Project no.: 027657
Project full title: Perception, Action & Cognition through Learning of Object-Action Complexes
Project Acronym: PACO-PLUS
Deliverable no.: D6.9
Title of the deliverable: Publication of RL on ARMAR: Report supplemented by one or two publications on RL for glass-filling and on the results of applying RL to high-dimensional action spaces.

Contractual Date of Delivery to the CEC:	31. Jan 2010
Actual Date of Delivery to the CEC	15. June 2010
Organisation name of lead contractor for this deliverable:	BCCN
Authors:	Wörgötter, F., Ales Ude
Participants:	BCCN, JSI
Work package contributing to the deliverable	WP6
Nature:	R/D
Version	1.0
Total number of pages:	43
Start date of project:	1st Feb. 2006
Duration:	52 months

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)

Dissemination Level		
PU	Public	**X**
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Abstract:

This deliverable contains two publications (”Humanoids 2009” as well as ”submitted to RAS” on the combination of goal and shape reinforcement learning for the task of learning to pour.

Keyword list: Reinforcement Learning, PI^2-method, Natural Actor Critic, Value Function Approximation, Dynamic Movement Primitives

Publication of RL on ARMAR: Report supplemented by one or two publications on RL for glass-filling and on the results of applying RL to high-dimensional action spaces.

Wörgötter F. and Ales Ude.

Executive Summary

This deliverable consists of two papers (Nemec et al., 2009 and Tamosiunaite et al, 2011), which address the problem of learning to pour liquid from one container into another by a robot with many DOF.

Background: When describing robot motion with dynamic motion primitives (DMPs) one employs two types of parameters: "goal" position, towards which the movement is attracted; and parameters for the non-linear part of the DMP-equations (called "weights"), which determine the shape of the generated trajectory. When considering reinforcement learning (RL) with DMPs, usually goals are pre-defined and only the weights for shaping a DMP are subject to learning. Many tasks, however, exist where the best goal position is not a priori known, requiring to learn it. Here we chose the task of learning to pour liquid from one container into another. Due to the turbulent motion of the liquid, it is very difficult to "guess", which goal position should be used allowing for appropriate trajectory shape learning. Thus, using this task as an example we specifically address the question of how to simultaneously combine of goal and shape parameter learning. As such this is a difficult problem because goal and shape parameters could easily interfere in a destructive way jeopardizing convergence of learning. The pouring task was chosen as it represents an example of a generic set of learning tasks which often occur especially in service robotics like those addressed by PACO-PLUS. There one often finds "fuzzy" tasks of a kind where many successful solutions exist and where highest accuracy (such as that needed for industrial robots) is not required.

To shape robot movements, efficient implementations of the policy gradient approach to reinforcement learning have been proposed recently and drew much attention in the robotics community (NAC, Peters and Schaal, 2008; PI^2, Theodorou et al., 2010, PoWER, Kober and Peters, 2009, Policy search via signed derivative, Kolter and Ng, 2009). The NAC, PI^2 and PoWER methods were used for learning of DMP shape parameters. We use PI^2 and NAC methods for DMP shaping and augment the shape learning procedures by in parallel learning goal parameters of the DMP which has not been done before.

Results: The investigation had started a year earlier with the Natural Actor Critic (NAC) Method (Peters and Schaal 2008), employing it on the tasks of “learning to pour”. This, however, had led to substantial frustration as this method is brittle and contains too many mutually interfering parameters. A paper was published about this in a cooperation of BCCN with JSI on the Humanoids conference in Paris (Nemec et al., 2009, see Appendix to this deliverable). As a consequence of the existing problems we had switched to a very novel method called “Policy Improvement with Path Integrals” (PI^2, Theodorou et al., 2010). We found that this method works *very well* also for high dimensional problems and, thus, holds high promise in robot learning. A paper (by BCCN and JSI) is submitted (Tamosiunaite et al., 2011, see Appendix to this deliverable), where we compare NAC with PI^2 in simulation and also on a real robot. This is the only paper so far written by an “independent group” (hence, not by the inventors themselves) explaining and comparing these very complex methods. Integration with PACO-

PLUS has been performed by using these methods on the PA10 robot of JSI but – due to having been forced to switch learning methods from NAC to PI^2 – we have refrained from attempting integration on ARMAR III.

Novel Contribution: The scientific novelty of this contribution lies in the fact that we are here combining value function approximation RL (our older method, Tamosiunaite et al., 2009) with policy gradient RL methods (PI^2) performing simultaneous update. This is entirely novel and is described in Tamosiunaite et al., (2011).

Deviation from the Planned Work: This deliverable is delayed by about 4 months. This is due to the fact that the results from the NAC learning were negative. First we thought this is our fault and spent quite some time trying to optimize the approach, but without success. As a consequence we had to switch to a different method (PI^2), which works far better. All these methods are mathematically very demanding, hence analysis of the problems with the NAC as well as understanding and implementing PI^2 took quite some time. As a consequence we encountered a delay, additionally, we did not anymore attempt to implement PI^2 on ARMAR. The now submitted paper on the comparison of NAC and PI^2 (Tamosiunaite et al. 2011) should, however, be very helpful for the community to see where the problems lie and to avoid similar delays. The use of NAC for high dimensional trajectory learning is discouraged as compared to PI^2!

References

Nemec, B., Tamosiunaite, M., Wörgötter, F., Ude, A. (2009) Task adaptation through exploration and action sequencing. Proceedings of 9th IEEE-RAS International Conference on Humanoid Robots Humanoids2009, pp. 610-616.

Peters, J. and Schaal, S. (2008). Natural actor critic, Neurocomputing, 71, 7-9, pp.1180-1190.

Tamosiunaite, M., Asfour T., Wörgötter, F. (2009) Learning to reach by reinforcement learning using a receptive field based function approximation approach with continuous actions. Biological Cybernetics 100(3), pp. 249-260.

Tamosiunaite, M., Nemec, B., Ude, A. and Wörgötter, F. (2011) Combining value function approximation and policy gradient reinforcement methods for goal and shape learning on robots, RAS, submitted

Theodorou, E.A., Jonas Buchli, Stefan Schaal (2010). Reinforcement Learning of Motor Skills in High Dimensions: A Pah Integral Approach., International Conference of Robotics and Automation (ICRA 2010).

Appendix

Task adaptation through exploration and action sequencing (2009) Bojan Nemec, Minija Tamosiunaite, Florentin Wörgötter and Ales Ude, Int. Conf on Humanoid Robots (Paris, 2009), Paper No.78., pg. 610-616.

Minija Tamosiunaite, Bojan Nemec, Ales Ude, and Florentin Wörgötter (2011). Learning to pour combining goal and shape learning for dynamic movement primitives. RAS, submitted.

9th IEEE-RAS International Conference on Humanoid Robots
December 7-10, 2009 Paris, France

Task adaptation through exploration and action sequencing

Bojan Nemec Minija Tamošiūnaitė Florentin Wörgötter Aleš Ude

Abstract— **General-purpose autonomous robots need to have the ability to sequence and adapt the available sensorimotor knowledge, which is often given in the form of movement primitives. In order to solve a given task in situations that were not considered during the initial learning, it is necessary to adapt trajectories contained in the library of primitive motions to new situations. In this paper we explore how to apply reinforcement learning to modify the subgoals of primitive movements involved in the given task. As the underlying sensorimotor representation we selected nonlinear dynamic systems, which provide a powerful machinery for the modification of motion trajectories. We propose a new formulation for dynamic systems, which ensures that consecutive primitive movements can be splined together in a continuous way (up to second order derivatives).**

I. Introduction

Action generation and trajectory modulation are among the most important issues in humanoid robot motor control. An often used paradigm is learning from demonstration or imitation learning [10], where the demonstrated action is used to seed the learning process. Due to different kinematic and dynamic capabilities of the human demonstrator and the target humanoid robot, demonstrated trajectories cannot be simply copied as sequences of joint angles [15], but need to be adapted to the capabilities of the robot. Such problems can be avoided by kinesthetic guiding [3], where the robot arm is led through the action by a human teacher, but this method is not applicable to every robot. Even after the observed trajectories have been made feasible with respect to the robot's kinematics and dynamics, they still need to be modified when the configuration of the external world changes compared to the initial demonstration. Thus a suitable, higher-level adaptation process is needed to change the learned trajectories.

The adaptation can take place in the form of an autonomous exploration, where the robot modifies the available movements by exploring its action space in the neighborhood of the previously acquired movements, thus continuously expanding the available knowledge until optimal (or satisfactory) solution is found. This process is often realized using reinforcement learning techniques. Since we are interested in the development of intelligent robots in household environments, we took the pouring of a liquid into a glass as a representative example in our evaluation experiments. Cup filling is an appropriate task for learning because in general liquid streams are hard to model if one considers arbitrary vessels and liquids. Even if the appropriate robot movements that solve the task for some glasses and liquids are available, these movements require further adaptation procedures if the relative position of the glass with respect to the robot changes or if one of the vessels changes.

When considering reinforcement learning (RL) for attaining a delayed reward, learning at different levels of abstraction is possible. One possibility is to parametrize the shape of the selected trajectory and perform learning by adapting these parameters, so that the overall movement is changed towards better performance. Examples of adaptation of this type are provided in [7], [8], using specific policy gradient technique called natural actor-critic. Another possibility is to leave the parameters specifying the shape of the trajectory as they are. Instead, higher-level parameters specifying the relationship between the movement and the task space can be adapted, e. g. by shifting the trajectory to a new location in 3-D space, this way reducing the problem to an easier reinforcement learning problem with a smaller number of parameters. This second approach was chosen in our research. We applied reinforcement learning with function approximation and continuous actions.

To generate subgoals suitable for this kind of reinforcement learning, it is often useful to segment the overall movement into primitive movements that are related to the subgoals of the task. Such methodology was also used in [2] in the context of learning from demonstration and practicing. In this paper we employ dynamic movement primitives (DMPs) [4], [11], which are essentially parametrized trajectories encoded by dynamic systems, as a basic movement representation. DMPs explicitly contain the final goal position of the primitive motion among the parameters and thus provide suitable higher-level parameters for reinforcement learning. Since a smooth transition between primitive movements without coming to a full stop is often needed to effectively sequence the primitives, we propose a new formulation of dynamic systems that ensures smoothness (up to derivatives of second order) of the transition between two consecutive primitive movements.

In the following we first study movement sequencing with DMPs. In the second part we investigate an application of reinforcement learning to the adaptation of the available motor primitives in the context of cup filling.

II. Sequencing of Motion Primitives

Lets briefly consider the task of pouring a liquid into a glass. It depends on many factors including the position of

B. Nemec and A. Ude are with the Jožef Stefan Institute, Department of Automatics, Biocybernetics, and Robotics, Ljubljana, Slovenia, bojan.nemec@ijs.si, ales.ude@ijs.si

M. Tamošiūnaitė is with the Vytautas Magnus University, Kaunas, Lithuania, m.tamosiunaite@if.vdu.lt

F. Wörgötter is with the Bernstein Center for Computational Neuroscience, Göttingen, Germany, worgott@bccn-goettingen.de

the glass with respect to the body, the shape of the vessel containing the liquid and the shape of the glass to be filled. A general strategy for pouring is 1) approach the glass to be filled with a suitable approach trajectory and 2) start the pouring motion towards the end of the approach trajectory and execute the pouring motion while controlling the liquid flow until the glass is filled to the desired level. These two phases define two separate movement primitives. Note that there is a smooth transition between the two phases, i. e. the hand motion does not come to a full stop while transitioning from the approach phase to the pouring phase.

To successfully fill the glass placed at different locations on the table, the actual pouring motion does not need to be changed. Successful pouring can be achieved solely by selecting the appropriate goal position for the approach trajectory, which is automatically taken into consideration by the underlying dynamic system, followed by the previously learned and constant pouring movement. The goal position of the approach trajectory provides the parameters that can be learned by reinforcement learning. For this approach to work, we need to be able to smoothly sequence the available primitives. In this section we propose a dynamic systems formulation that allows smooth sequencing of DMPs without coming to a full stop at the end of each movement, as it is necessary in the case of pouring movements. While some of these parameters could be inferred analytically, this is at least a non-trivial task because many parameters need to be considered (shape of the glass, properties of liquid flow for different liquids, capabilities of the robotic arm, ...).

In the standard DMP formulation for discrete movements, motion in each task coordinate is represented as a damped mass-spring system perturbed by an external force. Such a system can be modeled with a set of differential equations [11][1]

$$\begin{aligned} \dot{v} &= \frac{1}{\tau}\left(K(g-y) - Dv + f(x)\right), \\ \dot{y} &= \frac{v}{\tau} \end{aligned} \tag{1}$$

where v and y are the velocity and position of the system, x is the phase variable, which defines the time evolution of the trajectory, τ is the temporal scaling factor, K is the spring constant, and D is the damping. The phase variable x is defined by

$$\dot{x} = -\frac{\alpha x}{\tau}. \tag{2}$$

For trajectory generation it is necessary that the dynamic system is critically damped and thus reaches the goal position without overshoots. A suitable choice is $D = 2\sqrt{K}$, where K is chosen to meet the desired velocity response of the system. Function $f(x)$ is a nonlinear function which is used to adapt the response of the dynamic system to an arbitrary complex movement. A suitable choice for $f(x)$ was proposed by Ijspeert et al. [4] in the form of a linear combination of M radial basis functions

$$f(x) = \frac{\sum_{j=1}^{M} w_j \psi_j(x)}{\sum_{j=1}^{M} \psi_j(x)} x, \tag{3}$$

where ψ_j are Gaussian functions defined as $\psi_j(x) = \exp(-\frac{1}{2\sigma_j^2}(x - c_j)^2)$. Parameters c_j and σ_j define the center and the width of the j-th basis function, while w_j are the adjustable weights used to obtain the desired shape of the trajectory.

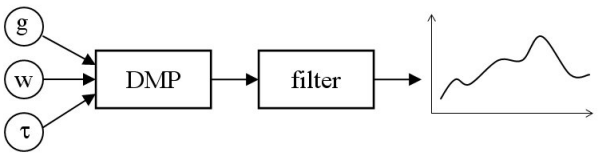

Fig. 1. Filtered output of the DMP

In the original DMP formulation [4], the system has an initial state $(x, y, v) = (1, y_0, 0)$ and a final state $(x, y, v) = (0, g, 0)$, which means that the previous motion has to completely stop before the next motion is generated. Pastor et al. [6] noted that by appropriately defining the initial conditions for the second movement, two consecutive movements can be joined together with continuous velocities. The acceleration, however, remains discontinuous even after this modification. Here we eliminate this restriction by replacing the second order system (1) with a third order system.

In order to overcome the jumps in velocities and accelerations when joining two trajectories, we propose to apply a first order low-pass filter at the output of the DMP generator, as shown in Fig. 1. The second order system (1) now turns into a third order system, which is defined by equations

$$\begin{aligned} \dot{v} &= \frac{1}{\tau}\left(K(g-y) - Dv + f(x)\right), \\ \dot{y} &= \frac{v}{\tau}, \\ \dot{q} &= \frac{H}{\tau}(y - q). \end{aligned} \tag{4}$$

Here H is an appropriately chosen filter constant and q is the new output of the modified DMP. The phase variable x remains defined by Eq. (2). Like in the original formulation (1), the third order system is stable if the constants K, D, H, and τ are appropriately selected. The proof is as follows. It is easy to see that if we omit the nonlinear term f from Eq. (4), a general solution for the remaining linear differential equations system is given by

$$\begin{bmatrix} v \\ y \\ q \end{bmatrix} = \begin{bmatrix} 0 \\ g \\ g \end{bmatrix} + \exp(t\mathbf{A})\,\mathbf{c},\ \mathbf{A} = \begin{bmatrix} -\frac{D}{\tau} & -\frac{K}{\tau} & 0 \\ \frac{1}{\tau} & 0 & 0 \\ 0 & \frac{H}{\tau} & -\frac{H}{\tau} \end{bmatrix}, \tag{5}$$

where $\mathbf{c} \in \mathbb{R}^3$ is an arbitrary constant, which is determined from the initial conditions. The system is guaranteed to

[1]While constants are denoted differently in this paper, the two formulations are equivalent.

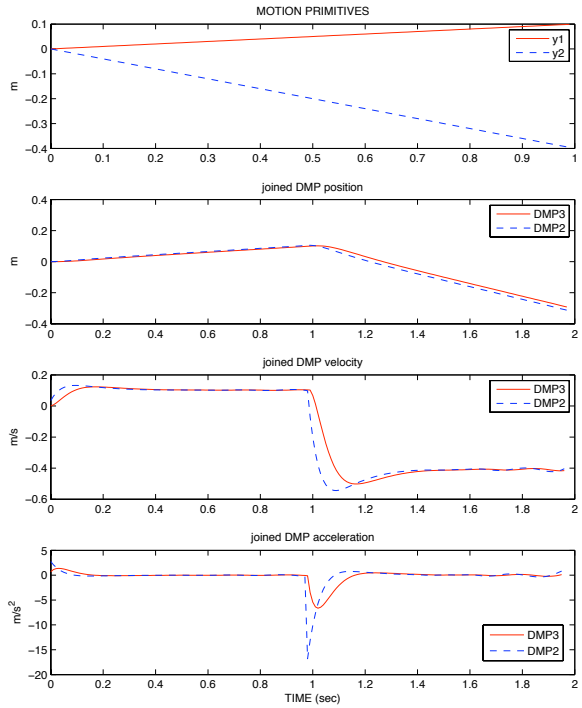

Fig. 2. Sequencing of dynamic movement primitives: the output of the second and third order DMP

converge to the attractor point $[0, g, g]$ if the eigenvalues of $\mathbf{A}$ are negative. The eigenvalues of $\mathbf{A}$ are given as solutions to the equation

$$\det(\mathbf{A}-\lambda\mathbf{I}) = -\left(\lambda^2 + \lambda D/\tau + K/\tau^2\right)(H/\tau + \lambda) = 0, \quad (6)$$

thus $\mathbf{A}$ has negative eigenvalues $-\sqrt{K}/\tau$ and $-H/\tau$ if $D = 2\sqrt{K}$, $H, K, \tau > 0$. Since the phase x and consequently the nonlinear term $f(x)$ tend to zero, the nonlinear system (4) is also guaranteed to converge to the attractor point $[0, g, g]$.

In simulation we have tested the sequencing of linear movements encoded by DMPs. Figure 2 shows the response of two consecutive second and third order DMPs and the corresponding velocities and accelerations. As we can see, the modified DMP formulation ensures continuous accelerations, whereas the accelerations in the original formulation are discontinuous.

A. Motion Acquisition

The trajectory represented by a third order dynamic system is parameterized with the initial acceleration, velocity, and position, the final goal position, and a set of weights w_j associated with radial basis functions. In this section we present the procedure for the calculation of weights w_j. We assume that from human demonstration or kinesthetic guiding we obtain trajectory data points $\{q_d(t_i), \dot{q}_d(t_i), \ddot{q}_d(t_i), \dddot{q}_d(t_i)\}_i$, $t_i \in [0, T]$. Note that unlike in the original DMP formulation, this formulation requires

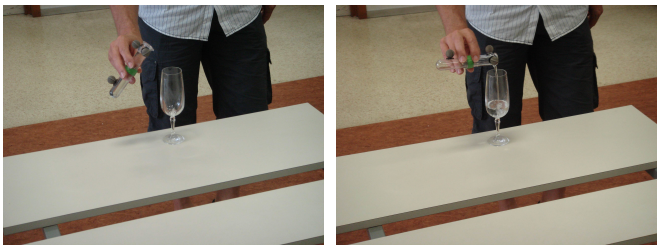

Fig. 3. Two frames from the pouring demonstration. The motion was captured by attaching markers to the container.

also to estimate the jerk. We define function f^* as follows

$$f^*(t) = \frac{\tau^3}{H}\dddot{q} + \tau^2\left(1 + \frac{D}{H}\right)\ddot{q} + \tau\left(\frac{K}{H} + D\right)\dot{q} + K(q-g), \quad (7)$$

where $q = q(t)$. This function is obtained by replacing the system of three first order equations (4) with one equation of the third order, where the nonlinear term $f(x)$ has been omitted. Our task is to find a set of weights $\{w_j\}$ that minimize the quadratic cost function

$$J = \sum_{i=0}^{N}(f^*(t_i) - f(x(t_i)))^2. \quad (8)$$

We use global regression methods to find the optimal weights w_j. Other authors [4], [6] applied locally weighted regression, which instead minimizes M separate cost functions

$$J_j = \sum_{i=0}^{N}\psi_j(x(t_i))(f^*(t_i) - w_j x(t_i))^2, \quad (9)$$

$j = 1, \ldots, M$. Locally weighted regression [1] was proposed as a method that prevents negative interference between task models. Local models are used to generalize in the neighborhood of the given data point. However, in the context of trajectory generation from human demonstration, a complete trajectory is observed over the entire time interval, therefore locally weighted regression has no advantages over the global regression except for the lower computational burden. On the other hand, when using global regression, significantly less kernel functions are necessary to encode the trajectory within the required precision and consequently less computation is required to track the previously calculated trajectory. Global regression results in the following linear system of equations

$$\mathbf{A}\mathbf{w} = \mathbf{f}^*, \quad (10)$$

$$\mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_M \end{bmatrix}, \quad \mathbf{f}^* = \begin{bmatrix} f^*(t_0) \\ \vdots \\ f^*(t_N) \end{bmatrix},$$

$$\mathbf{A} = \begin{bmatrix} \frac{\psi_1(x(t_0))x(t_0)}{\sum_{j=1}^{M}\psi_j(x(t_0))} & \cdots & \frac{\psi_M(x(t_0))x(t_0)}{\sum_{j=1}^{M}\psi_j(x(t_0))} \\ \vdots & \vdots & \vdots \\ \frac{\psi_1(x(t_N))x(t_N)}{\sum_{j=1}^{M}\psi_j(x(t_N))} & \cdots & \frac{\psi_M(x(t_N))x(t_N)}{\sum_{j=1}^{M}\psi_j(x(t_N))} \end{bmatrix}.$$

Similarly as in the case of locally weighted regression, it is possible to compute a solution to (10) recursively by incrementally updating the following quantities

$$\mathbf{P}_i = \mathbf{P}_{i-1} - \frac{\mathbf{P}_{i-1}\mathbf{a}_i\mathbf{a}_i^T\mathbf{P}_{i-1}}{1+\mathbf{a}_i^T\mathbf{P}_{i-1}\mathbf{a}_i}, \quad (11)$$

$$\mathbf{w}_i = \mathbf{w}_{i-1} + (f^*(t_i) - \mathbf{a}_i^T\mathbf{w}_{i-1})\mathbf{P}_i\mathbf{a}_i, \quad (12)$$

where $\mathbf{a}_i$ is the M dimensional column vector associated with the corresponding row of the matrix $\mathbf{A}$ and the optimal weights are $\mathbf{w} = \mathbf{w}_N$.

Using the proposed estimation method, we captured and successfully reconstructed the approach and the pouring movements (see Fig. 3). An optical tracking system with passive markers attached to the container was utilized to capture the motion of the container. The motion of the container was then mapped to the 3-D space motion of the robotic hand.

III. Goal Configuration Adaptation Using Reinforcement Learning

Now we turn our attention towards the adaptation of the available action knowledge through exploration. In the interest of better understanding of the learning process, we fist describe our experimental setup. The implemented learning process is, however, more general and is not limited to this experiment.

A. Experimental setting

The evaluation experiments were performed using two robots; a humanoid robot HOAP-3 and a seven degrees of freedom (DOFs) robotic arm PA-10. HOAP-3 is a small humanoid robot, whose arm has a limited workspace (5 DOFs), while the seven degress of freedom available to PA-10 allow the arm to reach any desired position and orientation of the wrist within the robot workspace.

The execution of the pouring action consists of two phases; the approach movement and the actual pouring movement. The pouring movement of HOAP-3 was fixed and was not changed during the exploration process. What the robot needed to learn was the optimal position and orientation of the robot's hand from where to start the pouring movement

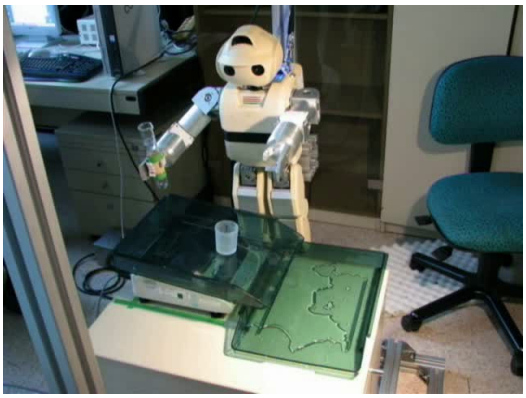

Fig. 4. Experimental setup with HOAP-3 humanoid robot

(with respect to the location of the glass to be filled). This parameter is given as a final position on the approach trajectory and is explicitly encoded within a DMP. Obviously, the optimal starting position for pouring changes when the glass is moved to a new location. However, because the arm has only 5 degrees of freedom and because of nonlinearity of the robot's kinematics, the new optimal position is not a linear function of the glass location. The task space to be explored was further limited by specifying the height from which to start pouring, which was defined as a function of the distance of the glass from the robot body using the formula

$$z = 0.5((x-0.1) + (-0.14-y)) + 0.01, \quad (13)$$

where (x, y, z) are the robot-centered coordinates with the following directions (x - forward, y - to the side, z - up). Thus the wrist was kept lower when being close to the body, and higher going away from the body. This was important to move the arm at sensible orientations because the 5 degrees of freedom arm of HOAP-3 cannot reach every desired configuration in the 6-D task space. The 3-D wrist orientation was defined so that the glass was not tilted at the beginning of the pouring movement, which fixed the remaining two degrees of freedom.

The reinforcement learning process described in the next sections explored the so defined planar surface to find the optimal position for pouring with respect to the given glass location. The outcome of pouring, which provided the reward for reinforcement learning, was measured using a scale that weighted the amount of liquid that remained in the glass to be filled (see Fig. 4). The scale could be replaced by a vision system measuring the level of liquid in the glass, but this was not the focus of our investigation.

We also evaluated our algorithms with a PA-10 robot arm. Since the problem to keep the wrist at a specific angle does not arise when using a 7 degrees of freedom arm, we now allowed variable height in addition to the y coordinate (direction to the side of the robot), but x coordinate (forward) was kept fixed. This coordinate can be estimated by vision. The 3-D wrist orientation was defined so that the glass was not tilted at the beginning of the pouring motion and the wrist axis orientation was parallel to the robot's sagittal plane.

B. Exploration

We implemented a reinforcement learning method with function approximation. We define the value function $V(s)$ as follows

$$V(s) = \sum_{k=1}^{N} \theta^k \Phi^k(s) / \sum_{k=1}^{N} \Phi^k(s) \quad (14)$$

where $\Phi^k(s)$ is the activation function of the k-th kernel in state s, θ^k are the weights associated with the k-th kernel function, and N is the overall number of kernels in the system. In the context of the our experiment, state s is defined as a pair of Cartesian coordinates at which the robot starts pouring the liquid ($[x, y]^T$ and $[y, z]^T$ for HOAP-3 and PA-10, respectively). Weights θ are adapted through learning

as described in the next section. We used spherical Gaussian kernels, uniformly distributed over the analyzed area with $\sigma = 0.5\ cm$ for the HOAP-3 setup and $\sigma = 2.1\ cm$ for the PA-10 setup. $N = 2000$ was used in both cases to thoroughly exclude effects from insufficient coverage of the state space, although according to our experience with function approximation methods of similar complexity [13], ten times smaller amount of kernels would often suffice to adequately represent the investigated space.

To define a new exploratory action, gradient of the current estimate of the value function was calculated $\Delta = \mathrm{grad}\, V(s)$, and an action was performed taking the direction of the gradient into account. Instead of using pure gradient ascent, the update for the next state was calculated as a combination of the current gradient and the previous action $A_{previous}$:

$$\Delta_{current} = (\Delta x, \Delta y)_{current}, \tag{15}$$

$$A_{previous} = (Ax, Ay)_{previous}, \tag{16}$$

which resulted in

$$A_{final} = c\Delta_{current} + (1 - c)A_{previous}, \tag{17}$$

where at the beginning of learning $c = 1.5$ was used, while in later trials c was reduced by 0.1 trial by trial. The proposed smoothing procedure helps to avoid jerky exploratory movements in the beginning of the learning process and also to some degree influences the process of refinement in RL with function approximation. Traditional proofs of convergence for RL are no longer valid when using action smoothing, but in practice learning converged reliably.

In the beginning of learning, the probability of random exploration was set to 0.5 due to the fact that the chosen method for performing continuous actions in the value function approximation scheme was sometimes attracted towards local minima. The exploration was diminished by 0.05 in each trial to adhere to the learned components better.

C. Learning method

In the proposed learning framework, the change in the value of θ^k follows the mean across all activated kernels of the next state s'

$$\theta^k = \theta^k + \mu[r + \gamma V(s') - \theta^k]\Phi_N^k(s), \tag{18}$$

where k is the number of the kernel to which the weight is associated, r is the reward, $\mu < 1$ the learning rate, $\gamma < 1$ the discount factor, $V(s')$ is the value of the next state, and $\Phi_N^k(s)$ is the normalized activity function for kernel k in state s

$$\Phi_N^k(s) = \Phi^k(s) / \sum_{k=1}^{N} \Phi^k(s). \tag{19}$$

The rule (18) is called averaging function approximation rule and is considered to perform more stably [9] in function approximation schemes as compared to standard methods (e. g. see [12]).

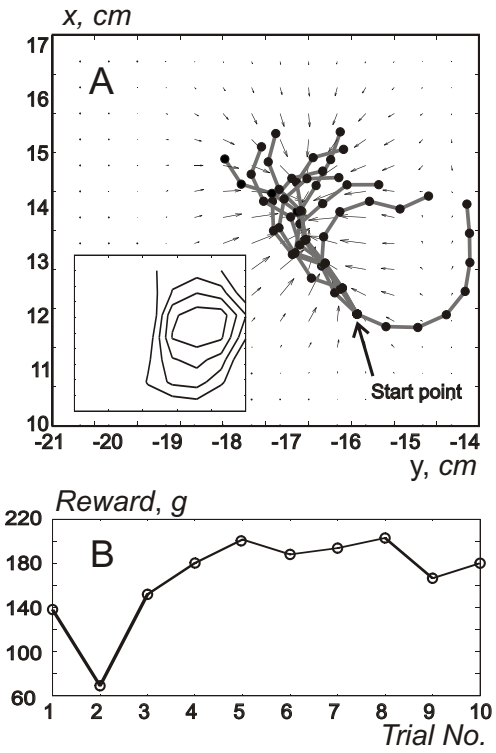

Fig. 5. Learning trajectories, vector field obtained in the process of learning (marked with pointers), and the contour plot of the reward (succesful pouring amounts) obtained through sampling (A); amount of reward accumulated in successive trials (B). Real HOAP-3 robot was used in this case.

D. Results

In experiments with HOAP-3, ten trials with eight pouring attempts each were performed. Exploratory paths together with the resultant vector field are shown in Fig. 5.A. All trials started at the same initial hand position and orientation. One can see that the learning was successful and the rewards starting with the 3rd trial were always high (see Fig. 5.B). Some oscilations in the reward profile are due to the random exploration component.

Several improvements were introduced to make learning quicker: 1) restarting the next trial from the best point of the previous trial; 2) shortening the step size with the number of trials; and 3) reducing the amount of smoothing of the trajectory with the number of trials. This led to a more reliable learning both in simulation and on the robot. The exploratory paths at the end of learning together with the obtained vector field are shown in Fig. 6.A. Statistics plots are shown in Fig. 6.B.

A similar experiment was performed using the PA-10 robot arm. Examples of the exploratory paths are shown in Fig. 7. In Fig. 7.A, a few initial paths are shown, while in Fig. 7.B, exploratory paths at the end of learning and the obtained vector field, are shown. The inset of Fig. 7.A shows the reward contours obtained through sampling. One can see from the inset that the same reward is obtained independently of the wrist height (z coordinate). Learning chooses an arbitrary height but prefers the appropriate y coordinate. Vector field

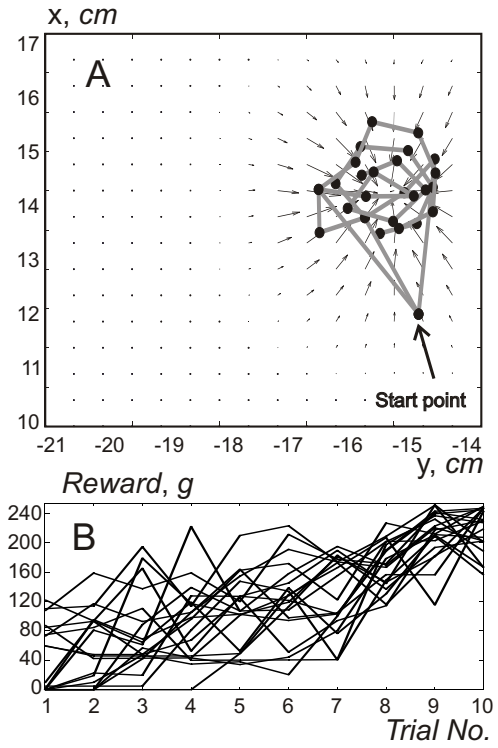

Fig. 6. Exploratory paths of the hand traversed during learning and the vector field obtained at the end of learning when using the strategy of jumping directly to the best point of the previous trial (A); sequences of accumulated reward for 20 learning experiments obtained using HOAP-3 simulator (B).

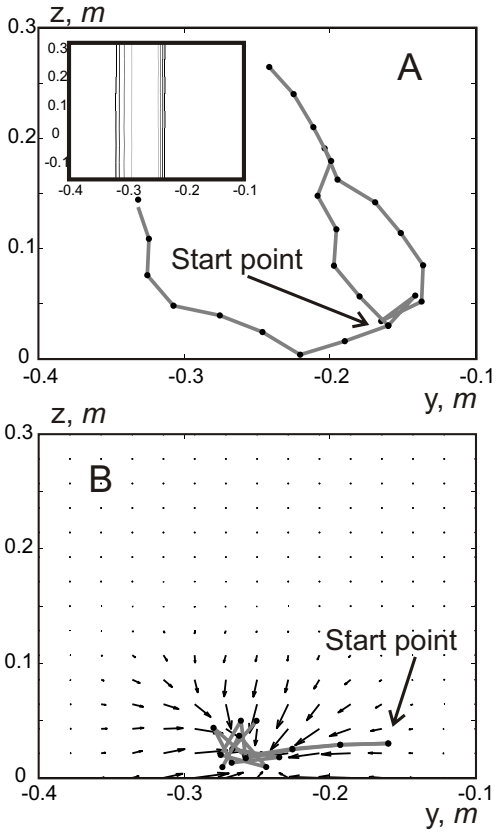

Fig. 7. Exploratory paths obtained at the beginning of learning in the experiment with PA-10 (A). The inset shows contours of the reward area obtained using sampling. Exploratory paths and vector field obtained at the end of learning are shown in (B).

is also stronger along the y coordinate as compared to the height coordinate.

In PA-10 experiments, good performance was obtained after 6-8 trials. In Fig. 8.A accumulated rewards over sequential trials from 10 experiments are shown. The average over those experiments is given in Fig. 8.B. In the current setup with PA-10, learning effects were achieved several trials later as compared to the analyzed HOAP-3 setup, which is probably due to the bigger area that needs to be explored.

IV. Summary and Conclusion

A reinforcement learning procedure with function approximation and continuous actions, which are encoded by dynamic systems, was developed. To support action sequencing, a new formulation for dynamic systems was proposed. This methodology was successfully applied to learn appropriate robot movements for liquid pouring. We have shown that good performance can be achieved within 3-8 trials (exploratory paths, or 20 to 60 attempts to pour), on two different robot arms. The number of trials is acceptable for a robotic application, where procedures with thousands of learning trials are often not feasible.

There are alternative ways to refine robot movements using reinforcement learning. In [8], a novel policy gradient method called *natural actor-critic* was used to solve the problem of throwing a ball into a cup with a 7 degrees of freedom robot arm. The method exhibits excellent performance when applied to problems involving multiple dimensions. One problem we had with the natural actor-critic technique – at least in our experiments – was that the initial trajectory had to be relatively close to the desired trajectory if one wants to achieve efficient convergence [14]. In comparison, approaches like ours need to explore a smaller task space to find a satisfactory solution and are therefore less susceptible to a bad initial approximation. Also, value function based reinforcement learning methods deal with delayed rewards in a systematic way, whereas gradient based methods like the natural actor-critic would fail because of the flat reward surface. This is another reason for why our method can correct larger inaccuracies in the demonstration as compared to the natural actor-critic. Hence although with methods like natural actor-critic, reinforcement learning can be applied to large state spaces, an alternative to divide the task into several smaller problems and solve one or several of the smaller tasks with reinforcement learning may provide more stable results. While other systems in which the task has been subdivided into smaller tasks by defining suitable subgoals were developed in the past [5], our approach shows how to refine the available movement primitives by encoding them as dynamic systems. We demonstrated that dynamic systems provide a suitable framework for task decomposition and proposed a formulation that allows smooth sequencing of

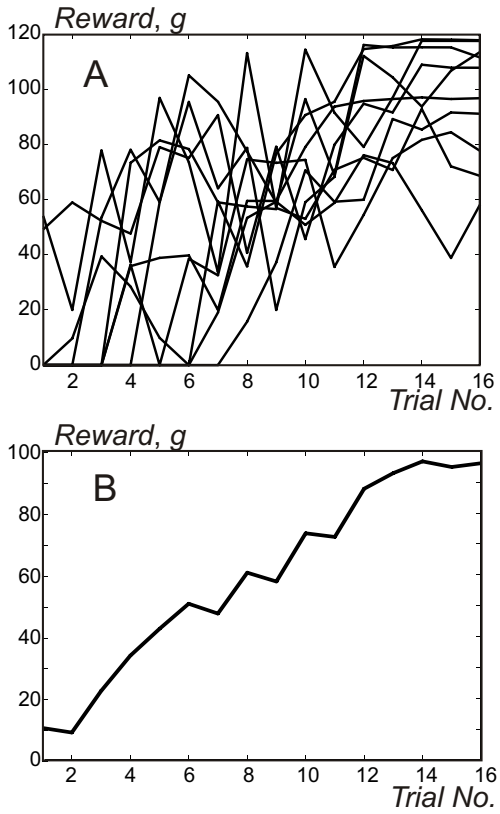

Fig. 8. Ten sequences of accumulated reward in PA-10 experiments (A); average value of the ten reward sequences (B).

consecutive movement primitives up to second-order derivatives.

Acknowledgment: The work described in this paper was conducted within the EU Cognitive Systems project PACO-PLUS (FP6-2004-IST-4-027657) funded by the European Commission.

REFERENCES

[1] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning," *AI Review*, vol. 11, pp. 11–73, 1997.

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

[3] M. Hersch, F. Guenter, S. Calinon, and A. Billard, "Dynamical system modulation for robot learning via kinesthetic demonstrations," *IEEE Trans. on Robotics*, vol. 24, no. 6, pp. 1463–1467, 2008.

[4] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Movement imitation with nonlinear dynamical systems in humanoid robots," in *Proc. IEEE Int. Conf. Robotics and Automation*, Washington, DC, 2002, pp. 1398–1403.

[5] J. Morimoto and K. Doya, "Acquisition of stand-up behavior by real robot using hierarchical reinforcement learning." *Robotics and Autonomous Systems*, vol. 36, pp. 37–51, 2001.

[6] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal, "Learning and generalization of motor skills by learning from demonstration," in *Proc. IEEE Int. Conf. Robotics and Automation*, Kobe, Japan, 2009, pp. 763–768.

[7] J. Peters and S. Schaal, "Reinforcement learning for parameterized motor primitives," in *International Joint Conference on Neural Networks*, 2006, pp. 73–80.

[8] ——, "Reinforcement learning of motor skills with policy gradients." *Neural Networks*, vol. 21, pp. 682–697, 2008.

[9] S. I. Reynolds, "The stability of general discounted reinforcement learning with linear function approximation." in *UK Workshop on Computational Intelligence (UKCI-02)*, September 2002, pp. 139–146.

[10] S. Schaal, "Is imitation learning the route to humanoid robots?" *Trends in Cognitive Sciences*, vol. 3, no. 6, pp. 233–242, 1999.

[11] S. Schaal, P. Mohajerian, and A. Ijspeert, "Dynamics systems vs. optimal control – a unifying view," *Progress in Brain Research*, vol. 165, no. 6, pp. 425–445, 2007.

[12] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction.* Cambridge, MA: MIT Press, 1998.

[13] M. Tamosiunaite, J. Ainge, T. Kulvicius, B. Porr, P. Dudchenko, and F. Wörgötter, "Path-finding in real and simulated rats: assessing the influence of path characteristics on navigation learning." *Journal of Computational Neuroscience*, vol. 25, pp. 562–582, 2008.

[14] M. Tamosiunaite, T. Asfour, and F. Wörgötter, "Learning to reach by reinforcement learning using a receptive field based function approximation approach with continuous actions." *Biological Cybernetics*, vol. 100, no. 3, pp. 249–260, 2009.

[15] A. Ude, C. G. Atkeson, and M. Riley, "Programming full-body movements for humanoid robots by observation," *Robotics and Autonomous Systems*, vol. 47, no. 2-3, pp. 93–108, 2004.

Learning to pour combining goal and shape learning for dynamic movement primitives

Minija Tamošiūnaitė[1,2], Bojan Nemec[3], Aleš Ude[3] and Florentin Wörgötter[1]

[1]University Göttingen, Institute for Physics 3 - Biophysics and Bernstein Center for Computational Neuroscience, Göttingen, Germany

[2]Vytautas Magnus University, Kaunas, Lithuania

[3]Jožef Stefan Institute, Department of Automatics, Biocybernetics, and Robotics, Ljubljana, Slovenia

m.tamosiunaite@if.vdu.lt;{bojan.nemec,ales.ude}@ijs.si;worgott@bccn-goettingen.de

Keywords: Reinforcement Learning, PI^2-method, Natural Actor Critic, Value Function Approximation, Dynamic Movement Primitives

Abstract

When describing robot motion with dynamic motion primitives (DMPs) one employs two types of parameters: "goal" position, towards which the movement is attracted; and parameters for the non-linear part of the DMP-equations (called "weights"), which determine the shape of the generated trajectory. When considering reinforcement learning (RL) with DMPs, usually goals are pre-defined and only the weights for shaping a DMP are subject to learning. Many tasks, however, exist where the best goal position is not a priori known, requiring to learn it. Here we chose the task of learning to pour liquid from one container into another. Due to the turbulent motion of the liquid, it is very difficult to "guess", which goal position should be used allowing for appropriate trajectory shape learning. Thus, using this task as an example in this study we specifically address the question of how to simultaneously combine of goal and shape parameter learning. As such this is a difficult problem because goal and shape parameters could easily interfere in a destructive way jeopardizing convergence of learning. Value function approximation RL techniques are used for goal-, and policy gradient RL methods for shape learning. Specifically, we use "policy improvement with path integrals (PI^2)" and "natural actor-critic (NAC)" for the policy gradient approach. The methods are analyzed with simulations using a pouring model and are implemented on a real robot setup using the Mitsubishi Pa10 robot arm. Results for learning from scratch, learning calibration starting from a human demonstrated trajectory, as well as learning re-calibration using a different different container for the liquid are presented. We observe that the combination of value function approximation based goal-learning together

with PI^2 shape learning is stable and robust within large parameter regimes, where we had tested up to six goal and shape combinations. Learning converges even in the presence of large disturbances in about 20-40 movement trials, which makes this combined method suitable for robotic applications.

1 Introduction

Dynamic movement primitives (DMPs) proposed by Ijspeert et al. (2002) have become one of the most widely used tools for the generation of robot movements. Numerous applications can be found in the literature (Perk and Slotine, 2006; Peters and Schaal, 2008; Kober and Peters, 2009; Theodorou et al., 2010). The DMP formalism is employed for describing goal-directed movements and includes second order dynamics towards an attractor point, called the goal point g of the movement, as well as several adjustable parameters, which are used to obtain the desired shape of the trajectory. In this study we will consider questions of robot reinforcement learning using dynamic movement primitives. Several efficient methods have been proposed for DMP shape parameter learning. These include the natural actor-critic (NAC, Peters and Schaal (2008)), policy improvement with path integrals (PI^2, Theodorou et al. (2010)), policy learning by weighting explorations with the returns, (PoWER, Kober and Peters (2009)). Using those methods, robots were trained to acquire specific skills, for example jumping across a gap by a robot dog (Theodorou et al., 2010), hitting a baseball ball with a robot arm (Peters and Schaal, 2008), or playing the ball-in-a-cup game using a humanoid robot (Kober and Peters, 2009).

Here we will consider the combination of DMP goal *and* shape learning. DMP goal learning was not much considered in robot experiments before (Peters et al., 2009) and the simultaneous combination of the two learning regimes is novel. The reason for this is that in most tasks considered so far the goal position is well-enough known. Thus, goal learning is not required. There are, however, many tasks where this is not the case. These include all tasks where it is next to impossible to pre-define the most appropriate goal position, which happens as soon as the goal has a hard-to-predict effect on the outcome. One example, which is also in the core of the current study, is pouring of liquid. The complex turbulent motion of the liquid taking place at the rim of the container makes it very hard to predict at what position (=goal) the container should be optimally placed for best pouring results. The same is true for other tasks,

like optimally hitting a ball with a tennis racket. Here the situation is a little bit better and target goal (as well as angle) can be calculated for an ideal racket. However, as soon as the tension of the cords is not optimal anymore these calculations might not reflect the optimal goal and goal-learning would again be the better strategy.

We will use function approximation reinforcement learning (Sutton and Barto, 1998; Reynolds, 2002) for goal learning and PI^2 or NAC for shape learning. It will be shown how two learning processes can be combined in the same learning experiment without mutually jeopardizing convergence.

As our example task we will teach the robot to pour. For this, the robot has a container with liquid in its hand and executes approach as well as pouring into an empty glass positioned on a table. As mentioned, the pouring task is a good example where both shape and goal of the trajectory need to be learned. The shape parameters determine the best way of curving the trajectory during the approach and while tilting the container. The goal parameters, on the other hand, determine the target position of the trajectory. This task was chosen as it represents an example of a generic set of learning tasks which often occur especially in service robotics. There one often finds "fuzzy" tasks of a kind where many successful solutions exist and where highest accuracy (such as that needed for industrial robots) is not required.

The paper is organized in the following way: in section 2 the setups and the methods used in this study will be presented. In section 3 results obtained with the pouring simulator as well as those obtained on the Mitsubishi Pa10 robot arm will be provided. With the pouring simulator a more detailed scan of the parameter space is possible. With the real robot experiments both, learning from scratch as well as starting to learn from a demonstrated human trajectory, will be presented. To demonstrate the potential of the applied learning algorithm also a more complex (redundant) learning task is analyzed using the pouring simulator. In section 4 the advantages as well as shortcomings of the methods will be analyzed, and comparisons with alternative approaches will be provided. In the Appendix algorithms will be described in more detail.

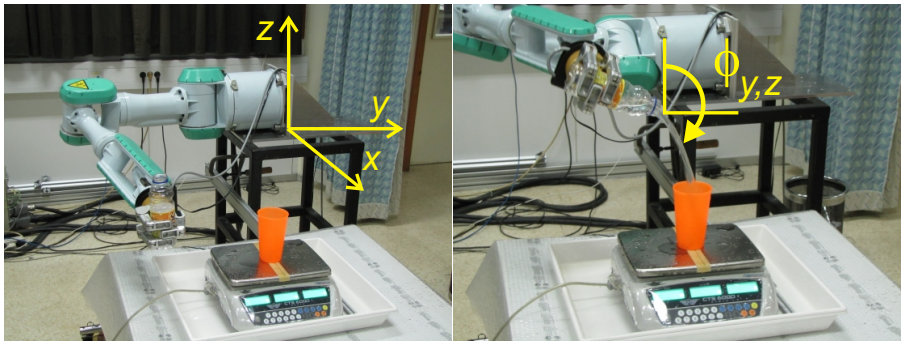

Figure 1: Robot setup: Mitsubishi Pa10 robot arm and the scales for measuring correctly poured water. Spilled water runs off the scales. On the left side the coordinate system used to evaluate the wrist position is shown. The right picture provides a close to planar view onto the (y, z) plain and the tilting angle ϕ is visualized in this plain. Rotation is performed around the neck of the bottle.

2 Methods

Note a summary of the parameter values used in this study is provided in Table 1.

2.1 General Setup

The task is to learn pouring liquid from one container into another using a robot arm. We use the assumption that the robot already has a full container in its hand and grasping the container is not included in the task. However, the correct pouring position is considered to be unknown. The pouring action should include both, the approach to the target container as well as tilting of the manipulated container to get the liquid out.

As a platform for the experiments the 7DOF Mitsubishi Pa10 robot arm was used. The arm was positioned so as to imitate the position of a human arm (see Fig. 1). The whole pouring procedure was performed in the following way: the robot arm was brought to the start position where the container was filled with water (approx 190 ± 10 grams). Afterwards, the pouring action was executed. Pouring success was determined by the changing mass of the lower container measured by a scales. The setup is such that all spilled water immediately runs off the scales and is, thus, not weighed.

Parameter T defines the number of samples in a trajectory. In the real robot experiments we had $T = 300$. This corresponds to pouring movements of 5 s duration (sampling frequency 60 Hz).

Table 1: Parameters used for simulation and real robot experiments.

Parameter type	**Parameter name**	**Simulation**	**Real robot**
Robot workspace	wrist side displacement $y(t)$	[-0.4, -0.1]	[-0.35, -0.1]
	wrist height $z(t)$	[-0.15, 0.15]	[-0.1, 0.1]
	tilting angle $\phi(t)$	$[0,\ \pi]$	$[0,\ \frac{7}{9}\pi]$
DMP	K	36	36
	D	3	3
	α	0.1	0.1
	τ	1	2
	No. of kernels L	15	15
	kernel width σ_D	0.005	0.005
	integration step dt	0.01 s	0.017 s
	trajectory length T	120	300
Value function approximation	No of kernels N in 2D	200	200
	No of kernels N in 3D	2000	n.a.
	kernel width σ_V	2 cm	2 cm
	learning rate μ	0.7	0.7
	discount factor γ_V	0.7	0.7
	initial straightening c	1.5	1.5
	c reduce per epoch	0.1	0.1
	initial exploration p	0.5	0.5
	p reduce per epoch	0.05	0.05
	step length	2.2 cm	2.2 cm
	step decay per epoch (starting after 2 epochs)	0.85	0.85
PI^2 procedure	No of trials in an epoch	8	8
	No of epochs performed	9	9
	No of trials in exploration K	7	7
	No of trials in testing	1	1
	Noise variance σ_P	30 (opt.) 80 130	30 40
	learning rate	1 3 10 (opt.)	3 5
	control penalty $\boldsymbol{R}$ (diagonal element)	10^{-9}	10^{-9}
NAC procedure	No of trials in an epoch	17	n.a.
	No of epochs performed	6	n.a.
	No of trials in exploration	7	n.a.
	No of trials in probing for γ_N	10	n.a.
	initial variance σ_N	10	n.a.
	σ_N decay factor per epoch	0.85	n.a.
	interval of γ_N	[0.1, 1]	n.a.

2.2 DMPs

Movements were generated using dynamic movement primitives (Schaal et al., 2007).

$$\begin{aligned} \dot{v}_D &= \frac{1}{\tau}\left(K(g - y_D) - Dv_D + f(x_D)\right), \\ \dot{y}_D &= \frac{v_D}{\tau}, \\ \dot{x}_D &= -\frac{\alpha x_D}{\tau} \end{aligned} \tag{1}$$

where v_D and y_D represent velocity and position of the system, x_D is the phase variable, which defines the time evolution of the trajectory, τ is the temporal scaling factor, K is the spring constant, and D the damping. here we use variables labeled with D (shortcut for "DMP"), to distinguish those from the variables used for notation of coordinates in the robot setup.

The non-linear part $f(x_D)$ was defined using radial basis functions (Ijspeert et al., 2002):

$$f(x_D) = \frac{\sum_{l=1}^{L} \omega_l \psi_l(x_D)}{\sum_{l=1}^{L} \psi_l(x_D)} x_D, \tag{2}$$

where ψ_l are Gaussian functions defined as $\psi_l(x_D) = \exp(-\frac{1}{2\sigma_{D,l}^2}(x_D - c_l)^2)$. Parameters c_l and $\sigma_{D,l}$ define the center and the width of the l-th basis function, while w_l are the adjustable weights used to obtain the desired shape of the trajectory.

Three DMPs were used: one for the horizontal side displacement, $y(t)$; one for the vertical displacement, $z(t)$; and the last one for the tilting angle parallel to the frontal plane, $\phi(t)$. To simplify the situation, the horizontal forward displacement was kept fixed and the tilting angle was constrained to a plane. Learning of the goal parameters g of $y(t)$ and $z(t)$ was performed, but for the tilting angle $\phi(t)$ not the goal but the shape ω_i was learned. Note, shapes of the horizontal and the vertical displacement of the wrist $y(t)$ and $z(t)$ were kept unchanged, using the initial trajectories of our second-order linear system, as we wanted to keep the setup non-redundant. The DMP trajectory for the tilting angle $\phi(t)$ was hard-limited between zero and $\frac{7}{9}\pi$ in the real robot implementation to avoid unrealistic actions (negative tilt) and unreachable configurations for too big tilt angles.

To show the potential of the learning methods used, we have in addition implemented learning in a more complex setup, where goal parameters as well as shape parameters were learned for all the three DMPs: $y(t)$, $z(t)$ and $\phi(t)$. That is, six entities were learned: three goals and three weight sets. Even though simpler formulation of the pouring task as described before is better for application on a robot, as it is non-redundant, the more complex formulation can show interactions of the employed learning algorithms better and provide results more interesting from the theoretical point of view.

We used parameters $K = 36$, $D = 3$, $\alpha = 0.1$, and $\tau = 1$ or $\tau = 2$ as well as step $dt = 0.017$ for the Euler integration. The value of α was kept small to obtain an almost constant variable x_D so as to obtain strong influence of kernels for the whole length of the trajectory. In the DMP framework the variable x_D is used to shape the influence of the non-linear part of the

equations over time (Eq. 1)). In other studies α was kept higher, as usually in those tasks it was important to have the biggest shaping effects in the beginning of the trajectory, but not at the end (Schaal et al., 2004; Kober and Peters, 2009). Consequently, the variable x_D had in these cases been strongly dampened to get close to zero at the end of the trajectory. In our pouring task, on the other hand, the shape of the tilting of the glass had to be considered along the whole length of the trajectory and thus x_D was kept (almost) constant. We used $L = 15$ kernels in the non-linear part of the DMP.

A bottle was used as the container from which the water was poured and a glass as the target container (Fig. 1). In addition we have evaluated the learning process in the case when initial shapes of the trajectories were obtained from human demonstration and adaptation to the robot setup (calibration) was required from the learning. Also we were testing the capability of the algorithm to relearn (recalibrate), giving the robot a different bottle after the movement was learned with the first bottle.

2.3 Simulation of the pouring process

To pre-tune the algorithm, as well as to gather statistics for comparing different learning methods, we were using a pouring model. The model was designed to simulate liquid pouring from a bottle. Note, no attempts were made to make this model fully accurate, which is very difficult, due to the problem of having to model turbulent flow. The level of model detail was matched to our purpose of being able to compare different learning methods with a high enough degree of trust. According to the Bernoulli equation, the exit velocity of the liquid equals

$$v_o = \begin{cases} \sqrt{(2gh_l)}, & h_l > 0 \\ 0, & h_l <= 0, \end{cases} \tag{3}$$

where g is the gravitational acceleration and h_l is the liquid level according to Fig. 2. The direction of the v_0 is determined by the bottle pouring angle φ. Hence, the liquid velocity components, v_y and v_z, are

$$\begin{aligned} v_y &= v_0 sin(\varphi) \\ v_z &= v_0 cos(\varphi) + gt, \end{aligned} \tag{4}$$

where t denotes time.

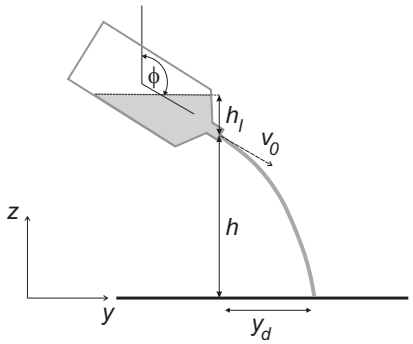

Figure 2: Pouring model setup

The predicted displacement of the liquid flow with respect to the bottle neck in y direction is thus

$$y_d = v_y t \tag{5}$$
$$t = \frac{-v_0 cos(\varphi)) + \sqrt{(v_0 cos(\varphi)^2 + 2gh}}{g}$$

and the liquid flow is

$$\phi = A v_0, \tag{6}$$

where A is the cross sectional area of the liquid at the bottle neck.

Even though the model presents a simplified scheme it allowed analyzing and pre-tuning the learning algorithms to the degree that they could be successfully applied on the real robot.

With the pouring model we used a trajectory duration of $T = 120$. We used coarser trajectories in the simulation experiments to reduce run time for obtaining larger statistics for the different parameters.

2.4 Reinforcement learning methods

Two different reinforcement learning approaches are of relevance here: (1) value function based RL, where for each state the value of being in the state is determined by learning (Sutton and Barto, 1998) and (2) policy gradient methods, where one does not keep information about values of states, but instead directly introduces action parametrization where parameter updates follow the gradient of the return. Throughout learning the parameters are adjusted to optimize the

desired goal directed behavior (Baxter and Bartlet, 2001). We are combining both techniques, where the value function approximation approach was used for goal learning, and the policy gradient approach for shape learning.

2.4.1 Goal learning method

For our value function approach we used a reinforcement learning method with function approximation developed by us in an earlier study (Tamosiunaite et al., 2009) and modified as described next. The value function $V(s)$ is defined as follows

$$V(s) = \sum_{k=1}^{N} \theta^k \Phi^k(s) / \sum_{k=1}^{N} \Phi^k(s) \tag{7}$$

where $\Phi^k(s)$ is the activation function of the k-th kernel in state s, θ^k are the weights associated with the k-th kernel function, and N is the overall number of kernels in the system. In the context of our experiment, every state s is defined as a pair of Cartesian coordinates, thus all coordinates (y, z) denote possible goal points of a DMP (parameter g in eq. 1). Weights θ are adapted by learning as described later. We used spherical Gaussian kernels, uniformly distributed over the analyzed area with $\sigma_V = 2\ cm$. A value of $N = 200$ was used in 2D case (non-redundant setup) and $N = 2000$ was used in 3D case (redundant setup).

To define a new exploratory action, the gradient of the current estimate of the value function was calculated $\Delta = \text{grad}\, V(s)$, and an action was performed taking the direction of the gradient into account. Instead of using pure gradient ascent, the update for the next state was calculated as a combination of the current gradient and the previous action $A_{previous}$:

$$\Delta_{current} = (\Delta y, \Delta z)_{current}, \tag{8}$$

$$A_{previous} = (Ay, Az)_{previous}, \tag{9}$$

which resulted in

$$A_{final} = c\Delta_{current} + (1 - c)A_{previous}, \tag{10}$$

where at the beginning of learning $c = 1.5$ was used, while in later trials c was reduced by 0.1 in each epoch. The proposed smoothing procedure helps avoiding jerky exploratory movements in the beginning of the learning process and also to some degree influences the process of

refinement in RL with function approximation. Traditional proofs of convergence for RL are no longer valid when using action smoothing, but in practice learning converged reliably.

In the beginning of learning, the probability of random exploration was set to 0.5 due to the fact that the chosen method for performing continuous actions in the value function approximation scheme was sometimes attracted towards local minima. Exploration was reduced by 0.05 in each epoch to increasingly better adhere to the learned components.

In the proposed learning framework, the change in the value of θ^k follows the mean across all activated kernels of the next state s'

$$\theta^k = \theta^k + \mu[r + \gamma_V V(s') - \theta^k]\Phi_N^k(s), \tag{11}$$

where k is the number of the kernel to which the weight is associated, r is the reward, $\mu < 1$ the learning rate, $\gamma_V < 1$ the discount factor, $V(s')$ is the value of the next state, and $\Phi_N^k(s)$ is the normalized activity function for kernel k in state s

$$\Phi_N^k(s) = \Phi^k(s) / \sum_{k=1}^{N} \Phi^k(s). \tag{12}$$

The rule in Eq. 11 is called averaging function approximation rule and is considered to perform more stably (Reynolds, 2002) in function approximation schemes as compared to standard methods (Sutton and Barto, 1998). As reward r we used the amount of liquid correctly poured into the target container.

2.4.2 Shape learning methods

For shape learning as a primary means we have used the recently developed PI^2 method (Theodorou et al., 2010), but also made a comparison to the natural actor critic (Peters and Schaal, 2008), which has been very influential in the last years. We are giving a detailed description of those algorithms in the Appendix and keep it brief in the main text. Note, both algorithms are quite complex, thus readers are in addition referred to the original papers of Theodorou et al. (2010) and Peters and Schaal (2008).

Policy Improvement with Path Integrals PI^2: Even though the PI^2 method is derived from the principles of stochastic optimal control, it results in a gradient-type update rule of the control parameters. The method is considering "cost" instead of "reward" and allows adding

noise (to emulate exploration) directly onto the weights of the DMP. The procedure of learning is organized in epochs: first for some trials exploration noise is added and the success of each of those noisy trials is evaluated. Afterwards, weights are updated. The update is organized in such a way that the weights move towards those values where smaller costs were evoked in the previously performed trials. Noise was generated using a Gaussian distribution. According to Theodorou et al. (2010), noise was only placed on the weight of the leading kernel and was kept fixed across the whole extension of the kernel. In addition to the recommendations of the authors, we were varying the learning rate parameter γ_P (see Appendix). Although the authors state that their method has only a single adjustable parameter (noise variance σ_P), our experiments on the pouring model with this algorithm have shown that e.g. ten-fold increase in noise level gives substantially different learning results as compared to a ten-fold increase in the learning rate (see results, below).

In the experiments with the pouring model for the immediate cost function, which we will call $q(t)$, we were using the amount of spilled liquid per time-step, where the amount of the spilled liquid was multiplied by a factor of ten in those time steps, where nothing was poured successfully as yet in the current trial:

$$q(t) = \begin{cases} 10a_{spill}(t), & if \ \sum_{j=0}^{t} a_{target}(j) = 0 \\ a_{spill}(t), & if \ \sum_{j=0}^{t} a_{target}(j) > 0, \end{cases} \tag{13}$$

where $a_{spill}(t)$ is an amount of spilled liquid in time step t and $a_{target}(t)$ is the amount of correctly targeted liquid in time step t. With the factor of 10 we were attempting to emphasize the punishment for the mistargeted liquid in the beginning of pouring which was more difficult to learn. As the terminal cost $q_{terminal}$ we were using the final amount of liquid remaining in the upper container. In the real robot experiments, in order to simplify the setup, we used only the terminal cost term:

$$q_{terminal} = \sum_{t=0}^{T-1} a_{spill}(t), \tag{14}$$

where T is the length of the trajectory, while immediate costs were considered zero.

Natural Actor-Critic (NAC): This is a policy gradient method where the gradient is transformed using the Fisher information matrix and then a so called "natural gradient" is obtained, which is better targeted to the optimum of the reward surface over parameter space as compared to the regular gradient. We have implemented the time-variant baseline version of the

methods from Peters and Schaal (2008), which takes into account immediate rewards. As immediate rewards we used the correctly poured amount of liquid in each discrete time step of the performed trajectory.

One of the main claims of the authors of the NAC method is about of the utility of the method to consider exploration noise σ_N *together* with other parameters in the learning procedure, e.g. weights of the kernels of a DMP in our case (Peters and Schaal, 2008). However, when using the NAC we did not join the exploration noise parameter into the same learning procedure with the shape parameters. To our observation the exploration noise parameter had dramatically different properties, as compared to trajectory shape parameters and was slowing down the learning procedure (one could not use high enough learning rates for the gradient procedure when including into the procedure the exploration noise). Instead, we were annealing the exploration noise independently.

Also, we did not manage to stabilize the NAC algorithm for our task without introducing probing for appropriate learning rate after the natural gradient was determined. Consequently, we added a learning rate probing block in our NAC implementation (see Appendix). In summary, the procedure is as follows: we were probing trajectories with noise for seven trials. Afterwards, the natural gradient was calculated. With the obtained gradient, in addition, we were probing for ten more trials without noise, but with variable learning rate. Afterwards, new weights were obtained using the learning rate providing the best performance. We were considering the trial with the best learning rate as the test trial. Thus the NAC epoch consisted of 17 trials. We were performing six such epochs.

2.4.3 Combination of goal and shape learning

We have combined value function approximation for goal parameter learning and policy gradient methods for weight learning into a single procedure. A block-diagram showing the principles of interaction of the two learning methods is presented in Fig. 3. For implementing the PI^2 (and NAC) methods, several epochs of experiments need to be performed. Let us say K trials compose one epoch. In each of those K trials different noise is added onto the weights and this results in differently shaped DMPs. For each of those K trials the cost is calculated. Both, noise profile and cost function, are memorized for each trial. After the epoch is finished, the weights are updated according to the collected noise-reward statistics. As we were combining

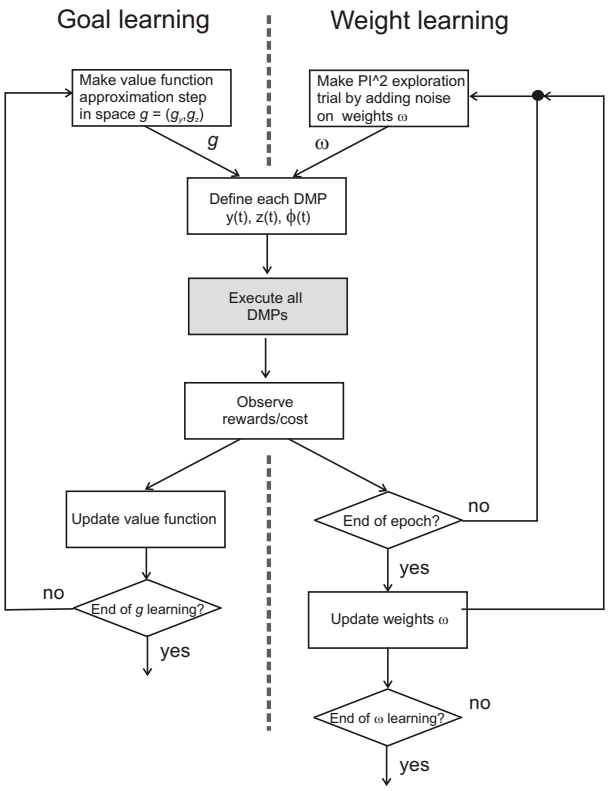

Figure 3: Block-diagram of interaction between goal and weight learning.

two reinforcement learning procedures, we were using the mentioned K trials inside one epoch not only to vary the weights, but also to vary the goal parameters. Goal parameters were varied according to the exploration/exploitation requirements of the reinforcement learning with our function approximation procedure. Updates of the function $V(t)$ were made after each trial, according to Eq. 11. Consequently, the two learning processes were acting together: weights were varied and shape learning was performed, but the same trials were at the same time used out to vary and learn the goal.

3 Results

3.1 Experiments with the pouring model

Central goal of this part of the study is to tune parameters preparing for the real robot task and to compare the PI^2 and NAC methods employed for the shape parameter learning. To this end we first carried out experiments with the pouring model, where many repetitions of an experiment could be executed and statistically evaluated. In Fig. 4 A an example learning

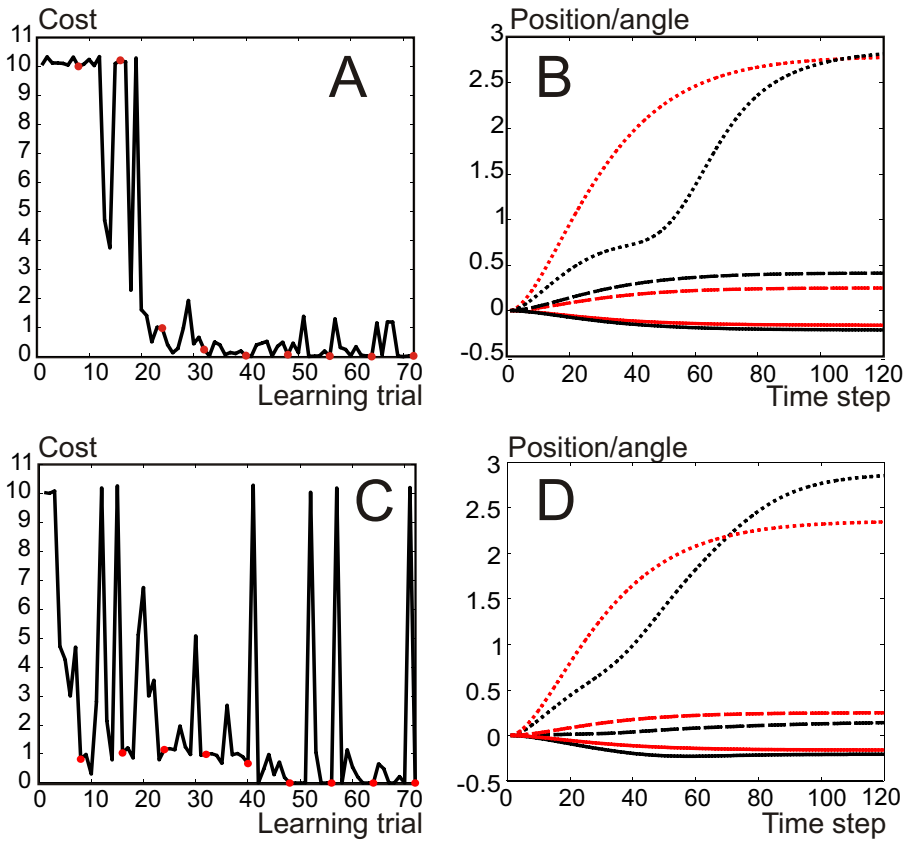

Figure 4: A) Cost decrease with the number of learning trials for shape and goal learning using the pouring model. Solid line - overall cost decrease, red dots - noise-free trials. Parameters: for noise generation $\sigma_P = 30$, learning rate $\gamma_P = 10$. B) trajectories generated by DMPs at the beginng (red) and end of learning (black) , solid line - $z(t)$, dashed line - $y(t)$, dotted line $\phi(t)$, learning applied for the goal parameters of $y(t)$ and $z(t)$ and shape parameters of the $\phi(t)$. C,D) analogous to A,B), but learning applied both for goal and weight learning of all three DMPs (i.e. three goal values and three sets of weights are learned)

curve is shown obtained using the combination of goal and shape learning, where the shape is learned using the PI^2 algorithm. We plot PI^2-cost (defined as described above in eq. 13) against trial number. The figure shows how the cost varies and becomes smaller with learning. The black curve shows the overall learning process. Each learning epoch consists of eight trials. The first seven represent exploration and the weights are disturbed by the exploration noise in these trials. After these seven trials the weight update is calculated. The eighth trial in an epoch is performed noise-free to measure the current system performance and is represented by a red dot in the plot. Nine such epochs are performed. Goal and weight learning is performed for the first six epochs ($6 \times 8 = 48$ trials), later for the final three epochs only weight learning is performed as the goal has by then already become stable. In Fig. 4 B the learned trajectories are shown, where the lowest trajectory is for the side displacement $y(t)$, the middle trajectory

is for the height $z(t)$, and the highest trajectory is for the tilt angle $\phi(t)$. For the $y(t)$ and $z(t)$ only the goal parameter was learned. For the tilting trajectory $\phi(t)$ shape was learned, and the goal parameter was kept fixed. One can see that the tilt trajectory obtains a steep slope in the middle of the movement which then crosses the margin of $\pi/2$ after which the actual running-out of the liquid starts in the employed pouring model. The increase in slope has to synchronize with the wrist movement to a position from which it is possible to successfully pour the liquid into the lower container.

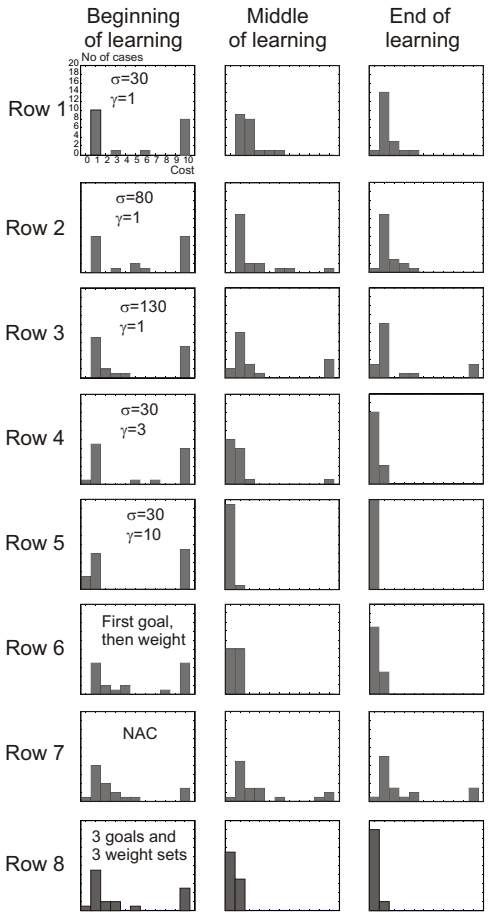

Figure 5: Cost value histograms showing our parameter investigation. First column - result after three epochs , second column - after six epochs and third column - after nine epochs (at the end of learning). First three rows show effects of exploration noise increase in combined goal and shape learning ($\sigma_P = 30, 80, 130$), rows 4-5 - effects of learning rate increase in combined goal and shape learning ($\gamma_P = 3, 10$, for $\gamma_P = 1$ see row 1), row 6 - first goal (4 epochs), then weight (the rest 5 epochs) learning, row 7 - NAC method for combined goal and shape learning, row 8 - redundant learning experiment where goal and shape parameters are learned for all $y(t)$, $z(t)$ and $\phi(t)$, for parameters see text.

We have performed experiments to determine the optimal learning parameters for the PI^2 method. Parameters for value function approximation employed for goal learning were taken from our previous studies (Nemec et al., 2009) (for the numerical values see Table 1). We have done a series of twenty experiments to discover the most appropriate noise level for the PI^2 learning. The results are shown using histograms (Fig. 5). In the first, second and third columns of the figure the cumulative cost along the trajectory $Q = \sum_t q(t)$ attained after three, six and nine learning epochs (first third, second third, and the end of learning) are provided. As the initial amount of liquid in the model bottle was normalized to one, the maximum cumulative cost that could be attained according to Eq. 13 was ten. Values slightly above ten

in Fig. 4 arise because of discretization effects. In Fig. 5, lower costs earlier in learning show better performance. In the first three rows we provide histograms for three different values of the noise variance $\sigma_P = 30$, 80 and 130. One can see that there are no essential differences between learning success in those three cases. Even though the performance grows a small fraction with bigger σ_P, with the highest noise ($\sigma_P = 130$) instabilities in learning are starting to emerge: the right histogram in the third row shows several experiments with high cost, thus, where the desired learning outcome was not achieved even at the end of learning. Trials with $\sigma_P = 10$ (not shown in the histograms), showed very stable but far from optimal performance. Consequently, for further experiments we were choosing $\sigma_P = 30$ (or $\sigma_P = 40$ for the real robot experiments). In the robot experiments the exploration noise should be big enough as to overcome the background (unobservable) noise.

As good learning results (zero spill) could not be obtained with changing the noise parameter alone, we increased the learning rate. In Fig. 5, rows 4 and 5 we show histograms, where the noise level is kept constant, but learning rate is increased ($\gamma_P = 3$ and 10). One can observe, that learning performance significantly increases with increasing the learning rate. We have done experiments with learning rate $\gamma_P = 15$, which, however, started to lead to instabilities.

We have also performed an experiment with first goal, then weight learning (row 6 in the histogram plot). In this case for the first four epochs only the goal was learned, and weight learning was introduced only after goal learning was stopped, for epochs five to nine. In this case $\sigma_P = 30$ and $\gamma_P = 10$ were used. One can observe, that goal and weight learning performed together (Fig. 5, row 5) produced better results as compared to first goal than weight learning (Fig. 5, row 6).

Finally we performed the same sets of experiments using NAC instead of PI^2. These experiments will not be shown in detail and only the very best result from a large parameter investigation is included in Fig. 5. To achieve this we had varied exploration noise and learning rate. Also we were testing the choice whether to include exploration noise into the scheme of the natural gradient of not. The best results were obtained with $\sigma_N = 10$ and a learning rate γ_N in the interval $[0.1,\ 1]$, where the actual learning rate was determined within this interval by the probing procedure described in the Methods section. We did not include exploration noise variance into the natural gradient evaluation procedure. This is due to the fact that, when including the variance, the learning rate had to be significantly reduced to stabilize the

NAC-procedure and appropriate pouring could be no longer learned in a reasonable number of trials. The results for the NAC method are shown in Fig. 5, row 7. As the NAC epochs were longer because we had to include additional trials for learning rate probing, in the histograms we show the results after 2, 4, and 6 learning epochs. Even though NAC shows a comparatively good performance in the beginning of learning, we did not manage to gain the same stable convergence towards very good pouring in the final epochs as compared to the PI^2 method.

Consequently, for the real robot experiments we have chosen PI^2 for shape learning. Goal and shape learning were combined, the noise level was in the range of $\sigma_P = 30$ to 40, and a learning rate of $\gamma_P = 3$ to 5 was used. We have decreased the learning rate for the real robot experiments, as compared to the optimum on the model, to be on the side of more stable performance, rather than attempting to get the best speed of learning.

3.2 Learning in higher dimensions

In order to better reveal the potential of our combined learning algorithm we have performed learning trials where goal and shape parameters were learned for all three DMPs: $y(t)$, $z(t)$ and $\phi(t)$. That is, six entities were learned: three goal parameters and three weight sets. This is a redundant setup, where goals and weights interfere within and - via the world - across DMPs. For each DMP, $L = 15$ shape parameters were used. The noise added on DMP weights for $y(t)$ and $z(t)$ was ten times smaller as compared to the noise added on the weights for $\phi(t)$ ($\sigma_P = 3$ vs. $\sigma_P = 30$), as those signals are approximately ten times smaller as compared to $\phi(t)$.

When combining goal and shape learning methods we were simultaneously updating all three goal parameters and all three sets of weights. The shape parameters of each DMP, on the other hand, were optimized independently of all other DMPs' shape parameters, as suggested by Schaal (personal communication). Note, due to goal and weight update one gets direct interference of the parameters at each individual DMP (Fig. **??**, above gray box). The two learning procedures were, however, also interfering with each other across DMPs (as in all previous cases), due to the fact that all DMPs are excuted at the robot at the same time (Fig. **??**, gray box).

An example of a learning curve, as well as trajectories before an after learning for this bigger (redundant) learning task are provided in Fig. 4 C,D. One can observe that relatively good performance is achieved already after the first epoch, that is, earlier as compared to

the non-redundant setup (see Fig. 4A), but the final cost-drop to zero happens only later, as compared to the non-redundant setup. Also in the redundant case towards the end of learning there are many exploration trials which are out of range of good performance, which, though, do not affect the noise-free trials, and consequently the overall learning result remains untouched. If one looks at the trajectories of the redundant setup before and after learning (Fig. 4 D), one can observe that the side displacement of the wrist which is crucial for correct pouring (solid line) at the end of learning is shaped by the weights in such a way that it reaches the correct pouring position earlier (trajectory is curved downwards), as compared to the non-redundant case (see Fig. 4 B). Even though we show only one example in the figure, these observations are quite general and hold when analyzing many examples from those two setups.

The results on performance statistics for the higher dimensional (redundant) learning task are summarized in the last row in Fig. 5 (row 8). One can see that learning results are similar to the best non-redundant setup in row 5. In the redundant setup only in two out of 20 trials learning was a fraction slower, which is remarkable as dimensionality is now much higher. The possible disruptive effect of the interference between DMP parameter is apparently very small. As there are many appropriate combinations of goals and shapes, we observe different DMPs as in the non-redundant case (Fig. 4), but in both cases the values of shape parameters were small at the end of the trajectory. The shape of the $z(t)$ trajectory (height of the wrist) was changed minimally through learning, while the shape of $y(t)$ trajectory (side displacement) was showing bigger weights in the beginning of the trajectory, which brought the mentioned quicker approach of the wrist to the correct pouring position. In general one can conclude that also the redundant setup shows quick and stable learning. Learning was only minimally delayed because of learning six quantities (three goals, three sets of weights)instead of three quantities (two goals and one set of weights).

3.3 Learning shape and goal parameters in a real robot experiment

For real robot experiments it is important that any learning algorithm will converge only after a few trials. Furthermore, learning should be robust against fine-tuning of parameters, against measurement errors, as well as against external, uncontrollable noise. To show this, in our experiments we purposefully chose a setup where the pouring success could only be measured with quite wide accuracy margins of $\approx \pm 10$ *grams* corresponding to only an accuracy of about

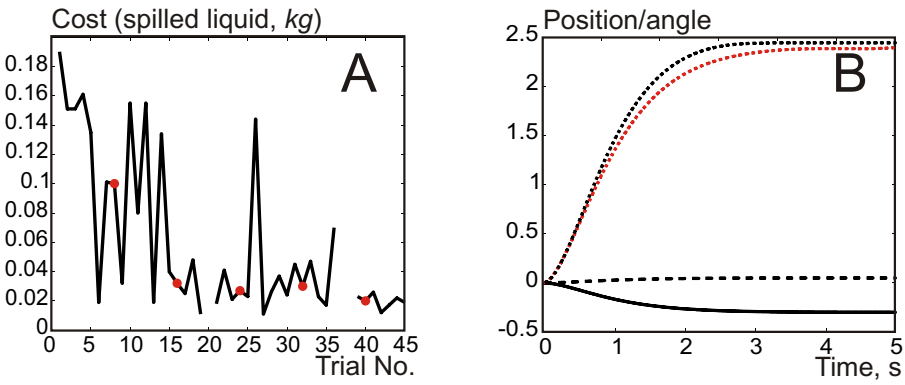

Figure 6: A) Cost decrease with the number of learning trials for the shape only learning. Solid line - overall cost decrease, red dots - noise-free trials; missing values stand for incorrect reading of the scales; parameters: $\sigma_P = 30$, $\gamma_P = 5$. *B) Trajectories generated by DMPs at the end of learning,* $z(t)$ *as solid line,* $y(t)$ *as dashed line,* $\phi(t)$ *as dotted line, red - before learning, black - after learning, learning applied only for the shape trajectory* $\phi(t)$.

$\pm 5\%$ of the total liquid. Furthermore, the scales used to measure pouring success were set up to produce frequent false readings (in one out of 20 cases read-out was zero as if pouring had been totally unsuccessful). The following experiments show an amazing degree of robustness of the combined PI^2-shape with our goal learning algorithm against such large contingencies. Furthermore, convergence is fast and the analysis above has already demonstrated that parameters are not terribly critical, either.

3.3.1 Learning shape only

Let us assume that the robot "knows" a good enough final position of the wrist (goal parameter of the DMPs for $y(t)$ and $z(t)$) and that only the weights of the tilting DMP $\phi(t)$ are being learned. The learning curve obtained in this case with the Pa10 robot arm is shown in Fig. 6 A plotting cost of a pouring attempt against trial number. The cost is measured in grams of liquid spilled. The black curve shows the overall learning process including exploration trials and the red dots show the noise-free trials performed to evaluate the learning process. The missing values stand for the trials where the scalew was giving incorrect readings. The final trajectories of the three curves ($\phi(t)$ as learned and $y(t)$, $z(t)$ fixed) are shown in Fig. 6 B. The solid line denotes $y(t)$, the dashed line denotes $z(t)$, and the dotted line denotes the tilting trajectory $\phi(t)$. In this experiment the goal position was set on purpose behind the glass and the learning process had, thus, to tune the tilting trajectory for an early tilt. This way the DMP weights obtained positive values with an emphasis on the first half of the trajectory and the learned

pouring trajectory was steeper as compared to the original trajectory that was obtained with zero weights of the DMP (shown in red dots in the figure).

One can observe that in the learning curve (Fig. 6 A) the behavior close to optimum (almost all the poured liquid is correctly poured into the lower container) is attained in two learning epochs (first 16 trials). Later the result is sometimes disturbed by noise (solid line), but the noise-free trials always show good pouring results after the first two epochs. The difference of the final cost from zero is in the range of precision of the measuring system. The unobservable part of the noise of the experiment is composed mainly of the imprecision in the initial amount of liquid poured into the bottle, as well as the not ideal running-off from the scales of the spilled amount.

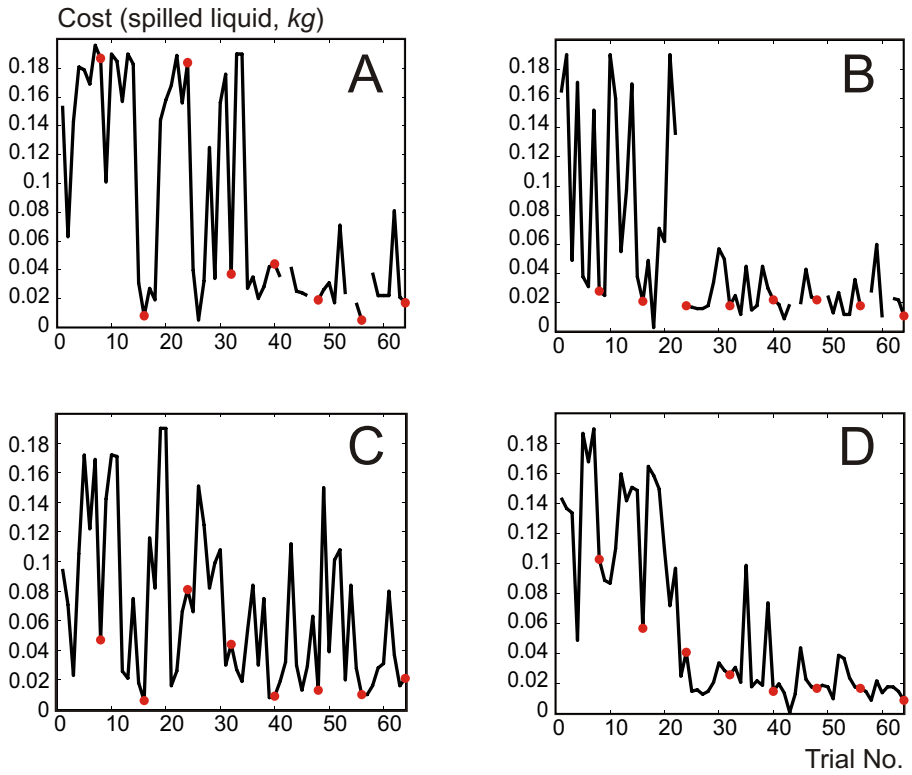

Figure 7: Cost decrease with the number of learning trials for combined goal and shape learning. Solid line - overall cost decrease, red dots - noise-free trials. Missing values show incorrect scales measurements. Parameters: A,B) $\sigma_P = 30$, $\gamma_P = 5$, *both constant over the length of the experiment, C)* $\sigma_P = 40$, $\gamma_P = 3$, *where the learning rate was multiplied by a factor of* 0.85 *after each epoch, the new noise is added and trajectory is recalculated if the last third of the trajectory has an average less than* $\frac{5}{8}\pi$, *D) like (C) but noise level was multiplied by a factor of* 0.85 *after each epoch.*

3.3.2 Combined learning of goal and shape in a real robot experiment

If one drops the assumption that the goal position for the DMPs is known, then both, goal and shape, need to be learned. The results of four trials with the Pa10 robot arm are shown in Fig. 7. In panel A and B one can see two trials with $\sigma_P = 30$ and $\gamma_P = 5$. The exploration noise provided some jumps in cost also at the end of learning, but no big jumps occurred for the noise-free trials (red dots), which is indicating that convergence was essentially reached. The variability of the red dots is within the precision limit for these experiments, which was about ± 10 g. With interrupts in the signal we denote the trials where the scalew provided bad readings. This is interesting, because, even with such relatively frequent false readings the proposed combined goal and shape learning algorithm was behaving in a stable way and was able to bring learning to convergence.

In Fig. 7 C the results with gradual learning rate reduction ("annealing") are shown (for details on the annealing procedure see figure legend).

For this experiment the trajectories with too small tilt were rejected, as with small tilt it is impossible to pour any water out of the bottle. In our setup the liquid could not be poured out of the bottle using tilt angles lower than approx. $\frac{5}{8}\pi$. This was judged only for the end of the tilting, i.e. for the last 1.67 s of the trajectory. On failure to comply, a new trajectory was generated. One can see that the noise-free trajectories start giving persistently low cost after 40 trials. In this experiment one can observe that for the first time good results were achieved after only two learning epochs (noise-free trial no 16), but that the cost has grown again in the next epoch. This could have happened because of a too high learning rate or because of the interactions between goal and shape learning procedures. This worse learning result was improved again after two more learning epochs and the result remained stable afterwards.

In Fig. 7 D in addition to the learning rate reduction also the exploration noise was reduced with the progression of learning. Precise pouring was achieved after $3 - 4$ epochs. One has, however, to be careful with reducing exploration noise in real-world experiments as there exist multiple sources of unobservable noise. If exploration noise is smaller than those sources a good result might be observed due to the uncontrollable noise and not due to the exploration noise, leading to incorrect learning.

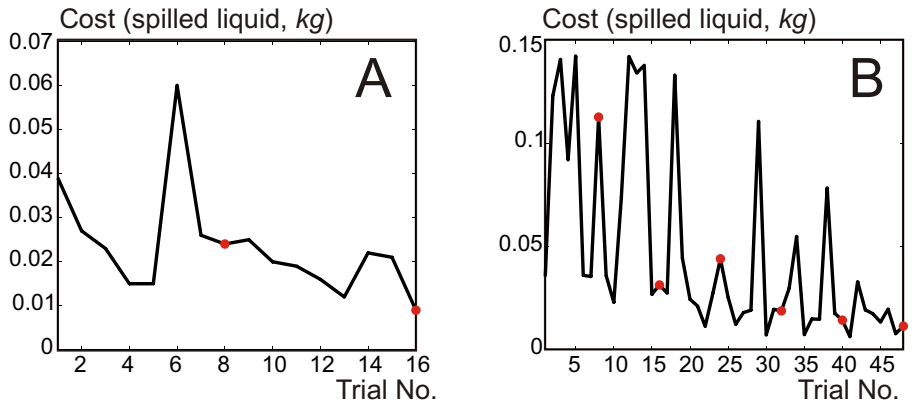

Figure 8: Cost decrease with the number of learning trials for relearning to pour from a different bottle (A) and learning using weights obtained from human demonstration (B). Solid line show overall cost decrease, red dots show noise-free trails. Parameters: $\sigma_P = 40$, $\gamma_P = 3$, *learning rate was multiplied by a factor of* 0.85 *after each epoch.*

3.3.3 Relearning after change of bottle

In Fig. 8 A the results for the relearning experiment using a different bottle are presented. First the pouring movement was learned with the regular bottle, then a bottle with a wider opening was given to the robot and the pouring movement was relearned. Good pouring results were obtained already after the first re-learning epoch. This experiment shows that after preliminary learning of the pouring movement much less trials are required after changing the tool, as compared to learning from scratch, and suggests that reinforcement learning may be successfully used for manipulation re-calibration.

3.3.4 Learning using data from human demonstration

Fig. 8 B shows results for the experiment where the movement was initialized with weights obtained from human demonstration. Weights were provided for all three DMPs and were obtained using regression techniques (Schaal et al., 2004). We assumed that the goal position cannot be extracted with good enough precision from such a demonstration. Instead, we assigned an arbitrary initial goal position close to target and just used the same goal position as in the previous experiments where learning was starting from zero weights. Both, goals and shapes, were allowed to change by learning. Relatively good pouring in this experiment was obtained already after only two learning epochs. Note, the trajectories remained human-like throughout learning. The initial weights were not much changed at the end of learning as compared to the ones obtained from the human demonstration, but the pouring success was substantially

improved. This way the demonstrated pouring movement was successfully adapted to the new setup.

4 Discussion

Two different reinforcement learning techniques, value function approximation-based learning and policy gradient learning, were joined together into one procedure to perform the task of learning to pour. Value function approximation was used for DMP goal parameter learning, while the policy gradient approach was used for the DMP shape parameter learning. The recently developed policy gradient algorithm PI^2 method was employed and compared to the Natural Actor Critic. Learning to pour liquid into a glass was achieved in 16 to 40 trials, which is a good score for a robotic application including complex learning. Relearning (of a different bottle) or re-calibration after human demonstration took about 16 and 32 trials, respectively. Furthermore, experiments were on purpose "dirty", including bad measurements and much limiting the resolution of the cost-determining function. In spite of this we observed an excellent robustness of PI^2 on its own as well as in combination with value function approximation for goal learning.

4.1 Motivating our Approach

Combination of DMP goal and shape learning for DMPs is a novel approach, where other authors have also pointed out that such a combination should be investigated (Peters et al., 2009). We used two different methods for the learning of the two components (shape and goal), instead of doing the whole learning with only the policy gradient procedure, because of several reasons. First, our previous experiments with the natural actor-critic (Peters and Schaal, 2008) have shown that this method was slow when trying an intrinsic combination of DMP goal and shape learning for pouring. Possibly the "reward surface" was not favorable for such an inner-combination. For the PI^2 algorithm, on the other hand one observes that its algorithmic structure as described in Theodorou et al. (2010) is not readily adaptable to goal parameter learning because PI^2 needs time-variable kernels, whereas for the adaptation of a goal the kernels have to be constant. These are the reasons that support the idea of trying a combination of different methods instead.

4.2 Stability and Robustness

The general problem when combining learning methods is that destructive interference could occur, where both methods - as they are essentially independent - work against each other. This has not been observed with the here pursued mix of goal and weight learning even though the same learning trials were used to provide information for the two learning procedures at the same time.

We found that the PI^2 is a very efficient procedure and is able to extract information from a very small number of successful trials. Also, it mainly ignores the information of the unsuccessful trials. This adds to the success of the combined algorithm. If "good" weights were accidentally combined with bad goals, those experiences could not disrupt learning due to the design of the PI^2 method. On the other hand, the value function approximation procedure employed in this study proved to be stable in spite of occasional combination of potentially good goal position with improper weights.

In addition, we performed simulation experiments with a redundant setup, where goals and weights were interfering with each other on a single DMP as well as across DMP during execution. When the goal is set a-priori, and only the weights are changed through learning, as in previous studies (Peters and Schaal, 2008; Kober and Peters, 2009), an intrinsic interference does not happen. When both quantities are allowed to change in the process of learning, specific requirements for the learning algorithm emerge, where it is important that goal and weights don't start counter-acting each other (big positive goals counteract the big negative weights, or vice versa). This was the case which we were obtaining when learning both goals and weights into a single NAC procedure, but not with the combined value function approximation and PI^2 that proved favorable in this study.

Learning in the redundant setup has also proven that the here introduced combination of algorithms can deal with tasks of relatively high dimensionality, as in this case six entities were learned in parallel, three goals and three sets of 15 weights each.

One can argue that using value function approximation method for goal learning does not generalize to much higher dimensions. Our prior work has demonstrated that good convergence is obtained for up to six and possibly even more dimensions as long as the total learning space remains restricted (Tamosiunaite et al., 2009). This is often the case for tasks where general targeting can be learned by supervised methods (e.g. learning from demonstration) or can be

achieved by (visual) servoing (Tamosiunaite et al., 2009) and only the final tuning requires RL methods. Many such tasks exist. Thus, our value function approximation method can be advantageously used for such tasks.

These arguments explain why the combination of goal and shape learning based on our value function approximation algorithm together with the PI^2 is stable supporting it potential usefulness also for other tasks.

Still the question remains to what degree such a combination would be robust against parameter changes. We have checked the methods in a wide parameter range, changing exploration noise and learning rate for the policy gradient methods. Even though the convergence rate proved to be different with different parameters, convergence of the combined learning algorithm persisted.

We have also specifically checked the stability of the proposed algorithm in respect to occasional incorrect feedback. The scales we were using was providing frequent bad readings. The algorithm managed to ignore those and they did not influence convergence. This proved the capability of the algorithm to tolerate incorrect feedback information, which is quite a generic case in experimental robotics. Also one would expect occasional incorrect measurements or incorrect interpretations of the environment in the operation of future home robots. Consequently, when developing adaptive (learning) algorithms for those robots one needs to make them robust against such errors.

4.3 Comparison of Methods and Alternative Approaches

While we were performing this study, new methods have emerged for policy gradient evaluation. One of those is the PoWER method (Kober and Peters, 2009). This method is based on the idea of expectation maximization, but its algorithmic formulation is very similar to the PI^2 method. Even though different methods for policy gradient learning are derived from different principles (e.g. stochastic optimal control for PI^2 and expectation maximization for PoWER), to our experience, the main difference that really matters between methods like PI^2 and PoWER, as compared to NAC, is how exploration noise is being introduced. The efficient way is to put noise on weights (like in PI^2 and PoWER), but not on acceleration like suggested for NAC. When putting noise directly on the weights, the correspondence between weight values and rewards is more straightforward and this brings faster convergence of the learning methods.

To our observation, modifications of the details of the algorithms do not lead to big changes in the final result[1]. Our observations are supported by those of Kolter and Ng (2009), who find that quite a coarse algorithm can perform very well in policy gradient learning. They simplified the policy gradient search by replacing the Jacobian terms with a signed derivative approximation ($+1, -1$ *or* 0) and obtain good results in complicated system control (robot-dog climbing stairs). Also more traditional gradient approaches have proven to produce good results in complex robot control tasks (Morimoto and Atkeson, 2009; Endo et al., 2008).

Potentially, one can also use policy gradient approaches for the goal learning, too, as these methods tolerate multiple dimensions better as compared to the value function approximation techniques. This question has to be investigated in the future but certain drawbacks are obvious as discussed above and neither NAC nor PI^2 seem well suited for such a combination. Alternatively, here we show a successful combination of goal and weight learning using different frameworks. This combination also performed well for calibration after having initially learned a movement from demonstration. This is important, because learning from demonstration remains possibly the most efficient learning framework in many complex robotic applications like service robotics (Calinon et al., 2007; Hersch et al., 2008; Pastor et al., 2009). We have done such an experiment in this study showing that our learning was able to arrive at the required calibration of the goal position in a small number of trials. Similarly only a small episode of reinforcement learning needs to be used for recalibration when changing the tool (e.g. changing the bottle in our experiment). Thus, we believe that the here demonstrated combination could be usefully applied also to other tasks of robot manipulation.

Acknowledgments: The work described in this paper was conducted within the EU Cognitive Systems project PACO-PLUS (FP6-2004-IST-4-027657) funded by the European Commission and the BCCN, Göttingen funded by the German Ministry of Science, grant BCCN Göttingen, W3.

[1]This notions is much supported by our own experimental observations. Namely, for this study we performed a big set of additional experiments (data not shown) where we made different modifications to the algorithms. For example, we did some experiments with PI^2, by introducing a discount factor, as well as using only the best trial of the epoch for weight update in step 4, see the section "Parameter update rule for PI^2" in the Appendix, etc., etc. Most of these modification did not significantly reduce learning success.

Appendix

This appendix will describe the implementation of PI^2 and NAC using consistent notation, allowing readers to implement it without having to heavily consult the original papers Theodorou et al. (2010) and Peters and Schaal (2008). Algorithmic "peculiarities" and specific alteration, which we found useful are being commented.

Definitions

1. DMPs are used as given in Eq. 1 (see main text) and integrated using the Euler method, where time steps are numbered $t = 0, 1, 2..., T-1$.

2. Kernels in the non-linear part of DMP are indexed with $l = 1, 2, ..., L$, where L is the overall number of Gaussian kernels in the DMP.

3. One attempt to pour we call a trial.

4. Learning epochs are sets of trials, consisting of the exploration stage (K trials), the learning rate probing stage (used only in NAC with D trials) and the testing stage (used only for PI^2 one trial). One learning epoch includes $K+1$ trials for the PI^2 and $K+D$ trials for the NAC. In the exploration stage of an epoch trials are indexed $k = 1, 2, ..., K$. In the learning rate probing stage trials are indexed $d = 1, 2, ..., D$.

PI^2

Here we will describe the algorithmic procedure for implementing PI^2, which consists of the learning procedure as such and the pseudo-code of the PI^2 parameter update rule.

Learning procedure for PI^2

1. Initialize algorithm with kernel weights ω_l, $l = 1, 2, ..., L$, the weight vector we will notate by $\boldsymbol{\omega}$.

2. Repeat for several learning epochs (or until convergence of cost measure Q):

   (a) Repeat for trials $k = 1, 2, ..., K$ in the exploration stage of an epoch:

      i. Generate noise values $\epsilon_{l,k,t}$ from $\mathcal{N}(0, \sigma_P)$, and consider those as $K \times T$ vectors $\boldsymbol{\epsilon}_{k,t}$ of length L, save in memory.

ii. Add noise to weight vector $\boldsymbol{\omega}$ and obtain weight vectors $\boldsymbol{\omega}_{k,t}$ (or add noise only on the leading kernel[2]).

iii. Generate trajectory $y_k(t)$ with noisy weights. Among the components required to obtain $y_k(t)$ are kernel activation vectors $\boldsymbol{\psi}_{k,t}$, save those vectors in memory[3].

iv. Execute trajectory $y_k(t)$ on a robot or simulator.

v. Measure/obtain cost function $q_k(t)$ and terminal cost term q_{term}, save in memory.

(b) Modify weights $\boldsymbol{\omega}$ according to PI^2, using memorized noise vectors $\boldsymbol{\epsilon}_{k,t}$, kernel activation vectors $\boldsymbol{\psi}_{k,t}$, and cost functions $q_k(t)$ (see Pseudo-Code in the subsection "Parameter update rule for the PI^2").

(c) Execute one trial in the testing stage of the epoch:.

i. Generate trajectory $y(t)$ with noise-free weights $\boldsymbol{\omega}$.

ii. Execute trajectory $y(t)$.

iii. Measure/obtain cost function $q(t)$ and terminal cost q_{term}.

iv. Evaluate overall cost measure $Q = q_{term} + \sum_{t=0}^{T-1} q(t)$.

Parameter update rule for the PI^2

1. For each k, t compute matrix $\boldsymbol{M}_{k,t} = \frac{\boldsymbol{R}^{-1}\boldsymbol{\psi}_{k,t}\boldsymbol{\psi}_{k,t}^T}{\boldsymbol{\psi}_{k,t}^T\boldsymbol{R}^{-1}\boldsymbol{\psi}_{k,t}}$, where $\boldsymbol{R}$ is control penalty (regularization) matrix.

2. For each k, t compute cost including regularization term: $S_{k,t} = q_{term} + \sum_{j=t}^{T-1} q_{k,j} + 0.5 \sum_{j=t+1}^{T-1} (\boldsymbol{\omega} + \boldsymbol{M}_{k,t}\boldsymbol{\epsilon}_{k,t})^T \boldsymbol{R}(\boldsymbol{\omega} + \boldsymbol{M}_{k,t}\boldsymbol{\epsilon}_{k,t})$.

3. For each k, t compute relative goodness for each trajectory point (as compared to the same point on the other trajectories in an epoch) $P_{k,t} = \frac{e^{-\frac{1}{\lambda}S_{k,t}}}{\sum_{k=1}^{K} e^{-\frac{1}{\lambda}S_{k,t}}}$, where Theodorou et al. (2010) suggest to approximate $e^{-\frac{1}{\lambda}S_{k,t}} \approx exp(-c\frac{S_{k,t}-\min_k S_{k,t}}{\max_k S_{k,t}-\min_k S_{k,t}})$ with $c = 10$.

4. For each time point compute an update term $\delta\boldsymbol{\omega}_t = \sum_{k=1}^{K} P_{k,t}\boldsymbol{M}_{k,t}\boldsymbol{\epsilon}_{k,t}$. See note[4].

[2]While the core algorithm would allow adding noise to all kernels, Theodorou et al. (2010) suggested to only add noise to the kernel with the biggest activation level (leading kernel), as well as to add the same noise value over the extension where that kernel is remaining the leading kernel. This way one would be dealing with noise K vectors $\boldsymbol{\epsilon}_k$, as only one noise vector per trial is required. Our own tests in doing this or adding noise to all kernels showed indeed delayed convergence in the latter case. Hence we like Theodorou et al. (2010) used noise on the leading kernel only.

[3]For the sake of more intuitive notation we will use notation $f(t)$ (e.g. $y_k(t)$), where we talk about the entire function (trajectory) in time, and f_t where we talk about the specific value of the function in time.

[4]Here the method multiplies "goodness" by noise values used to obtain that score and introduces correlations in updates for kernels activated in a correlated way by multiplication by matrix $\boldsymbol{M}$.

5. Using cumulative sums, summarize updates along the trajectory into one weight update value $\delta\boldsymbol{\omega} = \frac{\sum_{t=0}^{T-1}(T-t)\delta\boldsymbol{\omega}_t}{\sum_{t=0}^{T-1}(T-t)}$.

6. Update weights $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \gamma_P \delta\boldsymbol{\omega}$, See note[5].

NAC

Here we will describe the algorithmic procedure for implementing NAC, of the learning procedure as such and the pseudo-code of the NAC parameter update rule.

Learning procedure for *NAC*

1. Analytically derive expression for log-policy derivatives $\boldsymbol{\delta} = \nabla_{\omega,\sigma_N} \log \Pi(\dot{v}(t,\omega), \sigma_N)$ according to parameters ω_l, $l = 1, 2, ..., L$ and σ_N (see note[6]), where policy $\Pi(\dot{v}(t,\omega), \sigma_N)$ is defined by distribution of acceleration values $\dot{v}(t) \sim \mathcal{N}(\bar{\dot{v}}(t), \sigma_N)$, where the expression for $\bar{\dot{v}}(t)$ is obtained from the right hand side of eq. 1.

2. Initialize algorithm with weights ω_l, $l = 1, 2, ..., L$ and exploration noise value σ_N.

3. Repeat for several learning epochs (or until convergence of return R):

   (a) Repeat for trials $k = 1, 2, ..., K$ in the exploration stage of an epoch:

      i. Generate noise values $\xi_{k,t}$ from $\mathcal{N}(0, \sigma_N)$ and save in memory.

      ii. Use weights $\boldsymbol{\omega}$ as they appear (without noise).

      iii. Add noise $\xi_{k,t}$, to acceleration $\dot{v}$ in all time steps of DMP integration.

      iv. Generate trajectory $y_k(t)$ using noisy acceleration, save in memory kernel activation vectors $\boldsymbol{\psi}_{k,t}$ and trajectory $x_D k(t)$ from DMP equation for $x_D(t)$ (1).

      v. Execute trajectory $y_k(t)$ on a robot or simulator.

      vi. Measure/obtain reward function $r_k(t)$ and save in memory.

   (b) Calculate natural gradients $\boldsymbol{g}_{NG}$ according to NAC, using memorized noise values $\xi_{k,t}$, kernel activation vectors $\boldsymbol{\psi}_{k,t}$, DMP variable x_D trajectories $x_k(t)$ and reward functions $r_k(t)$ (see Pseudo-Code in the subsection "Parameter update rule for the NAC").

[5] In original formulation by the authors strictly $\gamma_P = 1$ is used, but to our experience $\gamma_P > 1$ might work better.

[6] σ_N might be either included in the framework of the NAC learning, or annealed independently.

(c) Decrease exploration noise σ_N.

(d) Repeat for $d = 1...D$ trials in the learning rate probing stage of the epoch[7]:

i. Generate new learning rate value γ_d and save in memory.

ii. Modify weights $\boldsymbol{\omega}$ with the learning rate γ_d and obtain temporary weights $\boldsymbol{\omega}_{temp} = \boldsymbol{\omega} + \gamma_d \, \boldsymbol{g}_{NG}$.

iii. Generate trajectory $y_d(t)$ with the temporary weights $\boldsymbol{\omega}_{temp}$.

iv. Execute trajectory $y_d(t)$.

v. Measure/obtain reward function $r_d(t)$.

vi. Evaluate the return $R_d = \sum_{t=0}^{T-1} r_d(t)$ and save in memory.

(e) Find the optimum of γ_d according to saved R_d values and permanently modify the weights $\boldsymbol{\omega} = \boldsymbol{\omega} + \gamma_{opt} \, \boldsymbol{g}_{NG}$.

(f) Consider the return of the optimum trial from the learning rate probing procedure as the overall return of the current learning epoch $R = \min R_d$.

Parameter update rule for the NAC

We used the episodic NAC version with time-variant baseline from Peters and Schaal (2008).

1. For each l, k, t evaluate log-policy derivatives, let us assume they are kept as $K \times T$ vectors $\boldsymbol{\delta}_{k,t}$ of length $L + 1$, where $L + 1$ denotes the number of adjustable weights L plus one additional component for the variance σ_N.

2. Calculate Fisher matrices for each trial $\boldsymbol{F}_k = \sum_{t=0}^{T-1} (\sum_{j=0}^{t} \boldsymbol{\delta}_{k,j}) \boldsymbol{\delta}_{k,t}^T$.

3. Calculate the average over k trials $\boldsymbol{F} = \frac{1}{K} \sum_{k=1}^{K} F_k$.

4. Calculate gradient for each trial $\boldsymbol{g}_k = \sum_{t=0}^{T-1} (\sum_{j=0}^{t} \boldsymbol{\delta}_{k,j}) \alpha_t r_t$, where α_t is weighting factor introducing discount.

5. Calculate the average over K trials $\boldsymbol{g} = \frac{1}{K} \sum_{k=1}^{K} \boldsymbol{g}_k$.

6. Calculate cumulative log-policy derivative values $\boldsymbol{\eta}_{k,t} = \sum_{j=0}^{t} \boldsymbol{\delta}_{k,t}$.

7. Calculate the average over K trials $\boldsymbol{\eta}_t = \sum_{k=1}^{K} \boldsymbol{\eta}_{k,t}$, let us join the vectors $\boldsymbol{\eta}_t$ into matrix $\boldsymbol{H}$.

[7] This probing for the value of the learning rate was not included in the original procedure (Peters and Schaal, 2008) but we found it necessary in order to stabilize learning in our pouring task

8. Obtain weighted reward vector $\boldsymbol{\vartheta} = [\vartheta_0, \vartheta_1, ..., \vartheta_{T-1}]^T$, where $\boldsymbol{\vartheta}_t = \frac{1}{K} \sum_{k=1}^{K} \alpha_t r_t$

9. Calculate matrix $\boldsymbol{Q} = K^{-1}(\boldsymbol{I} + \boldsymbol{H}^T(K\boldsymbol{F} - \boldsymbol{H}\boldsymbol{H}^T)^{-1}\boldsymbol{H})$

10. Calculate time variable baseline $\boldsymbol{b} = \boldsymbol{Q}(\boldsymbol{\vartheta} - \boldsymbol{H}^T\boldsymbol{F}^{-1}\boldsymbol{g})$

11. Calculate natural gradient $\boldsymbol{g}_{NG} = \boldsymbol{F}^{-1}(\boldsymbol{g} - \boldsymbol{H}\boldsymbol{b})$

References

Baxter, J. and Bartlet, P. (2001). Infinite-horizon policygradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350.

Calinon, S., Guenter, F., and Billard, A. (2007). On learning, representing, and generalizing a task in a humanoid robot. *IEEE Trans. Syst. Man Cybern.*, 32(2):286–298.

Endo, G., Morimoto, J., Matsubara, T., Nakanishi, J., and Cheng, G. (2008). Learning cpg-based biped locomoion with a policy gradient method: Application to a humanoid robot. *The International Journal of Robotics Research*, 27(2):213–228.

Hersch, M., Guenter, F., Calinon, S., and Billard, A. (2008). Dynamical system modulation for robot learning via kinesthetic demonstrations. *IEEE Transactions on Robotics*, 24(6):1463–1467.

Ijspeert, A. J., Nakanishi, J., and Schaal, S. (2002). Movement imitation with nonlinear dynamical systems in humanoid robots. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 1398–1403, Washington, DC.

Kober, J. and Peters, J. (2009). Policy search for motor primitives in robotics. In *Advances in Neural Information Processing System*, volume 22.

Kolter, J. and Ng, A. (2009). Policy search via the signed derivative. In *Robotics: Science and Systems (RSS)*.

Morimoto, J. and Atkeson, C. (2009). Nonparametric representation of an approximated poincar map for learning biped locomotion. *Auton Robot*, 27:131–144.

Nemec, B., Tamosiunaite, M., Wörgötter, F., and Ude, A. (2009). Task adaptation through exploration and action sequencing. In *Proceedings of 9th IEEE-RAS International Conference on Humanoid Robots*, pages 610–616.

Pastor, P., Hoffmann, H., Asfour, T., and Schaal, S. (2009). Learning and generalization of motor skills by learning from demonstration. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 763–768, Kobe, Japan.

Perk, B. and Slotine, J. (2006). Motion primitives for robotic flight control.

Peters, J., Mulling, J., Kober, J., Nguyen-Tuong, D., and Kroemer, O. (2009). Towards motor skill learning for robotics. In *Proceedings of the 14th International Symposium on Robotics Research (ISRR 2009)*, pages 1–14.

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

Reynolds, S. I. (2002). The stability of general discounted reinforcement learning with linear function approximation. In *UK Workshop on Computational Intelligence (UKCI-02)*, pages 139–146.

Schaal, S., Mohajerian, P., and Ijspeert, A. (2007). Dynamics systems vs. optimal control – a unifying view. *Progress in Brain Research*, 165(6):425–445.

Schaal, S., Peters, J., Nakanishi, J., and Ijspeert, A. (2004). Learning movement primitives. In *International symposium on robotics research (isrr2003)*, page 1805. springer.

Sutton, R. and Barto, A. (1998). *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA.

Tamosiunaite, M., Asfour, T., and Wörgötter, F. (2009). Learning to reach by reinforcement learning using a receptive field based function approximation approach with continuous actions. *Biological Cybernetics*, 100(3):249–260.

Theodorou, E., Buchli, J., and Schaal, S. (2010). Reinforcement learning of motor skills in high dimensions: A pah integral approach. In *International Conference of Robotics and Automation*.