 10 by a red gradient[2]. $\ddot{x}_{avoid}$ is added to the transformation system (see Eq. 24) as a coupling term, as proposed in [4]. This causes the end-effector, and thus the object it is transporting, to smoothly move away from the shelve, as is shown in the video attachment.

$$\frac{1}{\tau}\ddot{x}_t = \alpha(\beta(g - x_t) - \dot{x}_t) + \mathbf{g}_t^T\boldsymbol{\theta} + \ddot{x}_{avoid} \quad (24)$$

The cost function for this task is

$$J(\boldsymbol{\tau}_i) = \phi_{t_N} + \frac{1}{N}\int_{t_i}^{t_N} (\ 4G(t) + |\ddot{x}_{avoid}| + |\ddot{x}|/250\ )\ dt \quad (25)$$

where the immediate costs consist three components: • $G(t)$, which is 0 if the object is in the gripper during the transport phase, and 1 otherwise (a high penalty for failing to grasp the object or dropping it). • $|\ddot{x}_{avoid}|$ is the acceleration due to the obstacle avoidance module (we don't want to come so close to the shelve such that obstacle avoidance needs to be active). • $|\ddot{x}|/250$ is the overall acceleration of the movement (we don't want high acceleration movements). The immediate costs are divided by the total number of time steps N, to make the cost independent from the duration of the movement. The terminal cost ϕ_{t_N} is the height of the object above the shelve at the time when releasing the object starts (we don't want to drop the object from too high). It only applies to the second DMP.

The number off roll-outs per update is $K = 5$. The exploration noise for the end-effector is the same as in the via-point task, i.e. $\boldsymbol{\Sigma}^{\boldsymbol{\theta}} = 1.0\gamma^u$ and $\boldsymbol{\Sigma}^g = 0.02\gamma^u$, where u is the number of updates so far. Exploration decays as learning progresses, with $\gamma = 0.9$. The 4-DOF posture of the hand over time is not learned, and exploration is therefore 0 for these transformation systems.

A. Results - Comparison of Upward and Downward Reaching

In simulation, the robot performed five learning sessions with hierarchical reinforcement learning for both placing the object on a higher and lower shelve. On the real robot, we performed one learning session for transporting the object up to the higher shelve. The end-effector paths before and after learning are depicted in Fig. 10. The learning curves, i.e. the total cost of the noise-free evaluation roll-outs as learning progresses, for moving the object up and down are depicted in the lower graph in Fig. 10. After 60 roll-outs, costs have converged towards the same costs for all learning sessions, as

[1]Currently, avoidance is based on known positions of the shelves. This approach could readily be replaced with a distance field based on point clouds, where $\mathbf{p}$ is the vector form the closest point in the point cloud

[2]Note that the top of the shelve is not avoided, else the object cannot be placed on the shelve. Also, the field is divided by 3 in the z direction, as the obstacle is initially quite close to the bottom of the shelve; we don't want the obstacle avoidance to be on right at the beginning of the movement.

the variance is very low. The residual costs after 60 roll-outs are all due to the end-effector acceleration ($|\ddot{x}|/250$); a certain amount of acceleration will always be required to perform the motion at all.

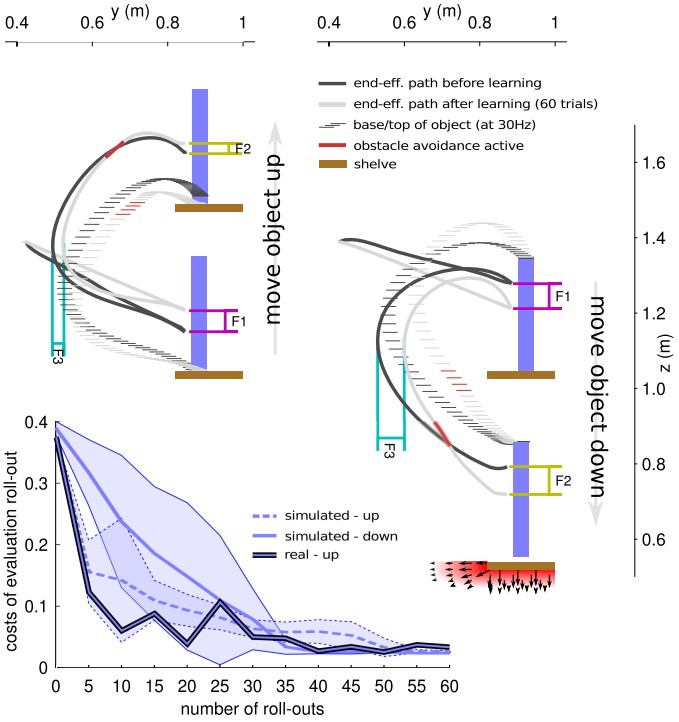

Fig. 10. For the simulation experiments, the end-effector paths for moving the object up (right graph) or down (left graph) both before (light gray) and after (dark gray) learning. Red segments of the path indicate that obstacle avoidance is active. The thin vertical lines represent the position of the object at 30Hz. For moving up it represents the base of the object, and for down the top. The learning curves for both the real robot (1 learning session) and the simulated robot ($\mu \pm \sigma$ over the five learning sessions) are depicted in the inset at the bottom.

Three relevant features are highlighted in Fig. 11: **F1** and **F2** are the z-coordinate of g_1 and g_2 respectively, relative to its initial value before learning. **F3** is the minimum value of the y-coordinate generated by the second DMP. The values of these variables as learning progresses are depicted in Fig. 11.

When moving the object up, F1 decreases, i.e. the object is grasped lower. This leads to a larger distance between the bottom of the object and the shelve when transporting it to the higher shelve (see Fig. 10, left), less activation of the obstacle avoidance module, and thus lower cost. Since grasping the object lower leads the object to be dropped from a higher position on release, this leads to an increased terminal cost ϕ_N. This cost is reduced by simultaneously decreasing F2[3], as seen in Fig. 11. When the object is to be moved downward, the effect is inverted. F1 increases to achieve more headroom for the object when transporting it, and F2 simultaneously

[3]Note that the cost due to not grasping an object $G(t)$ appears not to play a role in these results. The reason is that any roll-out in which the object is not grasped has such a high cost – and thus has almost zero probability – that it hardly contributes to probability weighted averaging. It plays a role during learning, but not in the end result.

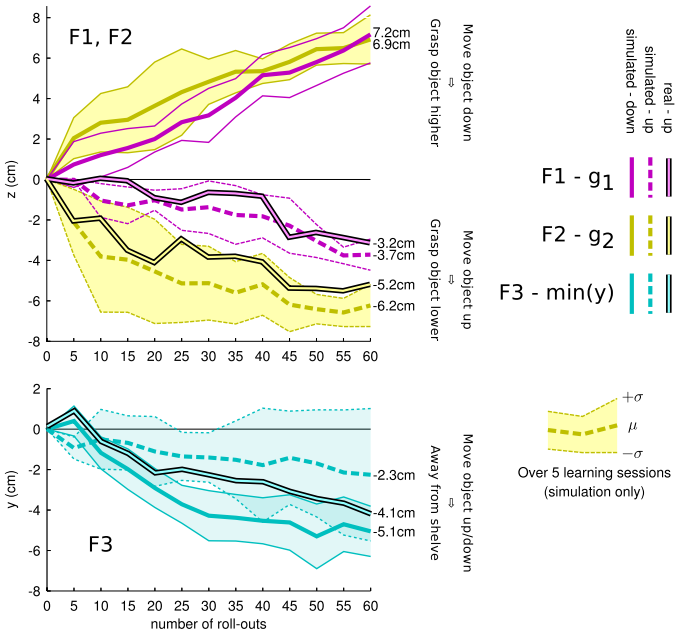

Fig. 11. Values of three features of the motion (F1/F2/F3) as learning progresses. Simulation: $\mu \pm \sigma$ over five learning sessions, for transporting the object both up and down. Real robot: one learning session for transporting up.

increases. Independent of whether the object is moved up or down, F3 decreases over time, i.e. the distance to the shelve is increased to reduce the cost due to the activation of the obstacle avoidance module. This last adaptation depends on changing the shape parameters $\boldsymbol{\theta}$ of the second DMP in the sequence.

In summary, hierarchical reinforcement learning adapts both the shape and goal to solve the task, and is able to reproduce the behavior seen in humans, who similarly adapt their grasp height to subsequent actions [14].

B. Results - Comparison of Learning Strategies

In this section, we compare the four learning strategies in simulation for the upward transport motion. We perform five learning session for each of the strategies used for the via-points experiment: 1) only shape; 2) only goal 3) shape *and* goal w.r.t. the costs of each DMP individually, i.e. 'greedy' 4) shape w.r.t the cost of each DMP individually, and goal w.r.t. the cost of the entire sequence cost, i.e. hierarchical reinforcement learning. The learning curves and resulting end-effector paths are depicted in Fig. 12.

Again, we see that HRL outperforms the other methods. After convergence of the costs at 60 roll-outs, it achieves both a lower mean and variance in cost over the five learning sessions. As can be seen from the end-effector paths in Fig. 12, the goal is obviously not changed when only learning shape, and coincides with the goal of the initial movement before learning. When learning the goal or using HRL, the grasp is moved downward, as we saw in the previous section. Interestingly enough, in the greedy case, the object is grasped *higher*. This is because the cost of the first DMP consists

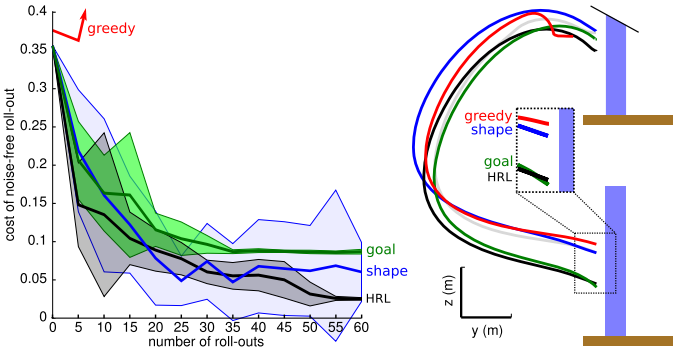

Fig. 12. Left: Learning curves ($\mu \pm \sigma$ over five learning sessions) for each of the learning strategies. Note that the greedy strategy soon jumps to a cost of over 1.0, and does not achieve a lower value afterwards. Hence, only its first two values are plotted. Right: End-effector paths for the different strategies after 60 roll-outs, averaged over the five sessions. The initial trajectory before learning is light gray. For clarity, only the path generated by the second DMP is depicted.

only of cost due to the acceleration ($|\ddot{x}|/250$), which can be reduced by placing the goal of the first DMP closer to its initial positions, i.e. moving the goal up. Unfortunately, this makes the second DMP much more difficult to execute, which leads to a very high cost for obstacle avoidance in the second DMP. This example clearly demonstrates the suboptimality of the greedy approach, and the necessity to optimize the goal of a DMP w.r.t. the cost of the overall DMP sequence, not the current DMP alone.

VI. Conclusion

In this paper, we address two of the main challenges in hierarchical reinforcement learning – compact representations and large applications [1]. Representing options as dynamical systems, rather than as sequences of discrete actions, has some very desirable properties for robotics in terms of compactness, scalability, and control. We demonstrate how simultaneous learning shapes and goals in sequences of such motion primitives leads to lower costs than taking into account local costs only, and, in the case of the pick-and-place task, behavior that mimics that of humans.

One remaining challenge [1], learning (or generating) task hierarchies, is not addressed in this paper. Within the movement primitive paradigm, we envision a three step approach to doing so: 1) Discover motion primitives in raw data streams; 2) Learn probabilistic pre-conditions/affordances of motion primitives that define when the motion primitive can be executed. 3) Chain motion primitives based on their pre-conditions with means-ends or symbolic planners. These steps are on our mid-term research agenda.

Appendix: Robot Platform

The robotic platform used in this paper is depicted in Fig. 1, and consists of a 7-DOF Barret WAM arm with a three-fingered 4-DOF Barret BH280 hand.

Low-level control and physical simulations of the robot are done with the SL software package [13], and high-level communications with the Robot Operating System `www.ros.org`. Desired task-space position/orientation trajectories are converted into joint space using the Jacobian pseudo-inverse. The resulting joint velocities are integrated and differentiated, to get joint positions and accelerations respectively. Feed-forward inverse dynamics torques for the arm are obtained from a recursive Newton Euler algorithm. Feed-back joint torques are obtained from low-gain joint PD controllers. All our controllers run at a rate of 300Hz on a host computer running the Xenomai real-time operating system.

References

[1] A. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event systems*, 13(1-2):41–77, 2003.

[2] J. R. Flanagan, M. C. Bowman, and R. S. Johansson. Control strategies in object manipulation tasks. *Current Opinion in Neurobiology*, 2006.

[3] S. Hart and R. Grupen. Learning generalizable control programs. *IEEE Transactions on Autonomous Mental Development*, 2010.

[4] H. Hoffmann, P. Pastor, Dae-H. Park, and S. Schaal. Biologically-inspired dynamical systems for movement generation: Automatic real-time goal adaptation and obstacle avoidance. In *IEEE International Conference on Robotics and Automation*, 2009.

[5] M. Huber and R. A. Grupen. Learning robot control - using control policies as abstract actions. In *NIPS'98 Workshop : Abstraction and Hierarchy in Reinforcement Learning*, 1998.

[6] A. J. Ijspeert, J. Nakanishi, and S. Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2002.

[7] J. Kober, 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*, pages 853–858, 2010.

[8] P. Kormushev, S. Calinon, R. Saegusa, and G. Metta. Learning the skill of archery by a humanoid robot icub. In *Proc. IEEE Intl Conf. on Humanoid Robots (Humanoids)*, Nashville, TN, USA, December 2010.

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

[10] K. Mülling, J. Kober, and J. Peters. Simulating human table tennis with a biomimetic robot setup. In *From Animals to Animats 11*. Springer Berlin / Heidelberg, 2010.

[11] B. Nemec, M. Tamosiunaite, F. Wörgötter, and A. Ude. Task adaptation thorough exploration and action sequencing. In *9th IEEE-RAS International Conference on Humanoid Robots*, 2009.

[12] R.Y. Rubinstein and D.P. Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*. Springer-Verlag, 2004.

[13] S. Schaal. The SL simulation and real-time control software package. Technical report, University of Southern California, 2009.

[14] A. Schubö, A. Maldonado, S. Stork, and M. Beetz. Subsequent actions influence motor control parameters of a current grasping action. In *IEEE 17th International Symposium on Robot and Human Interactive Communication (RO-MAN), Muenchen, Germany*, 2008.

[15] F. Stulp and M. Beetz. Refining the execution of abstract actions with learned action models. *Journal of Artificial Intelligence Research (JAIR)*, 32, June 2008.

[16] F. Stulp, J. Buchli, E. Theodorou, and S. Schaal. Reinforcement learning of full-body humanoid motor skills. In *10th IEEE-RAS International Conference on Humanoid Robots*, 2010. *Best paper finalist.*

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

[18] A. Ude, A. Gams, T. Asfour, and J. Morimoto. Task-specific generalization of discrete and periodic dynamic movement primitives. *IEEE Transactions on Robotics*, 26(5):800–815, 2010.