2014 IEEE International Conference on Systems, Man, and Cybernetics
October 5-8, 2014, San Diego, CA, USA

Integration of Evolutionary Computing and Reinforcement Learning for Robotic Imitation Learning

Huan Tan, Kannan Balajee, DeRose Lynn
GE Global Research
General Electric
Niskayuna, USA
huan.tan@ieee.org, balajee.kannan@ge.com, derose@ge.com

***Abstract*—This paper proposes an evolutionary reinforcement learning method by combining Estimation of Distribution Algorithm and Reinforcement Learning. The Reinforcement Learning method in our method is based on Policy Improvement with Path Integrals (PI^2). Estimation of Distribution Algorithm is incorporated into this reinforcement learning method to improve the generation of roll outs with certain noises. This method can accelerate the converging of the learning results and improve the overall system performance. Additionally, this method provides a potential solution to integrate the exploratory evolutionary algorithms and the greedy policy learning method. The proposed method is applied in a robotic imitation learning experiment in this paper and the experimental results demonstrate the effectiveness and robustness of our proposed algorithm.**

Keywords*— *Robotics, Imitation Learning, Reinforcement Learning, Evolutionary Algorithm

I. INTRODUCTION

A typical policy model for robots to generate desired motions or control policies is described as shown below [1] [2]:

$$\pi = f(s, g, t) \quad (1)$$

A generalized policy π is determined by current state s including environmental state and robotic state, the target state g, and the temporal information t. Learning of such policies enables robots to perform different tasks with different requirements and to deal with dynamic and unstructured environment.

Robotic imitation learning (also called learning from demonstration, learning by programming, etc.) enables robot to learn behaviors and skills from humans quickly. Current research on imitation learning can be categorized into two types [3]: one is trying to train robots to extract and learn the motion dynamics[4], and the other is trying to train robots to learn higher-level behaviors and action primitives through imitation [5] [6]. Such learning methods, especially the first type, provide a possible solution for researchers and engineers to program motions for high Degrees-Of-Freedom robots, especially humanoid robots.

An important skill for intelligent robots is to learn behaviors or skills through trials like human children. Some researchers applied reinforcement learning robots to learn desired knowledge by utilizing rewards from iterative trials. A typical flowchart for applying reinforcement learning in robotic imitation learning is shown below:

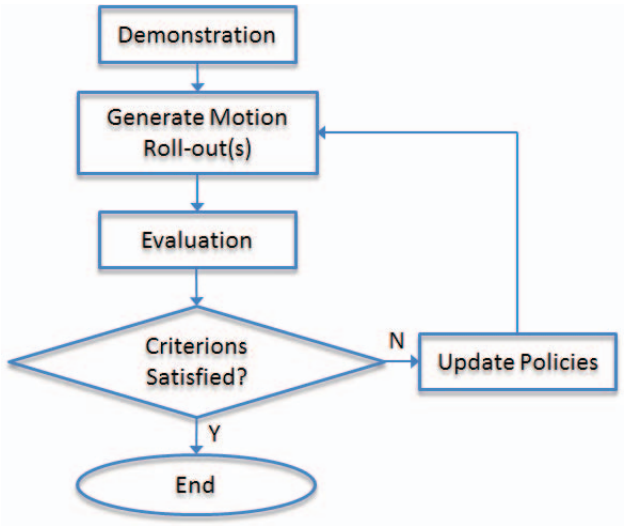

Fig.1. Reinforcement Learning Flowchart

The basic idea of reinforcement learning algorithm is to evaluate a current policy using some cost functions and to update the current policy according to these values from cost functions. Reinforcement learning is very attractive to researchers in robotic imitation learning community because it illustrates a way of learning newly knowledge through trials and such algorithms are approved to be effective for robotic imitation learning. For example, Atkeson applied reinforcement learning to teach a robot to swing up and balance a pendulum [7]. Peters used several reinforcement learning methods to train robots to learn the control policies [8]. Theodorou proposed applying reinforcement learning

978-1-4799-3840-7/14/$31.00 ©2014 IEEE

methods in optimal control to teach robots to learn and generate motion trajectories [9].

Generation of motion roll-outs in reinforcement learning for robotic imitation learning is based on a current policy of the system. So can robots use some exploratory methods to genera roll-outs and find the distribution of the parameters which can better fit the requirements of learning of the motion models? To address that problem, we try to seek solutions in evolutionary computing domain.

Evolutionary computing methods have been broadly applied in searching solutions, which is normally represented as bit strings, through an evolutionary process [10]. Through "crossing-over" parents selected from previous generation of population, which have high fitness function values, off-spring solutions gradually move to a target solution.

Fig.2 displays a general procedure of applying genetic algorithm in robotic imitation learning. By comparing Fig.1 and Fig.2, it is obvious that the basic idea of evolutionary computing is similar to reinforcement learning. The improvement of solutions (policies, parameters, etc.) in both of the two types of algorithms is based on evaluation of current solutions.

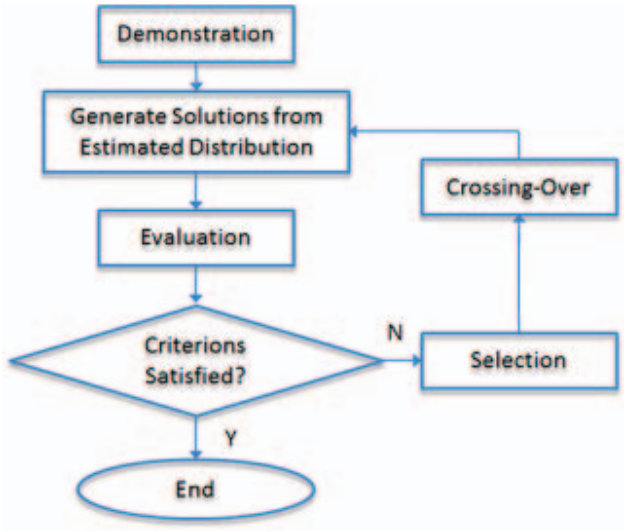

Fig.2. Evolutionary Computing Algorithm

Since similarity exists between reinforcement learning and the evolutionary computing the motivation of this paper is to integrate reinforcement learning with evolutionary computing algorithms to provide an exploratory robotic imitation learning method. Some researchers have applied the evolutionary computing methods to robotic imitation learning, e.g., Genetic Algorithm [11] [12], Genetic Computing [12], etc. However, directly applying traditional computing methods is not very effective partially because the optimization based operations on bit-level is time-consuming.

Recently, the Estimation of Distribution Algorithm (EDA) is proposed as a more robust method [13] [14] [15]. A general procedure of EDA is: Firstly, this probabilistic method computes the distribution of each chromosome in the individuals which have higher evaluation function values. Secondly, the corresponding chromosome is generated from the distribution by probabilistic sampling. The EDA provides a much more robust evolutionary method for finding a solution [15]. By replacing the representation of a population of parent solutions with distributions of the chromosomes, EDA largely reduce the workload on computing burden and provide more adaptive and flexible solutions to a large amount of practical solutions. Moreover, in this paper we also further extend the original idea of EDA algorithms on bit-operation to parameter-operation. Through this extension, evolutionary computing methods could be applied to more practical problems.

The rest of this paper is organized as follows: Section II summarizes the related work on Policy Improvement with Path Integrals (PI^2) Learning; Section III introduces our methodology of applying an EDA-based evolutionary reinforcement learning algorithm for robotic imitation learning; Section IV explains a typical robotic imitation learning experiment and discusses the results; Section V summarizes this paper and proposes the future work.

II. Related Work

Recently, PI^2 algorithm attracts the attention of researchers from both the machine learning and the robotic imitation learning. Originating from optimal control based on the stochastic Hamilton-Jacobi-Bellman (HJB) equations, PI^2 computes the cost values from the integrals along motion trajectories and improve policies based on the probabilistic distribution over the generated roll-outs. Policy Improvement with Path Integrals (PI^2) offers currently one of the most efficient, numerically robust, and easy to implement algorithms for RL based on trajectory roll-outs.

Given a general stochastic control system:

$$\dot{\boldsymbol{x}}_t = \boldsymbol{f}_t + \boldsymbol{G}_t(\boldsymbol{u}_t + \boldsymbol{\varepsilon}_t) \quad (2)$$

we can define some important issues in the learning algorithm:

- An immediate cost function $r_t = q_t + \boldsymbol{\theta}_t{}^T \boldsymbol{R}\boldsymbol{\theta}_t$
- A terminal cost term Φ_{tN}
- A stochastic parameterized policy $\boldsymbol{a}_t = \boldsymbol{g}_t{}^T(\boldsymbol{\theta} + \boldsymbol{\varepsilon}_t)$
- The basis function $\boldsymbol{g}_{t_i}$ from the system dynamics
- The variance Σ_ε of the mean-zero noise $\boldsymbol{\varepsilon}_t$
- The initial parameter vector $\boldsymbol{\theta}$

Based on these definitions, we can use a reinforcement learning algorithm to teach robot to generate a state path similar to demonstrations.

The pseudo code of the PI^2 algorithm is displayed in Fig.3. The general idea of PI^2 algorithm is to compute the cost over the generated motion trajectory and find the relationship between the cost from path integrals and the updates on parameters. This method takes the overall rewards over the generated paths into consideration. Then the computed updates can reflect the overall contribution from each parameter. This method is very effective and reliable.

The original pseudo code of EDA algorithm is shown in Fig.4. The major contribution of EDA is to extend the original

idea of representing the solutions as a population of bit-strings to find the distribution over the bits in "good" solutions.

Repeat until convergence of the trajectory cost $\boldsymbol{R}$:

1 Generate K roll-outs of the system from the same start $\boldsymbol{x}_0$ using stochastic parameters $\boldsymbol{\theta} + \boldsymbol{\varepsilon}_t$ at every time step

2 For $k = 1, \dots, K$, compute:

$$P(\tau_i, k) = \frac{e^{-\frac{1}{\lambda}S(\tau_i,k)}}{\sum_{k=1}^{K} e^{-\frac{1}{\lambda}S(\tau_i,k)}} \quad (4)$$

$$S(\tau_i, k) = \Phi_{t_N,k} + \sum_{j=i}^{N-1} q_{t_j,k} + \frac{1}{2}\sum_{j=i+1}^{N-1} \left(\boldsymbol{\theta} + \boldsymbol{M}_{t_j,k}\varepsilon_{t_j,k}\right)^T \boldsymbol{R}\left(\boldsymbol{\theta} + \boldsymbol{M}_{t_j,k}\varepsilon_{t_j,k}\right) \quad (5)$$

$$\boldsymbol{M}_{t_j,k} = \frac{\boldsymbol{R}^{-1}\boldsymbol{g}_{t_j,k}\boldsymbol{g}_{t_j,k}^T}{\boldsymbol{g}_{t_j,k}^T\boldsymbol{R}^{-1}\boldsymbol{g}_{t_j,k}} \quad (6)$$

3 For $i = 1, \dots, (N-1), compute$

$$\delta\theta^{[j]}{}_{t_i} = \sum_{k=1}^{K} P(\tau_i, k)\boldsymbol{M}_{t_j,k}S(\tau_i, k) \quad (7)$$

4 Compute

$$\delta\theta^{[j]} = \sum_{k=1}^{K} \frac{\sum_{i=0}^{N-1}(N-i)w_{j,t_i}\delta\theta^{[j]}{}_{t_i}}{\sum_{i=0}^{N-1}(N-i)w_{j,t_i}} \quad (8)$$

5 Update

$$\theta = \theta + \delta\theta \quad (9)$$

6 Create one noise less roll-out to check the trajectory cost $R = \Phi_{t_N} + \sum_{i=1}^{N-1} r_{t_j}$. In case the noise cannot be turned off, that is, a stochastic system multiple roll-outs need be averaged.

Fig.3. Pseudo Code of PI^2

1 Generate the initial population P (initial) with N individuals (paths) and set it as the current population

2 Repeat until the termination criterion is achieved

a. Evaluate the current population

b. Select M individuals ($M < N$) from the current population based on some criterions

c. Compute the distribution over the bits the selected individuals $p(x|\boldsymbol{D}^{se})$

d. Generate N individuals (paths) from the obtained distribution

End

3 Return the best individual in the current population

Fig.4. Pseudo Code of EDA

In order to adapt the EDA algorithm to robotic imitation learning, we also extend the traditional EDA to a probabilistic evolutional computing algorithm. Then the path generation is also stochastic which can be integrated with PI^2.

III. Methodology

As stated earlier, in order to adapt EDA to parameterized policy learning, we need to further extend the concepts in EDA. In this paper, we propose that we could find the distribution over on parameters not over bits in the parameters.

We extend the estimation of the distribution over bits: $p(x|\boldsymbol{D}^{se})$ to estimate the distribution over the overall individual (parameter) $p(\boldsymbol{x}|\boldsymbol{D}^{se})$.

In PI^2, cost values are directly used to update the parameterized policies. In our method, we obtain the cost values through path integrals but not use these cost values to update the policies directly. In order to integrate EDA, we use these cost values for selection and estimation.

Given the same problem definition in Section 2 and combine the extension of EDA and the utilization of the PI^2, we propose our algorithm as shown in Fig.5.

In our method, we arbitrarily assume that the probabilistic distributions of the parameters are Gaussian-based. Thus, the means of these Gaussian distributions reflect the optimum estimation of the parameters. The initialization step is flexible and researchers can specify the distributions with any choices. Normally, we select the means as 0 with stand deviation 1.

The algorithm is composed of two parts. The first part is based on the PI^2 algorithm which computes the updates from roll-outs, the second part is to select "better" individuals from the generated population and to estimate the distribution over the selected individuals.

A. Part 1

The step a, sampling from the distributions in iterations, is trivial. A used technique used in our system is to use limiter to confine the deviation between the generated parameters and the estimated mean from the last iteration. The second step is to compute the deviation between the generated parameters and the estimated mean from the last iteration. Using these parameters, K roll-outs can be generated from the given system defined in last sectioin. Step d is used to compute the cost values and the probabilities of the paths. This method is based on the PI^2, which can compute the overall contribution of the generated parameters to the over the generated paths. Step e computes the updates according to the path-integrals. Step f and step g are used to evaluate the estimation or learning results in iterations. A noiseless roll-out is generated by using the means of the probabilistic models of the parameters. If cost value of the roll-out is smaller than a threshold value, we considered that the termination criterion is met and system stops iteration. If the cost value is larger than a threshold value, the iteration repeats from step 1. However, in some cases, if the cost values converge in a small range, the system also stops iterations.

The steps in part 1 are repeated R times to generate a population of R individuals. Then the algorithm turns to the EDA part.

Initialization:

Arbitrarily assign the distributions over the n parameters in the policy:

$$\theta_1(0) \sim N(\boldsymbol{x}_1(0), \sigma_1(0)^2), \dots, \theta_n(0) \sim N(\boldsymbol{x}_n(0), \sigma_n(0)^2) \quad (3)$$

Iterations:

Repeat until the termination criterion is achieved or the cost values converge:

1 *Evaluation*

Repeat R Times from step a to step e

a. Generate K groups of new stochastic parameters by sampling the distributions obtained from the last iteration

b. Find the deviation of these parameters from the mean of the original estimated distribution, $\boldsymbol{\varepsilon}_t$

c. Create K roll-outs of the system from the same start state x_0 using these K groups of stochastic parameters at every time step

d. Compute the cost and the probability of these roll-outs:

$$\boldsymbol{M}_{t_j,k} = \frac{\boldsymbol{R}^{-1}\boldsymbol{g}_{t_j,k}\boldsymbol{g}_{t_j,k}^{T}}{\boldsymbol{g}_{t_j,k}^{T}\boldsymbol{R}^{-1}\boldsymbol{g}_{t_j,k}} \quad (4)$$

$$S(\tau_i,k) = \Phi_{t_N,k} + \sum_{j=i}^{N-1} q_{t_j,k} + \frac{1}{2}\sum_{j=i+1}^{N-1}\left(\boldsymbol{\theta} + \boldsymbol{M}_{t_j,k}\varepsilon_{t_j,k}\right)^{T}\boldsymbol{R}\left(\boldsymbol{\theta} + \boldsymbol{M}_{t_j,k}\varepsilon_{t_j,k}\right) \quad (5)$$

$$P(\tau_i,k) = \frac{e^{-\frac{1}{\lambda}S(\tau_i,k)}}{\sum_{k=1}^{K} e^{-\frac{1}{\lambda}S(\tau_i,k)}} \quad (6)$$

e. Compute the updates for the parameters

$$\delta\theta^{[j]}{}_{t_i} = \sum_{k=1}^{K} P(\tau_i,k)\boldsymbol{M}_{t_j,k}S(\tau_i,k) \quad (7)$$

$$\delta\theta^{[j]} = \sum_{k=1}^{K} \frac{\sum_{i=0}^{N-1}(N-i)w_{j,t_i}\delta\theta^{[j]}{}_{t_i}}{\sum_{i=0}^{N-1}(N-i)w_{j,t_i}} \quad (8)$$

$$\theta = \theta + \delta\theta \quad (9)$$

f. Generate one roll-out using the means of the distribution of parameters

g. Compute the cost value

2 *Estimation*

a. Select $M(M < R)$ solutions with higher cost value functions from all the generated roll-outs as $\boldsymbol{D^{se}}$

b. Compute customed distributions of the parameters of the selected solutions, $p(\theta_1|\boldsymbol{D^{se}}), p(\theta_2|\boldsymbol{D^{se}}), \dots, p(\theta_n|\boldsymbol{D^{se}})$

c. Generate one roll-out using the means of the distribution of parameters

d. Compute the cost value

Fig.5 Proposed Algorithm

B. Part 2

At step a, we select the M "best" policies by finding the paths with the smallest cost values. (Please be noticed that we have K candidate policies.) This step guarantee that the improvement of the policies when iterations happen. At step b, the means and the standard deviations of the parameters are computed to construct Gaussian models. We want to point out that this step is the basic essence of EDA. Instead of putting a group of candidate policies here, the idea of EDA is to represent these policies as specific probabilistic models. However, the selection of the stochastic models is flexible. Researchers could use any models that could fit the problem domain.

Step c and step d are used to evaluate the estimation or learning results in iterations using a noiseless roll-out generated by using the means of the probabilistic models of the parameters. Similar to step f and step g in part 1, any criterion satisfied signal would terminate the iterations.

From the explanation above, it is obvious that PI^2 and EDA are integrated to provide a probabilistic exploratory searching in reinforcement learning algorithms. One important modification of EDA in this method is to extend the estimation of distribution over bits to individuals. This extension can save time spending on computing and moreover release the problem of converging at local optimum point.

This method could be applied to not only the robotic imitation learning domain but also other domains which can describe systems in equation, which is well accepted.

Applying this method in robotic imitation learning require developing a nonlinear system following the form of equation (2), which can generate motion paths.

In this paper, we choose Dynamic Movement Primitive (DMP) which perfectly math the form of equation (2). DMP is proposed by Ijspeert in 2002 [8].

The formulation of the DMP algorithm is shown as differential equations:

$$\tau\dot{z} = \alpha_z(\beta_z(g - y) - z) \quad (10)$$

$$\tau\dot{y} = z + f \quad (11)$$

f is a Receptive Field Weighted Regression (RFWR) model [16]:

$$f = \frac{\sum_{i=1}^{N}\Psi_i\theta_i v}{\sum_{i=1}^{N}\Psi_i} \quad (12)$$

where

$$\Psi_i = exp\left(-h_i\left(\frac{x}{g} - c_i\right)^2\right) \quad (13)$$

Ψ_i are Gaussian basis function with a center c_i and a bandwidth h_i. θ_i, also considered as primitive in policies, is the parameter we want to teach the robot to learn. Given a demonstration, robot needs to learn these parameterized policies through iterations.

IV. Experimental Results

In order to test and evaluate our proposed algorithm, we obtain the Matlab simulation package from the website of the authors who originally proposed PI^2 algorithm [17]. The simulation package is modified according to our proposed algorithm in this paper.

The nominal movement is designated from $\boldsymbol{q} = [0,0]^T$ to $\boldsymbol{q} = [1,1]^T$. In other words, the demonstration is arbitrarily designed. The cost function is:

$$r_t = 0.5\ddot{\boldsymbol{q}}_t{}^T \ddot{\boldsymbol{q}}_t Q + 0.5\boldsymbol{\theta}^T \boldsymbol{\theta} R$$

with $Q = 1000, R = 1$.

In our experiments, all the initialized parameters are the same and updates are computed 200 times in part 1.

Fig.6 displays the generated motion trajectories drawn in red lines and the demonstrated motion trajectories in black lines. The generated motion trajectories are almost the same as the desired motion trajectories. They are almost overlapped. The experimental learning results demonstrate that our algorithm can enable robots to learn the demonstrations and generate similar motion trajectories.

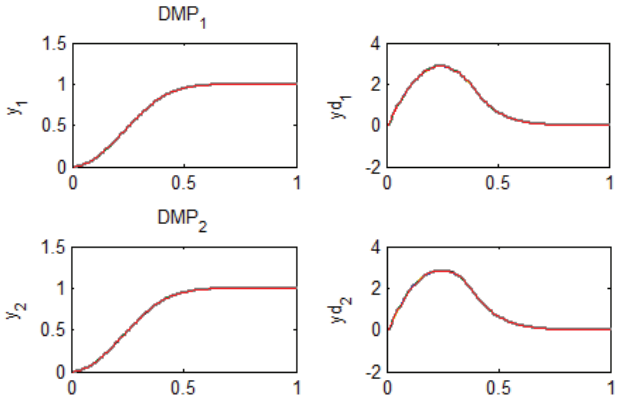

Fig.6. Learning results of the Proposed Algorithm

Fig.7 displays the learning curve using the proposed algorithm. In Fig.7, the cost function value drops and achieves the convergence range very quickly. This demonstrates that our algorithm is effective. The fast convergence can enable robots to learn motion quickly which is required in robotic imitation learning. Meanwhile, unlike some learning algorithms, our proposed algorithm enables the robustness of the convergence. In Fig.7, the cost function values keep low in a range which means the error between the demonstrated and the generated motion trajectory is very small. The values keep decreasing as the number of trials go up, which illustrates the robustness of the learning result. By incorporating the reinforcement learning and the evolutionary computing, robots can quickly find solutions with higher cost function values in an area. Unlike traditional reinforcement learning algorithms, which try to update the parameterized model gradually by analyzing the rewards, our proposed algorithm enables robots to find the solution in a broader area to avoid local optimum.

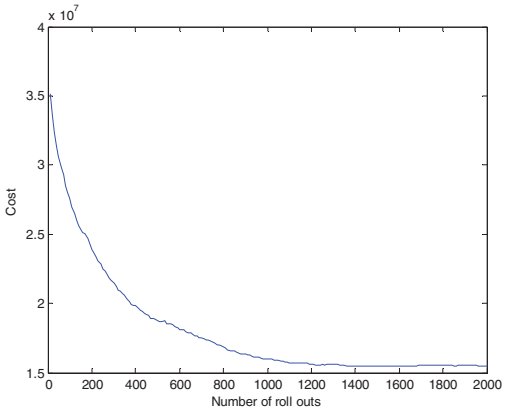

Fig.7. Learning Curve of the Proposed Algorithm

V. Discussion and Future Work

We cannot expect that robots can find the parameters by themselves. However, we can teach robots to learn these parameters through trials, which is the essence of reinforcement learning. In reinforcement learning, updates are computed from rewards, which is reasonable. A novel idea is to enable robots to learn parameters through trials, and also to find solutions in the configurations space through explorations. In this paper, the algorithm we proposed is attempting to solve such problems.

From the experimental results, we can see that our proposed algorithm is effective and reliable. In robotic imitation learning, many researchers have proposed various algorithms for robots to understand demonstrated behaviors or skills from humans and most of the algorithms focus on how to find a model to describe demonstrations. Using a parameterized motion primitive-based model enables us to describe the knowledge and skills using simple mathematical models. Another advantage of using such method is that it could be applied in many machine learning algorithms.

Reinforcement learning provides a robust and powerful tool for robots to learn knowledge and skills through trials, like we, humans, learn new stuff from trainings every day from our child hood. However, sometimes, learning could be trapped in a local minimum point or the convergence speed is not satisfying. Similar to the "jumping" operation in Hill-Climbing Searching algorithm, the evolutionary algorithm provides a possible solution to solve some problems in reinforcement learning. Moreover, evolutionary computing methods can expand the target configuration space more reasonably because of the computing of the distribution over the learning results with higher cost function values.

In 2011, we proposed a method for robots to integrate learning from demonstration with self-exploration [18]. That method enables robots to find a solution in some situations. The algorithm proposed in this paper provides another method for robots to learn complex parameterized motion primitives, which can be applied in various domains. In the future, we

also plan to apply our proposed algorithm in robotic manipulator control and other domains. Meanwhile, we plan to investigate using different probabilistic distribution models for the estimation part in our proposed algorithms so that this algorithm could be applied to more practical domains.

A crucial point is to compare our algorithm with other reinforcement learning methods. We will compare algorithm with other popular algorithms and the comparison should be based on quantitative measurement of system performances.

Another important issue we want to pay attention to is to integrate with lower level evolutionary reinforcement learning algorithms with higher level cognitive processes. Similar to what humans do, a robot needs to make a decision when the evolution needs to be taken into consideration. Such a decision making process could provide a more robust and reliable method for robots to learn new knowledge and skills.

The proposed algorithm is only tested to generate similar motion trajectories in this paper. In the future, we plan to use this algorithm to teach robots to learn decision-making models, environmental models, sensor-motor skill models, etc. As stated earlier in this paper, if we could find a method to describe the models mentioned above using parameterized mathematical models, the proposed algorithm can be quickly modified to solve these problems, depending on specific requirements in various domains and task-relevant situations. A more challenging problem is to apply the proposed algorithm in semantic learning problems. It is not expected that all semantic models could be represented as parameterized mathematical models as we described in this paper. But, it is possible to apply the basic idea of evolutionary reinforcement learning in such learning problems. We expect that our algorithm can be combined with Natural Language Processing (NLP) technologies to solve such semantic learning problems. Some researchers in NLP community try to represent the semantic models using probabilistic methods. So our plan is to teach robots to learn such probabilistic semantic models from evolutionary reinforcement learning and parameterize the learning results.

VI. CONCLUSION

This paper proposes a novel evolutionary reinforcement learning method and applies it to robotic imitation learning, which integrates EDA and PI^2 learning algorithm. This algorithm provides a solution to integrate exploratory learning methods with traditional reinforcement learning algorithms. The experimental results demonstrate that our algorithm could enable faster convergence than original PI^2 algorithm due to the integration of evolutionary computing methods. Meanwhile, the robustness of the learning results is guaranteed by the reinforcement learning part. By integrating these two parts, we obtain a fast and reliable evolutionary reinforcement learning algorithm.

The work described in this paper can be applied not only in learning and generating motion trajectories but also in other domains where the problems to be solved could be described as a well-known nonlinear state system as described in this paper. In the last section, we also propose that this algorithm can be applied to some domains that the parameterized model is not applicable.

REFERENCES

[1] An, M., Taura, T. And Shiose, T, "A Study On Acquiring Underlying Behavioral Criteria for Manipulator Motion By Focusing on Learning Efficiency", *IEEE Transactions on Systems, Man and Cybernetics*, Part A , vol.37, pp.445-455, 2007.

[2] Arikan, O. And Forsyth, D., "Interactive Motion Generation from Examples". *ACM Transactions on Graphics*, vol.21, pp.483-490, 2002.

[3] Atkeson, C., Moore, A. And Schaal, S. "Locally weighted learning". *Artificial Intelligence Review*, vol.11, pp.11-73, 1997.

[4] Atkeson, C. And Schaal, S., "Learning Tasks from a Single Demonstration". *Proceeding of the 1997 IEEE International Conference on Robotics and Automation*, Albuquerque, New Mexico, USA, pp.1706-1712, 1997.

[5] Calinon, S., Guenter, F. And Billard, A., "On Learning, Representing, and Generalizing a Task in a Humanoid Robot", *IEEE Transactions on Systems, Man, and Cybernetics*, Part B, vol.37, pp.286-298, 2007.

[6] Dillmann, R., Kaiser, M. And Ude, A., "Acquisition of Elementary Robot Skills from Human Demonstration". *Proceeding of the 1995 International Symposium on Intelligent Robotic System*, Citeseer, Pisa, Italy, pp.185-192, 1995.

[7] Hauschild, M. And Pelikan, M., "An Introduction and Survey of Estimation of Distribution Algorithms", *Swarm and Evolutionary Computation*, 2011.

[8] Ijspeert, A., Nakanishi, J. And Schaal, S., "Learning Attractor Landscapes for Learning Motor Primitives", *Advances in Neural Information Processing Systems*, vol.15, pp.1523-1530, 2003.

[9] Larranaga, P. And Lozano, J.A., "Estimation of Distribution Algorithms: a New Tool for Evolutionary Computation". Springer, Netherlands, 2002.

[10] Lozano, J.A., "Towards a New Evolutionary Computation: Advances in the Estimation of Distribution Algorithms", Springer-Verlag, New York Inc, 2006.

[11] Mataric, M., Williamson, M., Demiris, J. And Mohan, A., "Behavior-Based Primitives for Articulated Control", *Proceeding of the 1998 International Conference on Simulation of Adaptive Behavior*, Cambridge, Massachusetts, USA, pp.165–170, 1998.

[12] Peters, J. And Schaal, S., "Reinforcement Learning For Parameterized Motor Primitives", Proceeding of the 2006 International Joint Conference on Neural Networks, pp.73-80, 2006.

[13] Schaal, S., "Learning from Demonstration", In Advances in Neural Information Processing Systems, M.J. M.C. Mozer, & T. Petsche Ed. MIT Press, Cambridge, MA, pp.1040-1046, 1997.

[14] Schaal, S. And Atkeson, C., "Learning Control in Robotics", *IEEE Robotics & Automation Magazine*, vol.17, pp.20-29, 2010.

[15] Theodorou, E., Buchli, J. And Schaal, S., "Reinforcement Learning of Motor Skills in High Dimensions: a Path Integral Approach", Proceeding of the 2010 IEEE International Conference on Robotics and Automation IEEE, Anchorage, Alaska, USA, 2397-2403, 2010.

[16] Whitley, D., "A Genetic Algorithm Tutorial", *Statistics and Computing*, vol.4, pp.65-85, 1994

[17] Huan Tan, Kazuhiko Kawamura, "A Computational Framework for Integrating Robotic Exploration and Human Demonstration in Imitation Learning", Proceeding of the 2011 IEEE International Conference on System, Man, and Cybernetics, Anchorage, Alaska, USA, pp.2501-2506, 2011

[18] http://www-clmc.usc.edu/Resources/Software