Learning Parameterized Motor Skills on a Humanoid Robot

Bruno Castro da Silva[1], Gianluca Baldassarre[2], George Konidaris[3] and Andrew Barto[1]

Abstract— **We demonstrate a sample-efficient method for constructing reusable parameterized skills that can solve families of related motor tasks. Our method uses learned policies to analyze the policy space topology and learn a set of regression models which, given a novel task, appropriately parameterizes an underlying low-level controller. By identifying the disjoint charts that compose the policy manifold, the method can separately model the qualitatively different sub-skills required for solving distinct classes of tasks. Such sub-skills are useful because they can be treated as new discrete, specialized actions by higher-level planning processes. We also propose a method for reusing seemingly unsuccessful policies as additional, valid training samples for synthesizing the skill, thus accelerating learning. We evaluate our method on a humanoid iCub robot tasked with learning to accurately throw plastic balls at parameterized target locations.**

I. Introduction

Flexible skills are one of the fundamental building blocks of truly autonomous robots. When solving control problems, single policies can be learned but may fail if the tasks at hand vary or if the agent has to face novel, unknown contexts. Furthermore, learning a single policy for each possible variation of a task or context is infeasible. To deal with these problems we extend previous research and demonstrate a new sample-efficient method capable of constructing reusable, parameterized skills on a physical robot. Parameterized skills are flexible behaviors that can be used to tackle any instance of a family of related control tasks, given only a parameterized description of the task [1], [2]. Once such a skill is learned it can be applied to novel variations of a task without having to learn context-specific policies from scratch. Parameterized skills are also useful for dealing with high-dimensional control problems since they can be treated as adjustable primitive actions by higher-level planning processes, thus abstracting away details of low-level control.

As an example of the type of parameterized skill we refer to, consider a robot that has to move different objects in a warehouse. While executing such a task, the robot may have to pick up objects of different shapes and weights. For it to be truly competent it should be able to grasp different objects, even ones with a particular combination of shape and weight that has never been experienced before. In this case, learning a single policy for each possible combination of object properties is infeasible. The robot might, instead, wish to learn good policies for a few specific types of objects and use these to synthesize a single general grasping skill—parameterized by the characteristics of the target object—that it can execute on-demand.

Previous research has shown that it is possible to transfer information between pairs of related tasks [3], [4] and that behaviors encoded via fixed, pre-defined families of policies can be learned and used to tackle slight variations of a task [5], [2]. More general methods for synthesizing parameterized skills have been recently proposed; Kober et al. [2] propose learning a mapping from task description to metaparameters of a motion primitive, but assume that a single underlying motion primitive is sufficient to represent the whole class of tasks of interest. Neumann et al. [6] modify a policy search algorithm to deal with parameterized tasks, but require learning a forward-model prior to acquiring the skill. da Silva et al [1] propose a general framework for learning parameterized skills that estimates the topology of the policy space and models the set of disjoint low-dimensional charts that compose the policy manifold. These models are then combined to construct a single parameterized skill function that predicts policy parameters from task parameters.

In this paper we extend the framework proposed by da Silva et al. [1] so that *1)* it is more *sample-efficient, and therefore applicable to robotics problems*; *2)* it can *reuse seemingly unsuccessful policies as additional, valid training samples for synthesizing the skill*, thus accelerating learning via a type of off-policy parameterized skill learning. Furthermore, we demonstrate that *1)* our method is capable of constructing a *generalizable whole-body throwing skill, in a physical humanoid robot, from few training samples*; and that *2)* it automatically *identifies and separately models the qualitatively different sub-skills* required for solving different types of tasks. This latter capability is particularly important when constructing hierarchically structured policies, since each sub-skill can be labeled as a new discrete, specialized adjustable action for use by higher-level planning processes.

II. Setting

We assume an agent which is presented with a set of tasks drawn from some task distribution. Each task is modeled as a Markov Decision Process (MDP). Furthermore, we assume that the MDPs have dynamics and reward functions similar enough so that they can be considered variations of a same task. Our objective is to maximize the expected reward over the distribution of possible MDPs:

[1]B. da Silva and A. Barto are with the School of Computer Science, University of Massachusetts Amherst, 01003, USA. {bsilva,barto}@cs.umass.edu

[2]G. Baldassarre is with the Ist. di Scienze e Tecnologie della Cognizione, CNR, Rome, Italy. gianluca.baldassarre@istc.cnr.it

[3]G. Konidaris is with the Computer Science and Artificial Intelligence Lab (CSAIL), MIT, Cambridge, MA 02139, USA. gdk@csail.mit.edu

This work was partially funded by the EU project IM-CLeVeR, G.A. No. FP7-ICT-IP-231722.

$$\int P(\tau)J\big(\pi_\theta,\tau\big)d\tau, \tag{1}$$

where π_θ is a policy parameterized by $\theta \in \mathbb{R}^N$, τ is a task parameter vector drawn from a $|T|$-dimensional continuous space T, $J(\pi,\tau) = E\big\{\sum_{t=0}^{t_f} r_t|\pi,\tau\big\}$ is the expected return obtained when using policy π to solve task τ during an episode of length t_f, and $P(\tau)$ is a probability density function describing the probability of task τ occurring. Let a *parameterized skill* be a function

$$\Theta : T \to \mathbb{R}^N, \tag{2}$$

mapping task parameters to policy parameters. When using a parameterized skill to solve a distribution of tasks, the specific policy parameters to be used depend on the task currently being solved and are specified by Θ. Our goal is to construct a parameterized skill function Θ which maximizes:

$$\int P(\tau)J\big(\pi_{\boldsymbol{\Theta}(\tau)},\tau\big)d\tau. \tag{3}$$

Note that policies for solving tasks in T are points in an N-dimensional policy space. Because these tasks are related, it is reasonable to assume that there exists some structure in the policy space; more specifically, that policies for solving related tasks lie on *a lower-dimensional manifold* embedded in $\mathbb{R}^N$. This manifold may be composed of a set of piecewise-smooth surfaces, or charts, which together span the space of task policies. Within each chart, policy parameters vary smoothly as we vary the task parameters. Disjoint charts may exist in policy space because changes in task parameters might require abrupt changes in the parameterization of the policy. Since policies may be distributed over separate charts, it is important to automatically identify and separately model each one of the charts. Each disjoint chart typically encodes a specialized *sub-skill* required for solving a specific subclass of tasks. A parameterized skill should be able to identify and model these specialized strategies and to correctly select which one is appropriate for a task, thus obtaining a unified model by which different sub-skills are integrated.

III. Learning a Parameterized Skill

To learn a parameterized skill we first sample $|K|$ tasks from P, the task distribution, and use them to construct K, the set of training task instances and their corresponding learned policy parameters. More specifically, let $K \equiv \{(\tau_i,\theta_i)\}$, where τ_i is the i-th training task drawn from P and $\theta_i = [\theta_i(1),\ldots,\theta_i(N)]^\top \in \mathbb{R}^N$ are the parameters of a policy that solves task τ_i. We wish to use the set of sample tasks and corresponding policies to train a family of regression models mapping task parameters to policy parameters. However, because policies for different subsets of T might be embedded in different lower-dimensional charts of the policy manifold, it is necessary to first estimate the number D of such charts, and only then train a separate regression model for each one. We use ISOMAP [7] to analyze different properties of the geometry and topology of the policy space. ISOMAP is a method for identifying the underlying global geometry of high-dimensional spaces and for estimating the intrinsic number of non-linear degrees of freedom that underlie it. This latter information can be used to *1)* estimate D, the number of disjoint lower-dimensional charts in the manifold; and *2)* in which chart c the policy for a given task τ is embedded. We organize this information into a training set of pairs (τ,c), where τ is a training task and $c \in \{1,\ldots,D\}$ is an identifier of the chart in which that task's policy is embedded. Let us use this training set to learn a classifier χ mapping *task parameters to the identifier of the chart in which that task's policy is embedded*: $\chi : T \to \{1,\ldots,D\}$. In this paper, χ is learned using nonlinear regularized logistic regression and its parameters are optimized using standard gradient descent methods.

Let Θ be the parameterized skill function we wish to learn. This function will be defined in terms of a set of functions $\Theta_c : T \to \mathbb{R}^N$, each one modeling a distinct chart. More specifically, each Θ_c models how policy parameters change as task parameters change, and is defined over all tasks whose policies are embedded in chart c. Let K_c be the set of all tasks which χ assigns to chart c: $K_c \equiv \{(\tau,\theta) \in K\}$ s.t. $\chi(\tau) = c$. We use each K_c as a training set to learn the corresponding function Θ_c. Each function Θ_c is learned as a ℓ^2-regularized linear regression model mapping nonlinear task features to policy parameters. We can now define the overall parameterized skill as a function

$$\Theta(\tau) \equiv \Theta_c(\tau) \text{ s.t. } c = \chi(\tau) \tag{4}$$

and

$$\Theta_c(\tau) = [(\boldsymbol{w}^1_{K_c})^\top \ldots (\boldsymbol{w}^N_{K_c})^\top]\boldsymbol{\varphi}_{K_c}(\tau), \tag{5}$$

where $\boldsymbol{\varphi}_{K_c}(\tau)$ is an arbitrary V-dimensional vector of nonlinear task features computed over τ, and $\boldsymbol{w}^j_{K_c}$, for each $j \in \{1,\ldots,N\}$, is a V-dimensional vector given by

$$\boldsymbol{w}^j_{K_c} = (\Phi^\top_{K_c}\Phi_{K_c} + \lambda_{K_c}I)^{-1}\Phi^\top_{K_c}\boldsymbol{\Pi_j}. \tag{6}$$

Here, Φ_{K_c} is a $|K_c| \times V$ design matrix $[\boldsymbol{\varphi}_{K_c}(\tau_1)^\top \ldots \boldsymbol{\varphi}_K(\tau_{K_c})^\top]$, λ_{K_c} is a regularization term and $\boldsymbol{\Pi_j}$ is a $|K_c|$-dimensional vector containing the j-th policy feature of each one of the $|K_c|$ training policies θ_i: $\boldsymbol{\Pi_j} = [\theta_1(j)\ldots\theta_{K_c}(j)]^\top$. Note that Equation 5 corresponds to a multivariate regression model which implicitly assumes that policy features are approximately independent conditioned on the task. If this is known not to be the case, more expressive multivariate regression models, which directly encode dependencies between task and policy features, could be used.

Figure 1 depicts the overall process of executing a learned parameterized skill: first, a task τ is drawn from P; the classifier χ identifies the chart c in which that task's policy is most likely embedded; finally, the regression model Θ_c for that chart is selected and maps the task parameters of τ to policy parameters $\theta_1,\ldots,\theta_N$.

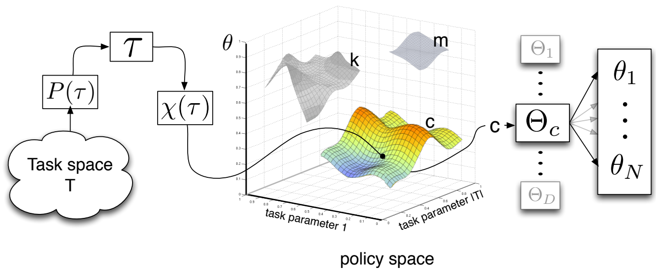

Fig. 1. Steps involved in executing a learned parameterized skill: a task is drawn; χ identifies the chart c to which the policy for that task belongs; the corresponding model Θ_c maps task parameters to policy parameters.

IV. The iCub Throwing Domain

We evaluate the method described in Section III on an iCub robot tasked with learning to accurately throw plastic balls at parameterized target locations. The iCub is a humanoid robot with 53 actuated degrees of freedom built to have dimensions similar to that of a 3.5 year old child [8]. Our goal with these experiments is twofold: *1)* to test the effectiveness of a learned parameterized throwing skill to accurately hit various targets of interest; and *2)* to evaluate whether the learning process can automatically identify and separately model the qualitatively different sub-skills that may be required for solving different classes of tasks[1].

We place a 90cm $\times$ 90cm target board on the floor in front of the robot and allow targets (plastic bottles) to be placed anywhere on that board. The space T of tasks consists of a 2-dimensional Euclidean space containing all (x, y) coordinates at which targets can be placed. The performance of a policy (throw) corresponds to the distance between where the ball landed and the intended target.

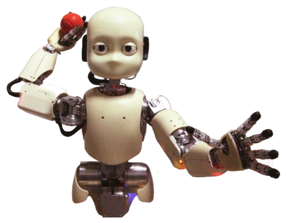

Fig. 2. The iCub preparing for a throw.

Prior to acquiring a parameterized throwing skill we recorded the parameters of a *whole-body overhand throw* from kinesthetic demonstration. This throwing movement required simultaneously rotating the robot's shoulder, torso and forearm, pitching its torso and extending its elbow. Based on this recorded motion we defined a base policy for throwing whose parameters can be modified so that it hits different target positions. This base policy is a function of a 7-dimensional vector $[\theta_1, \ldots, \theta_N]^\top$ whose elements modify different aspects of the throw. Concretely, they regulate the initial and final torso angles, torso rotation speed, torso pitch at the end of the movement, initial angle of rotation of the shoulder, amount of forearm rotation during the throw and total elbow extension. A throw initiates when the robot executes a predefined open-loop grasping motion to hold the plastic ball; it also involves commanding the iCub's fingers to release the ball at a fixed time during the movement. A sample throw is shown on Figure 3.

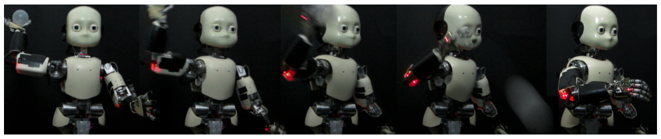

Fig. 3. A sample throw executed by the iCub.

Before we can apply the parameterized skill framework to this problem we need to make two domain-dependent design decisions: *1)* what policy search algorithm to use when constructing the training set K; and *2)* what expanded set of task features $\varphi_{K_c}(\tau)$ to use when modeling the surface of each individual chart. The former decision influences the time to construct the skill training set while the latter affects the expressiveness of each chart model Θ_c. We used the PI^2 algorithm [9] to learn individual policies. PI^2 is policy search method derived from first principles of stochastic optimal control; we chose it because it often outperforms standard gradient methods [10] and also due to its simplicity and low dependence on parameter-tuning. The stopping criteria for the search executed by PI^2 corresponds to when it finds a policy capable of hitting a target 4 out of 5 times. This requirement is needed to rule out noisy policy candidates. Finally, we constructed the expanded set of non-linear task features $\varphi_{K_c}(\tau)$ as a set of polynomial features over the original task representation. Since τ corresponds to the cartesian coordinates (x, y) of a given target, we define

$$\varphi_{K_c}(\tau) \equiv \varphi_{K_c}\big((x, y)\big) = [x^{e_1} y^{e_2} r^{e_3} \alpha^{e_4}]^\top, \qquad (7)$$

where $(r, \alpha) \equiv \big((x^2 + y^2)^{\frac{1}{2}}, \text{atan2}(y, x)\big)$ is the polar-coordinate representation of (x, y) and each exponent $e_1, \ldots, e_4$ varies in the range $\{0, \ldots, P\}$, where P is the desired degree of the polynomial expansion. We determined by cross-validation that an effective heuristic for choosing P is a logarithmic function of the number of training examples: $P = \lfloor \log_2(|K_c|) \rfloor$. Similarly, we determined that the regularization parameter, λ_{K_c}, can be effectively set by a linear function of the number of training examples: $\lambda_{K_c} = 0.1|K_c|$. This adaptive setting of metaparameters is important to regulate the complexity of each chart model as a function of the number of samples available.

A. Sample Reuse

While practicing tasks to construct K the robot may execute several suboptimal policies. Even though these fail to solve the task at hand they might nonetheless correspond to valid solutions to a *different* task. This is clearly true in the throwing domain: a policy π_θ for hitting some target position τ is suboptimal if it hits a different target τ'; even if not useful for solving τ, the pair (τ', θ) may nonetheless be

[1]Videos are available at the ICRA website or at `http://cs.umass.edu/~bsilva`.

incorporated into K without requiring the agent to explicitly train for τ'. The use of seemingly unsuccessful policies as additional training samples accelerates the construction of K, the most time-intensive step in learning a parameterized skill. In general, reusing samples requires access to a mapping from observed effects of a policy (e.g., its trajectories) to parameters describing the tasks in which the policy could be successfully applied. This mapping can be seen as an inverse model of the parameterized skill; since it allows unsuccessful policies to be reused as sample solutions for different tasks, it constitutes a type of off-policy skill learning.

It is important to understand when the reuse of unsuccessful policies is safe; that is, in which situations the reused policies are guaranteed not to be outliers with respect to a given chart. Some policy search methods, such as those based on gradients, generate candidate policies that, by definition, lie on the manifold of interest. This suggests that using them as additional training samples for the skill is safe. However, some policy representations may be redundant and allow different parameterizations, embedded in distinct charts, to solve a same task. In this case, reusing samples could cause models Θ_c to be trained with incompatible samples originating from different regions of the manifold. We used a simple rejection sampling strategy to ensure that a training set K_c does not mix samples from different charts. Let τ' be the task unintentionally solved by some policy π_θ while in search for a solution for τ. We add the tuple (τ', θ) to K_c, $c = \chi(\tau')$, iff $||\theta - \mathcal{P}_{K_c}(\theta)|| < \epsilon \text{ diam}(K_c)$. Here, $\mathcal{P}_S(p)$ is the projection operator of p onto set S and $\text{diam}(S)$ denotes the diameter of S, both of which are defined with respect to the ℓ^2-norm, and $\epsilon \in [0, 1]$ is a tunable parameter regulating how strict the rejection sampling strategy is. Intuitively, this condition tries to ensure that reused samples added to K_c are never too far away from the surface of a given chart.

V. Experiments

We start our experiments by measuring properties of the policy manifold learned while acquiring the parameterized throwing skill. We sampled 30 tasks (target positions) uniformly at random and learned policies for solving them. Based on these, we used ISOMAP to analyze the topological characteristics of the induced policy manifold generated as we vary tasks. Our first important observation is that the residual variance of the quasi-isometric embedding produced by ISOMAP indicates that the intrinsic dimensionality of the skill manifold is two. This is expected since we are essentially parameterizing a 7-dimensional policy space by task parameters, which are drawn from a 2-dimensional space T. This means that even though policies are 7-dimensional vectors, because there are just two degrees-of-freedom with which we can vary tasks, the policies themselves remain confined to a set of 2-dimensional charts. ISOMAP also detects that policies for different tasks are distributed among *two disjoint charts*. Figure 4 depicts the target board used in this domain and, overlaid on it, colors indicating which regions of the task space are assigned by χ to which chart. As can be seen, policies for targets to the left of the robot are embedded in one of the charts, while policies for targets to the right of the robot lie on a second chart. The learned nonlinear boundary separating these indicates the existence of two disjoint classes of policies, each of which encoding a qualitatively distinct parameterized sub-skill (*throwing to the left; throwing to the right*) needed for solving a specific type of task. The identification of these two classes of policy parameterizations is a demonstration of skill specialization: from a single root policy, learned from demonstration, two parameterized sub-skills are automatically identified and modeled. The need for these differentiated sub-skills partially reflects the dynamics of the robot, as well as physical constraints of its body and motor capabilities. For instance, the requirement that throws be overhand and right-handed implies that qualitatively different torso movements are needed to hit targets on the extreme left field of the robot.

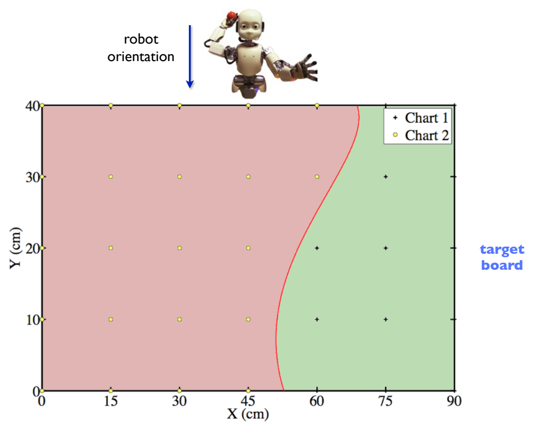

Fig. 4. Target board positioned in front of the robot (not to scale). Different target positions require distinct sub-skills whose policies are embedded in disjoint charts.

Next, we analyze how a chart model allows us to represent the way in which policies vary as we smoothly vary a task. Figure 5 shows a representative subset of policy dimensions varying within a given chart. The vertical axis on each inset figure corresponds to the value of a selected policy feature and the two other axes correspond to task parameters. The first important observation to be made is that as we smoothly vary tasks, the corresponding policy parameters also vary smoothly. Secondly, that even though the relation between task and policy parameters is nonlinear and arguably difficult to be known *a priori*, there exists a clear pattern to be learned—specifically, how policies for related tasks tend to be grouped on a same smooth chart.

We now discuss the performance of the skill learning method. Figure 6 presents the predicted policy parameter error, averaged over 35 novel validation tasks sampled uniformly at random, as a function of the number of examples used to train the skill. This error is a measure of the average distance between the policy parameters predicted by Θ and the parameters of a known good solution for the task; the lower the error, the closer the predicted policy is (in norm) to a correct solution. After approximately 8 samples

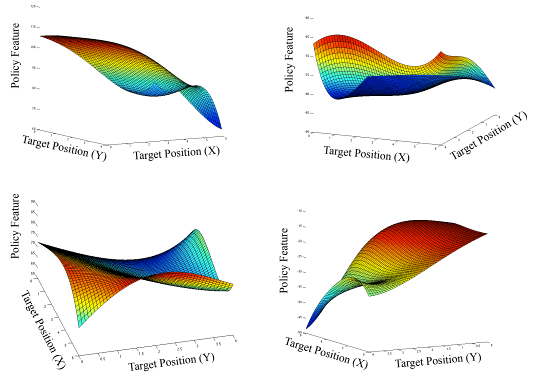

Fig. 5. Examples of lower-dimensional projections of a learned chart Θ_c.

are presented to the parameterized skill the average policy parameter error stabilizes around (but not at) zero. Obtaining this accuracy with few samples is only possible because the policy space is well-structured; specifically, because solutions to similar tasks lie on lower-dimensional charts whose regular topology can be exploited when generalizing known tasks to novel tasks.

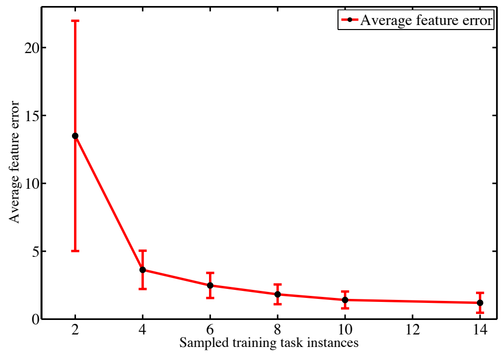

Fig. 6. Average predicted policy parameter error as a function of the number of sampled training tasks.

Note that since policy representations may be sensitive to noise, feature accuracy does not necessarily imply good performance. For this reason we additionally measured the actual performance of predicted policies when applied to a set of novel tasks. We first define a quantity called *minimum performance threshold*, which is the minimum acceptable performance of a predicted policy so that it is considered sufficiently accurate. For this domain we set the performance threshold as the diameter of the plastic bottles used as targets: 6cm. This threshold was selected because throws that miss the intended target position by that amount are still capable, on average, of knocking down the plastic bottle. Figure 7 shows the average distance between where the ball hit and the intended target position as a function of the number of training samples. This distance was measured by directly executing the predicted policy and *before* any further policy improvement took place. Figure 7 also depicts the expected performance of throws if using a nearest-neighbors baseline approach, which corresponds to selecting and executing the policy of the nearest known task. The generalization power of the parameterized skill allows its performance to strictly dominate that of the baseline. It can also be seen that the performance threshold is reached after around 8 samples; at this point, the robot is already capable of hitting target regions of 6cm $\times$ 6cm within the 90cm $\times$ 90cm target board. Policies predicted by the skill can be further improved via standard policy search methods if more accuracy is needed.

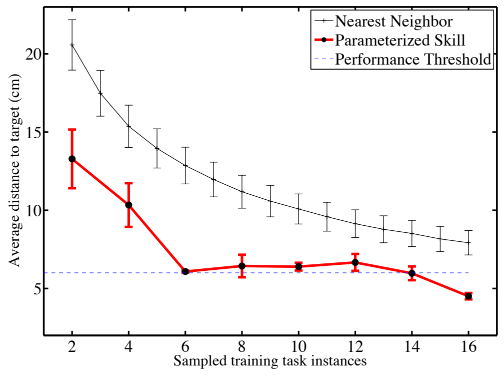

Fig. 7. Average distance to target (before any policy improvement) as a function of the number of sampled training tasks.

The experiments presented in this Section allow us to draw a few important conclusions: *1)* our method can exploit the policy manifold structure to automatically identify and model distinct sub-skills needed to solve different classes of tasks; *2)* even though policy features cannot be perfectly predicted, the skill is nonetheless capable of predicting effective policies with relatively few samples; this is possible because our method explicitly models different charts, which allows for simpler separate regression models; and *3)* a parameterized skill is capable of producing, on-demand and without any further learning, reasonable solutions even when trained with few examples—the reason being that it can quickly identify the overall region in policy space where the solution to a task most likely is.

VI. Related Work

Two baseline approaches are possible for learning a set of related control tasks. The first is to include τ, the task parameter vector, as part of a state descriptor and treat the entire class of tasks as a single MDP. If tasks are related, though, it may be more advantageous to learn them separately and to generalize from the sample experiences [1]. A second approach is to keep a library of sample task solutions and to select and execute the policy of the nearest known task. This does not scale well since the number of samples needed to uniformly cover a task space T grows exponentially with its dimensionality.

Other methods have been proposed that directly learn flexible skills. Soni et al. [11] construct skills whose termination criteria can be adapted on-the-fly to deal with changing aspects of a task. They do not, however, predict a complete parameterization of the policy for novel tasks. Liu et al. [4] propose transferring a value function between pairs of related tasks but, similarly, do not construct a parameterized solution. Taylor et al. [3] transfer high-level policies between tasks but assume that a mapping between features and actions of a source and target tasks is known *a priori*. Hausknecht and Stone [12] present a method for learning a kicking skill parameterized by desired force. They exhaustively test the effect of different policy parameters on the distance traveled by the ball and learn a closed-form, albeit domain-dependent, expression for predicting policy parameters for a given desired force.

Flexible policy representation have been proposed to tackle families of related problems. Schaal et al. [5] introduced Dynamic Movement Primitives (DMPs), a compact policy representation that allows the parameterization of the desired duration and initial and goal states of a trajectory or movement. Even though DMPs allow the parameterization of some aspects of a movement, they are not general enough to encode *arbitrary* parameterized skills since the number and nature of their tunable metaparameters are fixed and predetermined. Furthermore, in some cases it might be unclear how to set the metaparameters of a DMP to solve a given task. To deal with these limitations, Kober et al. [2] propose learning a mapping from task description to DMP metaparameters. They assume that a single underlying motion primitive is sufficient to represent the whole class of tasks of interest and do not model qualitatively different strategies required for solving distinct classes of tasks. Neumann et al. [6] present a similar idea but require learning a forward-model prior to training the skill. Their method also does not identify or model the distinct parameterized sub-skills needed for solving a distribution of tasks. Finally, Stulp et al. [13] propose further extensions to the DMP framework by allowing additional tunable parameters to be learned, but focus on the learning-from-demonstration setting.

VII. Conclusions and Future Work

We have presented a new sample-efficient method for constructing reusable parameterized skills. Our method exploits the structure of the policy manifold and separately models the disjoint, low-dimensional charts in which task solutions are embedded. Each chart encodes a qualitatively different sub-skill required for solving a given class of tasks. Such sub-skills are useful because they can be treated as new discrete, specialized actions by higher-level planning processes. We demonstrate that our method, starting from a single root policy, is capable of identifying and modeling semantically meaningful sub-skills needed to solve a distribution of related tasks. We also introduced a sample reuse scheme which allows seemingly unsuccessful policies to be used as additional, valid training samples for synthesizing a skill. This constitutes a type of off-policy skill learning which can reduce the number of samples needed to acquire the skill. We demonstrate the efficacy of our method by learning a complex whole-body parameterized throwing skill on an iCub with only a small number of samples.

An important question not addressed in our work is that of how to select training tasks. Naïve strategies draw tasks uniformly at random or directly from the target task distribution P. These sampling strategies, however, implicitly assume that all tasks are similarly difficult and that the policy manifold has approximately uniform curvature. *Actively choosing training tasks*, on the other hand, may allow the robot to more quickly uncover unknown parts of the policy manifold, thus improving the overall readiness of the skill with fewer samples. This is particularly important in robotics domains, since exploration is expensive.

Finally, note that reusing samples requires a mapping from observed effects of a policy (e.g., trajectories) to parameters of the tasks in which it could be applied. In some domains, such as the iCub throwing one, identifying these tasks is trivial. In general, though, an *inverse parameterized skill model* is needed—a mapping from policy parameters to tasks where they are applicable. This mapping allows parameterized skills to be efficiently acquired even in domains where the correspondence between policy and tasks spaces is non-trivial. Learning it from data is a challenging problem with possible connections to Inverse Reinforcement Learning [14].

References

[1] B. da Silva, G. Konidaris, and A. Barto, "Learning parameterized skills," in *Proceedings of the Twenty Ninth International Conference on Machine Learning*, June 2012, pp. 1679–1686.

[2] J. Kober, A. Wilhelm, E. Oztop, and J. Peters, "Reinforcement learning to adjust parametrized motor primitives to new situations," *Autonomous Robots*, pp. 1–19, 4 2012.

[3] M. Taylor, G. Kuhlmann, and P. Stone, "Autonomous transfer for reinforcement learning," in *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*, 2008.

[4] Y. Liu and P. Stone, "Value-function-based transfer for reinforcement learning using structure mapping," in *Proceedings to the Twenty-First National Conference on Artificial Intelligence*, 2006, pp. 415–420.

[5] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert, "Learning movement primitives," in *Proceedings of the Eleventh International Symposium on Robotics Research*. Springer, 2004.

[6] G. Neumann, C. Daniel, A. Kupcsik, M. Deisenroth, and J. Peters, "Information-theoretic motor skill learning," in *Proceedings of the AAAI Workshop on Intelligent Robotic Systems*, 2013.

[7] J. Tenenbaum, V. de Silva, and J. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[8] G. Metta, L. Natale, F. Nori, G. Sandini, D. Vernon, L. Fadiga, C. von Hofsten, K. Rosander, M. Lopes, J. Santos-Victor, A. Bernardino, and L. Montesano, "The iCub humanoid robot: An open-systems platform for research in cognitive development," *Neural Networks*, vol. 23, no. 8-9, pp. 1125–1134, 2010.

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

[10] F. Stulp and O. Sigaud, "Path integral policy improvement with covariance matrix adaptation," in *Proceedings of the Twenty-Ninth International Conference on Machine Learning*, 2012.

[11] V. Soni and S. Singh, "Reinforcement learning of hierarchical skills on the Sony Aibo robot," in *Proceedings of the Fifth International Conference on Development and Learning*, 2006.

[12] M. Hausknecht and P. Stone, "Learning powerful kicks on the Aibo ERS-7: The quest for a striker," in *RoboCup-2010: Robot Soccer World Cup XIV*, ser. Lecture Notes in Artificial Intelligence. Springer Verlag, 2011, vol. 6556, pp. 254–65.

[13] F. Stulp, G. Raiola, A. Hoarau, S. Ivaldi, and O. Sigaud, "Learning compact parameterized skills with a single regression," in *Proceedings of the IEEE-RAS International Conference on Humanoid Robots*, 2013, pp. 1–7.

[14] A. Ng and S. Russell, "Algorithms for inverse reinforcement learning," in *Proceedings of the Seventeenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 663–670.