Learning Complex Motions by Sequencing Simpler Motion Templates

Gerhard Neumann GERHARD@IGI.TUGRAZ.AT
Wolfgang Maass MAASS@IGI.TUGRAZ.AT
Institute for Theoretical Computer Science, Graz University of Technology, A-8010 Graz, Austria

Jan Peters MAIL@JAN-PETERS.NET
Max Planck Institute for Biological Cybernetics, D-72076 Tübingen, Germany

Abstract

Abstraction of complex, longer motor tasks into simpler elemental movements enables humans and animals to exhibit motor skills which have not yet been matched by robots. Humans intuitively decompose complex motions into smaller, simpler segments. For example when describing simple movements like drawing a triangle with a pen, we can easily name the basic steps of this movement.

Surprisingly, such abstractions have rarely been used in artificial motor skill learning algorithms. These algorithms typically choose a new action (such as a torque or a force) at a very fast time-scale. As a result, both policy and temporal credit assignment problem become unnecessarily complex - often beyond the reach of current machine learning methods.

We introduce a new framework for temporal abstractions in reinforcement learning (RL), i.e. RL with motion templates. We present a new algorithm for this framework which can learn high-quality policies by making only few abstract decisions.

1. Introduction

Humans use abstractions to simplify the motor tasks occurring during their daily life. For example when describing simple movements like drawing a triangle with a pen, we can easily name the basic steps of this movement. In a similar manner, many complex movements can be decomposed into smaller, simpler segments. This sort of abstraction is for example often used by engineers for designing hybrid control solutions (Xu & Antsaklis, 2002) where the single segments are implemented as local, linear continuous controllers. We will call these building blocks *motion templates*. Other names that can be found in the literature are "motion primitives", "movement schemas", "basis behaviors" or "options" (Ijspeert et al., 2002; Arbib, 1981; Dautenhahn & Nehaniv, 2002; Sutton et al., 1999).

Motor skill learning is a challenging problem for machine learning and, in particular, for the subfield of reinforcement learning (RL). Primarily used in motor skill learning is the flat RL setting without the use of abstractions. In this setting the agent has to choose a new action (typically a motor force or torque) at a very small sampling frequency. While this allows the representation of arbitrary policies, this flexibility makes the learning problem so complex that it is often beyond the reach of current methods. A common approach for limiting the potential complexity of the policy in the flat RL setting is to use a parametrized policy. Ijspeert et al. (2002) introduced a special kind of parametrized policies called *motion primitives*, which are based on dynamical systems. In most applications to date, only a single *motion primitive* is used for the whole movement. Parametrized policy search methods such as policy gradient descent and EM-like policy updates (Kober & Peters, 2009) have been used in order to improve single-stroke motor primitives.

Currently, only few abstractions are used in RL algorithms for continuous environments, with few exceptions such as (Huber & Grupen, 1998; Ghavamzadeh & Mahadevan, 2003). In (Huber & Grupen, 1998) the policy acquisition problem is reduced to learning to coordinate a set of closed loop control strategies. In (Ghavamzadeh & Mahadevan, 2003) the given task is

Appearing in *Proceedings of the 26th International Conference on Machine Learning*, Montreal, Canada, 2009.

manually decomposed into a set of subtasks. Both, the lower-level subtasks and the higher-level subtask-selection policies are learned. In all these approaches the structure for the hierarchy of abstraction is manually designed and fixed during learning which limits the generality of these approaches. In our approach, an arbitrary parametrization of the abstracted level can be learned.

In this paper, we introduce a new framework for abstraction in RL, i.e. RL with motion templates. Motion templates are our building blocks of motion. A template m_p is represented as parametrized policy and executed until its termination condition is fulfilled. We assume that the functional forms of the motion templates remain fixed, and thus, our task is to learn the correct order and parameters of the motion templates by reinforcement learning. As motion templates are temporally extended actions, they can be seen as parametrized *options* in continuous time. There are a few well-established learning algorithms for the options framework (Sutton et al., 1999). However, these algorithms are designed for discrete environments.

Choosing the parameters of a motion template is a continuous-valued decision. However, a single decision has now much more influence on the outcome of the whole motion than in flat RL. Thus, the decisions have to be made more precisely, though, the overall learning problem is simplified because much fewer decisions are needed to fulfill a task. As RL in continuous action spaces is already challenging in the flat RL setting, the requirement of learning highly-precise policies has limited the use of this sort of abstraction for motor control learning.

This paper introduces a new algorithm which satisfies this requirement and therefore permits learning at an abstract level. The algorithm is based on the Locally-Advantage WEighted Regression (LAWER) algorithm. LAWER is a fitted Q-Iteration (Ernst et al., 2005) based algorithm which has been shown to learn high-quality continuous valued policies for many flat RL settings (Neumann & Peters, 2009). However, two substantial extensions are needed to render motion template learning possible. Firstly, we propose an improved estimation of the goodness of an state action pair. Secondly, we introduce an adaptive kernel, which is based on randomized regression trees (Ernst et al., 2005).

We conduct experiments on 3 different tasks, a 1-link and a 2-link pendulum swing-up task and also a 2-link balancing task.

2. Motion Templates

A motion template m_p is defined by its k_p dimensional parameter space $\Theta_p \subseteq \mathcal{R}^{k_p}$, its parametrized policy $u_p(\mathbf{s}, t; \theta_p)$ ($\mathbf{s}$ is the current state, t represents the time spent executing the template and $\theta_p \in \Theta_p$ is the parameter vector) and its termination condition $c_p(\mathbf{s}, t; \theta_p)$.

At each decision-time point σ_k, the agent has to choose a motion template m_p from the set $\mathcal{A}(\sigma_k)$ and also the parametrization θ_p of m_p. Subsequently the agent follows the policy $p_p(\mathbf{s}, t; \theta_p)$ until the termination condition $c_p(\mathbf{s}, t; \theta_p)$ is fulfilled. Afterwards, we obtain a new decision-time point σ_{k+1}.

The functional forms of the policy $u_p(\mathbf{s}, t; \theta_p)$ and the termination condition $c_p(\mathbf{s}, t; \theta_p)$ are defined beforehand and can be arbitrary functions. For example, consider again the task of drawing a triangle. We can define a motion template m_{line} for drawing a line with the endpoint of the line and the velocity of moving the pen as parameters. The policy u_{line} moves the pen from the current position with the specified velocity in the direction of the endpoint of the line. The template is terminated when the pen has reached a certain neighborhood of the endpoint.

In our experiments, sigmoidal functions and linear controllers are used to model the motion templates.

2.1. Reinforcement Learning with Motion Templates

Each motion template is a temporally extended, continuous valued action. Thus, we deal with a continuous-time Semi-Markov Decision Process (SMDP). We will review only the relevant concepts from the continuous-time SMDP framework. For a detailed definition, please refer to (Bradtke & Duff, 1995).

Unlike in standard Markov Decision Processes (MDPs), the transition probability function $P(s', d|s, a)$ is extended by the duration d of an action. The Bellman equation for the value function $V^\pi(s)$ of policy π is given by

$$V^\pi(s) = \int_a \pi(a|s) \left(r(s,a) + \int_{s'} \int_{t=0}^{\infty} \exp(-\beta t) P(s', t|s, a) V^\pi(s') dt ds' \right) da, \quad (1)$$

where β is the discount factor[1]. The action value func-

[1] In order to achieve the same discounting rate as in a flat MDP, β can be calculated from the relation $\gamma =$

tion $Q^\pi(s,a)$ is given by

$$Q^\pi(s,a) = r(s,a) + \int_{s'}\int_{t=0}^{\infty} \exp(-\beta t) P(s',t|s,a) V^\pi(s') dt ds'. \quad (2)$$

A policy is now defined as $\pi(m_p, \theta_p|s_k)$. It can be decomposed into $\pi(m_p|s_k)\pi_p(\theta_p|s_k)$, where $\pi(m_p|s_k)$ is the template selection policy and $\pi_p(\theta_p|s_k)$ is the policy for selecting the parameters of template m_p.

3. Fitted Q-Iteration

As LAWER is a Fitted Q-iteration (FQI) (Ernst et al., 2005; Riedmiller, 2005) based algorithm we quickly review the relevant concepts. FQI is a batch mode reinforcement learning (BMRL) algorithm. In BMRL algorithms we assume that all the experience of the agent up to the current time is given in the form $H = \{< \mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i >\}_{1\le i\le N}$. FQI estimates an optimal control policy from this historical data. Therefore it approximates the state-action value function $Q(\mathbf{s},\mathbf{a})$ by iteratively using supervised regression techniques. New target values for the regression are generated by

$$\begin{aligned}\tilde{Q}_{k+1}(i) &= r_i + \gamma V_k(\mathbf{s}'_i), \\ &= r_i + \gamma \max_{\mathbf{a}'} Q_k(\mathbf{s}'_i, \mathbf{a}'),\end{aligned} \quad (3)$$

which are subsequently used to learn the Q-function $Q_{k+1}(\mathbf{s},\mathbf{a})$. For more details please refer to (Neumann & Peters, 2009).

3.1. Fitted Q-Iteration for SMDPs

For SMDPs we have to include the duration d_i of each action to our historical data $H = \{< \mathbf{s}_i, \mathbf{a}_i, r_i, d_i, \mathbf{s}'_i > \}_{1\le i\le N}$. Instead of using Equation 3, new Q-values can now be calculated by

$$\tilde{Q}_{k+1}(i) = r_i + \exp(-\beta d_i) \max_{\mathbf{a}'} Q_k(\mathbf{s}'_i, \mathbf{a}'). \quad (4)$$

3.2. Locally-Advantage-WEighted Regression (LAWER)

A severe problem when using fitted Q-iteration for continuous action spaces is the use of the greedy operation $V_k(\mathbf{s}) = \max_{a'} Q_k(\mathbf{s},\mathbf{a}')$ which is hard to perform. LAWER (Neumann & Peters, 2009) is a variant of FQI which avoids this max operator and is therefore well suited for continuous action spaces. The algorithm has been shown to learn high quality policies for many flat RL settings.

$\exp(-\beta\Delta t)$, where γ is the discount factor and Δt is the time step of the flat MDP.

Instead of using the max operator, a soft-max operator is used which can be efficiently approximated by an advantage-weighted regression. The advantage-weighted regression solely uses the given state action pairs $(\mathbf{s}_i, \mathbf{a}_i)$ to estimate the V-function and therefore avoids an exhaustive search in the action space. State-action pairs with an higher expected advantage[2] have a higher influence on the regression.

The regression uses the state vectors $\mathbf{s_i}$ as input dataset, the Q-values $\tilde{Q}_{k+1}(i)$ as target values and an additional weighting u_i for each data point. The authors proved that the result of the advantage-weighted regression is an approximation of the V-function $V(\mathbf{s}) = \max_{a'} Q_k(\mathbf{s},\mathbf{a}')$. The weighting u_i can be seen as goodness of using action $\mathbf{a}_i$ in state $\mathbf{s}_i$. It is estimated by $u_i = \exp(\tau \bar{A}(\mathbf{s}_i,\mathbf{a}_i))$, where $\bar{A}(\mathbf{s}_i,\mathbf{a}_i)$ denotes the normalized advantage function and the parameter τ sets the greediness of the soft-max operator. We skip the description of the normalization of the advantage function, because, for this paper, it is enough to know that the normalization, and also the proof of the algorithm, assume normally distributed advantage values. For a more detailed description of $\bar{A}(\mathbf{s}_i,\mathbf{a}_i)$ please refer to (Neumann & Peters, 2009).

LAWER uses Locally Weighted Regression (LWR, by Atkeson et al., 1997) for approximating the Q and the V-function. It therefore needs to be able to calculate the similarity $w_i(\mathbf{s})$ between a state $\mathbf{s}_i$ in the dataset H and state $\mathbf{s}$. The state similarities $w_i(\mathbf{s})$ can be calculated by a Gaussian kernel $w_i(\mathbf{s}) = \exp(-(\mathbf{s}_i - \mathbf{s})^T \mathbf{D} (\mathbf{s}_i - \mathbf{s}))$. In this paper we also introduce an adaptive kernel in Section 4.1. For simplicity, we will denote $w_i(\mathbf{s}_j)$ as w_{ij} for all $\mathbf{s}_j \in H$.

Standard LWR is used to estimate the Q-function. The V-function is approximated by a combination of LWR and advantage-weighted regression. In order to do so, the advantage weighting u_i is multiplicatively combined with the state similarity weighting, resulting again in a standard weighted linear regression. For the exact equations, please refer to (Neumann & Peters, 2009).

The optimal policy $\pi(\mathbf{a}|\mathbf{s}) = \mathcal{N}(\mathbf{a}|\mu(\mathbf{s}), \Sigma(\mathbf{s}))$ is modelled as stochastic policy with Gaussian exploration. The mean $\mu(\mathbf{s})$ can be determined by a similar locally and advantage-weighted regression, just the actions $\mathbf{a}_i$ are used as targets instead of the Q-values. The covariance matrix $\Sigma(\mathbf{s})$ is given by calculating the advantage-weighted covariance of locally neighbored actions.

Intuitively speaking, the V-function is calculated by

[2] The advantage function is given by $A(\mathbf{s}_i,\mathbf{a}_i) = Q(\mathbf{s}_i,\mathbf{a}_i) - V(\mathbf{s}_i)$

interpolating between the Q-values of locally neighbored state action pairs, but only examples with a high goodness u_i (i.e. high normalized advantage value) are used. The same is true for the policy, we just interpolate between the action vectors.

4. Fitted Q-iteration for Motion Templates

In order to apply the LAWER algorithm to the motion template framework we use a separate dataset H^p and individual estimations Q^p and V^p of the Q and V-function for each motion template m_p. The functions V^p and Q^p represent the state and state-action value function when choosing motion template m_p in the first decision and subsequently following the optimal policy. We implement the template selection policy $\pi(m_p|\mathbf{s}_k)$ by a soft-max policy. The overall value function is determined by $V(\sigma_k) = \max_{m_p \in \mathcal{A}(\sigma_k)} V^p(\sigma_k)$. LAWER is used to learn the single Q and V-function estimates Q^p and V^p.

In this section we present two extensions which improve the accuracy of LAWER and render learning with motion templates possible. Firstly, adaptive tree-based kernels are used to improve the estimation of the state similarities w_{ij}. This kernel also adapts to spatially varying curvatures of the regression surface and therefore needs an estimate of the V-function. Secondly, we show how to improve the estimate of the goodness u_i by the use of an additional optimization. Based on the current estimate of the state similarities w_{ij}, new u_i values, and subsequently also new estimates of the V-function are calculated. Both algorithms are applied intertwined to get improved estimates of w_{ij} and u_i.

4.1. Adaptive Tree-based Kernels

The use of an uniform weighting kernel is often problematic in the case of high dimensional input spaces ('curse of dimensionality'), spatially varying data densities or spatially varying curvatures of the regression surface. This problem can be alleviated by varying the 'shape' of the weighting kernel.

We use the Extremely Randomized Tree (Extra-Tree) algorithm (Ernst et al., 2005) to obtain a varying kernel function. This algorithm has been particularly successful for approximating the Q-function in FQI. We modify this approach to calculate the weighting kernel. The resulting kernel has the same properties as the Extra-Trees, and therefore adapts to the local state density as well as to the local curvature of the V-function.

The standard Extra-Tree algorithm builds an ensemble of regression trees. It has 3 parameters, the number M of regression trees, the number K of randomized splits to evaluate per node and the maximum number of samples per leaf $n_{\min}$. For more details about the algorithm please refer to (Ernst et al., 2005).

We use the trees for calculating the state similarities w_{ij} instead of approximating the Q-function. In order to do so, we learn the mapping from the states $\mathbf{s}_i$ to the V-values $V(\mathbf{s}_i)$ with the Extra-Tree algorithm. The kernel is then given by the fraction of trees in which two states $\mathbf{s}_i$ and $\mathbf{s}_j$ are located in the same leaf

$$w_{ij} = \frac{1}{M} \sum_{k=1}^{M} \text{isSameLeaf}(T_k, \mathbf{s}_i, \mathbf{s}_j), \tag{5}$$

where T_k is the kth tree in the ensemble and isSameLeaf is a function returning 1 if both examples are located in the same leaf and 0 otherwise. In our experiments we will show the superiority of the tree-based kernels to the Gaussian kernels.

4.2. Optimized LAWER

As already pointed out in Section 3.2, LAWER assumes normally distributed advantage values. Often this assumption does not hold or the normalization of the advantages is imprecise due to too few data points in the neighborhood. This effect is even more drastic if high τ values are used because the inaccuracies may result in low activations in areas with a low sample density and therefore also in inaccurate regressions. This restriction on the τ parameter also limits the quality of the estimated policy.

But how can we improve the estimation of the weightings u_i? Let us first consider a greedy policy π_D in a discrete environment. We formulate π_D as stochastic policy $u_{ij} = \pi_D(\mathbf{a}_j|\mathbf{s}_i)$. The u_{ij} can be found by solving the following constraint optimization problem

$$\begin{aligned} \mathbf{u} = \text{argmax}_{\mathbf{u}} & \textstyle\sum_{i,j} u_{ij} A(\mathbf{s}_i, \mathbf{a}_j) \\ \text{subject to:} \quad & \textstyle\sum_j u_{ij} = 1 \text{ for all states } \mathbf{s}_i \\ & 0 \le u_{ij} \le 1 \text{ for all } i, j, \end{aligned} \tag{6}$$

where $\mathbf{u}$ is the vector of all u_{ij} and A is again the advantage function. In our setting, we also have a finite number of state-action pairs $(\mathbf{s}_i, \mathbf{a}_i)$, but typically all the states are different. However, the states are linked by the state similarities w_{ij}. The first constraint of the optimization problem can therefore be reformulated as

$$\sum_j w_{ij} u_j = 1 \text{ for all states } s_i, \tag{7}$$

while the remaining formulation of the optimization is unchanged. We also skipped the second index of u_{ij} because there is only one action for each state $\mathbf{s}_i$. Due to this optimization we only use the u_i with the highest advantage values while ensuring that the summed activation $\sum_j w_{ij}u_j$ is high enough at each state $\mathbf{s}_i$ for applying an accurate weighted linear regression.

We solve the constraint optimization problem by maximizing the performance function C

$$C = \frac{1}{Z}\sum_j u_j(Q(\mathbf{s}_j,\mathbf{a}_j) - V(\mathbf{s}_j)) - \lambda \sum_i \frac{(\sum_j w_{ij}u_j - \eta)^2}{\sum_j w_{ij}}, \tag{8}$$

with $\eta = 1$, where Z is a normalization constant for the advantage values given by $Z = \sum_i |Q(\mathbf{s}_i,\mathbf{a}_i)|/N$. The second term of Equation 8 specifies the squared summed activation error for each state $\mathbf{s}_i$. It is normalized by the summed state-similarity of this state (i.e. $\sum_j w_{ij}$). This ensures that the activation error is equally weighted throughout the state space, independent of the local state density. We also introduced a new parameter λ which sets the tradeoff between maximizing the greediness of u_i or minimizing the summed activation error. It replaces the greediness parameter τ of the LAWER algorithm.

The function C can be maximized with respect to u_i using gradient ascent, the derivation of C is given by

$$\frac{dC}{du_k} = \frac{1}{Z}(Q(\mathbf{s}_k,\mathbf{a}_k) - V(\mathbf{s}_k)) - 2\lambda \sum_i \frac{(\sum_j w_{ij}u_j - \eta)}{\sum_j w_{ij}} w_{ik}. \tag{9}$$

The learning rate for the gradient ascent algorithm is always chosen such that the maximum change of an activation u_i is fixed to 0.01. After each gradient update the weights u_i are restricted to the interval $[0;1]$. The gradient ascent update is repeated for N_{opt} iterations, every $M_{opt} << N_{opt}$ iterations the value estimates $V(\mathbf{s}_i)$ are recalculated using the current weights u_i. When using the tree-based kernels, we also recalculate the state similarities w_{ij} with the new estimate of $V(\mathbf{s}_i)$. Typical values for N_{opt} and M_{opt} are 1000 and 100.

The covariance matrix of the exploration policy is also calculated slightly differently to the original LAWER algorithm. We require that always the best η_{exp} locally neighbored actions are used. We therefore use a separate set of advantage weightings u_{exp} for the covariance calculation which can be obtained by the same optimization defined in Equation 8, we just have to set η to η_{exp}. With η_{exp} we can scale the exploration rate of the algorithm.

5. Results

We evaluated the motion template approach on a 1-link and a 2-link pendulum swing-up task and a 2-link balancing task. For each task the immediate reward function was quadratic in the distance to the goal position $\mathbf{x}_G$ and in the applied torque/force, i.e., $r = -c_1|\mathbf{x} - \mathbf{x_G}|^2 - c_2|\mathbf{a}|^2$. For all our experiments we assume that the goal position $\mathbf{x_G}$ is known.

We collect L new episodes with the currently estimated exploration policy and one episode with the greedy policy (without exploration). After estimating the optimal policy, its performance is evaluated (without exploration) and the data collection is repeated. The initial distributions of the motion template parameters were set intuitively and were by no means optimal. We compared the motion template approach to flat RL with the standard LAWER algorithm.

5.1. Swing-Up Tasks

In this task a pendulum needs to be swung up from the position at the bottom to the top position.

5.1.1. 1-link Pendulum

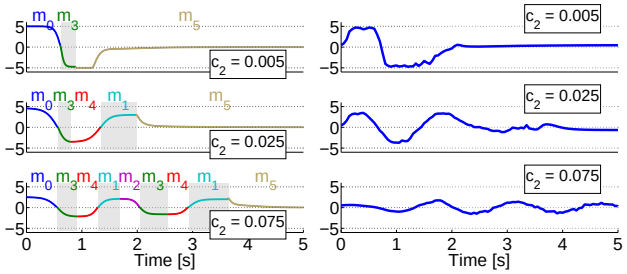
The link of the pendulum had a length of 1m and a mass of 1kg, no friction was used. The used motion templates represent positive (m_1 and m_2) and negative peaks (m_3 and m_4) in the torque trajectory. There is also an individual template m_5 for balancing the robot at the top position. One peak consists of 2 successive motion templates, one for the ascending and one for the descending part of the peak.

The parametrization of the motion templates can be seen in Table 1. In order to form a proper peak, template m_2 and m_4 always start with the last torque u_t taken in the end of the previous template. Therefore parameter a_2 of these templates is already determined by u_t and consequently the outcome of template m_2 and m_4 depend on u_t. For this reason, the state space of template m_2 and m_4 was extended by u_t. The balancing template m_5 is implemented as linear PD-controller (see Table 1). The duration of the peak templates is an individual parameter of the templates (d_i), m_5 is always the final template and runs for $20s$. Subsequently the episode is ended.

The agent always started from the bottom position with motion template m_0. Afterwards it could either

Table 1. MTs for the swing up motion. The functional forms resemble sigmoid functions. Parameter a_i coresponds to the height of the peak, o_i to the initial time offset and d_i to the duration of the motion template. k_1 and k_2 are the PD-controller constants of the balancer template. m_3 and m_4 resemble m_1 and m_2 except for a negative sign. The sketches illustrate the torque trajectories of these templates (x-axis: time, y-axis: acceleration).

MT	Functional Form	Parameters	Sketch
m_0	$a_0(1 - \frac{2}{1+\exp(o_0 - \frac{o_0}{d_0}t)})$	a_0, o_0, d_0	
$m_{1,3}$	$a_1(\frac{2}{1+\exp(-\frac{6o_1}{d_1}t)} - 1)$	a_1, o_1, d_1	
$m_{2,4}$	$a_2(1 - \frac{2}{1+\exp(o_2 - \frac{o_2}{d_2}t)})$	o_2, d_2	
m_5	$-k_1\theta - k_2\theta'$	k_1, k_2	



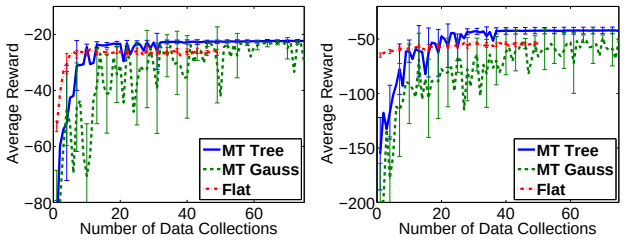
(a) (b)

Figure 1. (a) Torque trajectories and motion templates learned for different action punishment factors c_2. (b) Torque trajectories learned with flat RL

choose to use the peak templates in the predefined order (m_3, m_4, m_1, m_2, m_3...) or use the balancing template m_5. Thus, the agent had to learn the correct parametrization of the motion templates and the number of swing-up motions.

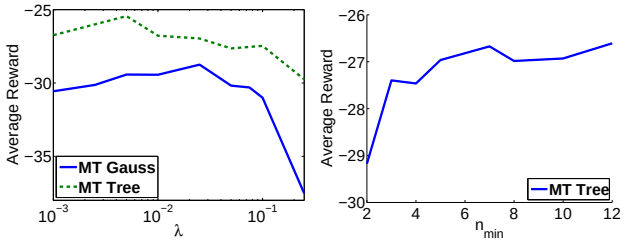
For all experiments a discount factor of $\beta = 0.2$ was used, λ was set to 0.025 and η_{exp} to 20. For the Gaussian kernel we used a bandwidth matrix of $\mathbf{D} = \text{diag}(30, 3)$ for m_1, m_3 and m_5 and $\mathbf{D} = \text{diag}(30, 3, 1)$ for the extended state space of templates m_2 and m_4. For the tree-based kernels we used the parameters $n_{min} = 7$, $M = 80$ and $K = 20$. For the comparison with the flat LAWER algorithm τ was set to 4 and a time step of 50ms was used. We used $L = 50$ episodes per data collection.

We carried out 3 experiments with different torque punishment factors ($c_2 = 0.005$, $c_2 = 0.025$ and $c_2 = 0.075$). We compared the learning process of flat RL, motion template learning with Gaussian state



(a) (b)

Figure 2. Learning curves for the Gaussian kernel (MT Gauss) and the tree-based kernel (MT Tree) for (a) $c_2 = 0.025$ and (b) $c_2 = 0.075$



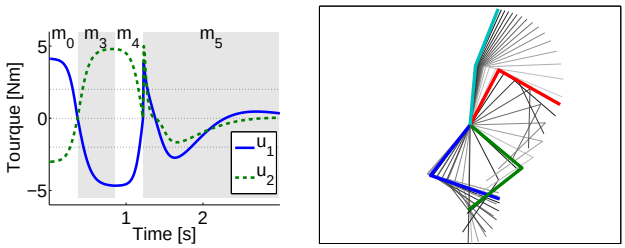
(a) (b)

Figure 3. (a) Evaluation of the influence of λ for the Gaussian (MT Gauss) and the tree-based kernel (MT Tree, $n_{\min} = 5$) (b) Evaluation of the $n_{\min}$ parameter for $\lambda = 0.025$. c_2 was set to 0.025 for both evaluations.

similarities (MT Gauss) and with adaptive tree-based state similarities (MT Tree) (see Figure 2). In the initial learning phase, the flat RL approach is superior to motion template learning, probably due to the larger number of produced training examples. However, RL with motion templates is able to produce policies of significantly higher quality and quickly outperforms the flat RL approach. This can also be seen in Figure 1(a) and (b), where the resulting torque trajectories are compared. Flat RL has difficulties particularly with the hardest setting ($c_2 = 0.075$) where we received a maximum average reward of -48.6 for flat RL and -38.5 for the motion template approach. From Figure 2 we can also see that the tree-based kernel is much more sample efficient than the Gaussian kernel. An evaluation of the influence of the λ parameter can be seen in Figure 3(a) and of the parameter n_{min} of the tree-based kernel in Figure 3(b). The approach works robustly for a wide range of parameters.

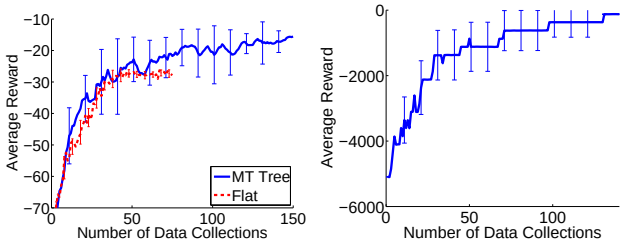
5.1.2. 2-link Pendulum

We also conducted experiments with a 2-link pendulum. The lenghts of the links were set to 1m, each link had a mass of 1kg (located at the center of the link). We use the same templates as for the 1-dimensional



(a) (b)

Figure 4. (a) Torque trajectories and decomposition in the motion templates for the 2-link pendulum swing-up task. (b) Illustration of the motion. The bold postures represent the switching time points of the motion templates.



(a) (b)

Figure 5. Learning curves for motion template learning with tree-based kernels for the (a) 2-link swing-up task and the (b) 2-link balancing task.

task, the peak templates have now 2 additional parameters, the height of the peak a_i and the time offset o_i for the second control dimension u_2. Including the duration parameter, this results in 5 parameters for m_0, m_1 and m_3 and 3 parameters for m_2 and m_4. The parameters of the balancer template m_5 consists of two 2×2 matrices for the controller gains.

Experiments were done for the tree-based kernels with $n_{min} = 8$, $\lambda = 0.025$ and $\eta_{exp} = 25$. At each data collection, 50 new episodes were collected. For comparison to the flat RL approach we used a bandwidth matrix of $D = \text{diag}(6.36, 2.38, 3.18, 1.06)$ and $\tau = 4$. The evaluation of the learning process can be seen in Figure 5(a) and the learned motion and torque trajectories are shown in Figure 4. Also for this challenging task, the motion template approach was able to learn high-quality policies. While the flat RL approach stagnates at an average reward of -28.7, the motion template approach reaches an average reward of -15.6.

5.2. 2-link Balancing

In this task a 2-link pendulum needs to be balanced at the top position after being pushed. The model parameters were chosen to loosely match the characteristics of a human, i.e. $l_i = 1$m and $m_i = 35$kg. The hip-joint was limited to $[-0.1; 1.5]$rad and the ankle-joint to $[-0.8; 0.4]$rad. Whenever the robot left this area of the state space, we assumed that the robot had fallen, i.e. a negative reward of -10000 was given. The hip-torque was limited to ± 500Nm and the ankle torque to ± 70Nm.

In the beginning of an episode, the robot stands upright and gets pushed with a certain force F. This results in an immediate jump of the joint velocities. The agent has to learn to keep balance for different perturbations. In (Atkeson & Stephens, 2007) this problem was solved exactly using Dynamic Programming techniques. The authors found out that two different balancing strategies emerge. For small perturbations, the ankle strategy, which uses almost only the ankle joint, is optimal. For larger perturbations ($F > 17.5$Ns), the ankle-hip strategy, which results in a fast bending movement, is optimal. In this experiment we want to reproduce both strategies by motion template learning.

We use two motion templates to model the balancing behavior, both resemble linear controllers. The first motion template (m_0) keeps the robot at the upright position and is similar to m_5 from the previous experiment. The second template m_1 additionally defines a set-point of the linear controller for each joint and a duration parameter d_1. In addition to the 8 controller gains, this results in 11 parameters. The agent can now choose to use m_0 directly in the beginning or to use m_1 and subsequently m_0. We used 4 different perturbations, i.e., $F = 10, 15, 20$ and 25Ns. For each perturbation, we collected $L = 20$ episodes.

We again used the tree-based approach with the same parameter setting as in the previous experiment. The learning curve can be seen in Figure 5(b). The resulting torque trajectories are shown in Figure 6(a) and (b). We can clearly identify the ankle strategy for the two smaller perturbations and the ankle-hip strategy for larger perturbations using both motion templates.

6. Conclusion and Future Work

In this paper we proposed a new framework for temporal abstraction for RL in continuous environments, i.e. RL with motion templates. Learning the overall control task is decomposed into learning a sequence of simpler controllers. Because of the used abstractions the agent has to make fewer decisions, which simplifies the learning task. We strongly belief that this kind of abstractions may help scaling RL algorithms to more

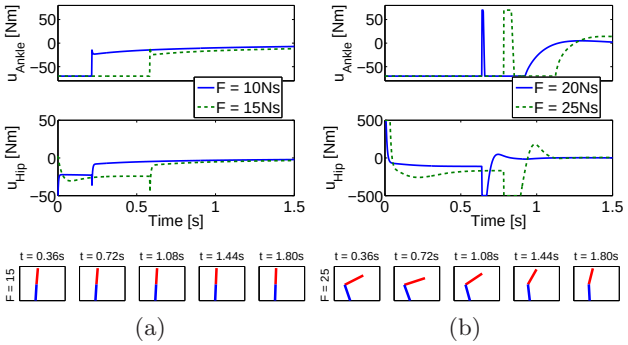


Figure 6. Learned solutions for the 2-link balancing problem for (a) $F = 10$Ns and $F = 15$Ns (ankle strategy) (b) $F = 20$Ns and $F = 25$Ns (ankle-hip strategy). The sketches bellow illustrate the temporal course of the balancing movement for the ankle strategy (a) and the ankle-hip strategy (b)

complex domains.

The motion templates approach also raises several interesting research questions to which we will dedicate our future work. For example, how can we efficiently add feedback to the motion templates? Which functional forms of the templates can facilitate learning? When do we terminate a motion template, in particular in the case of unforeseen events? Future work will also concentrate on applying the approach to more complex environments such as planar walking robots.

7. Acknowledgment

Written under partial support by the Austrian Science Fund FWF project # P17229-N04 and the projects # FP7-216593 (SECO) and # FP7-506778 (PASCAL2) of the European Union.

References

Arbib, M. A. (1981). Perceptual structures and distributed motor control. *Handbook of physiology, section 2: The nervous system vol. ii, motor control, part 1*, 1449–1480.

Atkeson, C., & Stephens, B. (2007). Multiple balance strategies from one optimization criterion. *7th IEEE-RAS International Conference on Humanoid Robots*.

Atkeson, C. G., Moore, A. W., & Schaal, S. (1997). Locally weighted learning. *Artificial Intelligence Review, 11*, 11–73.

Bradtke, S. J., & Duff, M. O. (1995). Reinforcement learning methods for continuous-time markov decision problems. *Advances in Neural Information Processing Systems 7, 7*, 393–400.

Dautenhahn, K., & Nehaniv, C. L. (2002). *Imitation in animals and artifacts*. Campridge: MIT Press.

Ernst, D., Geurts, P., & Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. *J. Mach. Learn. Res., 6*, 503–556.

Ghavamzadeh, M., & Mahadevan, S. (2003). Hierarchical policy gradient algorithms. *Twentieth International Conference on Machine Learning (ICML-2003)* (pp. 226–233).

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

Ijspeert, A., Nakanishi, J., & Schaal, S. (2002). Learning attractor landscapes for learning motor primitives. *Advances in Neural Information Processing Systems 15 (NIPS2002)* (pp. 1523–1530).

Kober, J., & Peters, J. (2009). Policy search for motor primitives in robotics. *Advances in Neural Information Processing Systems 22 (NIPS 2008)* (pp. 849–856). MA: MIT Press.

Neumann, G., & Peters, J. (2009). Fitted Q-iteration by Advantage Weighted Regression. *Advances in Neural Information Processing Systems 22 (NIPS 2008)* (pp. 1177–1184). MA: MIT Press.

Riedmiller, M. (2005). Neural fitted Q-iteration - first experiences with a data efficient neural reinforcement learning method. *Proceedings of the European Conference on Machine Learning (ECML)* (pp. 317–328).

Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and Semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence, 112*, 181–211.

Xu, X., & Antsaklis, P. (2002). An approach to optimal control of switched systems with internally forced switchings. *Proceedings of the American Control Conference* (pp. 148–153). Anchorage, USA.