Reinforcement Learning to Adjust Robot Movements to New Situations

Jens Kober
MPI for Biol. Cybernetics, Germany
Email: jens.kober@tuebingen.mpg.de

Erhan Oztop
ATR Comput. Neuroscience Labs, Japan
Email: erhan@atr.jp

Jan Peters
MPI for Biol. Cybernetics, Germany
Email: jan.peters@tuebingen.mpg.de

***Abstract*—Many complex robot motor skills can be represented using elementary movements, and there exist efficient techniques for learning parametrized motor plans using demonstrations and self-improvement. However, in many cases, the robot currently needs to learn a new elementary movement even if a parametrized motor plan exists that covers a similar, related situation. Clearly, a method is needed that modulates the elementary movement through the meta-parameters of its representation. In this paper, we show how to learn such mappings from circumstances to meta-parameters using reinforcement learning. We introduce an appropriate reinforcement learning algorithm based on a kernelized version of the reward-weighted regression. We compare this algorithm to several previous methods on a toy example and show that it performs well in comparison to standard algorithms. Subsequently, we show two robot applications of the presented setup; i.e., the generalization of throwing movements in darts, and of hitting movements in table tennis. We show that both tasks can be learned successfully using simulated and real robots.**

I. Introduction

In robot learning, motor primitives based on dynamical systems [1], [2] 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 [1], T-ball batting [3], drumming [4], biped locomotion [5], ball-in-a-cup [6], and even in tasks with potential industrial applications [7]. The dynamical system motor primitives [1] can be adapted both spatially and temporally without changing the overall shape of the motion. While the examples are impressive, they do not address how a motor primitive can be generalized to a different behavior by trial and error without re-learning the task. For example, if the string length has been changed in a ball-in-a-cup [6] movement[1], the behavior has to be re-learned by modifying the movements parameters. Given that the behavior will not drastically change due to a string length variation of a few centimeters, it would be better to generalize that learned behavior to the modified task. Such generalization of behaviors can be achieved by adapting the meta-parameters of the movement representation[2].

In machine learning, there have been many attempts to use meta-parameters in order to generalize between tasks [8].

[1]In this movement, the system has to jerk a ball into a cup where the ball is connected to the bottom of the cup with a string.

[2]Note that the tennis-like swings [1] could only hit a static ball at the end of their trajectory, and T-ball batting [3] was accomplished by changing the policy's parameters.

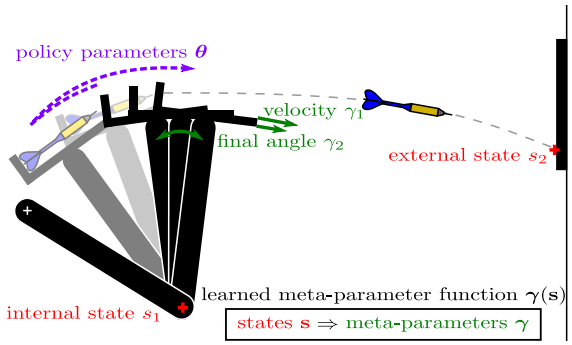


Figure 1: This figure illustrates a 2D dart throwing task. The situation, described by the state $\mathbf{s}$ corresponds to the relative height. The meta-parameters $\boldsymbol{\gamma}$ are the velocity and the angle at which the dart leaves the launcher. The policy parameters represent the backward motion and the movement on the arc. The meta-parameter function $\boldsymbol{\gamma}(\mathbf{s})$, which maps the state to the meta-parameters, is learned.

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) [9]. 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 robots 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. 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. Here, the human player has learned by trial and error how he has to adapt the global parameters of a generic strike to various situations [10]. 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.

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 [1] 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 [11], [6].

As it may be hard to realize a parametrized representation for meta-parameter determination, we reformulate the reward-weighted regression [11] in order to obtain a Cost-regularized Kernel Regression (CrKR) that is related to Gaussian process regression [12]. We compare the Cost-regularized Kernel Regression with a traditional policy gradient algorithm [3] and the reward-weighted regression [11] 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* [13] and for *table tennis*.

II. 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. In Section II-A, we first review how a single primitive movement can be represented and learned. We discuss how such 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 II-B. 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 II-C and employ for meta-parameter learning in Section II-D.

A. Motor Primitives with Meta-Parameters

In this section, we review how the dynamical systems motor primitives [1], [2] can be used for meta-parameter learning. The dynamical system motor primitives [1] are a powerful movement representation that allows ensuring the stability of the movement, 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 [14], [2]. Therefore, we will always focus on the discrete version of the dynamical system motor primitives in this paper (however, the results may generalize well to rhythmic motor primitives and hybrid settings). We use the most recent formulation of the discrete dynamical systems motor primitives [2] 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 $\mathbf{x}$ of a second system is chosen such that positions $\mathbf{q}$ of all degrees of freedom are given by $\mathbf{q} = \mathbf{x}_1$, the velocities by $\dot{\mathbf{q}} = \tau \mathbf{x}_2 = \dot{\mathbf{x}}_1$ and the accelerations by $\ddot{\mathbf{q}} = \tau \dot{\mathbf{x}}_2$. The learned dynamics of Ijspeert motor primitives can be expressed in the following form

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

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 $\mathbf{g}$, a transformation function $\mathbf{f}$ and an amplitude matrix $\mathbf{A} = \mathrm{diag}\left(a_1, a_2, \ldots, a_I\right)$, with the amplitude modifier $\mathbf{a} = [a_1, a_2, \ldots, a_I]$ allow representing complex movements. In [2], the authors use $\mathbf{a} = \mathbf{g} - \mathbf{x}_1^0$, with the initial position $\mathbf{x}_1^0$, which ensures linear scaling. Other choices are possibly better suited for specific tasks, see for example [15]. The transformation function $\mathbf{f}\left(z\right)$ 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

$$\mathbf{f}\left(z\right) = \textstyle\sum_{n=1}^{N} \psi_n\left(z\right) \boldsymbol{\theta}_n z. \tag{3}$$

Here, $\boldsymbol{\theta}_n$ 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 [2]. Normalized Gaussian kernels are used as weighting functions given by

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

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 $\mathbf{f}\left(z\right)$ in Equation (3) vanishes and the system stays at the goal position $\mathbf{g}$. 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 $\mathbf{f}\left(z\right)$ 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 [1], [2] 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 [1] and reinforcement learning [6]. The

resulting policy is invariant under transformations of the initial position $\mathbf{x}_1^0$, the goal $\mathbf{g}$, the amplitude $\mathbf{A}$ and the duration T [1]. These four modification parameters can be used as the meta-parameters $\boldsymbol{\gamma}$ of the movement. Obviously, we can make more use of the motor primitive framework by adjusting the meta-parameters $\boldsymbol{\gamma}$ depending on the current situation or state $\mathbf{s}$ according to a meta-parameter function $\boldsymbol{\gamma}(\mathbf{s})$. The state $\mathbf{s}$ 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 $\boldsymbol{\gamma}(\mathbf{s})$ by episodic reinforcement learning.

Illustration of the Learning Problem: As an illustration of the meta-parameter learning problem, we take a 2D dart throwing task with a dart on a launcher which is illustrated in Figure 1 (in Section III-B, we will expand this example to a robot application). Here, 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 [1], the combination of retracting and throwing 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 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 $\boldsymbol{\gamma}(\mathbf{s})$ maps the state (the relative height) to the meta-parameters $\boldsymbol{\gamma}$ (the final position $\mathbf{g}$ 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 [16], [17], [18] our approach does not require explicit (re)planning of the motion.

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

B. Problem Statement: Meta-Parameter Self-Improvement

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

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

where $R(\mathbf{s}, \boldsymbol{\gamma})$ denotes all the rewards following the selection of the meta-parameter $\boldsymbol{\gamma}$ according to a situation described by state $\mathbf{s}$. The return of an episode is $R(\mathbf{s}, \boldsymbol{\gamma}) = T^{-1} \sum_{t=0}^{T} r^t$ with number of steps T and rewards r^t. For a parametrized policy π with parameters $\mathbf{w}$ it is natural to first try a policy gradient approach such as finite-difference methods, vanilla policy gradient approaches and natural gradients[3]. Reinforcement learning of the meta-parameter function $\boldsymbol{\gamma}(\mathbf{s})$ 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.

Algorithm 1: Meta-Parameter Learning

Preparation steps:
- Learn one or more motor primitives by imitation and/or reinforcement learning (yields shape parameters $\boldsymbol{\theta}$).
- Determine initial state $\mathbf{s}^0$, meta-parameters $\boldsymbol{\gamma}^0$, and cost C^0 corresponding to the initial motor primitive.
- Initialize the corresponding matrices $\mathbf{S}, \boldsymbol{\Gamma}, \mathbf{C}$.
- Choose a kernel $\mathbf{k}, \mathbf{K}$.
- Set a scaling parameter λ.

For all iterations j:
- Determine the state $\mathbf{s}^j$ specifying the situation.
- Calculate the meta-parameters $\boldsymbol{\gamma}^j$ by:
  - Determine the mean of each meta-parameter i $\gamma_i(\mathbf{s}^j) = \mathbf{k}(\mathbf{s}^j)^{\mathrm{T}} (\mathbf{K} + \lambda \mathbf{C})^{-1} \boldsymbol{\Gamma}_i$,
  - Determine the variance $\sigma^2(\mathbf{s}^j) = k(\mathbf{s}^j, \mathbf{s}^j) - \mathbf{k}(\mathbf{s}^j)^{\mathrm{T}} (\mathbf{K} + \lambda \mathbf{C})^{-1} \mathbf{k}(\mathbf{s}^j)$,
  - Draw the meta-parameters from a Gaussian distribution $\boldsymbol{\gamma}^j \sim \mathcal{N}(\boldsymbol{\gamma}|\boldsymbol{\gamma}(\mathbf{s}^j), \sigma^2(\mathbf{s}^j)\mathbf{I})$.
- Execute the motor primitive using the new meta-parameters.
- Calculate the cost c^j at the end of the episode.
- Update $\mathbf{S}, \boldsymbol{\Gamma}, \mathbf{C}$ according to the achieved result.

Dayan & Hinton [19] showed that an immediate reward can be maximized by instead minimizing the Kullback-Leibler divergence $D(\pi(\boldsymbol{\gamma}|\mathbf{s})R(\mathbf{s}, \boldsymbol{\gamma})||\pi'(\boldsymbol{\gamma}|\mathbf{s}))$ between the reward-weighted policy $\pi(\boldsymbol{\gamma}|\mathbf{s})$ and the new policy $\pi'(\boldsymbol{\gamma}|\mathbf{s})$. Williams [20] suggested to use a particular policy in this context; i.e., the policy

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

where we have the deterministic mean policy $\boldsymbol{\gamma}(\mathbf{s}) = \boldsymbol{\phi}(\mathbf{s})^{\mathrm{T}}\mathbf{w}$ with basis functions $\boldsymbol{\phi}(\mathbf{s})$ and parameters $\mathbf{w}$ as well as the variance $\sigma^2(\mathbf{s})$ that determines the exploration $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2(\mathbf{s})\mathbf{I})$. The parameters $\mathbf{w}$ can then be adapted by reward-weighted regression in an immediate reward [11] or episodic reinforcement learning scenario [6]. 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.

C. 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 [21] for regular supervised learning). Inserting the reward-weighted regression solution $\mathbf{w} = (\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{R}\boldsymbol{\Phi} + \lambda\mathbf{I})^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{R}\boldsymbol{\Gamma}_i$, and using the Woodbury formula $(\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{R}\boldsymbol{\Phi} + \lambda\mathbf{I})\boldsymbol{\Phi}^{\mathrm{T}} = \boldsymbol{\Phi}^{\mathrm{T}}\mathbf{R}(\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} + \lambda\mathbf{R}^{-1})$, we

[3] While we will denote the shape parameters by $\boldsymbol{\theta}$, we denote the parameters of the meta-parameter function by $\mathbf{w}$.

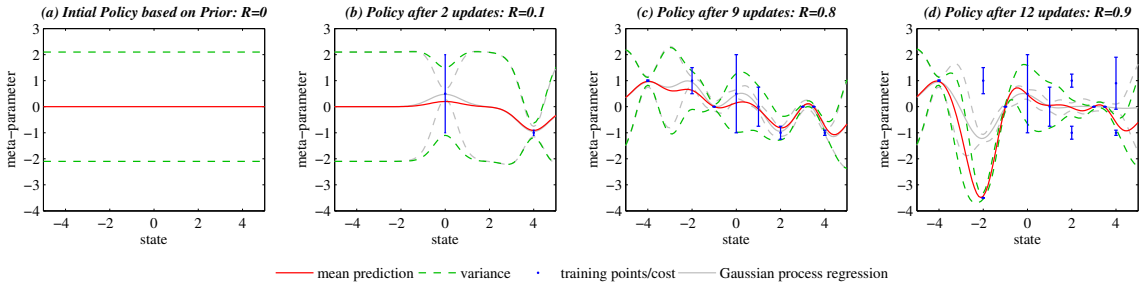


Figure 2: This figure illustrates the meaning of policy improvements with Cost-regularized Kernel Regression. Each sample consists of a state, a meta-parameter and a cost where the cost is indicated the blue error bars. The red line represents the improved mean policy, the dashed green lines indicate 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.

transform reward-weighted regression into a Cost-regularized Kernel Regression

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

where the rows of $\boldsymbol{\Phi}$ correspond to the basis functions $\boldsymbol{\phi}(\mathbf{s}_i) = \boldsymbol{\Phi}_i$ of the training examples, $\boldsymbol{\Gamma}_i$ is a vector containing the training examples for meta-parameter component $\boldsymbol{\gamma}_i$, 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 $\mathbf{C} = \mathbf{R}^{-1} = \mathrm{diag}(R_1^{-1}, \ldots, R_n^{-1})$ with the cost of all n data points. After replacing $\mathbf{k}(\mathbf{s}) = \boldsymbol{\phi}(\mathbf{s})^{\mathrm{T}}\boldsymbol{\Phi}^{\mathrm{T}}$ and $\mathbf{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}$, we obtain the Cost-regularized Kernel Regression

$$\bar{\boldsymbol{\gamma}}_i = \boldsymbol{\gamma}_i(\mathbf{s}) = \mathbf{k}(\mathbf{s})^{\mathrm{T}}\left(\mathbf{K} + \lambda\mathbf{C}\right)^{-1}\boldsymbol{\Gamma}_i,$$

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

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

where $k(\mathbf{s},\mathbf{s}) = \boldsymbol{\phi}(\mathbf{s})^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{s})$ is the distance of a point to itself. We call this algorithm Cost-regularized Kernel Regression.

The algorithm corresponds to a Gaussian process regression where the costs on the diagonal are input-dependent noise priors. Gaussian processes have been used previously for reinforcement learning [22] in value function based approaches while here we use them to learn the policy.

If several sets of meta-parameters have similarly low costs the algorithm's convergence 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-parameters.

D. Meta-Parameter Learning by Reinforcement Learning

As a result of Section II-C, we have a framework of motor primitives as introduced in Section II-A that we can use for reinforcement learning of meta-parameters as outlined in Section II-B. 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 $\boldsymbol{\gamma}$, (ii) an initial example containing state $\mathbf{s}^0$, meta-parameter $\boldsymbol{\gamma}^0$ and cost C^0, as well as (iii) a scaling parameter λ. The initial motor primitive can be obtained by imitation learning [1] and, subsequently, improved by parametrized reinforcement learning algorithms such as policy gradients [3] or Policy learning by Weighting Exploration with the Returns (PoWER) [6]. 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 2D dart throwing task introduced in Section II-A. 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. The initial policy is based on a prior, illustrated in Figure 2(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 throw a dart, we sample the meta-parameters from the policy based on the current state[4]. After the trial the cost is determined and, in conjunction with

[4]In the dart setting, we could choose the next target and thus employ CrKR as an active learning approach by picking states with large variances. However, often the state is determined by the environment, e.g., the ball trajectory in the table tennis experiment (Section III-C) depends on the opponent.

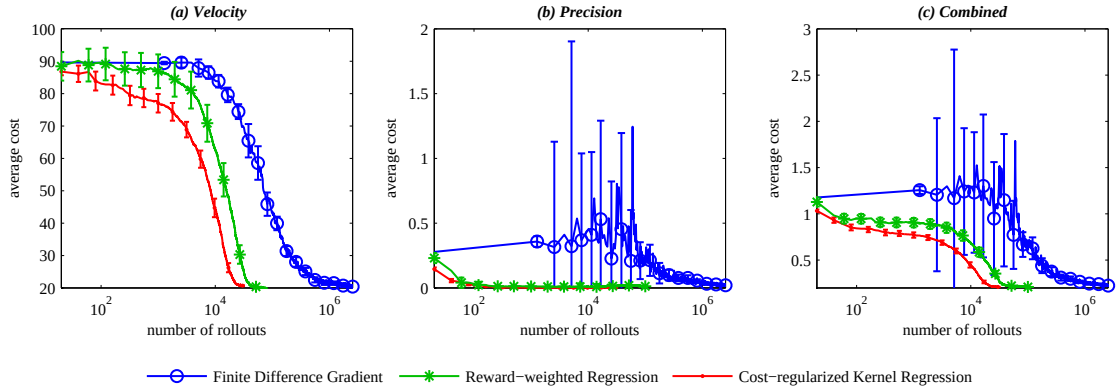


Figure 3: This figure shows the performance of the compared algorithms averaged over 10 complete learning runs. Cost-regularized Kernel Regression finds solutions with the same final performance two orders of magnitude faster than the finite difference gradient (FD) approach and twice as fast as the 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 employed meta-parameters, used to update the policy[5]. If the cost is large (for example the impact is far from the target), 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 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 2(b). Each additional sample refines the policy and the overall performance improves (see Figure 2(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. Algorithm 1 exhibits this behavior as illustrated in Figure 2(d).

III. Evaluation

In Section II, 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 [23] with the preceding reward-weighted regression and an off-the-shelf finite difference policy gradient approach.

[5]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.

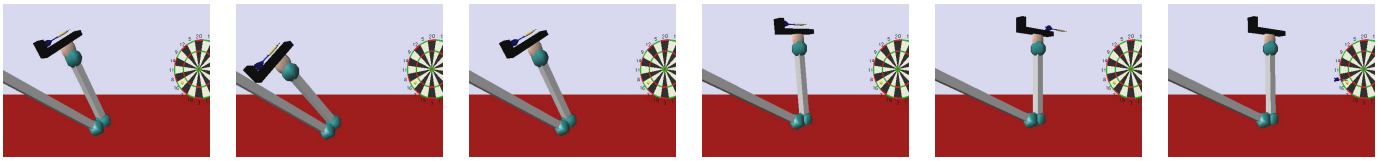
The proposed reinforcement learning method only requires a cost associated with the outcome of the trial. In the table tennis experiment (Section III-C), the state corresponds to the position and velocity of the ball over the net. We only observe the cost related to how well we hit the ball. After a table tennis trial, we do not know which state would have matched the employed meta-parameters, as would be required in a regression setting.

The resulting meta-parameter learning framework can be used in a variety of settings in robotics. We consider two scenarios here, i.e., (i) dart throwing with a simulated robot arm, a real Barrett WAM and the JST-ICORP/SARCOS humanoid robot CBi, and (ii) table tennis with a simulated robot arm and a real Barrett WAM. Some of the real-robot experiments are still partially work in progress.

A. Benchmark Comparison

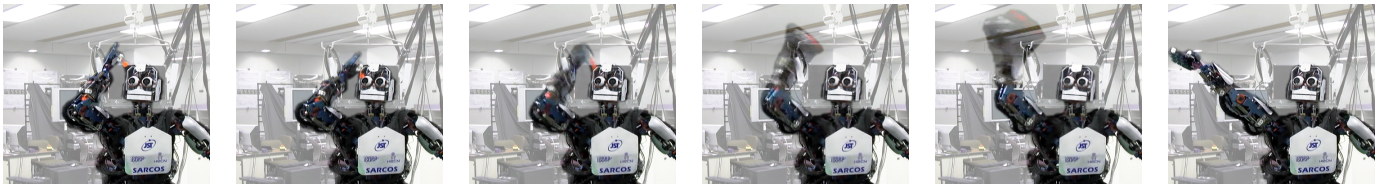
In the first task, we only consider a simple simulated planar cannon shooting where we benchmark our Reinforcement Learning by Cost-regularized Kernel Regression approach against a finite difference gradient estimator and the reward-weighted regression. Here, we want to learn an optimal policy for a 2D toy cannon environment similar to [23].

The setup is given as follows: A toy cannon is at a fixed location $[0.0, 0.1]\,m$. The meta-parameters are the angle with respect to the ground and the speed of the cannon ball. In this benchmark we do not employ the motor primitives but set the parameters directly. 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 parameters, the velocities in horizontal and vertical directions are predicted (which influences the angle and the speed of the ball leaving the cannon). 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 consists of the sum of the squared distance between the desired and the actual impact point and one hundredth of the squared norm of the velocity at impact of the cannon ball. It corresponds to maximizing the precision while minimizing the employed energy according to the chosen weighting. All approaches performed well in this setting, first driving the position error to zero and, subsequently, optimizing the impact



(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.

Figure 4: 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.

Figure 5: This figure shows a dart throw on the real JST-ICORP/SARCOS humanoid robot CBi.

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 3.

We compare our novel algorithm to a finite difference policy gradient (FD) method [3] and to the reward-weighted regression (RWR) [11]. The FD method uses a parametric policy that employs radial basis functions in order to represent the policy and adds Gaussian exploration. 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 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 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.

After convergence, the costs of CrKR are the same as for RWR and slightly lower than those of the FD method. The 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 3.

B. Dart-Throwing

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 [23]. However, we take a more complicated scenario and choose dart games such as *Around the Clock* [13] 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 [1]. Only the meta-parameter function is learned using CrKR or RWR.

The dart is placed on a launcher attached to the end-effector and held there by stiction. We use the Barrett WAM robot arm in order to achieve the high accelerations needed to overcome the stiction. See Figure 4, 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 inputs. The parameter for the final position, the duration of the motor primitive and the angle around the vertical axis are the meta-parameters. 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 the sum of ten times the squared error on impact and the velocity of the motion. After approximately 1000 throws the algorithms have converged but CrKR yields a high performance already much earlier (see Figure 6). We again used a parametric policy with radial basis functions for RWR. Designing a good parametric policy proved very difficult in this setting as is reflected by the poor performance of RWR.

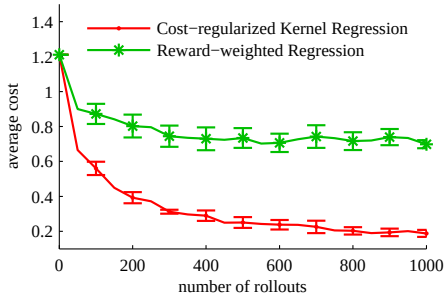


Figure 6: This figure shows the cost function of the dart-throwing task for a whole game *Around the Clock* in each rollout. The costs are averaged over 10 runs with the error-bars indicating standard deviation.

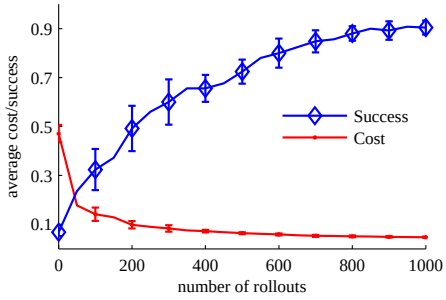


Figure 7: This figure shows the cost function of the table tennis task averaged over 10 runs with the error-bars indicating standard deviation. The red line represents the percentage of successful hits and the blue line the average cost. At the beginning the robot misses the ball 95% of the episodes and on average by 50 cm. At the end of the learning the robot hits almost all balls.

This experiment is also being carried out on two real, physical robots, i.e., a Barrett WAM and the humanoid robot CBi (JST-ICORP/SARCOS). 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 5 for a throwing movement. Parts of these experiments are still in-progress.

C. Table Tennis

In the second evaluation of the complete framework, we use it 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 predicted. The second phase is the hitting 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 8 for an illustration of a complete episode. The movements in the three phases are represented by motor primitives obtained by imitation learning.

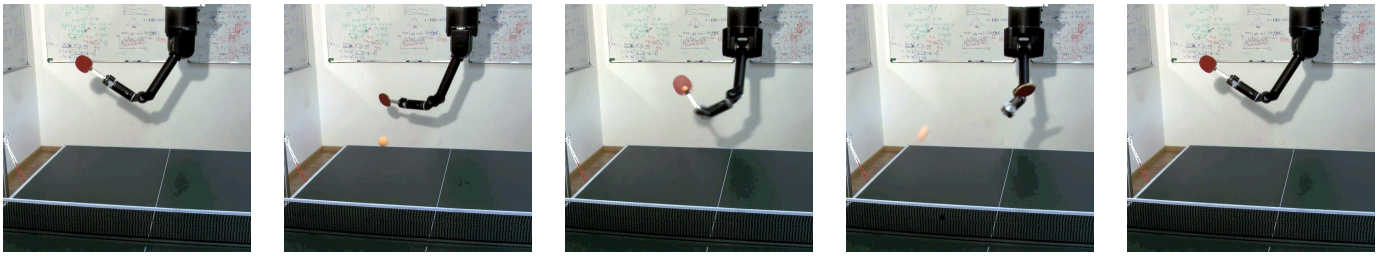
The meta-parameters are the joint positions and velocities for all seven degrees of freedom at the end of the second phase (the instant of hitting the ball) and a timing parameter that controls when the swing back phase is transitioning to the hitting phase. We learn these 15 meta-parameters as a function of the ball positions and velocities when it is over the net. We employed a Gaussian kernel and optimized the open parameters according to typical values for the input and output. As cost function we employ the metric distance between the center of the paddle and the center of the ball at the hitting time. 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 player. After initializing the meta-parameter function with only these five initial examples, the robot misses ca. 95% of the balls as shown in Figure 7. Trials are only used to update the policy if the robot has successfully hit the ball. Figure 9 illustrates different positions of the ball the policy is capable of dealing with after the learning. Figure 7 illustrates the costs over all episodes. Preliminary results suggest that the resulting policy performs well both in simulation and for the real system. We are currently in the process of executing this experiment also on the real Barrett WAM.

IV. 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.

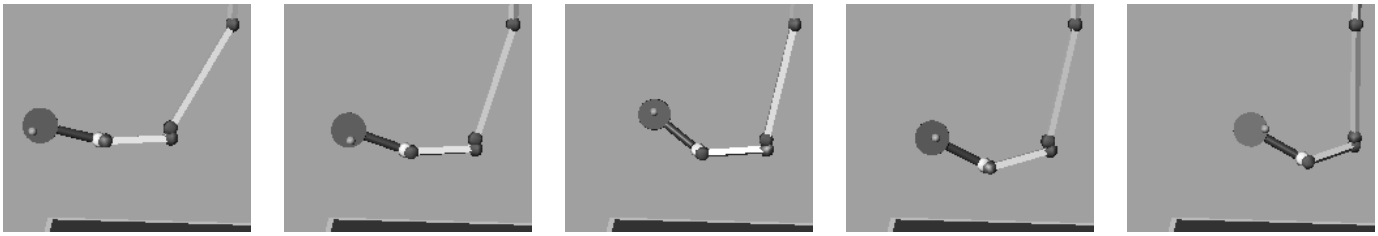
We show that the necessary mapping from situation to meta-parameter can be learned using a Cost-regularized Kernel Regression (CrKR) while the parameters of the motor primitive can still be acquired through traditional approaches. The predictive variance of CrKR is used for exploration in on-policy meta-parameter 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. To demonstrate the system in a complex scenario, we have chosen the *Around the Clock* dart throwing game and table tennis implemented both on simulated and real robots.

Adapting movements to situations is also discussed in [16] 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.



(a) The robot is 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.

Figure 8: This figure shows a table tennis stroke on the real Barrett WAM.



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

Figure 9: 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 and velocity and it requires a careful observer to note the important configuration differences resulting from the meta-parameters.

Future work will require to sequence different motor primitives by a supervisory layer. This supervisory layer would for example in a table tennis task 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. This supervisory layer could be learned by an hierarchical reinforcement learning approach [24] (as introduced in the early work by [25]). In this framework, the motor primitives with meta-parameter functions could be seen as robotics counterpart of options [9] or macro-actions [26].

REFERENCES

[1] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Learning attractor landscapes for learning motor primitives," in *Advances in Neural Information Processing Systems 16*, 2003.

[2] S. Schaal, P. Mohajerian, and A. J. Ijspeert, "Dynamics systems vs. optimal control — a unifying view," *Progress in Brain Research*, vol. 165, no. 1, pp. 425–445, 2007.

[3] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *Proc. Int. Conf. Intelligent Robots and Systems*, 2006.

[4] D. Pongas, A. Billard, and S. Schaal, "Rapid synchronization and accurate phase-locking of rhythmic motor primitives," in *Proc. Int. Conf. Intelligent Robots and Systems*, 2005.

[5] J. Nakanishi, J. Morimoto, G. Endo, G. Cheng, S. Schaal, and M. Kawato, "Learning from demonstration and adaptation of biped locomotion," *Robotics and Autonomous Systems*, vol. 47, no. 2-3, pp. 79–91, 2004.

[6] J. Kober and J. Peters, "Policy search for motor primitives in robotics," in *Advances in Neural Information Processing Systems 22*, 2009.

[7] H. Urbanek, A. Albu-Schäffer, and P. van der Smagt, "Learning from demonstration repetitive movements for autonomous service robotics," in *Proc. Int. Conf. Intelligent Robots and Systems*, 2004.

[8] R. Caruana, "Multitask learning," *Machine Learning*, vol. 28, pp. 41–75, 1997.

[9] A. McGovern and A. G. Barto, "Automatic discovery of subgoals in reinforcement learning using diverse density," in *Proc. Int. Conf. Machine Learning*, 2001.

[10] K. Mülling, "Motor control and learning in table tennis," Master's thesis, University of Tübingen, 2009.

[11] J. Peters and S. Schaal, "Reinforcement learning by reward-weighted regression for operational space control," in *Proc. Int. Conf. Machine Learning*, 2007.

[12] C. E. Rasmussen and C. K. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[13] Masters Games Ltd., "The rules of darts," online *http://www.mastersgames.com/rules/darts-rules.htm*, July 2010.

[14] G. Wulf, *Attention and motor skill learning*. Champaign, IL: Human Kinetics, 2007.

[15] D.-H. Park, H. Hoffmann, P. Pastor, and S. Schaal, "Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields," in *Proc. Int. Conf. Humanoid Robots*, 2008.

[16] N. Jetchev and M. Toussaint, "Trajectory prediction: learning to map situations to robot trajectories," in *Proc. Int. Conf. Machine Learning*, 2009.

[17] D. B. Grimes and R. P. N. Rao, "Learning nonparametric policies by imitation," in *Proc. Int. Conf. Intelligent Robots and System*, 2008.

[18] D. C. Bentivegna, A. Ude, C. G. Atkeson, and G. Cheng, "Learning to act from observation and practice," *Int. Journal of Humanoid Robotics*, vol. 1, no. 4, pp. 585–611, 2004.

[19] P. Dayan and G. E. Hinton, "Using expectation-maximization for reinforcement learning," *Neural Computation*, vol. 9, no. 2, pp. 271–278, 1997.

[20] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 1992.

[21] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer Verlag, 2006.

[22] Y. Engel, S. Mannor, and R. Meir, "Reinforcement learning with gaussian processes," in *Proc. Int. Conf. Machine Learning*, 2005.

[23] G. Lawrence, N. Cowan, and S. Russell, "Efficient gradient estimation for motor control learning," in *Proc. Int. Conf. Uncertainty in Artificial Intelligence*, 2003.

[24] A. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Systems*, vol. 13, no. 4, pp. 341 – 379, 2003.

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

[26] A. McGovern, R. S. Sutton, and A. H. Fagg, "Roles of macro-actions in accelerating reinforcement learning," in *Grace Hopper Celebration of Women in Computing*, 1997.