# Replacing reward function with user feedback

**Zvezdan Lončarević, Rok Pahič, Aleš Ude, Bojan Nemec, Andrej Gams**

*All authors are with the Department of Automatics, Biocybernetics and Robotics, Jožef Stefan Institute, and with the Jožef Stefan International Postgraduate School, Jamova 39, 1000 Ljubljana, Slovenia*
*e-mail:* zvezdan.loncarevic@ijs.si

## Abstract

*Reinforcement learning refers to powerful algorithms for solving goal related problems by maximizing the reward over many time steps. By incorporating them into the dynamic movement primitives (DMPs) which are now widely used parametric representations in robotics, movements obtained from a single human demonstration can be adapted so that a robot learns how to execute different variations of the same task. Reinforcement learning algorithms require carefully designed cost function, which in most cases uses additional sensors to evaluate some environment criteria.*

*In this paper we explore possibilities of learning robotic actions using only user feedback as a reward function. Two reward functions have been used and their results are presented and compared. These were user feedback and a simplified reward function. Experimental results show that for the simple actions where only the terminal reward is given, these 2 reward functions work almost as good as having a reward function based on the exact measurement.*

## 1 Introduction

As robots become a mass consumer product, besides standard industrial environments, they are expected to be able to perform in unstructured environments, such as households or some other real-life situations. One of the biggest barriers to wider implementation of robots in our home environment is the lack of environment models. In most cases, a complex sensory system is required to estimate relevant environment parameters. That is why standard ways of programming robotic movements, that need an expert programmer for every variation of the task, are not sufficient. With humanoid robots with many degrees of freedom, autonomy is one of the main unresolved issues in contemporary robotics [1]. Imitation and reinforcement learning are the two most common approaches for solving this issue. Because robot actions are mainly recorded as parametric representations with many parameters search space for reinforcement learning algorithms that use gradient methods (finite difference gradient, natural gradient, vanilla gradient, etc.) is large. Only recently, probabilistic algorithms such as $PI^2$ and PoWER have been developed and able to deal with high dimensional search space [2, 3].

Figure 1: Experimental setup with Mitsubishi PA-10 robot

One of the major problems in reinforcement learning is determining the reward function. It was usually solved by modifying reward function according to the teachers behavior [4, 5, 6].

The main goal of this paper is to show that a simplified, reduced reward function can be used directly for some robot actions. An example of such is throwing a ball, i. e. the robot learns how to perform an accurate throwing action relying only on feedback from naive user or cheap imprecise sensors.

The paper is organized as follows: In the next section, we briefly present dynamical movement primitives that are already a widely used method in programming by demonstration, and a probabilistic algorithm called PoWER. We used it to learn new DMP parameters for the task. In Section III we introduce the reduced reward functions that were used. Next section presents experimental setup and results obtained in simulation and on the real robot. The paper concludes with a short outlook on the obtained results and suggestions for future work.

## 2 Reinforcement Learning

In this section we provide the DMPs and PoWER basics.

### 2.1 Dynamic Movement Primitives

The basic idea of Dynamic Movement Primitives (DMPs) [7] is to represent the trajectory with the well known dynamical system described with equations for the mass on spring-damper system. For a single degree of freedom (DOF) denoted by $y$, in our case one of the joint task-space coordinates, DMP is based on the following second order differential equation:

$$\tau^2 \ddot{y} = \alpha_z(\beta_z(g - y) - \tau\dot{y}) + f(x), \tag{1}$$

where $\tau$ is the time constant and it is used for time scaling (time in which the trajectory needs to be reproduced), $\alpha_z$ and $\beta_z$ are damping constants ($\beta_z = \alpha_z/4$) that make system critically damped and $x$ is the phase variable. The nonlinear term $f(x)$ 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 $g$ [8]. The phase and kernel functions are given by

$$f(x) = \frac{\sum_{i=1}^{N} \psi(x)\omega_i}{\sum_{i=1}^{N} \psi_i(x)} x, \tag{2}$$

$$\psi_i(x) = \exp\left(-\frac{1}{2\delta_i^2}(x - c_i)^2\right), \tag{3}$$

where $c_i$ are the centers of radial basis functions ($\psi_i(x)$) distributed along the trajectory and $\frac{1}{2\delta_i^2}$ (also labeled as $h_i$ [9]) their widths. Phase $x$ makes the forcing term $f(x)$ disappear when the goal is reached because it exponentially converges to 0. Its dynamics are given by

$$x = \exp\left(-\alpha_x t/\tau\right), \tag{4}$$

where $\alpha_x$ is a positive constant and $x$ starts from 1 and converges to 0 as the goal is reached. Multiple DOFs are realized by maintaining separate sets of (1–3), while a single canonical system given by (4) is used to synchronize them. The weight vector **w**, which is composed of weights $w_i$, defines the shape of the encoded trajectory. Learning of the weight vector using a batch approach has been described in [10] and [7] and it is based on solving a system of linear equations in a least-squares sense.

### 2.2 PoWER

Before the robots become more autonomous, they would have to be capable of autonomous learning and adjusting their control policies based on the feedback from the interaction with varying environment. Policy learning is an optimization process by which we want to maximize the state value of cost function:

$$J(\theta) = E\Big[\sum_{k=0}^{H} \alpha_k r_k(\theta)\Big]. \tag{5}$$

with varying policy parameters $\theta \in R^n$ [11]. In the last equation (5), $E$ is the expectation value, $r_k$ is reward given for time step $k$ and it depends of parameters that are chosen ($\theta$). $H$ is the number of time steps in which reward is given and $\alpha_k$ are the time step dependent weighting factors. Although there are numerous methods that can be used to optimize this function, the main problem is high dimensionality of parameters $\theta$ (for DMP it is typically 20-50 per joint, we use 25 per joint). Stochastic methods as PI$^2$ and PoWER were recently introduced. It was proven that PI$^2$ and PoWER perform identically when only terminal reward for periodic learning was available. With PoWER it is easy to incorporate also other policy parameters [1] such as starting ($y_0$) and ending point ($g$) and time of execution ($\tau$) of the trajectory for each DOF and not only DMP weights (**w**) so in this research the parameters that are learned are:

$$\theta = \big[\mathbf{w}, g, y_0\big]. \tag{6}$$

During the learning process this parameters are updated using the rule:

$$\theta_{m+1} = \theta_m + \frac{\sum_{k=1}^{L}(\theta_{ik} - \theta_m)r_k}{\sum_{k=1}^{L} r_k} \tag{7}$$

where $\theta_{m+1}$ and $\theta_m$ are parameters after and before update, $L$ is number of parameters in importance sampler matrix and they represent $L$ executions with highest rewards ($r_k$). $\theta_i$ is selected using stochastic exploration policy

$$\theta_i^* = \theta_m^* + \epsilon_i, \tag{8}$$

where $\epsilon_i$ is Gaussian zero noise. Variance of the noise ($\sigma^2$) is the only tuning parameter of this method. Because $\theta$ consists of three DMP parameters that cannot be searched with the same variance, it requires three different variances to be chosen. In general, higher $\sigma^2$ would lead to faster, and lower $\sigma^2$ to more precise convergence.

## 3 Reward function

As the goal of this paper is to examine the possibility to avoid need for expensive and precise sensors that can be set and calibrated only in laboratories for some actions to be successful, and enable a user to train a robot to perform variation of some action with simple instructions, thus discretized reward function is introduced. Two reward functions are compared between each other and to the exact reward function based on the distance measured with sensors.

In the first reward function (unsigned), rewards are given on a five-star scale where the terms {"one star","two stars","three stars","four stars" and "five stars"} have the corresponding rewards: $r$={1/5, 2/5, 3/5, 4/5, 1}. This means that the robot did not know in which direction to change its throws. Similar is presented in [12].

In the second reward function (signed), robot converged to its target using the feedback in which reward was formed using five possible rewards:"too short" ($r = -1/3$), "short" ($r = -2/3$), "hit" ($r = 1$), "long" ($r = 2/3$), and "too long" ($r = 1/3$). This allowed us to always put in importance sampler matrix the shots that are on the different sides of the target.

This means that both functions are of the same complexity because they have only five possible rewards. Although the five-star system can discretize better, the second should be able to compensate its lower discretization

with the new way of choosing movements that will be in the importance sampler.

# 4 Experimental evaluation

## 4.1 Experimental setup

The experiment was conducted in simulation and on a real-system. People as well as computer-simulated human reward systems were used.

The participants used a GUI to rate (give terminal reward) to the shots until the robot manages to hit the target.

In order to evaluate statistical parameters describing the success of learning, we have also created the program that should simulate human ratings with discretized reward function and with the variance between the reward borders (to simulate human uncertainty). The uncertainty was determined empirically. Simulation and this program allowed us to make much more trials without human participants. This way we tested our algorithm for ten different positions of the target with the diameter of 10 cm. (30 trials for each position) within the possible range of the robot (between 2.4m to 4m providing that the robot base was mounted on a stand of 1 m height).

Finally, the functionality of the algorithm was confirmed in the real-world using the Mitsubishi PA-10 robot. It needed to hit the basket with a 20 cm diameter with a 13 cm diameter ball. The experimental setup with the real robot is shown In Fig. 1.

## 4.2 User study

In the following, human-robot interaction (HRI) study with volunteer participants that are naive to the learning algorithm is described. Participants were using graphical user interface (GUI) in which they could rate the success of the shot in the five-star rating system (first reward function) and in the GUI where they could choose if the shot was "too short", "short", "hit", "long", or "too long" (second reward function) in a randomized order. They were not informed on how the reward function works, but only that learning was finished after they rate a shot with five stars in the first reward function system, or after they pressed "hit" in the second reward function system. The maximum number of iterations was 120. Tests with users were conducted using a simulation created in Matlab, where the robot model was made using the same measurements and limitations as on the real robot.

## 4.3 Results

Figure 2 shows throwing error convergence, which tends towards zero for all experiments. The top plot shows the results where human judgment was simulated using Matlab environment and presents statistics of 300 trials. Each trial had update in 120 iterations. Bottom figure represents error convergence from the results with human judgment. Although human judgment criteria differs among participants, the robot was still able to converge to its goal.

In Fig. 3 rewards that were given to the executed shot are shown. The top plot shows rewards ($r$) that were computer generated and the bottom one shows rewards that were given using human judgment. Statistics for computer generated rewards is taken from 300 trials and for human rewards from 10 participants that did the experiment.

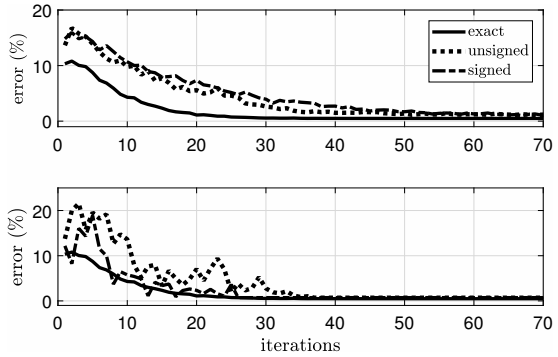


Figure 2: The top picture shows mean error convergence caused by computer simulated human judgment and the bottom one by real human judgment. Solid line denotes the results based on the real measurement reward function, dotted denotes the results for the unsigned reward function and dashed denotes results for the signed reward function.

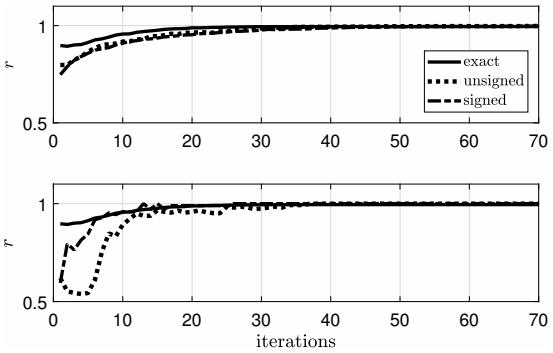


Figure 3: The top picture shows computer simulated reward and the bottom one real human mean reward convergence. Solid line denotes the results based on the real measurement reward function, dotted denotes results for the unsigned reward function and dashed denotes results for the signed reward function.

Figure 5 (top) shows the statistics of average and the last (bottom) iteration in which the first shot happened in computer simulated judgment (shaded bars) and in the experiment with people (white bars). Results are shown for the cases where exact distance was measured, for five-star reward function (unsigned discrete), and for the reward function where people were judging according to the side of the basket where the ball fell (signed discrete). It shows that with computer simulated human judgment (shaded bars), unsigned works only slightly better, but with the signed one, the worst case scenario is better. Surprisingly, with the human judgment, signed reward function drastically outperformed the unsigned. This can be explained by the fact that people tend to rate the shots in comparison to the previous one and that way unintentionally form a gradient. This is a fact similar to what was discussed in [13].

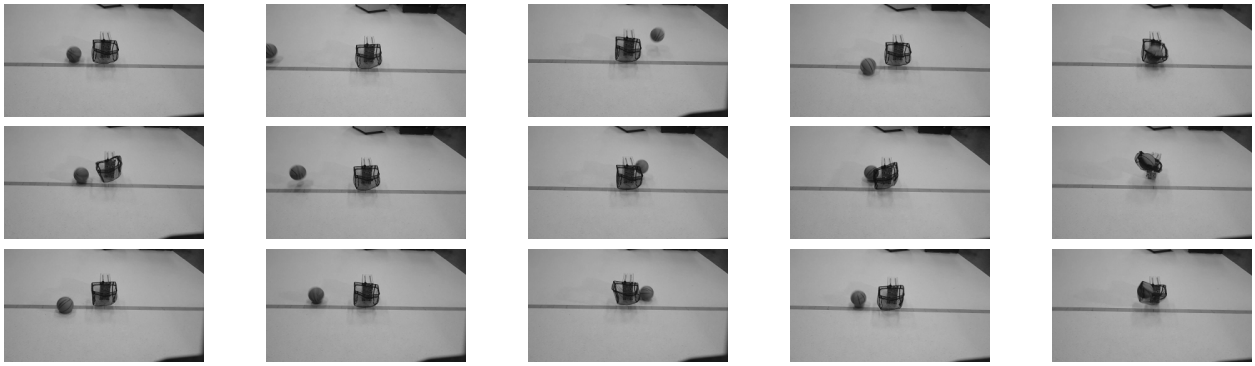

Figure 4: Experiment on the real robot: In the first row, exact distance was measured, in the second row unsigned reward system was used and in the third one signed reward function was used. The robot was throwing the ball from the right.

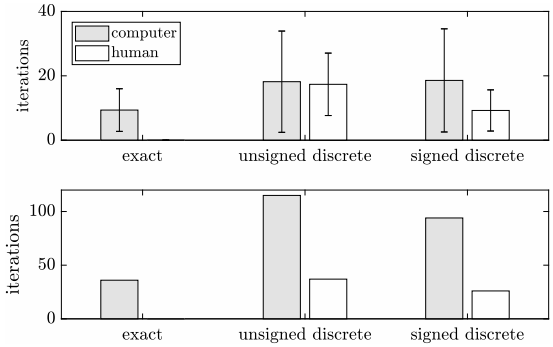


Figure 5: Average first shot and its deviation in Nakagami distribution (top) and worst case first shot (bottom): computer (shaded bars), human (white bars)

## 5 Conclusion

Results show that reinforcement learning of some simple tasks can be done with the reduced reward function almost equally good as with exact measurement based reward function. Even so, the problems related to reinforcement learning in general stay the same: it is very difficult and time-consuming task to set the appropriate noise variance (especially if more parameters are varied like in this case). Note that excessive noise variance results in jerky robot trajectories, which might even damage the robot itself.

That is why in the future, we will test this using the latent space of the neural network, where we have less different search variances to tune and also a possibility to train neural network on executable shots so that too big variance cannot lead to the trajectories that differ a lot from the original, demonstrated one.

## References


[1] B. Nemec, D. Forte, R. Vuga, M. Tamosiunaite, F. Worgotter, and A. Ude, "Applying statistical generalization to determine search direction for reinforcement learning of movement primitives," *IEEE-RAS International Conference on Humanoid Robots*, pp. 65–70, 2012.

[2] E. Theodorou, J. Buchli, and S. Schaal, "A Generalized Path Integral Control Approach to Reinforcement Learning," *Journal of Machine Learning Research*, vol. 11, pp. 3137–3181, 2010.

[3] J. Kober and J. Peters, "Learning motor primitives for robotics," *2009 IEEE International Conference on Robotics and Automation*, pp. 2112–2118, 2009.

[4] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," *Twenty-first international conference on Machine learning - ICML '04*, p. 1, 2004.

[5] W. Knox, C. Breazeal, and P. Stone, "Learning from feedback on actions past and intended," *In Proceedings of 7th ACM/IEEE International Conference on Human-Robot Interaction, Late-Breaking Reports Session (HRI 2012)*, 2012.

[6] S. Griffith, K. Subramanian, and J. Scholz, "Policy Shaping: Integrating Human Feedback with Reinforcement Learning," *Advances in Neural Information Processing Systems (NIPS)*, pp. 1–9, 2013.

[7] A. Ijspeert, J. Nakanishi, and S. Schaal, "Movement imitation with nonlinear dynamical systems in humanoid robots," *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, vol. 2, no. May, pp. 1398–1403, 2002.

[8] D. Forte, A. Gams, J. Morimoto, and A. Ude, "On-line motion synthesis and adaptation using a trajectory database," *Robotics and Autonomous Systems*, vol. 60, no. 10, pp. 1327–1339, 2012.

[9] A. Gams, A. J. Ijspeert, S. Schaal, and J. Lenarčič, "On-line learning and modulation of periodic movements with nonlinear dynamical systems," *Autonomous Robots*, vol. 27, no. 1, pp. 3–23, 2009.

[10] 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.

[11] B. Nemec, A. Gams, and A. Ude, "Velocity adaptation for self-improvement of skills learned from user demonstrations," *IEEE-RAS International Conference on Humanoid Robots*, vol. 2015-February, no. February, pp. 423–428, 2015.

[12] A.-L. Vollmer and N. J. Hemion, "A User Study on Robot Skill Learning Without a Cost Function: Optimization of Dynamic Movement Primitives via Naive User Feedback," *Frontiers in Robotics and AI*, vol. 5, no. July, 2018.

[13] A. L. Thomaz and C. Breazeal, "Teachable robots: Understanding human teaching behavior to build more effective robot learners," *Artificial Intelligence*, vol. 172, no. 6-7, pp. 716–737, 2008.