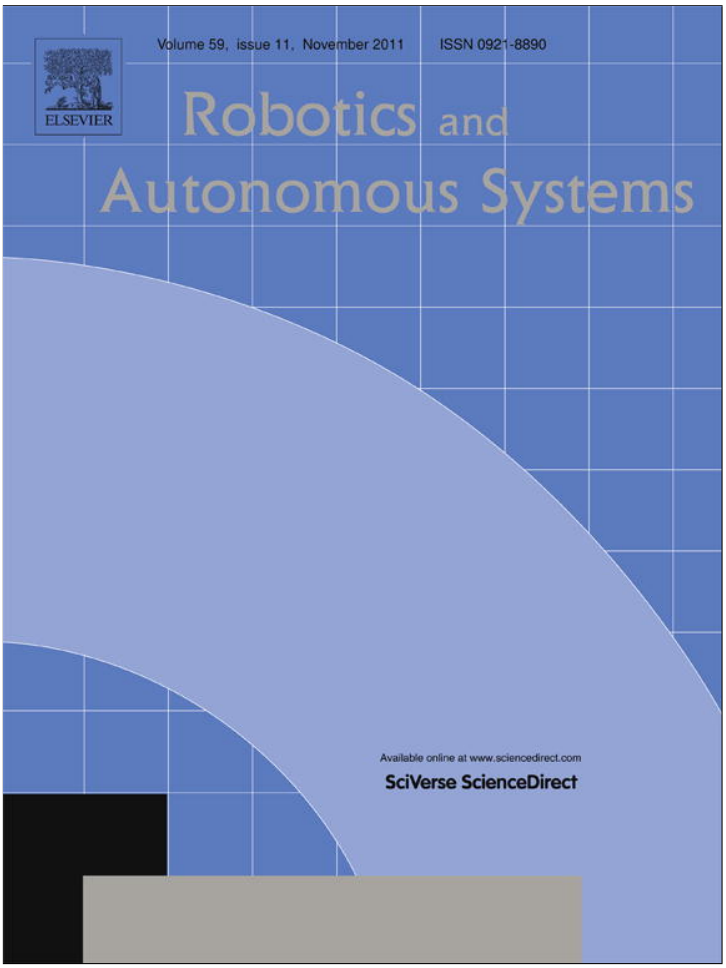

Robotics and Autonomous Systems 59 (2011) 910–922

Contents lists available at SciVerse ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Learning to pour with a robot arm combining goal and shape learning for dynamic movement primitives

Minija Tamosiunaite [a,b,*], Bojan Nemec [c], Aleš Ude [c], Florentin Wörgötter [a]

[a] University Göttingen, Institute for Physics 3 - Biophysics, Bernstein Center for Computational Neuroscience, Friedrich-Hund-Platz 1, 37077 Göttingen, Germany

[b] Vytautas Magnus University, Department of Informatics, Vileikos 8, 44404 Kaunas, Lithuania

[c] Jožef Stefan Institute, Department of Automatics, Biocybernetics, and Robotics, Jamova 39, 1000 Ljubljana, Slovenia

ARTICLE INFO

Article history:
Received 1 July 2010
Received in revised form
30 June 2011
Accepted 4 July 2011
Available online 12 July 2011

Keywords:
Reinforcement learning
PI^2-method
Natural actor critic
Value function approximation
Dynamic movement primitives

ABSTRACT

When describing robot motion with dynamic movement primitives (DMPs), goal (trajectory endpoint), shape and temporal scaling parameters are used. In reinforcement learning with DMPs, usually goals and temporal scaling parameters are predefined and only the weights for shaping a DMP are learned. Many tasks, however, exist where the best goal position is not a priori known, requiring to learn it. Thus, here we specifically address the question of how to simultaneously combine goal and shape parameter learning. This is a difficult problem because learning of both parameters could easily interfere in a destructive way. We apply value function approximation techniques for goal learning and direct policy search methods for shape learning. Specifically, we use "policy improvement with path integrals" and "natural actor critic" for the policy search. We solve a learning-to-pour-liquid task in simulations as well as using a Pa10 robot arm. Results for learning from scratch, learning initialized by human demonstration, as well as for modifying the tool for the learned DMPs are presented. We observe that the combination of goal and shape learning is stable and robust within large parameter regimes. Learning converges quickly even in the presence of disturbances, which makes this combined method suitable for robotic applications.

1. Introduction

Dynamic movement primitives (DMPs) proposed by Ijspeert et al. [1] have become one of the most widely used tools for the generation of robot movements. Numerous applications can be found in the literature [2–5]. The DMP formalism is employed for describing goal-directed movements and includes second-order dynamics toward 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 the 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*, [3]), policy improvement with path integrals (PI^2, [5]), and policy learning by weighting explorations with the returns, (PoWER, [4]). Using those methods, robots were trained to acquire specific skills, for example jumping across a gap by a robot dog [5], hitting a baseball with a robot arm [3], or playing the ball-in-a-cup game using a humanoid robot [4].

Here we will consider the combination of DMP goal *and* shape learning. DMP goal learning was not much considered in robot experiments before [6,7] 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 known well enough. Thus, goal learning is not required. There are, however, many tasks where this is not the case, which happen 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 a 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 dynamic tasks, like throwing objects to hit a predefined target [7] or placing one object on top of the other where the stable configuration of the two objects is not known in advance. Similar problems will arise when working with tools, e.g. hammering, where an arm would be stopped at some specific position letting the weight of the hammer do the final hitting. Goal and shape are also important when working with soft materials, e.g. when putting a table cloth on the table, goal position as well as shape of the swinging movement when unfolding the cloth will be important.

* Corresponding author at: University Göttingen, Institute for Physics 3 - Biophysics, Bernstein Center for Computational Neuroscience, Friedrich-Hund-Platz 1, 37077 Göttingen, Germany. Tel.: +49 370 687 37788; fax: +49 0 551 397720.
E-mail addresses: m.tamosiunaite@if.vdu.lt (M. Tamosiunaite), bojan.nemec@ijs.si (B. Nemec), ales.ude@ijs.si (A. Ude), worgott@physik3.gwdg.de (F. Wörgötter).

0921-8890/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.robot.2011.07.004

Some indications of learning improvement by introducing task-level components (which might be compared to DMP goal learning) are found in [8]. In [7] meta-parameter learning is introduced where a function, mapping the target (e.g. height of a dart hit) to the goal and duration parameters of a DMP producing throwing movement has being developed, using Cost-regularized Kernel Regression; but simultaneous goal and shape learning was not addressed in this work.

We will use function approximation reinforcement learning [9,10] for goal learning and PI^2 or *NAC* for shape learning. A value function approximation technique for goal learning is chosen considering the structure of the reward landscape for the goal point in pouring. On the reward landscape one finds big areas with zero reward from where no liquid can be successfully poured and only a limited patch where reward can be obtained. Value-function-based procedures are known to be able to deal well with such reward structures. PI^2 and *NAC* methods are on the other hand chosen for shape learning as they have proven to be very successful for weight learning in DMPs in reinforcement learning scenarios. It will be shown how goal and weight learning processes can be combined in the same learning experiment without mutually jeopardizing each other's convergence. The pouring 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 methods used in this study are presented. In Section 3 results obtained with the pouring simulator as well as those obtained on the Mitsubishi Pa10 robot arm are provided. With the pouring simulator a more detailed scan of the parameter space is performed. With the real robot experiments both, learning from scratch as well as starting to learn from a demonstrated human trajectory, is shown. To demonstrate the potential of the applied learning algorithm, two more complex (redundant) learning tasks are also analyzed using the pouring simulator. In Section 4 the advantages as well as shortcomings of the methods are analyzed, and comparisons with alternative approaches are provided. In Appendices A and B algorithms are described in more detail. In Appendix C parameter values used in the experiments are provided.

2. Methods

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.

Statistical evaluation of the employed algorithms was performed using a pouring simulation. The simulation algorithm is provided in Appendix A.

As a platform for real robot 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 g). Afterward, the pouring action was executed. Pouring success was determined by the changing mass of the lower container measured by a scale. The setup is such that all spilled water immediately runs off the scale and is, thus, not weighed.

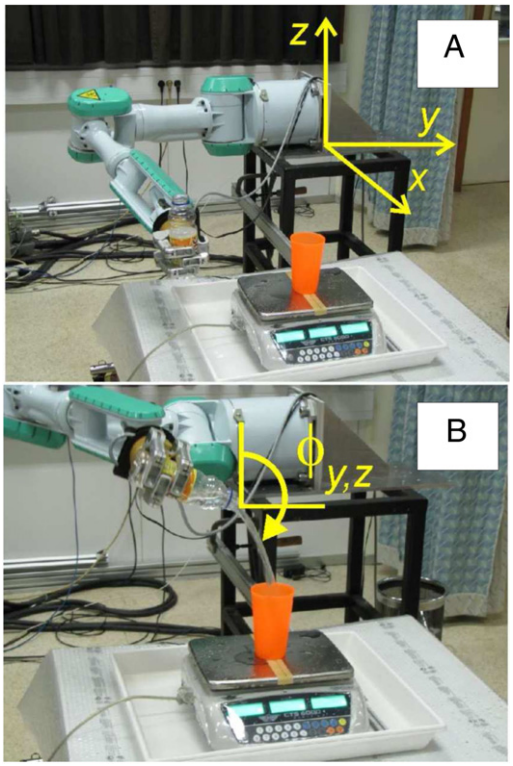

Fig. 1. Robot setup: Mitsubishi Pa10 robot arm and the scale for measuring correctly poured water. Spilled water runs off the scale. In panel A, the coordinate system used to evaluate the wrist position is shown. Panel B provides a close to planar view onto the (y, z) plain and the tilting angle ϕ is visualized in this plain. Rotation is performed around the neck of the bottle.

2.2. DMP usage

Movements were generated using dynamic movement primitives [11].

$$\dot{v} = \frac{1}{\tau}\left(C(g-u) - Dv + f(w)\right), \tag{1}$$
$$\dot{u} = \frac{v}{\tau},$$
$$\dot{w} = -\frac{\alpha w}{\tau}$$

where v and u represent velocity and position of the system, w is the phase variable, which defines the time evolution of the trajectory, τ is the temporal scaling factor, C is the spring constant, and D the damping.[1]

The non-linear part $f(w)$ was defined using radial basis functions [1]:

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

where ψ_l are Gaussian functions defined as $\psi_l(w) = \exp(-\frac{1}{2\kappa_l^2}(w - c_l)^2)$. Parameters c_l and κ_l define the center and the width of the lth basis function, while ρ_l are the adjustable weights used to obtain the desired shape of the trajectory. We use DMPs with local generalization properties as global generalization [12] is not required for our task.

[1] Note a summary of the parameter values used in this study is provided in Appendix C.

In most experiments three DMPs were used: one for the horizontal side displacement of the end effector, $y(t)$; one for the vertical displacement of the end effector, $z(t)$, both in the robot-based coordinates (see Fig. 1(A)), and the last one for the tilting angle parallel to the frontal plane of the robot, $\phi(t)$ (Fig. 1(B)) which corresponds to roll in tool coordinate frame. Tilting was performed around the opening of the bottle. To simplify the situation, the horizontal forward displacement of the end effector 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 ρ_l, $l = 1, 2, \ldots, L$ was learned. Note, shapes of the horizontal and the vertical displacement of the end effector $y(t)$ and $z(t)$ were kept unchanged, using the initial trajectories of our second-order linear system, in order 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. In the simulation experiments the trajectory was hard-limited between zero and π. The aforementioned setup we will call $2 + 1$ *setup* (as two goals and one weight set are being learned).

To show the potential of the learning methods used, we have in addition implemented learning in several more complex setups. In the first of those more complex setups, goal parameters as well as shape parameters were learned for all three DMPs: $y(t), z(t)$ and $\phi(t)$. That is, six entities were learned: three goals and three weight sets ($3+3$ *setup*). Even though a simpler formulation of the pouring task as described before is better for the 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. Finally, we performed several experiments for learning to pour in 5D, where (x, y, z) position (robot-based coordinates) and roll and pitch for rotation (tool coordinate frame) were used ($5 + 5$ *setup*).

For DMPs we used parameters $C = 36, D = 3, \alpha = 0.1$, and $\tau = 1$ or $\tau = 2$ as well as step $\mathrm{d}t = 0.017$ for the Euler integration. In the real robot experiments we used $T = 300$ discretization steps. This corresponds to pouring movements of 5 s duration (sampling frequency 60 Hz). 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 different parameters. We used $L = 15$ kernels in the non-linear part of the DMP.

2.3. Reinforcement learning methods

Two different reinforcement learning approaches were used: (1) value-function-based RL, where for each state the value of being in the state is determined by learning [9] and (2) direct policy search methods, where one does not keep information about values of states, but instead directly introduces action parametrization [13]. Throughout learning the parameters are adjusted to optimize the desired goal-directed behavior. We are combining both techniques, where the value function approximation approach is used for goal learning, and the direct policy search approach for shape learning.

One attempt to pour we will call a trial. The learning procedures we will describe next are organized in "epochs", where an epoch is a set of trials. Note, for different methods the utility of an epoch will be different.

2.3.1. Goal learning method

For value function approach we used a reinforcement learning method with function approximation developed by us in an earlier study [14] and modified as described next. The value function $V(s)$ is defined as follows

$$V(s) = \sum_{k=1}^{N} \theta^k \Phi^k(s) \Big/ \sum_{k=1}^{N} \Phi^k(s), \tag{3}$$

where $\Phi^k(s)$ is the activation function of the kth kernel in state s, θ^k are the weights associated with the kth kernel function, and N is the overall number of kernels in the system. Let us further explain the method using the example of goal learning in 2D coordinates (y, z) ($2 + 1$ *setup*). 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 of 30×30 cm in simulation and 25×20 cm on a real robot (see Appendix C) with $\sigma_V = 2$ cm. A value of $N = 200$ was used in the $2 + 1$ *setup*, $N = 2000$ was used in the $3 + 3$ *setup* and $N = 10\,000$ was used for the $5 + 5$ *setup*.

The exploration actions were divided into epochs of seven trials each. First, the algorithm was started with initial state (y_0, z_0). For a given epoch, to define a new exploratory action, the 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}:

$$A_{\text{final}} = \frac{1}{1+c}(\Delta_{\text{current}} + cA_{\text{previous}}), \tag{4}$$

where at the beginning of learning $c = 1.5$ was used, while later c was reduced by 0.1 in each epoch, $c \geq 0$. This 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. For the beginning of a new epoch the state (y, z) that had produced the highest reward in the previous epoch was used as starting point.

In the beginning of learning, the probability of random exploration was set to 0.5. Exploration was reduced by 0.05 in each epoch to increasingly better adhere to the learned components, but was not allowed to become smaller than zero.

The change in the value of θ^k is performed as follows:

$$\theta^k = \theta^k + \mu[r + \gamma_V V(s') - \theta^k]\Phi_N^k(s), \tag{5}$$

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

$$\Phi_N^k(s) = \Phi^k(s) \Big/ \sum_{k=1}^{N} \Phi^k(s). \tag{6}$$

The rule in Eq. (5) is called averaging function approximation rule and is considered to perform more stably in function approximation schemes [10,15] as compared to standard methods [9]. As reward r we used the amount of liquid correctly poured into the target container.

One has to note that reinforcement learning here was used in a non-standard way, where the objective was to optimize the reward function for parameter "g" using value function approximation. However, the advantage of the technique to systematically deal with "delayed reward" persists.

2.3.2. Shape learning methods

For shape learning as a primary means we have used the recently developed PI^2 method [5], but also made a comparison to

the natural actor critic (NAC) [3], which has been very influential in the last years. We are giving a detailed description of those algorithms in Appendix B 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. [5] and Peters and Schaal [3].

Policy improvement with path integrals PI^2: This method is derived from the principles of stochastic optimal control. 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. Afterward, weights are updated to reduce cost.

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:

$$q(t) = \begin{cases} 10\, a_{\text{spill}}(t), & \text{if } \sum_{j=0}^{t} a_{\text{target}}(j) = 0 \\ a_{\text{spill}}(t), & \text{if } \sum_{j=0}^{t} a_{\text{target}}(j) > 0, \end{cases} \tag{7}$$

where $a_{\text{spill}}(t)$ is an amount of spilled liquid in time step t and $a_{\text{target}}(t)$ 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:

$$q_{\text{terminal}} = \sum_{t=0}^{T-1} a_{\text{spill}}(t), \tag{8}$$

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 method from [3], 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. Also, we were annealing the exploration noise independently of the shape parameters. 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.

2.3.3. Combination of goal and shape learning

We have combined value function approximation for goal parameter learning and direct policy search methods (policy gradient or reward-weighted averaging) for weight learning into a single procedure. A block diagram showing the principles of interaction of the two learning methods (on an example where the PI^2 method is used for weight learning) is presented in Fig. 2. 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 two reinforcement learning procedures, we were using the same K trials to also vary the goal parameters. Updates of the function $V(t)$ were made after each trial, according to Eq. (5). Consequently, the two learning processes were acting together.

3. Results

3.1. Learning in simulation experiments in $2 + 1$ setup

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. In Fig. 3(A) an example learning curve is shown obtained using the combination of goal and shape learning, where the shape is learned using the PI^2 algorithm in the $2 + 1$ *setup*. We plot PI^2-cost (Eq. (7)) 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 to the degree which the finite steps performed in goal search procedure allow. One cannot make steps arbitrarily small in goal learning, as the step size has to be matched to the kernel size in the value function approximation approach. In Fig. 3(B) the learned trajectories are shown. For $y(t)$ 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 steeper 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.

Parameters for value function approximation employed for goal learning were taken from our previous studies [16] (for the numerical values see Appendix C). 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. 4). 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 was ten (see Eq. (7)). Values slightly above ten arise because of discretization effects. In Fig. 4, lower costs earlier in learning indicate better performance. In the first four rows we provide histograms for four different values of the noise variance $\sigma_P = 10, 30, 50$ and 80 for the $2 + 1$ *setup*. One can see that learning results are good in the range of $\sigma_P = 30$–50. Consequently, for further simulation experiments we were using $\sigma_P = 30$.

Next, we have performed an experiment with first goal, then weight learning (row 5 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 ($\sigma_P = 30$). One can observe, that goal and weight learning performed together (Fig. 4, row 2) produced better results as compared to first goal then weight learning (Fig. 4, row 5).

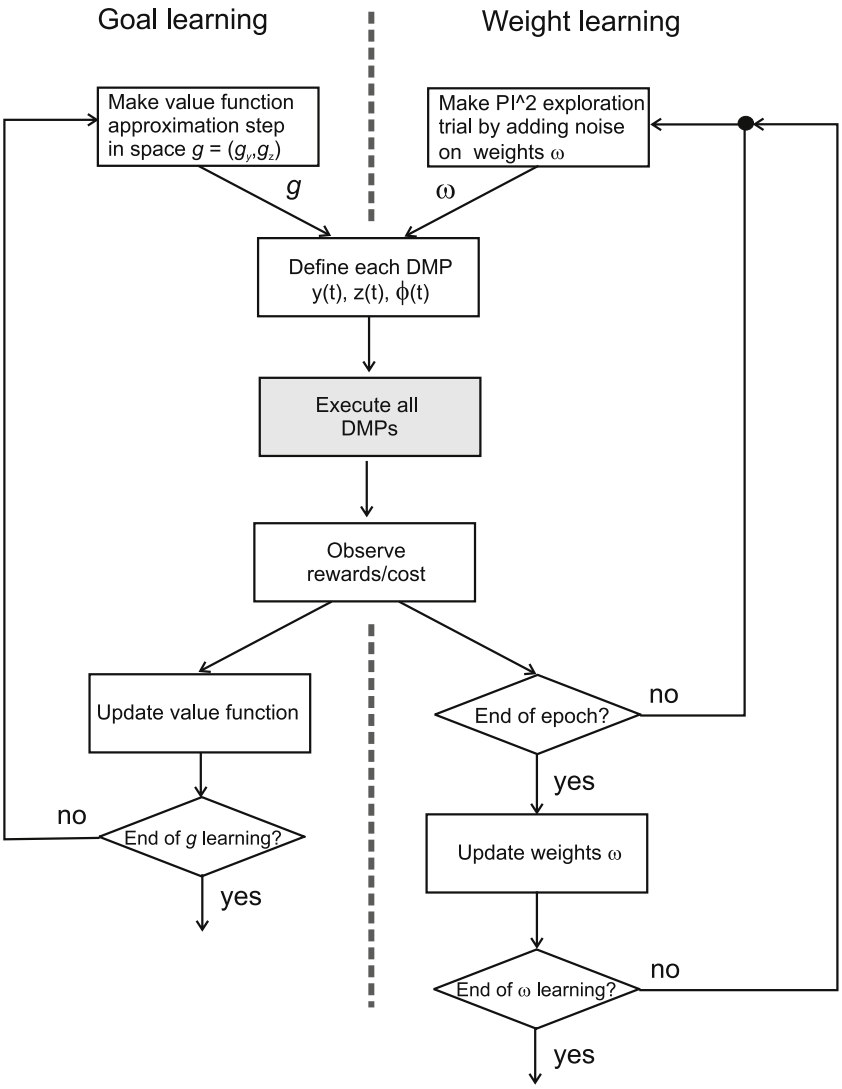

Fig. 2. Block diagram of interaction between goal and weight learning.

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. 4 row 6. 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 Appendix B. We did not include exploration noise variance into the natural gradient evaluation procedure. This is due to the fact that, when including 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. 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 toward 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 $\sigma_P = 30$.

3.2. Learning in simulation experiments 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 $\phi(t)$ simultaneously. That is, six entities were learned: three goal parameters and three weight sets leading to a redundant setup ($3 + 3$ *setup*). For each DMP, $L = 15$ shape parameters were used. The noise added on the 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 in amplitude 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. Note, due to goal and weight update one gets direct interference of the parameters at each individual DMP (Fig. 2, box 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 executed by the robot at the same time (Fig. 2, gray box). The shape parameters of each DMP, on the other hand, were optimized independently from all other DMPs' shape parameters, as suggested by Schaal (personal communication).

An example of a learning curve, as well as trajectories before and after learning for this redundant learning task are provided in

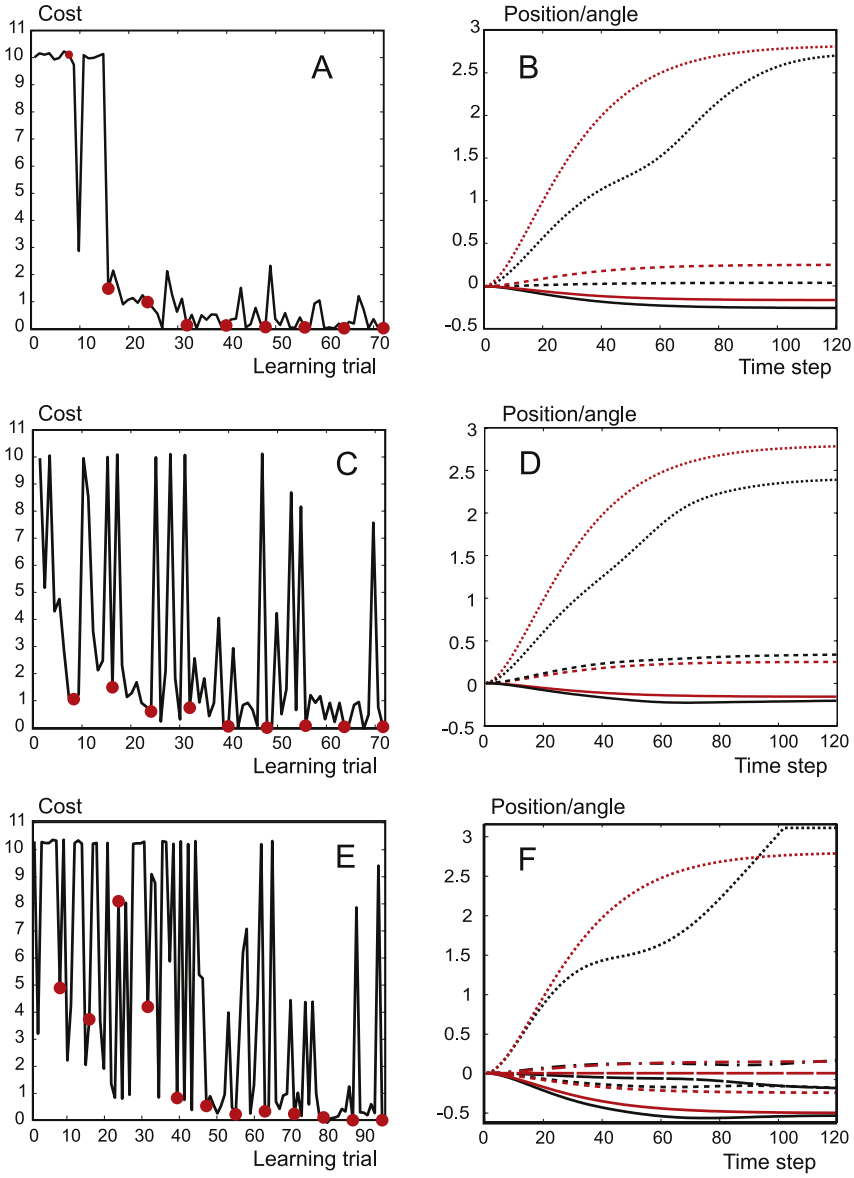

Fig. 3. (A) Cost decrease with the number of learning trials in $2+1$ *setup*. Solid line—overall cost decrease, red dots—noise-free trials. Noise standard deviation $\sigma_P = 30$. (B) trajectories generated by DMPs at the beginning (red) and end of learning (black), solid line—$y(t)$, dashed line—$z(t)$, 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 for the $3+3$ *setup* (i.e. three goal values and three sets of weights are learned). (E, F) analogous to (A, B) for $5+5$ *setup*. In addition $x(t)$ is shown by the long-dashed line and the pitch of the bottle by the dash-dotted line. Note, several different tilting shapes can lead to correct pouring, also variability is allowed in the height component $z(t)$. Consequently, final trajectories in B, D and F are not identical. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Fig. 3(C) and (D). One can observe that relatively good performance is achieved already after the first epoch, but the final cost-drop to zero happens later. Also in the redundant case toward the end of learning there are many exploration trials which show bad performance. This, though, does not affect the noise-free trials. If one looks at the trajectories of the redundant setup ($3+3$) after learning (Fig. 3(D)), one can observe that the displacement of the wrist toward the glass $y(t)$, which is crucial for correct pouring (solid line), is shaped by the weights in such a way that it reaches the correct pouring position slightly earlier (trajectory is curved downwards), as compared to the non-redundant ($2+1$) case (see Fig. 3(B)).

The results on performance statistics for the $3+3$ *setup* are summarized in row 7 in Fig. 4. One can see that learning is a fraction slower as compared to the best non-redundant setup in row 2. In the $3+3$ *setup* only in one out of 20 trials learning was less successful at the end, which is remarkable as dimensionality is now much higher. Thus, the disruptive effect of the interference between DMP parameters was small.

Finally, in Fig. 3(E) and (F) we show learning curves and trajectory examples for the $5+5$ *setup*. Additional trajectories on the plot are plotted by a long-dashed line (x coordinate) and a dash-dotted line (pitch of the bottle). The cost histograms for the $5+5$ *setup* are provided in the last row of Fig. 4. As more experiments were required in higher dimensions, we provide results after four, eight and 12 epochs in the histogram columns. One can observe that learning is within 12 epochs successful in half of the cases. In the rest of the cases more learning epochs are required. In addition, in higher dimensions it would be advantageous to use more trials in an epoch or reuse previous successful trials [4], as in the higher-dimensional space at the end of learning it is hard to hit on an improved behavior that reduces cost within the seven trials per epoch used throughout this study. In spite of this, to remain comparable, we did not change the trial number per epoch for obtaining the histograms for the $5+5$ *setup*.

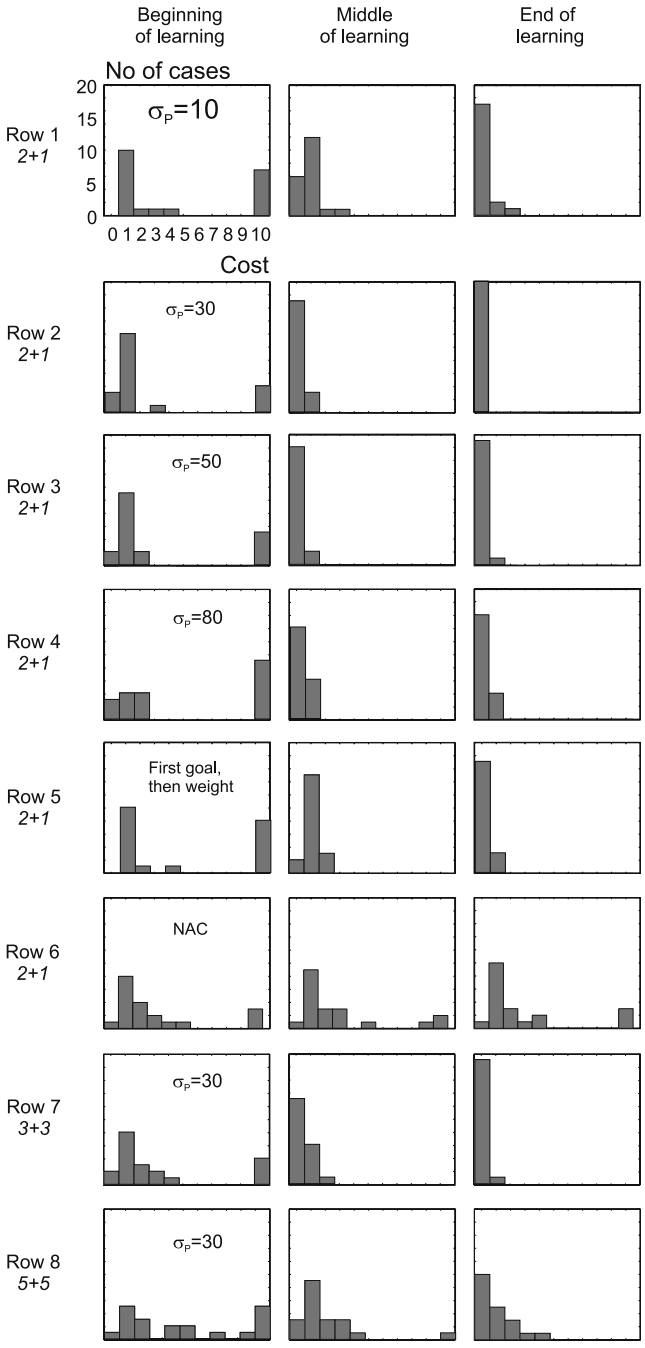

Fig. 4. Cost value histograms showing our parameter investigation. First column—result after the first third of learning, second column—after the second third, third column—at the end of learning. First four rows show effects of exploration noise increase ($\sigma_P = 10, 30, 50, 80$) in $2+1$ *setup*, row 5 – first goal (4 epochs), then weight (the rest 5 epochs) learning ($\sigma_P = 30$), $2+1$ *setup*, row 6 – *NAC* method for the $2+1$ *setup*, row 7 – $3+3$ *setup*, row 8 – $5+5$ *setup*, $\sigma_P = 30$ for $\phi(t)$ and $\sigma_P = 3$ for the rest of the variables, for other parameters see Appendix C.

3.3. Learning in a real robot experiment

For real robot experiments any learning algorithm must converge quickly because real-world trials are in general expensive. Furthermore, learning should be robust against fine-tuning of parameters, against measurement errors, as well as against external, uncontrollable noise. We performed real robot pouring experiments for the $2+1$ *setup*. In our experiments, the reward had a margin of error of about ± 10 g, which corresponds to an accuracy of about $\pm 5\%$ of the total liquid. Furthermore, the readings from the scale in some experiments were on purpose occasionally wrong (in one out of 20 trials there was a false reading of zero as if pouring was totally unsuccessful).

First we performed an experiment with only shape learning using the PI^2 algorithm. This experiment is designed to test whether in the analyzed setup the shape of the tilting trajectory matters and if pouring results can be improved with learning of the tilting shape alone. Then goal and shape learning are performed to evaluate success of the combined learning procedure. Then an experiment with tool change is performed, in order to check if previously learned parameters can be usefully applied after change of the tool (the bottle from which the water is poured in our case). Finally, an experiment is performed, showing that information from human demonstration can be included into the learning procedure.

3.3.1. Learning the shape

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. 5(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 final trajectories of the three curves ($\phi(t)$ as learned and $y(t)$, $z(t)$ fixed) are shown in Fig. 5(B). The solid line denotes $y(t)$, the dashed line $z(t)$, and the dotted line the tilting trajectory $\phi(t)$. In this experiment the goal position was set on purpose a bit in front of the glass and the learning process had, thus, to tune the tilting trajectory for a shallow tilt. This way the DMP weights obtained negative values and the learned pouring trajectory was shallower 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. 5(A)) a behavior close to optimum (almost all the poured liquid is correctly poured into the lower container) is attained after one learning epoch, but then the weight update is unsuccessful and the correct pouring shape is stably attained only after three learning epochs (first 24 trials). The instability in the update arises due to complex patterns of water running out of the bottle and because of measurement imprecisions (imprecisions in filling of the glass and imprecisions of water running off the scale).

3.3.2. Combined learning of goal and shape

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 experiments with the Pa10 robot arm are shown in Fig. 6. In panels A and B one can see two experiments where simulated measurement errors were not introduced. The exploration noise provided some jumps in cost also at the end of learning, but much less jumps occurred for the noise-free trials (red dots). Update No. 6 in Fig. 6(A) was unsuccessful due to small measurement imprecisions (imprecisions in glass filling and small amounts of water not completely running of the scale) but in the next epoch this upward deviation was corrected. Variability of the remaining red dots at the end is within the precision limit for these experiments.

In Fig. 6(C) and (D) learning curves obtained in the case where measurement errors were introduced are shown. The false readings are shown as gaps in the curves. We observe that even with relatively frequent false readings (gaps in the curve) the proposed combined goal and shape learning algorithm was stable and convergent.

For all experiments in Fig. 6, trajectories with too small tilt were rejected, as with small tilt it is impossible to pour any water

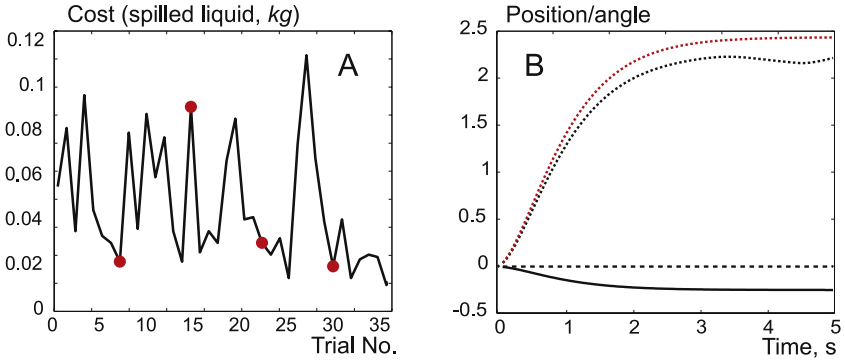

Fig. 5. (A) Cost decrease with the number of learning trials for shape only learning in real robot experiments. Solid line—overall cost decrease, red dots—noise-free trials; $\sigma_P = 30$. (B) Trajectories generated by DMPs: red—before learning, black—after learning, $y(t)$ is shown as solid line, $z(t)$ as dashed line, $\phi(t)$ as dotted line. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

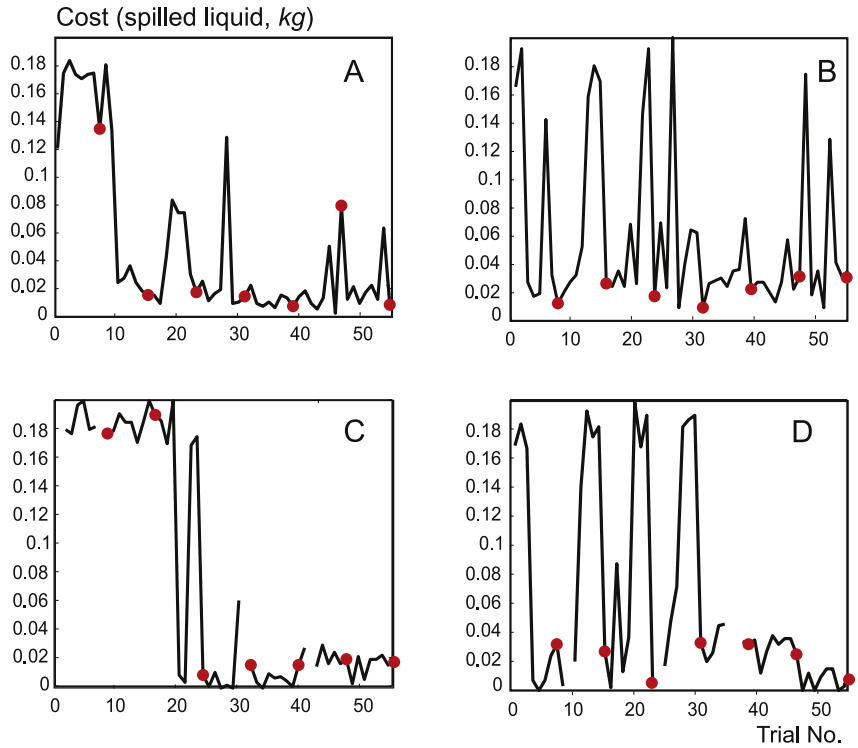

Fig. 6. Cost decrease with the number of learning trials for combined goal and shape learning. Solid line—overall cost decrease, red dots—noise-free trials. (A, B) without simulated measurement errors, (C, D) with simulated measurement errors denoted by breaks in the curves. Parameter $\sigma_P = 30$; σ_P is multiplied by a factor of 0.85 after each epoch starting with epoch No. 3. The new noise is added and trajectory is recalculated if the last third of the trajectory has an average less than $\frac{5}{8}\pi$. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

out of the bottle. In our setup the liquid could not be poured out of the bottle using tilt angles less than approx. $\frac{5}{8}\pi$, judged at the end of tilting. On failure to pass this value, a new trajectory was generated. Also the exploration noise starting from the epoch No. 3 was reduced with progression of learning. The noise was reduced to the extent that at the end of a learning experiment it was approximately twice lower than in the beginning.

3.3.3. Relearning after change of bottle

In Fig. 7(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 relearning epoch. Following preliminary learning less trials are required after changing the tool, as compared to learning from scratch. This suggests that reinforcement learning may be successfully used for modifying a learned trajectory with respect to small changes in the configuration of the task.

3.3.4. Learning using data from human demonstration

Fig. 7(B) and (C) shows results for an experiment where the movement was initialized with weights obtained from human demonstration. Weights were provided for the tilting DMP and were obtained using regression techniques [17]. We assumed that the goal position for $y(t)$ and $z(t)$ cannot be extracted with good enough precision from such a demonstration. Instead, we assigned an arbitrary initial goal position close to target. Goal for $y(t)$ and $z(t)$ and shape for $\phi(t)$, were allowed to change by reinforcement learning. Relatively good pouring in this experiment was obtained already after two learning epochs. The correct tilting trajectory assigned at the beginning helped to successfully accomplish learning. Some properties of the initial shape of the trajectory persisted through learning, while performance was substantially improved.

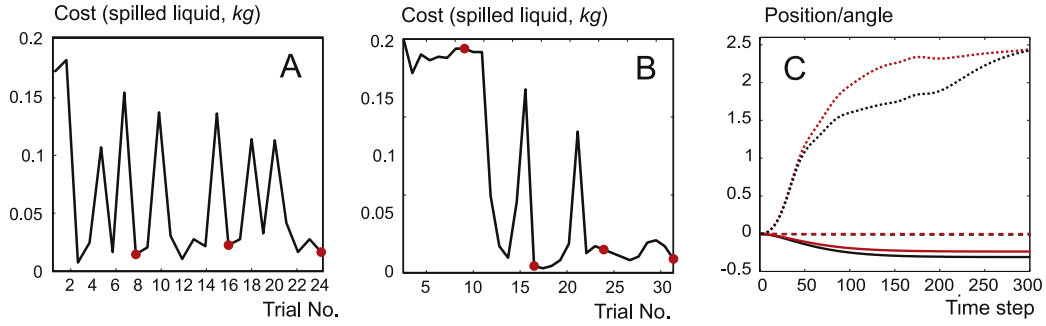

Fig. 7. (A) Cost decrease with the number of learning trials for relearning to pour from a different bottle and (B) learning using weights obtained from human demonstration. Solid line shows overall cost decrease, red dots show noise-free trails. In (C) DMP trajectory $\phi(t)$ obtained from human demonstration (red-dotted) and the one modified by learning (black-dotted), $y(t)$—solid, $z(t)$—dashed, red—before goal learning, black after goal learning, $\sigma_P = 30$. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

4. Discussion

Two different reinforcement learning techniques, value function approximation-based learning and direct policy search (policy gradient or reward-weighted averaging), were joined together into one procedure for learning to pour. Value function approximation was used for DMP goal learning, while the direct policy search approach was used for the DMP shape learning. The recently developed PI^2 method was employed and compared to the natural actor critic. Learning to pour liquid into a glass was achieved in 8–32 trials, which is a good score for a robotic application including complex learning. Relearning (of a different bottle) or recalibration after human demonstration took 8 and 16 trials, respectively. In real experiments, despite the presence of relatively large measurement noise and outliers, we observed excellent robustness of the PI^2 algorithm in combination with value function approximation for goal learning.

4.1. Motivating our approach

Combination of DMP goal and shape learning is a novel approach, where other authors have also pointed out that such a combination should be investigated [6] and only very few related studies are existing [8,7]. We used two different methods for the learning of the two components (shape and goal), instead of doing the whole learning using the same procedure, to avoid mutual interference (working against each other) of the two entities. Using goal and weight (at the end of DMP) as free parameters makes the shape of the DMP over-determined. One can change the final position of the trajectory either by changing the goal or the weights at the end of the DMP. Our previous experiments with the natural actor critic [3] have shown that learning both goals and weights in the same procedure lead to goal and weights working against each other. Excessively high weights were obtained to compensate for small goal values or vice versa. Thus we have chosen to use two entirely different procedures to reduce this interference.

The rationale for making a combination of value function approximation with direct policy search methods was to use value function approximation for a small yet important subtask inside a bigger problem, and direct policy search for the rest of the task. Though value function approximation cannot handle very high dimensions well, it provides quick and reliable learning for constrained problems [14].

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 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 the occasional combination of a 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 DMPs during execution. Learning in the redundant setups ($3 + 3$ and $5 + 5$) has also proven that the here introduced combination of algorithms can deal with tasks of relatively high dimensionality. In this case up to ten entities were learned in parallel, up to five goals and up to five sets of 15 weights each.

When the goal is set a priori, and only the weights are changed by learning, as in previous studies [3,4], 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 do not start counter-acting each other (big positive goals counteract the big negative weights, or vice versa). On the other hand, finding direct path to optimum in the higher dimension can be easier. The interference of goal and weight was not the case with the combined value function approximation and PI^2 and the combination proved favorable in this study.

One could argue that using value function approximation 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 [14]. 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 [14] 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.

Still the question remains to what degree such a combination would be robust against parameter changes. We have checked all methods in a wide parameter range, changing exploration noise for the PI^2 and exploration noise and learning rate for the NAC. 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 with respect to occasional incorrect feedback. Our scale sometimes returned 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 important for experimental robotics. 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 direct policy search. One of those is the PoWER method [4]. This method is based on the idea of expectation maximization. Even though different methods for direct policy search 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 more reliable convergence of the learning methods. A similar conclusion is derived by Kober and Peters [18]. To our observation, modifications of the details of the PI^2 algorithms do not lead to big changes in the final result.[2] Although, one would have to check if those conclusions still hold in extremely high-dimensional tasks, which was not investigated in our study. Our observations on direct policy search being non-sensitive to details are also supported by those of Kolter and Ng [19], who found that quite a coarse algorithm can perform very well in direct policy 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 shown good results in complex robot control tasks [20,21].

Potentially, one can also learn goals as policy parameters, because these methods tolerate multiple dimensions better as compared to value function approximation techniques. This question has to be investigated in the future as gradient procedures have been observed to have certain drawbacks in searching the goal point for the pouring task. As the reward landscape for the goal point has big areas of zeros, and only in limited area non-zero values are obtained, when sampling for gradient one obtains a very sparse structure (all zeros or e.g. just one or two non-zero points). Thus e.g. a finite difference gradient [22,23] estimate cannot be performed reliably.

In this paper we showed a successful combination of goal and weight learning using different frameworks. This combination also performed well for trajectory modification after having initially learned a movement from demonstration. This is important because learning from demonstration remains possibly the most efficient movement learning framework in many complex robotic applications like service robotics [24–26]. Similarly only a short episode of reinforcement learning needs to be applied for recalibration when changing the tool (e.g. changing the bottle in our experiment). Thus, we believe that the here demonstrated combination can be successfully applied also to other robotic tasks.

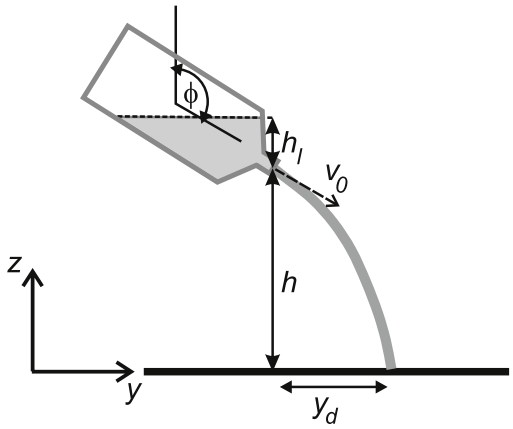

Fig. A.8. Pouring model setup.

Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007–2013 (Specific Programme Cooperation, Theme 3, Information and Communication Technologies) under grant agreement no. 269959, Intellact and from the German Ministry of Science, grant BCCN II, D1, BCCN II, Göttingen, no. 01GQ1005A and Slovenian Research Agency grant J2-2348.

Appendix A. 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. Note, no attempts were made to make this model fully accurate, which is very difficult, due to having to model turbulent flow. According to the Bernoulli equation, the exit velocity of the liquid equals

$$v_o = \begin{cases} \sqrt{(2gh_l)}, & h_l > 0 \\ 0, & h_l \leq 0, \end{cases} \tag{A.1}$$

where g is the gravitational acceleration and h_l is the liquid level according to Fig. A.8. 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

$$\begin{aligned} v_y &= v_0 \sin(\varphi) \\ v_z &= v_0 \cos(\varphi) + gt, \end{aligned} \tag{A.2}$$

where t denotes time.

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

$$y_d = v_y t \tag{A.3}$$

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

and the liquid flow is

$$\phi = A v_0, \tag{A.4}$$

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

[2] This notion 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 trials of the epoch for weight update in step 4, see the section "Parameter update rule for PI^2" in Appendix B, etc., etc. Most of these modification did not significantly change learning success.

Appendix B. Direct policy search algorithms

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 [3,5]. Algorithmic "peculiarities" and specific alteration, which we found useful are being commented.

B.1. Definitions

1. DMPs are used as given in Eq. (1) (see main text) and integrated using the Euler method, where time steps are numbered $t = 0, 1, 2, \ldots, T-1$.
2. Kernels in the non-linear part of the DMP are indexed with $l = 1, 2, \ldots, L$, where L is the overall number of Gaussian kernels.
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 M trials), and the testing stage (used only for PI^2, one trial). Thus, one learning epoch includes $K+1$ trials for the PI^2 and $K+M$ trials for the *NAC*. In the exploration stage of an epoch trials are indexed $k = 1, 2, \ldots, K$. In the learning rate probing stage trials are indexed $m = 1, 2, \ldots, M$.

B.2. 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 $\rho_l,\ l = 1, 2, \ldots, L$, the weight vector we will notate by $\boldsymbol{\rho}$.
2. Repeat for several learning epochs (or until convergence of cost measure Q):
   (a) Repeat for trials $k = 1, 2, \ldots, K$ 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 $\boldsymbol{\rho}$ and obtain weight vectors $\boldsymbol{\rho}_{k,t}$ (or add noise only on the leading kernel).[3]
      iii. Generate trajectory $u_k(t)$ with noisy weights. Among the components required to obtain $u_k(t)$ are kernel activation vectors $\boldsymbol{\psi}_{k,t}$, save those vectors in memory.[4]
      iv. Execute trajectory $u_k(t)$ 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 $\boldsymbol{\rho}$ 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 $u(t)$ with noise-free weights $\boldsymbol{\rho}$.
      ii. Execute trajectory $u(t)$.
      iii. Measure/obtain cost function $q(t)$ and terminal cost q_{term}.
      iv. Evaluate overall cost measure $Q = q_{\text{term}} + \sum_{t=0}^{T-1} q(t)$.

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 the control penalty matrix.
2. For each k, t compute cost including control penalty term: $S_{k,t} = q_{\text{term}} + \sum_{j=t}^{T-1} q_{k,j} + 0.5\sum_{j=t+1}^{T-1}(\boldsymbol{\rho} + \boldsymbol{M}_{k,t}\boldsymbol{\epsilon}_{k,t})^T\boldsymbol{R}(\boldsymbol{\rho} + \boldsymbol{M}_{k,t}\boldsymbol{\epsilon}_{k,t})$.
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. [5] suggest to approximate $e^{-\frac{1}{\lambda}S_{k,t}} \approx \exp\left(-c\frac{S_{k,t}-\min_k S_{k,t}}{\max_k S_{k,t}-\min_k S_{k,t}}\right)$ with $c = 10$, which we also did.
4. For each time point compute an update term $\delta\boldsymbol{\rho}_t = \sum_{k=1}^{K} P_{k,t}\boldsymbol{M}_{k,t}\boldsymbol{\epsilon}_{k,t}$. See note.[5]
5. Using cumulative sums, summarize updates along the trajectory into one weight update value $\delta\rho_l = \frac{\sum_{t=0}^{T-1}(T-t)\delta\rho_{l,t}\psi_{l,t}}{\sum_{t=0}^{T-1}(T-t)\psi_{l,t}}$.
6. Update weights $\boldsymbol{\rho} \leftarrow \boldsymbol{\rho} + \delta\boldsymbol{\rho}$.

B.3. 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 $\boldsymbol{\delta} = \nabla_{\rho,\sigma_N}\log\Pi(\dot{v}(t,\rho),\sigma_N)$ according to parameters $\rho_l,\ l = 1, 2, \ldots, L$ and σ_N (see note[6]), where the policy $\Pi(\dot{v}(t,\rho),\sigma_N)$ is defined by distribution of acceleration values $\dot{v}(t) \sim \mathcal{N}(\bar{\dot{v}}(t), \sigma_N)$ and the expression for $\bar{\dot{v}}(t)$ is obtained from the right hand side of Eq. (1).
2. Initialize algorithm with weights $\rho_l,\ l = 1, 2, \ldots, L$ and exploration noise value σ_N.
3. Repeat for several learning epochs (or until convergence of return R):
   (a) Repeat for trials $k = 1, 2, \ldots, K$ 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 $\boldsymbol{\rho}$ as they appear (without noise).
      iii. Add noise $\xi_{k,t}$, to acceleration $\dot{v}$ in all time steps of the DMP integration.
      iv. Generate trajectory $u_k(t)$ using noisy acceleration, save in memory kernel activation vectors $\boldsymbol{\psi}_{k,t}$ and trajectory $w_k(t)$ from DMP equation for $w(t)$ (Eq. (1)).
      v. Execute trajectory $u_k(t)$ 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 $\boldsymbol{\psi}_{k,t}$, trajectories $w_k(t)$, and reward functions $r_k(t)$ (see Pseudo-Code in the subsection "Parameter update rule for the *NAC*").

[3] While the core algorithm would allow adding noise to all kernels, Theodorou et al. [5] 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 K noise vectors (instead of $K \times T$) $\boldsymbol{\epsilon}_k$, 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. [5] – used noise on the leading kernel only.

[4] For the sake of more intuitive notation we will use notation $f(t)$ (e.g. $u_k(t)$), 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.

[5] 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 $\boldsymbol{M}$.

[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 $m = 1, 2, \ldots, M$ trials in the learning rate probing stage of the epoch[7]:
  i. Generate new learning rate value γ_m and save in memory.
  ii. Modify weights $\boldsymbol{\rho}$ with the learning rate γ_m and obtain temporary weights $\boldsymbol{\rho}_{\text{temp}} = \boldsymbol{\rho} + \gamma_m \boldsymbol{g}_{NG}$.
  iii. Generate trajectory $u_m(t)$ with the temporary weights $\boldsymbol{\rho}_{\text{temp}}$.
  iv. Execute trajectory $u_m(t)$.
  v. Measure/obtain reward function $r_m(t)$.
  vi. Evaluate the return $R_m = \sum_{t=0}^{T-1} r_m(t)$ and save in memory.
(e) Find the optimum γ_m according to saved R_m values, let us call it γ_N, and permanently modify the weights $\boldsymbol{\rho} = \boldsymbol{\rho} + \gamma_N \boldsymbol{g}_{NG}$.
(f) Choose the return of the optimum trial from the learning rate probing procedure as the overall return of the current learning epoch $R = \max R_m$.

Parameter update rule for the NAC

We used the episodic *NAC* version with time-variant baseline from [3].

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 $\boldsymbol{F}_k = \sum_{t=0}^{T-1} \left(\sum_{j=0}^{t} \boldsymbol{\delta}_{k,j}\right) \boldsymbol{\delta}_{k,t}^T$.
3. Calculate the average over k trials $\boldsymbol{F} = \frac{1}{K} \sum_{k=1}^{K} F_k$.
4. Calculate gradient for each trial $\boldsymbol{g}_k = \sum_{t=0}^{T-1} \left(\sum_{j=0}^{t} \boldsymbol{\delta}_{k,j}\right) \alpha_t r_t$, 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 $\boldsymbol{H}$.
8. Obtain weighted reward vector $\boldsymbol{\vartheta} = [\vartheta_0, \vartheta_1, \ldots, \vartheta_{T-1}]^T$, where $\vartheta_t = \frac{1}{K} \sum_{k=1}^{K} \alpha_t r_t$.
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 $\boldsymbol{g}_{NG} = \boldsymbol{F}^{-1}(\boldsymbol{g} - \boldsymbol{H}\boldsymbol{b})$.

Appendix C. Parameters used in this study

Parameter type	Parameter name	Simulation	Real robot
Robot workspace	Wrist forward displacement $x(t)$	[0.5, 0.8]	n.a.
	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 $\phi(t)$/roll	$[0, \pi]$	$[0, \frac{7}{9}\pi]$
	Pitch	$[-\pi/4, \pi/4]$	n.a.
DMP	C	36	36
	D	3	3
	α	0.1	0.1
	τ	1	2
	No of kernels L	15	15
	Kernel width κ	0.005	0.005
	Integration step dt	0.017 s	0.017 s
	Trajectory length T	120	300
Value function approximation	No of trials in an epoch	7	7
	No of epochs performed	6	6
	No of kernels N in 2D	200	200
	No of kernels N in 3D	2000	n.a.
	No of kernels N in 5D	10 000	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	10	30
		30	
		50	
		80	
	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 K	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.

[7] This probing for the value of the learning rate was not included in the original procedure [3], but we found it necessary in order to stabilize learning in our pouring task.

References

[1] A.J. Ijspeert, J. Nakanishi, S. Schaal, Movement imitation with nonlinear dynamical systems in humanoid robots, in: Proc. IEEE Int. Conf. Robotics and Automation, ICRA, 2002, Washington, DC, 2002, pp. 1398–1403.
[2] B. Perk, J. Slotine, Motion primitives for robotic flight control, 2008. arXiv:cs/0609140v2.
[3] J. Peters, S. Schaal, Reinforcement learning of motor skills with policy gradients, Neural Networks 21 (2008) 682–697.
[4] J. Kober, J. Peters, Policy search for motor primitives in robotics, in: Adv Neural Inf. Process Syst., vol. 22, 2009.
[5] E. Theodorou, J. Buchli, S. Schaal, Reinforcement learning of motor skills in high dimensions: a path integral approach, in: Int. Conf. of Robotics and Automation, ICRA, 2010, 2010, pp. 2397–2403.
[6] J. Peters, J. Mulling, J. Kober, D. Nguyen-Tuong, O. Kroemer, Towards motor skill learning for robotics, in: Proc. 14th Int. Symp. on Robotics Research, ISRR 2009, 2009, pp. 1–14.
[7] J. Kober, E. Oztop, J. Peters, Reinforcement learning to adjust robot movements to new situations, in: Robotics: Science and Systems Conf., RSS 2010, MIT Press, Cambridge, MA, USA, 2010, pp. 1–8.
[8] C.G. Atkeson, S. Schaal, Robot learning from demonstration, in: Proc. 14th Int. Conf. Machine Learning, ICML, 1997, Morgan Kaufmann, San Francisco, CA, 1997, pp. 12–20.
[9] R. Sutton, A. Barto, Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, 1998.
[10] S.I. Reynolds, The stability of general discounted reinforcement learning with linear function approximation, in: UK Workshop on Compu. Intell., UKCI-02, 2002, pp. 139–146.
[11] H. Hoffmann, Peter, D.-H. Park, S. Schaal, Biologically-inspired dynamical systems for movement generation: automatic real-time goal adaptation and obstacle avoidance, in: Proc. IEEE Int. Conf. Robotics and Automation, ICRA, 2009, Kobe, Japan, 2009, pp. 2587–2592.
[12] A. Ijspeert, J. Nakanishi, P. Pastor, H. Hoffmann, S. Schaal, Learning nonlinear dynamical systems models, Neural Computation (2011) (submitted for publication).
[13] J. Baxter, P. Bartlet, Infinite-horizon policygradient estimation, Journal of Artificial Intelligence Research 15 (2001) 319–350.
[14] M. Tamosiunaite, T. Asfour, F. Wörgötter, Learning to reach by reinforcement learning using a receptive field based function approximation approach with continuous actions, Biological Cybernetics 100 (3) (2009) 249–260.

[15] M. Wiering, Convergence and divergence in standard and averaging reinforcement learning, in: Proc. 15th Eur. Conf. Machine Learning, ECML, 2004, Springer-Verlag, 2004, pp. 477–488.
[16] B. Nemec, M. Tamosiunaite, F. Wörgötter, A. Ude, Task adaptation through exploration and action sequencing, in: Proc. 9th IEEE-RAS Int. Conf. on Humanoid Robots, 2009, pp. 610–616.
[17] S. Schaal, J. Peters, J. Nakanishi, A. Ijspeert, Learning movement primitives, in: Int. Symp. on Robotics Res., ISRR2003, springer, 2004, p. 1805.
[18] J. Kober, J. Peters, Practical algorithms for motor primitives in robotics, IEEE Robotics and Automation Magazine 17 (2) (2010) 55–62.
[19] J. Kolter, A. Ng, Policy search via the signed derivative, in: Robotics: Science and Systems, RSS, 2009.
[20] J. Morimoto, C. Atkeson, Nonparametric representation of an approximated Poincare map for learning biped locomotion, Autonomous Robots 27 (2009) 131–144.
[21] G. Endo, J. Morimoto, T. Matsubara, J. Nakanishi, G. Cheng, Learning CPG-based biped locomotion with a policy gradient method: application to a humanoid robot, The International Journal of Robotics Research 27 (2) (2008) 213–228.
[22] J. Peters, S. Schaal, Reinforcement learning for parameterized motor primitives, in: Int. Joint Conf. on Neural Networks, 2006, pp. 73–80. http://www-clmc.usc.edu/publications/P/peters-IJCNN2006.pdf.
[23] T.H. Meyer, M. Eriksson, R.C. Maggio, Gradient estimation from irregularly spaced data sets, Journal of Mathematical Geology 33 (6) (2001) 693–717.
[24] S. Calinon, F. Guenter, A. Billard, On learning, representing, and generalizing a task in a humanoid robot, IEEE Transactions on Systems, Man, and Cybernetics 32 (2) (2007) 286–298.
[25] M. Hersch, F. Guenter, S. Calinon, A. Billard, Dynamical system modulation for robot learning via kinesthetic demonstrations, IEEE Transactions on Robotics 24 (6) (2008) 1463–1467.
[26] P. Pastor, H. Hoffmann, T. Asfour, S. Schaal, Learning and generalization of motor skills by learning from demonstration, in: Proc. IEEE Int. Conf. Robotics and Automation, ICRA, 2009, Kobe, Japan, 2009, pp. 763–768.

Minija Tamosiunaite has received the Ph.D. in Informatics in Vytautas Magnus University, Lithuania, in 1997. Her research interests include machine learning, biological signal analysis, application of learning methods in robotics.

Bojan Nemec is senior research associate at Department of Automatics, Biocybernetics and Robotics, Jozef Stefan Institute. He received the BS, M.Sc. and Ph.D. degrees from the University of Ljubljana in 1979, 1982 and 1988, respectively. In 1993, he spent his sabbatical leave at the Institute for Real-Time Computer Systems and Robotics, University of Karlsruhe. His research interests include robot control, robot simulation, sensor guided control, service robots and biomechanical measurements in sport. During 2002–2005, he was a task leader in the largest NAS European project EUROShoE. He has published over 100 conference and journal papers and is author of 1 patent, and co-author of a book.

Aleš Ude studied applied mathematics at the University of Ljubljana, Slovenia, and received his doctoral degree from the Faculty of Informatics, University of Karlsruhe, Germany. He was awarded the STA fellowship for postdoctoral studies in ERATO Kawato Dynamic Brain Project, Japan. He has been a visiting researcher at ATR Computational Neuroscience Laboratories, Kyoto, Japan, for a number of years and is still associated with this group. Currently he is a senior researcher at the Department of Automatics, Biocybernetics, and Robotics, Jozef Stefan Institute, Ljubljana, Slovenia. His research focuses on imitation and action learning, perception of human activity, humanoid robot vision, and humanoid cognition.

Florentin Wörgötter has studied Biology and Mathematics in Düsseldorf. He received his Ph.D. in 1988 in Essen working experimentally on the visual cortex before he turned to computational issues at the Caltech, USA (1988–1990). After 1990, he was a researcher at the University of Bochum concerned with experimental and computational neuroscience of the visual system. During 2000–2005, he had been Professor for Computational Neuroscience at the Psychology Department of the University of Stirling, Scotland where his interests strongly turned toward "Learning in Neurons". Since July 2005, he leads the Department for Computational Neuroscience at the Bernstein Center at the University of Göttingen, Physics III. His main research interest is information processing in closed-loop perception–action systems, which includes aspects of sensory processing, motor control and learning/plasticity. These approaches are tested in walking as well as driving robotic implementations. His group has developed the RunBot a fast and adaptive biped walking robot.