International Journal of Humanoid Robotics
Vol. 12, No. 3 (2015) 1550028 (18 pages)

Enhanced Policy Adaptation Through Directed Explorative Learning

Rok Vuga*, Bojan Nemec†, and Aleš Ude‡
Humanoid and Cognitive Robotics Lab,
Department of Automatics, Biocybernetics, and Robotics,
Jožef Stefan Institute, Ljubljana, Slovenia
**rok.vuga@ijs.si*
†*bojan.nemec@ijs.si*
‡*ales.ude@ijs.si*

Received (15 April 2015)
Accepted (7 May 205)
Published (9 June 2015)

In this paper, we propose an integrated policy learning framework that fuses iterative learning control (ILC) and reinforcement learning. Integration is accomplished at the exploration level of the reinforcement learning algorithm. The proposed algorithm combines fast convergence properties of iterative learning control and robustness of reinforcement learning. This way, the advantages of both approaches are retained while overcoming their respective limitations. The proposed approach was verified in simulation and in real robot experiments on three challenging motion optimization problems.

Keywords: Reinforcement learning; iterative learning control; dynamic movement primitives, movement optimiziation.

1. Introduction

Autonomous acquisition and refinement of skills is one of the major challenges of contemporary humanoid robotics. An often used paradigm to initiate knowledge transfer from humans to humanoid robots is programming by demonstration[1,2,3], where the demonstrated actions are used to seed the learning process. The initially obtained knowledge should then be adjusted and refined to account for different kinematic and dynamic capabilities of a human demonstrator and a target humanoid robot[4]. Adaptation process, where the robot modifies the available movements by exploring its action space in the neighborhood of previously acquired movements, is usually based on reinforcement learning (RL) techniques[5,6,7,8]. A major problem here is the huge search space that needs to be explored, which is affected not only by the high number of degrees of freedom of a humanoid robot, but also by the

‡Corresponding author.

underlying policy representation and the robot's environment. Recently proposed probabilistic RL algorithms like PI^2 proposed by Theodorou et al.[9] and PoWER[10] can scale to significantly more complex problems and reduce the number of tuning parameters. However, due to the high dimensionality of the parameter space, the adaptation speed of these algorithms is still low compared to humans, who are able to quickly adapt to new situations by exploiting previous experience and by generalizing from previously solved similar situations. Therefore, the issues of how to speed up the adaptation process and how to include knowledge from previous, similar situations are among the biggest challenges that need to be overcome to develop truly autonomous humanoid robots.

In this paper we focus on how to improve the adaptation speed by combining the ideas of Iterative Learning Control (ILC)[11] and reinforcement learning. In contrast to the standard approach, where Gaussian noise is used for parameter exploration, we propose to use ILC for the initial parameter exploration. The approach can be related to the covariance matrix adaptation methods, which follow a similar idea, i. e. to apply guided search in the parameter space in order to increase the adaptation speed. The adaptation based on ILC is much faster than what can be achieved by RL updates based on random exploration and results in skill knowledge that is much better adapted to the capabilities of the robot than knowledge originating in user demonstrations. Random exploration, however, is still needed to achieve the final fine tuning of the task. The proposed approach was evaluated in three robotics scenarios. The first is adaptation of a multi-DOF trajectory passing through a specified via-point. In the second scenarios, a robot holds a pendulum, which needs to follow a prescribed trajectory. In the third scenario we focus on adaptation of trajectory speed profile. The robot's task is to carry a cup of liquid along a specified path as fast as possible without spilling.

The paper is organized as follows. In Section 2 we outline the dynamic movement primitives framework along with the extension for speed profile adaptation. Next, in Section 3 we present the main contribution of this paper, the integrated application of ILC and reinforcement learning to accelerate policy iteration. Section 4 presents simulation and experimental evaluation of the proposed approach. Discussion and further steps are outlined in Section 5.

2. DMPs and speed profile

We start with initial user demonstration acquired either in joint or Cartesian space

$$\mathcal{G} = \{\boldsymbol{y}(k), \dot{\boldsymbol{y}}(k), \ddot{\boldsymbol{y}}(k), t(k)\}_{k=1}^{T}, \tag{1}$$

where $\boldsymbol{y}(k) \in \mathbb{R}^D$ are the corresponding coordinates and T is the number of samples on the demonstrated trajectory. In the following we use notation $\boldsymbol{y}(k), \dot{\boldsymbol{y}}(k), \ddot{\boldsymbol{y}}(k), x(k)$ to denote the measurements on the trajectory and $\boldsymbol{y}(x), \dot{\boldsymbol{y}}(x), \ddot{\boldsymbol{y}}(x), x$ to denote values obtained through integration of (2) – (11).

Next, we parameterize the given policy with dynamic movement primitives

(DMP)[12]. In the original formulation, the speed profile is embedded into the representation with the DMP parameters, which are normally obtained with regression. For discrete movements the dynamic system is specified with the following set of differential equations:

$$\tau \dot{z} = \alpha_z(\beta_z(g - y) - z) + f(x), \tag{2}$$

$$\tau \dot{y} = z, \tag{3}$$

$$f(x) = \frac{\sum_{i=1}^{N} w_i \Psi_i(x)}{\sum_{i=1}^{N} \Psi_i(x)} x, \tag{4}$$

$$\Psi_i(x) = \exp\left(-h_i \left(x - c_i\right)^2\right) \tag{5}$$

$$\tau \dot{x} = -\alpha_x x. \tag{6}$$

Here, y is the variable describing a one-dimensional trajectory, z is an auxiliary variable, and x is the phase variable. This way, y is defined as a response of a second order system, the dynamics of which are governed by constants α_z and β_z. In case of $\alpha = 4\beta$, the response of the system is critically damped, such that y converges from its starting value towards g, which defines the goal. The term $f(x)$ is used to define the more complex path we usually want y to take. It is defined as a weighted combination of non-linear Gaussian kernel functions and is superimposed onto the original second order response. The values of weights w_i thus define the shape of the trajectory y will take while converging towards the goal g.

To encode a multi-dimensional (either Cartesian or joint space) trajectory, each degree of freedom has to be represented with a separate dynamical system. The phase of the movement is common to all degrees of freedom; only one canonical system is thus needed. Consequently, every DOF has their own shape parameters w_i^d and goal g_d, but the duration τ is common for all.

Given a (possibly multi-dimensional) trajectory (1), it is necessary to calculate a set of parameters that encode it with the desired precision. By rewriting (2) – (3) as a second order differential equation

$$\tau^2 \ddot{y} + \alpha_z \tau \dot{y} - \alpha_z \beta_z (g - y) = f(x), \tag{7}$$

we obtain one linear equation in $\{w_i\}$ for each data point k in the sequence (1), i. e.

$$\tau^2 \ddot{y}_{\mathrm{d}}(t_k) + \alpha_z \tau \dot{y}_{\mathrm{d}}(t_k) - \alpha_z \beta_z (g - y_{\mathrm{d}}(t_k)) = \frac{\sum_{i=1}^{N} w_i \Psi_i(x(t_k))}{\sum_{i=1}^{N} \Psi_i(x(t_k))} x(t_k), \tag{8}$$

where $x(t_k)$ are obtained by integrating (6), which results in $x(t_k) = \exp(-\alpha_x t_k / \tau)$. Weights w_i are thus estimated by solving overdetermined equation system (8) for all $k = 1, \ldots, T$. Finally, we set $\tau = t_T$ and $g_d = y_d(T)$. Repeating the process for every degree of freedom d, we obtain the parameters of the DMP corresponding to the training trajectory (1).

In the above formulation the speed profile of the motion is encoded along with the shape of the path by the free parameters w. The speed of the trajectory can only be adapted uniformly, by choosing the desired duration τ. Here, we present a modified formulation which allows for non-uniform changes to the encoded speed profile[13]. This way, we can modulate the speed of the motion without affecting its shape. To achieve this, we include an additional, phase-dependent temporal scaling factor into the DMP equations. The original DMP equations for discrete movements are extended with the additional temporal scaling factor ν as follows:

$$\tau \dot{z} = \nu(x)(\alpha_z(\beta_z(g-y)-z)+f(x)), \tag{9}$$

$$\tau \dot{y} = \nu(x) z, \tag{10}$$

$$\tau \dot{x} = -\nu(x)\alpha_x x. \tag{11}$$

Note that we still keep the constant temporal scaling factor τ to preserve the ease of uniform time scaling. Smaller ν results in a slower motion while larger ν increases the speed of motion. The phase-dependent factor ν is parameterized as a weighted combination of M radial basis functions

$$\nu(x) = 1 + \frac{\sum_{j=1}^{M} v_j \Psi_j(x)}{\sum_{j=1}^{M} \Psi_j(x)} = 1 + \boldsymbol{v}^T \boldsymbol{\Psi}(x), \tag{12}$$

$$\boldsymbol{v} = \left[v_1, \ldots, v_M \right]^T, \tag{13}$$

$$\boldsymbol{\Psi}(x) = \frac{1}{\sum_{j=1}^{M} \Psi_j(x)} \left[\Psi_1(x), \ldots, \Psi_M(x) \right]^T. \tag{14}$$

Just like τ, the scaling factor ν should be the same for all of the robot's degrees of freedom in order to keep them synchronized. The modulation of kernel weights v_j allows for a nonuniform modulation of the speed profile in the same way as the modulation of weights w_i in (5) allows for modulation of the shape of the desired trajectory. Initially, the weights v_j are set to zero, so that $\nu(x) = 1, \ \forall x$. This corresponds to the movement at the original speed. Thus no re-training of the initial DMP (2) – (6) is needed.

In the above formulation factor $\nu(x)$ directly affects the speed of the movement. Lower values of $\nu(x)$ correspond to slower movements and higher values to faster movements, respectively. Note the differences between the proposed approach and our previous work[13]. With the new definition (12), the speed of the demonstrated motion is preserved by setting all weights v_j to 0. We also inverted relationship between $\nu(x)$ and τ. Nemec et al.[13] achieved slowing down by increasing the value of ν, which caused near singularity behavior at parts of the phase corresponding to low speed.

3. Reinforcement learning with directed exploration

Policy search algorithms, the type of reinforcement learning most commonly used in robotics, follow the general recipe:

(1) Perform exploratory trials with the current policy,
(2) Analyze the collected rewards,
(3) Improve policy and repeat at step 1.

In essence, different algorithms vary in how steps 2 and 3 are implemented. Drawing from the normal distribution with mean at the current policy parameters is typically used to generate random exploration. A class of policy gradient algorithms exploits the obtained knowledge to estimate the gradient of the reward function with respect to the policy parameters. They have strict convergence properties, but in general suffer from low adaptation speed[5]. Recently, a novel class of methods gained popularity, which return directly the update as a weighted combination of exploration policies[10,9,14]. These algorithms employ some notion of "eliteness" of exploratory trials, from which the update is computed, while the "non-elite" trials are rejected[15].

Note that in gradient based methods, performing random exploration makes sense, as sampling of the local vicinity of the policy is needed to obtain a good gradient estimate. In the case of ranking samples by eliteness, however, every step made in the wrong direction is rejected by the algorithm. Obviously, the mapping between policy and reward is unknown in general and random exploration is a safe way to achieve the optimal desired policy. However, in many cases, some estimate of this mapping can be obtained, either from prior knowledge, user input, reward function design, etc. Therefore, we propose to exploit this knowledge as much as possible by way of using policies inferred from exploration trials in the earlier stages of learning. Random exploration can still be used in combination to preserve the generality of the RL method.

We propose to use Iterative Learning Control (ILC)[16,11] to provide directed exploration strategy in the first stages of learning. In every iteration, the obtained exploration trajectory is executed and its cost collected. Only this strategy ceases to yield reduction of the cost, the policy is further improved with random exploration.

The basic idea of iterative learning control is that information about the tracking error can be used to improve performance in the next repetition of the same trajectory. The general form of ILC is to update the control signal as follows

$$u(k, j+1) = \kappa(u(k, j) + \eta e(k+1, j)), \tag{15}$$

where u is the control signal, k denotes the k-th time sample, j denotes iteration, and κ and η are the learning parameters. ILC is distinguished from simple feedback control by the prediction of the error $e(k+1, j)$, which serves to anticipate the error caused by the action taken at the k-th time step. ILC modifies the control input in the next iteration based on the control input and error in the previous iteration. In the context of policy learning, the error is associated with the policy cost and u is parameterized. In our case, the policy is encoded by DMP parameters defining the shape profile $\boldsymbol{w}$ or speed profile $\boldsymbol{v}$, which is mapped to the control signal $\mathbf{u} = [u(1), \ldots, u(N)]$ (see also Section 3.2). In order to keep consistent notation

for both shape profile parameters $\boldsymbol{w}$ and speed profile parameters $\boldsymbol{v}$, we denote them with $\boldsymbol{\theta}$. $\boldsymbol{\theta}$ thus denotes shape profile parameters in tasks subjected to the trajectory adaptation or speed profile parameters in tasks subjected to the speed adaptation, respectively. Mapping from DMP parameters $\boldsymbol{\theta}$ to the control signal $\mathbf{u}$ is accomplished with DMP integration (9), (10), (11). Similarly, mapping from time or phase dependant signal $\mathbf{u}$ to $\boldsymbol{\theta}$ is accomplished by solving (8) in least-square sense[17,12].

The main requirement for the application of ILC is to carefully design the error signal e in (15) and to tune the learning parameters κ and η. Note that ILC works in a controller-like fashion: at every iteration it makes a step in the direction given by the sign of the obtained error. This is hard to achieve for problems where a balance needs to be found between opposing criteria. Another example are systems with large and unknown time-delays. In such cases, reinforcement learning with random exploration can find a better solution. Therefore, once we observe that ILC does not improve the cost anymore, a switch to step 3 is made.

Algorithm 1 summarizes the proposed approach, where we applied reinforcement learning algorithm PI[29] for parameter update. This way, a significant increase in the speed of convergence can be achieved without sacrificing convergence properties.

The PI^2 implementation in Algorithm 1 contains two subtle adjustments of the original implementation as proposed by Theodorou et al.[9]. First, we omitted the quadratic control cost term from calculation of S, as it is not significant in our case. Penalizing high values of policy parameters may slow the trajectory down, which is the opposite of what we want to achieve. Furthermore, in the original formulation the update rule was defined as $\Delta\boldsymbol{\theta}_n = \sum_{k=1}^{N_b} P_k^b(n)\mathbf{M}_k^b(n)\boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon}$ is current exploration of the policy parameters, i.e. $\boldsymbol{\epsilon} = \boldsymbol{\theta} - \boldsymbol{\theta}_i$. Instead, we replaced $\boldsymbol{\epsilon}$ with the exploration of the current parameters with respect to the k-th best known policy from the whole history of trials. This is needed in order to perform the update using data gathered during past trials: P_b^k and $\mathbf{M}_b^k$ have been collected during executions of policies $\boldsymbol{\theta}_b^k$, which were different than the current policy $\boldsymbol{\theta}_i$.

3.1. *Trajectory adaptation using ILC*

Trajectory adaptation using ILC is rather straightforward. Error function $e(k, j)$ used to calculated the next control input $u(k, j+1)$ (15) is usually associated with the temporary difference between the desired and obtained position or difference between the desired and measured forces and torques. A vast range of problems can be solved in such a way, such as the via-point problem (Section 4.1), trajectory tracking[18], (Section 4.2), peg in hole[19], wiping[20], visual servoing[21], etc. However, ILC can not be used to learn policies with the delayed reward function, such as ball throwing[22], playing 'ball in a cup' game[23,24], etc.

given	parameterized policy $\boldsymbol{\theta}$ basis functions $\boldsymbol{\Psi}$ for parameterization intermediate cost functions $r(n), n = 1, \ldots, N$ terminal cost function $r(N+1)$ ILC error function $e(x_n)$ learning parameters $\boldsymbol{\Sigma}$, N_b, κ, η, δ_s initial approximation $\boldsymbol{\theta}_1$
learn	$use\_random = false$ **for** $i = 1...$ max iterations: calculate exploration step: **if** C decreasing **and** $use\_random = false$ calculate trajectory using ILC **for** $n = 1...N$ $u(x_n, i) = \kappa(u(x_n, i-1) + \eta e(x_n + \delta_{x_n}, i-1))$ find $\boldsymbol{\theta}$ which parameterizes $\boldsymbol{u_i}$ with $\boldsymbol{\Psi}$ **else** use random exploration policy: draw $\boldsymbol{\theta}$ from $\mathcal{N}(\boldsymbol{\theta}_i, \boldsymbol{\Sigma})$ $use\_random = true$ perform exploration experiment using $\boldsymbol{\theta}$ and collect costs $r(n)$, $n = 1, \ldots, N+1$ calculate total cost $C = \sum_{n=1}^{N+1} r(n)$ sort all past trials by C, so that $\boldsymbol{\theta}_k^b$ is k-th best policy and r_k^b its return, i.e. C **if** $i < N_b$ $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_1$ **else** update policy using algorithm PI^2: **for** $n = 1...N$ **for** $k = 1...N_b$ $S_k^b(n) = r_k^b(N+1) + \sum_{j=n}^{N} r_k^b(j)$ $P_k^b(n) = \frac{\exp(-S_k^b(n)/\lambda)}{\sum_{m=1}^{N_b} \exp(-S_m^b(n)/\lambda)}$ $M_k^b(n) = \frac{\boldsymbol{\Psi}(n)_k^b \boldsymbol{\Psi}(n)_k^{b\,T}}{\boldsymbol{\Psi}(n)_k^{b\,T} \boldsymbol{\Psi}(n)_k^b}$ $\delta\boldsymbol{\theta}_n = \sum_{k=1}^{N_b} P_k^b(n) M_k^b(n)(\boldsymbol{\theta}_k - \boldsymbol{\theta}_i)$ **for** $j = 1...M$ $\delta\theta_j = \frac{\sum_{n=1}^{N}(N+1-n)\Psi_j \Delta\theta_{n,j}}{\sum_{n=1}^{N} \Psi_j(n)(N+1-n)}$ $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + [\delta\theta_1, ..., \delta\theta_M]^T$ until convergence

Table 1. PI^2 algorithm augmented with ILC directed exploration.

3.2. *Speed profile adaptation using ILC*

The goal of modifying the speed profile is to accelerate the task execution without degrading the overall performance of the task. A similar idea was exploited by Abu-

Dakka et al.[19] in order to gradually increase the speed of an assembly task until contact forces remained within the prescribed tolerances. The idea is simple: increase the speed of task execution until some essential task constrains are violated. These task constrains together with policy execution time determine the error function $e(x)$, used for parameter update with ILC, as well as the cost function used for RL. However, it is important to note that the two criteria are not the same. RL algorithms require unsigned value to determine the cost of an experiment, while ILC operates with signed error function.

A suitable error function for ILC to adapt the speed of motion is given by

$$e(x_n) = \xi_\nu(\nu_{\max} - \nu(x_n)) - \xi_d b(x_n), \tag{16}$$

where ν is the previously defined temporal scaling function, b is a scalar function quantifying deviations from the pre-specified task constraints, $\nu_{\max} > 0$ is the upper bound for the temporal scaling factor, x_n is the phase along the trajectory at step n, $x_n = x(t_n)$ is the phase, and ξ_ν, ξ_d are the corresponding weighting factors. Function b describing deviations from task constraints might be anything from a norm of excessive forces and torques[19] to a signal indicating the spilling of liquid as in this paper.

As noted by Nemec et al.[13], standard ILC approach cannot be applied to the trajectory speed learning in time domain, since one of the assumptions of ILC is that each trial has the same number of samples[11]. Therefore, Nemec et al.[13] implemented ILC in phase domain, where the phase signal was anticipated for a fixed phase offset. Since mapping from the phase to the displacement is not linear, this choice is not optimal. Instead, we propose the following ILC-type algorithm to realize the learning of the temporal scaling function

$$\nu(x_n, j+1) = \kappa(\nu(x_n, j) + \eta e(x_n + \delta_{x_n}, j), \tag{17}$$

where x_n denotes the phase, k is the sampling index, j is the learning iteration index, and κ and η are the manually selected ILC gains. The error signal $e(x_n + \delta_{x_n}, j)$ is used to anticipate the deviation from the desired behavior, with δ_{x_n} denoting the magnitude of the step in the phase domain from which the prediction is calculated, and can be calculated from (11) as follows

$$\delta_{x_n} = -\frac{\alpha_x \nu_j(x)}{\tau} x \delta_t, \tag{18}$$

where δ_t is the user chosen parameter which corresponds to the constant prediction step in the time domain.

4. Experimental evaluation

4.1. *Multi-DOF via-point problem*

The first evaluation considers optimal performance of a via-point task. This experiment is identical to the experiment described by Theodorou et al.[9], where a

multi-DOF planar robot is tasked with moving from a fully extended pose to a bent configuration resembling a semi circle. The robot should move from the starting to the final pose in such a way that the cost function

$$r(t) = \frac{\sum_{i=1}^{N_{dof}} (d+1-i)(K_a \ddot{q}(t)_i^2 + K_u \boldsymbol{\theta}_i^T \boldsymbol{\theta}_i)}{\sum_{i=1}^{N_{dof}} (d+1-i)}, \tag{19}$$

$$r(t_{300ms}) = K_{vp} \left((x_{vp} - x(t_{300ms}))^2 + (y_{vp} - y(t_{300ms}))^2 \right) \tag{20}$$

is minimized. Here, every sample of the trajectory is penalized based on the squared acceleration of the trajectory as well as the magnitude of the policy parameters. The weighing term $d + 1 - i$ penalizes DOFs proximal to the robot base more than those distal to the base. Additionally, the sample at $t = 300\ ms$ is penalized based on the euclidean distance between the robot end effector and the desired via-point located at $x_{vp} = 0.5$, $y_{vp} = 0.5$. The constants $K_a = 0.1$, $K_u = 0.5$ and $K_{vp} = 1 \times 10^8$ regulate the relative importance of each criterion. We used the same values as reported by Theodorou et al.[9].

This way, the task of the robot is to move from the starting to the final position in such a way, that the tip of the robot passes through the via-point $(0.5, 0.5)$, while keeping the control parameters low in value.

The robot consists of 10 segments, each of which is 0.1 m long. The motion is controlled in the joint space and encoded with Dynamic Movement Primitives (Section 2). The number of Gaussian kernel functions used to represent the shape of the trajectory was 5 for each of the degrees of freedom. Since the robot has 10 joints, the total number of policy parameters needed to be learned is 50.

The error function for the ILC directed exploration was defined in the Cartesian space as

$$\boldsymbol{e}(t) = (\boldsymbol{x}_{vp} - \boldsymbol{x}(t)) \exp\left(-\frac{(t-0.3)^2}{\sigma} \right), \tag{21}$$

where $\boldsymbol{x} = (x, y)$ represents Cartesian position of the tip of the robot and $\boldsymbol{x}_{vp} = (x_{vp}, y_{vp})$ is the position of the via-point. Simply, this type of error function 'pulls' the end-effector trajectory towards the via-point located at $(0.5, 0.5)$. However, since we only wish for the robot to pass close to the via-point at the specified time, the error is weighted with a Gaussian function which peaks at $t = 300$ ms. The width of the weighing function is defined by $\sigma = 0.05$. Parameters of the ILC update (15) were set to $\kappa = 1$, $\eta = 0.4$.

With the above choice the exploration trajectory was obtained in the Cartesian space. The corresponding joint space trajectory was calculated using Jacobian pseudo-inverse based inverse kinematics. The obtained joint space trajectory was then encoded with DMPs and used in the policy update process, as detailed by Algorithm 1.

Figure 1 shows the results of the learning. The left graph shows the initial robot trajectory. The robot moves from the fully extended to the semi-circular

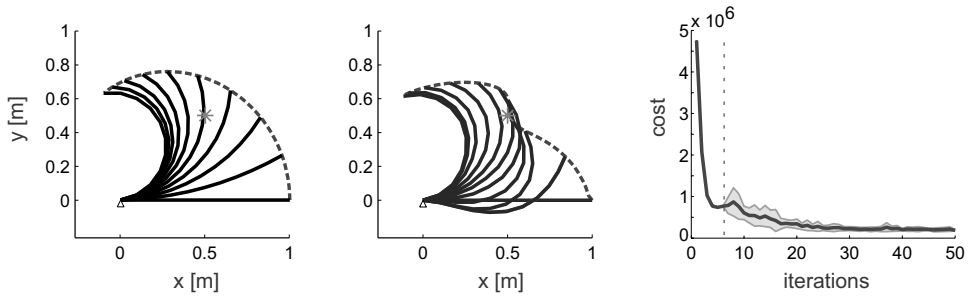

Fig. 1. Learning the via-point task. Left: initial robot trajectory. Middle: robot trajectory after 50 learning iterations. The star indicates the specified via-point. Right: convergence of cost. The shaded area shows standard deviation of 15 learning series. The dashed line marks the iteration where the switch to random exploration were made.

configuration, with the end-effector tracing an arc. The middle graph shows the situation after the learning process converged. Note that the tip of the robot passes through the desired via-point. The plots were obtained using Planar Manipulation Toolbox[25]. Learning convergence is detailed in the right graph. As you can see, the ILC-directed exploration dramatically reduced the costs in the first 6 iterations. After these initial steps, ILC was not able to further improve the policy since it could not account for additional joint related constraints imposed by (19). Therefore, as the sum of the trajectory cost stopped decreasing, the learning switched to random exploration, which further improved the solution, albeit at a much slower rate.

4.2. *Trajectory adaptation*

The next example demonstrates the performance of the proposed learning scheme in a challenging trajectory adaptation scenario. A robot holds a pendulum which hangs from its wrist. The pendulum has to follow a prescribed trajectory reference; the robot thus needs to find a solution which accounts for its own dynamics as well as for the dynamics of the pendulum. We performed this experiment in simulated environment, where the Kuka LWR robot dynamics and pendulum dynamics were calculated at 1Khz sampling rate using Matlab/Simulink/Simechanics programming environment. The pendulum mass and length were 0.2 Kg and 0.4 m, respectively. The pendulum had to follow a square shaped reference trajectory with side length of 0.3 m in XY plane with constant speed of 0.6 m/s and to stop at the final position. The duration of the reference trajectory was 2 seconds. After that, the pendulum had to remain still for another second. Thus, the required accelerations were infinite in all four corners of the reference square, which made this task very challenging. We simulated a compliant robot with Cartesian stiffness of 500 N/m in X and Y direction. Such a robot is not able to follow sharp edges, which makes the task even harder. The robot had to learn a trajectory which resulted in minimal error between the position of the pendulum and the square shaped reference trajectory. Figure 2

shows an instance of the simulated scene.

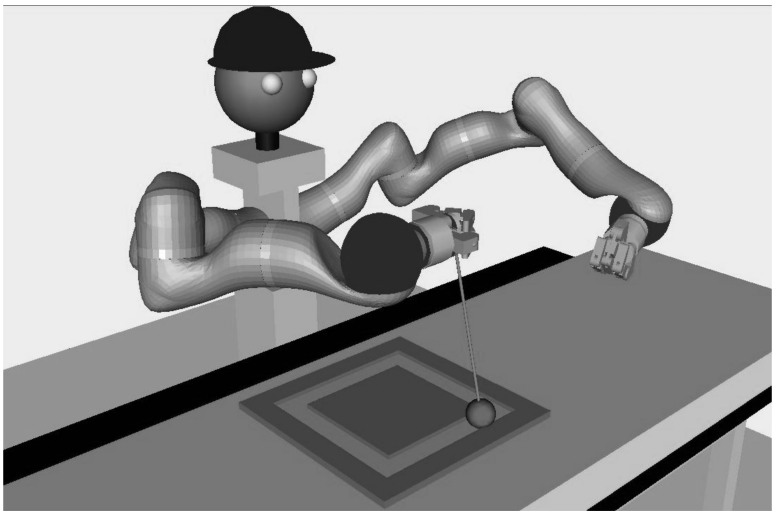

Fig. 2. Simulated scene, where the pendulum attached to the KUKA LWR robot has to follow the square with constant speed.

The ILC based exploration was obtained in the form

$$\mathbf{p}_r(k,j) = \kappa \mathbf{p}_r(k,j-1) + \nu \mathbf{e}(k+1,j-1)$$
$$\mathbf{e}(k,j) = \mathbf{p}_d(k) - \mathbf{p}_p(k,j),$$

where $\mathbf{p}_r \in \mathbb{R}^2$ and $\mathbf{p}_p \in \mathbb{R}^2$ are the commanded robot position and measured pendulum position in X and Y plane, respectively. $\mathbf{e}$ is the tracking error, and $\mathbf{p}_d$ denotes the desired pendulum position. Gains κ and ν were set to 1 and 0.5, respectively. Initially, the desired robot position was set to the desired pendulum position, $\mathbf{p}_d(k,1) = \mathbf{p}_r(k),\ \forall k$. Cost function was defined as $C = \sum_{k=1}^{T} \|\mathbf{e}(k)\|$, where T denotes the duration of the experiment in discrete time samples. The trajectory was updated at the rate of 0.01 sec. The robot reference was parameterized with the Gaussian kernel functions using (4) and fed as exploration to the PI^2 policy update as shown in Algorithm 1. After approx 25 ILC iterations the cost C was not decreasing any more, therefore the algorithm continued with random exploration, $\boldsymbol{\theta} = \mathcal{N}(\boldsymbol{\theta}_i, 0.003)$. Overall, 300 learning iterations were performed. Figure 3 shows convergence of the cost. We can notice very fast initial convergence using ILC based exploration. After switching to the random exploration, PI^2 algorithm further improved the policy. However, the cost drop was much slower, which clearly shows the benefit of the ILC based exploration.

Figure 4 shows the initial trajectory, the trajectory learned with ILC based exploration, and the final trajectory obtained after random exploration was used. Dotted, dashed and solid lines denote the learned robot reference trajectory $\mathbf{p}_r$,

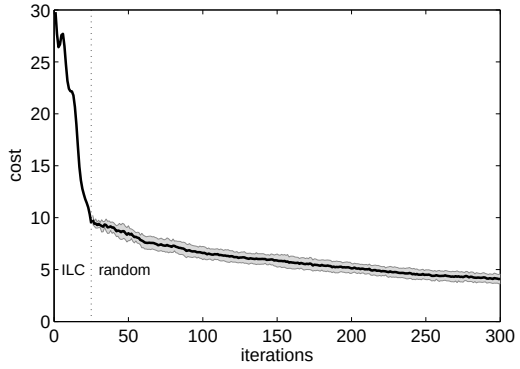

Fig. 3. Cost convergence for the pendulum trajectory adaptation. Graph left to the dotted vertical line corresponds to the ILC exploration, Shaded region denotes standard deviation of 10 simulated experiments

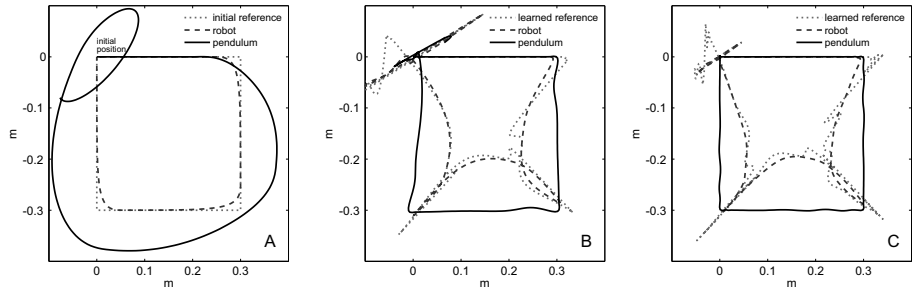

Fig. 4. Robot reference trajectory, robot tool trajectory and pendulum trajectory in XY plane. Left (A) graph shows the initial trajectories, center graph (B) the learned trajectories with ILC exploration and right graph (C) the trajectories refined with the random exploration

robot tool center point trajectory, and pendulum trajectory $\mathbf{p}_p$, respectively. Note that in this experiment the learning algorithm had to account for both pendulum and the robot dynamics while learning the required robot reference.

4.3. *Speed profile adaptation*

In the third experiment, the proposed methodology was used for speeding up the task of delivering a glass full of liquid from a tray onto a table. The goal of adaptation was to speed up the execution time as much as possible, without spilling the liquid. This is an example of a process, for which it would be very difficult to find a model-based solution. Our experimental setup consisted of the Kuka LWR arm with FRI interface and a three finger Barret hand, shown in Fig. 6. The desired path of the robot was demonstrated using kinesthetic guiding. The cup was equipped with a

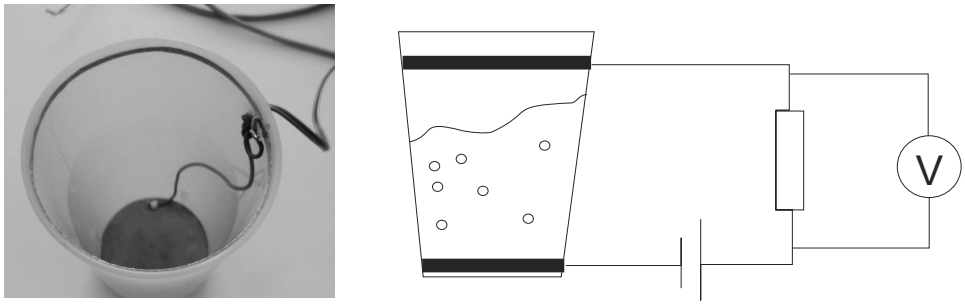

Fig. 5. Sensory system used to detect spilling. The circuit was powered with a 9 volt battery and voltage measured on a 10 MΩ resistor. When the water closes the circuit, the resistor takes an overwhelming part of the voltage drop and the reading increases from 0 to 9 V.

resistance sensor to detect spilling (see Figure 5). Whenever the level of the liquid reached the conductive ring at the edge of the cup, an increase in voltage was detected.

The intermediate cost function was defined as

$$r(n) = \begin{cases} 0 & \text{if } U(n) < U_0 \\ \gamma & \text{if } U(n) \geq U_0 \end{cases}, \tag{22}$$

where $r(n)$ denotes an intermediate cost at sample n, $U(n)$ the current voltage, U_0 the threshold voltage, which signifies spilling, and γ is a positive constant. In our experiment we used $\gamma = 10$, $U_0 = 5V$. The terminal cost was defined as duration of the trajectory in seconds multiplied by scaling factor 50.

At the beginning of learining, exploration policies were calculated and executed using ILC. Error function (16) was finalized by monitoring the spilling sensor voltage

$$b(x_n) = U(n) - U_0, \tag{23}$$

where $U(n)$ denotes the sensor's output voltage and U_0 denotes the threshold voltage. For $U(n)$ smaller than U_0, signifying no spilling, b will be negative and will cause the trajectory at the corresponding phase to speed up. Conversely, $U(n)$ higher than U_0 slows down the motion. The speed profile for the next iteration ν_{j+1} was obtained by (17), with constants chosen as $\kappa = 1$ and $\eta = 0.01$. The policy update $\Delta\boldsymbol{\theta}_{j+1}$ was then obtained by parameterizing ν_{j+1} as per (12) and calculating the difference to the current policy provided by reinforcement learning as explained in Section 3.

As shown in Fig. 7, the policy improves quickly. However, the main drawback of ILC is that it cannot efficiently find a balance between opposing effects. Imagine that during an episode, no liquid spilling is detected. Therefore, for the next learning iteration, ILC will generate a faster trajectory. As a result, some liquid will spill and in the next iteration, the movement will be slower again, catching the learning process in an oscillatory cycle. This is due to the controller-like nature of ILC - it tries to reduce the error signal to zero, which is sometimes not possible to achieve.

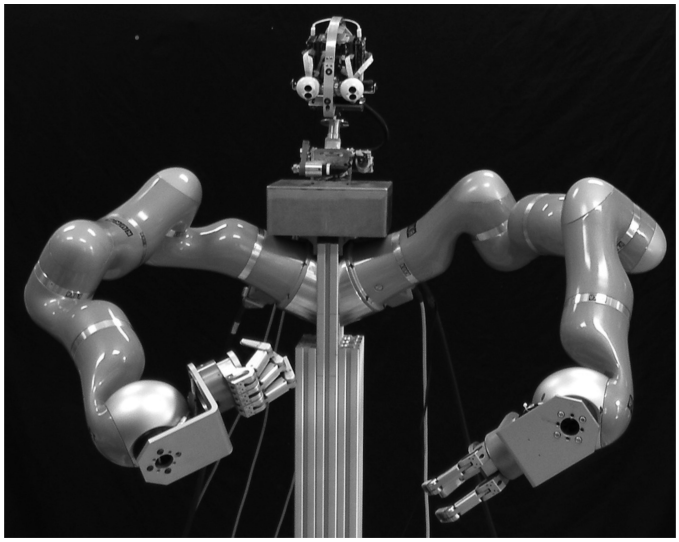

Fig. 6. Upper body humanoid system used in the experiments.

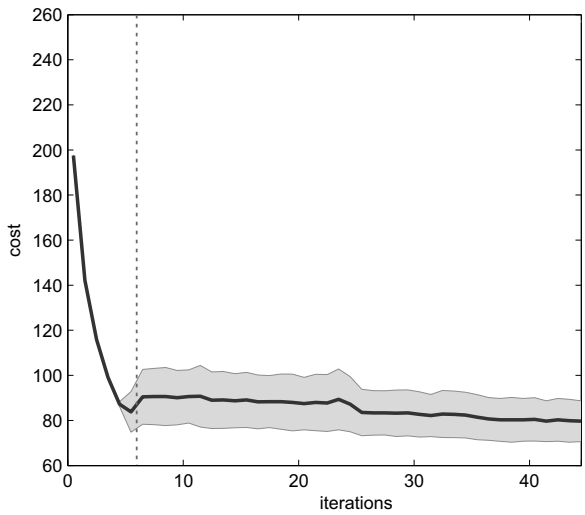

Fig. 7. Convergence of the learning process. The shaded area shows standard deviation of 8 learning trials. The dashed line marks iteration six, where the ILC based exploration stopped converging. The trials after this mark were performed using random exploration.

Thus, after the 6th iteration, the cost has been observed to stop converging. Random exploration is then used to perform fine tuning of the policy.

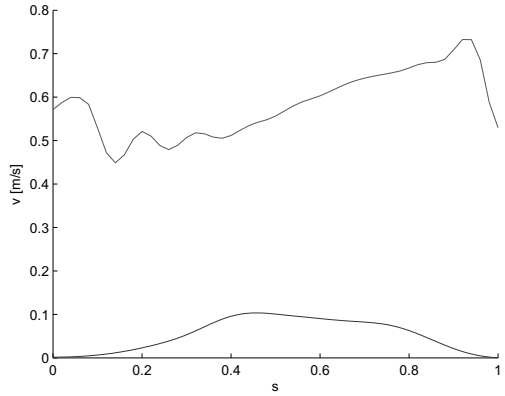

Fig. 8. Result of the learning process. The bottom graph shows speed of the initially demonstrated task. The upper line shows an example of a learned speed profile after 45 learning trials. Note that the profiles are plotted against normalized arc length s, not time.

Fig. 9. Robot moves the cup from the tray onto the table. The path was obtained using kinesthetic guiding and the speed profile (shown in Fig. 8) learned using our novel approach.

5. Conclusions

In this paper we presented a novel method for autonomous learning of control policies. Reinforcement learning methods typically use random exploration to obtain information about reward fluctuation in the local vicinity of the policy. We proposed to supplement random exploration by using iterative learning control (ILC), which

exploits a reference signal to achieve faster convergence. Since ILC's convergence properties are not as strong as those of RL, the algorithm stops converging at some point, and the exploration switches to the standard random exploration. This way we retain the best of both worlds, the initial fast convergence of ILC and fine tuning achievable by RL. The drawback of the proposed approach is that the ILC based exploration steps introduces a few additional open parameters, which require tuning for optimal performance. Note also that our algforithm is not applicable to every problem that reinforcement learning can solve; unlike RL, ILC requires a reference signal. In our future work we will therefore exploit self tuning of ILC parameters, which might contribute to the improved robustness and convergence of the ILC directed exploration.

The proposed approach was verified in simulation and in real robot experiment. We showed that it can be successfully applied to the trajectory adaptation tasks, which comprise simultaneous path and velocity adaptation and to tasks where only velocity adaptation is required.

Acknowledgements

Research leading to these results was supported by the European Community's Seventh Framework Programme FP7/2007-2013 (Specific Programme Cooperation, Theme 3, Information and Communication Technologies) grant agreement no. 270273, Xperience.

References

1. S. Schaal, "Is imitation learning the route to humanoid robots?" *Trends in Cognitive Sciences*, vol. 3, no. 6, pp. 233–242, 1999.
2. R. Dillmann, "Teaching and learning of robot tasks via observation of human performance," *Robotics and Autonomous Systems*, vol. 47, no. 2-3, pp. 109–116, 2004.
3. R. Palm and B. Iliev, "Programming-by-demonstration and adaptation of robot skills by fuzzy time modeling," *International Journal of Humanoid Robotics*, vol. 11, no. 1, pp. 1 450 009–1–1 450 009–28, 2014.
4. A. Billard, S. Calinon, R. Dillmann, and S. Schaal, "Robot programming by demonstration," in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Springer Berlin Heidelberg, 2008, pp. 1371–1394.
5. J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural Networks*, vol. 21, pp. 682–697, 2008.
6. M. Tamosiunaite, B. Nemec, A. Ude, and F. Wörgötter, "Learning to pour with a robot arm combining goal and shape learning for dynamic movement primitives," *Robotics and Autonomous Systems*, vol. 59, no. 11, pp. 910–922, 2011.
7. J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *International Journal of Robotic Research*, vol. 32, no. 11, pp. 1238–1274, 2013.
8. P. Wawrzynski, "Reinforcement learning with experience replay for model-free humanoid walking optimization," *International Journal of Humanoid Robotics*, vol. 11, no. 3, pp. 1 450 024–1–1 450 024–21, 2014.
9. E. A. Theodorou, J. Buchli, and S. Schaal, "A generalized path integral control ap-

proach to reinforcement learning," *Journal of Machine Learning Research*, vol. 11, pp. 3137–3181, 2010.
10. J. Kober and J. Peters, "Policy search for motor primitives in robotics," in *Advances in Neural Information Processing Systems 21 (NIPS)*, Vancouver, B. C., Canada, 2008, pp. 852–859.
11. D. Bristow, M. Tharayil, and A. Alleyne, "A survey of iterative learning control," *IEEE Control Systems*, vol. 26, no. 3, pp. 96–114, 2006.
12. A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: Learning attractor models for motor behaviors," *Neural Computation*, vol. 25, no. 2, pp. 328–373, 2013.
13. B. Nemec, A. Gams, and A. Ude, "Velocity adaptation for self-improvement of skills learned from user demonstrations," in *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, Atlanta, Georgia, 2013, pp. 423–428.
14. V. Heidrich-Meisner and C. Igel, "Neuroevolution strategies for episodic reinforcement learning," *Journal of Algorithms*, vol. 64, no. 4, pp. 152–168, 2009.
15. F. Stulp and O. Sigaud, "Path integral policy improvement with covariance matrix adaptation," in *29th International Conference on Machine Learning (ICML)*, Edinburgh, UK, 2012, pp. 281–288.
16. K. L. Moore, Y. Chen, and H.-S. Ahn, "Iterative learning control: A tutorial and big picture view," in *IEEE Conference on Decision and Control*, San Diego, CA, 2006, pp. 2352–2357.
17. A. Ude, A. Gams, T. Asfour, and J. Morimoto, "Task-specific generalization of discrete and periodic dynamic movement primitives," *IEEE Transactions on Robotics*, vol. 26, no. 5, pp. 800–815, 2010.
18. C. J. Ostafew, A. P. Schoellig, and T. D. Barfoot, "Visual teach and repeat, repeat, repeat: Iterative learning control to improve mobile robot path tracking in challenging outdoor environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Tokyo, Japan, 2013.
19. F. J. Abu-Dakka, B. Nemec, J. A. Jørgensen, T. R. Savarimuthu, N. Krüger, and A. Ude, "Adaptation of manipulation skills in physical contact with the environment to reference force profiles," *Autonomous Robots*, 2015, doi: 10.1007/s10514-015-9435-2.
20. A. Gams, T. Petrič, B. Nemec, and A. Ude, "Learning and adaptation of periodic motion primitives based on force feedback and human coaching interaction," in *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, Madrid, Spain, 2014, pp. 166–171.
21. P. Jiang, L. Bamforth, Z. Feng, J. E. F. Baruch, and Y.-Q. Chen, "Indirect iterative learning control for a discrete visual servo without a camera-robot model," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 37, no. 4, pp. 863–876, 2007.
22. B. Nemec, R. Vuga, and A. Ude, "Exploiting previous experience to constrain robot sensorimotor learning," in *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, Bled, Slovenia, 2011, pp. 727–723.
23. J. Kober and J. Peters, *Learning Motor Skills: From Algorithms to Robot Experiments*. Heidelberg: Springer-Verlag, 2014.
24. B. Nemec and A. Ude, "Reinforcement learning of ball-in-a-cup playing robot," in *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Karon Beach, Thailand, 2011, pp. 2682–2987.
25. L. Žlajpah, "Integrated environment for modelling, simulation and control design for robotic manipulators," *Journal of Intelligent and Robotic Systems*, vol. 32, no. 2, pp. 219–234, 2001.

Rok Vuga is a Ph.D. student with the Humanoid and Cognitive Robotics Lab, Dept. of Automatics, Biocybernetics and Robotics, Jožef Stefan Institute, Ljubljana. He graduated from the Faculty of Electrical Engineering, University of Ljubljana in 2011, with work on reinforcement learning in constrained dimensionality for robotics. His research interests among others include learning by demonstration, reinforcement learning, and balancing for humanoids.

Bojan Nemec is Senior Research Associate at Humanoid and Cognitive Robotics Lab, Department of Automatics, Biocybernetics and Robotics, Jožef Stefan Institute in Ljubljana, Slovenia. He received BSc, MSc and PhD from the University of Ljubljana. He spent his sabbatical leave at the Institute for Real-Time Computer Systems and Robotics, University of Karlsruhe, Germany. His research interests include robot control, robot learning, sensor-guided control, service robots and biomechanical measurements in sport.

Aleš Ude received the Diploma degree in Applied Mathematics from the University of Ljubljana, Slovenia, and the Ph.D. degree from the Faculty of Informatics, University of Karlsruhe, Germany. He is currently the head of Department of Automatics, Biocybernetics, and Robotics, at Jožef Stefan Institute, Ljubljana and is also associated with the ATR Computational Neuroscience Laboratories, Kyoto, Japan. His research interests include autonomous robot learning, imitation learning, humanoid robot vision, humanoid cognition, and humanoid robotics in general.