Multi-Task Policy Search for Robotics

Marc Peter Deisenroth[1,2], Peter Englert[3], Jan Peters[2,4], and Dieter Fox[5]

***Abstract*— Learning policies that generalize across multiple tasks is an important and challenging research topic in reinforcement learning and robotics. Training individual policies for every single potential task is often impractical, especially for continuous task variations, requiring more principled approaches to share and transfer knowledge among similar tasks. We present a novel approach for learning a nonlinear feedback policy that generalizes across multiple tasks. The key idea is to define a parametrized policy as a function of both the state *and* the task, which allows learning a single policy that generalizes across multiple known and unknown tasks. Applications of our novel approach to reinforcement and imitation learning in real-robot experiments are shown.**

I. Introduction

Complex robots often violate common modeling assumptions, such as rigid-body dynamics. A typical example is a tendon-driven robot arm, shown in Fig. 1, for which these typical assumption are violated due to elasticities and springs. Therefore, learning controllers is a viable alternative to programming robots. To learn controllers for complex robots, reinforcement learning (RL) is promising due to the generality of the RL paradigm [21]. However, without a good initialization (e.g., by human demonstrations [19], [3]) or specific expert knowledge [4] RL often relies on data-intensive learning methods (e.g., Q-learning). For a fragile robotic system, however, thousands of physical interactions are practically infeasible because of time-consuming experiments and hardware wear.

To make RL practically feasible in robotics, we need to speed up learning by reducing the number of necessary interactions. For this purpose, model-based RL is often more promising than model-free RL, such as Q-learning or TD-learning [5]. In model-based RL, data is used to learn a model of the system. This model is then used for policy evaluation and improvement, reducing the interaction time with the system. However, model-based RL suffers from *model errors* as it typically assumes that the learned model closely resembles the true underlying dynamics [20], [19]. These model errors propagate through to the learned policy, which inherently depends on the quality of the learned model. A principled way of accounting for model errors and

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement #270327, ONR MURI grant N00014-09-1-1052, and the Department of Computing, Imperial College London.

[1]Department of Computing, Imperial College London, UK
[2]Department of Computer Science, TU Darmstadt, Germany
[3]Department of Computer Science, University of Stuttgart, Germany
[4]Max Planck Institute for Intelligent Systems, Germany
[5]Department of Computer Science and Engineering, University of Washington, WA, USA

An extended paper version is available at http://arxiv.org/abs/1307.0813.

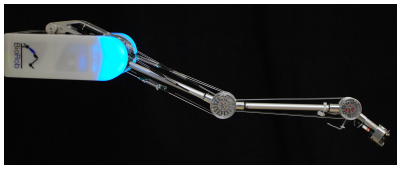


Fig. 1. Tendon-driven BioRob X4.

the resulting optimization bias is to take the uncertainty about the learned model into account for long-term predictions and policy learning [20], [6], [4], [9]. Besides sample-efficient learning, generalizing learned concepts to new situations is a key research topic in RL. Learned controllers often deal with a single situation/context, e.g., they drive the system to a desired state. In a robotics context, solutions for multiple related tasks are often desired, e.g., for grasping multiple objects [17] or in robot games, such as robot table tennis [18] or soccer [7].

We consider a multi-task learning set-up with a continuous set of tasks for which a policy has to be learned. Since it is often impossible to learn individual policies for all conceivable tasks, a multi-task learning approach is required that generalizes across these tasks. Two general approaches exist to tackle this challenge by either hierarchically combining local controllers or a richer policy parametrization.

First, local policies can be learned, and generalization can be achieved by combining them, e.g., by means of a gating network [14]. This approach has been successfully applied in RL [22] and robotics [18]. In [18], a gating network generalizes a set of motor primitives for playing robot-table tennis. The limitation of this approach is that it can only deal with convex combinations of local policies, implicitly requiring local policies that are linear in the policy parameters. In [23], [7], it was proposed to share state-action values across tasks to transfer knowledge. This approach was successfully applied to kicking a ball with a NAO robot in the context of RoboCup. However, a mapping from source to target tasks is explicitly required. In [8], it was proposed to sample a number of tasks from a task distribution, learn the corresponding individual policies, and generalize them to new problems by combining classifiers and nonlinear regression. In [15], [8] mappings from tasks to meta-parameters of a policy were learned to generalize across tasks. The task-specific policies are trained independently, and the elementary movements are given by Dynamic Movement Primitives [13]. Second, instead of a combination of local policies, we can parametrize the policy directly by the task. For instance, in [16], a value function-based transfer learning approach is proposed that generalizes across tasks by finding a regression function mapping a task-augmented

state space to expected returns.

We follow this second approach as it allows for generalizing nonlinear policies: During training, access to a set of tasks is given and a *single* controller is learned jointly for all tasks using policy search. Generalization to new tasks in the same domain is achieved by defining the policy as a function of both the state and the task. At test time, this allows for generalization to unseen tasks *without* retraining. For learning the parameters of the multi-task policy, we use the PILCO policy search framework [9]. PILCO learns flexible Gaussian process (GP) forward models and uses fast deterministic approximate inference for long-term predictions for data-efficient learning. In a robotics context, policy search methods have been successfully applied to many tasks [10] and seem to be more promising than value function-based methods. Hence, this paper addresses two key problems in robotics: multi-task and data-efficient policy learning.

II. Policy Search for Learning Multiple Tasks

We consider dynamical systems $\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t) + \boldsymbol{w}$ with continuous states $\boldsymbol{x} \in \mathbb{R}^D$ and controls $\boldsymbol{u} \in \mathbb{R}^F$ and unknown transition dynamics f, where $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_w)$ is i.i.d. Gaussian noise with covariance $\boldsymbol{\Sigma}_w$. In (single-task) policy search, we seek a deterministic *policy* $\pi : \boldsymbol{x} \mapsto \pi(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{u}$ that minimizes the *expected long-term cost*

$$J^\pi(\boldsymbol{\theta}) = \sum_{t=1}^{T} \mathbb{E}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)], \quad p(\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{\mu}_0^x, \boldsymbol{\Sigma}_0^x), \quad (1)$$

of following π for T steps. Note that the trajectory $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T$ depends on the policy π and, thus, the parameters $\boldsymbol{\theta}$. In (1), $c(\boldsymbol{x}_t)$ is a cost function of state $\boldsymbol{x}$ at time t. The policy π is parametrized by $\boldsymbol{\theta} \in \mathbb{R}^P$. Typically, the cost c incorporates some information about a task $\boldsymbol{\eta}$, e.g., a desired target location $\boldsymbol{x}_{\text{target}}$. A policy that minimizes (1) solves the task $\boldsymbol{\eta}$ of controlling the robot toward the target.

A. Task-Dependent Policies

We propose to learn a *single* policy for all tasks jointly to generalize classical policy search to a multi-task scenario. We assume that the dynamics are stationary with the transition probabilities and control spaces shared by all tasks. By finding a single policy that is sufficiently flexible to learn a set of training tasks $\boldsymbol{\eta}_i^{\text{train}}$, we aim to obtain good generalization performance to related test tasks $\boldsymbol{\eta}_j^{\text{test}}$ by reducing the danger of overfitting to the training tasks, a common problem with current hierarchical approaches.

To learn a single controller for multiple tasks $\boldsymbol{\eta}_k$, we propose to make the policy a function of the state $\boldsymbol{x}$, the parameters $\boldsymbol{\theta}$, *and* the task $\boldsymbol{\eta}$, such that $\boldsymbol{u} = \pi(\boldsymbol{x}, \boldsymbol{\eta}, \boldsymbol{\theta})$. In this way, a trained policy has the potential to generalize to previously unseen tasks by computing different control signals for a fixed state $\boldsymbol{x}$ and parameters $\boldsymbol{\theta}$ but varying tasks $\boldsymbol{\eta}_k$. Fig. 2 gives an intuition of what kind of generalization power we can expect from such a policy: Assume a given policy parametrization, a fixed state, and five training targets $\boldsymbol{\eta}_i^{\text{train}}$. For each pair $(\boldsymbol{x}, \boldsymbol{\eta}_i^{\text{train}}, \boldsymbol{\theta})$, the policy determines the corresponding controls $\pi(\boldsymbol{x}, \boldsymbol{\eta}_i^{\text{train}}, \boldsymbol{\theta})$, which are denoted by the red circles. The differences in these control signals are

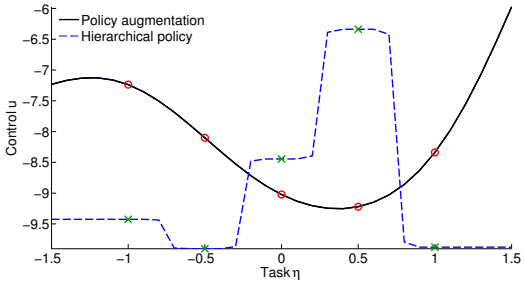


Fig. 2. Two multi-task policies for a given state, but varying task η. The change in the controls is solely due to the change in the task. The black policy is determined by our proposed multi-task approach (MTPS+); the blue, dashed policy is obtained by hierarchically combining local controllers (RW-IC). The controls of the training tasks $u = \pi(\boldsymbol{x}, \eta)$ are marked by the red circles (MTPS+) and the green stars (RW-IC), respectively. MTPS+ generalizes more smoothly across tasks, whereas the hierarchical combination of independently trained local policies does not generalize well.

Fig. 3. Multi-Task Policy Search

1: **init:** Pass in training tasks $\boldsymbol{\eta}_i^{\text{train}}$, initialize policy parameters $\boldsymbol{\theta}$ randomly. Apply random control signals and record data.
2: **repeat**
3: Update GP forward dynamics model using all data
4: **repeat**
5: Long-term predictions: Compute $\mathbb{E}_{\boldsymbol{\eta}}[J^\pi(\boldsymbol{\theta}, \boldsymbol{\eta})]$
6: Analytically compute gradient $\mathbb{E}_{\boldsymbol{\eta}}[dJ^\pi(\boldsymbol{\theta}, \boldsymbol{\eta})/d\boldsymbol{\theta}]$
7: Update policy parameters $\boldsymbol{\theta}$ (e.g., BFGS)
8: **until** convergence; **return** $\boldsymbol{\theta}^*$
9: Set $\pi^* \leftarrow \pi(\boldsymbol{\theta}^*)$
10: Apply π^* to robot and record data
11: **until** $\forall i$: task $\boldsymbol{\eta}_i^{\text{train}}$ learned
12: Apply π^* to test tasks $\boldsymbol{\eta}_j^{\text{test}}$

achieved solely by changing $\boldsymbol{\eta}_i^{\text{train}}$ in $\pi(\boldsymbol{x}, \boldsymbol{\eta}_i^{\text{train}}, \boldsymbol{\theta})$ as $\boldsymbol{x}$ and $\boldsymbol{\theta}$ were assumed fixed. The parametrization of the policy by $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$ determines the generalization power of π to new (but related) tasks $\boldsymbol{\eta}_j^{\text{test}}$ at test time. The policy for a fixed state but varying test tasks $\boldsymbol{\eta}_j^{\text{test}}$ is represented by the black curve. A corresponding policy for a hierarchical combination of local policies is shown in the blue curve. To find good parameters of the multi-task policy, we incorporate our multi-task learning approach into the model-based PILCO policy search framework [9]. The high-level steps of the resulting algorithm are summarized in Fig. 3. We assume that a set of training tasks $\boldsymbol{\eta}_i^{\text{train}}$ is given. The parametrized policy π is initialized randomly, and, subsequently, applied to the robot, see line 1 in Fig. 3. Based on the initial collected data, a probabilistic GP forward model of the robot dynamics is learned (line 3) to consistently account for model errors.

We define the policy π as an explicit function of both the state $\boldsymbol{x}$ and the task $\boldsymbol{\eta}$, i.e., the policy depends on a task-augmented state and $\boldsymbol{u} = \pi(\boldsymbol{x}, \boldsymbol{\eta}, \boldsymbol{\theta})$. Before going into details, let us consider the case where a function g relates state and task. In this paper, we consider two cases: (a) A linear relationship between the task $\boldsymbol{\eta}$ and the state $\boldsymbol{x}_t$ with $g(\boldsymbol{x}_t, \boldsymbol{\eta}) = \boldsymbol{\eta} - \boldsymbol{x}_t$. For example, the state and the task (corresponding to a target location) can be both defined in camera coordinates, and the target location parametrizes

and defines the task. (b) The task variable $\boldsymbol{\eta}$ and the state vector are not directly related, in which case $g(\boldsymbol{x}_t, \boldsymbol{\eta}) = \boldsymbol{\eta}$. For instance, the task variable could simply be an index. We approximate the joint distribution $p(\boldsymbol{x}_t, g(\boldsymbol{x}_t, \boldsymbol{\eta}))$ by

$$\mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_t^x\\ \boldsymbol{\mu}_t^\eta\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}_t^x & \boldsymbol{C}_t^{x\eta}\\ \boldsymbol{C}_t^{\eta x} & \boldsymbol{\Sigma}_t^\eta\end{bmatrix}\right) =: \mathcal{N}\big(\boldsymbol{x}_t^{x,\eta} \,|\, \boldsymbol{\mu}_t^{x,\eta}, \boldsymbol{\Sigma}_t^{x,\eta}\big)\,, \quad (2)$$

where the state distribution is $\mathcal{N}\big(\boldsymbol{x}_t \,|\, \boldsymbol{\mu}_t^x, \boldsymbol{\Sigma}_t^x\big)$ and $\boldsymbol{C}_t^{x\eta}$ is the cross-covariance between the state and $g(\boldsymbol{x}_t, \boldsymbol{\eta})$. The cross-covariances for $g(\boldsymbol{x}_t, \boldsymbol{\eta}) = \boldsymbol{\eta} - \boldsymbol{x}_t$ are $\boldsymbol{C}_t^{x\eta} = -\boldsymbol{\Sigma}_t^x$. If state and task are unrelated, i.e., $g(\boldsymbol{x}_t, \boldsymbol{\eta}) = \boldsymbol{\eta}$, then $\boldsymbol{C}_t^{x\eta} = \boldsymbol{0}$.

The Gaussian approximation of the joint $p(\boldsymbol{x}_t, g(\boldsymbol{x}_t, \boldsymbol{\eta}))$ in (2) serves as the input distribution to the controller function π. Although we assume that the tasks $\boldsymbol{\eta}_i^{\text{test}}$ are given deterministically at test time, introducing a task uncertainty $\boldsymbol{\Sigma}_t^\eta > \boldsymbol{0}$ during *training* can make sense: First, $\boldsymbol{\Sigma}_t^\eta$ defines a *task distribution*, which may allow for better generalization performance compared to $\boldsymbol{\Sigma}_t^\eta = \boldsymbol{0}$. Second, $\boldsymbol{\Sigma}_t^\eta > \boldsymbol{0}$ serves as a regularizer and makes policy overfitting less likely.

B. Multi-Task Policy Evaluation

For policy evaluation, we analytically approximate the expected long-term cost $J^\pi(\boldsymbol{\theta})$ by averaging over all tasks $\boldsymbol{\eta}$, see line 5 in Fig. 3, according to

$$\mathbb{E}_{\boldsymbol{\eta}}[J^\pi(\boldsymbol{\theta}, \boldsymbol{\eta})] \approx \tfrac{1}{M} \sum\nolimits_{i=1}^{M} J^\pi(\boldsymbol{\theta}, \boldsymbol{\eta}_i^{\text{train}})\,, \quad (3)$$

where M is the number of tasks considered during training. The expected cost $J^\pi(\boldsymbol{\theta}, \boldsymbol{\eta}_i^{\text{train}})$ corresponds to (1) for a specific training task $\boldsymbol{\eta}_i^{\text{train}}$. The intuition behind the expected long-term cost in (3) is to allow for learning a single controller for multiple tasks *jointly*. Hence, the controller parameters $\boldsymbol{\theta}$ have to be updated in the context of *all* tasks. The resulting controller is not necessarily optimal for a *single* task, but (neglecting approximations and local minima) optimal across *all tasks* on average, presumably leading to good generalization performance. The expected long-term cost $J^\pi(\boldsymbol{\theta}, \boldsymbol{\eta}_i^{\text{train}})$ in (3) is computed as follows.

First, based on the learned GP dynamics model, approximations to the long-term predictive state distributions $p(\boldsymbol{x}_1|\boldsymbol{\eta}), \ldots, p(\boldsymbol{x}_T|\boldsymbol{\eta})$ are computed analytically: For a joint Gaussian prior $p(\boldsymbol{x}_t, \boldsymbol{u}_t|\boldsymbol{\eta})$, the successor state distribution

$$p(\boldsymbol{x}_{t+1}|\boldsymbol{\eta}_i^{\text{train}}) = \iiint p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t) p(\boldsymbol{x}_t, \boldsymbol{u}_t|\boldsymbol{\eta}_i^{\text{train}}) df\, d\boldsymbol{x}_t\, d\boldsymbol{u}_t \quad (4)$$

cannot be computed analytically for nonlinear covariance functions. However, we approximate it by a Gaussian distribution $\mathcal{N}\big(\boldsymbol{x}_{t+1} \,|\, \boldsymbol{\mu}_{t+1}^x, \boldsymbol{\Sigma}_{t+1}^x\big)$ using exact moment matching [9]. In (4), the transition probability $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t) = p(f(\boldsymbol{x}_t, \boldsymbol{u}_t)|\boldsymbol{x}_t, \boldsymbol{u}_t)$ is the GP predictive distribution at $(\boldsymbol{x}_t, \boldsymbol{u}_t)$. Iterating the moment-matching approximation of (4) for all time steps of the finite horizon T yields Gaussian marginal predictive distributions $p(\boldsymbol{x}_1|\boldsymbol{\eta}_i^{\text{train}}), \ldots, p(\boldsymbol{x}_T|\boldsymbol{\eta}_i^{\text{train}})$.

Second, these approximate Gaussian long-term predictive state distributions are used to compute the expected immediate cost $\mathbb{E}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)|\boldsymbol{\eta}_i^{\text{train}}] = \int c_\eta(\boldsymbol{x}_t) p(\boldsymbol{x}_t|\boldsymbol{\eta}_i^{\text{train}}) d\boldsymbol{x}_t$ for a particular task $\boldsymbol{\eta}_i^{\text{train}}$, where $p(\boldsymbol{x}_t|\boldsymbol{\eta}_i^{\text{train}}) = \mathcal{N}\big(\boldsymbol{x}_t \,|\, \boldsymbol{\mu}_t^x, \boldsymbol{\Sigma}_t^x\big)$ and c_η is a task-specific cost function. This integral can be solved analytically for many choices of the immediate cost function c_η, such as polynomials or unnormalized Gaussians. Summing the values $\mathbb{E}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)|\boldsymbol{\eta}_i^{\text{train}}]$ from $t = 1, \ldots, T$ finally yields $J^\pi(\boldsymbol{\theta}, \boldsymbol{\eta}_i^{\text{train}})$ in (3).

C. Gradient-based Policy Improvement

The closed-form approximation of $J^\pi(\boldsymbol{\theta}, \boldsymbol{\eta})$ by means of moment matching allows for an analytic computation of the corresponding gradient $dJ^\pi(\boldsymbol{\theta}, \boldsymbol{\eta})/d\boldsymbol{\theta}$ with respect to the policy parameters $\boldsymbol{\theta}$, see (3) and line 6 in Fig. 3, given by

$$dJ^\pi(\boldsymbol{\theta}, \boldsymbol{\eta})/d\boldsymbol{\theta} = \sum\nolimits_{t=1}^{T} \frac{d}{d\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)|\boldsymbol{\eta}]\,. \quad (5)$$

These gradients can be used in any gradient-based optimization toolbox, e.g., BFGS (line 7). The derivatives of $J^\pi(\boldsymbol{\theta}, \boldsymbol{\eta}_i^{\text{train}})$ with respect to the policy parameters $\boldsymbol{\theta}$ requires repeated application of the chain-rule. Defining $\mathcal{E}_t := \mathbb{E}_{\boldsymbol{x}_t}[c(\boldsymbol{x}_t)|\boldsymbol{\eta}_i^{\text{train}}]$ in (5) yields

$$\frac{d\mathcal{E}_t}{d\boldsymbol{\theta}} = \frac{d\mathcal{E}_t}{dp(\boldsymbol{x}_t)} \frac{dp(\boldsymbol{x}_t)}{d\boldsymbol{\theta}} := \frac{\partial \mathcal{E}_t}{\partial \boldsymbol{\mu}_t^x} \frac{d\boldsymbol{\mu}_t^x}{d\boldsymbol{\theta}} + \frac{\partial \mathcal{E}_t}{\partial \boldsymbol{\Sigma}_t^x} \frac{d\boldsymbol{\Sigma}_t^x}{d\boldsymbol{\theta}}\,, \quad (6)$$

where we took the derivative with respect to $p(\boldsymbol{x}_t)$, i.e., the parameters (mean $\boldsymbol{\mu}_t^x$ and covariance $\boldsymbol{\Sigma}_t^x$) of the Gaussian approximation to the state distribution $p(\boldsymbol{x}_t)$. The chain-rule yields the total derivative

$$\frac{dp(\boldsymbol{x}_t)}{d\boldsymbol{\theta}} = \frac{\partial p(\boldsymbol{x}_t)}{\partial p(\boldsymbol{x}_{t-1})} \frac{dp(\boldsymbol{x}_{t-1})}{d\boldsymbol{\theta}} + \frac{\partial p(\boldsymbol{x}_t)}{\partial \boldsymbol{\theta}}\,. \quad (7)$$

We assume that the total derivative $dp(\boldsymbol{x}_{t-1})/d\boldsymbol{\theta}$ is known from the computation for the previous time step. Hence, we only need to compute the partial derivative $\partial p(\boldsymbol{x}_t)/\partial \boldsymbol{\theta}$. Note that $\boldsymbol{x}_t = f(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}) + \boldsymbol{w}$ and $\boldsymbol{u}_{t-1} = \pi(\boldsymbol{x}_{t-1}, g(\boldsymbol{x}_{t-1}, \boldsymbol{\eta}), \boldsymbol{\theta})$. Therefore, we obtain $\partial p(\boldsymbol{x}_t)/\partial \boldsymbol{\theta} = \{\partial \boldsymbol{\mu}_t^x/\partial \boldsymbol{\theta}, \partial \boldsymbol{\Sigma}_t^x/\partial \boldsymbol{\theta}\}$ with

$$\begin{aligned}\frac{\partial\{\boldsymbol{\mu}_t^x, \boldsymbol{\Sigma}_t^x\}}{\partial \boldsymbol{\theta}} &= \frac{\partial\{\boldsymbol{\mu}_t^x, \boldsymbol{\Sigma}_t^x\}}{\partial p(\boldsymbol{u}_{t-1})} \frac{\partial p(\boldsymbol{u}_{t-1})}{\partial \boldsymbol{\theta}}\\ &= \frac{\partial\{\boldsymbol{\mu}_t^x, \boldsymbol{\Sigma}_t^x\}}{\partial \boldsymbol{\mu}_{t-1}^u} \frac{\partial \boldsymbol{\mu}_{t-1}^u}{\partial \boldsymbol{\theta}} + \frac{\partial\{\boldsymbol{\mu}_t^x, \boldsymbol{\Sigma}_t^x\}}{\partial \boldsymbol{\Sigma}_{t-1}^u} \frac{\partial \boldsymbol{\Sigma}_{t-1}^u}{\partial \boldsymbol{\theta}}\,. \quad (8)\end{aligned}$$

We approximate the distribution of the control signal $p(\boldsymbol{u}_{t-1}) = \int \pi(\boldsymbol{x}_{t-1}, g(\boldsymbol{x}_{t-1}, \boldsymbol{\eta}), \boldsymbol{\theta}) p(\boldsymbol{x}_{t-1}) d\boldsymbol{x}_{t-1}$ by a Gaussian with mean $\boldsymbol{\mu}_{t-1}^u$ and covariance $\boldsymbol{\Sigma}_{t-1}^u$. These moments (and their gradients with respect to $\boldsymbol{\theta}$) can often be computed analytically, e.g., in linear models with polynomial or Gaussian basis functions. The augmentation of the policy with the (transformed) task variable requires an additional layer of gradients for computing $dJ^\pi(\boldsymbol{\theta})/d\boldsymbol{\theta}$. The variable transformation affects the partial derivatives of $\boldsymbol{\mu}_{t-1}^u$ and $\boldsymbol{\Sigma}_{t-1}^u$ (marked red in (8)), such that

$$\begin{aligned}\frac{\partial\{\boldsymbol{\mu}_{t-1}^u, \boldsymbol{\Sigma}_{t-1}^u\}}{\partial\{\boldsymbol{\mu}_{t-1}^x, \boldsymbol{\Sigma}_{t-1}^x, \boldsymbol{\theta}\}} &= \frac{\partial\{\boldsymbol{\mu}_{t-1}^u, \boldsymbol{\Sigma}_{t-1}^u\}}{\partial p(\boldsymbol{x}_{t-1}, g(\boldsymbol{x}_{t-1}, \boldsymbol{\eta}))}\\ &\quad \times \frac{\partial p(\boldsymbol{x}_{t-1}, g(\boldsymbol{x}_{t-1}, \boldsymbol{\eta}))}{\partial\{\boldsymbol{\mu}_{t-1}^x, \boldsymbol{\Sigma}_{t-1}^x, \boldsymbol{\theta}\}}\,. \quad (9)\end{aligned}$$

We incorporate these derivatives into (8) via the chain and product-rules for an analytic gradient $dJ^\pi(\boldsymbol{\theta}, \boldsymbol{\eta}_i^{\text{train}})/d\boldsymbol{\theta}$ in (3) to update the policy (lines 6–7 in Fig. 3).

III. Evaluations and Results

We applied our approach to multi-task policy search to three tasks: 1) the under-actuated cart-pole swing-up, 2) a low-cost robotic manipulator system that learns block stacking, 3) an imitation learning ball-hitting task with a tendon-driven robot. In all cases, the system dynamics were unknown and inferred from data using GPs.

A. Multi-Task Cart-Pole Swing-up

The cart-pole system consists of a cart with mass $0.5\,\mathrm{kg}$ and a pendulum of length $0.6\,\mathrm{m}$ and mass $0.5\,\mathrm{kg}$ attached to the cart. Every $0.1\,\mathrm{s}$, an external force was applied to the cart, but not to the pendulum. The state $\boldsymbol{x} = [\chi, \dot{\chi}, \varphi, \dot{\varphi}]$ of the system comprised the position χ and velocity $\dot{\chi}$ of the cart, the angle φ and angular velocity $\dot{\varphi}$ of the pendulum. The nonlinear controller was parametrized as a regularized RBF network with 100 Gaussian basis functions. The controller parameters were the locations of the basis functions, a shared (diagonal) width-matrix, and the weights, resulting in approximately 800 policy parameters.

Initially, the system was expected to be in a state, where the pendulum hangs down; more specifically, $p(\boldsymbol{x}_0) = \mathcal{N}(\mathbf{0}, 0.1^2\boldsymbol{I})$. By pushing the cart to the left and to the right, the objective was to swing the pendulum up and to balance it in the inverted position at a target location η of the cart specified at *test time*, such that $\boldsymbol{x}_{\text{target}} = [\eta, *, \pi + 2k\pi, *]$ with $\eta \in [-1.5, 1.5]\,\mathrm{m}$ and $k \in \mathbb{Z}$. The cost function c in (1) was chosen as $c(\boldsymbol{x}) = 1 - \exp(-8\|\boldsymbol{x} - \boldsymbol{x}_{\text{target}}\|^2) \in [0, 1]$ and penalized the Euclidean distance of the tip of the pendulum from its desired inverted position with the cart being at target location η. Optimally solving the task required the cart to stop at the target location η. Balancing the pendulum with the cart offset by $20\,\mathrm{cm}$ caused an immediate cost (per time step) of about 0.4. We considered four experimental set-ups:

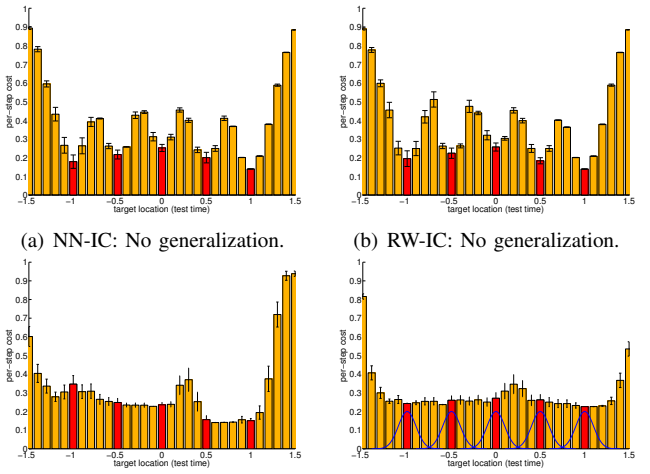
Nearest neighbor independent controllers (NN-IC): A baseline experiment with five independently learned controllers for the desired swing-up locations $\eta = \{\pm 1\,\mathrm{m}, \pm 0.5\,\mathrm{m}, 0\,\mathrm{m}\}$. Each controller was learned by the PILCO framework [9] in 10 trials with a total experience of $200\,\mathrm{s}$. For the test tasks η^{test}, we applied the controller with the nearest training task η^{train}.

Re-weighted independent controllers (RW-IC): Training was identical to NN-IC. At test time, we combined individual controllers using a gating network, similar to [18], resulting in a convex combination of local policies with weights

$$v_i = \frac{\exp\left(-\frac{1}{2\kappa}\|\boldsymbol{\eta}^{\text{test}} - \boldsymbol{\eta}_i^{\text{train}}\|^2\right)}{\sum_j^{|\boldsymbol{\eta}^{\text{train}}|} \exp\left(-\frac{1}{2\kappa}\|\boldsymbol{\eta}^{\text{test}} - \boldsymbol{\eta}_j^{\text{train}}\|^2\right)}, \tag{10}$$

such that the applied control signal was $\boldsymbol{u} = \sum_i v_i \pi_i(\boldsymbol{x})$. An extensive grid search resulted in $\kappa = 0.0068\,\mathrm{m}^2$, leading to the best test performance in this scenario, making RW-IC nearly identical to the NN-IC.

Multi-task policy search, $\boldsymbol{\Sigma}^\eta = \mathbf{0}$ (MTPS0): Multi-task policy search with five known tasks during training, which only differ in the location of the cart where the pendulum is supposed to be balanced. The target locations were $\eta^{\text{train}} = \{\pm 1\,\mathrm{m}, \pm 0.5\,\mathrm{m}, 0\,\mathrm{m}\}$. Moreover, $g(\boldsymbol{x}_t, \eta) = \eta - \chi(t)$ and $\boldsymbol{\Sigma}^\eta = \mathbf{0}$. We show results after 20 trials, i.e., a total experience of $70\,\mathrm{s}$ only.



(a) NN-IC: No generalization. (b) RW-IC: No generalization.

(c) MTPS0: Some generalization. (d) MTPS+: Good generalization in the "coverage" of the Gaussians.

Fig. 4. Generalization performance for the multi-task cart-pole swing-up. The graphs show the expected cost per time step along with twice the standard errors.

Multi-task policy search, $\boldsymbol{\Sigma}^\eta > \mathbf{0}$ (MTPS+): Multi-task policy search with the five training tasks $\eta^{\text{train}} = \{\pm 1\,\mathrm{m}, \pm 0.5\,\mathrm{m}, 0\,\mathrm{m}\}$, but with training task covariance $\boldsymbol{\Sigma}^\eta = \mathrm{diag}([0.1^2, 0, 0, 0])$. We show results after 20 trials, i.e., $70\,\mathrm{s}$ total experience.

For the performance analysis, we applied the learned policies 100 times to the test-target locations $\boldsymbol{\eta}^{\text{test}} = -1.5, -1.4, \ldots, 1.5$. The initial state of a rollout was sampled from $p(\boldsymbol{x}_0)$. For the MTPS experiments, we plugged the test tasks into (2) to compute the corresponding controls.

Fig. 4 illustrates the generalization performance of the learned controllers. The horizontal axes denote the locations η^{test} of the target position of the cart at *test* time. The height of the bars show the average (over trials) cost per time step. The means of the *training* tasks η^{train} are the location of the red bars. Fig. 4(d) shows the distribution $p(\eta_i^{\text{train}})$ used during training as the bell-curves, which approximately covers the task range $\eta \in [-1.2\,\mathrm{m}, 1.2\,\mathrm{m}]$.

The NN-IC controller, see Fig. 4(a), balanced the pendulum at a cart location that was not further away than $0.2\,\mathrm{m}$, which incurred a cost of up to 0.45. In Fig. 4(b), the performances for the hierarchical RW-IC controller are shown. The performance for the best value κ in the gating network, see (10), was similar to the performance of the NN-IC controller. However, between the training tasks for the local controllers, i.e., in the range of $[-0.9\,\mathrm{m}, 0.9\,\mathrm{m}]$, the convex combination of local controllers led to more failures than in NN-IC, where the pendulum could not be swung up successfully: Convex combinations of nonlinear local controllers eventually decreased the (non-existing) generalization performance of RW-IC.

Fig. 4(c) shows the performance of the MTPS0 controller. The MTPS0 controller successfully performed the swing-up plus balancing task for all tasks η^{test} close to the training

tasks. However, the performance varied strongly. Fig. 4(d) shows that the MTPS+ controller successfully performed the swing-up plus balancing task for all tasks η^{test} at test time that were sufficiently covered by the uncertain training tasks $\eta_i^{\text{train}}, i = 1, \ldots, 5$, indicated by the bell curves representing $\boldsymbol{\Sigma}^{\eta} > 0$. Relatively constant performance across the test tasks covered by the bell curves was achieved. An average cost of 0.3 meant that the pendulum might be balanced with the cart slightly offset. Fig. 2 shows the learned MTPS+ policy for all test tasks η^{test} with the state $\boldsymbol{x} = \boldsymbol{\mu}_0$ fixed.

Tab. I summarizes the controller performances. We averaged over all test tasks $\boldsymbol{\eta}^{\text{test}}$ and 100 applications of the learned policy, where the initial state was sampled from $p(\boldsymbol{x}_0)$. Although NN-IC and RW-IC performed the swing-up reliably, they incurred the largest cost: For most test tasks, they balanced the pendulum at the wrong cart position as they did not generalize from training tasks to unseen test tasks. In the MTPS experiments, the average cost was lowest. MTPS+ led to the best overall generalization performance, although it might not solve each individual test task optimally.

Fig. 2 illustrates the difference in generalization performance between our MTPS+ approach and the RW-IC approach, where controls u_i from local policies π_i are combined. Since the local policies are trained independently, a (convex) combination of local controls makes only sense in special cases, e.g., when the local policies are linear in the parameters. In this example, however, the local policies are nonlinear. Since the local policies are learned independently, their overall generalization performance is poor. On the other hand, MTPS+ learns a single policy for a task η_i always in the light of all other tasks $\eta_{j \neq i}$ as well, and, therefore, leads to an overall smooth generalization.

B. Multi-Task Robot Manipulator

We applied our proposed multi-task learning method to a block-stacking task with a low-cost, off-the-shelf robotic manipulator ($370) by Lynxmotion [1], see Fig. 5, and a PrimeSense [2] depth camera ($130) as a visual sensor. The arm had six controllable degrees of freedom: base rotate, three joints, wrist rotate, and a gripper (open/close). The plastic arm could be controlled by commanding both a desired configuration of the six servos and the duration for executing the command [11]. The camera was identical to the Kinect sensor, providing a synchronized depth and RGB image at 30 Hz. We used the camera for 3D-tracking of the block in the robot's gripper.

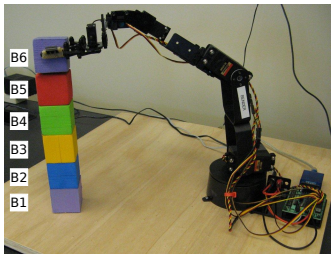


Fig. 5. Low-cost manipulator by Lynxmotion [1] performing a block-stacking task using visual feedback from a PrimeSense depth camera.

TABLE I

MULTI-TASK CART-POLE: AVERAGE COSTS ACROSS 31 TEST TASKS.

	NN-IC	RW-IC	MTPS0	MTPS+
Cost	0.39	0.4	0.33	**0.30**

The goal was to make the robot learn to stack a tower of six blocks using multi-task learning. The cost function c in (1) penalized the distance of the block in the gripper from the desired drop-off location. We only specified the 3D camera coordinates of the blocks B2, B4, and B5 as the training tasks $\boldsymbol{\eta}^{\text{train}}$, see Fig. 5. Thus, at test time, stacking B3 and B6 required exploiting the generalization of our multi-task policy search. We chose $g(\boldsymbol{x}, \boldsymbol{\eta}) = \boldsymbol{\eta} - \boldsymbol{x}$ and set $\boldsymbol{\Sigma}^{\eta}$ such that the task space, i.e., all 6 blocks, was well covered. The mean $\boldsymbol{\mu}_0$ of the initial distribution $p(\boldsymbol{x}_0)$ corresponded to an upright configuration of the arm.

A GP dynamics model was learned that mapped the 3D camera coordinates of the block in the gripper and the commanded controls at time t to the corresponding 3D coordinates of the block in the gripper at time $t + 1$. The control signals were changed at a rate of 2 Hz. Note that the learned model is not an inverse kinematics model as the robot's joint state is unknown. We used an affine policy $\boldsymbol{u}_t = \pi(\boldsymbol{x}_t, \boldsymbol{\eta}, \boldsymbol{\theta}) = \boldsymbol{A}\boldsymbol{x}_t^{x,\eta} + \boldsymbol{b}$ with $\boldsymbol{\theta} = \{\boldsymbol{A}, \boldsymbol{b}\}$. The policy defined a mapping $\pi : \mathbb{R}^6 \to \mathbb{R}^4$, where the four controlled degrees of freedom were the base rotate and three joints.

We report results based on 16 training trials, each of length 5 s, which amounts to a total experience of 80 s only. The test phase consisted of 10 trials per stacking task, where the arm was supposed to stack the block on the currently topmost block. The tasks $\boldsymbol{\eta}_j^{\text{test}}$ at test time corresponded to stacking blocks B2–B6 in Fig. 5. Fig. 6 shows the average distance of the block in the gripper from the target position, which was $b = 4.3$ cm above the topmost block. Here, "block2" means that the task was to move block B2 in the gripper on top of block 1. The horizontal axis shows times at which the manipulator's control signal was changed (rate 2 Hz), the vertical axis shows the average distances (over 10 test trials) to the target position in meters. For all blocks (including blocks B3 and B6, which were not part of the training tasks $\boldsymbol{\eta}^{\text{train}}$) the distances approached zero over time. Thus, the learned multi-task controller interpolated (block B3) and extrapolated (block B6) from the training tasks to the test tasks *without* re-training.

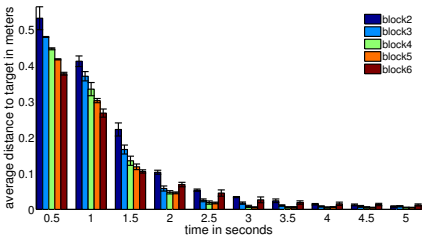
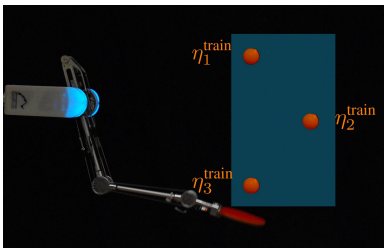


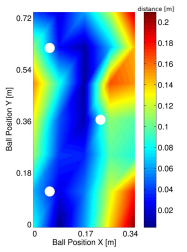
Fig. 6. Average distances of the block in the gripper from the target position with twice the standard error.

C. Multi-Task Imitation Learning of Ball-Hitting Movements

We demonstrate the successful application of our MTPS approach to imitation learning. Instead of defining a cost function c in (1), a teacher provides demonstrations that the robot should imitate. We show that our MTPS approach allows to generalize from demonstrated behavior to behaviors that have not been observed before. In [12], we developed a method for model-based imitation learning based on probabilistic trajectory matching for a single task. The key idea



(a) Set-up for the imitation learning experiments. Orange balls represent the three training tasks $\boldsymbol{\eta}_i^{\text{train}}$. The blue rectangle indicates the regions of the test tasks $\boldsymbol{\eta}_j^{\text{test}}$ to which the learned controller is supposed to generalize.



(b) Training task locations (white disks). Blue and cyan indicate that the task was solved successfully.

Fig. 7. Set-up and results for the imitation learning experiments with a bio-inspired BioRob$^{\text{TM}}$.

is to match a distribution over predicted robot trajectories $p(\boldsymbol{\tau}^{\pi})$ directly with an observed distribution $p(\boldsymbol{\tau}^{\text{exp}})$ over expert trajectories $\boldsymbol{\tau}^{\text{exp}}$ by finding a policy π^* that minimizes the KL divergence between them.

In this paper, we extend this imitation learning approach to a multi-task scenario to jointly learning to imitate multiple tasks from a small set of demonstrations. In particular, we applied our multi-task learning approach to learning a controller for hitting movements with variable ball positions in a 2D-plane using the tendon-driven BioRob$^{\text{TM}}$ X4, a five DoF compliant, light-weight robotic arm, capable of achieving high accelerations, see Fig. 7(a). While the BioRob's design has advantages over traditional approaches, modeling and controlling such a compliant system is challenging.

We considered three joints of the robot, such that the state $\boldsymbol{x} \in \mathbb{R}^6$ contained the joint positions $\boldsymbol{q}$ and velocities $\dot{\boldsymbol{q}}$ of the robot. The corresponding motor torques $\boldsymbol{u} \in \mathbb{R}^3$ were directly determined by the policy π. For learning a controller, we used an RBF network with 250 Gaussian basis functions with about 2300 policy parameters. Unlike in the previous examples, we represented a task as a two-dimensional vector $\boldsymbol{\eta} \in \mathbb{R}^2$ corresponding to the ball position in Cartesian coordinates in an arbitrary reference frame within the hitting plane. As the task representation $\boldsymbol{\eta}$ was basically an index and, hence, unrelated to the state of the robot, $g(\boldsymbol{x}, \boldsymbol{\eta}) = \boldsymbol{\eta}$, the cross-covariances $\boldsymbol{C}^{x\eta}$ in (2) were $\mathbf{0}$.

As training tasks $\boldsymbol{\eta}_j^{\text{train}}$, we defined hitting movements for three different ball positions, see Fig. 7(a). For each training task, an expert demonstrated two hitting movements via kinesthetic teaching. Our goal was to learn a single policy that a) learns to imitate three distinct expert demonstrations, and b) generalizes from demonstrated behaviors to tasks that were not demonstrated. These tests tasks were defined as hitting balls in a larger region around the training locations, indicated by the blue box in Fig. 7(a). We set the matrices $\boldsymbol{\Sigma}^{\eta}$ such that the blue box was covered well. Fig. 7(b) shows the performance results as a heatmap after 15 iterations of the algorithm in Fig. 3. The evaluation measure was the distance in m between the ball position and the center of the table-tennis racket. We computed this error in a regular 7x5 grid of the blue area in Fig. 7(a). The distances in the blue and cyan areas were sufficient to successfully hit the ball (the racket's radius is about 0.08 m). Hence, our approach successfully generalized from given demonstrations to new tasks.

IV. Conclusion

We have presented a policy-search approach to multi-task learning for robots with stationary dynamics. Instead of combining local policies, our approach learns a single policy jointly for all tasks. The key idea is to explicitly parametrize the policy by the task and, thus, enable the policy to generalize from training tasks to similar, but unknown, tasks at test time. This generalization is phrased as an optimization problem, which can be solved jointly with learning the policy parameters. For solving this optimization problem, we incorporated our approach into the PILCO policy search framework, which allows for data-efficient policy learning. We have reported promising real-robot results on both multi-task RL and imitation learning, the latter of which allows to generalize imitated behavior to solving tasks that were not in the library of demonstrations.

References

[1] http://www.lynxmotion.com.
[2] http://www.primesense.com.
[3] P. Abbeel and A. Ng. Exploration and Apprenticeship Learning in Reinforcement Learning. In *ICML*, pages 1–8, 2005.
[4] P. Abbeel, M. Quigley, and A. Ng. Using Inaccurate Models in Reinforcement Learning. In *ICML*, pages 1–8, 2006.
[5] C. Atkeson and J. Santamaría. A Comparison of Direct and Model-Based Reinforcement Learning. In *ICRA*, 1997.
[6] J. Bagnell and J. Schneider. Autonomous Helicopter Control using Reinforcement Learning Policy Search Methods. In *ICRA*, 2001.
[7] S. Barrett, M. Taylor, and P. Stone. Transfer Learning for Reinforcement Learning on a Physical Robot. In *AAMAS*, 2010.
[8] B. da Silva, G. Konidaris, and A. Barto. Learning Parametrized Skills. In *ICML*, 2012.
[9] M. Deisenroth, D. Fox, and C. Rasmussen. Gaussian Processes for Data-Efficient Learning in Robotics and Control. *IEEE TPAMI*, 2014.
[10] M. Deisenroth, G. Neumann, and J. Peters. *A Survey on Policy Search for Robotics*. NOW Publishers, 2013.
[11] M. Deisenroth, C. Rasmussen, and D. Fox. Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning. In *RSS*, 2011.
[12] P. Englert, A. Paraschos, J. Peters, and M. Deisenroth. Probabilistic Model-based Imitation Learning. *Adaptive Behavior*, 21, 2013.
[13] A. Ijspeert, J. Nakanishi, and S. Schaal. Learning Attractor Landscapes for Learning Motor Primitives. In *NIPS*, pages 1523–1530, 2002.
[14] R. Jacobs, M. Jordan, S. Nowlan, and G. Hinton. Adaptive Mixtures of Local Experts. *Neural Computation*, 3:79–87, 1991.
[15] J. Kober, A. Wilhelm, E. Oztop, and J. Peters. Reinforcement Learning to Adjust Parametrized Motor Primitives to New Situations. *Autonomous Robots*, 33(4):361–379, 2012.
[16] G. Konidaris, I. Scheidwasser, and A. Barto. Transfer in Reinforcement Learning via Shared Features. *JMLR*, 13:1333–1371, 2012.
[17] O. Kroemer, R. Detry, J. Piater, and J. Peters. Combining Active Learning and Reactive Control for Robot Grasping. *Robotics and Autonomous Systems*, 58:1105–1116, 2010.
[18] K. Mülling, J. Kober, O. Kroemer, and J. Peters. Learning to Select and Generalize Striking Movements in Robot Table Tennis. *IJRR*, 2013.
[19] S. Schaal. Learning From Demonstration. In *NIPS*. 1997.
[20] J. Schneider. Exploiting Model Uncertainty Estimates for Safe Dynamic Control Learning. In *NIPS*. 1997.
[21] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.
[22] M. Taylor and P. Stone. Cross-Domain Transfer for Reinforcement Learning. In *ICML*, 2007.
[23] M. Taylor and P. Stone. Transfer Learning for Reinforcement Learning Domains: A Survey. *JMLR*, 10(1):1633–1685, 2009.