Noname manuscript No.
(will be inserted by the editor)

Reinforcement Learning to Adjust Parametrized Motor Primitives to New Situations

Jens Kober · **Andreas Wilhelm** · **Erhan Oztop** · **Jan Peters**

Received: 20 May 2011 / Accepted: 17 March 2012

Abstract Humans manage to adapt learned movements very quickly to new situations by generalizing learned behaviors from similar situations. In contrast, robots currently often need to re-learn the complete movement. In this paper, we propose a method that learns to generalize parametrized motor plans by adapting a small set of global parameters, called meta-parameters. We employ reinforcement learning to learn the required meta-parameters to deal with the current situation, described by states. We introduce an appropriate reinforcement learning algorithm based on a kernelized version of the reward-weighted regression. To show its feasibility, we evaluate this algorithm on a toy example and compare it to several previous approaches. Subsequently, we apply the approach to three robot tasks, i.e., the generalization of throwing movements in darts, of hitting movements in table tennis, and of throwing balls where the tasks are learned on several different real physical robots, i.e., a Barrett WAM, a BioRob, the JST-ICORP/SARCOS CBi and a Kuka KR 6.

Keywords: skill learning, motor primitives, reinforcement learning, meta-parameters, policy learning

J. Kober · J. Peters
MPI for Intelligent Systems, Tübingen, Germany / TU Darmstadt, Darmstadt, Germany
E-mail: {jens.kober,jan.peters}@tuebingen.mpg.de

A. Wilhelm
University of Applied Sciences Ravensburg-Weingarten, Weingarten, Germany
E-mail: AWilhelm@arcor.de

E. Oztop
ATR, Kyoto, Japan / NICT, Kyoto, Japan / Özyeğin University, Istanbul, Turkey
E-mail: erhan@atr.jp

1 Introduction

Human movements appear to be represented using movement templates, also called motor primitives (Schmidt and Wrisberg, 2000). Once learned, these templates allow humans to quickly adapt their movements to variations of the situation without the need of re-learning the complete movement. For example, the overall shape of table tennis forehands are very similar when the swing is adapted to varied trajectories of the incoming ball and a different targets on the opponent's court. To accomplish such behavior, the human player has learned by trial and error how the global parameters of a generic forehand need to be adapted due to changes in the situation (Mülling et al, 2010, 2011).

In robot learning, motor primitives based on dynamical systems (Ijspeert et al, 2002; Schaal et al, 2007) can be considered a technical counterpart to these templates. They allow acquiring new behaviors quickly and reliably both by imitation and reinforcement learning. Resulting successes have shown that it is possible to rapidly learn motor primitives for complex behaviors such as tennis-like swings (Ijspeert et al, 2002), T-ball batting (Peters and Schaal, 2008b), drumming (Pongas et al, 2005), biped locomotion (Nakanishi et al, 2004), ball-in-a-cup (Kober and Peters, 2011b), and even in tasks with potential industrial applications (Urbanek et al, 2004). While the examples are impressive, they do not yet address how a motor primitive can be generalized to a different behavior by trial and error without re-learning the task. Such generalization of behaviors can be achieved by adapting the meta-parameters of the movement representation. Meta-parameters are defined as a small set of parameters that adapt the global movement behavior. The dynamical system motor primitives can be adapted both spatially and temporally without changing the overall shape of the motion (Ijspeert et al, 2002). In this paper, we learn a mapping from a range of changed situations, described by states, to the meta-parameters to adapt the template's behavior. We consider movements where it is sufficient to reshape (e.g., rescale the motion spatially and/or temporally) the global movement by optimizing meta-parameters to adapt to a new situation instead of tuning the movement primitive's shape parameters that describe the fine details of the movement.

Dynamical systems motor primitives have the capability to adapt the movement to a changed end positions. Here, the end position is a meta-parameter. This was exploited in (Ijspeert et al, 2002) for tennis-like swings with static ball targets and in (Pastor et al, 2009) for object manipulation. In these papers, the desired end position is given in Cartesian coordinates and the movement primitives operate in Cartesian coordinates as well. Thus, the meta-parameters of the motor primitives are straightforward to set. In this paper, we are interested in non-intuitive connections , where the relation between the desired outcome and the meta-parameters is not straightforward. There is related prior work in the context of programming by demonstration by Ude et al (2010) and Kronander et al (2011) who employ supervised learning to learn a mapping from desired outcomes to meta-parameters for tasks such as reaching, throwing, drumming, and mini-golf. They assume that a teacher has presented a number of demonstrations that cannot be contradictory and the task is to imitate and generalize these demonstrations. Lampariello et al (2011) employ a global planner to provide demonstrations of optimal catching meta-parameters and use supervised learning approaches to generalize these in real-time. In contrast, in our setting the robot actively explores different movements and improves the behavior according to a cost function. It can deal with contradictory demonstrations and actively generate its own scenarios by exploration combined with self-improvement. As mentioned in (Ude et al, 2010), the two approaches may even be complimentary: reinforcement learning can provide demonstrations for supervised learning, and supervised learning can be used as a starting point for reinforcement learning.

Adapting movements to situations is also discussed in (Jetchev and Toussaint, 2009) in a supervised learning setting. Their approach is based on predicting a trajectory from a previously demonstrated set and refining it by motion planning. The authors note that kernel ridge regression performed poorly for the prediction if the new situation is far from previously seen ones as the algorithm yields the global mean. In our approach, we employ a cost weighted mean that overcomes this problem. If the situation is far from previously seen ones, large exploration will help to find a solution.

In machine learning, there have been many attempts to use meta-parameters in order to generalize between tasks (Caruana, 1997). Particularly, in grid-world domains, significant speed-up could be achieved by adjusting policies by modifying their meta-parameters, e.g., re-using options with different subgoals (McGovern and Barto, 2001). The learning of meta-parameters of the learning algorithm has been proposed as a model for neuromodulation in the brain (Doya, 2002). In contrast, we learn the meta-parameters of a motor skill in this paper. In robotics, such meta-parameter learning could be particularly helpful due to the complexity of reinforcement learning for complex motor skills with high dimensional states and actions. The cost of experience is high as sample generation is time consuming and often requires human interaction (e.g., in cart-pole, for placing the pole back on the robot's hand) or supervision (e.g., for safety during the execution of the trial). Generalizing a teacher's demonstration or a previously learned policy to new situations may reduce both the complexity of the task and the number of required samples. Hence, a reinforcement learning method for acquiring and refining meta-parameters of pre-structured primitive movements becomes an essential next step, which we will address in this paper.

This paper does not address the problem of deciding whether it is more advantageous to generalize existing generic movements or to learn a novel one. Similar to most reinforcement learning approaches, the states and meta-parameters (which correspond to actions in the standard reinforcement learning settings) as well as the cost or reward function need to be designed by the user prior to the learning process. Here, we can only provide a few general indications with regard to the choice of these setting. Cost functions need to capture the desired outcome of the reinforcement learning process. Often the global target can be described verbally - but it is not obvious how the cost needs to be scaled and how to take secondary optimization criteria into account. For example, when throwing at a target, the global goal is hitting it. However, it is not obvious which distance metric should be used to score misses, which secondary criteria (e.g. required torques) should be included, and which weight each criterion should be assigned. These choices influence both the learning performance and the final policy. Even for human reaching movements, the underlying cost function is not completely understood (Bays and Wolpert, 2007). In practice, informative cost functions (i.e., cost functions that contain a notion of closeness) often perform better than binary reward functions in robotic tasks. In this paper, we used a number of cost functions both with and without secondary objectives. In the future, inverse reinforcement learning (Russell, 1998) may be a useful alternative to automatically recover underlying cost functions from data as done already in other settings.

The state of the environment needs to enable the robot to obtain sufficient information to react appropriately. The proposed algorithm can cope with superfluous states at a cost of slower learning. Similarly, the meta-parameters are defined by the underlying representation of the movement. For example, the dynamical systems motor primitives (Ijspeert et al, 2002; Schaal et al, 2007) have meta-parameters for scaling the duration and amplitude of the movement as well as the possibility to change the final position. Restricting the meta-parameters to task relevant ones, may often speed up the learning process.

We present current work on automatic meta-parameter acquisition for motor primitives by reinforcement learning. We focus on learning the mapping from situations to meta-parameters and how to employ these in dynamical systems motor primitives. We extend the motor primitives of Ijspeert et al (2002) with a learned meta-parameter function and re-frame the problem as an episodic reinforcement learning scenario. In order to obtain an algorithm for fast reinforcement learning of meta-parameters, we view reinforcement learning as a reward-weighted self-imitation (Peters and Schaal, 2008a; Kober and Peters, 2011b). Compared to the preliminary version of this paper (Kober et al, 2010b), this paper includes significantly extended real robot evaluations, an extended description of the approach, a comparison to related approaches, as well as an application to active learning.

To have a general meta-parameter learning, we adopted a parametric method, the reward-weighed regression (Peters and Schaal, 2008a), and turned it into a non-parametric one. We call this method Cost-regularized Kernel Regression (CrKR), which is related to Gaussian process regression (Rasmussen and Williams, 2006) but differs in the key aspects of incorporating costs and exploration naturally. We compare the CrKR with a traditional policy gradient algorithm (Peters and Schaal, 2008b), the reward-weighted regression (Peters and Schaal, 2008a), and supervised learning (Ude et al, 2010; Kronander et al, 2011) on a toy problem in order to show that it outperforms available previously developed approaches. As complex motor control scenarios, we evaluate the algorithm in the acquisition of flexible motor primitives for dart games such as *Around the Clock* (Masters Games Ltd., 2010), for *table tennis*, and for *ball target throwing*.

2 Meta-Parameter Learning for Motor Primitives

The goal of this paper is to show that elementary movements can be generalized by modifying only the meta-parameters of the primitives using learned mappings based on self-improvement. In Section 2.1, we first review how a single primitive movement can be represented and learned. We discuss how meta-parameters may be able to adapt the motor primitive spatially and temporally to the new situation. In order to develop algorithms that learn to automatically adjust such motor primitives, we model meta-parameter self-improvement as an episodic reinforcement learning problem in Section 2.2. While this problem could in theory be treated with arbitrary reinforcement learning methods, the availability of few samples suggests that more efficient, task appropriate reinforcement learning approaches are needed. To avoid the limitations of parametric function approximation, we aim for a kernel-based approach. When a movement is generalized, new parameter settings need to be explored. Hence, a predictive distribution over the meta-parameters is required to serve as an exploratory policy. These requirements lead to the method which we derive in Section 2.3 and employ for meta-parameter learning in Section 2.4.

2.1 Motor Primitives with Meta-Parameters

In this section, we review how the dynamical systems motor primitives (Ijspeert et al, 2002; Schaal et al, 2007) can be used for meta-parameter learning. The dynamical system motor primitives are a powerful movement representation that allows ensuring the stability of the movement[1], choosing between a rhythmic and a discrete movement and is invariant under rescaling of both duration and movement amplitude. These modification parameters can become part of the meta-parameters of the movement.

In this paper, we focus on single stroke movements which appear frequently in human motor control (Wulf, 2007; Schaal et al, 2007). Therefore, we will always focus on the discrete version of the dynamical system motor primitives in this paper. We use the most recent formulation of the discrete dynamical systems motor primitives (Schaal et al, 2007) where the phase z of the movement is represented by a single first order system

$$\dot{z} = -\tau \alpha_z z. \tag{1}$$

This canonical system has the time constant $\tau = 1/T$ where T is the duration of the motor primitive and a parameter α_z, which is chosen such that $z \approx 0$ at T. Subsequently, the internal state x of a second system is chosen such that positions q of all degrees of freedom are given by $\mathrm{q} = \mathrm{x}_1$, the velocities by $\dot{\mathrm{q}} = \tau \mathrm{x}_2 = \dot{\mathrm{x}}_1$ and the accelerations by $\ddot{\mathrm{q}} = \tau \dot{\mathrm{x}}_2$. The learned dynamics of Ijspeert motor primitives can be expressed in the following form

$$\begin{aligned} \dot{\mathrm{x}}_2 &= \tau \alpha_x \left(\beta_x \left(\mathrm{g} - \mathrm{x}_1\right) - \mathrm{x}_2\right) + \tau \mathrm{A} \mathrm{f}\left(z\right), \qquad (2) \\ \dot{\mathrm{x}}_1 &= \tau \mathrm{x}_2. \end{aligned}$$

This set of differential equations has the same time constant τ as the canonical system and parameters α_x, β_x are set such that the system is critically damped. The goal parameter g, a transformation function f and an amplitude matrix $\mathrm{A} = \mathrm{diag}\left(a_1, a_2, \ldots, a_I\right)$, with the amplitude modifier $\mathrm{a} = \left[a_1, a_2, \ldots, a_I\right]$ allow representing complex movements.

[1] Note that thehe dynamical systems motor primitives ensure the stability of the movement generation but cannot guarantee the stability of the movement execution (Ijspeert et al, 2002; Schaal et al, 2007).

In (Schaal et al, 2007), the authors use $\mathrm{a} = \mathrm{g} - \mathrm{x}_1^0$, with the initial position x_1^0, which ensures linear scaling. Other choices are possibly better suited for specific tasks, see for example (Park et al, 2008). The transformation function $\mathrm{f}(z)$ alters the output of the first system, in Equation (1), so that the second system in Equation (2), can represent complex nonlinear patterns and is given by

$$\mathrm{f}(z) = \sum_{n=1}^{N} \psi_n(z)\,\theta_n z. \tag{3}$$

Here, θ_n contains the n^{th} adjustable parameter of all degrees of freedom, N is the number of parameters per degree of freedom, and $\psi_n(z)$ are the corresponding weighting functions (Schaal et al, 2007). Normalized Gaussian kernels are used as weighting functions given by

$$\psi_n = \frac{\exp\left(-h_n(z-c_n)^2\right)}{\sum_{m=1}^{N}\exp\left(-h_m(z-c_m)^2\right)}. \tag{4}$$

These weighting functions localize the interaction in phase space using the centers c_n and widths h_n. As $z \approx 0$ at T, the influence of the transformation function $\mathrm{f}(z)$ in Equation (3) vanishes and the system stays at the goal position g. Note that the degrees of freedom (DoF) are usually all modeled independently in the second system in Equation (2). All DoFs are synchronous as the dynamical systems for all DoFs start at the same time, have the same duration and the shape of the movement is generated using the transformation $\mathrm{f}(z)$ in Equation (3), which is learned as a function of the shared canonical system in Equation (1).

One of the biggest advantages of this motor primitive framework (Ijspeert et al, 2002; Schaal et al, 2007) is that the second system in Equation (2), is linear in the shape parameters $\boldsymbol{\theta}$. Therefore, these parameters can be obtained efficiently, and the resulting framework is well-suited for imitation (Ijspeert et al, 2002) and reinforcement learning (Kober and Peters, 2011b). The resulting policy is invariant under transformations of the initial position x_1^0, the initial velocity x_2^0, the goal g, the amplitude A and the duration T (Ijspeert et al, 2002). In (Kober et al, 2010a), a variant of the motor primitive framework has been introduced that allows specifying non-zero goal velocities $\dot{\mathrm{g}}$. These six modification parameters can be used as the meta-parameters γ of the movement. Appendix A illustrates the influence of these meta-parameters on the movement generation. Obviously, we can make more use of the motor primitive framework by adjusting the meta-parameters γ depending on the current situation or state s according to a meta-parameter function $\bar{\gamma}(\mathrm{s})$. The meta-parameter γ is treated as a random variable where the variance correspond to the uncertainty. The state s can for example contain the current position, velocity and acceleration of the robot and external objects, as well as the target to be achieved. This paper focuses on learning the meta-parameter function $\bar{\gamma}(\mathrm{s})$ by episodic reinforcement learning.

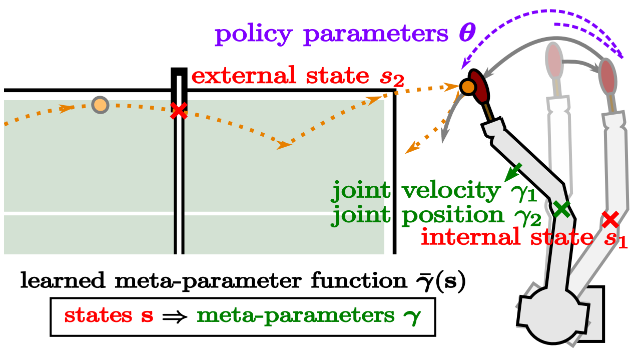

policy parameters $\boldsymbol{\theta}$
external state s_2
joint velocity γ_1
joint position γ_2
internal state s_1
learned meta-parameter function $\bar{\boldsymbol{\gamma}}(\mathbf{s})$
states $\mathbf{s} \Rightarrow$ meta-parameters $\boldsymbol{\gamma}$

Fig. 1: This figure illustrates a table tennis task. The situation, described by the state s, corresponds to the positions and velocities of the ball and the robot at the time the ball is above the net. The meta-parameters γ are the joint positions and velocity at which the ball is hit. The policy parameters represent the backward motion and the movement on the arc. The meta-parameter function $\bar{\gamma}(\mathrm{s})$, which maps the state to the meta-parameters, is learned.

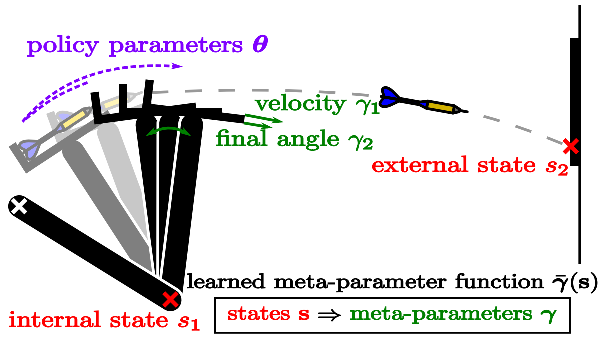

policy parameters $\boldsymbol{\theta}$
velocity γ_1
final angle γ_2
external state s_2
learned meta-parameter function $\bar{\boldsymbol{\gamma}}(\mathbf{s})$
internal state s_1
states $\mathbf{s} \Rightarrow$ meta-parameters $\boldsymbol{\gamma}$

Fig. 2: This figure illustrates a 2D dart throwing task. The situation, described by the state s corresponds to the relative height. The meta-parameters γ are the velocity and the angle at which the dart leaves the launcher. The policy parameters represent the backward motion and the movement on the arc. The meta-parameter function $\bar{\gamma}(\mathrm{s})$, which maps the state to the meta-parameters, is learned.

Illustrations of the Learning Problem

We discuss the resulting learning problem based on the two examples shown in Figures 1 and 2.

As a first illustration of the meta-parameter learning problem, we take a table tennis task which is illustrated in Figure 1 (in Section 3.3, we will expand this example to a robot application). Here, the desired skill is to return a table tennis ball. The motor primitive corresponds to the hitting movement. When modeling a single hitting movement with dynamical-systems motor primitives (Ijspeert et al, 2002), the combination of retracting and hitting motions would be represented by one movement primitive and can be learned by determining the movement parameters $\boldsymbol{\theta}$. These parameters can either be estimated by imitation learning or acquired by reinforcement learning. The return can be adapted by changing the paddle position and velocity at the hitting point. These variables can be influenced by modifying the meta-parameters of the motor primitive such as the final joint positions and velocities. The state consists of the current positions and velocities of the ball and the robot at the time the ball is directly above the net. The meta-parameter function $\bar{\gamma}(\mathrm{s})$ maps the state (the state of the ball and the robot before the return) to the meta-parameters γ (the final positions and velocities of the motor primitive). Its variance corresponds to the uncertainty of the mapping.

In a 2D dart throwing task with a dart on a launcher which is illustrated in Figure 2 (in Section 3.2, we will expand this example to a robot application) the desired skill is to hit a specified point on a wall with a dart. The dart is placed on the launcher and held there by friction. The motor primitive corresponds to the throwing of the dart. When modeling a single dart's movement with dynamical-systems motor primitives (Ijspeert et al, 2002), the combination of retracting and throwing motions would be represented by the movement parameters θ of one movement primitive. The dart's impact position can be adapted to a desired target by changing the velocity and the angle at which the dart leaves the launcher. These variables can be influenced by changing the meta-parameters of the motor primitive such as the final position of the launcher and the duration of the throw. The state consists of the current position of the hand and the desired position on the target. If the thrower is always at the same distance from the wall the two positions can be equivalently expressed as the vertical distance. The meta-parameter function $\bar{\gamma}(\mathrm{s})$ maps the state (the relative height) to the meta-parameters γ (the final position g and the duration of the motor primitive T).

The approach presented in this paper is applicable to any movement representation that has meta-parameters, i.e., a small set of parameters that allows to modify the movement. In contrast to (Lampariello et al, 2011; Jetchev and Toussaint, 2009; Grimes and Rao, 2008; Bentivegna et al, 2004) our approach does not require explicit (re-)planning of the motion.

In the next sections, we derive and apply an appropriate reinforcement learning algorithm.

2.2 Problem Statement: Meta-Parameter Self-Improvement

The problem of meta-parameter learning is to find a stochastic policy $\pi(\gamma|\mathrm{x}) = p(\gamma|\mathrm{s})$ that maximizes the expected return

$$J(\pi) = \int_{\mathbb{S}} p(\mathrm{s}) \int_{\mathbb{G}} \pi(\gamma|\mathrm{s}) R(\mathrm{s},\gamma) d\gamma d\mathrm{s}, \quad (5)$$

where $\mathbb{S}$ denotes the the space of states s, $\mathbb{G}$ denotes the the space of meta-parameters γ, and $R(\mathrm{s},\gamma)$ denotes all the rewards following the selection of the meta-parameter γ according to a situation described by state s. Such a policy $\pi(\gamma|\mathrm{x})$ is a probability distribution over meta-parameters given the current state. The stochastic formulation allows a natural incorporation of exploration, and the optimal time-invariant policy has been shown to be stochastic in the case of hidden state variables (Sutton et al, 1999; Jaakkola et al, 1993). The return of an episode is $R(\mathrm{s},\gamma) = T^{-1}\sum_{t=0}^{T} r^t$ with number of steps T and rewards r^t. For a parametrized policy π with parameters w it is natural to first try a policy gradient approach such as finite-difference methods, vanilla policy gradient approaches and natural gradients. While we will denote the shape parameters by θ, we denote the parameters of the meta-parameter function by w. Reinforcement learning of the meta-parameter function $\bar{\gamma}(\mathrm{s})$ is not straightforward as only few examples can be generated on the real system and trials are often quite expensive. The credit assignment problem is non-trivial as the whole movement is affected by every change in the meta-parameter function. Early attempts using policy gradient approaches resulted in tens of thousands of trials even for simple toy problems, which is not feasible on a real system.

Dayan and Hinton (1997) showed that an immediate reward can be maximized by instead minimizing the Kullback-Leibler divergence $D(\pi(\gamma|\mathrm{s})R(\mathrm{s},\gamma)||\pi'(\gamma|\mathrm{s}))$ between the reward-weighted policy $\pi(\gamma|\mathrm{s})$ and the new policy $\pi'(\gamma|\mathrm{s})$. As we are in an episodic setting, this form of optimization solves the considered problem. Williams (1992) suggested to use Gaussian noise in this context; hence, we employ a policy of the form

$$\pi(\gamma|\mathrm{s}) = \mathcal{N}(\gamma|\bar{\gamma}(\mathrm{s}), \sigma^2(\mathrm{s})\mathrm{I}),$$

where we have the deterministic mean policy $\bar{\gamma}(\mathrm{s}) = \phi(\mathrm{s})^{\mathrm{T}}\mathrm{w}$ with basis functions $\phi(\mathrm{s})$ and parameters w as well as the variance $\sigma^2(\mathrm{s})$ that determines the exploration $\varepsilon \sim \mathcal{N}(0, \sigma^2(\mathrm{s})\mathrm{I})$ as e.g., in (Peters and Schaal, 2008b). The parameters w can then be adapted by reward-weighted regression in an immediate reward (Peters and Schaal, 2008a) or episodic reinforcement learning scenario (Kober and Peters, 2011b). The reasoning behind this reward-weighted regression is that the reward can be treated as an improper probability distribution over indicator variables determining whether the action is optimal or not.

2.3 A Task-Appropriate Reinforcement Learning Algorithm

Designing good basis functions is challenging, a nonparametric representation is better suited in this context. There is an intuitive way of turning the reward-weighted regression into a Cost-regularized Kernel Regression. The kernelization of the reward-weighted regression can be done straightforwardly (similar to Section 6.1 of (Bishop, 2006) for regular supervised learning). Inserting the reward-weighted regression solution $\mathrm{w} = (\Phi^{\mathrm{T}}\mathrm{R}\Phi + \lambda \mathrm{I})^{-1}\Phi^{\mathrm{T}}\mathrm{R}\Gamma_i$ and using the Woodbury formula[2] (Welling, 2010), we transform reward-weighted regression into a Cost-regularized Kernel Regression

$$\begin{aligned}\bar{\gamma}_i &= \phi(\mathrm{s})^{\mathrm{T}}\mathrm{w} = \phi(\mathrm{s})^{\mathrm{T}}\left(\Phi^{\mathrm{T}}\mathrm{R}\Phi + \lambda\mathrm{I}\right)^{-1}\Phi^{\mathrm{T}}\mathrm{R}\Gamma_i \\ &= \phi(\mathrm{s})^{\mathrm{T}}\Phi^{\mathrm{T}}\left(\Phi\Phi^{\mathrm{T}} + \lambda\mathrm{R}^{-1}\right)^{-1}\Gamma_i, \qquad (6)\end{aligned}$$

where the rows of Φ correspond to the basis functions $\phi(\mathrm{s}_i) = \Phi_i$ of the training examples, Γ_i is a vector containing the training examples for meta-parameter component γ_i, and λ is a ridge factor. Next, we assume that the accumulated rewards R_k are strictly positive $R_k > 0$ and can be transformed into costs by $c_k = 1/R_k$. Hence, we have a cost matrix $\mathrm{C} = \mathrm{R}^{-1} = \mathrm{diag}(R_1^{-1}, \dots, R_n^{-1})$ with the cost of all n data points. After replacing $\mathrm{k}(\mathrm{s}) = \phi(\mathrm{s})^{\mathrm{T}}\Phi^{\mathrm{T}}$ and $\mathrm{K} = \Phi\Phi^{\mathrm{T}}$, we obtain the Cost-regularized Kernel Regression

$$\bar{\gamma}_i = \bar{\gamma}_i(\mathrm{s}) = \mathrm{k}(\mathrm{s})^{\mathrm{T}}(\mathrm{K} + \lambda\mathrm{C})^{-1}\Gamma_i,$$

which gives us a deterministic policy. Here, costs correspond to the uncertainty about the training examples. Thus, a high cost is incurred for being further away from the desired optimal solution at a point. In our formulation, a high cost therefore corresponds to a high uncertainty of the prediction at this point.

In order to incorporate exploration, we need to have a stochastic policy and, hence, we need a predictive distribution. This distribution can be obtained by performing the policy update with a Gaussian process regression and we directly see from the kernel ridge regression that

$$\sigma^2(\mathrm{s}) = k(\mathrm{s},\mathrm{s}) + \lambda - \mathrm{k}(\mathrm{s})^{\mathrm{T}}(\mathrm{K} + \lambda\mathrm{C})^{-1}\mathrm{k}(\mathrm{s}),$$

where $k(\mathrm{s},\mathrm{s}) = \phi(\mathrm{s})^{\mathrm{T}}\phi(\mathrm{s})$ is the norm of the point in the kernel space. We call this algorithm Cost-regularized Kernel Regression. Algorithm 1 describes the complete learning procedure, where the rows of S correspond to the states of the training examples $\mathrm{s}_i = \mathrm{S}_i$.

The algorithm corresponds to a Gaussian process regression where the costs on the diagonal are input-dependent noise priors. The parameter λ acts as a exploration-exploitation trade-off parameter as illustrated in Figure 6. Gaussian processes have been used previously for reinforcement learning (Engel et al, 2005) in value function based approaches while here we use them to learn the policy.

[2] The equality $(\Phi^{\mathrm{T}}\mathrm{R}\Phi + \lambda\mathrm{I})^{-1}\Phi^{\mathrm{T}}\mathrm{R} = \Phi^{\mathrm{T}}(\Phi\Phi^{\mathrm{T}} + \lambda\mathrm{R}^{-1})^{-1}$ is straightforward to verify by left and right multiplying the non-inverted terms: $\Phi^{\mathrm{T}}\mathrm{R}(\Phi\Phi^{\mathrm{T}} + \lambda\mathrm{R}^{-1}) = (\Phi^{\mathrm{T}}\mathrm{R}\Phi + \lambda\mathrm{I})\Phi^{\mathrm{T}}$.

Algorithm 1: Meta-Parameter Learning

```
Preparation steps:
  Learn one or more motor primitives by imitation and/or
      reinforcement learning (yields shape parameters $\theta$).
  Determine initial state $s^0$, meta-parameters $\gamma^0$, and cost $C^0$
      corresponding to the initial motor primitive.
  Initialize the corresponding matrices $S, \Gamma, C$.
  Choose a kernel $k, K$.
  Set a scaling parameter $\lambda$.
For all iterations $j$:
  Determine the state $s^j$ specifying the situation.
  Calculate the meta-parameters $\gamma^j$ by:
      Determine the mean of each meta-parameter $i$
          $\bar{\gamma}_i(s^j) = k(s^j)^T (K + \lambda C)^{-1} \Gamma_i$,
      Determine the variance
          $\sigma^2(s^j) = k(s^j, s^j) - k(s^j)^T (K + \lambda C)^{-1} k(s^j)$,
      Draw the meta-parameters from a Gaussian distribution
          $\gamma^j \sim \mathcal{N}(\gamma | \bar{\gamma}(s^j), \sigma^2(s^j) I)$.
  Execute the motor primitive using the new meta-parameters.
  Calculate the cost $c^j$ at the end of the episode.
  Update $S, \Gamma, C$ according to the achieved result.
```

2.4 Meta-Parameter Learning by Reinforcement Learning

As a result of Section 2.3, we have a framework of motor primitives as introduced in Section 2.1 that we can use for reinforcement learning of meta-parameters as outlined in Section 2.2. We have generalized the reward-weighted regression policy update to instead become a Cost-regularized Kernel Regression (CrKR) update where the predictive variance is used for exploration. In Algorithm 1, we show the complete algorithm resulting from these steps.

The algorithm receives three inputs, i.e., (i) a motor primitive that has associated meta-parameters γ, (ii) an initial example containing state s^0, meta-parameter γ^0 and cost C^0, as well as (iii) a scaling parameter λ. The initial motor primitive can be obtained by imitation learning (Ijspeert et al, 2002) and, subsequently, improved by parametrized reinforcement learning algorithms such as policy gradients (Peters and Schaal, 2008b) or Policy learning by Weighting Exploration with the Returns (PoWER) (Kober and Peters, 2011b). The demonstration also yields the initial example needed for meta-parameter learning. While the scaling parameter is an open parameter, it is reasonable to choose it as a fraction of the average cost and the output noise parameter (note that output noise and other possible hyper-parameters of the kernel can also be obtained by approximating the unweighted meta-parameter function).

Illustration of the Algorithm

In order to illustrate this algorithm, we will use the example of the table tennis task introduced in Section 2.1. Here, the robot should hit the ball accurately while not destroying its mechanics. Hence, the cost could correspond to the distance between the ball and the paddle, as well as the squared torques. The initial policy is based on a prior, illustrated in

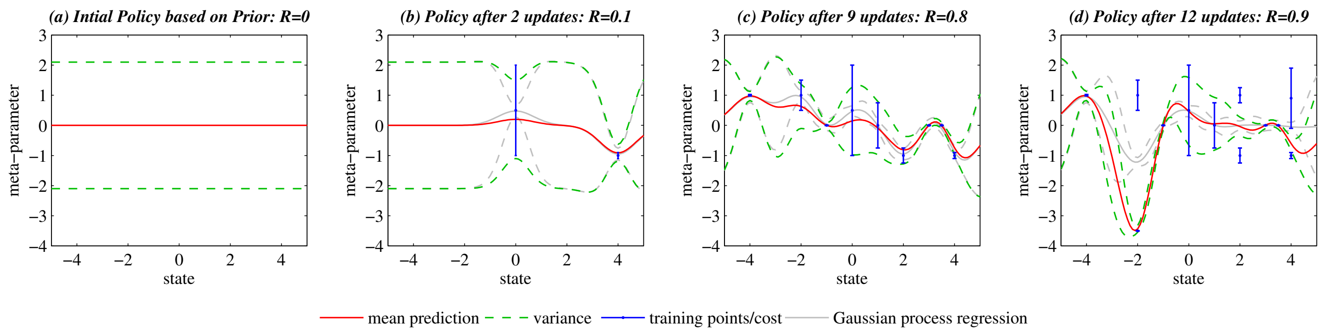


Fig. 3: This figure illustrates the meaning of policy improvements with Cost-regularized Kernel Regression. Each sample consists of a state, a meta-parameter and a cost where the cost is indicated the blue error bars. The red line represents the improved mean policy, the dashed green lines indicate the exploration/variance of the new policy. For comparison, the gray lines show standard Gaussian process regression. As the cost of a data point is equivalent to having more noise, pairs of states and meta-parameter with low cost are more likely to be reproduced than others with high costs.

Figure 3(a), that has a variance for initial exploration (it often makes sense to start with a uniform prior). This variance is used to enforce exploration. To return a ball, we sample the meta-parameters from the policy based on the current state. After the trial the cost is determined and, in conjunction with the employed meta-parameters, used to update the policy. If the cost is large (e.g., the ball was far from the racket), the variance of the policy is large as it may still be improved and therefore needs exploration. Furthermore, the mean of the policy is shifted only slightly towards the observed example as we are uncertain about the optimality of this action. If the cost is small, we know that we are close to an optimal policy (e.g., the racket hit the ball off-center) and only have to search in a small region around the observed trial. The effects of the cost on the mean and the variance are illustrated in Figure 3(b). Each additional sample refines the policy and the overall performance improves (see Figure 3(c)). If a state is visited several times and different meta-parameters are sampled, the policy update must favor the meta-parameters with lower costs. If several sets of meta-parameters have similarly low costs, where it converges depends on the order of samples. The cost function should be designed to avoid this behavior and to favor a single set. The exploration has to be restricted to safe meta-parameter ranges. Algorithm 1 exhibits this behavior as the exploration is only local and restricted by the prior (see Figure 3). If the initial policy is safe, exploring the neighboring regions is likely to be safe as well. Additionally, lower level controllers as well as the mechanics of the robot ensure that kinematic and dynamic constrains are satisfied and a term in the cost function can be used to discourage potentially harmful movements.

In the example of the 2D dart throwing task, the cost is similar. Here, the robot should throw darts accurately while not destroying its mechanics. Hence, the cost corresponds to the error between desired goal and the impact point, as well as the absolute velocity of the end-effector. Often the state is determined by the environment, e.g., the ball trajectory in table tennis depends on the opponent. However, for the dart setting, we could choose the next target and thus employ CrKR as an active learning approach by picking states with large variances. In the dart throwing example we have a correspondence between the state and the outcome similar to a regression problem. However, the mapping between the state and the meta-parameter is not unique. The same height can be achieved by different combinations of velocities and angles. Averaging these combinations is likely to generate inconsistent solutions. The regression must hence favor the meta-parameters with the lower costs. CrKR can be employed as a regularized regression method in this setting.

3 Evaluations and Experiments

In Section 2, we have introduced both a framework for meta-parameter self-improvement as well as an appropriate reinforcement learning algorithm used in this framework. In this section, we will first show that the presented reinforcement learning algorithm yields higher performance than off-the shelf approaches. Hence, we compare it on a simple planar cannon shooting problem (Lawrence et al, 2003) with the preceding reward-weighted regression, an off-the-shelf finite difference policy gradient approach, and show the advantages over supervised learning approaches.

The resulting meta-parameter learning framework can be used in a variety of settings in robotics. We consider three scenarios here, i.e., (i) dart throwing with a simulated Barrett WAM, a real Kuka KR 6, and the JST-ICORP/SARCOS humanoid robot CBi (Cheng et al, 2007), (ii) table tennis with a simulated robot arm and a real Barrett WAM, and

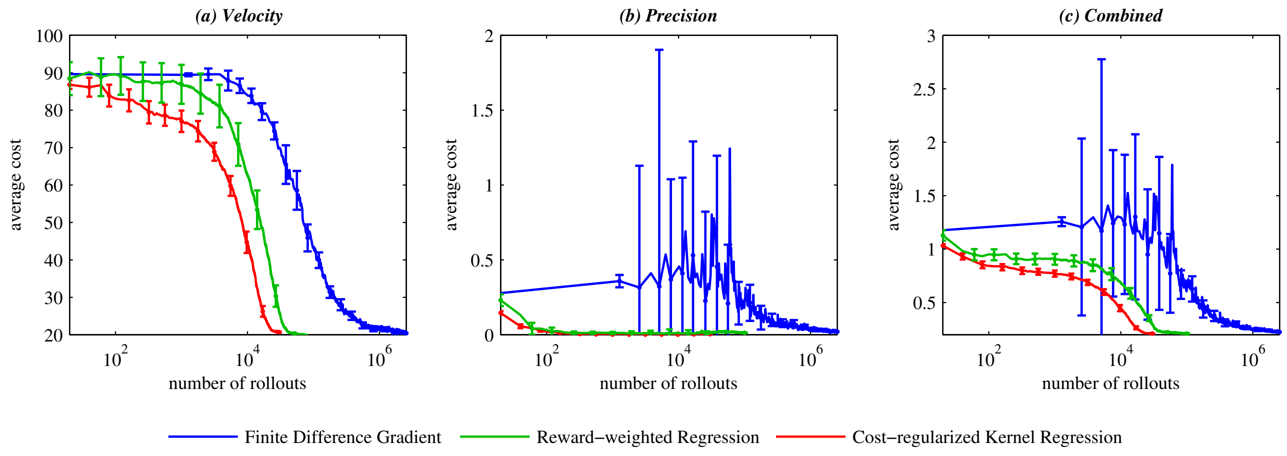


Fig. 4: This figure shows the performance of the compared algorithms averaged over 10 complete learning runs. Cost-regularized Kernel Regression finds solutions with the same final performance two orders of magnitude faster than the finite difference gradient (FD) approach and twice as fast as the reward-weighted regression. At the beginning FD often is highly unstable due to our attempts of keeping the overall learning speed as high as possible to make it a stronger competitor. The lines show the median and error bars indicate standard deviation. The initialization and the initial costs are identical for all approaches. However, the omission of the first twenty rollouts was necessary to cope with the logarithmic rollout axis. The number of rollouts includes the rollouts not used to update the policy.

(iii) throwing a ball at targets with a MATLAB simulation and a real BioRob (Lens et al, 2010).

3.1 Benchmark Comparison: Toy Cannon Shots

In the first task, we only consider a simple simulated planar cannon shooting where we benchmark our Reinforcement Learning by Cost-regularized Kernel Regression approach against a finite difference gradient estimator and the reward-weighted regression. Additionally we contrast our reinforcement learning approach to a supervised one. Here, we want to learn an optimal policy for a 2D toy cannon environment similar to (Lawrence et al, 2003). This benchmark example serves to illustrate out approach and to compare it to various previous approaches.

The setup is given as follows: A toy cannon is at a fixed location $[0.0, 0.1]\,m$. The trajectory of the cannon ball depends on the angle with respect to the ground and the speed at which it leaves the cannon. The flight of the canon ball is simulated as ballistic flight of a point mass with Stokes's drag as wind model. The cannon ball is supposed to hit the ground at a desired distance. The desired distance $[1..3]\,m$ and the wind speed $[0..1]\,m/s$, which is always horizontal, are used as input states, the velocities in horizontal and vertical directions are the meta-parameters (which influences the angle and the speed of the ball leaving the cannon). In this benchmark we do not employ the motor primitives but set the meta-parameters directly. Lower speed can be compensated by a larger angle. Thus, there are different possible policies for hitting a target; we intend to learn the one which is optimal for a given cost function. This cost function is defined as

$$c = (b_x - s_x)^2 + 0.01\left(\dot{b}_x^2 + \dot{b}_z^2\right),$$

where b_x is the impact position on the ground, s_x the desired impact position as indicated by the state, and $\dot{b}_{\{x,z\}}$ are the horizontal and vertical velocities of the cannon ball at the impact point respectively. It corresponds to maximizing the precision while minimizing the employed energy according to the chosen weighting. The input states (desired distance and wind speed) are drawn from a uniform distribution and directly passed to the algorithms. All approaches performed well in this setting, first driving the position error to zero and, subsequently, optimizing the impact velocity. The experiment was initialized with $[1, 10]\,m/s$ as initial ball velocities and $1\,m/s$ as wind velocity. This setting corresponds to a very high parabola, which is far from optimal. For plots, we evaluate the policy on a test set of 25 uniformly randomly chosen points that remain the same throughout of the experiment and are never used in the learning process but only to generate Figure 4.

We compare our novel algorithm to a finite difference policy gradient (FD) method (Peters and Schaal, 2008b) and to the reward-weighted regression (RWR) (Peters and Schaal, 2008a). The FD method uses a parametric policy that employs radial basis functions in order to represent the policy and perturbs the parameters. We used 25 Gaussian basis functions on a regular grid for each meta-parameter, thus a total of 50 basis functions. The number of basis functions,

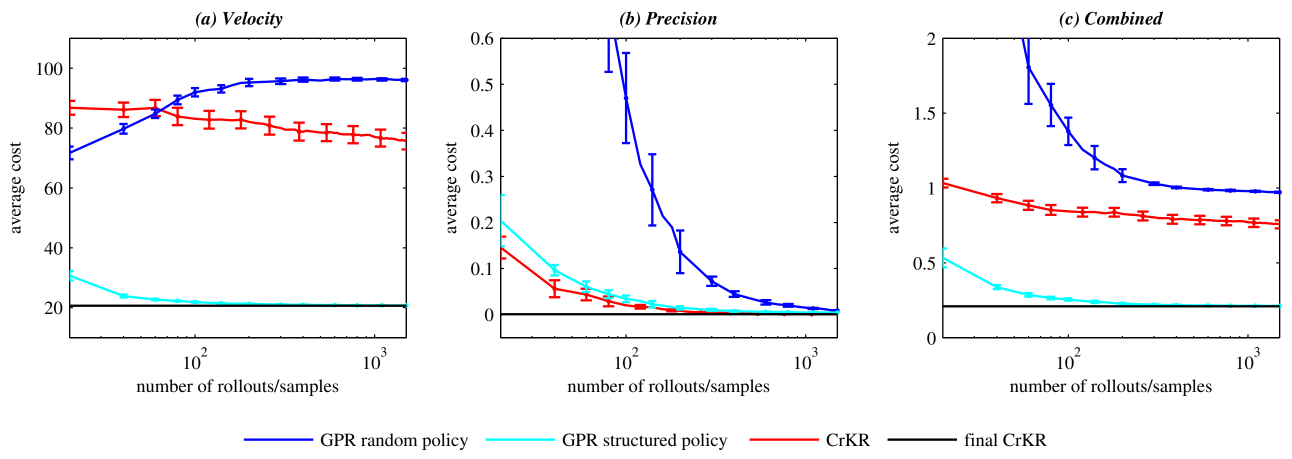


Fig. 5: In this figure, we compare Gaussian process regression (GPR) in a supervised learning setting as proposed by (Ude et al, 2010; Kronander et al, 2011) to Cost-regularized Kernel Regression (CrKR) in a reinforcement learning setting. The red curve corresponds to the red curve (Cost-regularized Kernel Regression) in Figure 4. The GPR is trained with samples from the prior used for the CrKR (blue line) and with samples of the final CrKR policy (cyan line) respectively. The black line indicates the cost after CrKR has converged. GPR with samples drawn from the final policy performs best. Please note that this comparison is contrived as the role of CrKR is to discover the policy that is provided to “GPR structured policy”. GPR can only reproduce the demonstrated policy, which is achieved perfectly with 1000 samples. GPR can reproduce the demonstrated policy more accurately if more samples are available. However, it cannot improve the policy according to a cost function and it is impacted by contradictory demonstrations. The results are averaged over 10 complete learning runs. The lines show the median and error bars indicate standard deviation. The number of rollouts includes the rollouts not used to update the policy.

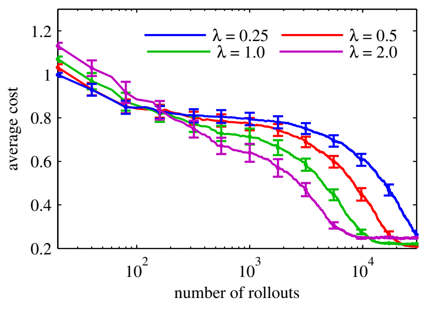
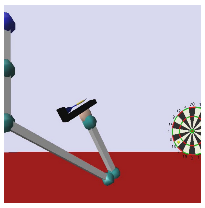


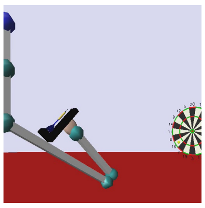
Fig. 6: This figure illustrates the influence of the parameter λ for the Cost-regularized Kernel Regression. The red curve ($\lambda = 0.5$) corresponds to the red curve (Cost-regularized Kernel Regression) in Figure 4(c). The parameter λ trades off the exploration versus the exploitation. A higher λ leads to larger exploration and, thus, faster convergence to a suboptimal solution. The results are averaged over 10 complete learning runs. The lines show the median and error bars indicate standard deviation. The number of rollouts includes the rollouts not used to update the policy.

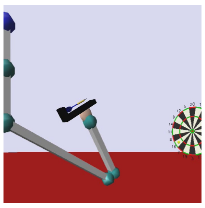
the learning rate, as well as the magnitude of the perturbations were tuned for best performance. We used 51 sets of uniformly perturbed parameters for each update step. The perturbed policies were evaluated on a batch of 25 input parameters to avoid over-fitting on specific input states.The FD algorithm converges after approximately 2000 batch gradient evaluations, which corresponds to 2,550,000 shots with the toy cannon.

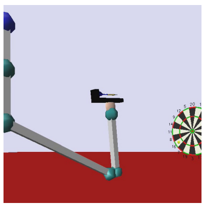
The RWR method uses the same parametric policy as the finite difference gradient method. Exploration is achieved by adding Gaussian noise to the mean policy. All open parameters were tuned for best performance. The reward transformation introduced by Peters and Schaal (2008a) did not improve performance in this episodic setting. The RWR algorithm converges after approximately 40,000 shots with the toy cannon. For the Cost-regularized Kernel Regression (CrKR) the inputs are chosen randomly from a uniform distribution. We use Gaussian kernels and the open parameters were optimized by cross-validation on a small test set prior to the experiment. Each trial is added as a new training point if it landed in the desired distance range. The CrKR algorithm converges after approximately 20,000 shots with the toy cannon. The bandwidth of the kernels used for CrKR is in the same order of magnitude as the bandwidth of the basis functions. However, due to the non-parametric nature of CrKR, narrower kernels can be used to capture more details in order to improve performance. Figure 6 illustrates the influence of the parameter λ for the CrKR.

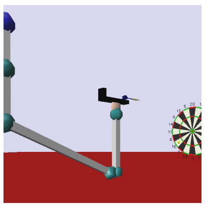
After convergence, the costs of CrKR are the same as for RWR and slightly lower than those of the FD method. The


(a) The dart is placed on the launcher.


(b) The arm moves back.


(c) The arm moves forward on an arc.


(d) The arm stops.


(e) The dart is carried on by its momentum.

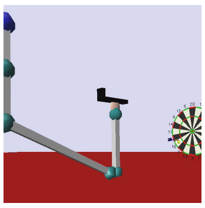
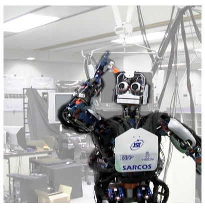
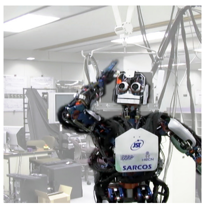
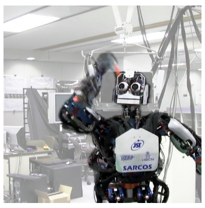
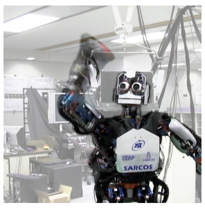

(f) The dart hits the board.

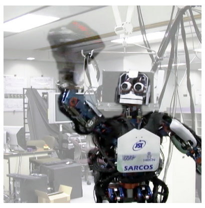
Fig. 7: This figure shows a dart throw in a physically realistic simulation.


(a) The dart is placed in the hand.


(b) The arm moves back.


(c) The arm moves forward on an arc.


(d) The arm continues moving.


(e) The dart is released and the arm follows through.

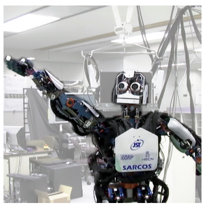

(f) The arm stops and the dart hits the board.

Fig. 8: This figure shows a dart throw on the real JST-ICORP/SARCOS humanoid robot CBi.

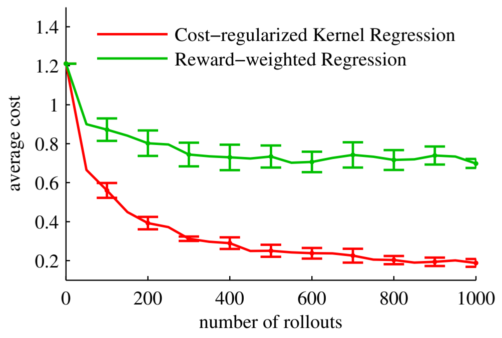


Fig. 9: This figure shows the cost function of the dart-throwing task for a whole game *Around the Clock* in each rollout. The costs are averaged over 10 runs with the error-bars indicating standard deviation. The number of rollouts includes the rollouts not used to update the policy.

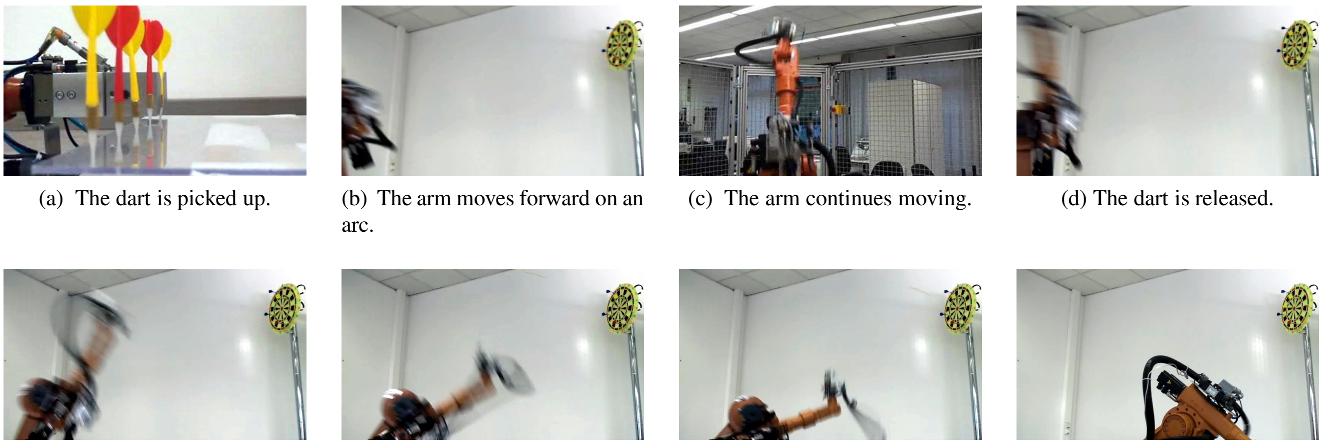
CrKR method needs two orders of magnitude fewer shots than the FD method. The RWR approach requires twice the shots of CrKR demonstrating that a non-parametric policy, as employed by CrKR, is better adapted to this class of problems than a parametric policy. The squared error between the actual and desired impact is approximately 5 times higher for the finite difference gradient method, see Figure 4.

Compared to standard Gaussian process regression (GPR) in a supervised setting, CrKR can improve the policy over time according to a cost function and outperforms GPR in settings where different combinations of meta-parameters yield the same result. For details, see Figure 5.

3.2 Robot Dart-Throwing Games

Now, we turn towards the complete framework, i.e., we intend to learn the meta-parameters for motor primitives in discrete movements. We compare the Cost-regularized Kernel Regression (CrKR) algorithm to the reward-weighted regression (RWR). As a sufficiently complex scenario, we chose a robot dart throwing task inspired by (Lawrence et al, 2003). However, we take a more complicated scenario and choose dart games such as *Around the Clock* (Masters Games Ltd., 2010) instead of simple throwing at a fixed location. Hence, it will have an additional parameter in the state depending on the location on the dartboard that should come next in the sequence. The acquisition of a basic motor primitive is achieved using previous work on imitation learning (Ijspeert et al, 2002). Only the meta-parameter function is learned using CrKR or RWR. For the learning process, the targets (which are part of the state) are uniformly distributed on the dartboard. For the evaluation the targets are placed in the center of the fields. The reward is calculated based on the impact position observed by a vision system in the real robot experiments or the simulated impact position.

The dart is placed on a launcher attached to the end-effector and held there by stiction. We use the Barrett WAM


(a) The dart is picked up. (b) The arm moves forward on an arc. (c) The arm continues moving. (d) The dart is released.

(e) The arm follows through. (f) The arm continues moving. (g) The arm returns to the pick-up position. (h) The dart has hit the board.

Fig. 10: This figure shows a dart throw on the real Kuka KR 6 robot.

robot arm in order to achieve the high accelerations needed to overcome the stiction. See Figure 7, for a complete throwing movement. The motor primitive is trained by imitation learning with kinesthetic teach-in. We use the Cartesian coordinates with respect to the center of the dart board as input states. In comparison with the benchmark example, we cannot directly influence the release velocity in this setup. Hence, we employ the parameter for the final position g, the time scale of the motor primitive τ and the angle around the vertical axis (i.e., the orientation towards the dart board to which the robot moves before throwing) as meta-parameters instead. The popular dart game *Around the Clock* requires the player to hit the numbers in ascending order, then the bulls-eye. As energy is lost overcoming the stiction of the launching sled, the darts fly lower and we placed the dartboard lower than official rules require. The cost function is defined as

$$c = 10\sqrt{\sum_{i\in\{x,z\}} (d_i - s_i)^2} + \tau,$$

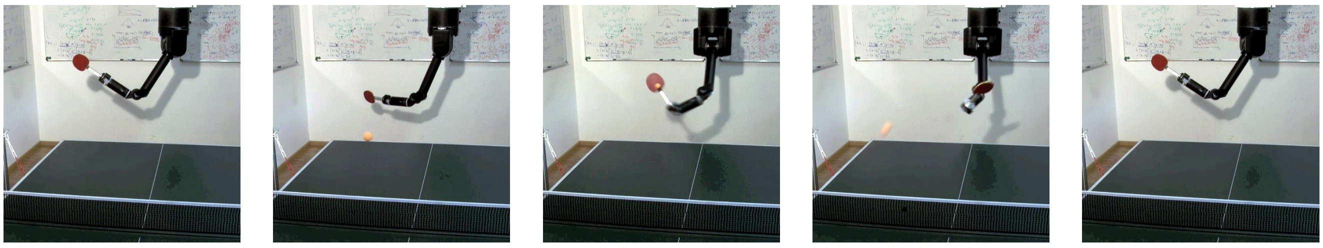
where d_i are the horizontal and vertical positions of the dart on the dartboard after the throw, s_i are the horizontal and vertical positions of the target corresponding to the state, and τ corresponds to the velocity of the motion. After approximately 1000 throws the algorithms have converged but CrKR yields a high performance already much earlier (see Figure 9). We again used a parametric policy with radial basis functions for RWR. Here, we employed 225 Gaussian basis function on a regular grid per meta-parameter. Designing a good parametric policy proved very difficult in this setting as is reflected by the poor performance of RWR.

This experiment has also being carried out on three real, physical robots, i.e., a Barrett WAM, the humanoid robot CBi (JST-ICORP/SARCOS), and a Kuka KR 6. CBi was developed within the framework of the JST-ICORP Computational Brain Project at ATR Computational Neuroscience Labs. The hardware of the robot was developed by the American robotic development company SARCOS. CBi can open and close the fingers which helps for more human-like throwing instead of the launcher employed by the Barrett WAM. See Figure 8 for a throwing movement.

We evaluated the approach on a setup using the Kuka KR 6 robot and a pneumatic gripper. The robot automatically picks up the darts from a stand. The position of the first degree of freedom (horizontal position) as well as the position of the fifth degree of freedom and the release timing (vertical position) were controlled by the algorithm. Due to inaccurate release timing the vertical position varied in a range of $10cm$. Additionally the learning approach had to cope with non-stationary behavior as the outcome of the same set of parameters changed by one third of the dart board diameter upward. Despite these additional complications the robot learned to reliably (within the reproduction accuracy of $10cm$ as noted above) hit all positions on the dart board using only a total of 260 rollouts. See Figure 10 for a throwing movement.

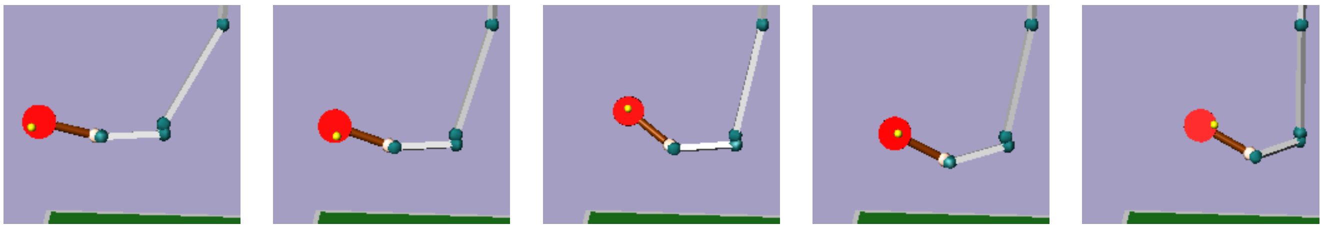
3.3 Robot Table Tennis

In the second evaluation of the complete framework, we use the proposed method for hitting a table tennis ball in the air. The setup consists of a ball gun that serves to the forehand of the robot, a Barrett WAM and a standard sized table. The movement of the robot has three phases. The robot is in a rest posture and starts to swing back when the ball is launched. During this swing-back phase, the open parameters for the stroke are to be learned. The second phase is the hitting



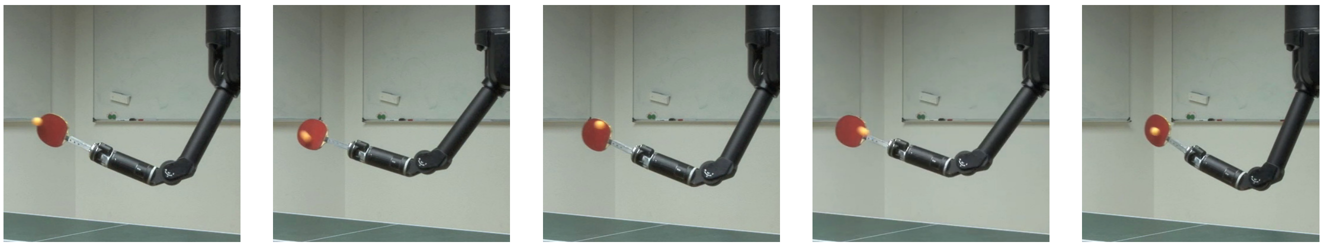
(a) The robot is in the rest posture. (b) The arm swings back. (c) The arm strikes the ball. (d) The arm follows through and decelerates. (e) The arm returns to the rest posture.

Fig. 11: This figure shows the phases of a table tennis stroke on the real Barrett WAM.



(a) Left. (b) Half left. (c) Center high. (d) Center low. (e) Right.

Fig. 12: This figure shows samples of the learned forehands. Note that this figure only illustrates the learned meta-parameter function in this context but cannot show timing (see Figure 14) and velocity and it requires a careful observer to note the important configuration differences resulting from the meta-parameters.



(a) Left high. (b) Left low. (c) Center high. (d) Center low. (e) Right.

Fig. 13: This figure shows samples of the learned forehands on the real robot.

phase which ends with the contact of the ball and racket. In the final phase, the robot gradually ends the stroking motion and returns to the rest posture. See Figure 11 for an illustration of a complete episode and (Kober et al, 2010a) for a more detailed description. The movements in the three phases are represented by three motor primitives obtained by imitation learning. We only learn the meta-parameters for the hitting phase.

The meta-parameters are the joint positions g and velocities ġ for all seven degrees of freedom at the end of the second phase (the instant of hitting the ball) and a timing parameter t_{hit} that controls when the swing back phase is transitioning to the hitting phase. For this task we employ a variant of the motor primitives that allows to set non-zero end velocities (Kober et al, 2010a). We learn these 15 meta-parameters as a function of the state, which corresponds to the ball positions and velocities when it is directly over the net. We employed a Gaussian kernel and optimized the open kernel parameters according to typical values for the input and output beforehand. As cost function we employ

$$c = \sqrt{\sum_{i\in\{x,y,z\}} \left(b_i\left(t_{\text{hit}}\right) - p_i\left(t_{\text{hit}}\right)\right)^2},$$

where $b_i\left(t_{\text{hit}}\right)$ are the Cartesian positions of the ball and $p_i\left(t_{\text{hit}}\right)$ are the Cartesian positions of the center of the paddle, both at the predicted hitting time t_{hit}. The policy is evaluated every 50 episodes with 25 ball launches picked randomly at the beginning of the learning. We initialize the behavior with five successful strokes observed from another

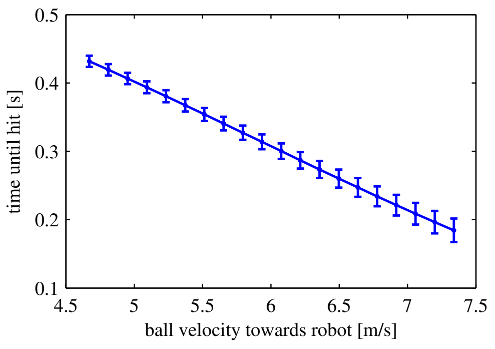


Fig. 14: This figure illustrates the effect of the velocity of the ball towards the robot on the time it has until the ball needs to be hit. The plot was generated by sweeping through the velocity component towards the robot, keeping the other position and velocity values fixed. The line is the mean of 100 sweeps drawn from the same ball distribution as used during the learning.

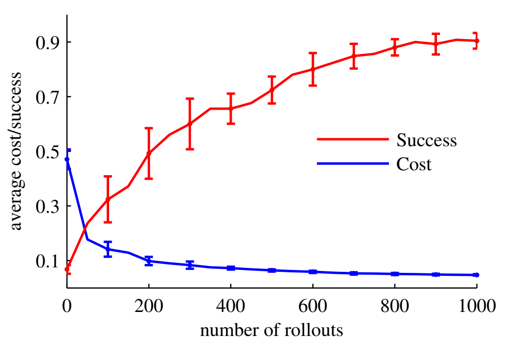


Fig. 15: This figure shows the cost function of the simulated table tennis task averaged over 10 runs with the error-bars indicating standard deviation. The red line represents the percentage of successful hits and the blue line the average cost. The number of rollouts includes the rollouts not used to update the policy. At the beginning the robot misses the ball 95% of the episodes and on average by $50\,cm$. At the end of the learning the robot hits almost all balls.

player. After initializing the meta-parameter function with only these five initial examples, the robot misses approximately 95% of the balls as shown in Figure 15. Trials are only used to update the policy if the robot has successfully hit the ball as they did not significantly improve the learning performance and in order to keep the calculation sufficiently fast. Figures 12 and 13 illustrate different positions of the ball the policy is capable of dealing with after the learning. Figure 14 illustrates the dependence of the timing parameter on the ball velocity towards the robot and Figure 15 illustrates the costs over all episodes. For the results in Figure 15, we have simulated the flight of the ball as a simple ballistic point mass and the bouncing behavior using a restitution constant for the velocities. The state is directly taken from the simulated ball data with some added Gaussian noise. In the real robot experiment (Figure 16), the ball is shot with a ball cannon. The position of the ball is determined by two pairs of stereo cameras and the velocity is obtained by numerical differentiation. In this second setting, the state information is a lot less reliable due to noise in the vision system and even the same observed state can lead to different outcomes due to unobserved spin.

3.4 Active Learning of Ball Throwing

As an active learning setting, we chose a ball throwing task where the goal is to improve the throws while trying to perform well in a higher level game. For this scenario, it is important to balance learning of the individual actions by practicing them while at the same time, focusing on the overall performance in order to achieve the complete skill. Prominent examples are leisure time activities such as sports or motor skill games. For example, when playing darts with

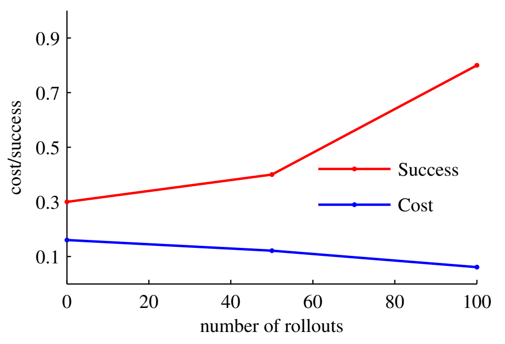


Fig. 16: This figure shows the cost function of the table tennis task on the real robot. The policy was learned entirely on the real robot. The red line represents the percentage of successful hits and the blue line the average cost. The number of rollouts includes the rollouts not used to update the policy. At the beginning the robot misses the ball 70% of the episodes and on average by $15\,cm$. At the end of the learning the robot hits 80% of the balls.

friends, you will neither always attempt the lowest risk action, nor always try to practice one particular throw, which will be valuable when mastered. Instead, you are likely to try plays with a reasonable level of risk and rely on safe throws in critical situations. This exploration is tightly woven into higher order dart games.

The higher level is modeled as a standard reinforcement learning problem with discrete states and actions. The lower level learning is done using CrKR. The higher level determines the target the robot is supposed to hit. The lower level has to learn how to hit this target. The transition probabilities of the higher level can be estimated from the learned meta-

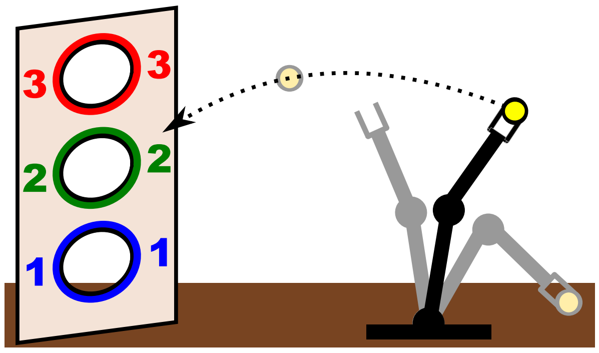


Fig. 17: This figure illustrates the side-stall game. The player throws the ball and if it lands in the target (illustrated by a wall with target holes) gets the number of points written next to it. Missing the targets is not punished, however, going over ten points leads to a loss of ten points.

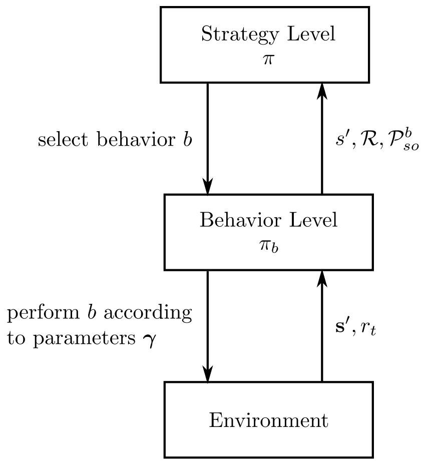


Fig. 18: This figure illustrates the setup of the roles of the different levels.

parameter function as explained in Section 3.4.2. We will discuss the rules of the game in Section 3.4.1, a simulated experiment in Section 3.4.3, and the results of an evaluation with a real BioRob in Section 3.4.4.

3.4.1 Game used for the Evaluations

The game is reminiscent of blackjack as the goal is to collect as many points as possible without going over a threshold. The player throws a ball at three targets. The three rewards of one, two, and three are assigned to one target each. The setup of the game is illustrated in Figure 17. If the ball lands in the target, the player receives the corresponding number of points. The player starts with zero points if he gets more than 10 points he "busts" and incurs a loss of -10. The player has the option to "stand" (i.e., stop throwing and collect the accumulated number of points) at all times. Missing all targets does not entail a cost.

3.4.2 Two-Level Learning Approach

Our framework considers a hierarchy of two levels: a strategy level and a behavior level. The strategy level determines the strategy for the high-level moves, here termed "behaviors", of the game. The behavior level deals with executing these behaviors in an optimal fashion. The strategy level chooses the next behavior, which is then executed by the behavior level. Upon completion of the behavior, the strategy level chooses the next behavior. The setup is illustrated in Figure 18.

We assume that the game has discrete states $s \in \mathbb{S}$ and discrete behaviors $b \in \mathbb{B}$. In the dart setting a behavior could be attempting to hit a specific field and the state could correspond to the current score. Given the current state, each behavior has an associated expected outcome $o \in \mathbb{O}$. For example, the behavior "throw at target X" has the outcome "change score by X" as a result of hitting target X. The transition probabilities $\mathscr{P}^b_{so}$ of the strategy level would express how likely it is to hit a different field. The game can be modeled as an Markov decision process or MDP (Sutton and Barto, 1998), where the states consist of the number of accumulated points (zero to ten) and two additional game states ("bust" and "stand"). The behaviors correspond to attempting to throw at a specific target or to "stand" and are fixed beforehand. We assume to have an episodic game with a finite horizon, which can be expressed equivalently as an infinite horizon problem where we define an absorbing terminal state in which all actions receive an immediate reward of 0.

On the behavior level, we augment the state space with continuous states that describe the robot and the environment to form the combined state space $\mathbf{s}$. This state space could, for example, include the position and velocity of the arm, the position of the targets as well as the current score. The actions are considered to be continuous and could, for example, be the accelerations of the arm. As the strategy level has to wait until the behavior is completed, the behaviors need to be of episodic nature as well. We have a single motor primitive representing the three behaviors of aiming at the three targets. Hitting the desired target is learned using CrKR. We employ Policy Iteration (Sutton and Barto, 1998) to learn on the strategy level.

The rewards for the strategy learning are fixed by the rules of the game. The possible states and behaviors also result from the way the game is played. The missing piece for the strategy learning is the transition probabilities $\mathscr{P}^b_{so}$. The behavior learning by CrKR associates each behavior with a variance. Each of these behaviors correspond to an expected change in state, the outcome o. For example "aim at 2" corresponds to "increase score by 2". However, the meta-parameter function does not explicitly include information

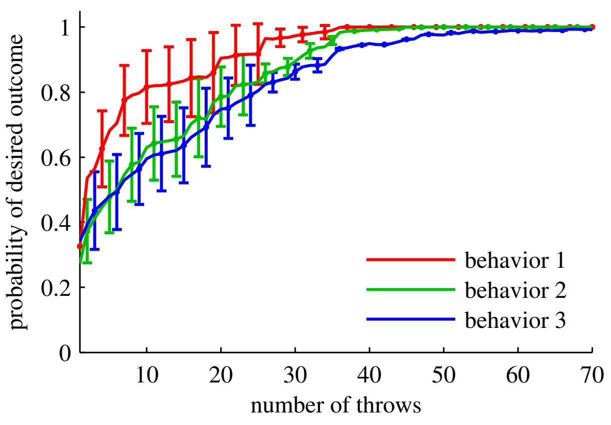


Fig. 19: This figure illustrates the transition probabilities of the three behaviors to their associated outcome in simulation. For example, the red line indicates the probability of gaining one point when throwing at target 1. After approximately 50 throws the player has improved his accuracy level such that he always hits the desired target. The plots are averaged over 10 runs with the error-bars indicating standard deviations.

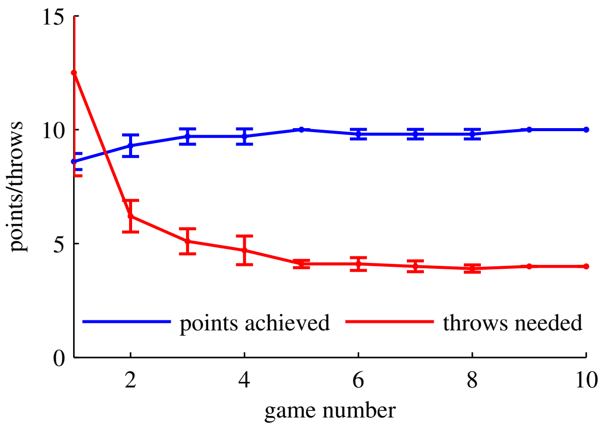


Fig. 20: This figure illustrates the improvement of the player over the number of games in simulation. Due to the large penalty for busting the framework always uses a safe strategy. Already after five completed games the player reaches almost always the maximum possible score of 10. As the number of throws is not punished there are initially many throws that miss the target. After 7 games the number of throws has converged to 4, which is the minimum required number. The plots are averaged over 10 runs with the error-bars indicating standard deviations.

regarding what happens if the expected change in state is not achieved. We assume that there is a discrete set of outcomes $o \in \mathbb{O}$ (i.e., change in state) for all behaviors b for a certain state s. For example in this game hitting each target, and missing, is associated with either increasing the player's score, winning or to bust (i.e., going over ten). With the meta-parameter function, we can calculate the overlaps of the ranges of possible meta-parameters for the different behaviors. These overlaps can then be used to determine how likely it is to end up with a change of state associated with a behavior different from the desired one. This approach relies on the assumption that we know for each behavior the associated range of meta-parameters and their likelihood.

The meta-parameters are drawn according to a normal distribution, thus the overlap has to be weighted accordingly. The probability of the outcome o when performing behavior b can be calculated as follows:

$$\mathscr{P}^b_{so} = \int p^b(\gamma) \frac{p^o(\gamma)}{\sum_{k\in\mathbb{O}} p^k(\gamma)} d\gamma,$$

where γ is the meta-parameters, $p^b(\gamma)$ is the probability of picking the meta-parameter γ when performing behavior b, $p^o(\gamma)$ is the probability of picking the meta-parameter γ when performing the action associated to the considered outcome o, and $\sum_{k\in\mathbb{O}} p^k(\gamma)$ is the normalizing factor. This scenario has first been treated in (Kober and Peters, 2011a).

3.4.3 Evaluation in Simulation

We first evaluated our approach using a MATLAB based simulation. The throw is modeled as a two dimensional ballistic flight of a point mass. The targets correspond to segments of the ground line. The meta-parameters are the initial horizontal and vertical velocities of the ball. The meta-parameters used to initialize the learning make the ball drop in front of the first target. The cost function for the behavior level is

$$c = \sum_{i\in\{x,z\}} \dot{b}_i^2 + (b_x - s_x)^2,$$

where $\dot{b}_i$ are the initial velocities, b_x is the impact position and s_x the desired impact position. The state corresponds to the three targets and is determined by the higher level. Figure 19 illustrates how the player learns to throw more accurately while playing. Figure 20 illustrates how learning to perform the lower level actions more reliably enables the player to perform better in the game.

3.4.4 Evaluation on a real BioRob

We employ a BioRob to throw balls in a catapult like fashion. The arm is approximately $0.75m$ long, and it can reach $1.55m$ above the ground. The targets are located at a distance of $2.5m$ from the robot at a height of $0.9m$, $1.2m$, and $1.5m$

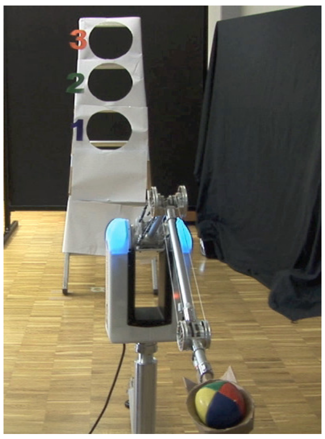


Fig. 21: This figure illustrates the setup of the robot evaluation.

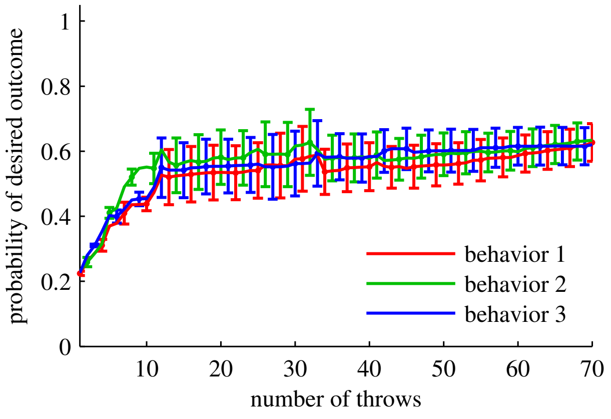


Fig. 22: This figure illustrates the transition probabilities of the three behaviors to their associated outcome like in Figure 19. The skill improves a lot in the first 15 throws after that the improvement levels of. Initially behavior 2, associated with target 2 (which lies in the center) is most likely to succeed. The success rate of 60% corresponds to the level of reproducibility of our setup. The framework manages to handle this large uncertainty by choosing to “stand” early on. The plots are averaged over 4 runs with the error-bars indicating standard deviations.

respectively. The ball is placed in a funnel-shaped receptacle. In this setup, the initial horizontal and vertical velocities of the ball cannot directly be set. Instead, the meta-parameters are defined as the duration and amount of acceleration for two joints that are in the throwing plane. The robot starts in a fixed initial position, accelerates the two joints according to the meta-parameter indicating the magnitude, and accelerates in the opposite direction after the time determined by the other meta-parameter in order to break. Finally the robot returns to the initial position. See Figure 23 for an illustration of one throwing motion. The state corresponds to the three targets and is determined by the higher level. The outcome of the throw is observed by a vision system.

Executing the throw with identical parameters will only land at the same target in approximately 60% of the throws, due to the high velocities involved and small differences in putting the ball in the holder. Thus, the algorithm has to deal with large uncertainties. The cost function for the behavior level is

$$c = \sum_{i\in\{1,2\}} \ddot{\theta}_i^2 + t_{\mathrm{acc}}^2 + (b_x - s_x)^2,$$

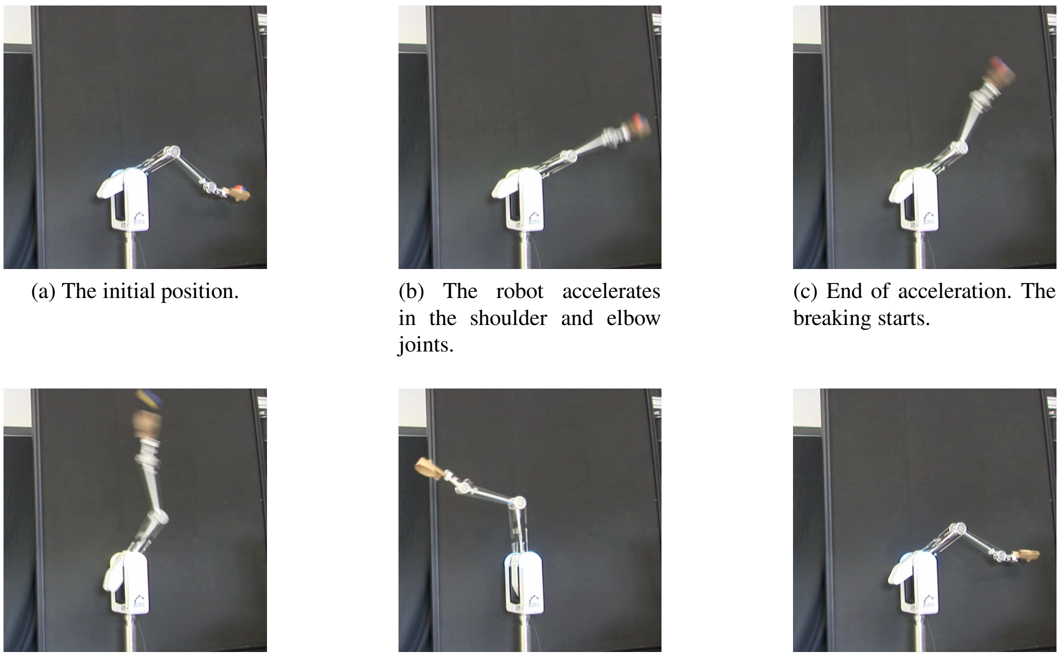
where $\ddot{\theta}_i$ is the acceleration magnitude, t_{acc} the acceleration duration, b_x is the impact position and s_x the desired impact position. The setup makes it intentionally hard to hit target 3. The target can only be hit with a very restricted set of parameters. For targets 1 and 2 increasing the amount of acceleration or the duration will result in a higher hit. Target 3 is at the limit where higher accelerations or longer durations will lead to a throw in a downward direction with a high velocity.

The typical behavior of one complete experiment is as follows: At the beginning the robot explores in a very large area and stands as soon as it reaches a score of 8, 9, or 10. Due to the large punishment it is not willing to attempt to throw at 1 or 2 while having a large uncertainty, and, thus, a high chance of busting. Later on, it has learned that attempting to throw at 2 has a very low chance of ending up in 3 and hence will attempt to throw 2 points if the current score is 8. We setup the policy iteration to favor behaviors with a higher number, if the values of the behaviors are identical. The first throws of a round will often be aimed at 3, even if the probability of hitting target 2 using this action is actually higher than hitting the associated target 3. Until 8 or more points have been accumulated, action 3 is safe (i.e., cannot lead to busting), does not entrain a punishment if missing or hitting a lower target, and has a large learning potential. Figure 22 illustrates how the robot learns to throw more accurately within the physical limits of the system.

4 Conclusion & Future Work

In this paper, we have studied the problem of meta-parameter learning for motor primitives. It is an essential step towards applying motor primitives for learning complex motor skills in robotics more flexibly. We have discussed an appropriate reinforcement learning algorithm for mapping situations to meta-parameters.

We show that the necessary mapping from situation to meta-parameter can be learned using a Cost-regularized Kernel Regression (CrKR) while the parameters of the motor primitive can still be acquired through traditional approaches. The predictive variance of CrKR is used for ex-



(a) The initial position.

(b) The robot accelerates in the shoulder and elbow joints.

(c) End of acceleration. The breaking starts.

(d) The robot slows down while the ball flies towards the target.

(e) The robot has stopped and returns to the initial position.

(f) The robot is again in the initial position, ready for the next throw.

Fig. 23: These frames illustrate one throwing motion with the BioRob.

ploration in on-policy meta-parameter reinforcement learning. We compare the resulting algorithm in a toy scenario to a policy gradient algorithm with a well-tuned policy representation and the reward-weighted regression. We show that our CrKR algorithm can significantly outperform these preceding methods. We also illustrate the advantages of our reinforcement learning approach over supervised learning approaches in this setting. To demonstrate the system in a complex scenario, we have chosen the *Around the Clock* dart throwing game, table tennis, and ball throwing implemented both on simulated and real robots. In these scenarios we show that our approach performs well in a wide variety of settings, i.e. on four different real robots (namely a Barrett WAM, a BioRob, the JST-ICORP/SARCOS CBi and a Kuka KR 6), with different cost functions (both with and without secondary objectives), and with different policies in conjunction with their associated meta-parameters.

In the ball throwing task, we have discussed first steps towards a supervisory layer that deals with sequencing different motor primitives. This supervisory layer is learned by an hierarchical reinforcement learning approach (Huber and Grupen, 1998; Barto and Mahadevan, 2003). In this framework, the motor primitives with meta-parameter functions could also be seen as robotics counterpart of options (McGovern and Barto, 2001) or macro-actions (McGovern et al, 1997). The presented approach needs to be extended to deal with different actions that do not share the same underlying parametrization. For example in a table tennis task the supervisory layer would decide between a forehand motor primitive and a backhand motor primitive, the spatial meta-parameter and the timing of the motor primitive would be adapted according to the incoming ball, and the motor primitive would generate the trajectory. Future work will require to automatically detect which parameters can serve as meta-parameters as well as to discovering new motor primitives.

Acknowledgements The project receives funding from the European Community's Seventh Framework Programme under grant agreement no. ICT-248273 GeRT. The project receives funding from the European Community's Seventh Framework Programme under grant agreement no. ICT-270327 CompLACS. The authors thank Prof. K. Wöllhaf from the University of Applied Sciences Ravensburg-Weingarten for supporting the Kuka KR 6 experiment.

A Motor Primitive Meta-Parameters

The motor primitives based on dynamical systems (Ijspeert et al, 2002; Schaal et al, 2007; Kober et al, 2010a) have six natural meta-parameters: the initial position x_1^0, the initial velocity x_2^0, the goal g, the goal velocities $\dot{g}$, the amplitude A and the duration T. The meta-parameters modify the global movement by rescaling it spatially or temporally, or by reshaping it with respect to the desired boundary conditions. In the table tennis task the initial position and velocity are determined by the phase preceding the hitting phase. In Figure 24 we illustrate influence of the goal, goal velocity and duration meta-parameters on the movement generation.

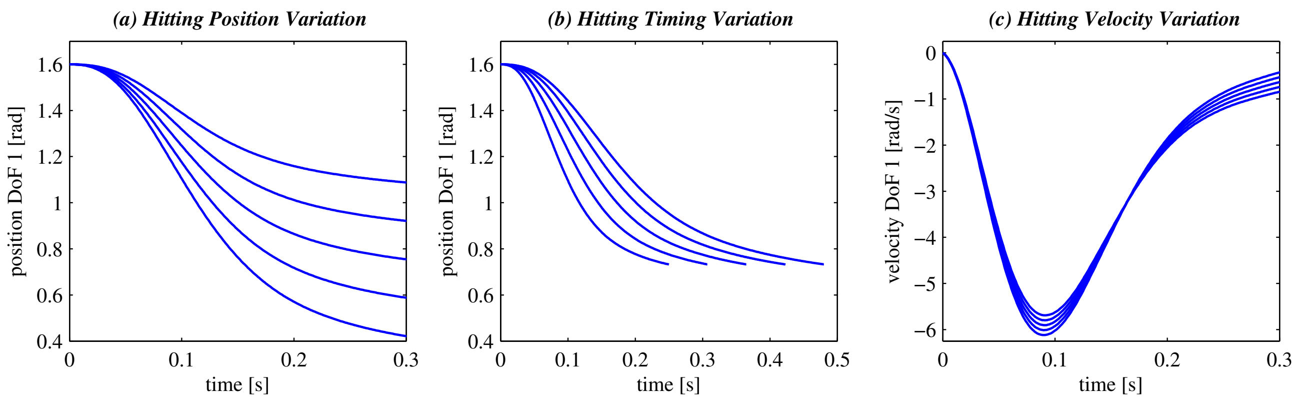


Fig. 24: In this figure, we demonstrate the influence of the goal, goal velocity and duration meta-parameters. The movement represents the hitting phase of the table tennis experiment (Section 3.3) and we demonstrate the variation of the meta-parameters employed in this task. The ball is hit at the end of the movement. In these plots we only vary a single meta-parameter at a time and keep the other ones fixed. In subfigure (a) the goal g is varied, which allows to hit the ball in different locations and with different orientations. In subfigure (b) duration T is varied, which allows to time the hit. In subfigure (c) the goal velocity $\dot{g}$ is varied, which allows to aim at different locations on the opponent's side of the table.

References

Barto A, Mahadevan S (2003) Recent advances in hierarchical reinforcement learning. Discrete Event Dynamic Systems 13(4):341 – 379

Bays P, Wolpert D (2007) Computational principles of sensorimotor control that minimise uncertainty and variability. Journal of Physiology 578:387–396

Bentivegna DC, Ude A, Atkeson CG, Cheng G (2004) Learning to act from observation and practice. Int Journal of Humanoid Robotics 1(4):585–611

Bishop CM (2006) Pattern Recognition and Machine Learning. Springer Verlag

Caruana R (1997) Multitask learning. Machine Learning 28:41–75

Cheng G, Hyon S, Morimoto J, Ude A, Hale JG, Colvin G, Scroggin W, Jacobsen SC (2007) CB: A humanoid research platform for exploring neuroscience. Journal of Advance Robotics 21(10):1097–1114

Dayan P, Hinton GE (1997) Using expectation-maximization for reinforcement learning. Neural Computation 9(2):271–278

Doya K (2002) Metalearning and neuromodulation. Neural Networks 15(4-6):495 – 506

Engel Y, Mannor S, Meir R (2005) Reinforcement learning with gaussian processes. In: Proc. Int. Conf. Machine Learning, pp 201–208

Grimes DB, Rao RPN (2008) Learning nonparametric policies by imitation. In: Proc. Int. Conf. Intelligent Robots and System, pp 2022–2028

Huber M, Grupen R (1998) Learning robot control – using control policies as abstract actions. In: NIPS'98 Workshop: Abstraction and Hierarchy in Reinforcement Learning

Ijspeert AJ, Nakanishi J, Schaal S (2002) Learning attractor landscapes for learning motor primitives. In: Advances in Neural Information Processing Systems 15, pp 1523–1530

Jaakkola T, Jordan MI, Singh SP (1993) Convergence of stochastic iterative dynamic programming algorithms. In: Advances in Neural Information Processing Systems 6, pp 703–710

Jetchev N, Toussaint M (2009) Trajectory prediction: learning to map situations to robot trajectories. In: Proc. Int. Conf. Machine Learning, p 57

Kober J, Peters J (2011a) Learning elementary movements jointly with a higher level task. In: Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems, pp 338–343

Kober J, Peters J (2011b) Policy search for motor primitives in robotics. Machine Learning 84(1-2):171–203

Kober J, Mülling K, Krömer O, Lampert CH, Schölkopf B, Peters J (2010a) Movement templates for learning of hitting and batting. In: Proc. IEEE Int. Conf. Robotics and Automation, pp 853–858

Kober J, Oztop E, Peters J (2010b) Reinforcement learning to adjust robot movements to new situations. In: Proc. Robotics: Science and Systems Conf., pp 33–40

Kronander K, Khansari-Zadeh MS, Billard A (2011) Learning to control planar hitting motions in a minigolf-like task. In: Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems, pp 710–717

Lampariello R, Nguyen-Tuong D, Castellini C, Hirzinger G, Peters J (2011) Trajectory planning for optimal robot catching in real-time. In: Proc. IEEE Int. Conf. Robotics and Automation, pp 3719–3726

Lawrence G, Cowan N, Russell S (2003) Efficient gradient estimation for motor control learning. In: Proc. Int. Conf. Uncertainty in Artificial Intelligence, pp 354–361

Lens T, Kunz J, Trommer C, Karguth A, von Stryk O (2010) Biorobarm: A quickly deployable and intrinsically safe, light-weight robot arm for service robotics applications. In: 41st International Symposium on Robotics / 6th German Conference on Robotics, pp 905–910

Masters Games Ltd (2010) The rules of darts. URL `http://www.mastersgames.com/rules/darts-rules.htm`

McGovern A, Barto AG (2001) Automatic discovery of subgoals in reinforcement learning using diverse density. In: Proc. Int. Conf. Machine Learning, pp 361–368

McGovern A, Sutton RS, Fagg AH (1997) Roles of macro-actions in accelerating reinforcement learning. In: Grace Hopper Celebration of Women in Computing

Mülling K, Kober J, Peters J (2010) Learning table tennis with a mixture of motor primitives. In: Proc. IEEE-RAS Int. Conf. Humanoid Robots, pp 411–416

Mülling K, Kober J, Peters J (2011) A biomimetic approach to robot table tennis. Adaptive Behavior 9(5):359 – 376

Nakanishi J, Morimoto J, Endo G, Cheng G, Schaal S, Kawato M (2004) Learning from demonstration and adaptation of biped locomotion. Robotics and Autonomous Systems 47(2-3):79–91

Park DH, Hoffmann H, Pastor P, Schaal S (2008) Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields. In: Proc. IEEE-RAS Int. Conf. Humanoid Robots, pp 91–98

Pastor P, Hoffmann H, Asfour T, Schaal S (2009) Learning and generalization of motor skills by learning from demonstration. In: Proc. IEEE Int. Conf. Robotics and Automation, pp 1293–1298

Peters J, Schaal S (2008a) Learning to control in operational space. Int Journal of Robotics Research 27(2):197–212

Peters J, Schaal S (2008b) Reinforcement learning of motor skills with policy gradients. Neural Networks 21(4):682–697

Pongas D, Billard A, Schaal S (2005) Rapid synchronization and accurate phase-locking of rhythmic motor primitives. In: Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems, pp 2911–2916

Rasmussen CE, Williams CK (2006) Gaussian Processes for Machine Learning. MIT Press

Russell S (1998) Learning agents for uncertain environments (extended abstract). In: Proc. Eleventh Annual Conference on Computational Learning Theory, ACM Press, pp 101–103

Schaal S, Mohajerian P, Ijspeert AJ (2007) Dynamics systems vs. optimal control – a unifying view. Progress in Brain Research 165(1):425–445

Schmidt R, Wrisberg C (2000) Motor Learning and Performance, 2nd edn. Human Kinetics

Sutton R, Barto A (1998) Reinforcement Learning. MIT PRESS

Sutton RS, McAllester D, Singh S, Mansour Y (1999) Policy gradient methods for reinforcement learning with function approximation. In: Advances in Neural Information Processing Systems 12, pp 1057–1063

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

Urbanek H, Albu-Schäffer A, van der Smagt P (2004) Learning from demonstration repetitive movements for autonomous service robotics. In: Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems, vol 4, pp 3495–3500

Welling M (2010) The Kalman filter. Lecture Notes

Williams RJ (1992) Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning 8:229–256

Wulf G (2007) Attention and motor skill learning. Human Kinetics, Champaign, IL

Jens Kober is a Ph.D. student at the Robot Learning Lab at the Max-Planck Institute for Intelligent Systems (MPI). He is also affiliated to the FG Intelligente Autonome Systeme at Technische Universität Darmstadt. He graduated from the Ecole Centrale Paris and holds a German M.Sc. degree in Engineering Cybernetics from Stuttgart University. His research interests include robotics, control, and machine learning.

Andreas Wilhelm is a computer science graduate student at the University of Applied Sciences Ravensburg-Weingarten.

Erhan Oztop earned his PhD from the University of Southern California in 2002. After holding several research positions, including vice department head, in Japan, he joined the Özyeğin University, Istanbul, Turkey as a faculty in 2011. Currently, he is also affiliated with Advanced Telecommunications Research Institute International (ATR) as cooperate researcher. His research interests include computational modeling of the brain mechanisms of action understanding and intelligent behavior, human-machine interface, robotics, machine learning and cognitive neuroscience.

Jan Peters is a full professor (W3) at Technische Universität Darmstadt heading the FG Intelligente Autonome Systeme while at the same time leading the Robot Learning Lab at the Max Planck Institute for Intelligent Systems. In 2007-2011, Jan Peters has been a senior research scientist and group leader at the Max Planck Institute for Biological Cybernetics. Jan Peters is a computer scientist (holding a German M.Sc. from FernUni Hagen, an M.Sc. and Ph.D. from University of Southern California), an electrical engineer (receiving a German M.Sc. in EE from TU München), and a mechanical engineer (with a M.Sc. in Mechanical Engineering from USC). Jan has held visiting research positions at ATR, Japan and at National University of Singapore during his graduate studies. Jan Peters' Ph.D. thesis received the 2007 Dick Volz Best US Robotics Ph.D. Runner-Up Award. Jan Peters' research interests span a large variety of topics in robotics, machine learning and biomimetic systems with a strong focus on learning of motor skills.