2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)
Beijing, China, November 6-9, 2018

User Feedback in Latent Space Robotic Skill Learning

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

***Abstract*— In order to operate in everyday human environment, humanoids robots will need to autonomously learn and adapt their actions, using among other reinforcement learning methods (RL). A common challenge in robotic RL is also the generation of appropriate reward functions. A vast body of literature investigates how active human feedback can be introduced into an interactive learning loop, with recent publications showing that user feedback can be used for the RL reward. However, increased complexity of robotic skills in everyday environment also increases their dimensionality, which can practically prevent use of user feedback for the reward, because too many trials are needed.**

In the paper we present the results of using discretized, user-assigned reward for RL of robotic throwing, with an emphasis on learning in the feature space, i. e., latent space of a deep autoencoder network. Statistical evaluation of a user study with 15 participants, who provided feedback for robotic throwing experiments, show that for certain tasks, RL with discrete user feedback can be effectively applied for robot learning.

I. INTRODUCTION

Autonomous operation and interaction of humanoid robots in everyday human environment requires continuous learning and adaptation of their actions and skills [1]. Learning complete actions and/or skills from scratch is not feasible, because the search space is simply too large [2]. Typically, an initial motion is somehow provided, often through learning by demonstration (LbD) [3]. This initial demonstration is unlikely to be directly applicable for the current situation of the external world, and is adapted [4]. Furthermore, sometimes skill transfer from a human to a robot is prohibitively hard, as the demonstrator may not be able to effectively achieve the same action on the robot, either due to different kinematic and dynamic properties of the robot (correspondence problem [5]), or due to the nature of the task itself.

One of the common methods for autonomous refinement of skills is reinforcement learning (RL), which offers a framework and a set of tools for the design of sophisticated and hard-to-engineer behaviors [6]. RL is an area of machine learning where an agent (software agent, or robot in the real-world) tries to maximize the accumulated reward [6]. In an episodic setting, the task is restarted after each episode.

However, the high number of degrees of freedom (DoFs) of humanoid robots with continuous states and actions increases the dimensionality beyond the range of practical use (curse of dimensionality) [7]. Through the use of

All authors are with the Humanoid and Cognitive Robotics Lab, Department of Automatics, Biocybernetics and Robotics, Jožef Stefan Institute, and with the Jozef Stefan International Postgraduate School, Jamova 39, 1000 Ljubljana, Slovenia, `name.surname@ijs.si`

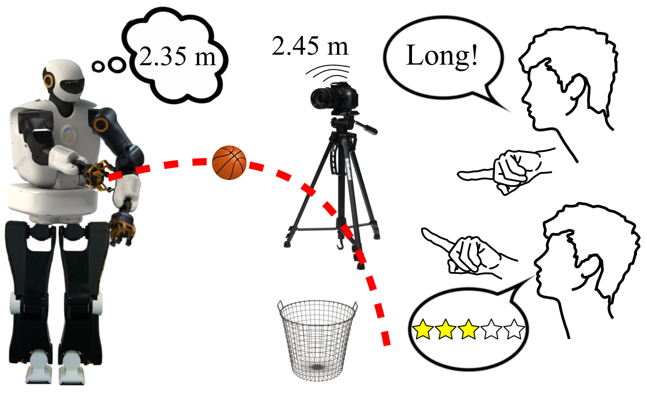

Fig. 1. The measurement of an action is different when using an on-board sensors or an external measuring mechanism. Human feedback can only be qualitative.

parametrized policies RL can be scaled into continuous actions in high dimensional spaces. Furthermore, the aforementioned introduction of prior knowledge through LbD reduces the search space. Policy search, a subfield in reinforcement learning, which focuses on finding good parameters for a given policy parametrization [8], is particularly applicable in such cases.

One of the challenges in RL is the design of an appropriate reward function, which is in many cases far from trivial. On the contrary, it is often a complex problem even for domain experts [9]. Furthermore, acting in the real-world and receiving feedback through sensors implies that the true-state may not be completely observable and/or noise-free [6]. Besides the robot's on-board sensors, additional external sensors are often applied.

These listed challenges are preventing practical use of RL by non-experts in everyday, home environments, where robotic assistants of the future are expected to operate. However, it has been recently shown that a modern optimization system can be intuitively used by non-experts in such an environment [9]. The authors investigated whether a non-modified optimization system, where the feedback and reward function are replaced by naive user feedback through a simple $1-5$ user interface, can learn the ball-in-cup game (Kendama). Despite using human-feedback, commonly assumed noisy, unreliable, and not optimal for teaching [10], Vollmer & Hemion [9] showed that the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) RL algorithm [11] was successful in learning the ball-in-cup skill in most cases.

In this paper we took this as the motivation and tested whether the same successful application of human feedback can be achieved in the feature space of a task. This paper:

- investigates how well learning of robotic skills through

user feedback fares in reduced, latent space of autoencoders;
- investigates how this compares with learning in the configuration space of the robot;
- discusses generality of using user reward functions.

Our research is based on a user study and statistical evaluation of learning the skill of robotic throwing at a target using the PoWER [7] algorithm.

The rest of this paper is organized as follows. The next Section provides a short overview of related work. Robotic throwing use-case is presented in Section III. Section IV gives the basics on used trajectory representation and introduces deep autoencoder networks applied for dimensionality reduction. Section V explains the used RL algorithms and reward functions. Results of a user study are presented in Section VI and discussed in Section VII.

II. Related Work

This work falls into three topics of research: designing a reward function for RL, interactive machine learning, and RL in latent (feature) space.

Starting with the latter, our baseline for comparison of RL is policy search on dynamic movement primitives (DMPs) [12], using an expert-defined reward. Even learning of DMP parameters in combination with tactile and visual feedback might make the search space too large for practical application [13], and RL in latent space of actuator redundancies and leveraging autoencoder representations have been proposed [14], [13]. Deep autoencoders and variational autoencoders have also been used to train movement primitives in a low-dimensional latent space [15], [16]. In this paper we run RL in the latent space of a deep autoencoder network, which greatly reduces the dimensionality. However, depending on the size of the latent space, it can also reduce the accuracy of the representation [16].

Regarding the reward function for RL, the reward design process is far from trivial. In [6], RL is listed as highest in both sequential complexity and reward structure complexity. Methods of determining the reward function algorithmically (learning the reward) have been proposed. This is called Inverse Reinforcement Learning (IRL) [17], where given a policy or history of behavior, one tries to find a reward function that explains it. A huge body of work exists on the topic of IRL, including algorithms that can learn disentangled rewards that generalize effectively under variation in the underlying domain [18]. Rewards are also often manually tweaked in order to guide the learning system to quick success. This is called reward shaping [19].

The literature is rich with examples of using human-provided reward signals for RL. However, in many cases the action space of the robot is comprised of known, discrete actions [9]. For the continuous action space, where humans have also been added to the loop in supervised learning settings, e. g. for coaching [20], less examples exist. Preference-based learning, where the human selects the preferred action from two examples, has been proposed [21], [22]. Learning a model of a reward, provided by a human, was also previously discussed [23], and has been applied to RL tasks in continuous space [24], [25].

Vollmer & Hemion [9] have applied user feedback to an unmodified learning algorithm and conducted a user-study to determine the strategy the users undertake. One of their conclusions is that a simple, generic user-interface, used by non-experts, can be applied instead of a complex reward function and an elaborate feedback system. In this paper we go beyond this by evaluating the effectiveness of using discrete user feedback for reward in reduced, feature space of robotic tasks.

III. Use-case: Robotic throwing

The use-case scenario is learning of accurate robotic throwing of a ball into a basket using the PoWER RL algorithm. The learning only takes place in one direction, i. e., in the distance of the throw. The orientation of the robot is assumed correct. We used a Mitsubishi PA-10 robot for the execution of the throw. Three degrees of freedom (DOF) of the robot, which contribute to its motion in the sagital plane, were used for the throwing. The experimental setup is depicted in Fig. 2.

Throwing was chosen as a task that was previously already considered in RL settings [26], and can thus provide a benchmark for comparison between using an exact reward and using a discretized user-feedback reward in the feature space. Furthermore, throwing is a task that is relatively easy for the users to estimate. In Section VII we discuss this aspect of RL tasks with user-feedback.

Throwing can also be relatively accurately mathematically modeled. We exploited this to conduct a statistical comparison between using an exact reward, and a discretized reward.

IV. TRAJECTORY REPRESENTATION

In this paper we apply policy search with different reward functions in two spaces: in the configuration space of the robot and in the latent space of a deep autoencoder network.

A. Configuration Space

To encode the trajectories of motion in the configuration space of the robot (joint trajectories) for an efficient policy

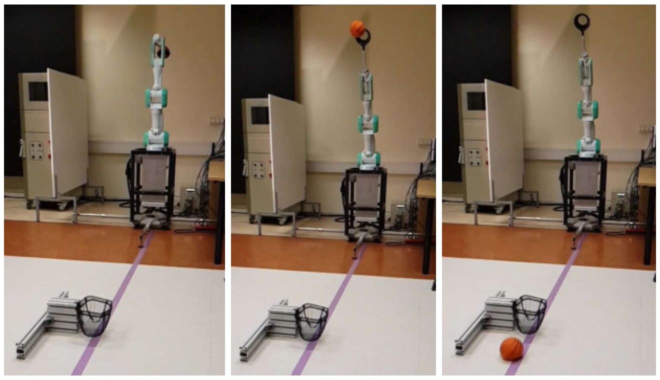
Fig. 2. Experimental setup for evaluation of RL with user-feedback in the real world. Initial posture (left), ball in-flight (center), landing spot (right). Final configuration of the robot is shown in the right-two frames.

search, we use the well-known DMP framework. We refer the reader to [12] for details on DMPs. For completeness we provide a short recap for 1 DOF in Appendix A.

The weight vector $\mathbf{w}$ of a DMP defines the shape of the encoded trajectory. RL is applied for modifying (optimizing, learning) the weights from the initial demonstration, so that the reward is maximized. However, the goal g and starting position y_0 can also be learned. In this paper we learn DMP parameters that describe the trajectories of 3 active DOFs of the robot

$$\begin{aligned} \boldsymbol{\theta} &= \left[\boldsymbol{\theta}_1^T, \boldsymbol{\theta}_2^T, \boldsymbol{\theta}_3^T\right]^T, \\ \boldsymbol{\theta}_i &= \left[\mathbf{w}_i^T, g_i, y_{0,i}\right]^T, \ i = 1,2,3. \end{aligned} \tag{1}$$

We used $N = 20$ weights per DOF, which makes the learning space of $\boldsymbol{\theta}$ 66 dimensional.

B. Latent Space

To reduce the dimensionality of the learning problem from 66 we used autoencoders. An antoencoder is a neural network used for unsupervised encoding of data, typically applied for dimensionality reduction [15]. It is comprised of two parts: an encoder and a decoder network. Using the encoder network part, where the number of neurons in the hidden layers is less than that of the input layer, forces the data through a bottleneck (latent space), where the most relevant features are extracted. We designed the autoencoder with a 3-dimensional[1] latent space

$$\boldsymbol{\theta}^l = [\boldsymbol{\theta}_1^l, \boldsymbol{\theta}_2^l, \boldsymbol{\theta}_3^l]^T. \tag{2}$$

The decoder network part reconstructs the feature representation so that the output data $\boldsymbol{\theta}'$ matches the input data $\boldsymbol{\theta}$. We used (1) as the input data, which also defines the input and output layer sizes to 66. The autoencoder was comprised of 6 hidden layers with 25, 18, 3, 8, 19, and 24 neurons. Activation function for each hidden layer is $\mathbf{y} = \tanh(\mathbf{W}\boldsymbol{\theta}^\# + \mathbf{b})$, with $\boldsymbol{\theta}^\star = \{\mathbf{W},\ \mathbf{b}\}$ the autoencoder parameters and $\boldsymbol{\theta}^\#$ the input into the neurons. Note that the input is different for each layer, because it is the output of the previous layer. The activation function of the output layer was linear. After the training we split the autoencoder in the encoder and the decoder parts. The encoder maps input into the latent space $\boldsymbol{\theta}^l = g(\boldsymbol{\theta})$ and the decoder maps from latent space to the output $\boldsymbol{\theta}' = h(\boldsymbol{\theta}^l)$, i. e., again into DMP parameters that describe the robot joint trajectories.

Training of the parameters of the autoencoder ($\boldsymbol{\theta}^\star$) is described by

$$\boldsymbol{\theta}^\star = \arg\min_{\boldsymbol{\theta}^\star} \frac{1}{n}\sum_{i=1}^{n} L(\boldsymbol{\theta}^{(i)}, \boldsymbol{\theta}'^{(i)}) \tag{3}$$

$$= \arg\min_{\boldsymbol{\theta}^\star} \frac{1}{n}\sum_{i=1}^{n} L(\boldsymbol{\theta}^{(i)}, h(g(\boldsymbol{\theta}^{(i)}))), \tag{4}$$

[1]Optimization of the autoencoder showed that using more than 3 latent space variables resulted in some of them being constant, while using less reduced autoencoder performance. See also the next subsection.

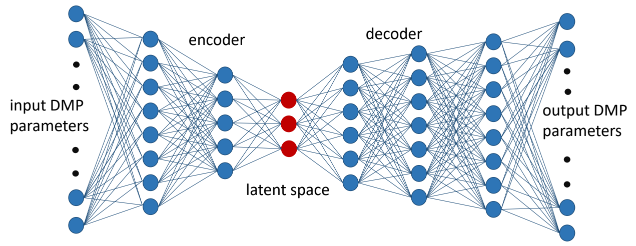

Fig. 3. Illustration of an autoencoder structure with six hidden layers. Note that the number of neurons per layer in the used autoencoder is too high for an effective illustration. The depicted number of neurons per layer does not match the number we used (66 for input and output layers, 25, 18, 3, 8, 19, 24 for hidden layers).

where L is Euclidian distance between the input and output vectors and n is the number of samples. Figure 3 shows an illustration of such autoencoder architecture.

1) Database for training: Database of trajectories for training the autoencoder, i. e., of DMP parameters describing throwing trajectories, was generated in simulation, where we neglected air drag and modeled ball release position when the robot starts to decelerate. In the generation process we first defined the throwing target for each example, that is, the desired range, height and hitting angle of the throw. These three parameters define the ball trajectory. Consequently, the size of the autoencoder latent space layer is intuitively three. We then searched for the points on this trajectory that can be reached with the desired robot end-effector position and velocity. We optimized the ball release position to have the minimal weighted norm of the joint velocity vector. Finally, we produced minimal-jerk trajectories bounded so that the desired end-effector position had 0 initial and final velocities, and a maximal velocity at the desired time.

We generated the database for a target grid in the range from 2 to 4 meters and the height from 0 to 2 meters. For each point we generated trajectories of the same duration, but with maximal velocity at 5 different times. In the process of generation we discarded all trajectories and targets that required joint positions or velocities outside of robot limits. This way we got 2400 trajectories examples. We used 70% of the database for training 15% for validation and 15% for testing. Figure 4 shows the trained latent space of the autoencoder.

V. LEARNING ALGORITHM & REWARDS

A. PoWER

Policy Learning by Weighting Exploration with the Returns (PoWER) [7] is an Expectation-Maximization (EM) based RL algorithm that can be combined with importance sampling to better exploit previous experience. It uses a parametrized policy and tries to maximize the expected return of trials. In this paper we use PoWER because it is robust with respect to reward functions [7]. It also uses only the terminal reward, and no intermediate rewards, which makes it easier for users to assign it. For completeness, PoWER is explained in Appendix B.

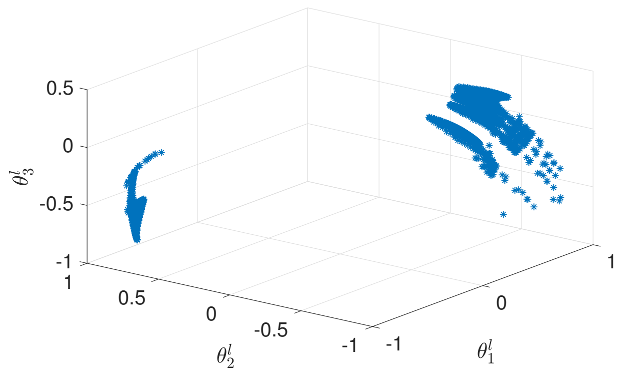

Fig. 4. Latent space of the DMP-parameters autoencoder shows clustering of the training data-base. Each of the five clusters, four on the right and one on the left, represents a different ball-release time. The cluster on the left corresponds to training trajectories that make a slight movement in the opposite direction before stopping.

In the 66-dimensional configuration space we learn parameters $\boldsymbol{\theta}_{n+1}$ using (14) – (16). In latent space we use (14) – (16) to learn $\boldsymbol{\theta}^l_{n+1}$. However, the learning space in this case is only 3 dimensional. The values of parameters in latent space $\boldsymbol{\theta}_l$ define the DMP parameters, and therefore the shape of the trajectory on the robot, trough the decoder network

$$\boldsymbol{\theta}'_{n+1} = h(\boldsymbol{\theta}^l_{n+1}). \tag{5}$$

B. Reward functions and applying them

Three distinct reward functions were used for the given use-case: *exact* reward, calculated from an exact measurement using an external measuring system; *unsigned* reward, assigned as 1-5; and what we called *signed* reward[2], where over- and under throw were rewarded differently.

Exact ($\cdot_e$) reward was assigned to the throw k by an algorithm through

$$R_{k,e} = L_d - |L_d - L_k|, \tag{6}$$

where L_d is the distance to the basket and L_k is length of k-th trial (throw). L_k can be easily determined in simulation. In the real-world we used an external vision system, similar as in [26], for detecting L_k, while L_d was input manually. Importance sampler, defined in (15), sorts the trials with respect to the reward in a descending manner, and takes $l = 3$ best trials to update the policy.

Unsigned ($\cdot_u$) reward uses a five-star scale to assign the reward. Therefore, RL does not know in which direction to change the throw. 5-stars was assigned to a hit, while the rest were assigned descending with the distance to the target,

$$R_{k,u} \in [1, 2, 3, 4, 5]^T. \tag{7}$$

Importance sampler randomly takes three from the trials with the highest rewards.

Signed ($\cdot_s$) reward distinguishes between long and short throws. The user reward is in this case

$$R_{k,s} \in [\text{very long}, \text{long}, \text{hit}, \text{short}, \text{very short}]^T. \tag{8}$$

[2]PoWER only uses positive reward, the sign was only used in the importance sampler.

For PoWER, this translates to

$$R_{k,s} \in [1, 2, 3, 2, 1]^T. \tag{9}$$

Because PoWER only takes positive reward, it can't distinguish between short and long throws. However, one can use the sign in the importance sampler. Signs [+ + / - -] were given to each of the awards in (9), respectively (long throws got negative signs), and saved. The importance sampler ranks the trials so that two with different signs are ranked the same and used for the policy update ($l = 2$). If more trials have the same reward, random trials with the highest reward are chosen (one for each sign).

Augmenting the reward with knowledge on where the reward is the highest, and therefore incorporating the sign of the reward, has been applied to RL with the ARCHER algorithm [27]. In this paper, however, we use original PoWER formulation from [7], but take the sign into consideration in the importance sampler.

VI. User Study & Results

A. User Study

We performed a randomized cross-over study that compared the effectiveness of RL with user-feedback, using the signed and unsigned rewards.

10 adults, 7 males and 3 females, volunteered for the study. They were informed of the procedure and free informed consent was obtained from all. They had various experience with robotics, ranging from no experience to PhD in robotics.

In the experiment, the participant were asked to give rewards to throws (trials) using a simple GUI with 5 buttons, each corresponding to a reward. The assigned reward was used to update the throwing policy. The participants were asked to make the robot hit the target by assigning the reward to throws. No other instructions were given to the participants, and they had no prior knowledge on the underlying learning algorithms. They were only told that once the highest reward was given, this will stop the learning, and that they have as many attempts as needed. Every participant first saw the same initial throw to 3.1 m. Every participant had to make the robot hit the targets twice, at 2.7 m and at 3.5 m, both in latent and in configuration spaces.

Besides the user study, we also conducted 75 simulated learning experiments with modeled human-assigned reward, marked in the figures with *computer*. The same initial and target conditions applied. Modeling of human-assigned rewards is described in Appendix C. The goal of the simulated learning study was to see if human intelligence behind the reward has an effect on the outcome.

B. Results

Figure 5 show the convergence of throwing over the iterations. Average error, given as the distance between the target (basket) and the landing spot of the thrown ball, is shown. The results show that all approaches, i. e., in latent and in configuration space, and with human-assigned reward and simulated human-assigned reward converge. Using exact reward, shown in human experiments for comparison, is not

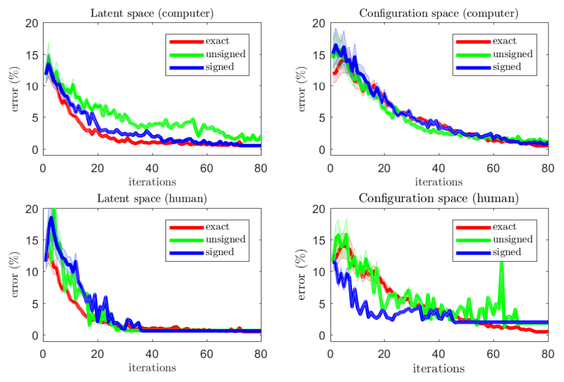

Fig. 5. Average error of throwing in each iteration, for four different experiments, each with three different rewards. RL in latent space is shown in the left plots, and in configuration space in the right plots. The top line shows the average results of the simulated study (computer), and the bottom line the results of the user study (human). Exact results for human-study are depicted for comparison. In all the plots, the exact reward is marked with the red line, the unsigned reward with the green line and the signed reward with the blue line. Shaded areas show the exponential distribution.

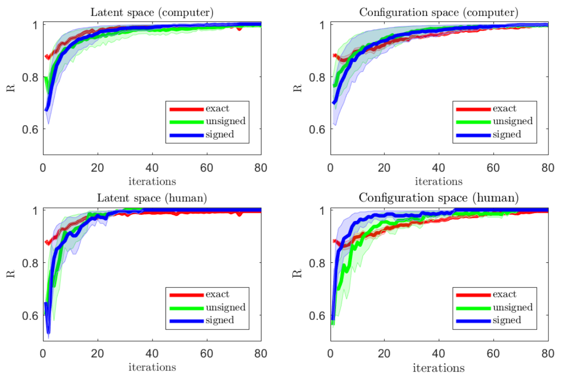

Fig. 6. Average reward of throwing in each iteration, for four different experiments, each with three different rewards. Shaded areas show Beta distribution, applicable for datasets in the interval $[0, 1]$. See caption of Fig. 5 for other details of plots.

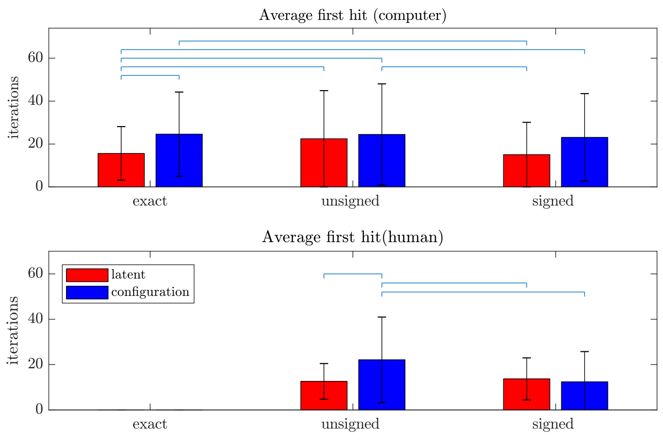

Fig. 7. Average required number of throws until the first hit. Top plot: simulated study (computer). Vertical lines depict statistically significant difference between experiments. Bottom plot: human-assigned reward (human).

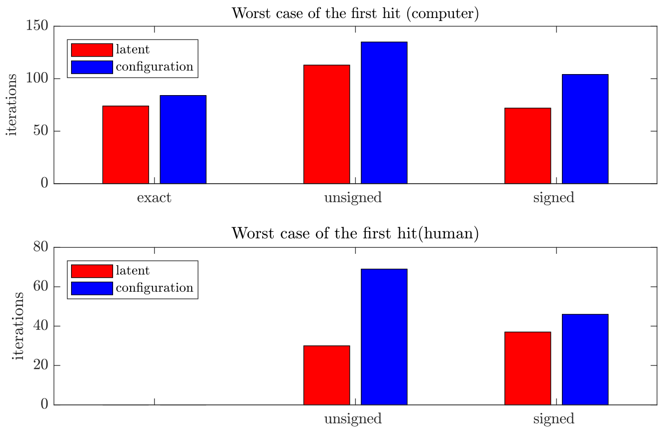

Fig. 8. Maximal number of needed iterations until first hit for computer (top) and human (bottom) datasets.

evidently faster in all cases. Configuration space RL was on average slower than latent space RL, no matter the reward. Configuration space (computer) with either exact, signed or unsigned reward was the slowest. Figure 6 shows the average normalized reward assigned to the iterations. The same relations as in Fig. 5 can be observed.

Different average rate of convergence for different cases is shown in Fig. 7, where the average number of throws until the first hit and standard deviation are depicted. We investigated the effects of the learning space (latent, configuration) and reward (exact, signed, unsigned) using two-way repeated measures ANOVA with independent variables [learning-space(2) $\times$ reward(3)] (computer) and [learning-space(2) $\times$ reward(2)] (human). The differences between different pairs were tested with posthoc t-tests with Bonferroni correction. The level of statistical significance used was .05 for all statistical tests.

For computer simulated study, using exact reward is statistically significantly different when applied in latent space or configuration space. Using either of the other two rewards is not statistically significantly different between the learning spaces. Also of note is that in latent space using unsigned reward takes statistically significantly longer than using the exact reward.

Results of the user study (human) show statistically significant difference between using unsigned reward in latent and configuration spaces, and statistically significant difference between using signed and unsigned reward in the configuration space.

Faster learning of humans as compared to simulated humans is evident from comparing the top and the bottom plots of Fig. 8, which shows the worst case scenarios. The simulated study (computer) needed more iterations. Furthermore, worst case in the configuration space was always worse than in the latent space.

VII. DISCUSSION

We have shown that user-assigned reward for PoWER algorithm was good enough for the throwing error to converge practically to zero, i. e., to hit the basket; and we have shown that it can be applied in the feature space and in the configuration space.

The results of the simulated user-study have shown that learning in feature space, i. e., in the latent space of autoencoder, is statistically significantly faster that in the configuration space. This is far from surprising, as the number of parameters of learning is much smaller. Why this is not generally applicable is the consequence of the required amount of data needed to train the autoencoder. In our simulated database, which we could also use for the real robot, there were 2400 examples. While the task of throwing allows rather accurate and easy modeling, this is not true in general. Therefore, it might be easier to train the system in the configuration space for a small number of examples, and then use generalization for the initial learning, as was proposed in, e. g., [26]. All successful throws could, over a longer period of time, make up an appropriate database for training of the autoencoder.

As for using a different kind of rewards, user-study showed statistically significant difference between using signed and unsigned rewards in configuration space. One should note that the way we set-up the rewards, the unsigned reward has 5 different levels, while the signed reward only has 3, and therefore carries less information. The inclusion of sign, which is quite intuitive in the case where the highest reward can be easily determined, is obviously enough for humans to accelerate learning. The simple models of the reward, used in the simulated study, were not sufficient to accelerate learning.

Human intelligence is thus obviously a factor in learning. However, in the throwing use-case, the reward was easy to assign. When the reward is not as straightforward and a person cannot easily tell whether it is higher or lower, user-assigned reward might not work. In Fig. 9 we show the results of toy RL example of trajectory fitting, where the exact error is the difference of surfaces under the desired and the learned trajectories. With user feedback, where the reward is discrete, trajectory fitting is not very accurate even when the initial approximation is at .75 of the desired trajectory. Using also intermediate reward would probably help, but this opens up a lot of new problems, for example, how does one assign intermediate reward.

For real autonomous RL of robots, without user feedback, assigning of rewards remains an open issue. Simple replacement of the rewards with 1 – 5 stars is not possible, because that implies a transformation from a quantitative information (exact reward) to a qualitative reward. In other words, it means that the reward itself needs to be assigned, which closes the circle on itself.

VIII. CONCLUSIONS

In general, and as expected, results show that learning in latent space is faster. Using the knowledge of expected highest award through adding the sign to the reward and using this information in the importance sampler was shown to work, but statistically significant difference between such and usnigned rewards was only shown in configuration space.

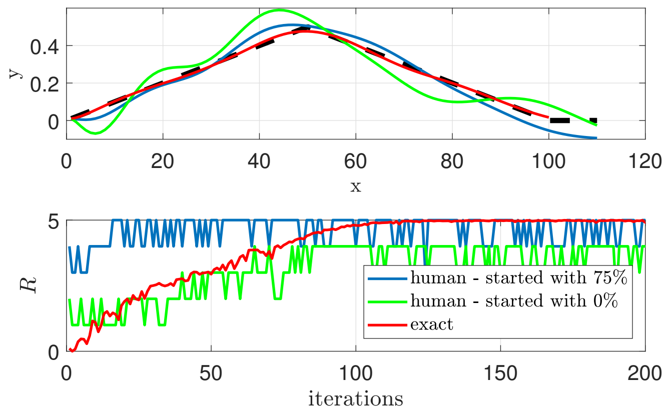

Fig. 9. Trajectory fitting results when using exact, or human-assigned rewards, for 1 user. Top: desired trajectory (black dashed), fitting results using exact reward (red), fitting result using human assigned reward with 0 initial trajectory (green), and with .75 of desired trajectory for the initial trajectory (blue).

Results of this user study show that future humanoid robot household assistant will be able to utilize RL with human assigned reward in configuration space and also in feature space, if an appropriate database for the creation of such space will be available. However, a simple and intuitive reward system is only applicable to tasks where the reward is straightforward to the user.

Acknowledgement: This work has received funding from the GOSTOP programme C3330-16-529000 co-financed by Slovenia and EU under ERDF and by the Slovenian Research Agency grant J2-7360, ”Learning and autonomous adaptation of dual arm assembly and service tasks”.

REFERENCES

[1] J. Peters, J. Kober, K. Muelling, O. Kroemer, and G. Neumann, “Towards robot skill learning: From simple skills to table tennis,” in *European Conference on Machine Learning (ECML)*, 2013.

[2] S. Schaal, “Is imitation learning the route to humanoid robots?,” *Trends in Cognitive Sciences*, vol. 3, no. 6, pp. 233–242, 1999.

[3] M. Hersch, F. Guenter, S. Calinon, and A. Billard, “Dynamical system modulation for robot learning via kinesthetic demonstrations,” *IEEE Transactions on Robotics*, vol. 24, no. 6, pp. 1463–1467, 2008.

[4] B. Nemec, R. Vuga, and A. Ude, “Efficient sensorimotor learning from multiple demonstrations,” *Advanced Robotics*, vol. 27, no. 13, pp. 1023–1031, 2013.

[5] A. Alissandrakis, C. L. Nehaniv, and K. Dautenhahn, “Solving the correspondence problem between dissimilarly embodied robotic arms using the alice imitation mechanism,” in *Second International Symposium on Imitation in Animals & Artifacts (AISB)*, 2003.

[6] J. Kober, D. Bagnell, and J. Peters, “Reinforcement learning in robotics: A survey,” *International Journal of Robotics Research*, no. 11, pp. 1238–1274, 2013.

[7] J. Kober and J. Peters, “Policy search for motor primitives in robotics,” *Machine Learning*, no. 1-2, pp. 171–203, 2011.

[8] M. P. Deisenroth, G. Neumann, and J. Peters, “A survey on policy search for robotics,” *Foundations and Trends in Robotics*, pp. 388–403, 2013.

[9] 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,” *Front. in Rob. and AI*, vol. 5, p. 77, 2018.

[10] P. Weng, R. Busa-Fekete, and E. Hüllermeier, “Interactive q-learning with ordinal rewards and unreliable tutor,” in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECMLPKDD)*, 2013.

[11] F. Stulp and O. Sigaud, “Robot Skill Learning: From Reinforcement Learning to Evolution Strategies,” *Paladyn, Journal of Behavioral Robotics*, vol. 4, no. 1, pp. 49–61, 2013.

[12] A. Ijspeert, J. Nakanishi, P. Pastor, H. Hoffmann, and S. Schaal, "Dynamical movement primitives: Learning attractor models for motor behaviors," *Neural Computation*, vol. 25, no. 2, pp. 328–373, 2013.

[13] K. S. Luck, G. Neumann, E. Berger, J. Peters, and H. B. Amor, "Latent space policy search for robotics," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 1434–1440, Sept 2014.

[14] H. van Hoof, N. Chen, M. Karl, P. van der Smagt, and J. Peters, "Stable reinforcement learning with autoencoders for tactile and visual data," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3928–3934, Oct 2016.

[15] N. Chen, J. Bayer, S. Urban, and P. van der Smagt, "Efficient movement representation by embedding dynamic movement primitives in deep autoencoders," in *IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pp. 434–440, Nov 2015.

[16] N. Chen, M. Karl, and P. van der Smagt, "Dynamic movement primitives in latent space of time-dependent variational autoencoders," in *IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pp. 629–636, Nov 2016.

[17] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 663–670, 2000.

[18] S. L. Justin Fu, Katie Luo, "Learning robust rewards with adversarial inverse reinforcement learning," *arXiv:1710.11248*, 2017.

[19] A. D. Laud, *Theory and Application of Reward Shaping in Reinforcement Learning*. PhD thesis, University of Illinois at Urbana-Champaign, 2004.

[20] A. Gams, T. Petrič, M. Do, B. Nemec, J. Morimoto, T. Asfour, and A. Ude, "Adaptation and coaching of periodic motion primitives through physical and visual interaction," *Robotics and Autonomous Systems*, vol. 75, pp. 340 – 351, 2016.

[21] P. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," *arXiv preprint arXiv:1706.03741*, 2017.

[22] D. Sadigh, A. Dragan, S. Sastry, and S. Seshia, "Active preference-based learning of reward functions," in *Proceedings of Robotics: Science and Systems*, 2017.

[23] W. B. Knox and P. Stone, "Interactively shaping agents via human reinforcement: The tamer framework," in *Proceedings of the Fifth International Conference on Knowledge Capture*, K-CAP '09, (New York, NY, USA), pp. 9–16, ACM, 2009.

[24] N. A. Vien and W. Ertel, "Reinforcement learning combined with human feedback in continuous state and action spaces," in *IEEE Int. Conf. on Develop. and Learn. and Epigen. Rob. (ICDL)*, pp. 1–6, 2012.

[25] R. Akrour, M. Schoenauer, M. Sebag, and J.-C. Souplet, "Programming by Feedback," in *International Conference on Machine Learning*, no. 32 in JMLR Proceedings, (Beijing, China), pp. 1503–1511, 2014.

[26] B. Nemec, R. Vuga, and A. Ude, "Exploiting previous experience to constrain robot sensorimotor learning," in *11th IEEE-RAS International Conference on Humanoid Robots*, pp. 727–732, Oct 2011.

[27] P. Kormushev, S. Calinon, R. Saegusa, and G. Metta, "Learning the skill of archery by a humanoid robot icub," in *10th IEEE-RAS Int. Conf. on Humanoid Robots*, pp. 417–423, Dec 2010.

APPENDIX

A. Dynamic Movement Primitives

A second order differential equation with a nonlinear part $f(x)$ defines a DMP [12],

$$\tau^2 \ddot{y} = \alpha_z(\beta_z(g - y) - \tau\dot{y}) + f(x), \tag{10}$$

where the output of the DMP is y. τ is the time constant α_z and β_z are damping constants ($\beta_z = \alpha_z/4$), 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,

$$f(x) = \frac{\sum_{i=1}^{N} \psi(x)\omega_i}{\sum_{i=1}^{N} \psi_i(x)} x, \tag{11}$$

$$\psi_i(x) = \exp\left(-\frac{1}{2\delta_i^2}(x - c_i)^2\right). \tag{12}$$

Here c_i are the centers of N radial basis functions $\psi_i(x)$ and $\frac{1}{2\delta_i^2}$ their widths. The phase is governed by

$$x = \exp\left(-\alpha_x t/\tau\right), \tag{13}$$

where α_x is a positive constant and x starts from 1.

B. PoWER

In PoWER, policy parameters $\boldsymbol{\theta}_n$ at current iteration n are updated to new parameters using [7]

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n + \frac{\langle(\boldsymbol{\theta}_k - \boldsymbol{\theta}_n)R_k\rangle_{w(\tau_k)}}{\langle R_k\rangle_{w_k}}, \tag{14}$$

where k denotes the number of the trial, and R_k the reward of that trial. Importance sampling, denoted by $\langle\cdot\rangle_{w(\tau_k)}$, minimizes the number of required trials (rollouts). It is defined by

$$\langle f(\boldsymbol{\theta}_l, \tau_l)\rangle_{w(\tau_l)} = \sum_{l=1}^{m} f(\boldsymbol{\theta}_{\mathrm{in}(l)}, \tau_{\mathrm{in}(l)}). \tag{15}$$

Here m is a predefined number specifying how many trials the importance sampler uses, and $\mathrm{in}(l)$ is a function that returns the index of the $l-$th best trial τ in the list of all trials. $\boldsymbol{\theta}_k$ is selected using stochastic exploration policy

$$\boldsymbol{\theta}_k^* = \boldsymbol{\theta}_n^* + \epsilon_k, \tag{16}$$

where ϵ_i is Gaussian zero noise. Variance of the noise (σ^2) needs to be set. Higher σ^2 leads to faster, and lower σ^2 to more precise convergence.

C. Simulated Human Rewards

We modeled the human reward with a simple discretization of the exact reward. We empirically defined the regions of the same, discrete reward.

Unsigned reward for the algorithm $(\cdot_a)$ was defined

$$R_{k,u,a} = \begin{cases} 5 & \text{if } 0.00 \text{ m} \leq |L_d - L_k| \leq 0.10 \text{ m} \\ 4 & \text{if } 0.10 \text{ m} < |L_d - L_k| \leq 0.35 \text{ m} \\ 3 & \text{if } 0.35 \text{ m} < |L_d - L_k| \leq 0.60 \text{ m} \\ 2 & \text{if } 0.60 \text{ m} < |L_d - L_k| \leq 1.00 \text{ m} \\ 1 & \text{else.} \end{cases} \tag{17}$$

We added random uncertainty in the range of 0.05 m to simulate noisy human feedback. *Signed* reward for the algorithm was set to

$$R_{k,s,a} = \begin{cases} (-)\ 1 & \text{if } -0.70 \text{ m} \leq L_d - L_k \\ (-)\ 2 & \text{if } -0.70 \text{ m} < L_d - L_k \leq -0.10 \text{ m} \\ (\ \ )\ 3 & \text{if } -0.10 \text{ m} \leq L_d - L_k \leq 0.10 \text{ m} \\ (+)\ 2 & \text{if } 0.10 \text{ m} < L_d - L_k \leq 0.70 \text{ m} \\ (+)\ 1 & \text{else,} \end{cases} \tag{18}$$

which is similar to (17). Again, the regions were set empirically. Only positive rewards were used for PoWER and the signs were stored and used for the importance sample.