Reinforcement Learning to Adjust Robot Movements to New Situations

Jens Kober
MPI Tübingen, Germany
jens.kober@tuebingen.mpg.de

Erhan Oztop
ATR/CMC, Japan
erhan@atr.jp

Jan Peters
MPI Tübingen, Germany
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 with current techniques, in many cases, the robot currently needs to learn a new elementary movement even if a parametrized motor plan exists that covers a related situation. A method is needed that modulates the elementary movement through the meta-parameters of its representation. In this paper, we describe how to learn such mappings from circumstances to meta-parameters using reinforcement learning. In particular we use a kernelized version of the reward-weighted regression. We show two robot applications of the presented setup in robotic domains; the generalization of throwing movements in darts, and of hitting movements in table tennis. We demonstrate that both tasks can be learned successfully using simulated and real robots.

1 Introduction

In robot learning, motor primitives based on dynamical systems [Ijspeert *et al.*, 2003; Schaal *et al.*, 2007] 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.*, 2003], T-ball batting [Peters and Schaal, 2006], drumming [Pongas *et al.*, 2005], biped locomotion [Nakanishi *et al.*, 2004], ball-in-a-cup [Kober and Peters, 2010], and even in tasks with potential industrial applications [Urbanek *et al.*, 2004]. The dynamical system motor primitives [Ijspeert *et al.*, 2003] 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 [Kober and Peters, 2010] movement, 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[1].

In reinforcement 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]. 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 [Mülling *et al.*, 2010]. 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 (DMPs) of [Ijspeert *et al.*, 2003] 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, 2007; Kober and Peters, 2010].

[1] Note that the tennis-like swings [Ijspeert *et al.*, 2003] could only hit a static ball at the end of their trajectory, and T-ball batting [Peters and Schaal, 2006] was accomplished by changing the policy's parameters.

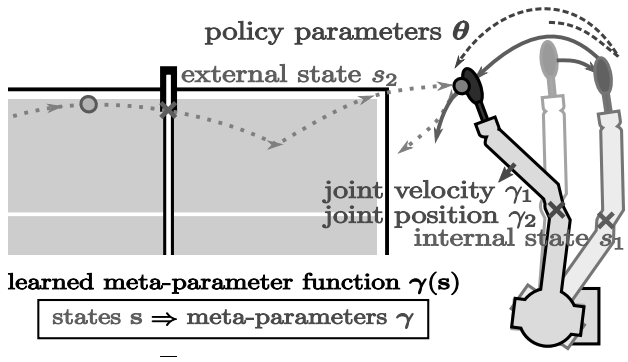

Figure 1: This figure illustrates a table tennis task. The situation, described by the state $\mathbf{s}$, corresponds to the positions and velocities of the ball and the robot at the time the ball is above the net. The meta-parameters $\boldsymbol{\gamma}$ are the joint positions and velocity at which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

As it may be hard to realize a parametrized representation for meta-parameter determination, we reformulate the reward-weighted regression [Peters and Schaal, 2007] in order to obtain a Cost-regularized Kernel Regression (CrKR) that is related to Gaussian process regression [Rasmussen and Williams, 2006]. We evaluate the algorithm in the acquisition of flexible motor primitives for dart games such as *Around the Clock* [Masters Games Ltd., 2011] and for *table tennis*.

2 Meta-Parameter Learning for DMPs

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 2.1, 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 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 employ for meta-parameter learning in Section 2.3.

2.1 DMPs with Meta-Parameters

In this section, we review how the dynamical systems motor primitives [Ijspeert *et al.*, 2003; Schaal *et al.*, 2007] can be used for meta-parameter learning. The dynamical system motor primitives [Ijspeert *et al.*, 2003] are a powerful movement representation that allows ensuring the stability of the movement, choosing between a rhythmic and a discrete movement. One of the biggest advantages of this motor primitive framework is that it 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.*, 2003] and reinforcement learning [Kober and Peters, 2010]. The resulting policy is invariant under transformations of the initial position, the goal, the amplitude and the duration [Ijspeert *et al.*, 2003]. 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 table tennis task which is illustrated in Figure 1 (in Section 3.2, 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.*, 2003], 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 $\boldsymbol{\gamma}(\mathbf{s})$ maps the state (the state of the ball and the robot before the return) to the meta-parameters $\boldsymbol{\gamma}$ (the final positions and velocities of the motor primitive). Its variance corresponds to the uncertainty of the mapping.

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

2.2 Kernalized 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

$$J(\pi) = \int_{\mathbb{S}} p(\mathbf{s}) \int_{\mathbb{G}} \pi(\boldsymbol{\gamma}|\mathbf{s}) R(\mathbf{s}, \boldsymbol{\gamma}) d\boldsymbol{\gamma}\, d\mathbf{s}, \qquad (1)$$

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[2]. Reinforce-

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

Algorithm 1: Meta-Parameter Learning

Preparation steps:

Learn one or more DMPs 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 DMP.

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

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

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 DMP 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.

ment 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.

Dayan and Hinton [1997] 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 [Williams, 1992] 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 [Peters and Schaal, 2007] or episodic reinforcement learning scenario [Kober and Peters, 2010]. 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.

Designing good basis functions is challenging. A non-parametric representation is better suited in this context. We can transform the reward-weighted regression into a Cost-regularized Kernel Regression

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

where $\boldsymbol{\Gamma}_i$ is a vector containing the training examples $\boldsymbol{\gamma}_i$ of the meta-parameter component, $\mathbf{C} = \mathbf{R}^{-1} = \mathrm{diag}(R_1^{-1}, \ldots, R_n^{-1})$ is a cost matrix, λ is a ridge factor, and $\mathbf{k}(\mathbf{s}) = \boldsymbol{\phi}(\mathbf{s})^{\mathrm{T}}\boldsymbol{\Phi}^{\mathrm{T}}$ as well as $\mathbf{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}$ correspond to a kernel where the rows of $\boldsymbol{\Phi}$ are the basis functions $\boldsymbol{\phi}(\mathbf{s}_i) = \boldsymbol{\Phi}_i$ of the training examples. Please refer to [Kober *et al.*, 2010] for a full derivation. 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 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. 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.

2.3 Reinforcement Learning of Meta-Parameters

As a result of Section 2.2, 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 $\boldsymbol{\gamma}$, (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 [Ijspeert *et al.*, 2003] and, subsequently, improved by parametrized reinforcement learning algorithms such as policy gradients [Peters and Schaal, 2006] or Policy learning by Weighting Exploration with the Returns (PoWER) [Kober and Peters, 2010]. 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 corresponds to the distance between the ball and the paddle,

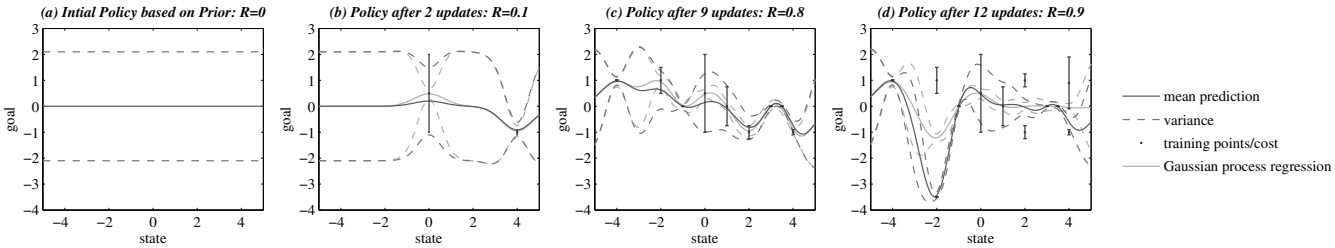

Figure 2: This figure illustrates the meaning of policy improvements with Cost-regularized Kernel Regression. Each sample consists of a state, a meta-parameter and a cost where the cost is indicated the blue error bars. The red line represents the improved mean policy, the dashed green lines indicate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

as well as the squared torques. 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 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 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).

In the dart throwing example (Section 3.1) 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 could be employed as a regularized regression method in this case. 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.

3 Evaluation

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 [Kober *et al.*, 2010] we have shown that the presented reinforcement learning algorithm yields higher performance than the preceding reward-weighted regression and an off-the-shelf finite difference policy gradient approach on a benchmark example. The 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 Barrett WAM and the real JST-ICORP/SARCOS humanoid robot CBi, and (ii) table tennis with a simulated robot arm and a real Barrett WAM.

3.1 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 [Lawrence *et al.*, 2003]. However, we take a more complicated scenario and choose dart games such as *Around the Clock* [Masters Games Ltd., 2011] 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.*, 2003]. Only the meta-parameter function is learned using CrKR or RWR.

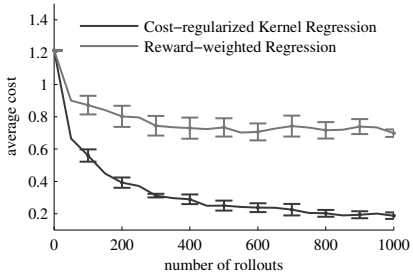

Figure 3: This figure shows the cost function of the dart-throwing task in simulation for a whole game *Around the Clock* in each rollout. The costs are averaged over 10 runs with the error-bars indicating standard deviation.

The 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. The motor primitive is trained by imitation learning with kinesthetic teach-in. We use the Cartesian coor-

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

dinates 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 3). 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.

This experiment is carried out in simulation and on a real, physical robot, i.e., 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. See Figure 4 for a throwing movement. The results on the real robot are significantly more noisy but qualitatively comparable to the simulation.

3.2 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 6 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 5. Trials are only used to update the policy if the robot has successfully hit the ball. Figure 5 illustrates the costs over all episodes. Current results suggest that the resulting policy performs well both in simulation and for the real 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 exploration in on-policy meta-parameter reinforcement learning. To demonstrate the system, we have chosen the *Around the Clock* dart throwing

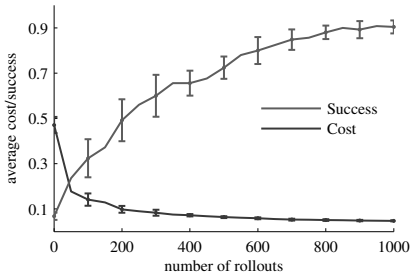

Figure 5: This figure shows the cost function of the table tennis task in simulation averaged over 10 runs with the error-bars indicating standard deviation. The red line represents the percentage of successful hits and the blue line the average cost.

(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.

Figure 6: This figure shows a table tennis stroke on the real Barrett WAM.

game and table tennis implemented both on simulated and real robots.

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 [Barto and Mahadevan, 2003] (as introduced in the early work by [Huber and Grupen, 1998]). In this framework, the motor primitives with meta-parameter functions could be seen as robotics counterpart of options [McGovern and Barto, 2001].

References

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

[Caruana, 1997] R. Caruana. Multitask learning. *Machine Learning*, 28:41–75, 1997.

[Dayan and Hinton, 1997] P. Dayan and G. E. Hinton. Using expectation-maximization for reinforcement learning. *Neural Computation*, 9(2):271–278, 1997.

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

[Ijspeert *et al.*, 2003] A. J. Ijspeert, J. Nakanishi, and S. Schaal. Learning attractor landscapes for learning motor primitives. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2003.

[Kober and Peters, 2010] J. Kober and J. Peters. Policy search for motor primitives in robotics. *Machine Learning*, 2010.

[Kober *et al.*, 2010] J. Kober, E. Oztop, and J. Peters. Reinforcement learning to adjust robot movements to new situations. In *Proceedings of the Robotics: Science and Systems Conference (R:SS)*, 2010.

[Lawrence *et al.*, 2003] G. Lawrence, N. Cowan, and S. Russell. Efficient gradient estimation for motor control learning. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2003.

[Masters Games Ltd., 2011] Masters Games Ltd. The rules of darts, *http://www.mastersgames.com/rules/darts-rules.htm*, 2011.

[McGovern and Barto, 2001] A. McGovern and A. G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2001.

[Mülling *et al.*, 2010] K. Mülling, J. Kober, and J. Peters. Learning table tennis with a mixture of motor primitives. In *Proceedings of the International Conference on Humanoid Robots (HUMANOIDS)*, 2010.

[Nakanishi *et al.*, 2004] 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*, 47(2-3):79–91, 2004.

[Peters and Schaal, 2006] J. Peters and S. Schaal. Policy gradient methods for robotics. In *Proceedings of the International Conference on Intelligent RObots and Systems (IROS)*, 2006.

[Peters and Schaal, 2007] J. Peters and S. Schaal. Reinforcement learning by reward-weighted regression for operational space control. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2007.

[Pongas *et al.*, 2005] D. Pongas, A. Billard, and S. Schaal. Rapid synchronization and accurate phase-locking of rhythmic motor primitives. In *Proceedings of the International Conference on Intelligent RObots and Systems (IROS)*, 2005.

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

[Schaal *et al.*, 2007] S. Schaal, P. Mohajerian, and A. J. Ijspeert. Dynamics systems vs. optimal control – a unifying view. *Progress in Brain Research*, 165(1):425–445, 2007.

[Urbanek *et al.*, 2004] H. Urbanek, A. Albu-Schäffer, and P. v.d. Smagt. Learning from demonstration repetitive movements for autonomous service robotics. In *Proceedings of the International Conference on Intelligent RObots and Systems (IROS)*, 2004.

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