Comparative Evaluation of Reinforcement Learning with Scalar Rewards and Linear Regression with Multidimensional Feedback

Petar Kormushev and Darwin G. Caldwell

Department of Advanced Robotics,
Istituto Italiano di Tecnologia,
Via Morego 30, 16163 Genova, Italy
{petar.kormushev,darwin.caldwell}@iit.it

Abstract. This paper presents a comparative evaluation of two learning approaches. The first approach is a conventional reinforcement learning algorithm for direct policy search which uses scalar rewards by definition. The second approach is a custom linear regression based algorithm that uses multidimensional feedback instead of a scalar reward. The two approaches are evaluated in simulation on a common benchmark problem: an aiming task where the goal is to learn the optimal parameters for aiming that result in hitting as close as possible to a given target. The comparative evaluation shows that the multidimensional feedback provides a significant advantage over the scalar reward, resulting in an order-of-magnitude speed-up of the convergence. A real-world experiment with a humanoid robot confirms the results from the simulation and highlights the importance of multidimensional feedback for fast learning.

Keywords: reinforcement learning, multidimensional feedback, linear regression

1 Introduction

A well-established tradition in Reinforcement Learning (RL) is to use a single scalar reward as a feedback signal [1]. Almost all existing RL algorithms and techniques rely on the assumption that trials are evaluated with a single numeric value. This paper aims to challenge the established tradition by showing that a state-of-the-art RL algorithm for direct policy search is easily outperformed by a simple linear regression algorithm that relies on multidimensional feedback instead of a simple scalar reward.

The paper presents a comparative evaluation of the following two learning approaches. The first approach is a conventional reinforcement learning algorithm for direct policy search which uses scalar rewards by definition. As a representative of this approach we use a state-of-the-art Expectation-Maximization based RL algorithm described in Section 2.

The second approach in this comparison is a custom linear regression based algorithm that we have proposed which uses multidimensional feedback instead

of a scalar reward. The algorithm is based on iterative vector regression with shrinking support region as explained in Section 3.2.

The two approaches are evaluated in simulation on a common benchmark problem that we have proposed. It is an aiming task where the goal is to learn the optimal parameters for aiming that result in hitting as close as possible to a given target. This problem was chosen because of its simplicity and ease of defining the feedback signal. For example, the natural measure for the performance of a trial in this task is the distance between the given target and the trial hit location.

The comparative evaluation between the two approaches shows that the multidimensional feedback provides a significant advantage over the scalar reward, resulting in an order-of-magnitude speed-up of the convergence. This is demonstrated in Section 4.1. We have also conducted a real-world experiment with a humanoid robot that confirms the results from the simulation and highlights the importance of multidimensional feedback for fast learning.

The following section gives a brief overview of RL algorithms for direct policy search, in order to position the comparative evaluation in the context of the existing RL literature.

2 Reinforcement Learning for Direct Policy Search

In conventional RL, the goal is to find a policy π that maximizes the expected future return, calculated based on a scalar reward function $R : \mathcal{I} \rightarrow \mathbb{R}$, where $\mathcal{I}$ is an input space that depends on the problem. The input space can be defined in different ways, e.g. it could be a state s, or a state transition, or a state-action pair, or a whole trial as in the case of episodic RL, etc. The policy π determines what actions will be performed by the RL agent.

Very often, the RL problem is formulated in terms of a Markov Decision Process (MDP) or Partially Observable MDP (POMDP). In this formulation, the policy π is viewed as a direct mapping function ($\pi : s \longmapsto a$) from state $s \in S$ to action $a \in A$. Alternatively, instead of trying to learn the explicit mapping from states to actions, it is possible to perform *direct policy search*, as shown in [2]. In this case, the policy π is considered to depend on some parameters $\theta \in \mathbb{R}^N$, and is written as a parameterized function $\pi(\theta)$. The episodic reward function becomes $R(\tau(\pi(\theta)))$, where τ is a trial performed by following the policy. The reward can be abbreviated as $R(\tau(\theta))$ or even as $R(\theta)$, which reflects the idea that the behaviour of the RL agent can be influenced by only changing the values of the policy parameters θ. Therefore, the outcome of the behaviour, which is represented by the reward $R(\theta)$, can be optimized by only optimizing the values θ. This way, the RL problem is transformed into a black-box optimization problem with cost function $R(\theta)$, as shown in [3].

The following is a non-exhaustive list of state-of-the-art direct policy search RL approaches:

– **Policy Gradient based RL** - in which the RL algorithm is trying to estimate the gradient of the policy with respect to the policy parameters,

and to perform gradient descent in policy space. The Episodic Natural Actor-Critic (eNAC), in [4], and Episodic REINFORCE in [5] are two of the well-established approaches of this type.
- **Expectation-Maximization based RL** - in which the EM algorithm is used to derive an update rule for the policy parameters at each step, trying to maximize a lower bound on the expected return of the policy. A state-of-the-art RL algorithm of this type is PoWER (Policy learning by Weighting Exploration with the Returns), in [6], as well as its generalization MCEM, in [7].
- **Path Integral based RL** - in which the learning of the policy parameters is based on the framework of stochastic optimal control with path integrals. A state-of-the-art RL algorithm of this type is PIˆ2 (Policy Improvement with Path Integrals), in [8].
- **Regression based RL** - in which regression is used to calculate updates to the RL policy parameters using the rewards as weights. One of the approaches that are used extensively is LWPR (Locally Weighted Projection Regression), in [9].
- **Model-based policy search RL** - in which a model of the transition dynamics is learned and used for long-term planning. Policy gradients are computed analytically for policy improvement using approximate inference. A state-of-the-art RL algorithm of this type is PILCO (Probabilistic Inference for Learning COntrol), in [10].

Direct policy search is one of the preferred methods when applying reinforcement learning in robotics [11]. This is due to the robots' inherently high-dimensional continuous action spaces for which direct policy search tends to scale up better than MDP-based methods.

3 Two Learning Approaches for Comparative Evaluation

In this section we present the two different learning algorithms that are evaluated on the common aiming task.

3.1 Learning algorithm 1: PoWER

As a first approach for learning the aiming task, we use the state-of-the-art EM-based RL algorithm PoWER by Kober *et al* [6]. We selected PoWER algorithm because it does not need a learning rate (unlike policy-gradient methods) and also because it can be combined with importance sampling to make better use of the previous experience of the agent in the estimation of new exploratory parameters. Moreover, PoWER has demonstrated superior performance in tasks learned directly on real robots, such as the ball-in-a-cup task [12] and the pancake flipping task [13].

PoWER uses a parameterized policy and tries to find values for the parameters which maximize the expected return of trials (also called trials) under the

corresponding policy. For the aiming task the policy parameters are represented by the elements of a 3D vector corresponding to the aiming direction and initial velocity of the projectile.

We define the return of a shooting trial τ to be:

$$R(\tau) = e^{-||\hat{r}_T - \hat{r}_A||}, \tag{1}$$

where $\hat{r}_T$ is the estimated 2D position of the center of the target on the target's plane, $\hat{r}_A$ is the estimated 2D position of the projectile, and $||\cdot||$ is Euclidean distance.

As an instance of EM algorithm, PoWER estimates the policy parameters θ to maximize a lower bound on the expected return from following the policy. The policy parameters θ_n at the current iteration n are updated to produce the new parameters θ_{n+1} using the following rule (as described in [12]):

$$\theta_{n+1} = \theta_n + \frac{\Big\langle (\theta_k - \theta_n) R(\tau_k) \Big\rangle_{w(\tau_k)}}{\Big\langle R(\tau_k) \Big\rangle_{w(\tau_k)}}. \tag{2}$$

In Eq. (2), $(\theta_k - \theta_n) = \Delta\theta_{k,n}$ is a vector difference which gives the relative exploration between the policy parameters used on the k-th trial and the current ones. Each relative exploration $\Delta\theta_{k,n}$ is weighted by the corresponding return $R(\tau_k)$ of trial τ_k and the result is normalized using the sum of the same returns. Intuitively, this update rule can be thought of as a weighted sum of parameter vectors where higher weight is given to these vectors which result in better returns.

In order to minimize the number of trials which are needed to estimate new policy parameters, we use a form of importance sampling technique adapted for RL [1][6] and denoted by $\langle\cdot\rangle_{w(\tau_k)}$ in Eq. (2). It allows the RL algorithm to re-use previous trials τ_k and their corresponding policy parameters θ_k during the estimation of the new policy parameters θ_{n+1}. The importance sampler is defined as:

$$\Big\langle f(\theta_k, \tau_k) \Big\rangle_{w(\tau_k)} = \sum_{k=1}^{\sigma} f(\theta_{\mathrm{ind}(k)}, \tau_{\mathrm{ind}(k)}), \tag{3}$$

where σ is a fixed parameter denoting how many trials the importance sampler is to use, and $\mathrm{ind}(k)$ is an index function which returns the index of the k-th best trial in the list of all past trials sorted by their corresponding returns, i.e. for $k = 1$ we have:

$$\mathrm{ind}(1) = \underset{i}{\mathrm{argmax}}\, R(\tau_i), \tag{4}$$

and the following holds: $R(\tau_{\mathrm{ind}(1)}) \geq R(\tau_{\mathrm{ind}(2)}) \geq ... \geq R(\tau_{\mathrm{ind}(\sigma)})$. The importance sampler allows the RL algorithm to calculate new policy parameters using the top-σ best trials so far. This reduces the number of required trials to converge and makes this RL algorithm applicable to online learning.

3.2 Learning algorithm 2: ARCHER

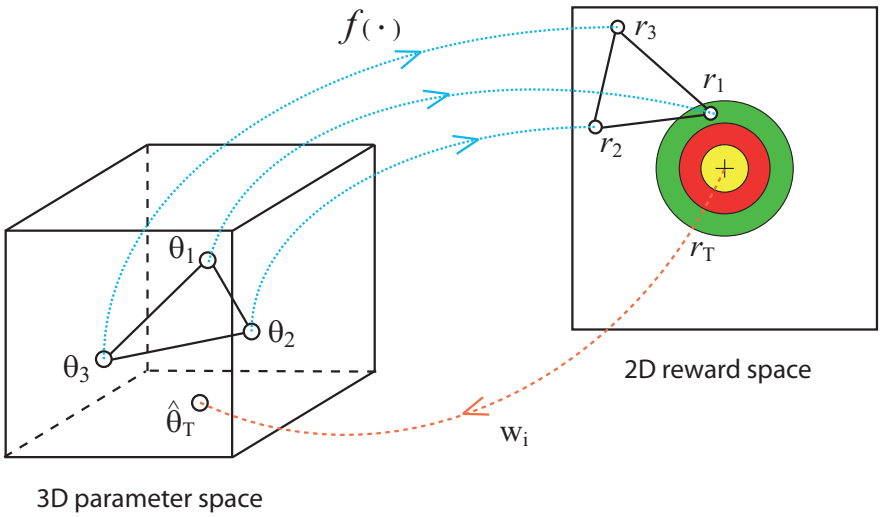


Fig. 1. The conceptual idea underlying the linear regression based algorithm (ARCHER). The goal is to find the optimal parameters from the 3D parameter space that result in hitting the given target.

For a second learning approach we propose to use a custom algorithm developed and optimized specifically for problems like the aiming task, which have a smooth solution space and prior knowledge about the goal to be achieved. We will refer to it as the ARCHER algorithm (Augmented Reward CHainEd Regression). The motivation for ARCHER is to make use of richer feedback information about the result of a trial. Such information is ignored by the PoWER RL algorithm because it uses scalar feedback which only depends on the distance to the target's center. ARCHER, on the other hand, is designed to use the prior knowledge we have on the optimum reward possible. In this case, we know that hitting the center corresponds to the maximum reward we can get. Using this prior information about the task, we can view the position of the projectile as an augmented reward. In this case, it consists of a 2-dimensional vector giving the horizontal and vertical displacement of the projectile with respect to the target's center. This information is obtained either directly from the simulated experiment in Section 4.1 or calculated by an image processing algorithm for the real-world experiment. Then, ARCHER uses a chained local regression process that iteratively estimates new policy parameters which have a greater probability of leading to the achievement of the goal of the task, based on the experience so far.

Each trial τ_i, where $i \in \{1, \ldots, N\}$, is initiated by input parameters $\theta_i \in \mathbb{R}^3$, which is the vector describing the relative position of the hands and is

produced by the learning algorithms. Each trial has an associated observed result (considered as a 2-dimensional reward) $r_i = f(\theta_i) \in \mathbb{R}^2$, which is the relative position of the projectile with respect to the target's center $r_T = (0,0)^T$. The unknown function f is considered to be non-linear due to air friction, wind flow, and etc. A schematic figure illustrating the idea of the ARCHER algorithm is shown in Fig. 1.

Without loss of generality, we assume that the trials are sorted in descending order by their scalar return calculated by Eq. 1, i.e. $R(\tau_i) \geq R(\tau_{i+1})$, i.e. that r_1 is the closest to r_T. For convenience, we define vectors $r_{i,j} = r_j - r_i$ and $\theta_{i,j} = \theta_j - \theta_i$. Then, we represent the vector $r_{1,T}$ as a linear combination of vectors using the N best results:

$$r_{1,T} = \sum_{i=1}^{N-1} w_i r_{1,i+1}. \tag{5}$$

Under the assumption that the original parameter space can be linearly approximated in a small neighborhood, the calculated weights w_i are transferred back to the original parameter space. Then, the unknown vector to the goal parameter value $\theta_{1,T}$ is approximated with $\hat{\theta}_{1,T}$ as a linear combination of the corresponding parameter vectors using the same weights:

$$\hat{\theta}_{1,T} = \sum_{i=1}^{N-1} w_i \theta_{1,i+1}. \tag{6}$$

In a matrix form, we have $r_{1,T} = WU$, where W contains the weights $\{w_i\}_{i=2}^{N}$, and U contains the collected vectors $\{r_{1,i}\}_{i=2}^{N}$ from the observed rewards of N trials. The least-norm approximation of the weights is given by $\hat{W} = r_{1,T}U^{\dagger}$, where $U^{\dagger}$ is the pseudoinverse of U.[1] By repeating this regression process when adding a new couple $\{\theta_i, r_i\}$ to the dataset at each iteration, the algorithm refines the solution by selecting at each iteration the N closest points to r_T. ARCHER can thus be viewed as a linear vector regression with a shrinking support region.

In order to find the optimal value for N (the number of samples to use for the regression), we have to consider both the observation errors and the function approximation error. The observation errors are defined by $\epsilon_\theta = ||\tilde{\theta} - \theta||$ and $\epsilon_r = ||\tilde{r} - r||$, where $\tilde{\theta}$ and $\tilde{r}$ are the real values, and θ and r are the observed values. The function approximation error caused by non-linearities is defined by $\epsilon_f = ||f - A\theta||$, where A is the linear approximation.

On the one hand, if the observations are very noisy ($\epsilon_r \gg \epsilon_f$ and $\epsilon_\theta \gg \epsilon_f$), it is better to use bigger values for N, in order to reduce the error when estimating the parameters w_i. On the other hand, for highly non-linear functions f ($\epsilon_f \gg \epsilon_r$ and $\epsilon_f \gg \epsilon_\theta$), it is better to use smaller values for N, i.e. to use a small subset of points which are closest to r_T in order to minimize the function approximation error ϵ_f. For the experiments presented in this paper we used $N = 3$ in both

[1] In this case, we used a least-squares estimate. For more complex solution spaces, ridge regression or other regularization scheme can be considered.

the simulation and the real-world, because the observation errors were kept very small in both cases.

The ARCHER algorithm can also be used for other tasks, provided that: (1) a-priori knowledge about the desired target reward is known; (2) the reward can be decomposed into separate dimensions; (3) the task has a smooth solution space.

4 Comparative Evaluation

4.1 Simulation Experiment

The two proposed learning algorithms (PoWER and ARCHER) are evaluated and compared in a simulation experiment. The aiming task in this case is an *archery-based task*, where the goal is specified as the center of the archery target. Even though the archery task is hard to model explicitly (e.g., due to the unknown parameters of the bow and arrow used), the trajectory of the arrow can be modeled as a simple ballistic trajectory, ignoring air friction, wind velocity and etc. A typical experimental result for each algorithm is shown in Fig. 2. In both simulations, the same initial parameters are used. The simulation is terminated when the arrow hits inside the innermost ring of the target, i.e. the distance to the center becomes less than 5 cm.

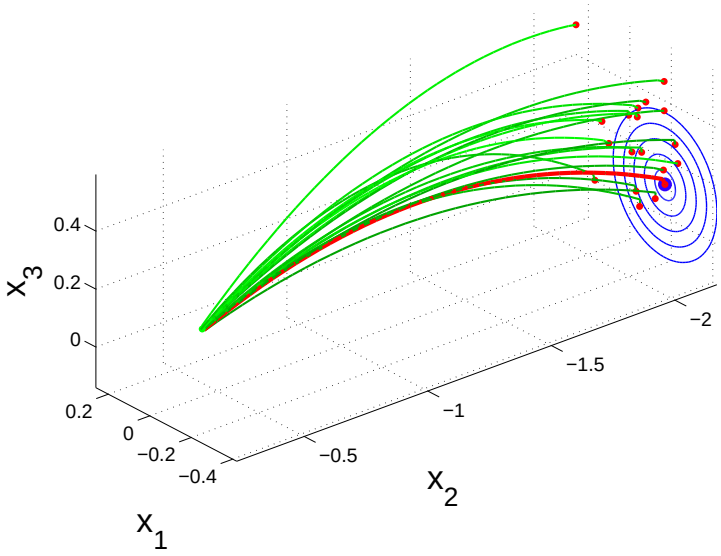
For a statistically significant observation, the same experiment was repeated 40 times with a fixed number of trials (60) in each session. The averaged experimental result is shown in Fig. 3. The ARCHER algorithm clearly outperforms the PoWER algorithm for the archery task. This is due to the use of 2D feedback information which allows ARCHER to make better estimations/predictions of good parameter values, and to the prior knowledge concerning the maximum reward that can be achieved. PoWER, on the other hand, achieves reasonable performance despite using only 1D feedback information.

Based on the results from the simulated experiment, the ARCHER algorithm was chosen to conduct the following real-world experiment.

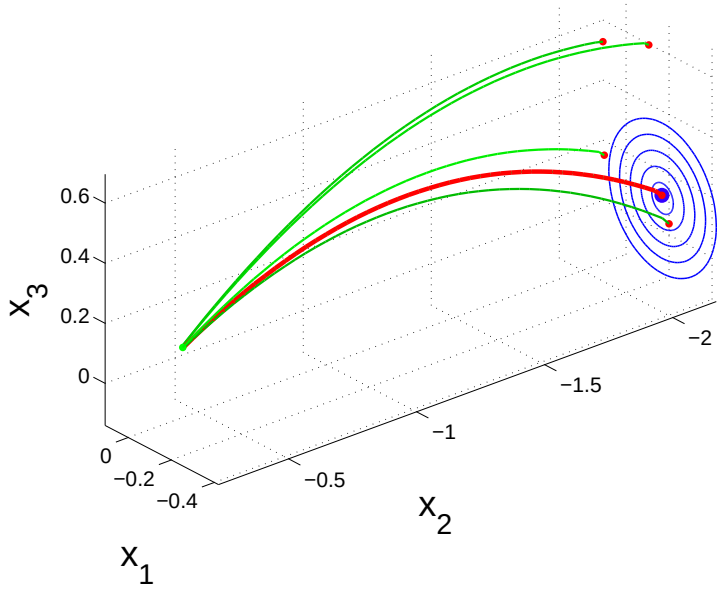
4.2 Robot Experiment

The real-world robot experimental setup is shown in Fig.4. The experiment was conducted using the iCub humanoid robot[14].

In the experiment, we used the torso, arms, and hands of the robot. The torso has 3 DOF (yaw, pitch, and roll). Each arm has 7 DOF, three in shoulder, one in the elbow and three in the wrist. Each hand of the iCub has 5 fingers and 19 joints although with only 9 drive motors several of these joints are coupled. We manually set the orientation of the neck, eyes and torso of the robot to turn it towards the target. The finger positions of both hands were also set manually to allow the robot to grip the bow and release the string suitably. We used one joint in the index finger to release the string. It was not possible to use two fingers simultaneously to release the string because of difficulties with



(a) PoWER



(b) ARCHER

Fig. 2. Simulation of the archery task. Learning is performed under the same starting conditions with two different algorithms. The red trajectory is the final trial. (a) PoWER algorithm needs 19 trials to reach the center. (b) ARCHER algorithm needs 5 trials to do the same.

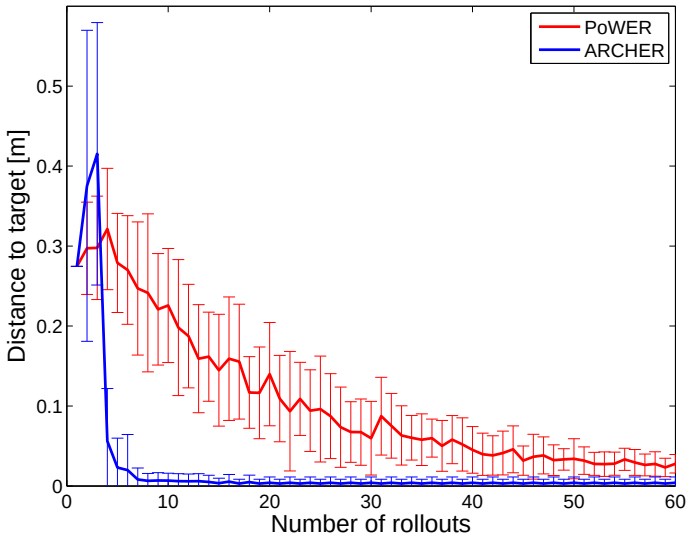


Fig. 3. Comparison of the speed of convergence for the PoWER and ARCHER algorithms. Statistics are collected from 40 learning sessions with 60 trials in each session. The first 3 trials of ARCHER are done with large random exploratory noise, which explains the big variance at the beginning.

synchronizing their motion. The posture of the left arm (bow side) was controlled by the proposed system, as well as the orientation of the right arm (string side). The position of the right hand was kept within a small area, because the limited range of motion of the elbow joint did not permit pulling the string close to the torso.

The real-world experiment was conducted using the proposed ARCHER algorithm. The 2-dimensional reward needed by ARCHER after each trial was estimated by tracking the target and the arrow in the camera image.

For the learning part, the number of trials until convergence in the real world is higher than the numbers in the simulated experiment. This is caused by the high level of noise (e.g. physical bow variability, measurement uncertainties, robot control errors, etc.). Fig. 5 visualizes the results of a learning session performed with the real robot. In this session, the ARCHER algorithm needed 10 trials to converge to the center.

Another point for comparison is that with a RL algorithm it is possible to incorporate a bias/preference in the reward. For ARCHER, a similar effect could be achieved using a regularizer in the regression.

5 Conclusion

We have presented a comparative evaluation of two learning approaches: a state-of-the-art RL algorithm for direct policy search (PoWER) which uses scalar re-

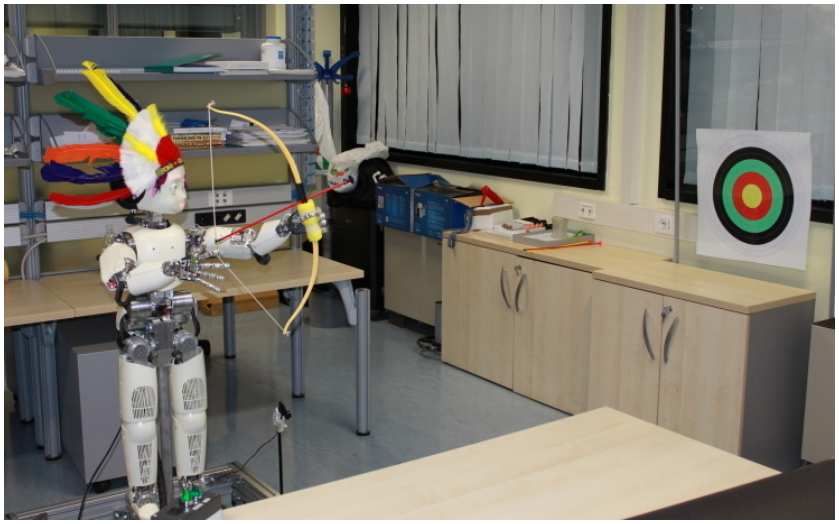


Fig. 4. Real-world experiment using the iCub humanoid robot. The distance between the target and the robot is 2.2 meters. The diameter of the target is 50 cm.

wards, and a custom linear regression based algorithm (ARCHER) that uses multidimensional feedback instead of a scalar reward. The comparative evaluation indicates that the multidimensional feedback provides a significant advantage over the scalar reward, resulting in an order-of-magnitude speed-up of the convergence. The real-world experiment with the iCub humanoid robot confirms these results.

Acknowledgments. This work is partially supported by the EU-funded project PANDORA: "Persistent Autonomy through learNing, aDaptation, Observation and ReplAnning", under contract FP7-ICT-288273.

References

1. Sutton, R.S., Barto, A.G.: Reinforcement learning: an introduction. Adaptive computation and machine learning. MIT Press, Cambridge, MA, USA (1998)
2. Rosenstein, M., Barto, A.: Robot weightlifting by direct policy search. In: International Joint Conference on Artificial Intelligence. Volume 17., Citeseer (2001) 839–846
3. Rückstieß, T., Sehnke, F., Schaul, T., Wierstra, D., Sun, Y., Schmidhuber, J.: Exploring parameter space in reinforcement learning. Paladyn. Journal of Behavioral Robotics **1**(1) (2010) 14–24
4. Peters, J., Schaal, S.: Natural actor-critic. Neurocomput. **71**(7-9) (2008) 1180–1190
5. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach. Learn. **8**(3-4) (1992) 229–256

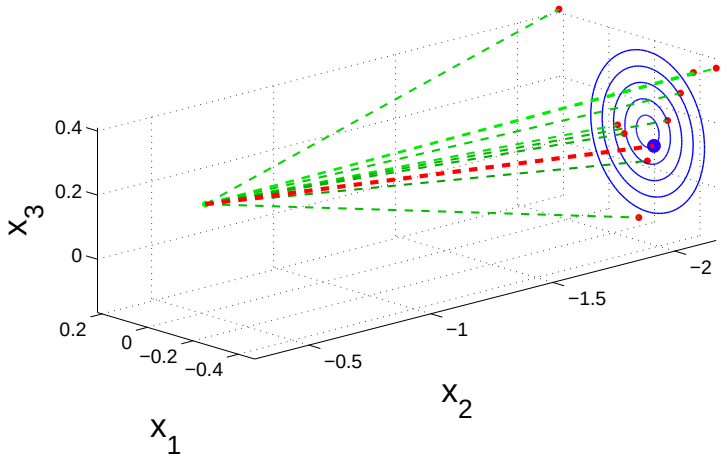


Fig. 5. Results from a real-world experiment with the iCub robot. The arrow trajectories are depicted as straight dashed lines, because we do not record the actual trajectories from the real-world experiment, only the final position of the arrow on the target's plane. In this session the ARCHER algorithm needed 10 trials to converge to the innermost ring of the target.

6. Kober, J., Peters, J.: Learning motor primitives for robotics. In: Proc. IEEE Intl Conf. on Robotics and Automation (ICRA). (May 2009) 2112–2118
7. Vlassis, N., Toussaint, M., Kontes, G., Piperidis, S.: Learning model-free robot control by a Monte Carlo EM algorithm. Autonomous Robots **27**(2) (2009) 123–130
8. Theodorou, E., Buchli, J., Schaal, S.: A Generalized Path Integral Control Approach to Reinforcement Learning. The Journal of Machine Learning Research **11** (December 2010) 3137–3181
9. Vijayakumar, S., Schaal, S.: Locally weighted projection regression: An O(n) algorithm for incremental real time learning in high dimensional spaces. In: Proc. Intl Conf. on Machine Learning (ICML), Haifa, Israel (2000) 288–293
10. Deisenroth, M.P., Rasmussen, C.E.: PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In Getoor, L., Scheffer, T., eds.: Proceedings of the 28th International Conference on Machine Learning, Bellevue, WA, USA (June 2011)
11. Kormushev, P., Calinon, S., Caldwell, D.G.: Reinforcement learning in robotics: Applications and real-world challenges. Robotics **2**(3) (2013) 122–148
12. Kober, J.: Reinforcement learning for motor primitives. Master's thesis, University of Stuttgart, Germany (August 2008)
13. Kormushev, P., Calinon, S., Caldwell, D.G.: Robot motor skill coordination with EM-based reinforcement learning. In: Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS), Taipei, Taiwan (October 2010) 3232–3237
14. Tsagarakis, N.G., Metta, G., Sandini, G., Vernon, D., Beira, R., Becchi, F., Righetti, L., Santos-Victor, J., Ijspeert, A.J., Carrozza, M.C., Caldwell, D.G.: iCub: The design and realization of an open humanoid platform for cognitive and neuroscience research. Advanced Robotics **21**(10) (2007) 1151–1175