Integration of Genetic Programming and Reinforcement Learning for Real Robots

Shotaro Kamio, Hideyuki Mitsuhashi, and Hitoshi Iba

Graduate School of Frontier Science, The University of Tokyo,
7-3-1 Hongo, Bunkyo-ku, Tokyo, 113-8656, Japan.
{kamio,mituhasi,iba}@miv.t.u-tokyo.ac.jp

Abstract. We propose an integrated technique of genetic programming (GP) and reinforcement learning (RL) that allows a real robot to execute real-time learning. Our technique does not need a precise simulator because learning is done with a real robot. Moreover, our technique makes it possible to learn optimal actions in real robots. We show the result of an experiment with a real robot AIBO and represents the result which proves proposed technique performs better than traditional Q-learning method.

1 Introduction

When executing tasks by autonomous robots, we can make the robot learn what to do so as to complete the task from interactions with its environment but not manually pre-program for all situations. We know that such learning techniques as genetic programming (GP)[1] and reinforcement learning (RL)[2] work as means for automatically generating robot programs.

When applying GP, we should repeatedly evaluate many individuals over several generations. Therefore, it is difficult to apply GP to problems that requires too much time for evaluations of individuals. That is why we find very few previous studies on learning with a real robot.

To obtain optimal actions using RL, it is necessary to repeat learning trials time after time. The huge amount of learning time required presents a great problem when using a real robot. Accordingly, most studies deal with the problems of receiving an immediate reward from an action as shown in [3], or loading the results learned with a simulator into a real robot as shown in [4,5].

Although it is generally accepted to learn with a simulator and apply the result to a real robot, there are many tasks that are difficult to make a precise simulator. Applying these methods with an imprecise simulator could result in creating programs which may function optimally on the simulator but cannot provide optimal actions with a real robot. Furthermore, the operating characteristics of a real robot show certain variations due to minor errors in the manufacturing process or to changes with time. We cannot cope with such differences of robots only using a simulator.

Learning process with a real robot is surely necessary, therefore, for it to acquire optimal actions. Moreover, learning with a real robot sometimes makes

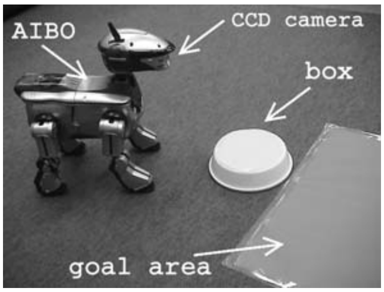


Fig. 1. The robot AIBO, the box and the goal area.

possible to learn even hardware and environmental characteristics, thus allowing the robot to acquire unexpected actions.

To solve the above difficulties, we propose a technique that allows a real robot to execute real-time learning in which GP and RL are integrated. Our proposed technique does not need a precise simulator because learning is done with a real robot. As a result of this idea, we can greatly reduce the cost to make the simulator much precise and acquire the program which acts optimally in the real robot.

The main contributions of this paper are summarized as follows:

1. We propose an integrated method of GP and RL.
2. We give empirical results to show how our approach works well for real-robot learning.
3. We conduct comparative experiments with traditional Q-learning to show the superiority of our method.

The next section gives the definition of a task in this study. After that, Section 3 explains our proposed technique and Section 4 presents experimental results with a real robot. Section 5 provides the result of comparison and future researches. Finally, a conclusion is given.

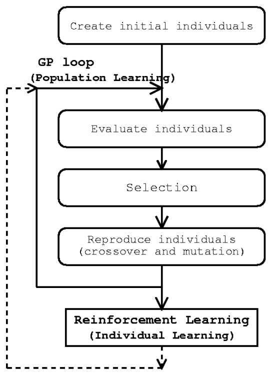
2 Task Definition

We used an "AIBO ERS-220" (Fig. 1) robot sold by SONY as the real robot in this experiment. AIBO's development environment is freely available for non-commercial use and we can program with C++ language on it [6]. An AIBO has a CCD camera on its head, and moreover, an AIBO is equipped with image processor. It is able to easily recognize objects of specified colors on a CCD image at high speed.

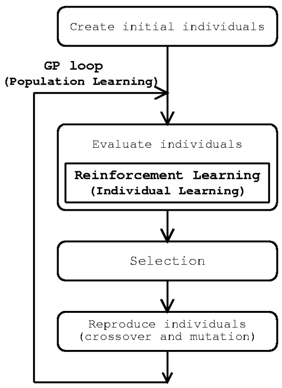
The task in this experiment was to carry a box to a goal area. One of the difficulties of this task is that the robot has four legs. As a result, when the robot moves ahead, we see cases where the box sometimes is moved ahead or deviates from side to side, depending on the physical relationship between the box and AIBO legs. It is extremely difficult, in fact, to create a precise simulator that accurately expresses this box movements.

3 Proposed Technique

In this paper, we propose a technique that integrates GP and RL. As can be seen in Fig. 2(a), RL as individual learning is outside of the GP loop in the proposed technique. This technique enables us (1) to speed up learning in real robot and (2) to cope with the differences between a simulator and a real robot.



(a) Proposed technique of the integration of GP and RL.



(b) Traditional method combined GP and RL [7,8].

Fig. 2. The flow of the algorithm.

The proposed technique consists of two stages (GP part and RL part).

1. Carry out GP on a simplified simulator, and formulate programs that have the standards for robot actions required for executing a task.
2. Conduct individual learning (= RL) after loading the programs obtained in Step 1 above.

In the first step above, the programs that have the standards for the actions required of a real robot to execute a task are created through the GP process. The learning process of RL can be speeded up in the second step because the state space is divided into partial spaces under the judgment standards obtained in the first step. Moreover, preliminary learning with a simulator allows us to anticipate that a robot performs target-oriented actions from the beginning of the second stage. We used Q-learning as RL method in this study.

Although the process expressed by the external dotted line in Fig.2(a) was not realized in this study, it is a feedback loop. We consider that the parameters in a real environment that have been acquired via individual learning should ideally be fed back through this loop route.

The comparison with the traditional method (Fig. 2(b)) is discussed later in Sect. 5.2.

3.1 RL Part Conducted on the Real Robot

Action set. We prepared six selectable robot actions (move forward, retreat, turn left, turn right, retreat + turn left, and retreat + turn right). These actions are far from ideal ones: e.g. "move forward" action is not only to move the robot straightly forward but also has some deviations from side to side and "turn left" action is not only to turn left but also move the robot a little bit forward. The robot has to learn these characteristics of actions.

Every action takes approximately four seconds and eight seconds including the swinging of the head described below. It is, therefore, advisable that the learning time is as short as possible.

State Space. The state space was structured based on positions from where the box and the goal area can be seen in the CCD image, as described in [4]. The viewing angle of AIBO CCD is so narrow that the box or the goal area cannot be seen well with only one-directional images, in most cases. To avoid this difficulty, we added a mechanism to compensate for the surrounding images by swinging AIBO's head so that state recognition can be conducted by the head swinging after each action. This head swinging operation was always uniformly given throughout the experiment as it was not an element to be learned in this study.

Figure 3 is the projection of the box state on the ground surface. The "near center" position is where the box fits into the two front legs. The box can be moved if the robot pushes it forward in this state. The box remains same position "near center" after the robot turns left or right in this state because the robot holds the box between two front legs. The state with the box not being in view was defined as "lost"; the state with the box not being in view and one preceding step at the left was defined as "lost into left" and, similarly, "lost into right" was defined.

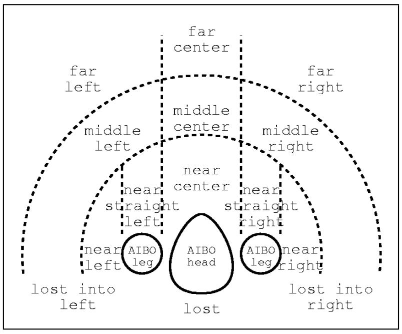


Fig. 3. States in real robot for the box. The front of the robot is upside of this figure.

We should pay special attention to the position of legs. Depending on the physical relationship between the box and AIBO legs, the movement of the box varies from moving forward to deviating from side to side. If an appropriate state space is not defined, the Markov property of the environment, which is a premise of RL, cannot be met, thereby optimal actions cannot be found. Therefore, we defined "near straight left" and "near straight right" states at the frontal positions of the front legs.

We thus defined 14 states for the box. We similarly defined states of the goal area except that "near straight left" and "near straight right" states do not exist in them. There are 14 states for the box and 12 for the goal area; hence, this environment has states of their product, i.e., 168 states totally.

3.2 GP Part Conducted on the Simulated Robot

Simulator. The simulator in our experiment uses a robot expressed in circle on a two-dimensional plane, a box, and a goal area fixed on a plane. The task is completed when the robot pushes the box forward and overlaps the goal area on this plane.

We defined three actions (move forward, turn left, turn right) as action set and defined the state space in the simulator which is the simplified state space used for a real robot as Fig. 4. While actions of the real robot are not ideal ones, these actions in the simulator are ideal ones.

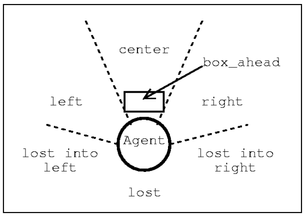


Fig. 4. States for box and goal area in the simulator. The area `box_ahead` is not the state but the place where `if_box_ahead` executes first argument.

Such actions and a state division is similar to that of a real robot, but is not exactly the same. In addition, physical parameters such as box weight and friction were not measured nor was the shape of the robot taken into account. Therefore, this simulator is very simple and it is possible to build it in low cost.

The two transfer characteristics of the box expressed by the simulator are the following.

1. The box moves forward if the box comes in contact with the front of the robot when the robot goes ahead[1].
2. After rotation, the box is near the center of the robot if the box is near the center of the robot when the robot turns[2].

[1] This corresponds to the situation that real robot pushes the box forward.

[2] This corresponds to the situation in which the box is placed between the front legs of a real robot when it is turning.

Settings of GP. The terminals and functions used in GP were as follows:

Terminal set: `move_forward`, `turn_left`, `turn_right`
Function set: `if_box_ahead`, `box_where`, `goal_where`, `prog2`

The terminal nodes above respectively correspond to the "move forward", "turn left", and "turn right" actions in the simulator. The functional nodes `box_where` and `goal_where` are the functions of six arguments, and they execute one of the six arguments, depending on the states (Fig. 4) of the box and the goal area as seen by the robot's eyes. The function `if_box_ahead` which has two arguments executes the first argument if the box is positioned at "box ahead" position in Fig. 4. We arranged conditions so that only the `box_where` or the `goal_where` node becomes the head node of a gene of GP. The gene of GP is set to start executing from the head node and the execution is repeated again from the head node if the execution runs over the last leaf node until reaches maximum steps.

A trial starts with the state in which the robot and the box are randomly placed at the initial positions, and ends when the box is placed in the goal area or after a predetermined number of actions are performed by the robot. The following fitness values are allocated to the actions performed in a trial.

– If the task is completed:

$$\begin{aligned} f_{goal} &= 100 \\ f_{\mathrm{remaining\_moves}} &= 10 \times \left(0.5 - \frac{(\text{ Number of moves })}{(\text{ Maximum limit of number of moves })}\right) \\ f_{\mathrm{remaining\_turns}} &= 10 \times \left(0.5 - \frac{(\text{ Number of turns })}{(\text{ Maximum limit of number of turns })}\right) \end{aligned}$$

– If the box is moved at least once: $f_{\mathrm{move}} = 10$
– If the robot faces the box at least once: $f_{\mathrm{see\_box}} = 1$
– If the robot faces the goal at least once: $f_{\mathrm{see\_goal}} = 1$
– $f_{\mathrm{lost}} = -\dfrac{(\text{ Number of times having lost sight of the box })}{(\text{ Number of steps })}$

The sum of the above figures indicates a fitness value for the i-th trial in an evaluation, or $fitness_i$.

To make robot acquire robust actions that do not depend on the initial position, the average values of 100 trials in which the initial position is randomly changed was taken when calculating the fitness of individuals. The fitness of individuals is calculated by the following equation.

$$fitness = \frac{1}{100}\sum_{i=0}^{99} fitness_i + 2.0 \cdot \frac{(\text{Maximum gene length}) - (\text{Gene length})}{(\text{Maximum gene length})} \quad (1)$$

The second term of the right side of this equation has the meaning that a penalty is given to a longer gene length.

Using the fitness function determined above, learning was executed for 1,000 individuals of 50 generations with maximum gene length = 150. Learning costs about 10 minutes on the Linux system equipped with an Athlon XP 1800+.

We finally applied the individuals that had proven to have the best performance to learning with a real robot.

Table 1. Action nodes and their selectable real actions.

action node	real actions which Q-table can select.
`move_forward`	"move forward"*, "retreat + turn left", "retreat + turn right"
`turn_left`	"turn left"*, "retreat + turn left", "retreat"
`turn_right`	"turn right"*, "retreat + turn right", "retreat"

* The action which Q-table prefers to select with a biased initial value.

3.3 Integration of GP and RL

Q-learning is executed to adapt actions acquired via GP to the operating characteristics of a real robot. This is aimed at revising the `move_forward`, `turn_left` and `turn_right` actions with the simulator to their optimal actions in a real world.

We allocated a Q-table, on which Q-values were listed, to each of the `move_forward`, `turn_left` and `turn_right` action nodes. The states on the Q-tables are regarded as those for a real robot. Therefore, actual actions selected with Q-tables can vary depending on the state, even if the same action nodes are executed by a real robot. Figure 5 illustrates the above situation. The states "near straight left" and "near straight right", which exist only in a real robot, are translated into a "center" state in function nodes of GP.

Each Q-table is arranged to set the limits of selectable actions. This refers to the idea that, for example, "turn right" actions are not necessary to learn in the `turn_left` node. In this study, we defined three selectable robot actions for each action node as Table 1. With this technique, each Q-table was initialized with a biased initial value[3]. The initial value of 0.0001 was entered into the respective Q-tables so that preferred actions were selected for each Q-table, while 0.0 was entered for other actions. The actions which are preferred to select on each action node are described in Table 1.

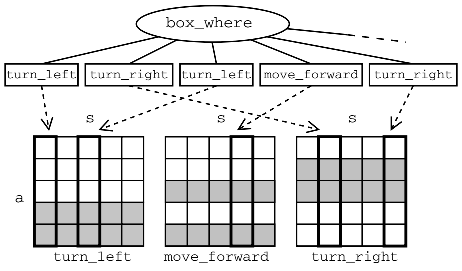


Fig. 5. Action nodes pick up a real action according to the Q-value of a real robot's state.

The total size of the three Q-tables is 1.5 times that of ordinary Q-learning. Theoretically, convergence with the optimal solution is considered to require

[3] According to the theory, we can initialize Q-values with arbitrary values, and Q-values converge with the optimum solution regardless of the initial value [2].

more time than ordinary Q-learning. However, the performance of this technique while programs are executed is relatively good. This is because all the states on the Q-table are not necessarily used as the robot performs actions according to the programs obtained via GP and the task-based actions are available after the Q-learning starts.

The "state-action deviation" problem should be taken into account when executing Q-learning with the state constructed from a visual image [4]. This is the problem that optimal actions cannot be achieved due to the dispersion of state transitions because the state composed only of the images remains the same without clearly distinguishing differences in image values. To avoid this problem, we redefined "changes" in states. The redefinition is that the current state is unchanged if the terminal node executed in the program remains the same and so does the executing state of a real robot[4]. Until the current state changes, the Q-value is not updated and the same action is repeated.

As for parameters for Q-learning, the reward was set at 1.0 when the goal is achieved and 0.0 for other states. We set the parameters as the learning rate $\alpha = 0.3$ and the discount factor $\gamma = 0.9$.

4 Experimental Results with AIBO

Just after starting learning: The robot succeeded in completing the task when Q-learning with a real robot started using this technique. This was because the robot could perform actions by taking advantage of the results learned via GP.

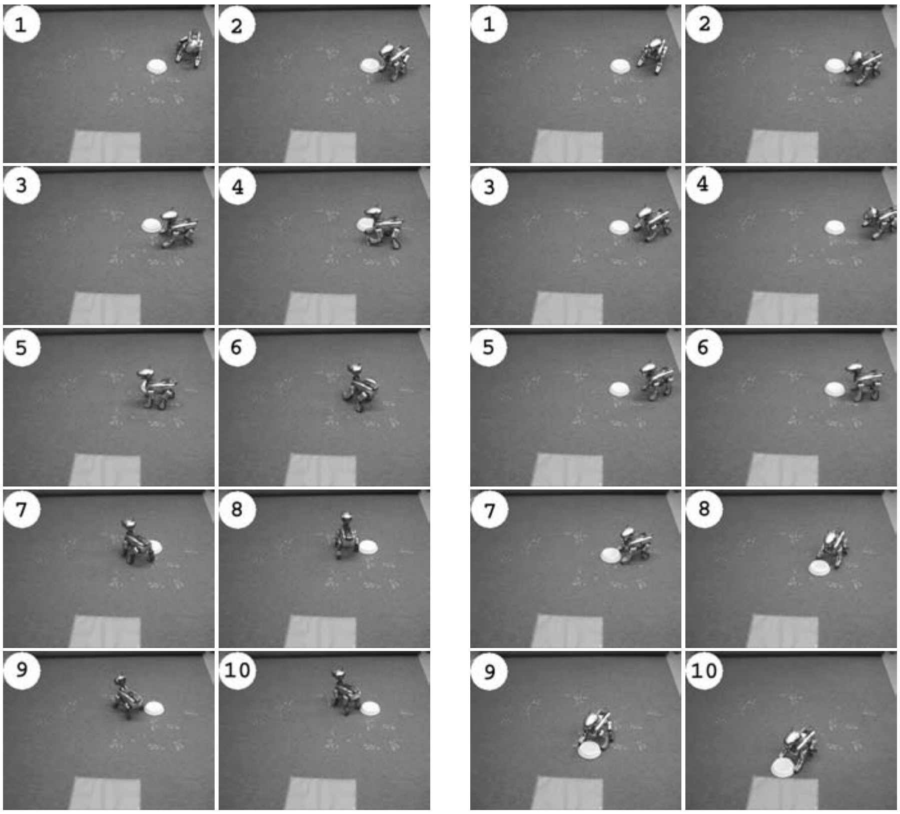
At the situation in which the box was placed near the center of the robot along with robot movements, the robot always achieved the task with regard to all the states tried. Whereas, if the box was not placed near the center of the robot after its displacement (e.g. if the box was slightly outside the legs), the robot sometimes failed to move the box properly. The robot repeatedly turned right to face the box, but continued vain movements going around the box because it did not have a small turning circle, unlike the actions in the simulator. Figure 6(a) shows typical series of actions. In some situation, the robot turned right but could not face the box and lost it in view (at the last of Fig. 6(a)).

This typical example proves that optimal actions with the simulator are not always optimal in a real environment. This is because of differences between the simulator and the real robot.

After ten hours (after about 4000 steps): We observed optimal actions as Fig. 6(b). The robot selected "retreat" or "retreat + turn" action in the situations in which it could not complete the task at the beginning of Q-learning. As a result, the robot could face the box and pushed the box forward to the goal, and finally completed the task.

Learning effects were found in other point, too. As the robot approached the box smoothly, the number of occurrence of "lost" was reduced. This means the robot acts more efficiently than the beginning of learning.

[4] We modified Asada et al.'s definition [4] in order to deal with several Q-tables.



(a) Failed actions losing the box at the beginning of learning.

(b) Successful actions after 10-hour learning.

Fig. 6. Typical series of actions.

5 Discussion

5.1 Comparison with Q-Learning in Both Simulator and Real Robot

We compared our proposed technique with the method of Q-learning which learns in a simulator and re-learns in a real world (we call this method as RL+RL in this section). For Q-learning in the simulator, we introduced the qualitative distance ("far", "middle", and "near") so that the state space could be similar to the one for the real robot[5].

For this comparison, we selected ten situations which are difficult to complete at the beginning of Q-learning because of the gap between the simulation and

[5] This simulator has 12 states for each of the box and the goal area; hence, this environment has 144 states.

Table 2. Comparison of proposed technique (GP+RL) with Q-learning (RL+RL).

#. situation	GP+RL			RL+RL		
	avg. steps	lost box	lost goal	avg. steps	lost box	lost goal
1	**19.6**	0	1	20.0	0	1
2	**14.7**	0	0	53.0	2	2
3	**24.0**	0	1	26.7	0	1
4	**10.3**	0	0	11.0	0	0
5	**21.6**	0	0	88.0	3	3
6	13.5	0	0	**10.5**	0	0
7	26.7	0	1	**26.0**	0	1
8	23.0	0	1	**13.0**	0	0
9	21.5	0	0	**10.5**	0	0
10	**13.5**	0	0	29.0	0	1

the real robot. We measured action efficiency after ten-hour Q-learning for these ten situations. These tests are executed in a greedy policy in order that the robot always selects the best action in each state.

Table 2 shows the result of both methods, i.e., proposed technique (GP+RL) and Q-learning method (RL+RL). This table represents the average number of steps to complete the task and the number of occurrences when the robot has lost the box or the goal area in completing the task.

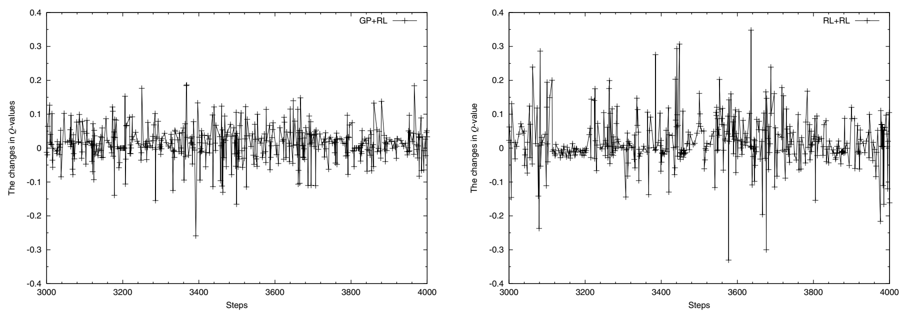
While RL+RL performed better than the proposed technique in four situations on the average of the steps, the proposed technique performed much better than RL+RL in other six situations (bold font in Table 2). Moreover, the robot evolved by the proposed technique less often lost the box and the goal area than that by RL+RL. This result proves that our proposed technique learned more efficient actions than RL+RL method.

Figure 7 shows the changes in Q-values when they are updated in Q-learning with the real robot. The absolute value of the Q-value change represents how far the Q-value is from the optimal one. According to Fig. 7, large changes occurred to RL+RL method more frequently than to our technique. This may be because RL+RL has to re-learn optimal Q-values starting from the ones which have already been learned with the simulator. Therefore, we can conclude that RL+RL requires more time to converge to optimal Q-values.

5.2 Related Works

There are many studies combined evolutionary algorithms and RL [9,10]. Although the approaches differ from our proposed technique, we see several studies in which GP and RL are combined [7,8]. With these traditional techniques, Q-learning is adopted as a RL, and individuals of GP represents the structure of the state space to be searched. It is reported that searching efficiency is improved in QGP, compared to traditional Q-learning [7].

However, the techniques used in these studies are also a kind of population learning using numerous individuals. RL must be executed for numerous individuals in the population because RL is inside the GP loop, as shown in Fig. 2(b). A huge amount of time would become necessary for learning if all the processes



(a) Proposed technique (GP+RL). (b) Q-learning (RL+RL).

Fig. 7. Comparison of changes in Q-values after about 8-hour to 10-hour Q-learning with a real robot.

are directly applied to a real robot. As a result, no studies using any of these techniques with a real robot have been reported.

Several studies on RL pursue the use of hierarchical state space to enable us to deal with complicated tasks [11,12]. The hierarchical state spaces in such studies is structured manually in advance. It is generally considered difficult to automatically build the hierarchical structure only through RL. We can consider the programs automatically generated in GP of proposed technique represents the hierarchical structure of state space which is manually structured in [12].

Noises in simulators are often effective to overcome the differences between a simulator and real environment [13]. However, the robot learned with our technique showed sufficient performance in noisy real environment, while it learned in ideal simulator. One of the reasons is that the coarse state division absorbs the image processing noise. We plan to perform a comparison the robustness produced by our technique with that by noisy simulators.

5.3 Future Researches

We used only several discrete actions in this study. Although this is simple, continuous actions are more realistic in applications. In that situation, for example, "turn left in 30.0 degrees" in the beginning of RL can be changed to "turn left in 31.5 degrees" after learning, depending on the operating characteristics of the robot. We plan to conduct an experiment with such continuous actions.

We intend to apply the technique to more complicated tasks such as the multi-agent problem and other real-robot learning. Based on our method, it can be possible to use almost the same simulator and settings of RL as described in this paper. Experiments will be conducted with various robots, e.g., a humanoid robot "HOAP-1" (manufactured by Fujitsu Automation Limited) or "Khepera". The preliminary results were reported in [14]. We are in pursuit of the applicability of the proposed approach to this wide research area.

6 Conclusion

In this paper, we proposed a technique for executing real-time learning with a real robot based on an integration of GP and RL techniques, and verified its effectiveness experimentally.

At the initial stage of Q-learning, we sometimes observed unsuccessful displacements of the box due to a lack of data concerning real robot characteristics, which had not been reproduced by a simulator. The technique, however, was adapted to the operating characteristics of the real robot through the ten hour learning period. This proves that the step of individual learning in this technique performed effectively in our experiment.

This technique, however, still has several points to be improved. One is feeding back data from learning in a real environment to GP and the simulator, which corresponds to the loop represented by the dotted line in Fig.2(a). This may enable us to improve simulator precision automatically in learning. Its realization is one of the future issues.

References

1. John R. Koza: Genetic Programming, On the Programming of Computers by means of Natural Selection. MIT Press (1992)
2. Richard S. Sutton and Andrew G. Barto: Reinforcement Learning: An introduction. MIT Press in Cambridge, MA (1998)
3. Hajime Kimura, Toru Yamashita and Shigenobu Kobayashi: Reinforcement Learning of Walking Behavior for a Four-Legged Robot. In: 40th IEEE Conference on Decision and Control. (2001)
4. Minoru Asada, Shoichi Noda, Sukoya Tawaratsumida and Koh Hosoda: Purposive Behavior Acquisition for a Real Robot by Vision-Based Reinforcement Learning. Machine Learning **23** (1996) 279–303
5. Yasutake Takahashi, Minoru Asada, Shoichi Noda and Koh Hosoda: Sensor Space Segmentation for Mobile Robot Learning. In: Proceedings of ICMAS'96 Workshop on Learning, Interaction and Organizations in Multiagent Environment. (1996)
6. OPEN-R Programming Special Interest Group: Introduction to OPEN-R programming (in Japanese). Impress corporation (2002)
7. Hitoshi Iba: Multi-Agent Reinforcement Learning with Genetic Programming. In: Proc. of the Third Annual Genetic Programming Conference. (1998)
8. Keith L. Downing: Adaptive genetic programs via reinforcement learning. In: Proc. of the Third Annual Genetic Programming Conference. (1998)
9. Moriarty, D.E., Schultz, A.C., Grefenstette, J.J.: Evolutionary algorithms for reinforcement learning. Journal of Artificial Intelligence Research **11** (1999) 199–229
10. Dorigo, M., Colombetti, M.: Robot Shaping: An Experiment in Behavior Engineering. MIT Press (1998)
11. L.P. Kaelbling: Hierarchical Learning in Stochastic Domains: preliminary Results. In: Proc. 10th Int. Conf. on Machine Learning. (1993) 167–173
12. T.G. Dietterich: Hierarchical reinforcement learning with the MAXQ value function decomposition. Journal of Artificial Intelligence Research **13:227 303** (2000)

13. Schultz, A.C., Ramsey, C.L., Grefenstette, J.J.: Simulation-assisted learning by competition: Effects of noise differences between training model and target environment. In: Proc. of Seventh International Conference on Machine Learning, San Mateo, Morgan Kaufmann (1990) 211–215
14. Kohsuke Yanai and Hitoshi Iba: Multi-agent Robot Learning by Means of Genetic Programming: Solving an Escape Problem. In Liu, Y., et al., eds.: Evolvable Systems: From Biology to Hardware. Proceedings of the 4th International Conference on Evolvable Systems, ICES'2001, Tokyo, October 3-5, 2001, Springer-Verlag, Berlin, Heidelberg (2001) 192–203