2011 11th IEEE-RAS International Conference on Humanoid Robots
Bled, Slovenia, October 26-28, 2011

Exploiting Previous Experience to Constrain Robot Sensorimotor Learning

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

***Abstract*—A truly autonomous robot should be able to generalize known actions to new situations and to autonomously refine its knowledge base. In this paper we present a three stage approach to the problem of expanding and refining the database of sensorimotor knowledge. The first stage is based on the generalization of previously trained movements associated with a specific task, which results in a first approximation of a suitable control policy in a new situation. The second stage applies learning on the manifold defined by the previously acquired training data, which results in a learning problem of reduced dimensionality. The final tuning of the desired control policy is accomplished by learning in the full state space, where the dimensionality of the problem is much higher. The assumption is that the first two steps already provide a good initial estimate for the optimal control policy so that this final step only locally refines the parameters learned in the first two steps. This significantly reduces the number of test trials needed by standard reinforcement learning techniques. The proposed approach was evaluated in simulation as well as on the real robot in a ball throwing experiment.**

I. Introduction

Autonomous learning of new actions is a difficult problem because the search space that has to be explored is potentially huge [1]. The dimensionality of the search space is not affected only by the degrees of freedom of the robot, which can be very high with modern humanoids, but also by the underlying representations and the robot's environment. Therefore, concern how to reduce the dimensionality of the problem while retaining the necessary flexibility to find an optimal solution to the problem is one of the biggest challenges of contemporary robotics. Among the most promising paradigms that are used for this purpose are imitation learning, where the robot obtains a rough approximation for the movements needed to accomplish the desired action by demonstration [1], [2], and reinforcement learning [3], [4], where the robot iteratively refines its movements until the desired task has been accomplished.

The goal of this paper is to speed-up the learning process by combining the ideas from imitation and reinforcement learning with statistical generalization [5]. Statistical generalization enables the inference of movements suitable for a given situation from past successful movements in similar situations. Hence, once the robot has learned how to solve a task in a number of different but related situations, it can generalize the available motor knowledge to all situations covered by the database of training examples. It should be noted that since statistical generalization is not based on a proper physical model, the generalized movement can only be an approximation for the movement, which is optimal in for a given situation. The generalization process results in a parameterization of the trajectory space with query parameters that are used for generalization. The number of these parameters and thus the dimensionality of the associated trajectory manifold is much lower than the number of parameters that are usually needed to accurately encode general trajectories. For example, in the case of ball throwing, the task is characterized by a target at which the robot should throw the ball. If the target lies in a plane, this parameterization is two dimensional. On the other hand, the number of parameters needed to encode a single throwing trajectory, for example by splines, is usually much larger. This leads us to propose to expand the initial motor knowledge in the following manner:

1) Acquire a number of initial example movements, which successfully solve the task in situations, if possible distributed across the complete task space. This can be accomplished for example by direct imitation.
2) Generalize the available training data to new situations as they arise (see Section II).
3) Refine the generalized movement by reinforcement learning on the manifold defined by the trajectories in the example database (see Section III).
4) In most cases we will not reach the optimal performance in the manifold spanned by the training trajectories. The fine tuning of the control policy should finally be accomplished by policy learning in a high dimensional trajectory space, which is required to encode the desired movement (see Section IV).
5) The newly learned movement is added to the database of example movements (together with the parameters describing the task) and the process is repeated at 2).

In the following sections we present the details of our approach.

II. Action Generalization from Previous Experiences

We use dynamic movement primitives (DMPs) proposed by [6] as our basic movement representation. With this representation, every degree of freedom is described by its own dynamic system, but with a common phase to synchronize them. In the case of point-to-point (discrete) movements, the trajectory of each robot degree of freedom y (given either in joint or in task space) is described by the following system of nonlinear differential equations

$$
\begin{aligned}
\tau\dot{z} &= \alpha_z(\beta_z(g-y)-z)+f(x), && (1)\\
\tau\dot{y} &= z, && (2)\\
\tau\dot{x} &= -\alpha_x x, && (3)
\end{aligned}
$$

where x is the phase variable and z is an auxiliary variable. α_x, α_z, β_z and τ need to be specified in such a way that the system converges to the unique equilibrium point $(z,y,x)=(0,g,0)$. The nonlinear term f contains free parameters that enable the robot to follow any smooth point-to-point trajectory from the initial position y_0 to the final configuration g

$$f(x)=\frac{\sum_{k=1}^{N} w_k\Psi_k(x)}{\sum_{k=1}^{N}\Psi_k(x)}x,\ \Psi_k(x)=\exp\left(-h_k\left(x-c_k\right)^2\right). \tag{4}$$

Here c_k are the centers of radial basis functions distributed along the trajectory and $h_k>0$ their widths. Weights w_k are estimated in such a way that the DMP encodes the desired trajectory. In the equations above, α_x, α_z, and β_z are constant. They are set so that the convergence of the underlying dynamic system is ensured [7].

Lets assume that we have a set of example trajectories together with the parameters characterizing the task

$$\mathcal{Z}=\{y_{\mathrm{d}}^k(t_{k,j}),\dot{y}_{\mathrm{d}}^k(t_{k,j}),\ddot{y}_{\mathrm{d}}^k(t_{k,j});\mathbf{q}_k|\ \begin{array}{l}k=1,\ldots,M,\\ j=1,\ldots,T_k\},\end{array} \tag{5}$$

where $y_d^k(t_j),\ \dot{y}_d^k(t_j),\ \ddot{y}_d^k(t_j)$ are the measured positions, velocities, and accelerations on trajectory k, M is the number of examples, and T_k the number of sampling points on each trajectory. Indexing of the degrees of freedom is omitted from Eq. (5) for clarity. $\mathbf{q}_k\in\mathbb{R}^n$ are the parameters describing the task in a given example situation, e. g. the location of a target at which the robot should throw a ball in the case of ball throwing. We use these parameters as query points into a database of example trajectories. The trajectories can be specified either in joint or task space. The issue is how to generate a DMP specifying a movement for every new query point $\mathbf{q}$, which in general will not be one of the example queries $\mathbf{q}_k$.

To generalize the example movements to new situations, we need to learn a function

$$\mathbf{G}\left(\mathcal{Z},\mathbf{q}\right)\longmapsto[\mathbf{w}^T,\tau,g]^T=\theta. \tag{6}$$

In general, the functional relationship between $\mathbf{q}$ and $[\mathbf{w}^T,\tau,g]^T$ given in a set of examples $\mathcal{Z}$ is unknown. Note that $\mathbf{G}\left(\mathcal{Z},\mathbf{q}\right)$ becomes a function only by constraining the generalized trajectories to be as similar as possible to the example trajectories. For example, there are many different ways of how to throw a ball into a basket at a certain location. The relationship between the basket positions (query points) and DMP parameters becomes a function only by requiring that the generalized throwing movements are similar to the example throws. In most cases it is difficult to find a global model that provides good approximation for the function $\mathbf{G}\left(\mathcal{Z},\mathbf{q}\right)$. We therefore avoid global model identification and rather apply regression techniques to generalize the movements.

Due to significantly different sizes of data sets involved in the calculation of parameters $\mathbf{w}$ on the one hand, and g and τ on the other hand, we utilized different methods to estimate them. In particular, we applied locally weighted regression for the estimation of the shape parameters and Gaussian process regression [8] to estimate g and τ. We relate the reader to [5] for details about this work. The important point for this paper is that a functional relationship between the query points and DMP parameters exists and that it can be learned from example movements.

III. Policy Learning in Constrained Domain

The general goal of policy gradient learning is to optimize the policy parameters $\theta\in\mathbb{R}^k$, maximizing the expected return of the state value cost function

$$J(\theta)=E\left[\sum_{k=0}^{H}a_k r_k(\theta)\right], \tag{7}$$

where k is the time step, a_k are time-step dependent weighting factors, H is the horizon which can be infinite and r_k is the reward received at each time step. It has become a widely accepted alternative to the value function-based reinforcement learning [9]. Here we assume that our task can be described as episodic task. The general parameter update rule of the policy gradient methods, which follows the steepest descent on the expected return, is

$$\theta_{m+1}=\theta_m+\alpha_m\nabla_\theta J(\theta), \tag{8}$$

where α_m denotes a learning rate. If the gradient estimate is unbiased and the learning rate fulfills $\sum_{m=0}^{\infty}\alpha_m>0$ and $\sum_{m=0}^{\infty}\alpha_m^2=const$, the learning process is guaranteed to converge at least to a local minimum [4]. One of the most important advantages of the policy gradient methods over the traditional reinforcement learning techniques is that we can easily limit and control the update steps. Namely, a drastic change of parameters can be hazardous for the robot and its environment. Additionally, drastic changes make the initialization of the policy based on domain knowledge or imitation learning useless, as the initial parameters can vanish after a single update step [1].

The main problem of policy gradient methods is how to obtain a good estimator for the policy gradient $\nabla_\theta J(\theta)$. If the deterministic model of the system-environment would be available, we could compute this gradient by

$$\nabla J=\frac{\partial\sum_{k=0}^{H}a_k r_k}{\partial\theta} \tag{9}$$

Unfortunately, such a model is normally not available, therefore a number of policy gradient estimation methods were proposed, such as finite gradient methods [4], likelihood ratio methods [10], natural policy gradients [11], etc. Policy gradient estimation becomes problematic as the dimensionality of policy parameters increases, since a large number of roll-outs has to be performed in order to accurately estimate the gradient. However, if a sufficient amount of previous experiences is available to perform statistical generalization (like described in Section II), we can estimate a mapping from some lower dimensional parameters $\mathbf{q}$ to the corresponding policy parameters θ. Thus our learning process defined in Eq. (8) turns into

$$\mathbf{q}_{m+1} = \mathbf{q}_m + \alpha_m \nabla_{\mathbf{q}} J(\mathbf{q}), \tag{10}$$

$$\nabla_{\mathbf{q}} J \approx \frac{\Delta \sum_{k=0}^{k=H} a_k r_k}{\Delta \mathbf{q}}, \tag{11}$$

where the dimensionality of $\mathbf{q}$ is much lower than the dimensionality of θ.

In our ball throwing experiment, function $\mathbf{G}(\mathcal{Z}, \mathbf{q})$ is estimated using example queries $\mathbf{q}_k$ obtained from previous throwing trajectories in the neighborhood of our query point $\mathbf{q}$, which is actually the desired target position. The search domain of the learning algorithm is constrained within query points $\mathbf{q}_k$.

Since the dimensionality of query point $\mathbf{q}$ is usually low and the search domain is constrained within the example queries $\mathbf{q}_k$, we found finite gradient method for policy gradient estimation appropriate. In finite gradient method, the policy parametrization is varied I times by small increments $\Delta \mathbf{q}_i, i = 1, \ldots, I$. For each policy parameter variation $\mathbf{q}_m + \Delta \mathbf{q}_i$, we perform roll-outs, collect rewards and calculate central difference estimator of cost function perturbation $\Delta J_i = J(\mathbf{q}_m - \Delta \mathbf{q}_i) - J(\mathbf{q}_m + \Delta \mathbf{q}_i)$ The policy gradient estimate can be computed by

$$\nabla J = \Delta \mathbf{Q}^{+} \Delta \mathbf{J}, \tag{12}$$

where $\Delta \mathbf{Q} = [\Delta \mathbf{q}_1, ..., \Delta \mathbf{q}_i]^T$, $\Delta \mathbf{J} = [\Delta J_1, ..., \Delta J_i]$ and $+$ denotes Moore-Penrose pseudo-inverse.

In general it is hard to predict whether the low dimensional training manifold contains the optimal control policy. Hence, once the algorithm stops converging to the desired query $\mathbf{q}$, we switch to the policy learning in the full domain of θ.

IV. Unconstrained Policy Parameters Learning

To explore the space outside of the manifold defined by the training trajectories, we have to perform learning in full unconstrained space of the control policy parameters θ. In general, this is a hard problem due to a potentially high dimensionality of parameters to be learned. For example, a typical control policy encoded by DMPs might require 30 to 50 parameters for each joint, which makes the search space huge. Recently, new efficient methods which combine the well-developed methods from statistical learning and empirical inference with classical RL approaches were proposed [12], [13]. Such algorithms can scale to significantly more complex learning systems and minimize the number of tuning parameters.

For our experiments, we selected one of state-of-the-art Reinforcement Learning algorithm PoWER [13]. It is a probabilistic policy improvement method, derived from an Expectation-Maximization framework using probability matching [14]. Unlike the policy gradient methods, it does not require the tuning of learning rate α_m because it returns the optimal parameter update. The parameter update in PoWER follows the rule

$$\theta_{m+1} = \theta_m + \frac{\langle(\theta_i - \theta_k) r(\tau_i)\rangle_{w(\tau_i)}}{\langle r(\tau_i)\rangle_{w(\tau_i)}} \tag{13}$$

where i denotes the i-th roll-out, $r(\tau_i)$ is the reward accumulated from the trajectory τ in i-th roll-out and $\langle\rangle_{w(\tau_i)}$ denotes the importance sampling. Parameters θ_i used in roll-outs are selected according stochastic exploration policy $\theta_i = \theta_m + \epsilon_i, \epsilon_i \sim \mathcal{N}(0, \sigma^2)$. The update rule is the sum of parameter exploration in each roll-out weighted by its return. The role of the importance sampling is to minimize the number of roll-outs, which are needed to estimate new policy parameters. Additionally, it automatically rejects unsuccessful roll-outs, which can be caused for example by false sensor readings. It allows the RL algorithm to re-use previous most successful roll-outs τ_i during the estimation of the new policy parameters θ_{m+1}. Importance sampler sorts all past parameters update by descending order of their return and rejects less successful ones and re-weights past explorations according to the θ_m [15].

V. Simulation and Experimental Results

In order to demonstrate the effectiveness of the proposed approach, we evaluated our approach in simulation and on a real robot. Two case studies were carried out: the first is throwing the ball at the target position and the second is a ball-in-a-cup game, also known as kendama. For these experiments we used the 7 degrees of freedom Mitsubishi Pa-10 robot equipped with Barrett hand and a vision system for ball tracking running at 120 Hz. In simulation, we used our own Matlab-Simulink simulation system [16]. Human demonstrated trajectories were captured by the Optotrak motion capture system.

In the ball throwing experiment, we constrained ball throwing to targets located in X-Z plane, as shown in Fig. 1. In simulation we first manually generated four throwing trajectories such that the ball hits the targets located at $\mathbf{q}_1 = [2.0, 0], \mathbf{q}_2 = [1.5, 0.5], \mathbf{q}_3 = [3.5, 0]$, and $\mathbf{q}_4 = [3.3, 0.5]$, all in X-Z plane. These locations together with the associated throwing trajectories form the training data. Next we searched for the throwing movements at arbitrary located query points using the proposed procedure that constrains RL to a manifold defined by the training trajectories (see Section III). First approximation of the goal trajectory was calculated using Eq. (6). Then we performed a policy gradient search in order to find a query point that generates a throwing trajectory that results in a throw within the desired precision of 0.01 m. The reward function was defined as $r = e^{-\|\mathbf{q} - \hat{\mathbf{q}}_a\|}$, where $\mathbf{q}$ denotes the desired target in X-Z plane and $\hat{\mathbf{q}}_a$ the calculated landing position.

Fig. 2 shows the error convergence of the proposed learning algorithm and its variance. We can see that the learning algorithm reaches the desired precision of 0.01 m within 20 parameter updates on the average. As it is often the case, it is possible to achieve additional speed-ups by studying the physics of the problem. Fig. 3 shows the length of the throws as a function of query points $\mathbf{q}$ in X-Z plane. The length distribution is close to linear within the training domain, which enables additional simplifications of the learning algorithm. Intuitively, we can skip the subsequent roll-outs for gradient estimation and update the parameters using the query point update law

$$\mathbf{q}_{m+1} = \mathbf{q}_m + \alpha_m \mathbf{e}_m, \tag{14}$$

where $\mathbf{e}_m = \mathbf{q} - \mathbf{q}_a$, and $\mathbf{q}_a$ is the actual query in the m-th roll-out Fig. 2 shows that learning becomes much faster and reaches the desired precision within 4 trials. Note also that each parameter update with central gradient estimation requires 4 additional roll-outs for the gradient estimation.

In the next experiment we performed the policy learning for ball throwing where we specified also the inclination angle of the ball trajectory at the target. Here we had to perform policy search in the domain of θ using PoWER because the solution was not contained in the training manifold. In the ball throwing experiment, three joints were used to encode a throwing trajectory and each joint trajectory was encoded by 12 parameters (kernel function weights $\mathbf{w}$, goal position g and trajectory duration τ), altogether 36 parameters. Comparing to only 2 parameters needed for learning on the training manifold, it is not surprising that here learning was significantly slower (see Fig. 4).

Next we considered the learning of ball-in-cup game, which is known as kendama in Japanese. It is a commonly used test case for reinforcement learning algorithms, which was studied in many previous experiments [17], [18], [19], [20]. Here the dynamics of the task is more complicated, which makes ball-in-cup a hard problem for machine learning. In order to show the effectiveness of our learning approach, we provided two human-demonstrated trajectories for the ball-in-cup, one "too-short" and one "too-long", as illustrated in Fig. 5a and 5b, respectively. Performing the search in only one dimension among the two demonstration trajectories, the algorithm quickly finds the appropriate trajectory, where the ball is caught by the cup, as illustrated in Fig. 5c. Each trajectory in X-Z plane is parameterized using 32 values, which would otherwise require to tune 64 parameters in a full-dimensional policy search. This explains why the proposed algorithm is so efficient in comparison with the previous results [17], [18], [19], [20]. Note also that there are infinitely many possible ways for how to swing the ball and catch it with the cup. Combining any two arbitrary example trajectories does not result in an appropriate policy. Rather, the two example trajectories should use the same execution strategy, as shown in Fig. 6. With properly selected demonstration trajectories, the robot learns the appropriate policy in only 2-3 roll-outs.

We also tested ball-throwing learning and ball-in-a-cup game learning in real environment. Due to the low accuracy of ball position estimation by vision and due to the limited repeatability of the robot, we could only reach the precision of 0.05 m in ball-throwing. For this precision, learning on the manifold of training movements was sufficient. The ball

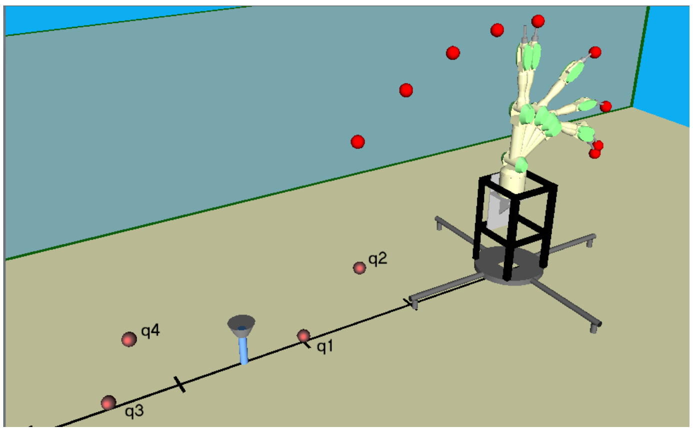

Fig. 1. Learning of ball throwing in simulated environment. $q_{1..4}$ denote four demonstration query points in X-Z plane

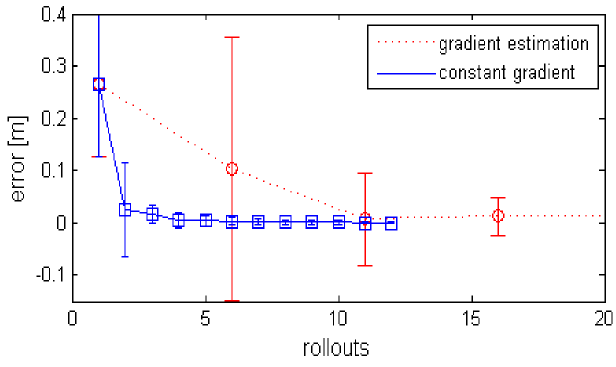

Fig. 2. Learning error convergence on the training manifold using a) finite gradient method and b) constant gradient. Vertical bars denote standard deviation of 100 different query points.

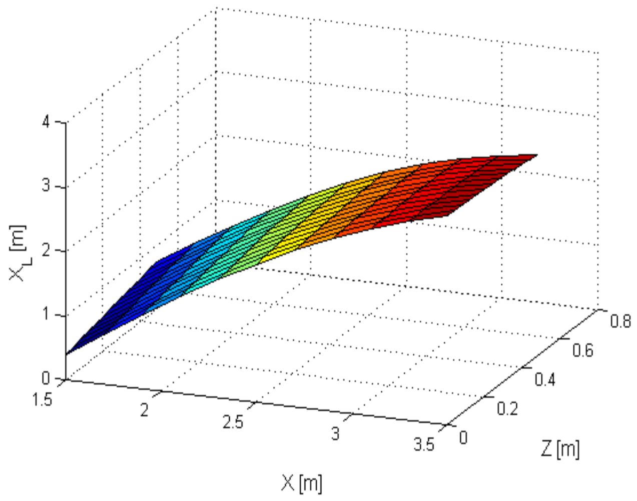

Fig. 3. Throw length as a function of query points $\mathbf{q}$, which are used to generate throwing trajectories

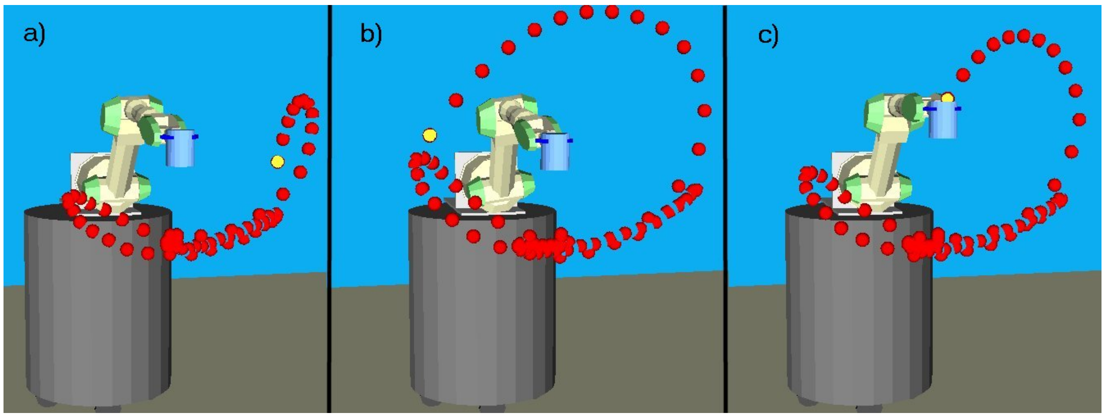

Fig. 5. Three sequences of the ball-in-a-cup game. Sequences a) and b) show two demonstration trajectories respectively, learned action is shown in sequence c). Final ball position for the corresponding robot trajectory is marked with yellow color.

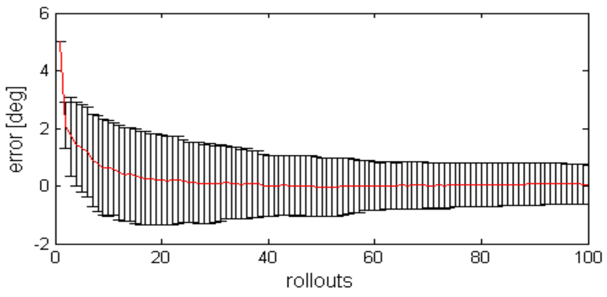

Fig. 4. Error convergence during learning of the ball trajectory inclination using PoWER. Vertical bars denote standard deviation of 100 runs

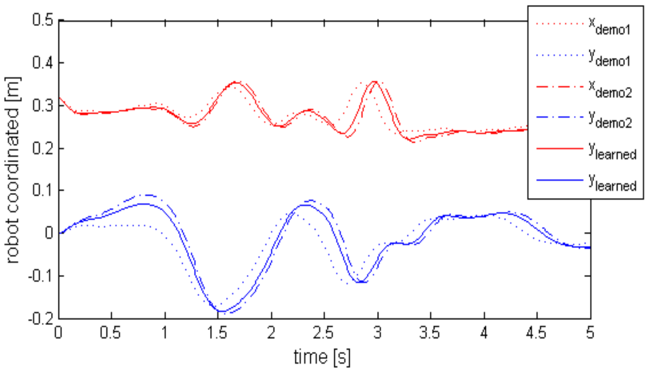

Fig. 6. Demonstrated and learned trajectories for the ball-in-a-cup game. Subscripts 1 and 2 denote the first and second demonstration trajectory, respectively.

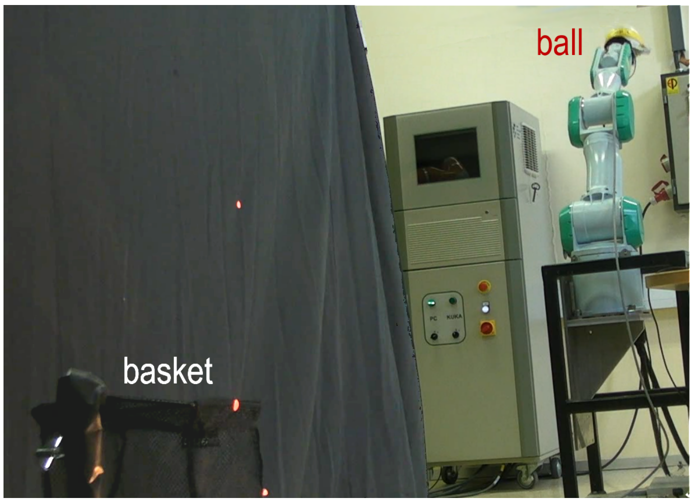

Fig. 7. Ball throwing setup

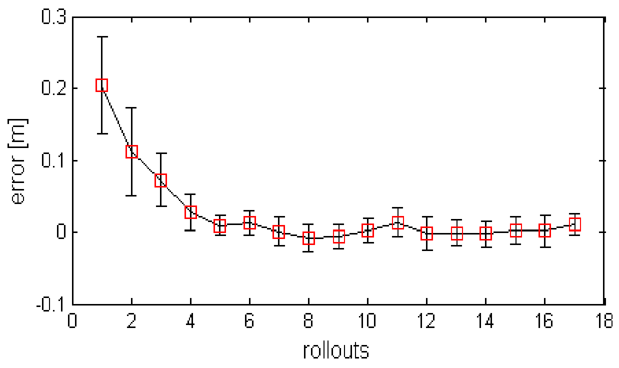

Fig. 8. Error convergence for the learning of ball throwing on a real robot. Vertical bars denote standard deviation of 20 experiments.

throwing instance and error convergence for this experiment are shown in Fig. 7 and 8, respectively. We can see that the robot learns to hit the target in 4 roll-outs on the average. The next experiment was ball-in-a-cup game learning, where it took just 4 trails in average to learn the appropriate policy. The error convergence of the real-robot experiment is shown in Fig. 10. Fig. 9 shows the real robot during the ball-in-a-cup game. The algorithms succeeded to learn the appropriate action even in the case when we shortened the rope length from $0.37m$ to $0.35m$. In this latter case it took 7 trails to learn the right action.

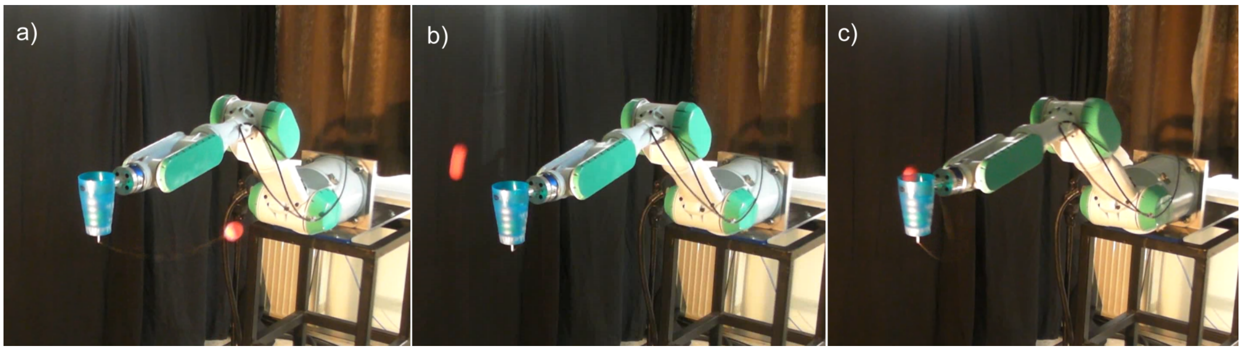

Fig. 9. Three sequences of the learning ball-in-a-cup game on Mitsubishi Pa10 robot. Sequences a) and b) show two demonstration trajectories respectively. Learned action is shown in sequence c).

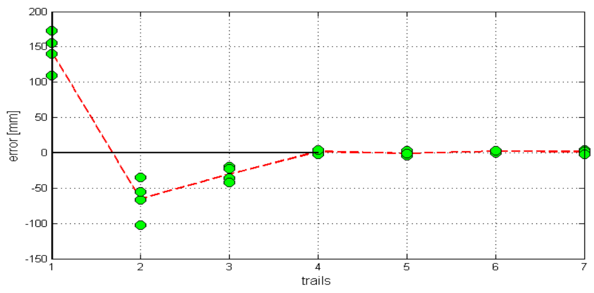

Fig. 10. Error convergence of the experimental results for the ball-in-a-cup learning. Dashed line denotes mean value of 4 experiments

VI. Conclusions

In this paper we presented a three-stage approach to expanding the database of sensorimotor knowledge. In the first stage we generalize the available training data to compute a control policy suitable for the current situation. This initial approximation can be improved using learning on the manifold defined by the training data. This enables us to perform reinforcement learning in a state space of reduced dimensionality. Since it is not possible to guarantee that the optimal solution is contained in this manifold, we explore the solutions outside of this training manifold in the third stage. A state-of-the-art reinforcement leaning technique PoWER was applied in this stage. The proposed approach was verified both in simulation on the real robot for the task of ball throwing and ball-in-a-cup game. We demonstrated that the proposed approach can result in faster learning rates.

Acknowledgment

The research leading to these results has received funding from the European Communitys Seventh Framework Programme FP7/2007-2013 (Specific Programme Cooperation, Theme 3, Information and Communication Technologies) under grant agreement no. 269959, IntellAct. It was also funded in part by Slovenian Research Agency grant J2-2348.

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] J. Peters and S. Schaal, "Reinforcement learning for parameterized motor primitives," in *International Joint Conference on Neural Networks*, 2006, pp. 73–80. [Online]. Available: http://www-clmc.usc.edu/publications/P/peters-IJCNN2006.pdf

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

[5] A. Ude, A. Gams, T. Asfour, and J. Morimoto, "Task-specific generalization of discrete and periodic dynamic movement primitives," *IEEE Trans. Robotics*, vol. 26, no. 5, pp. 800–815, 2010.

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

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

[8] C. E. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. Cambridge, MA: MIT Press, 2006.

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

[10] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 23, 1992.

[11] S. A. Kakade, "Natural policy gradient," *Advances in neural information processing systems*, vol. 14, pp. 1531–1538, 2002.

[12] E. A.Theodorou, J. Buchli, and S. Schaal, "A generalized path integral control approach to reinforcement learning," *Journal of Machine Learning Research*, no. 11, pp. 3137–3181, 2010.

[13] J. Kober and J. Peters, "Learning motor primitives for robotics," in *Proc. IEEE Int. Conf. Robotics and Automation*, Kobe, Japan, 2009, pp. 2112 – 2118.

[14] P. Dayan and G. Hinton, "Using em for reinforcement learning," *Neural Computation*, vol. 9, 1997.

[15] P. Kormushev, S. Calinon, R. Saegusa, and G. Metta, "Learning the skill of archery by a humanoid robot icub," in *Proc. IEEE-RAS International Conference on Humanoid Robots*, Nashville, USA, 2010.

[16] B. Nemec and L. Žlajpah, "Modeling of robot learning in matlab/simulink environment," in *Proc. of EUROSIM Int. Conf.*, Prague, Czech Republic, September 2010.

[17] H. Miyamoto, S. Schaal, F. Gandolfo, H. Gomi, Y. Koike, R. Osu, E. Nakano, Y. Wada, and M. Kawato, "A kendama learning robot based on bi-directional theory," *Neural Networks*, vol. 9, no. 8, pp. 1281–1302, 1996.

[18] H. Arisumi, K. Yokoi, and K. Komoriya, "Proceedings of the ieee/rsj int. conf. intelligent robots and systems," in *Kendama game by casting manipulator*, Edmonton, Canada, 2005, pp. 3187–3194.

[19] J. Kober, J. Peters, "Policy search for motor primitives in robotics," *Neural Information Processing Systems (NIPS)*, 2008.

[20] B. Nemec, M. Zorko, and L. Žlajpah, "Learning of a ball-in-a-cup playing robot," in *Proc. of 19th International Workshop on Robotics in Alpe-Adria-Danube Region RAAD*, Budapest, Hungary, June 2010.