Proceedings of the 27th IEEE International Symposium
on Robot and Human Interactive Communication,
Nanjing, China, August 27-31, 2018

Smooth and Efficient Policy Exploration for Robot Trajectory Learning

Shidi Li, Chee-Meng Chew, and Velusamy Subramaniam

***Abstract*— Many policy search algorithms have been proposed for robot learning and proved to be practical in real robot applications. However, there are still hyperparameters in the algorithms, such as the exploration rate, which requires manual tuning. The existing methods to design the exploration rate manually or automatically may not be general enough or hard to apply in the real robot. In this paper, we propose a learning model to update the exploration rate adaptively. The overall algorithm is a combination of methods proposed by other researchers. Smooth trajectories for the robot can be produced by the algorithm and the updated exploration rate maximizes the lower bound of the expected return. Our method is tested in the ball-in-cup problem. The results show that our method can receive the same learning outcome as the previous methods but with fewer iterations.**

I. Introduction

Direct policy search approach for robot learning has received much attention recently [1], [2]. The algorithms like PoWER [1], [3] and PI^2 [4], [5] are proposed to learn complex motor skills in continuous high-dimensional space. They can handle the robot tasks like grasping [6], ball-in-cup [3], [7], dart throwing [8] and so on. Researchers are also trying to make these algorithms convenient to use. For example, both the PoWER and PI^2 do not need to adjust the learning rate [5] which usually exists in some other learning algorithms and affects the performance dramatically.

However, some hyperparameters such as the exploration rate in these policy search algorithms still require to be manually tuned [5], [6]. Tuning the exploration rate is the key to reduce the number of iterations that a robot is required to learn a trajectory. A reduction in the number of iterations results in lower training costs, as it may take unbelievably long time to train the physical robot [9]. On the other hand, when the learning algorithms are applied to a real robot, it is important for the algorithms to generate smooth trajectories which are suitable for the actual robot execution. Therefore, both the sample efficiency of learning and the dynamics constraints of the real robot should be considered when designing the exploration rate.

In this paper, we propose an adaptive exploration rate adjustment for PoWER, making it more efficient and safer for applications in the real robot. We make use of the same updating rule as in [1] to update the policy and the exploration rate. We design the covariance matrix as the exploration rate inspired by [10], [11] to avoid excessive acceleration and generate smooth trajectories, which overcomes the difficulty that the naive PoWER algorithm has when applied to the real robot [7] without the state-dependent exploration [12]. The algorithm also resulted in faster learning in both simulations and real-robot experiments. This work reduces the human efforts on tuning the exploration rate when a robot is learning the trajectories for the manipulation tasks. Meanwhile, the number of iterations required for the robot learning a trajectory is reduced. Our method provides a general way to apply the policy search algorithms to the real robot and improves their generality, applicability, and efficiency.

The paper is structured as follows. Section II discusses the current approaches to adjust the exploration rate for robot policy search algorithms in details and their shortcomings. Section III demonstrates the methodology of policy search and our approach that updates the exploration rate. Section IV illustrates the results of the simulations and experiments. The contribution and the future work are concluded in Section V.

II. Related Work

A framework of the policy search for the robot trajectory learning is presented in Fig. 1. The robot Learning from Demonstration (LfD) [13] is usually used for locating an initial policy for a robot learning task. However, the initial policy from LfD may fail or have a low success rate in the robot tasks where precise trajectory learning is required [14]. To overcome the issues, the robot policy is parameterized by a set of parameters θ and a number of policy search algorithms are proposed to locally update the policy [2].

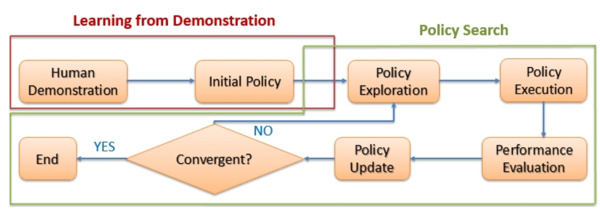

Fig. 1. Framework for policy search in the real robot applications.

Policy Learning by Weighted Exploration with the Return (PoWER) [1] is a practical policy search algorithm for robot trajectory learning. It updates the parameterized policy as well as the exploration rate by maximizing the lower bound of the expected return. Unfortunately, the resultant trajectories planned by this method may contain excessive acceleration, which may result in hazardous operation and shorten the life-span of the robot [1], [7]. Therefore, the updating rule for the exploration rate in [1] works well in simulation but results in an unsatisfied performance in the real robot [7]. Although Kober [1] proposes that we can use the state-dependent exploration [12] to avoid this shortcoming, it not only increases the complexity of the control system, but also reduces the range of the tasks which can benefit from this algorithm.

Instead of using uncorrelated exploration, researchers propose to use the correlated exploration where the sum of squared accelerations along the trajectory will be constrained [6], [7]. The covariance matrix as the exploration rate can generate reasonable perturbations when sampling the stochastic policy, leading to safer desired trajectories within the capacities of the robot control system. This method simplifies the learning model by avoiding state-dependent exploration.

In addition, some previous work [15] can adaptively update the covariance matrix as the exploration rate. However, to use this method in a physical robot, another hyperparameter to control the scale of the exploration rate is usually introduced [6], [7]. The hyperparameter still requires manual tuning without standard guideline. Poorly chosen hyperparameter may result in poor performance of the learning algorithm. It may also lead to low sample efficiency and cannot guarantee that the policy converges to a local optimum since the exploration rate may approach zero too early. These disadvantages will be illustrated through experiments in later part of the paper.

III. Methodology

This section describes the methodology adopted for a robot to learn a manipulation trajectory while adaptively updating the exploration rate for the policy search algorithm during the robot learning. We use the model of Dynamic Movement Primitives (DMP) [16] as the policy representation. The weight parameters of DMP are directly used as the policy parameters of PoWER. We also derive the updating rule for the exploration rate that ensures a safe exploration and high learning efficiency.

A. Dynamic Movement Primitives

For each dimension, the movement is generated by integrating a set of differential equations as shown in (1). The equations can be viewed as a modified linear spring-damper system with external forcing term [16]

$$\begin{aligned} &\hat{\tau}\dot{v} = K(x_g - x) - Dv - K(x_g - x_0)s + Kf(s) \\ &\hat{\tau}\dot{x} = v \end{aligned}, \quad (1)$$

where x and v are displacement and velocity; x_g and x_0 are the goal and start point; $\hat{\tau}$ is a time scaling factor, K is like a spring constant; D is like a damping constant; the third term $K(x_g - x_0)s$ serves to smooth the motion especially at the starting point, and the external forcing term $f(s)$ can be written as

$$f(s) = \frac{\sum_i \omega_i \psi_i(s) s}{\sum_i \psi_i(s)}. \quad (2)$$

In the above equation, $\psi_i(s) = e^{-h_i(s-c_i)^2}$ is a Gaussian basis function with parameters h_i and c_i which correspond to the width and center of the Gaussian kernel; ω_i is the i^{th} element of the adjustable weights. The objective of LfD is to learn the policy parameters ω_i, which correspond to the skill of the robot. The forcing term $f(s)$ is dependent on a phase variable s governed by a first order equation $\hat{\tau}\dot{s} = -\alpha s$, where constants $\hat{\tau}$ and α are set to one in the implementation.

After capturing the human's demonstrations, we can obtain the desired or target forcing function in each dimension through the following equation

$$f_{target}(s) = \frac{\hat{\tau}\dot{v} + Dv}{K} - (x_g - x) + (x_g - x_0)s.$$

As we use (2) to approximate the desired forcing term, the cost function can be defined as

$$J(\mathbf{w}) = \sum_s \Big(f_{target}(s) - f(s)\Big)^2.$$

From which, we can solve for the weights ω_i which minimize the cost function using standard regression technique. The weights serve as the policy parameters learned from demonstration.

The DMP can be used for the robot to learn the motor primitives in the continuous action domain. In the robot LfD, each kind of movements is encoded by the weights of DMP. To generalize the DMP to a new situation, the new start and goal positions x_0 and x_g can be substituted into (1). The new trajectory will be generated automatically without any human programming.

The movement generated by the DMP is a sequence of intermediate points in Cartesian space with respect to time. In real robot applications, inverse kinematics is required to convert the desired trajectories in the Cartesian space into the joint trajectories [16] in the joint space as shown in Fig. 2.

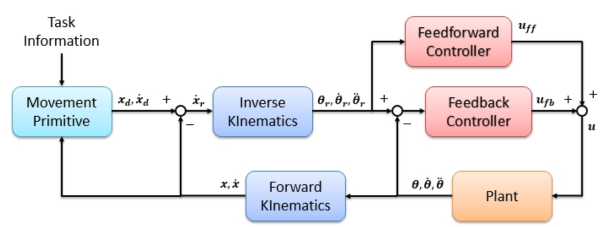

Fig. 2. DMP control diagram: the x_d, $\dot{x}_d$ are the desired positions and velocities in Cartesian space, the x, $\dot{x}$ are the actual robot executing results.

The nice features of DMP approach have been discussed in [17], especially concerning its convergence, smoothness, robustness, and the most importantly, the ease to work with policy search algorithms. The learned parameters ω_i of the DMP serve as the initial policy parameters of the robot task. They can be further improved by the policy search algorithms so that the robot can achieve the desired task.

B. PoWER

Despite the nice features of the DMP approach, the robot may still have difficulty replicating the learned tasks where precise trajectories are required [1], [3], [14], [18]. To rectify this problem, one common way is to adopt Reinforcement Learning (RL) technique to improve the initial policy learned through the DMP approach. An example of the RL techniques is PoWER [1]. It is an episode-based method that gradually improves the performance of robot policy through an exploring-executing-updating cycle as shown in Fig. 1. It updates the policy by maximizing the lower bound of the expected return. The algorithm has a simple structure and requires the users to adjust only the exploration rate and define the reward function.

In PoWER, we use the ω learned by DMP as the initial policy parameters θ_0. The agent explores the current policy θ in the policy space by expressing the policy as a stochastic one and sampling in it. A trajectory τ will be sampled with a probability $p(\tau|\theta)$ and the performance of this trajectory $J(\tau)$ will be evaluated after the robot executes the trajectory. The expected return of the current policy θ can be written as $J(\theta) = \int_\tau p(\tau|\theta)J(\tau)d\tau$. The goal of policy search is to adjust θ to maximize $J(\theta)$.

It has been shown that if we select the new policy θ' that maximizes the lower bound $L_\theta(\theta')$, the expected return will be improved [1]. The lower bound is expressed as

$$\begin{aligned} L_\theta(\theta') &= \int_\tau p(\tau|\theta)J(\tau)\log\frac{p(\tau|\theta')}{p(\tau|\theta)J(\tau)}d\tau \\ &= -D\Big(p(\tau|\theta)J(\tau)||p(\tau|\theta')\Big). \end{aligned}$$

The $D\Big(p(x)||q(x)\Big)$ denotes the Kullback-Leibler (KL) divergence between two probability distributions.

Therefore, to find the updated policy θ' in each episodic iteration, we solve the equation

$$\lim_{\theta'\to\theta}\frac{\partial}{\partial\theta'}L_\theta(\theta') = 0, \tag{3}$$

with the stochastic policy adopting a multi-dimensional Gaussian distribution with a covariance matrix $\hat{\Sigma}$. Solving the Equation (3) results in the policy updating rule of

$$\theta' = \theta + E\left\{\sum_{t=1}^{T} W_t Q^\pi_{s,a,t}\right\}^{-1} E\left\{\sum_{t=1}^{T} W_t\epsilon Q^\pi_{s,a,t}\right\}, \tag{4}$$

with $W_t = \phi(t)\phi(t)^T\Big(\phi(t)^T\hat{\Sigma}\phi(t)\Big)^{-1}$, where the $\phi(t)$ denotes a set of state-independent basis functions for the movement primitive and the ϵ is the policy exploration. The relationship between the state-action value $Q^\pi_{s,a,t}$ here and the reward can be written as

$$Q^\pi_{s,a,t} = E\left\{\sum_{\tilde{t}=t}^{T} r(s_{\tilde{t}}, a_{\tilde{t}}, s_{\tilde{t}+1}, \tilde{t})|s_t = s, a_t = a\right\},$$

where the reward function is usually defined by the human referring to the particular learning tasks.

The PoWER algorithm can iteratively update the policy by applying Equation (4) until the policy converges to the ideal optimum for the robot to complete the tasks in the real world. In practice, when estimating a target value by sampling, like PoWER in the real robot, the concept of importance sampling is usually included in the policy search. The previous rollouts with high rewards will be reused in order to increase the efficiency of learning.

C. Exploration rate

Although the policy search algorithms like PoWER and PI^2 are free to tune the learning rate, the exploration rate sill requires the users to take care [1], [5]. Since the most common way to explore in the policy space is to sample the policy parameters in a multi-dimensional Gaussian distribution [19], the essence of the exploration rate here is the covariance matrix. Tuning the exploration rate properly is the key to a better sample efficiency. The sample efficiency will increase if we update the covariance matrix also by maximizing the lower bound of the expected return [1]. Therefore, we apply the Equation (3) to the exploration rate $\hat{\Sigma}$ and we have

$$\frac{\partial}{\partial\hat{\Sigma}'}L_{\hat{\Sigma}}(\hat{\Sigma}') = 0. \tag{5}$$

It yields

$$\hat{\Sigma}' = \frac{E\left\{\sum_{t=1}^{T} diag\left\{\epsilon\right\}^2 Q^\pi_t\right\}}{E\left\{\sum_{t=1}^{T} Q^\pi_t\right\}}, \tag{6}$$

where the ϵ is the exploration in the previous rollouts.

However, it works only for uncorrelated exploration for the $\hat{\Sigma}'$ is diagonal. The uncorrelated exploration of the policy parameters may potentially cause a excessive acceleration in the output trajectory of DMP. Noted that in the Fig. 2, the output of DMP with unreasonable accelerations may surpass the capabilities of the inverse kinematics or the control system of the real robot. Although using Equation (6) to update the exploration rate can theoretically guide the robot to explore the regions with the higher rewards and work well in simulation, it has no way of showing its due power in the real robot experiments [7].

To constrain the excessive accelerations in sample trajectories, at the same time, maintaining the as simple format of policy representation as DMP, we should make use of the correlated exploration for policy search. Inspired by the optimal control, we can take the matrix $\beta_k R^{-1}$ as the exploration rate to generate the correlated exploration [6], [7]. The matrix R is defined [10], [11] by $R = A_2^T A_2$, with the 2^{nd} order differential matrix

$$A_2 = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ -2 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 1 & -2 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & -2 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 & -2 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix}.$$

There are mainly two reasons to define the matrix R in this way: (1) the sum of the square acceleration alone one trajectory will be constrained; (2) the perturbations with only small deviations will be created at the start and the end of the trajectories [10].

The hyperparameter β_k is a single scaling value changing with the iteration number. It is usually set as decreasing with the robot learning going on while the policy is approaching the ideal optimum. However, the human needs to fine tune this hyperparameter for the particular robot learning tasks. The sample efficiency is low and a unsatisfied β_k can cause a failure for the robot to find the policy optimum. For example, the β_k converges to zero faster than the policy converges to the optimum, then the robot stops exploring and never reaches the optimum.

In this paper, we conduct an eigendecomposition to R^{-1} like

$$\hat{\Sigma} = R^{-1} = \Psi \Lambda \Psi^T, \tag{7}$$

where the Λ is a diagonal matrix while the Ψ is an orthogonal matrix. We substitute the Equation (7) into Equation (5). Note that although the equation has more than one solution, if we consider the Λ' to be diagonal, it yields

$$\Lambda' = \frac{E\left\{\sum_{t=1}^{T} diag\left\{\Psi^T \epsilon\right\}^2 Q_t^\pi\right\}}{E\left\{\sum_{t=1}^{T} Q_t^\pi\right\}}. \tag{8}$$

Now, in the real robot experiments of policy search, the initial policy learned from human demonstration starts exploring with the initial exploration rate R^{-1}. Right after each iteration of updating the policy θ with PoWER, we apply Equation (8) to update the eigenvalues of the exploration rate. We then use the updated Λ' to generate the uncorrelated random vector $\lambda' \sim N(0, \Lambda')$. Thus, the $\epsilon' = \Psi\lambda'$ can be taken as the correlated exploration sample for the next iteration of policy search. Therefore, the exploration rate is updated automatically. In the experiments, it can be seen that our method to update the exploration rate can generate smooth trajectories for the robot to execute due to the inherent property of the matrix R^{-1}. It also helps the robot find the ideal optimum much more quickly than the previous methods. In addition, importance sampling and rollout reusing are also required when using Equation (8). The algorithm of PoWER updating the exploration rate with our method can be concluded as the Algorithm 1.

IV. Evaluation

In this section, we test the PoWER algorithm in the trajectory fitting benchmark simulation based on the MATLAB code provided by [1]. Then we build a ball-in-cup learning system to test the algorithm with a KUKA robot manipulator.

A. Simulation

In the basic motor skill learning task, the agent is learning a policy to produce a trajectory $x(t)$ with a 10-kernel DMP, trying to fit the given goal trajectory $g(t)$. The Q-value and the reward for a trajectory $x(t)$ are defined by

$$Q(t) = \int_t^T r(t)dt, \qquad r(t) = e^{-[x(t)-g(t)]^2},$$

where T is the duration of one trajectory. The agent starts with a set of random policy parameters and episodically updates the policy to force the output trajectory of DMP to converge to the given goal $g(t)$. Fig. 3 shows a comparison of the learning curves using different exploration rate updating methods to the same goal trajectory.

Algorithm 1 PoWER with Smooth and Efficient Policy Exploration

Require: initial policy parameters θ_0,
initial exploration rate $\hat{\Sigma}_0 = R^{-1}$.
Ensure: optimal policy parameters θ_k.
repeat
Sample: Perform rollout(s) using
$$a = (\theta_0 + \epsilon)^T \phi(t),$$
with $\epsilon = \Psi\lambda$, $\lambda \sim N(0, \Lambda)$, $\hat{\Sigma} = \Psi\Lambda\Psi^T$ as stochastic policy and collect all the $(t, s_t, a_t, s_{t+1}, r_{t+1})$ for $t = \{1, 2, ..., T+1\}$.
Estimate: Use unbiased estimate
$$\hat{Q}^\pi_{s,a,t} = \sum_{\tilde{t}=t}^{T} r(s_{\tilde{t}}, a_{\tilde{t}}, s_{\tilde{t}+1}, \tilde{t}).$$
Reweight: Compute importance weights and reweight rollouts, discard low-importance rollouts.
Update: Use
$$\Delta\theta = E\left\{\sum_{t=1}^{T} W_t Q^\pi_{s,a,t}\right\}^{-1} E\left\{\sum_{t=1}^{T} W_t \epsilon Q^\pi_{s,a,t}\right\},$$
$$\theta_{k+1} = \theta_k + \Delta\theta,$$
with $W_t = \phi(t)\phi(t)^T \left(\phi(t)^T \hat{\Sigma} \phi(t)\right)^{-1}$. Use
$$\Lambda' = E\left\{\sum_{t=1}^{T} diag\left\{\Psi^T\epsilon\right\}^2 Q_t^\pi\right\} \Big/ E\left\{\sum_{t=1}^{T} Q_t^\pi\right\}.$$
until Convergence $\theta_{k+1} \approx \theta_k$.

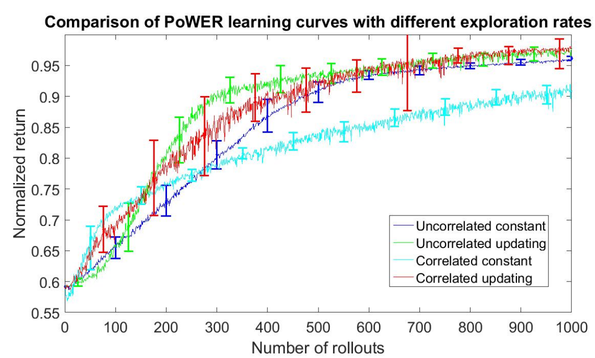

Fig. 3. A comparison of PoWER learning curves with different exploration rate updating methods: uncorrelated constant, uncorrelated updating (Equation (6)), correlated constant (βR^{-1}) and correlated updating (Equation (8)). The x-axis is the number of rollouts while the y-axis is the normalized return $Q(0)/T$.

As shown in Fig. 3, our method updating the exploration rate with Equation (8) can improve the performance of the policy like the previous methods do. Both correlated and uncorrelated explorations with the updating rules have better sample efficiency than the ones using the constant exploration rates, which means they obtain the same normalized return with fewer rollouts of iterations.

Note that the updating uncorrelated exploration with Equation (6) may perform as well as, or even better than the updating correlated exploration with Equation (8), depending on different choices of $g(t)$. However, as discussed previously, the updating uncorrelated exploration may not be suitable for the real robot [7]. Therefore, in the real robot experiments, we mainly focus on how much the correlated updating method can perform better than merely multiplying a scaling factor β_k to the constant correlated matrix R^{-1}.

B. Ball-in-cup manipulation

The ball-in-cup manipulation is that a human or a robot holds a cup with a little ball attached beneath the bottom of the cup with a string, then swings the ball into the air and catches it with the cup when the ball is falling. It can be used as a benchmark experiment to test the policy search algorithms in the real robot [1], [7].

We use a KUKA LBR iiwa 7 R800 robot manipulator with 7 degrees of freedom to execute the test trajectories. It weights at $23kg$ and has a positional repeatability of $\pm 0.1mm$. The ball is painted red and the cup is painted cyan. Both colors are chosen intentionally so as to be easy for color segmentation in image processing. The ball has a diameter of $40mm$ and a weight of $33.3g$. The diameter of cup rim is $70mm$. The initial length of the string is $320mm$. We use a Kinect RGB-D camera to track the position of the ball and the cup, measuring the execution error to calculate the reward for the PoWER algorithm. The whole view of the hardware setup for experiments is shown in Fig. 4.

Fig. 4. The initialization of the test platform for ball-in-cup manipulation.

We use Robot Operating System (ROS) as the middleware to communicate KUKA robot controller, the RGB-D camera with the learning algorithms. In the experiments, we force the end-effector of the manipulator within only x-axis and z-axis while locking the other dimensions.

The error of the execution of a trajectory is defined as the distance between the center of the ball and cup when $t = t_c$, the moment the cup catches or misses the ball as shown in Fig. 5. Then the reward can be defined as $r(t_c) = e^{-\alpha d^2}$ and $r(t) = 0$ for all $t \neq t_c$. The α is chosen as 0.01 if d is counted as millimeter.

We demonstrate the ball-in-cup manipulation in the x-z plane to the robot with kinesthetic teaching. The duration of the movement is 3.5 seconds. A DMP with 17 kernels for each dimension of the movement learns the skill and results in the initial policy with 34 parameters. Afterward, a random vector sampled from a multi-dimensional Gaussian distribution will be added to the policy parameters as the exploration.

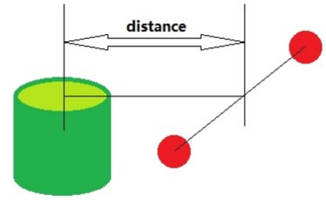

Fig. 5. Definition of distance between the ball and the cup.

The desired trajectories can be generated with the policy disturbed by both uncorrelated and correlated exploration. However, the trajectories generated by uncorrelated exploration may have potential difficulties when applied to the real robot. The perturbations from the neighbouring policy parameters may cancel out each other and sometimes a trajectory with excessive accelerations is produced due to some elements of the exploration rate grow dramatically during learning [1]. The output trajectory $x(t)$ in Fig. 2 will be far away from the desired one $x_d(t)$. It is not only a potential danger to the robot hardware, also the learning algorithms cannot work if $x(t)$ fails to track $x_d(t)$ within a reasonable range of errors. In [7], the experiments learning the ball-in-cup manipulation also indicate that PoWER with uncorrelated exploration results in an extremely low learning efficiency.

Eventually, in our experiments, both the previous and our correlated exploration methods can help the robot to learn the ball-in-cup manipulation skill. The policy is redeemed ideally optimal if the robot can catch the ball for the continuous 5 rollouts. Fig. 6 and Fig. 7 demonstrate the learning curves showing that our method produces a much higher sample efficiency than the previous method in the ball-in-cup manipulation task. For the previous method, we use the $\beta_k R^{-1}$ as the exploration rate to generate the correlated exploration. The hyperparameter β_k is defined as

$$\beta_k = \beta_0 \left(\frac{1 - \tanh\left(\frac{k-60}{30}\right)}{2} \right)$$

by some pre-experiments. The k denotes the rollout number. The reason to select the $\tanh$ function is to make sure that the robot explores drastically at first for sufficient rollouts, while narrows down the search by decreasing the value of β_k gradually. This method is simple but has no guarantee to find the policy optimum. The exploration rate may become zero before the policy is sufficiently improved and then the robot will never catch the ball. In our experiments, only a probability of 50% for the policy to converge to the ideal optimum with this β_k, while our method can find the ideal optimum every time. The β_k can also be defined as performance related [7]. However, the pre-experiments are still required and the sample efficiency will still not be higher than the method of maximizing the lower bound.

The numbers of rollouts for importance sampling in our method are 10 for updating the policy and 20 for updating the exploration rate. The experiments are conducted with several initial policies, including the ones making the ball fly too far

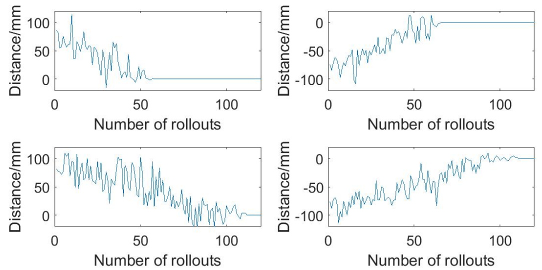

Fig. 6. This figure selects 4 successful runs of robot learning ball-in-cup manipulation with PoWER. The upper 2 plots update the exploration rate with our method while the others update with $\beta_k R^{-1}$. The left 2 plots start learning with an initial policy with a performance of $d \approx 80mm$ while the others start with $d \approx -80mm$.

or too shortly. As shown in Fig. 7, the average return starts to be improved significantly at around 20 rollouts and it takes around 60 rollouts for the robot to converge to the ideal optimum with our method. While the previous method with $\beta_k R^{-1}$ should require around 60 rollouts to substantively improve and converge at around 110 rollouts if lucky.

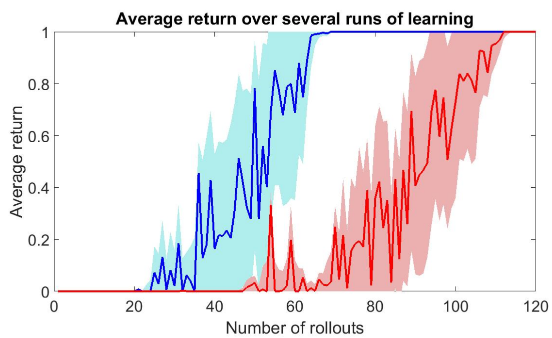

Fig. 7. The average return of the policy over 5 runs of learning the ball-in-cup manipulation with our method (blue) and 5 successful runs with the method of $\beta_k R^{-1}$ (red), excluding another 5 unsuccessful runs.

V. Conclusion

In this paper, we propose a model which allows PoWER or other learning algorithms with similar updating framework to plan smooth trajectories for the real robot without state-dependent exploration or surpassing the physical capabilities of the robot. Convergence is ensured and the sample efficiency is improved by applying our method to ball-in-cup manipulation. It enhances the policy search algorithms by making the covariance matrix changes with the learning process. Due to shorter training time, the algorithm results in lower training cost when applied to physical robot. The learning efficiency is also robust to the selection of the initial exploration rate in our experiments.

One drawback of this method is that it only works for episodic learning because Equation (8) cannot be derived if the exploration is time-variant. Nevertheless, the robot is still not intelligent enough since it takes a human surely much less than 60 rollouts to learn the ball-in-cup task. All exploring methods use Gaussian distribution by default, changing which may further improve the simple efficiency.

Acknowledgment

The authors thank the National Research Foundation, Keppel Corporation and National University of Singapore for supporting this work done in the Keppel-NUS Corporate Laboratory. The conclusions put forward reflect the views of the authors alone, and not necessarily those of the institutions within the Corporate Laboratory.

References

[1] J. Kober and J. R. Peters. Policy search for motor primitives in robotics. *Machine Learning*, 84:171–203, 2011.

[2] J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[3] J. Kober and J. R. Peters. Policy search for motor primitives in robotics. *Advances in neural information processing systems*, pages 849–856, 2009.

[4] E. Theodorou, J. Buchli, and S. Schaal. Reinforcement learning of motor skills in high dimensions: A path integral approach. *IEEE International Conference on Robotics and Automation*, pages 2397–2403, 2010.

[5] E. Theodorou, J. Buchli, and S. Schaal. Learning policy improvements with path integrals. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 828–835, 2010.

[6] F. Ficuciello, D. Zaccara, and B. Siciliano. Synergy-based policy improvement with path integrals for anthropomorphic hands. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1940–1945, 2016.

[7] J. Lundell, M. Hazara, and V. Kyrki. Generalizing movement primitives to new situations. *Conference Towards Autonomous Robotic Systems*, pages 16–31, 2017.

[8] J. Kober, E. Oztop, and J. Peters. Reinforcement learning to adjust robot movements to new situations. *Proc. Robot. Sci. Syst*, pages 33–40, 2010.

[9] Y. Chebotar, M. Kalakrishnan, A. Yahya, et al. Path integral guided policy search. *IEEE International Conference on Robotics and Automation*, pages 3381–3388, 2017.

[10] M. Kalakrishnan, S. Chitta, E. Theodorou, et al. Stomp: Stochastic trajectory optimization for motion planning. *IEEE International Conference on Robotics and Automation*, pages 4569–4574, 2011.

[11] M. Kalakrishnan, L. Righetti, P. Pastor, et al. Learning force control policies for compliant manipulation. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4639–4644, 2011.

[12] T. Rückstieß, M. Felder, and J. Schmidhuber. State-dependent exploration for policy gradient methods. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, Berlin, Heidelberg*, pages 234–249, 2008.

[13] B. D. Argall, S. Chernova, M. Veloso, et al. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[14] P. Pastor, M. Kalakrishnan, S. Chitta, et al. Skill learning and task outcome prediction for manipulation. *IEEE International Conference on Robotics and Automation*, pages 3828–3834, 2011.

[15] F. Stulp. Adaptive exploration for continual reinforcement learning. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1631–1636, 2012.

[16] P. Pastor, H. Hoffmann, et al. Learning and generalization of motor skills by learning from demonstration. *IEEE International Conference on Robotics and Automation*, pages 763–768, 2009.

[17] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, et al. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373, 2013.

[18] P. Kormushev, S. Calinon, and D. G. Caldwell. Robot motor skill coordination with em-based reinforcement learning. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3232–3237, 2010.

[19] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.