View-based Programming with Reinforcement Learning for Robotic Manipulation

Yusuke Maeda
Div. of Systems Research, Fac. of Engineering,
Yokohama National University
79-5 Tokiwadai, Hodogaya-ku, Yokohama
240-8501 JAPAN
Email: maeda@ynu.ac.jp

Takumi Watanabe
Seiko Epson Corp.

Yuki Moriyama
Dept. of Mech. Eng.,
Div. of Systems Integration,
Graduate School of Engineering,
Yokohama National University

***Abstract*—In this paper, we study a method of robot programming with view-based image processing. It can achieve more robustness against changes of task conditions than conventional teaching/playback without losing its general versatility. In order to reduce human demonstrations required for the view-based robot programming, we integrate reinforcement learning with the method. First we construct an initial neural network as a mapping from images to appropriate robot motions using human demonstration data. Next we train the neural network with actor-critic reinforcement learning so that it can work well even in task conditions that are not identical to those in the demonstrations. Our proposed method is successfully applied to pushing and pick-and-place tasks in a virtual environment.**

I. Introduction

Conventional teaching/playback is still popular in robot programming for its simplicity. It is dependent on only internal sensors of robots and applicable to various tasks. Moreover, it is very reliable as far as task conditions, e.g. initial pose of the manipulated object, do not change. However, it is impossible for a robot to adapt to nontrivial variations in the initial pose of the object or unexpected fluctuations in the pose of the object during manipulation.

Thus image-based detection of the object is also widely used to adapt variations in the pose of the object. Model-based image processing such as feature extraction and pattern matching is performed to obtain the pose of the object accurately. In such model-based methods, however, how to detect an object is specific to the object. Therefore it is cumbersome for operators to register object models with the detection system and the general versatility of this method is limited.

In order to achieve more robustness against changes of task conditions than conventional teaching/playback without losing its general versatility, a robot programming method, "view-based teaching/playback" was proposed [1]. It uses PCA (Principal Component Analysis) to perform image processing that is not specific to the target object. Such model-free approaches are called "view-based" or "appearance-based" [2]. Our view-based teaching/playback also uses the generalization ability of artificial neural networks to obtain robustness. It was applied to pick-and-place and pushing in a virtual environment. Using multiple human demonstrations in the teaching phase, a virtual robot moved an object successfully to the goal position in the playback phase, even from some initial positions different from those in the demonstrations.

In this paper, in order to reduce human demonstrations required for the view-based teaching/playback, we integrate reinforcement learning with the method. First we construct an initial neural network as a mapping from images to appropriate robot motions using human demonstration data. Next we update the neural network with actor-critic reinforcement learning [3] so that it can work well even in task conditions that are not identical to those in demonstrations. Our proposed method is successfully applied to box-pushing and pick-and-place tasks in a virtual environment consisting of a 3D dynamics simulator.

II. Related Work

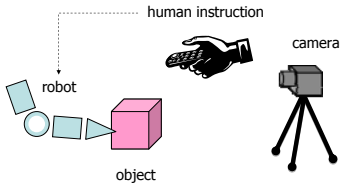
Shibata and Iida dealt with reinforcement learning of box pushing by a mobile robot with an artificial neural network in a view-based approach [4]. Kobayashi et al. proposed a vision-based method for reinforcement learning of robot motion with adaptive image resolution adjustment [5][6]; they applied it to pushing by a manipulator.

Both of the studies basically focused on reinforcement learning from scratch. On the other hand, from a practical point of view, reinforcement learning is used after supervised learning in our view-based programming.

III. Overview of View-based Robot Programming with Reinforcement Learning

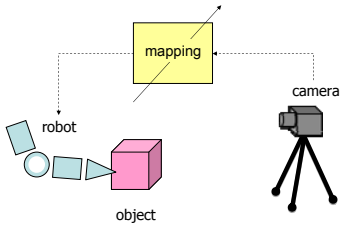
Our view-based robot programming method consists of the following steps:

1) View-based supervised learning from human demonstration.
   a) A human operator commands a robot to perform a manipulation task (Fig. 1a). All the motions of the robot are recorded. All the images of the teaching scenes are also recorded.
   b) A mapping from the recorded images to the motions is obtained as an artificial neural network (Fig. 1b). Basically the input of the neural network is a scene image, and its output is the desirable robot motion corresponding to the image.



(a) teaching by human operator

(b) obtaining mapping from image to robot motion



(c) playback with obtained mapping and reinforcement learning to update the mapping

Fig. 1. Outline of View-Based Robot Programming with Reinforcement Learning

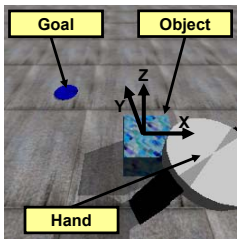


Fig. 2. Virtual Environment

2) View-based reinforcement learning for performance improvement.

   a) Manipulation in which the task condition is not identical to those in the human demonstrations is performed using the obtained mapping (Fig. 1c). The motion of the robot is determined by the output of the neural network calculated from scene images. Initially the motion may not be appropriate, but the mapping from images to robot motions is gradually updated with actor-critic reinforcement learning.
   b) As reinforcement learning goes on, the mapping is improved so that it can adapt to wider range of changes in task conditions.

Supervised learning, the first part of our method, is sufficient to play back human demonstrations and to adapt small range of changes in task conditions [1]. For more adaptability, however, we perform reinforcement learning, the latter part.

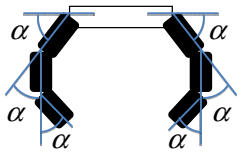


Fig. 3. Finger Bending

IV. Virtual Environment for Manipulation Tasks

We prepared a virtual environment for proof of concept of our view-based robot programming with reinforcement learning. A virtual robot hand (Fig. 2) is created on a dynamics simulator, ODE (Open Dynamics Engine) [7]. For simplicity, the hand has only five degrees of freedom: 3 DOF for translation (x, y and z directions), 1 DOF for rotation around its yaw axis (θ), and 1 DOF for finger bending. The hand has a "thumb" with three joints and an "index finger" with three joints. Bending of them is controlled with only one angle parameter, α as shown in Fig. 3.

In the teaching phase, the degrees of freedom of the hand are PD-controlled based on human keyboard commands. Thus a human operator can drive the virtual robot hand and manipulate an object (a cube) in the virtual environment. Scenes of the virtual environment can be obtained as grayscale camera images of 256×256 pixels.

V. View-based Supervised Learning

In our method, a human operator commands a robot with a keyboard to perform a manipulation task. Here we call it *demonstration*. The pairs of the movement of the robot ($\Delta \boldsymbol{x}$) and the scene image were recorded throughout the demonstrations.

A scene image $\boldsymbol{I}$ is composed of numerous pixel data and therefore it is not realistic to use raw pixel data as the input of the neural network. Here we use PCA (Principal Component Analysis) for all the recorded images as view-based image compression. Even a small number of factor scores [8] can reconstruct the original image approximately as follows:

$$\boldsymbol{I} = \boldsymbol{I}_{\text{avg}} + \sum_{i=1}^{N_{\text{pixel}}} w_i \boldsymbol{u}_i \approx \boldsymbol{I}_{\text{avg}} + \sum_{i=1}^{N_{\text{PC}}} w_i \boldsymbol{u}_i, \qquad (1)$$

where $\boldsymbol{I}_{\text{avg}}$ is the average of images in the demonstration, N_{pixel} is the number of pixels of the camera image, w_i are factor scores, $\boldsymbol{u}_i$ are principal components, and N_{PC} is the number of principal components to be used for approximation ($N_{\text{PC}} \ll N_{\text{pixel}}$). Thus we use the factor scores, $w_1, \ldots, w_{N_{\text{PC}}}$, as the input of the neural network instead of the raw pixel data.

We use a feedforward neural network as shown in Fig. 4 for supervised learning. It is three-layered and has N_{hidden} neurons in its hidden layer. The input of the neural network is the current state (at time t), $\boldsymbol{s}_t = [\boldsymbol{w}_t, \boldsymbol{a}_{t-1}]$, where $\boldsymbol{w}_t = [w_{t1}, \ldots, w_{tN_{\text{PC}}}]$ is factor scores for the first N_{PC} principal components of camera images, and $\boldsymbol{a}_{t-1} = \Delta \boldsymbol{x}_{t-1}$ is the action at the previous time step. The output of the neural

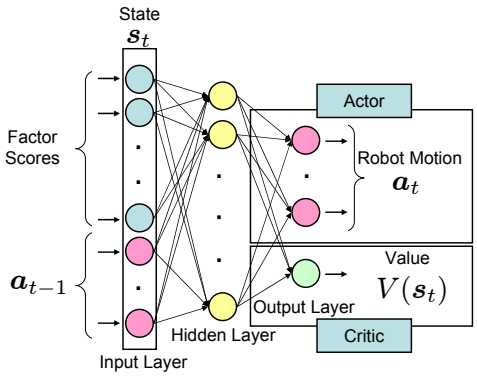


Fig. 4. Neural Network for Mapping from Images to Robot Motions

network is $[\boldsymbol{a}_t, V(\boldsymbol{s}_t)]$, where $\boldsymbol{a}_t = \Delta\boldsymbol{x}_t$ is the action of the robot and $V(\boldsymbol{s}_t)$ is the state value. Thus the neural network is composed of two parts: mapping from states to robot motions ("actor") and mapping from states to state values ("critic"). The latter is added for actor-critic reinforcement learning, which is described in the next section.

We train the neural network with backpropagation with momentum (BPM) [9]. The training signals for the input of the neural network are states ($\boldsymbol{s}_t$) in the demonstration. The training signals for the actor part of the output of the neural network are robot motions ($\boldsymbol{a}_t$) in the demonstration. The training signals for the critic part of the output are state values ($V(\boldsymbol{s}_t)$), which are calculated as follows:

$$V(\boldsymbol{s}_T) = 0 \tag{2}$$

$$V(\boldsymbol{s}_{t-1}) = \gamma V(\boldsymbol{s}_t) + r_t, \tag{3}$$

where T is the time step when the demonstration reached the goal; γ is a discount factor; r_t is a reward at time t, which is described in detail in Section VI-B.

The trained neural network (the actor part) can be used to control the robot hand to play back the demonstrations [1].

VI. View-based Reinforcement Learning

A. Actor-Critic Reinforcement Learning

Here we present the details of our view-based reinforcement learning to improve the neural network obtained in the supervised learning. It is based on actor-critic used in [4] to explore a continuous parameter space.

The TD (temporal-difference) error [3] is given as follows:

$$\mathrm{TD}_{\mathrm{error}} = r_{t+1} + \gamma V(\boldsymbol{s}_{t+1}) - V(\boldsymbol{s}_t). \tag{4}$$

In order to reduce the TD error, we adjust the neural network to modify the output of the critic. The desired output of the critic T_c for $\boldsymbol{s}_t$ is determined as follows:

$$T_c(\boldsymbol{s}_t) = V(\boldsymbol{s}_t) + \beta(\mathrm{TD}_{\mathrm{error}}), \tag{5}$$

where $\beta(>0)$ is a constant parameter.

Reinforcement learning requires exploration and therefore we have to modify the actor output. In our method, the actual robot motion $\boldsymbol{a}'_i(\boldsymbol{s}_t)$ in reinforcement learning is generated as follows:

$$\boldsymbol{a}'_i(\boldsymbol{s}_t) = \boldsymbol{a}_i(\boldsymbol{s}_t) + \boldsymbol{R}_t + \boldsymbol{R}_e, \tag{6}$$

where $\boldsymbol{a}_i(\boldsymbol{s}_t)$ is the output of the actor, and $\boldsymbol{R}_t$ and $\boldsymbol{R}_e$ are random vectors from Gaussian distribution for exploration calculated as follows:

$$\boldsymbol{R}_t = [R_{t1}, \ldots, R_{tn}]^T \tag{7}$$

$$R_{ti} = \sqrt{-2\sigma_i^2 \log \mathrm{rand}()}\sin(2\pi \cdot \mathrm{rand}()) \tag{8}$$

$$\boldsymbol{R}_e = [R_{e1}, \ldots, R_{en}]^T \tag{9}$$

$$R_{ei} = \sqrt{-2\sigma_{ei}^2 \log \mathrm{rand}()}\sin(2\pi \cdot \mathrm{rand}()), \tag{10}$$

where $\mathrm{rand}()$ is a uniform random value in [0,1], and σ_i^2 and σ_{ei}^2 are variances of the Gaussian distribution, which are described in Section VI-B. Note that $\boldsymbol{R}_t$ is updated at each time step t in one episode, while $\boldsymbol{R}_e$ is updated at each episode (that is, it is constant in one episode).

The desired output of the actor $\boldsymbol{T}_a$ for $\boldsymbol{s}_t$ is given as follows:

$$\boldsymbol{T}_a(\boldsymbol{s}_t) = \boldsymbol{a}(\boldsymbol{s}_t) + \rho(\mathrm{TD}_{\mathrm{error}})(\boldsymbol{a}'(\boldsymbol{s}_t) - \boldsymbol{a}(\boldsymbol{s}_t)), \tag{11}$$

where $\rho(>0)$ is a constant parameter.

In the end of an episode, the neural network is retrained by BPM using desired actor and critic outputs ((5) and (11)) obtained in the episode in addition to the teaching signals obtained in the demonstration in supervised learning.

$N_{\max}$ episodes are repeated in reinforcement learning to obtain an improved neural network that can achieve manipulation tasks in wider task conditions.

B. View-based Reward

Design of appropriate reward r_t is necessary for successful reinforcement learning. In order to carry the object to the goal, the Euclidean distance between the object and the goal is useful to design the reward function. In our method, however, the distance cannot be obtained explicitly because we use view-based approach. Thus we have to find a view-based alternative for the distance.

Here we define an image-based distance function as follows:

$$D_I(\boldsymbol{I}_1, \boldsymbol{I}_2) = \sum_{j=1}^{N_{\mathrm{pixel}}} \frac{|I_{1j} - I_{2j}|}{N_{\mathrm{pixel}}}, \tag{12}$$

where $\boldsymbol{I}_i$ is a camera image and $[I_{i1}, \ldots, I_{iN_{\mathrm{pixel}}}]$ is its [0,1]-normalized pixel values. We use $D_I(\boldsymbol{I}, \boldsymbol{I}_G)$, where $\boldsymbol{I}_G$ is the camera image at the goal in the demonstration, instead of the Euclidean distance between the object and the goal.

Fig. 5 is a plot of $D_I(\boldsymbol{I}, \boldsymbol{I}_G)$ versus the Euclidean distance between the object and the goal in a case. The image-based distance is not fully proportional to the Euclidean distance, but it can be used for its substitute to some extent.

We can obtain not the Euclidean distance between the object and the goal, but the Euclidean distance between the current position and the goal position of the hand as follows:

$$D_H(\boldsymbol{x}, \boldsymbol{x}_G) = \sqrt{(x - x_G)^2 + (y - y_G)^2 + (z - z_G)^2}, \tag{13}$$

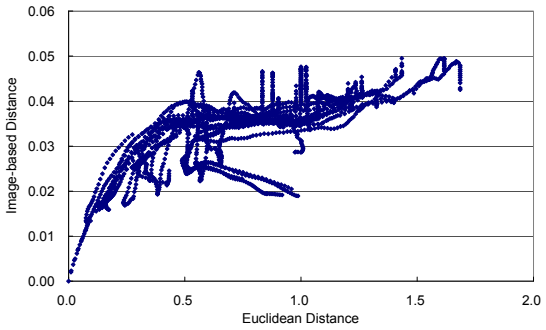


Fig. 5. Euclidean Distance and Image-based Distance

where $\boldsymbol{x}_G = [x_G, y_G, z_G]$ is the position of the hand at the goal in the demonstration.

We define a reward function using these distance functions as follows:

$$r_t = -\alpha_I D_I(\boldsymbol{I}, \boldsymbol{I}_G) - \alpha_H D_H(\boldsymbol{x}, \boldsymbol{x}_G), \tag{14}$$

where α_I and α_H are weight coefficients. The reward can be regarded as a penalty for staying away from the goal.

The view-based reward (14) is also used for the goal condition. When r_t becomes equal to or larger than r_{goal}, a threshold, we terminate the robot motion and finish the episode.

We also finish the episode when the goal condition is not satisfied in $t_{\max}$ steps. In this case, additional reward $r_{\text{fail}}(< 0)$ is given as a penalty.

The view-based reward (14) is also used to control exploration as follows:

$$\sigma_i = \begin{cases} \sigma_{\text{normal}i} & \text{when } r_{\max} < r_{\text{goal}}, \\ \sigma_{\text{goal}i} & \text{when } r_{\max} \geq r_{\text{goal}}, \end{cases} \tag{15}$$

where $\sigma_{\text{goal}i} < \sigma_{\text{normal}i}$ and $r_{\max}$ is the maximum reward obtained in the previous episode.

$$\sigma_{ei} = \begin{cases} C_i(N_{\max} - N) & \text{when } r_{\max} < r_{\text{goal}}, \\ 0 & \text{when } r_{\max} \geq r_{\text{goal}}, \end{cases} \tag{16}$$

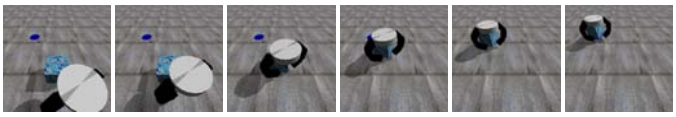
where N is the current number of episodes, and C_i is a coefficient. Those are to restrict exploration when the goal is reached in the previous episode.

VII. Learning Experiments

Using our proposed method, we performed learning experiments in the virtual environment presented in Section IV. The manipulation tasks were pushing and pick-and-place. Parameters for the experiments are found in TABLE I. We used a PC with Core i7-870 CPU (at 2.93 GHz) and GeForce GTX 460 GPU.

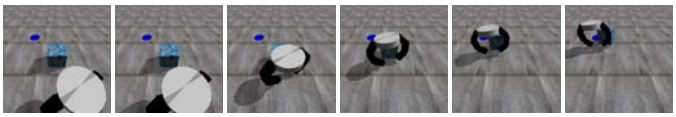
A. Pushing

Programming of pushing the object to a goal position by the robot hand was tested. For simplicity, we made z and α unchanged; that is, the robot hand had only three degrees of freedom (x, y and θ) in this experiment.



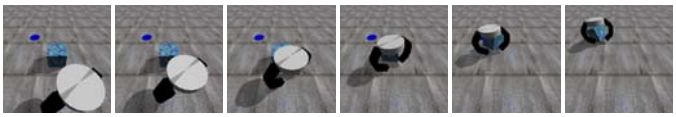
(a) (b) (c) (d) (e) (f)

Fig. 6. Human Demonstration of Pushing



(a) (b) (c) (d) (e) (f)

Fig. 7. Pushing before Reinforcement Learning



(a) (b) (c) (d) (e) (f)

Fig. 8. Pushing after Reinforcement Learning

In view-based supervised learning, a human operator demonstrated pushing with keyboard commands (Fig. 6). From this demonstration, 100 scene images were taken and an initial neural network was generated. The initial neural network can drive the virtual robot hand to push the object to the goal when the initial position of the object is in the neighborhood of that in the demonstration. Fig. 9a shows the range of the initial positions from which pushing to the goal is successful.

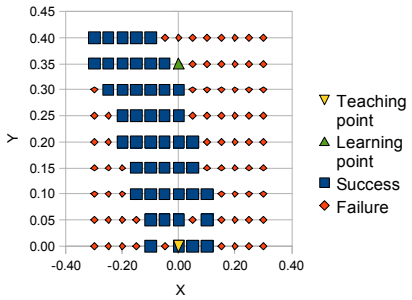
Next, view-based reinforcement learning was performed. In the reinforcement learning, the object was initially located at a shifted position from which the initial neural network was not able to carry it to the goal (Fig. 7). After $N_{\max}$ episodes

TABLE I
Parameters for Experiments

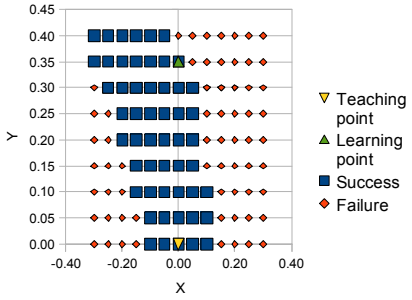
	Pushing	Pick-and-Place
N_{hidden}	100	100
N_{PC}	100	100
N_{pixel}	256×256	256×256
$N_{\max}$	1000	1000
β	0.1	0.3
ρ	0.1	0.03
α_I	1.0	0.5
α_H	0.1	0.1
γ	0.98	0.98
$\sigma_{\text{normal}i}$	0.1	0.1
$\sigma_{\text{goal}i}$	0.01	0.01
C_i	0.05	0.05
r_{goal}	-0.011	-0.018
r_{fail}	-10	-10
$t_{\max}$	120	120

TABLE II
Average Computation Time for Reinforcement Learning [s]

PCA	2 806
BPM	130
Dynamics simulation and others	3 265
Total	6 201



(a) before RL



(b) after RL

Fig. 9. Range of Initial Object Positions for Successful Pushing

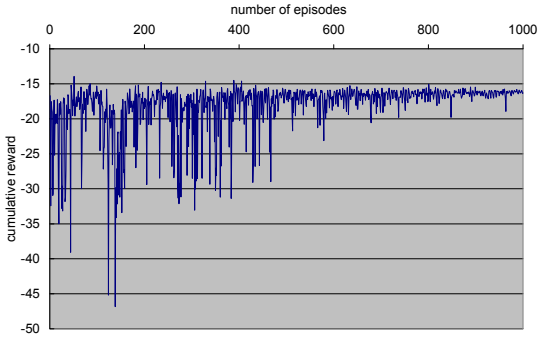


Fig. 10. Average Cumulative Reward

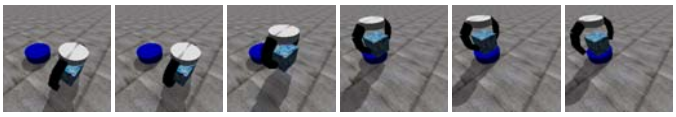
in reinforcement learning, an updated neural network was obtained. It can drive the virtual hand to the goal not only from the initial position of the object in the human demonstration but also from the shifted position (Fig. 8). Fig. 10 shows average cumulative reward in successful cases. TABLE II shows average computation time for reinforcement learning in successful cases

The range of the initial position of the object for successful manipulation after reinforcement learning is shown in Fig. 9b. It is found that wider variations in the initial position are allowed after reinforcement learning.

B. Pick-and-Place

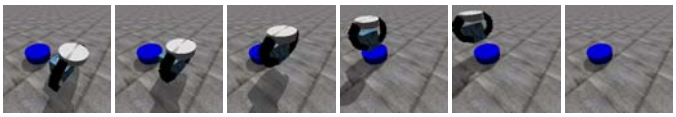
Programming of picking the object up and placing it to a goal position by the robot hand was tested. For simplicity, we made y and θ unchanged; that is, the robot hand had only three degrees of freedom (x, z and α) in this experiment.

In view-based supervised learning, a human operator demonstrated pick-and-place with keyboard commands



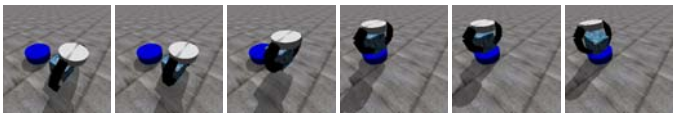
(a) (b) (c) (d) (e) (f)

Fig. 11. Human Demonstration of Pick-and-Place



(a) (b) (c) (d) (e) (f)

Fig. 12. Pick-and-Place before Reinforcement Learning



(a) (b) (c) (d) (e) (f)

Fig. 13. Pick-and-Place after Reinforcement Learning

TABLE III
AVERAGE COMPUTATION TIME FOR REINFORCEMENT LEARNING [S]

PCA	2 526
BPM	481
Dynamics simulation and others	3 887
Total	6 894

(Fig. 11). From this demonstration, 100 scene images were taken and an initial neural network was generated. The initial neural network can drive the virtual robot hand to carry the object to the goal when the initial position of the object is in the neighborhood of that in the demonstration (Fig. 14a).

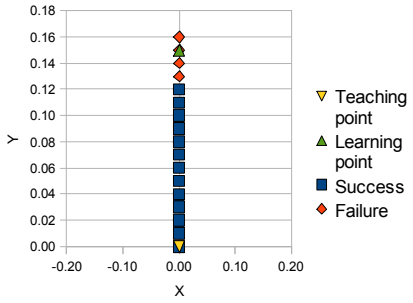
Next, view-based reinforcement learning was performed. In the reinforcement learning, the object was initially located at a shifted position from which the initial neural network was not able to carry it to the goal (Fig. 12). After $N_{\max}$ episodes in reinforcement learning, an updated neural network was obtained. It can drive the virtual hand to the goal not only from the initial position of the object in the human demonstration but also from the shifted position (Fig. 13). Fig. 15 shows average cumulative reward in successful cases. TABLE III shows average computation time for reinforcement learning in successful cases.

The range of the initial position of the object for successful manipulation after reinforcement learning is shown in Fig. 14b. It is found that wider variations in the initial position are allowed after reinforcement learning.

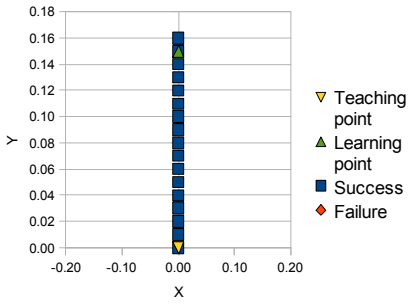
C. Discussion

The experimental results show that our view-based robot programming with reinforcement learning can achieve adaptability autonomously.

A major problem in our reinforcement learning is the local maximum trap. We have to explore a continuous parameter space, which usually has numerous local minima. Due to this problem, the success rate of reinforcement learning is about



(a) before RL



(b) after RL

Fig. 14. Range of Initial Object Positions for Successful Pick-and-Place

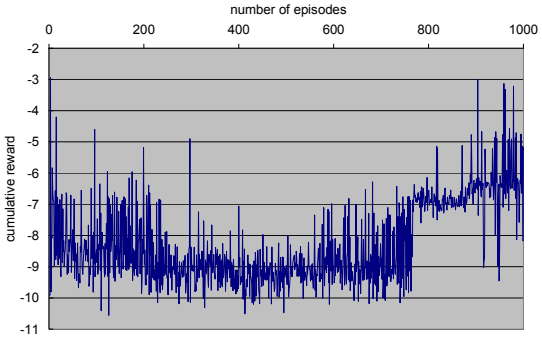


Fig. 15. Average Cumulative Reward

25%. However, we can restart our reinforcement learning, which uses randomness for exploration, until learning success. Because computation time for reinforcement learning in $N_{\max}$ episodes is about 7000 [s], the expected computation time until learning success can be estimated as follows:

$$7000 \times (1 \times 0.25 + 2 \times 0.75 \times 0.25 + 3 \times 0.75^2 \times 0.25 + \cdots) = 28000 \text{ [s]} \approx 8 \text{ [h]}. \quad (17)$$

For more efficient reinforcement learning, improvement of our method including the design of the reward function is necessary.

VIII. Conclusion

In this paper, we proposed view-based robot programming with reinforcement learning. It is an extended version of view-based teaching/playback [1] so that it can adapt to wider range of changes in task conditions without more human demonstrations.

In the learning experiments, our method acquired a mapping from camera images to robot motions after view-based supervised and reinforcement learning. The mapping can generate appropriate robot motions for pushing and pick-and-place not only from the initial position of the object in the human demonstration but from different initial positions.

Future work should address the improvement of the method for effective reinforcement learning. Application to manipulation tasks by actual robots should also be addressed. We have already applied view-based teaching/playback *without* reinforcement learning to pushing by an actual manipulator successfully [10], but successful introduction of reinforcement learning would require more effective learning.

References

[1] Y. Maeda, T. Nakamura, and T. Watanabe, "View-based teaching/playback for grasp and grasp less manipulation," in *Proc. of Int. Conf. on Advanced Mechatronics*, 2010, pp. 75–80.

[2] Y. Matsumoto, M. Inaba, and H. Inoue, "View-based navigation using an omniview sequence in a corridor environment," *Machine Vision and Applications*, vol. 14, no. 2, pp. 121–128, 2003.

[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[4] K. Shibata and M. Iida, "Acquisition of box pushing by direct-vision-based reinforcement learning," in *Proc. of SICE Annual Conf.*, 2003, pp. 1378–1383.

[5] M. Kato, Y. Kobayashi, and S. Hosoe, "Optimizing resolution for feature extraction in robotic motion learning," in *Proc. of IEEE Int. Conf. on Systems, Man and Cybernetics*, 2005, pp. 1086–1091.

[6] Y. Kobayashi, M. Kato, and S. Hosoe, "Optimizing resolution for feature extraction in robotic motion learning," *J. of Robotics Society of Japan*, vol. 25, no. 5, pp. 770–778, 2007, (in Japanese).

[7] "Open Dynamics Engine," http://www.ode.org/.

[8] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[9] M. R. G. Meireles, P. E. M. Almeida, and M. G. Simões, "A comprehensive review for industrial applicability of artificial neural networks," *IEEE Trans. on Industrial Electronics*, vol. 50, no. 3, pp. 585–601, 2003.

[10] Y. Maeda and Y. Moriyama, "View-based teaching/playback for industrial manipulators," in *Proc. of IEEE Int. Conf. on Robotics and Automation*, 2011, pp. 4306–4311.