Preprint of “Reinforcement Learning with Sequences of Motion Primitives for Robust Manipulation”

Full citation: F. Stulp, E. Theodorou, and S. Schaal. Reinforcement Learning with Sequences of Motion Primitives for Robust Manipulation. *IEEE Transactions on Robotics*, 2012, 28(6), 1360-1370

Winner of the King-Sun Fu Best Paper Award of the IEEE Transactions on Robotics, for the year 2012.

Digital Object Identifier: 10.1109/TRO.2012.2210294

Abstract at IEEE Xplore: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6295672

This is a preprint from 16.07.2012, and differs from the final published version on IEEE Xplore.

1552-3098/$31.00

Reinforcement Learning with Sequences of Motion Primitives for Robust Manipulation

Freek Stulp[*†‡], Evangelos Theodorou[*§], Stefan Schaal[*¶]
[*]Computational Learning and Motor Control Lab, University of Southern California, Los Angeles, USA.
[†] Robotics and Computer Vision, ENSTA-ParisTech, Paris, France.
[‡] FLOWERS research team, INRIA Bordeaux Sud-Ouest, Talence, France.
[§] Computer Science and Engineering, Washington, Seattle, USA.
[¶] Max-Planck-Insitute for Intelligent Systems, 72076 Tübingen, Germany.
Corresponding author: freek.stulp@ensta-paristech.fr

***Abstract*—Physical contact events often allow a natural decomposition of manipulation tasks into action phases and subgoals. Within the motion primitive paradigm, each action phase corresponds to a motion primitive, and the subgoals correspond to the goal parameters of these primitives.**

Current state-of-the-art reinforcement learning algorithms are able to efficiently and robustly optimize the parameters of motion primitives in very high-dimensional problems. These algorithms usually consider only shape parameters, which determine the trajectory between the start- and end-point of the movement. In manipulation however, it is also crucial to optimize the goal parameters, which represent the subgoals between the motion primitives. We therefore extend the Policy Improvement through Path Integrals (PI^2) algorithm to simultaneously optimize shape *and* goal parameters. Applying simultaneous shape and goal learning to sequences of motion primitives leads to the novel algorithm PI^{2}SEQ.

We use goal learning and reinforcement learning with sequences of motion primitives to address a fundamental challenge in manipulation: improving the robustness of everyday pick-and-place tasks.

I. INTRODUCTION

In almost all activities of daily living, related tasks are encountered over and over again. Therefore, humans *"tend to solve similar or even identical instances over and over, so we can keep recycling old solutions with minor modifications"* [7]. Motor primitives are an effective way of representing solutions to specific tasks, because they drastically reduce the search space for control, make learning control in high-dimensional movement systems feasible [21], facilitate the design and online adaptation of the control systems [6], have negligible computational cost during execution, and allow for sensory prediction [15].

The motion primitive paradigm is particularly well suited for object manipulation tasks, as such tasks *"typically involve a series of action phases in which objects are grasped, moved, brought into contact with other objects and released."* [5]. These phases are especially obvious in manipulation, as they are *"usually bound by mechanical events that are subgoals of the task"* [5], which arise due to contact events between the manipulator and the object, or the object and the environment. Thus, the physical contact inherent in manipulation tasks implies a natural decomposition of the overall task into distinct phases and subgoals. Within the motion primitive paradigm, each phase corresponds to a motion primitive, and the subgoals correspond to the goal parameters of the primitives.

In recent years, several direct reinforcement learning algorithms have been introduced that are able to optimize motion primitive parameters efficiently and robustly in very high-dimensional problems [27], [28]. One challenge in using motion primitives and reinforcement learning for manipulation tasks is that the end-point of the movement often represents a grasp, which must be carefully adapted to the pose of the object. The first contribution of this article is therefore to extend the state-of-the-art Policy Improvement with Path Integrals (PI^2) algorithm so that it is able to learn the optimal goal of a motion primitive, i.e. the end-point of the trajectory it generates. We demonstrate how learning the optimal goal is easily integrated into learning the shape of a motion.

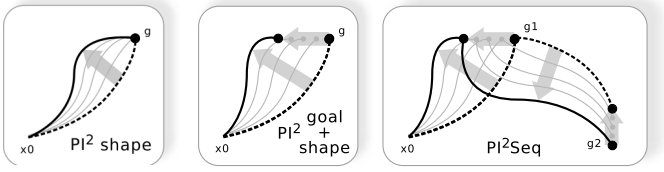

Fig. 1. Most direct reinforcement learning algorithms optimize only shape parameters (left), but do not consider the end-point of the movement, which we denote g. In this article, we extend the PI^2 algorithm so that it simultaneously learns shape *and* goal parameters (center). This approach is also applied to sequences of motion primitives (right).

Because each motion primitive has only a limited scope in terms of duration and the tasks it can achieve, more complex, temporally extended tasks *"involve sequentially organized action plans at various levels of complexity."* [5]. Therefore, the second contribution of this article is to apply PI^2 to *sequences* of motion primitives, which leads to the novel PI^{2}SEQ algorithm.

Our first two contributions advance path integral policy improvement beyond learning only shape parameters, and learning only with single motion primitives. We leverage these general contributions to address a fundamental challenge in

1552-3098/$31.00

the specific domain of autonomous manipulation: improving the robustness of skills for everyday pick-and-place tasks. Our contributions in the domain of manipulation – which we explain in more detail below using two application tasks – are learning to grasp under uncertainty with simultaneous shape and goal learning, and applying PI^{2}SEQ to a pick-and-place task that involves a sequence of motion primitives.

In the first task, depicted in Fig. 2, we consider grasping under uncertainty, where the object position is not known exactly but rather represented as a probability distribution. We formulate grasping under uncertainty as a reinforcement learning problem, where failed grasps are penalized. The third contribution of this article is to demonstrate that PI^2 is able to adapt the shape and goal of the trajectory to the uncertainty distribution of the object, as humans do [3]. The robot thus acquires *intrinsically robust* skills for manipulation under object position uncertainty.

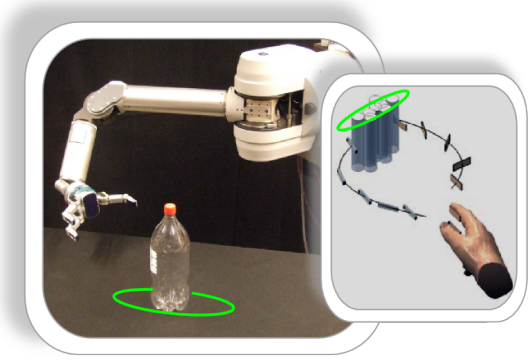

Fig. 2. Learning to grasp under uncertainty, the task that is used to evaluate goal learning. This task was inspired by behavioral experiments that analyze human grasping behavior under uncertainty [3].

The second task is a pick-and-place task where the robot has to reach for and grasp an object from a shelf, transport it, and place it on another shelf in the same cupboard, as depicted in Fig. 3. The fourth contribution is to demonstrate that learning on multiple levels of temporal abstraction with PI^{2}SEQ leads to adaptations that exploit the fact that *"subsequent actions influence motor control parameters of a current grasping action"* [20]. This leads to more robust pick-and-place behavior with sequences of motion primitive.

The experimental set-up for the two manipulation tasks we consider is equivalent to those in recent behavioral experiments [3], [20]. Though not intended as a modelling approach, it is interesting to see that our robot mimics the behavior of humans in both tasks.

The rest of this article is structured as follows. In the next section, we discuss related work. In Section III we present the main theoretical contributions: goal learning and reinforcement learning with sequences of motion primitives based on the direct policy algorithm PI^2. The practical application tasks – learning to grasp under uncertainty and pick-and-place with sequences of motion primitives – are presented in Section IV and V respectively. We conclude with Section VI.

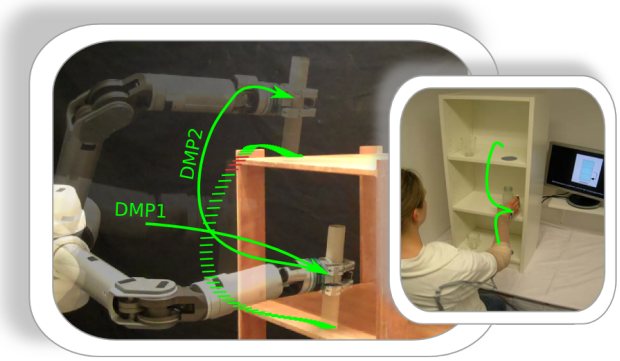

Fig. 3. The pick-and-place task that is used to evaluate direct reinforcement learning with sequences of motion primitives. This task was inspired by a similar experimental set-up used to analyze human pick-and-place behavior [20].

II. Related Work

A. Learning Motion Primitive Goals

Tamosiumaite et al. [26] optimize the goal parameters of a DMP through reinforcement learning, by iteratively approximating a continuous value function in state space. The robot learns to pour liquid into a container in a 2D parameter space, as 4 of the 6 degrees of freedom at the end-effector pose are kept fixed. Learning directly in policy parameter space without a value function allows us to scale to higher dimensional problems, enabling us to learn goal and shape parameters simultaneously in the full 11D action space (3D end-effector position, 4D end-effector orientation, and 4D hand posture).

Kober et al. [11], [16] present a variant of goal learning, where the robot learns a function $\gamma(s)$, which maps the current situation s to a set of meta-parameters, which are the goal and duration of the movement. The aim is to learn the mapping $\gamma(s)$ that minimizes a cost. Krömer et al. [12] also determine low-dimensional meta-parameters that automatically adapt the shape of the movement to a pre-specified goal, and apply it to grasping. Our specific aim in this article is to rather learn the goal and shape parameters simultaneously in the full dimensionality of the motion primitive.

Mülling et al. [14] determine the moving goal of a movement 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 [5]. 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 potentially be used for learning optimal motion primitive goals, such as cross-entropy methods [18] or reward-weighted regression [11]. However, learning the goal of the movement is tightly coupled to learning the shape of the movement, and these methods do not readily apply to learning shape parameters, as shape parameters have a temporally extended effect. Therefore, we propose a method that simultaneously learns shape and goal using the same cost function and update rule.

B. Grasping Under Uncertainty.

Uncertainty in an object's position can make even the most carefully planned motion fail, as the object might simply not be at the location where the planner expects it to be [13], [2]. There are several strategies to dealing with object pose uncertainty in grasping, for example: using sampling-based motion planners to generate a *robust motion plan* that is consistent with all object pose hypotheses [2]. This approach has high computational cost, and requires an accurate model of the robot and the environment; using exploratory actions to acquire more information to *actively reduce the uncertainty* [8]; using *reactive control* during execution, based on feedback from tactile or force/torque sensors, enabling the robot to adapt on-line to cases where the object is not at the expected location [9], [15]. In Section IV, we shall see that humans use yet another strategy, which forms the basis for the work described in Section IV-B.

III. From PI^2 Shape Learning to PI^{2}Seq

In Section III-A and III-B, we briefly introduce Dynamic Movement Primitives and the PI^2 reinforcement learning algorithm respectively. They are presented in more detail in [10] and [27]. These algorithms constitute the foundation which the main contributions of this article are built on. These contributions – goal learning and path integral policy improvement with sequences of motion primitives (PI^{2}Seq) – are explained in Section III-C and III-D respectively.

A. Dynamic Movement Primitives

Dynamic Movement Primitives (DMPs) are a flexible representation for motion primitives [10], which consist of a set of dynamic system equations, listed below.

Dynamic Movement Primitives

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

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

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

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

Fig. 4. The core idea behind DMPs is to perturb a simple linear dynamical system (the first part of Eq. 1) with a non-linear component ($\mathbf{g}_t^\intercal \boldsymbol{\theta}$) to acquire smooth movements of arbitrary shape. The non-linear component consists of basis functions $\mathbf{g}_t$, multiplied with a parameter vector $\boldsymbol{\theta}$. 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.

We leave the details of DMPs to [10], [27]. For this article, 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, for instance, are used as 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 $\boldsymbol{\theta}$.

B. Reinforcement Learning of Shape

The shape parameters $\boldsymbol{\theta}$ are commonly acquired through imitation learning, i.e. a DMP is trained with an observed trajectory through supervised learning [10]. 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 article, we consider the generic cost function

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

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 $\frac{1}{2}\boldsymbol{\theta}_t^\intercal \mathbf{R} \boldsymbol{\theta}_t$. 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. 5.

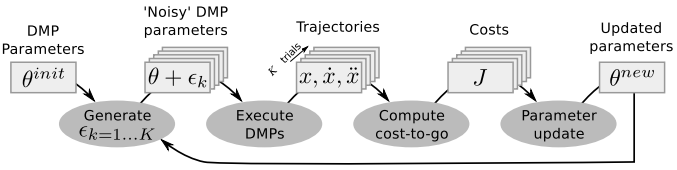

Fig. 5. Generic loop of policy improvement algorithms.

Exploration is done by executing a DMP K times, each time with slightly different parameters $\boldsymbol{\theta} + \{\boldsymbol{\epsilon}_t\}_k$, where $\{\boldsymbol{\epsilon}_t\}_k$ is noise which is added to explore the parameter space. This noise is sampled from a zero-mean Gaussian distribution with variance $\boldsymbol{\Sigma}$.

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

Each of these 'noisy' DMP parameters generate slightly different movements $\{\boldsymbol{\tau}_i\}_k = \{\ddot{x}_{t_i}, \dot{x}_{t_i}, x_{t_i}\}_k$, which each lead to different costs. We refer to the execution of a DMP as a 'trial', and the set of K exploration trials as an 'epoch'. Given the costs and noisy parameters of the K DMP executions, policy improvement methods then update the parameter vector $\boldsymbol{\theta}$ such that it is expected to generate movements that lead to lower costs. The process then continues with the new $\boldsymbol{\theta}$ as the basis for exploration.

The most crucial part of the policy improvement loop in Fig. 5 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 [27], we provide a post-hoc interpretation of the resulting update rule.

PI^2 Shape Parameter Update Rule

$$S(\{\boldsymbol{\tau}_i\}_k) = \{\phi_{t_N}\}_k + \sum_{j=i}^{N-1} \left\{ r_{t_j} + \frac{1}{2}\Theta_{t_j}^\intercal \mathbf{R}\Theta_{t_j} \right\}_k \quad (7)$$

$$\Theta_{t_j} = \boldsymbol{\theta} + \mathbf{M}_{t_j}\boldsymbol{\epsilon}_{t_j}, \qquad \mathbf{M}_{t_j} = \frac{\mathbf{R}^{-1}\mathbf{g}_{t_j}\,\mathbf{g}_{t_j}^\intercal}{\mathbf{g}_{t_j}^\intercal \mathbf{R}^{-1}\mathbf{g}_{t_j}}$$

$$P\left(\{\boldsymbol{\tau}_i\}_k\right) = \frac{e^{-\frac{1}{\lambda}S(\{\boldsymbol{\tau}_i\}_k)}}{\sum_{l=1}^{K}[e^{-\frac{1}{\lambda}S(\{\boldsymbol{\tau}_i\}_l)}]} \quad (8)$$

$$\delta\boldsymbol{\theta}_{t_i} = \sum_{k=1}^{K} \left[P\left(\{\boldsymbol{\tau}_i\}_k\right)\ \mathbf{M}_{t_i}\ \{\boldsymbol{\epsilon}_{t_i}\}_k\right] \quad (9)$$

$$[\delta\boldsymbol{\theta}]_j = \frac{\sum_{i=0}^{N-1}(N-i)\ w_{j,t_i}\ [\delta\boldsymbol{\theta}_{t_i}]_j}{\sum_{l=0}^{N-1} w_{j,t_l}(N-l)} \quad (10)$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \delta\boldsymbol{\theta} \quad (11)$$

Fig. 6. **Eq. 7 – Determine cost-to-go of each trial in the epoch.** Compute the cost-to-go $S(\{\boldsymbol{\tau}_i\}_k)$ at each time step i and for each trial 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. $\mathbf{M}_{t_j,k}$ is a projection matrix onto the range space of $\mathbf{g}_{t_j}$ under the metric $\mathbf{R}^{-1}$, cf. [27]. **Eq. 8 – Compute probability of each trial.** Compute the probability $P\left(\{\boldsymbol{\tau}_i\}_k\right)$ of each trial 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. The interpretation of P_k as a probability follows from applying the Feynman-Kac theorem to the stochastic optimal control problem, cf. [27]. **Eq. 9 – Average over trials.** The key algorithmic step. Compute the parameter update $\delta\boldsymbol{\theta}$ for each time step i through probability weighted averaging over the exploration $\boldsymbol{\epsilon}$ of all K trials, in short $\delta\boldsymbol{\theta} = \sum_{k=1}^{K} P_k\boldsymbol{\epsilon}_k$. Trajectories with higher probability, and thus lower cost, therefore contribute more to the parameter update. **Eq. 10 – Average over time-steps.** Average the parameter update $\delta\boldsymbol{\theta}_{t_i}$ over all time steps, where each parameter update at time step i 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. **Eq. 11 – Update parameters.** Finally, add the parameter update to the current parameters to acquire the new parameters.

As demonstrated in [27], 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. It also scales up to high-dimensional spaces, which enables PI^2 to learn full-body humanoid motor skills [21]. As an additional benefit, PI^2 has no open algorithmic parameters, except for the magnitude of the exploration noise $\boldsymbol{\epsilon}_t$ over time, and the number of trials per update K. We would like to emphasize that PI^2 is model-free, and *does not* require a model of the control system or the environment.

C. Reinforcement Learning of Goal and Shape

The PI^2 algorithm can optimize multiple sets of parameters at the same time [27]. Here, we exploit this PI^2 feature to learn both shape and goal parameters. To include goal exploration in the DMP, we extend Eq. 6 for shape exploration as follows:

$$\frac{1}{\tau}\dot{g}_t = \alpha_g(\underbrace{g + \{\epsilon^g\}_k}_{\text{Goal exploration}} - g_t) \quad (12)$$

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

The goal exploration noise ϵ^g is drawn from a Gaussian with variance $\boldsymbol{\Sigma}^g$. The set of explorations corresponding to one epoch $\{\epsilon^g\}_{k=1\ldots K}$ is sampled at the beginning of the epoch. We assume that α_g is chosen so large that, effectively, the goal state g_t converges immediately to $g + \epsilon^g{}_k$. Therefore, g_t is constant throughout the trial, with $g_t = g$.

In the goal parameter update rule in Eq. 14–16, the cost-to-go at $t = 0$ is used to compute the probability $P(\{\boldsymbol{\tau}_0\}_k)$. 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. 8, 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. Probability weighted averaging (Eq. 15) and goal updating (Eq. 16) are analogous to the update rule for $\boldsymbol{\theta}$.

PI^2 Goal Parameter Update Rule

$$P\left(\{\boldsymbol{\tau}_0\}_k\right) = \frac{e^{-\frac{1}{\lambda}S(\{\boldsymbol{\tau}_0\}_k)}}{\sum_{l=1}^{K}[e^{-\frac{1}{\lambda}S(\{\boldsymbol{\tau}_0\}_l)}]} \quad \text{Probability} \quad (14)$$

$$\delta g = \sum_{k=1}^{K} \left[P\left(\{\boldsymbol{\tau}_0\}_k\right)\ \{\epsilon^g\}_k\right] \quad \text{Weighted averaging} \quad (15)$$

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

By updating g in a similar fashion to updating $\boldsymbol{\theta}$, several important advantages are inherited from the PI^2 shape update rule[1]: the update rule is robust to discontinuous and noisy cost functions, as probability weighted averaging does not rely on computing a gradient; due to the probability-weighted averaging, $g + \delta g$ always lies within the convex hull of $g = g + \epsilon_k$, which alleviates the need for setting a learning rate – a crucial, but difficult to tune parameter in gradient-based methods; 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 $\boldsymbol{\theta}$.

[1]Shape parameters $\boldsymbol{\theta}$ are usually kept fixed after having been optimized with policy improvement methods, and the goal parameter g is used to adapt the end-point of the motion to specific scenarios, for instance to reach to different locations [10], [29]. Therefore, it might seem counter-intuitive to optimize g to a specific task context. But g is only specific to that context if it is learned in a global frame of reference. In the case of for instance object grasping, g should be learned relative to the object, thus representing an offset to the object (e.g. grasp 5cm to the right of the object). Therefore, if the object moves 20cm to the left, so does g.

D. Path Integral Policy Improvement with Sequences of Motion Primitives (PI^{2}SEQ)

Now that simultaneous learning of goal and shape is in place, we apply it to sequences of motion primitives. The most straightforward way of doing so is to optimize the shape and goal parameters of a DMP within the sequence with respect to the cost accumulated during the execution of that particular DMP. In sequences of DMPs however, the goal parameter g of one primitive is also the start parameter x_0 of the next primitive in the sequence, and may therefore influence the cost of executing the subsequent primitive. Therefore, we optimize goal parameters g with respect to the cost of the current DMP as well as the costs of the rest of the DMPs in the sequence.

We formalize this by denoting a sequence of trajectories as $\mathbf{\Pi} = [\boldsymbol{\tau}^1_{t_0 \dots t_N},\ \boldsymbol{\tau}^2_{t_0 \dots t_N},\ \dots,\ \boldsymbol{\tau}^D_{t_0 \dots t_N}]$, which is generated by a sequence of D DMPs. Furthermore, $\mathbf{\Pi}_d$ refers to the d^{th} trajectory $\boldsymbol{\tau}^d$ in the sequence of D trajectories. Given this notation, we define the cost-to-go of a trajectory in a DMP sequence as follows:

$$S(\mathbf{\Pi}_d) = \sum_{j=d}^{D} S(\boldsymbol{\tau}^j_{t_0}) \tag{17}$$

where $S(\boldsymbol{\tau}^j_{t_0})$ is the cost-to-go at t_0, i.e. the total cost of j^{th} trajectory in the sequence. The cost-to-go $S(\mathbf{\Pi}_d)$ is thus the total cost of the current trajectory, and all subsequent trajectories in the sequence.

When we add the notation for all K trajectories in an epoch (Eq. 17), the similarity between Eq. 17 and Eq. 7 (partially repeated in Eq. 18) becomes apparent. We are applying similar cost accumulation rules for $\boldsymbol{\theta}$ and g, but at different levels of temporal abstraction: for shape parameter updating we use $S(\{\boldsymbol{\tau}_i\}_k)$ over immediate costs r_{t_j} in a sequence of time steps, and for goal parameter updating we use $S(\{\mathbf{\Pi}_d\}_k)$ over total trajectory costs $S(\{\boldsymbol{\tau}^j_{t_0}\}_k)$ in a sequence of motion primitives. The similarity of these accumulations at different time scales is visualized in Fig. 7.

As a result, the update rule for subgoals in sequences of motion primitives is very similar to the rule for updating the goal of a single motion primitive, except that we use $S(\{\mathbf{\Pi}_d\}_k)$ instead of $S(\{\boldsymbol{\tau}_0\}_k)$, and need to update each of the subgoals $g_{d=1\dots D}$ in the sequence. During learning, shape parameters $\boldsymbol{\theta}$ are updated simultaneously according to Eqs. 7-11, i.e. using probability weighted averaging.

PI^2 Subgoal Parameter Update Rule

$$S(\{\boldsymbol{\tau}_{t_i}\}_k) = \dots + \sum_{j=i}^{N} \{r_{t_j}\}_k + \dots \qquad \text{Cost-to-go (trajectory)} \quad (18)$$

$$S(\{\mathbf{\Pi}_d\}_k) = \sum_{j=d}^{D} S(\{\boldsymbol{\tau}^j_{t_0}\}_k) \qquad \text{Cost-to-go (sequence)} \quad (19)$$

$$P(\{\mathbf{\Pi}_d\}_k) = \frac{e^{-\frac{1}{\lambda} S(\{\mathbf{\Pi}_d\}_k)}}{\sum_{l=1}^{K} [e^{-\frac{1}{\lambda} S(\{\mathbf{\Pi}_d\}_l)}]} \qquad \text{Probability} \quad (20)$$

$$\delta g_d = \sum_{k=1}^{K} [P(\{\mathbf{\Pi}_d\}_k)\ \{\epsilon^{g_d}\}_k] \qquad \text{Weighted averaging} \quad (21)$$

$$g_d \leftarrow g_d + \delta g_d \qquad \text{Update} \quad (22)$$

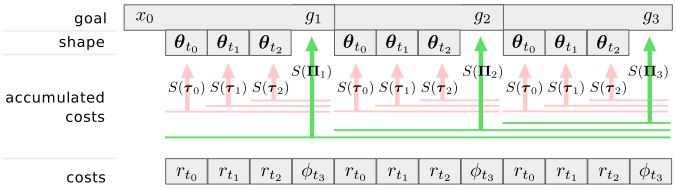

Fig. 7. The costs used to update $\boldsymbol{\theta}$ and g are accumulated similarly, but at different levels of temporal abstraction. The number of time steps t_N is usually much larger $N > 100$; it has been reduced to $N = 3$ for clarity of visualization.

IV. Application I – Grasping under Uncertainty

State estimation uncertainty can make even the most carefully planned motion fail, as the object might simply not be at the location where the planner expects it to be [13], [2]. In Section II-B, we listed three strategies for dealing with object pose uncertainty in robotic grasping: generate a robust motion plan [2]; execute exploratory actions to actively reduce the uncertainty [8]; use reactive control during execution [15], [9]. In recent experiments, Christopoulos and Schrater [3] demonstrate that humans use a fourth strategy to deal with position uncertainty.

In their experimental set-up, subjects perceived an object to be at a certain position $\hat{x}$. But when grasping the object, the actual position, which was hidden from the subject, was sampled from a Gaussian distribution $X \sim \mathcal{N}(\hat{x}, \Sigma)$. This *environmentally induced position uncertainty* artificially increases the uncertainty in the object's position. It was shown that over time, humans adapt their reaching motion and grasp to the shape of the uncertainty distribution, determined by the orientation of the main axis of the covariance matrix Σ. Furthermore, these adaptations lead to significantly better force-closure grasps [3]. Thus, rather than optimizing a grasp that achieves force-closure for the expected object position $\hat{x}$, humans learn one motion that optimizes the average force-closure for the grasps over *all* positions in the distribution $\mathcal{N}(\hat{x}, \Sigma)$.

In this article, we apply this experimental paradigm to robotics, by formulating it as a reinforcement learning problem

in which failed grasps are penalized, and inducing position uncertainty during learning. Since the resulting motion primitive has been trained off-line to deal with uncertainty that we have induced, it is more robust to state estimation uncertainty online during task execution.

A. Initialization of the Motion Primitive

The task is to grasp a plastic bottle with a radius of 5cm, as depicted in Fig. 2. The Dynamic Movement Primitive we use for this task has 11 dimensions; 3 to represent the position of the end-effector, 4 for its quaternion orientation, and 4 for the joint angles of the hand. The initial, preshape and grasp postures are demonstrated to the robot through kinesthetic teaching. The grasp posture constitutes the goal g which is optimized with PI^2. The shape of the DMP is initialized with two minimum jerk trajectories. One moves the hand from the initial to the grasp posture in 3s. The second sub-motion closes the gripper in 1s. The corresponding policy parameters $\boldsymbol{\theta}^{init}$ for this trajectory are acquired by training the DMP through supervised learning, as described in [10]. The parameters $\boldsymbol{\theta}$ are also optimized with PI^2. Fig. 8 illustrates the grasp procedure.

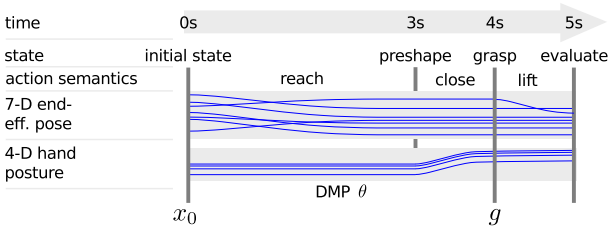

Fig. 8. Grasp procedure used for learning to grasp under uncertainty. The DMP generates desired trajectories for the 7-D end-effector pose (3-D position and 4-D quaternion orientation) and the 4 joint angles of the hand, from the initial state x_0 to the grasp posture g. The shape of this reaching movement is determined by $\boldsymbol{\theta}$. The whole movement takes 4 seconds. After grasping, the robot lifts the object, after which the grasp is evaluated.

B. Formulation as a Reinforcement Learning Problem

The cost function for this task is

$$J_{gr}(\boldsymbol{\tau}_i) = \phi_{t_N} + \int_{t_i}^{t_N} (10^{-9}(\ddot{\mathbf{x}}_t)^2 + \frac{1}{2}\boldsymbol{\theta}_t^{\intercal}\mathbf{R}\boldsymbol{\theta}_t)\, dt \qquad (23)$$

$$\phi_{t_N} = (1 - \text{success of lifting}) \qquad (24)$$

where $\mathbf{R} = 10^{-7}\mathbf{I}$. The terminal cost ϕ_{t_N} reflects if the robot successfully grasped and lifted the object, and is determined during the 1s lifting motion after executing the DMP. The 'success of lifting' is determined by measuring the time (0s-1s) each finger was in contact with the object, and averaging over the 3 fingers. For instance, if all three fingers were in contact with the objects throughout the lifting motion, the cost is $\phi_{t_N} = (1-1) = 0$, if the object was knocked over and not lifted at all $\phi_{t_N} = (1-0) = 1$, and if only two fingers were in contact and the object dropped out after 0.40s, $\phi_{t_N} = (1 - \mathbf{E}(0.40 + 0.40 + 0.00)) = 0.73$. A finger is deemed to be in contact with the object if the value of the strain gauge of the finger exceeds a certain threshold. Accelerations are penalized at each time step with $10^{-9}(\ddot{\mathbf{x}}_t)^2$ to avoid high-acceleration movements.

The parameters for the PI^2 learning algorithm are set as follows. The exploration noise for shape is $\boldsymbol{\Sigma}_{\boldsymbol{\epsilon}}$ = 3.0, 15.0 and 30.0 for the position, quaternion orientation and finger joint angles respectively. For the goals $\boldsymbol{\Sigma}_g$ = 0.03, 0.07, 0.03 respectively. All exploration decays with γ^u, where u is the number of updates so far, and $\gamma = 0.85$. For a discussion of how these parameters are selected, we refer to Section IV-D.

1) Environmentally Induced Position Uncertainty: We use two uncertainty distributions for the object position. The object is placed {-6,-4,0,4,6} centimeters from the perceived object position along either the x or y-axis. These distributions are intended to represent the $\mu \pm 2\sigma$ intervals for $\sigma = \{2\text{cm}, 3\text{cm}\}$; typical values for our state estimation module. The 'perceived' object position is made known to the robot, but not the actual possibly displaced position, allowing us to control the amount of uncertainty as in [3]. The robot must thus learn a motion that not only successfully grasps the object at the expected position, but also at the other positions in the distribution.

During learning, the robot executes the same exploratory motion for each of the 5 object positions. The cost for this exploratory motion is then the average of these 5 trials. We thus use the *expected* cost of an exploratory motion. For instance, if the robot successfully grasped 3 out of 5 objects, the cost for failed grasping for this exploratory motion is (0+0+0+1+1)/5=0.4. For each PI^2 update, we perform 10 new trials to compute the expected cost for 2 exploratory motions. To accelerate learning, we also reuse the 10 best trials from the previous update, as described in [21].

C. Empirical Evaluation

For each of the two uncertainty distributions (5 positions aligned with either the x or y-axis), 3 learning sessions were performed with 10 updates per session. Fig. 9 depicts the learning curves for these 6 learning sessions. The variation between learning session is rather large in the beginning. But after 7 updates (80 trials), all motions lead to successful grasping and lifting of objects at all the positions, although one session ends with the robot lifting the object rather precariously with only two fingers.

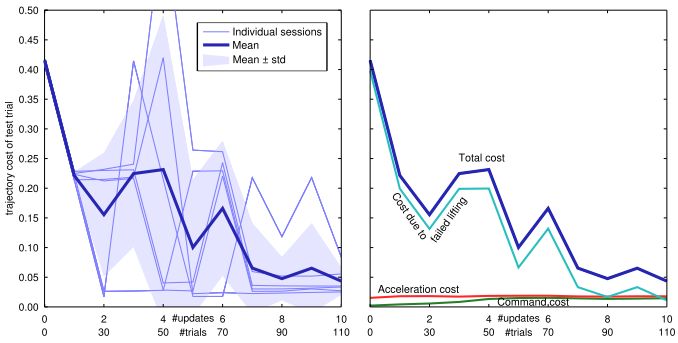

Fig. 9. Learning curves of the grasping under uncertainty experiments. Left: Learning curves of the individual learning sessions, as well as their $\mu \pm \sigma$. Right: Mean learning curves, split into the three different cost components that constitute the total cost of Eq. 23.

Initially before learning, the average cost over both uncertainty distributions is 0.42, and after learning it is 0.04

(averaged over the 6 learned movements). To analyze the importance of learning shape *and* goal, we executed the learned shape with the initial goal and vice versa. When the goal g is set to the initial value, but the learned $\boldsymbol{\theta}$s are used, the average cost is 0.32. When the learned goals g are used, but the initial $\boldsymbol{\theta}$, the average cost is 0.56. Learning both goal and shape adaptation is thus essential in achieving the low cost of 0.04, which corresponds to always successfully lifting the object, with only small costs due to acceleration and command cost.

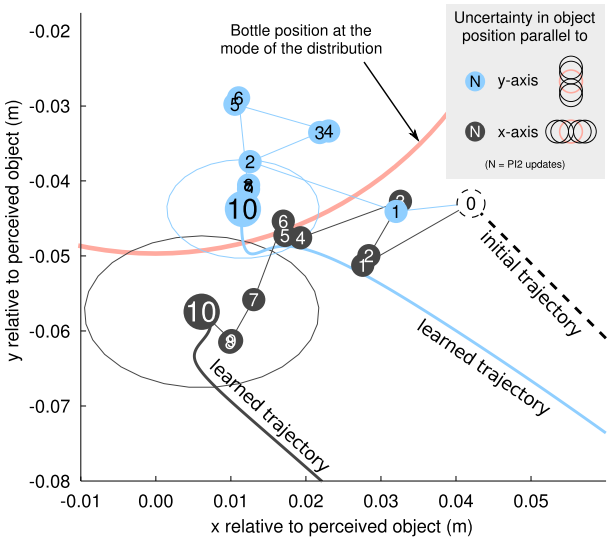

Fig. 10. The location of the goals g during learning after $0 \ldots 10$ updates, averaged over 3 learning sessions per uncertainty distribution. The trajectories towards the goal before (update=0) and after learning (update=10) are depicted as well. The variance in the goal location after learning (update=10) over the 3 learning session is depicted as an error ellipse around the final location. The legend depicts the two uncertainty distributions (horizontal and vertical). The partial red circle depicts the outline of the bottle (radius of 5cm) when it is at the mode of the distribution at (0,0).

To analyze the learned movements, Fig. 10 depicts how the location of the goals change during learning. Here, the mode of the object distribution is at (0,0). The goals are adapted in different directions, depending on the orientation of the uncertainty in object position. In both cases, they move closer to the objects to reach far enough to grasp the furthest ones in the distribution.

Because not all adaptations are consistent across all movements, we highlight some features of one of the learned motions in Fig. 11, which depicts the desired postures of the hand before and after a learning session where the object uncertainty distribution is aligned along the y axis.

In summary, PI^2 is able to adapt the shape and goal of motion primitives so that they are robust towards state estimation uncertainty in the position of the object to be grasped.

In previous work, we used only shape learning to acquire intrinsically robust motion primitives. We demonstrated that learned movements generalize well to different object positions on the table [23]. For instance, a motion primitive optimized to grasp an object at one perceived position was able to successfully grasp all perceived objects (with the same position uncertainty distribution) within a 40x30cm area. We also demonstrated that robust motion primitives can be learned for more complex non-convex objects. In several other tasks, we also previously demonstrated that learning goal *and* shape simultaneously can substantially outperform learning only shape [22], [24].

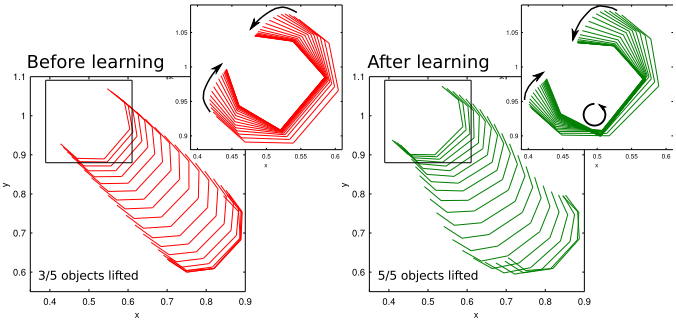

Fig. 11. Hand posture during the motion of one of the learning sessions, before (left) and after (right) learning. Larger graphs depict the reaching motion for [0s-2.8s] and smaller insets the closing of the gripper [2.8s-4s]. Before and after learning, the fingers of the hand close symmetrically. But after learning, the tip of the left finger moves less than the right finger (see arrows). This is because the hand is pivoting around a point near the left finger (see circle with arrow). In essence, the hand is wrapping around the object, thus avoiding collisions. The S-curve motion during reaching is seemingly a preparation for this pivoting. This leads to a higher success rate, and thus is an adaptation to the uncertainty in the object position.

D. Parameter Selection for PI^2

Before turning to the next application, we now present some guidelines for setting the open parameters of PI^2. In general, as we and others have noticed: *"PI^2 shape learning is stable and robust within large parameter regimes"* [26].

a) Number of trials per update K: In general, increasing K leads to parameter updates that are more robust to system noise, but also leads to slower convergence. We recommend $K \geq 5$, where $K = 10$ has proven to be sufficient for almost all tasks we have considered so far[2]. In the experiments in this article, we have reused low-cost trials by copying them from the set of trials gathered for the previous update. This has the disadvantage that certain trials might persist over several updates, and may cause parameter sets that do not conform to the sampling distribution. Based on our current experience, we would rather recommend importance mixing [30], as it does not have these disadvantages.

b) Exploration noise Σ: In principle, higher exploration noise at the beginning of learning is better because it leads to quicker convergence. In practice, the magnitude of exploration is usually constrained by safety issues: i.e. too vigorous exploration might cause the robot to collide with the environment, go beyond joint limits, reach for objects outside of the

[2]The grasping under uncertainty experiment is an exception, as executions of the same primitive will lead to very different costs, due to the different positions of the object for each attempt. Therefore, the task leads to an inherent necessity for the robot to gather statistics about the performance of a particular policy parameterization, which requires multiple executions of the same primitive. As described in IV.B.1, the robot thus executes the same motion primitive 5 times, once for each object position. Executing 4 parameter sets 5 times leads to $K = 20$ trials per update.

workspace, etc. For real robots, we recommend to first execute a few trials with low exploration, and increase exploration magnitude until a certain safety margin is reached. Usually 20 trials before learning suffice to determine a safe initial exploration noise level Σ_{init}. For a particular robot, the same Σ_{init} can often be used over a variety of tasks. Once a task has been learned, it is better to have low exploration noise, i.e. to exploit the optimized parameters. To this end, exploration decays during learning: $\Sigma = \gamma^u \Sigma_{init}$ with $0 \ll \gamma < 1$, and u the number of updates so far. The value γ takes depends on how many updates are needed to learn the task. Our default value for γ is 0.9, which means that exploration is reduced to $0.12\Sigma_{init}$ after 20 updates. This parameter is not crucial to convergence, and can be set to 0.99 to make sure the algorithm still explores after even after hundreds of updates. In recent work [25], we have shown that covariance matrix adaptation, as in the Cross-Entropy Method [18], enables PI^2 to determine this parameter automatically and on-line during learning.

c) Initial policy parameters $\boldsymbol{\theta}^{init}$: A disadvantage of direct policy methods, and thus PI^2, is that they are local methods, i.e. convergence to the global optimum cannot be guaranteed. This appears to be a general trade-off between global optimality and tractability. A good initialization of the policy parameters is therefore often required to converge to a solution. In robotics, imitation is a common and intuitive approach to providing an initialization, and we hence use it in this article as well.

d) Cost function $J(\boldsymbol{\tau})$: The cost function expresses the goal of the task, and is therefore clearly task-specific. PI^2 simplifies cost function design, as cost functions must not be quadratic, and may be discontinuous. Given this freedom, cost function design proceeds in two phases: 1) determine the relevant cost components, e.g. success of lifting, acceleration, etc. 2) determine the scaling factors between them. This relative scaling allows us to express the priorities of optimizing the different components. A practical approach here is to execute the initial motion primitive (or more efficiently: reuse the 20 trials that were recorded to determine the appropriate amount of exploration noise), and separately record the different cost components *without* scaling. Knowledge of the magnitudes of the cost components allows us to set the scale factor to express our priorities. In the grasping under uncertainty experiments for instance, minimizing accelerations is only a secondary objective; the main task is to successfully lift the object. Therefore, the scale factor 10^{-9} in Eq. 23 was chosen such that the accelerations initially do not constitute more than 5% of the total cost; 95% is due to not lifting the object.

V. Application II – Pick-and-place

We now go from goal learning in one motion primitive to direct reinforcement learning of subgoals in sequences of motion primitives with PI^{2}SEQ. In this section, we apply PI^{2}SEQ 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 shelf, and place it on another shelf in the same cupboard [22]. Exactly this task was recently used in experimental psychology [20], where humans are asked to grasp a bottle, and then place it on a shelf, as depicted in Fig. 3. Interestingly enough, humans grasp the bottle significantly lower if they are required to place it on a higher shelf, and vice versa [20]. Grasping the bottle in this way makes the second motion – transporting the bottle to the next shelf and placing it there – easier to perform due to the *end-state comfort effect* [4], i.e. the muscles need to be extended less to place the object at its final position. This task clearly shows that motion parameters (where do I grasp the object?) are influenced by subsequent motions (transporting it to another shelf), 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-shelf 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 shelf, and the robot is required to reach for the object, grasp it, transport it to a lower or higher shelf, and release the object. This object was chosen as it is quite easy to grasp; grasp planning for complex objects is not the focus of this experiment.

A. Initialization of the Motion Primitive

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 shelf 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. 12. This trajectory is acquired through kinesthetic teaching, and supervised learning is used to acquire the policy parameters $\boldsymbol{\theta}^{init}$ that reproduce this trajectory [10].

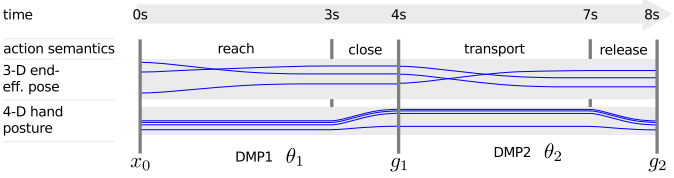

Fig. 12. Grasp procedure used for the pick-and-place task.

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 shelf that is closest to a point on the object[3]. If $||\mathbf{p}|| > 0.15m$ then there is no obstacle avoidance. Otherwise an acceleration away from the shelf 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. 13 by a

[3]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 from the closest point in the point cloud

red gradient[4]. $\ddot{x}_{avoid}$ is added to the transformation system (see Eq. 25) as a coupling term, as proposed in [6]. This causes the end-effector, and thus the object it is transporting, to smoothly move away from the shelf.

$$\frac{1}{\tau}\ddot{x}_t \quad = \quad \alpha(\beta(g - x_t) - \dot{x}_t) + \mathbf{g}_t^\intercal \boldsymbol{\theta} \quad + \quad \ddot{x}_{avoid} \qquad (25)$$

B. Formulation as a Reinforcement Learning Problem

The cost function for this task is

$$J(\boldsymbol{\tau}_i) = \phi_{t_N} + \frac{1}{N}\int_{t_i}^{t_N} (\ 4G(t) + |\ddot{x}_{avoid}| + |\ddot{x}|/250\ )\ dt \qquad (26)$$

where the immediate costs consist three components: 1) $G(t)$, 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); 2) $|\ddot{x}_{avoid}|$ is the acceleration due to the obstacle avoidance module, as we don't want to come so close to the shelf such that obstacle avoidance needs to be active. Thus instead of penalizing collisions between the object and the shelf (which might damage the robot or cupboard), the obstacle avoidance ensures that the task succeeds, and rather penalize the use of obstacle avoidance as a proxy for collisions; 3) $|\ddot{x}|/250$ 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 shelf 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 parameters for the PI^2 learning algorithm are set as follows. The number of trials per update is K=5. The exploration noise for the end-effector is $\boldsymbol{\Sigma}^{\boldsymbol{\theta}} = 1.0\gamma^u$ 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.

C. Empirical Evaluation

Our evaluation consists of two parts. In Section V-C1, we compare the learned behavior for transporting the object up or down to a higher/lower shelf, thus showing that the robot learns a motion with the same features as humans do. In Section V-C2, we compare the different learning strategies, i.e. only shape, only goal, greedy goal and shape, and PI^{2}SEQ.

1) Comparison of Upward and Downward Reaching: In simulation, the robot performed five learning sessions with PI^{2}SEQ for both placing the object on a higher and lower shelf. On the real robot, we performed one learning session for transporting the object up to the higher shelf.

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

The end-effector paths before and after learning are depicted in Fig. 13. The learning curves, i.e. the total cost of the noise-free evaluation trials as learning progresses, for moving the object up and down are depicted in the lower graph in Fig. 13. After 60 trials, costs have converged towards the same costs for all learning sessions, as the variance is very low. The residual costs after 60 trials are all due to the end-effector acceleration ($|\ddot{x}|/250$); a certain amount of acceleration will always be required to perform the motion.

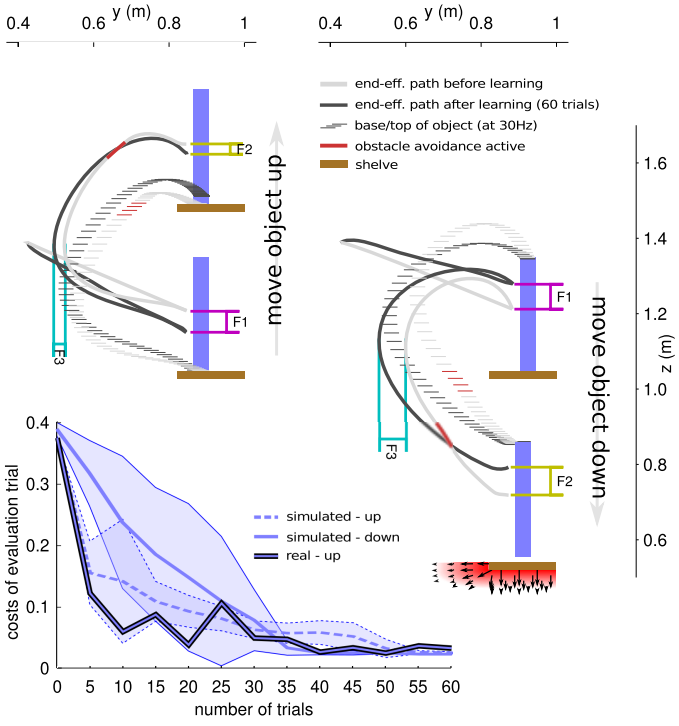

Fig. 13. For the simulation experiments, the end-effector paths for moving the object up (left) or down (right) both before (light gray) and after (dark gray) learning. Red segments of the path indicate that obstacle avoidance is active. The thin vertical lines (left) represent the base position of the object at 30Hz (cf. Fig. 3 for clarity). For moving down (right) it represents the top of the object. 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. 13: 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, also relative to its initial value. The values of these variables as learning progresses is depicted in Fig. 14.

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 shelf when transporting it to the higher shelf (see Fig. 13, 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

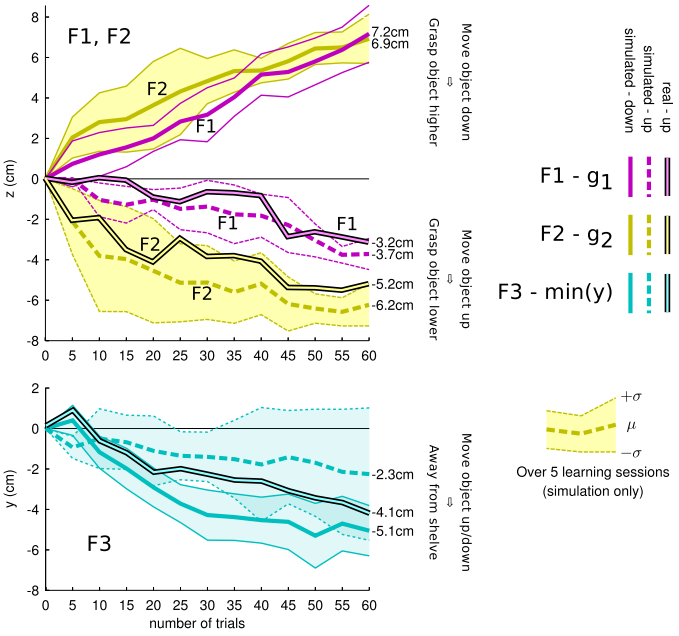

Fig. 14. Values of three features of the motion as learning progresses. Simulation: $\mu \pm \sigma$ over five learning sessions, for transporting the object both up and down. Real robot: one learning session for transporting up.

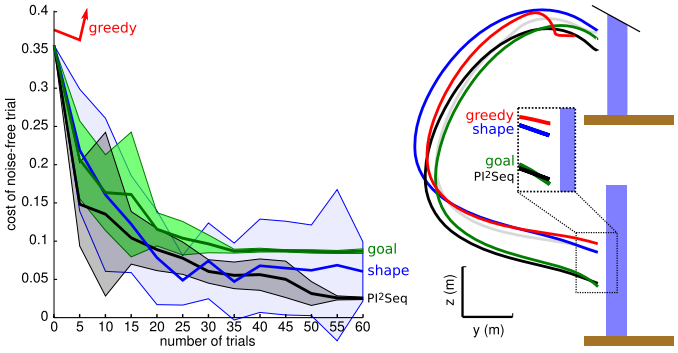

Fig. 15. Left: Learning curves ($\mu \pm \sigma$ over five learning sessions) for each of the learning strategies. Note that the greedy strategy soon jumps to a cost of over 1.0, and does not achieve a lower value afterwards. Hence, only its first two values are plotted. Right: End-effector paths for the different strategies after 60 trials, 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.

ϕ_N. This cost is reduced by simultaneously decreasing F2[5], as seen in Fig. 14. 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 increases. Independent of whether the object is moved up or down, F3 decreases over time, i.e. the distance to the shelf 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, PI^{2}SEQ 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 [20].

2) Comparison of Learning Strategies: We now compare the four learning strategies in simulation for the upward transport motion. We perform five learning session for each of the strategies: 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. PI^{2}SEQ. The learning curves and resulting end-effector paths are depicted in Fig. 15.

We see that PI^{2}SEQ outperforms the other methods in terms of convergence speed and final cost after learning. After convergence of the costs at 60 trials, 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. 15, the goal is not changed when only learning shape, and coincides with the goal of the initial movement before learning. When learning the goal or using PI^{2}SEQ, 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 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.

[5]The cost due to not grasping an object $G(t)$ appears not to play a role in these results. The reason is that any trial 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.

VI. Conclusion

In this article, we propose an update rule for learning motion primitive goals, based on the model-free reinforcement learning algorithm PI^2. Furthermore, we apply simultaneous learning of goal and shape at different levels of temporal abstraction in sequences of motion primitives with PI^{2}SEQ. This approach constitutes a new form of reinforcement learning with dynamical system equations as options, which makes it particularly well-suited for high-dimensional robotics problems.

We apply these theoretical contributions to two tasks – grasping under object position uncertainty and a pick-and-place task – to improve the robustness of object manipulation skills. We believe a key aspect of achieving more robust behavior is to exploit the experience a robot gathers during its operation; observed experience is the most important resource a robot has for determining the actual effects of its actions on the environment. Our reinforcement learning approach leverages this experience to improve future behavior, without requiring a model.

Our methods for learning robust motion primitives may well be combined with methods that actively reduce uncertainty [8], or use reactive control based on sensor feedback [15], [9]. In fact, we believe that learning motions that have an intrinsic capability to deal with uncertainty is one of many approaches,

albeit an important one, which are necessary to achieve truly robust robotic manipulation.

Appendix: Robot Platform

The robotic platform used in this article consists of a 7-DOF Barrett WAM arm with a three-fingered 4-DOF Barrett BH280 hand. Low-level control and physical simulations of the robot are done with the SL software package [19], and high-level communications with the Robot Operating System [17]. 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 with the Xenomai real-time operating system. We thank Ludovic Righetti, Mrinal Kalakrishnan, and Peter Pastor for their assistance.

Acknowledgements

We thank the reviewers for their constructive suggestions for improvement of the article. We thank the authors of [3] and [20] for their permission to use images of their experimental set-up, and the members of the Brain Body Dynamics Lab for emptying their cookie cupboard for the experimental set-up in Section V. 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, the ATR Computational Neuroscience Laboratories, and the MACSi project ANR-2010-BLAN-0216-03. F.S. was supported by a Research Fellowship and a Return Grant from the German Research Foundation (DFG). E.T. was supported by a Myronis Fellowship.

REFERENCES

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

[2] D. Berenson, S. Srinivasa, and J. Kuffner. Addressing pose uncertainty in manipulation planning using task space regions. In *Proc. of the IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS)*, 2009.

[3] V. Christopoulos and P. Schrater. Grasping objects with environmentally induced position uncertainty. *PLOS Computational Biology*, 5(10), 2009.

[4] R. G. Cohen and D. A. Rosenbaum. Where grasps are made reveals how grasps are planned: generation and recall of motor plans. *Exp. Brain Research*, 157(4):486–495, 2004.

[5] J. Flanagan, M. Bowman, and R. Johansson. Control strategies in object manipulation tasks. *Current Opinion in Neurobiol.*, 16:650–659, 2006.

[6] H. Hoffmann, P. Pastor, D. Park, and S. Schaal. Biologically-inspired dynamical systems for movement generation: Automatic real-time goal adaptation and obstacle avoidance. In *Proceedings of ICRA*, 2009.

[7] I. Horswill. *Specialization of perceptual processes*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1993.

[8] K. Hsiao, L. Kaelbling, and T. Lozano-Perez. Task-driven tactile exploration. In *Proceedings of Robotics: Science and Systems*, 2010.

[9] K. Hsiao, S. Chitta, M. Ciocarlie, and E. Jones. Contact-reactive grasping of objects with partial shape information. In *Proc. of the IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS)*, 2010.

[10] 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.

[11] J. Kober, E. Oztop, and J. Peters. Reinforcement learning to adjust robot movements to new situations. In *Proceedings of Robotics: Science and Systems*, 2010.

[12] O. Kroemer, R. Detry, J. Piater, and J. Peters. Combining active learning and reactive control for robot grasping. *Robotics and Autonomous Systems*, 58(9):1105–1116, 2010.

[13] A. Morales, E. Chinellato, A. H. Fagg, and A. P. del Pobil. Using experience for assessing grasp reliability. *International Journal of Humanoid Robotics*, 1(4):671–691, 2004.

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

[15] P. Pastor, L. Righetti, M. Kalakrishnan, and S. Schaal. Online movement adaptation based on previous sensor experiences. In *Proceedings of IROS*, San Francisco, USA, 2011. Best paper award.

[16] J. Peters, K. Mülling, J. Kober, D. Nguyen-Tuong, and O. Krömer. Towards motor skill learning for robotics. In *Proceedings of Robotics Research - The 14th International Symposium (ISRR)*, 2009.

[17] M. Quigley, K. Conley, B.P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A.Y. Ng. ROS: an open-source Robot Operating System In *ICRA Workshop on Open Source Software*, 2009.

[18] 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.

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

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

[21] 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.

[22] F. Stulp and S. Schaal. Hierarchical reinforcement learning with motion primitives. In *11th IEEE-RAS Int'l Conf. on Humanoid Robots*, 2011.

[23] F. Stulp, E. Theodorou, J. Buchli, and S. Schaal. Learning to grasp under uncertainty. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[24] F. Stulp, E. Theodorou, M. Kalakrishnan, P. Pastor, L. Righetti, and S. Schaal. Learning motion primitive goals for robust manipulation. In *Proc. of Int'l Conf. on Intelligent Robots and Systems (IROS)*, 2011.

[25] F. Stulp and O. Sigaud. Path Integral Policy Improvement with Covariance Matrix Adaptation. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*, 2012.

[26] M. Tamosiumaite, B. Nemec, A. Ude, and F. Wörgötter. Learning to pour with a robot arm combining goal and shape learning for dynamic movement primitives. *Robots and Autonomous Systems*, 59(11):910–922, 2011.

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

[28] E. Theodorou. *Iterative Path Integral Stochastic Optimal Control: Theory and Applications to Motor Control*. PhD thesis, University of Southern California, 2011.

[29] 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.

[30] S. Yi, D. Wierstra, T. Schaul, and J. Schmidhuber. Stochastic search using the natural gradient. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1161–1168, 2009.

Freek Stulp Dr. Freek Stulp's research interests include robotics, reinforcement learning, motion primitives, and developmental principles for robot learning. His main application domain is autonomous manipulation in human environments. He has a doctorandus degree (M.Sc. equivalent) in Cognitive Science and Engineering from the University of Groningen, and received his doctorate degree in Computer Science from the Technische Universität München in 2007. He was awarded post-doctoral research fellowships from the Japanese Society for the Promotion of Science and the German Research Foundation (DFG), to pursue his research at the Advanced Telecommunications Research Institute Int'l (Kyoto) and the University of Southern California (Los Angeles). He is now an assistant professor at the École Nationale Supérieure de Techniques Avancées (ENSTA-ParisTech) in Paris, and a member of the FLOWERS team at INRIA Bordeaux.

Evangelos A. Theodorou Evangelos A. Theodorou earned his Diploma (M.Sc. equivalent) in Electrical and Computer Engineering and an M.Sc. in Production Engineering from the Technical University of Crete in 2001 and 2004. In 2007 he earned a M.Sc. in Computer Science and Engineering from the University of Minnesota, USA and in 2010 and 2011 he received a M.Sc. in Electrical Engineering and PhD in Computer Science, from the Viterbi school of Engineering at USC. He has been a recipient of a Myronis Fellowship for engineering graduate students at USC. Since 2011 he has been holding a postdoctoral research associate position with the Computer Science and Engineering department, University of Washington, Seattle, USA. His research interest span over the areas of control, estimation and machine learning theory with focus on Stochastic Optimal Control, Stochastic Estimation, Reinforcement Learning, and applications to Robotics and Computational Neuroscience.

Stefan Schaal Stefan Schaal is Professor of Computer Science, Neuroscience, and Biomedical Engineering at the University of Southern California, and a Founding Director of the Max-Planck-Insitute for Intelligent Systems in Tübingen, Germany. He is also an Invited Researcher at the ATR Computational Neuroscience Laboratory in Japan, where he held an appointment as Head of the Computational Learning Group during an international ERATO project, the Kawato Dynamic Brain Project (ERATO/JST). Dr. Schaal's research interests include topics of statistical and machine learning, neural networks, computational neuroscience, functional brain imaging, nonlinear dynamics, nonlinear control theory, and biomimetic robotics. He applies his research to problems of artificial and biological motor control and motor learning, focusing on both theoretical investigations and experiments with human subjects and anthropomorphic robot equipment.