Adaptation Technique for Integrating Genetic Programming and Reinforcement Learning for Real Robots

Shotaro Kamio and Hitoshi Iba, *Member, IEEE*

***Abstract*—We propose an integrated technique of genetic programming (GP) and reinforcement learning (RL) to enable a real robot to adapt its actions to a real environment. Our technique does not require a precise simulator because learning is achieved through the real robot. In addition, our technique makes it possible for real robots to learn effective actions. Based on this proposed technique, we acquire common programs, using GP, which are applicable to various types of robots. Through this acquired program, we execute RL in a real robot. With our method, the robot can adapt to its own operational characteristics and learn effective actions. In this paper, we show experimental results from two different robots: a four-legged robot "AIBO" and a humanoid robot "HOAP-1." We present results showing that both effectively solved the box-moving task; the end result demonstrates that our proposed technique performs better than the traditional Q-learning method.**

***Index Terms*—Adaptation evolutionary computation, box moving, genetic programming (GP), real robot, reinforcement learning (RL).**

I. Introduction

MACHINE LEARNING techniques can be applied to a robot in order for it to achieve a task if the appropriate actions are not predetermined. In such a situation, the robot can learn the appropriate actions by using trial-and-error in a real environment. Established techniques for this purpose are genetic programming (GP) [1], [2], genetic algorithm (GA) [3], other evolutionary learning methods [4], and reinforcement learning (RL) [5].

GP can generate programs to control a robot directly, and many studies have been done showing this (e.g., [6] and [7]). GA in combination with neural networks (NNs) can also be used to control robots (e.g., [3]). Regardless of the method used, the evaluation of real robots requires a significant amount of time due partly to their complex mechanical actions. Moreover, evaluations have to be repeated for many individuals over several generations in both GP and GA. For example, Andersson *et al.* spent 15 hours evaluating 111 generations to acquire a galloping behavior using GP [6], and Floreano *et al.* spent ten days on 240 generations to evolve the motion of moving toward a light source using GA with NN [8]. Therefore, in most studies, the learning was conducted in simulation, and the acquired results were applied to real robots.

RL identifies optimal actions through interactions with the environment. To achieve optimal actions using RL, it is necessary to repeat learning trials many times. The enormous amount of learning time required for the task with a real robot is a critical problem. Accordingly, most studies deal with the problems of receiving an immediate reward from an action [9], or loading the results learned from a simulator into a real robot [10].

Learning by simulation requires the simulator to represent agents, the environment and their interactions with precision. Many tasks, however, are difficult for the simulator to do precisely. Learning with an imprecise simulator will lead to an ineffective performance in a real environment. For example, Jacobi *et al.* reported that a simulator with zero noise or too much noise never leads to the same exact behaviors in the real world [11]. They concluded that the simulation must include appropriate levels of noise comparable to the real world. This is generally not an easy task. Furthermore, certain variations, due to minor errors in the manufacturing process or to changes over time from normal wear, occur even if the real robots are modeled exactly the same way. The above approach, i.e., to learn with a simulator and apply the result to a real robot, has some limitations. Therefore, learning in a real robot is unavoidable in order to acquire effective actions.

Now that various kinds of robots have been developed, it is important to carry out a task employing these different robots. In this case, it will take much longer if each type of robot has to learn the task independently. To counteract this, we use general knowledge of the commonality of various robots and then fine-tune specific parameters for a particular robot, thereby establishing a more effective learning scheme.

To solve the difficulties with time, precision, and robot-specificity, we propose a technique that allows a real robot to adapt its actions to a real environment in which GP and RL are integrated. Since GP can generate programs, our technique has the possibility of producing more complex behaviors than the reactive behaviors of RL. The proposed technique does not need a precise simulator because learning is done in a real robot. In other words, the precision requirement is met if the task is completed. As a result of this concept, we can greatly reduce the need for and consequently the cost of making the simulator highly precise.

Using the proposed technique, we can apply the program, which has been acquired with the same simulator via GP, to

Manuscript received September 30, 2003; revised June 9, 2004 and December 3, 2004.

S. Kamio is with the IBA Laboratory, Graduate School of Frontier Sciences, University of Tokyo, Chiba 277-8561, Japan (e-mail: kamio@iba.k.u-tokyo.ac.jp).

H. Iba is with the Department of Engineering, University of Tokyo, Tokyo 113-8656, Japan (e-mail: iba@iba.k.u-tokyo.ac.jp).

Digital Object Identifier 10.1109/TEVC.2005.850290

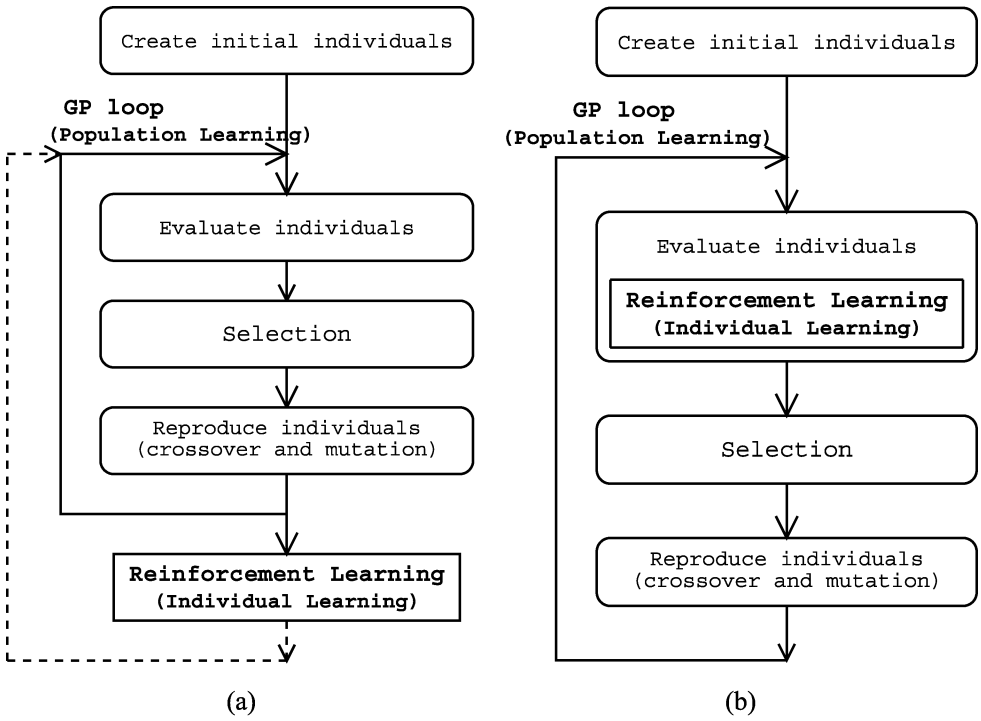


Fig. 1. Flow charts of our proposed algorithm and of a traditional algorithm. (a) Proposed technique of integrating GP and RL. (b) Traditional method of combining GP and RL [14], [15].

different types of robots. The program will adapt to the operational characteristics of each real robot and learn effective actions. The adaptation is faster than learning from scratch because it uses the knowledge acquired in the simulator.

We conducted two experiments on two different robots. First, we used an "AIBO" (a four-legged entertainment robot by SONY) [12] and showed the effectiveness of our proposed technique. Second, we performed an experiment with a humanoid robot "HOAP-1" (manufactured by Fujitsu Automation Limited) [13]. We provide the experimental results to confirm the adaptability of our methodology.

This paper is organized in the following manner. Section II explains our proposed technique. The task employed in this study and the GP results on the task are described in Section III. Next, Sections IV and V present the experimental results of our technique using real robots, "AIBO" and "HOAP-1," respectively. Section VI discusses related works and future research. The conclusion is given in Section VII.

II. Proposed Technique

A. Outline

In this paper, we propose a technique that integrates GP and RL. As seen in Fig. 1(a), RL, individual learning, is outside the GP loop in the proposed technique. This enables us to: 1) speed up learning in a real robot and 2) adapt to a real robot using the programs acquired from simulation.

The proposed technique consists of the following two stages, the GP part and the RL part.

1) Carry out GP on a simplified simulator and formulate programs that control a robot for the purpose of achieving a task.
2) Conduct RL after loading the best program obtained in 1).

In the first stage, the programs for the standard actions required for a real robot to execute a task are created through the GP process. The learning process of RL can be sped up in the second stage because the state space is divided into partial spaces, according to judgment standards established in the first stage. Moreover, preliminary learning from the simulator allows us to anticipate that a robot will perform target-oriented actions from the beginning of the second stage.

Comparison with the traditional method [Fig. 1(b)] is discussed later in Section VI-A.

We show details of our technique, the GP part, and the RL part in the following sections.

B. GP Part Conducted by the Simulated Robot

In the first stage, GP is executed in the simulation. GP is the method of generating programs using natural selection [1]. It uses these programs for its population. Each program is a candidate for achieving one task. The program is expressed by S-expression. There are two elements, the function set and the terminal set, which are used to generate the program. The process of GP is as follows.

1) Generate an initial population of programs.
2) Evaluate each program with a fitness measure.
3) Select programs based on the fitness value.
4) Generate new individual programs with genetic operators (crossover, mutation, etc.).
5) Repeat from 2) until obtaining a program with the satisfactory fitness value.

Upon completion of the algorithm, there is the possibility of acquiring programs that were previously unimaginable. A good

introductory resource is [16], which also describes the variations of GP and its implementation.

C. RL Part Conducted by the Real Robot

In the second stage, RL is executed to adapt actions acquired via GP to the operating characteristics of a particular real robot. In this study, we used Q-learning [5] as the type of RL.

1) Q-Learning: Q-learning is a learning technique for acquiring optimal actions based on the evaluation values $Q(s,a)$ (Q-value) for state-action sets. Assuming that an agent performs the action a_i in the state s_t at a time t, and receives the reward r_{t+1} in the new state s_{t+1} as a result of the action a_i, the updating formula of the Q-value is expressed by

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)\right] \quad (1)$$

where α is a parameter called rate of learning ($0 < \alpha \leq 1$), and γ is a parameter called rate of discount ($0 \leq \gamma \leq 1$). The learning process continues by repeatedly applying this formula to each agent action.

2) Adaptation Ability of Proposed Technique to Real Robots: An action set and a state space are needed for Q-learning. Actions are specific for each real robot, depending on its form and mobility. In the concept of our proposed technique, the real action set consists of simulated actions and their variations. This variation is a key to adaptability. The detailed implementations are shown in later experiments.

There are instances in our method when an acquired GP program cannot adapt to a real environment. When there is a significant difference in optimal actions between a simulation and a real environment, adaptation can fail. An example would be the real action "turn right" being needed in a real environment, while the simulated robot executes the optimal action "turn left" in its simulated environment. The success depends on the strength of the adaptability, i.e., which actions we use as the variation of "turn left" for the real robot.

Q-tables, on which Q-values are listed, are allocated to each of the action nodes in GP. The states used for the Q-tables are the same as those of a real robot. Therefore, actual actions selected with Q-tables can vary depending on the state of a real robot, even if the same action nodes of GP are executed. We slightly modified the updating formula of Q-learning in order to handle the multiple Q-tables. The Q-values were updated by the following formula:

$$Q_{A_t}(s_t,a_t) \leftarrow Q_{A_t}(s_t,a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q_{A_{t+1}}(s_{t+1},a) - Q_{A_t}(s_t,a_t)\right] \quad (2)$$

where Q_{A_t} is the Q-table for an action node A_t in the GP program at the time t, s_t is a state on the real robot at the time t and a_t is a selected real action from the Q-table. Because of this updating, our technique can appropriately calculate Q-values while learning, thereby handling the change of an active state and an active Q-table.

Each Q-table has a preferred action and other actions as variations. With this technique, each Q-table was initialized with a biased initial value.[1] The initial value 0.0001 was entered for the preferred action in each Q-table, while 0.0 was entered for other actions. Therefore, learning is biased when the program is evolved with GP.

The total size of the Q-tables can be larger than that of ordinary Q-learning. Theoretically, convergence to the optimal solution may require more time than ordinary Q-learning. However, the performance of this technique during program execution is relatively good. This is because of the two following factors: first, not all states of the Q-table are used when a robot performs actions according to the program obtained via GP and second, task-based actions are available soon after Q-learning starts.

The "state-action deviation" problem should be taken into account when executing Q-learning in a state constructed from a visual image [10]. This is a problem when optimal actions cannot be achieved due to the dispersion of state transitions. More precisely, if the state is composed of images only, there are often no remarkable differences in image values within an action. To avoid this problem, we redefined "changes" in states. Our definition interprets the current state as unchanged if both the terminal node executed in the program and the executing state of a real robot remain the same.[2] Unless the current state changes, the Q-value is not updated and the same action is repeated.

III. GP Results on the Box-Moving Task

A. Task Definition

We chose the box-moving task as the target task for this study. The task requires a robot to carry a box to the specified goal area and is a single agent problem.

The robot has a camera for its eye and recognizes the box and the goal area in its field of vision. There is no camera positioned outside the robot. Based on its own view, the robot has to decide what actions are needed to achieve the task.

B. GP Part Conducted by the Simulated Robot

1) Simulator: We do not use a highly precise simulator to evolve programs with GP. This use of a simple simulator can help with the adaptability of programs involving a real robot because it prevents programs from overspecializing in a simulation. The simulator in this study uses a robot expressed as a circle on a two-dimensional plane, a box, and a goal area fixed on a field. The task is completed when the robot pushes the box inside the goal area.

We have defined three actions: forward, left-turn, and right-turn. We assumed experimental conditions where the forward action alone is enough to push the box. These actions are ideal ones, i.e., the forward action moves the robot in an absolutely straightforward path and the left-turn action simply turns the robot left. We defined a state space of the environment in the simulator, which expresses the location of the box or the goal area (see Fig. 2). It is a simplified version of the one being

[1]According to the theory, we can initialize Q-values with arbitrary values because Q-values converge to the optimum solution, regardless of their initial value [5].

[2]We modified Asada *et al.'s* definition of "state change" [10] in order to deal with several Q-tables.

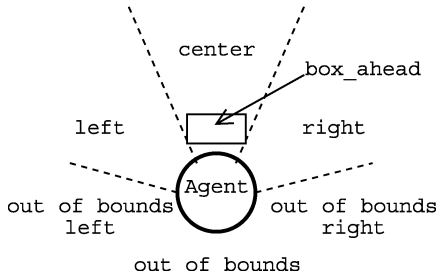


Fig. 2. States for the box and the goal area in the simulator. The area box_ahead is not a state but the determinate for if−box−ahead executing its first argument.

used for a real robot (as described in later experiments). The state "out of bounds" occurs when the box or the goal area is not in view. The states "out of bounds left" and "out of bounds right" are produced when the box or the goal area is not in view and the preceding step was either to the left or to the right, respectively.

The two motion characteristics of the box, as expressed in the simulator, are as follows.

1) The box moves forward if the box comes in contact with the front of the robot as the robot moves ahead. This means the robot is able to push the box with the forward action.
2) If the box is near the center of the robot when the robot turns, then the box remains near the center of the robot after the turn. This means that the robot is able to change its direction holding the box only by left-turn and right-turn actions.

Such actions and the division of states are similar to those of a real robot, which are described later, but are not exactly the same. In addition, physical parameters such as box weight and friction were not measured, and the shape of the robot was not taken into account. Therefore, this simulator is very simple and can be built cost effectively.

2) Settings of GP: We used a terminal set $= \{$move−forward, turn−left, turn−right$\}$ and a function set $= \{$if−box−ahead, box−where, goal−where, prog2$\}$. The terminal nodes correspond to the forward, left-turn, and right-turn actions in the simulator. The functional nodes box−where and goal−where are functions of six arguments, and they execute one of the six arguments, depending on the states (Fig. 2) of the box and the goal area as seen by the robot. The function if−box−ahead, which has two arguments, executes the first argument if the box is located at the box_ahead position in Fig. 2. We arranged conditions so that only the box−where or the goal−where node can become the head of a gene of GP. A gene of GP is set to start executing from its head node; the execution is repeated from the head node if the execution runs over the last leaf node, until it reaches the maximum number of actions. Other parameters used in the GP are summarized in Table I.

A trial starts with the robot and the box randomly placed; it ends when either the box is placed in the goal area or after a predetermined number of actions are performed by the robot. In order for the robot to acquire robust actions, independent of the initial position, the fitness of an individual is derived from the average value of 100 trials by using (3), as shown at the bottom of the page, where fitness_i is the fitness value of the ith trial, given in (4). The second term of the fitness equation represents the penalty given to a larger gene; the constant value 2.0 was determined from preliminary experiments

$$\begin{aligned}\text{fitness}_i = {} & f_{\text{goal}} + f_{\text{remaining\_moves}} + f_{\text{remaining\_turns}} \\ & + f_{\text{move}} + f_{\text{see\_box}} + f_{\text{see\_goal}} + f_{\text{out\_of\_bounds}}. \quad (4)\end{aligned}$$

- If the task is completed, see (5) shown at the bottom of the page. Otherwise, $f_{\text{goal}} = f_{\text{remaining\_moves}} = f_{\text{remaining\_turns}} = 0$.
- If the box is moved at least once, $f_{\text{move}} = 10$. Otherwise, $f_{\text{move}} = 0$.
- If the robot faces the box at least once, $f_{\text{see\_box}} = 1$. Otherwise, $f_{\text{see\_box}} = 0$.
- If the robot faces the goal at least once, $f_{\text{see\_goal}} = 1$. Otherwise, $f_{\text{see\_goal}} = 0$.
- $f_{\text{out\_of\_bounds}} = -\dfrac{(\text{Number of times having lost sight of the box})}{(\text{Number of actions})}$

The constant values, i.e., 100, 10, and 1, were determined from preliminary experiments. These values indicate the relative importance of their conditions, in order to achieve the task. The larger the fitness value is the better the program. Generally, the fitness value of an individual exceeds 114.0.

3) GP Results: Using the fitness function, learning was executed for 1000 individuals over 50 generations with a maximum gene length $= 150$. Learning took 10 min on our Linux system,

$$\text{fitness} = \frac{1}{100}\sum_{i=0}^{99}\text{fitness}_i + 2.0 \cdot \frac{(\text{Maximum number of gene nodes}) - (\text{Number of gene nodes})}{(\text{Maximum number of gene nodes})} \quad (3)$$

$$\begin{aligned} f_{\text{goal}} &= 100 \\ f_{\text{remaining\_moves}} &= 10 \times \left(0.5 - \frac{(\text{Number of moves})}{(\text{Maximum number of moves})}\right) \\ f_{\text{remaining\_turns}} &= 10 \times \left(0.5 - \frac{(\text{Number of turns})}{(\text{Maximum number of turns})}\right) \end{aligned} \quad (5)$$

TABLE I
SPECIFICATION OF THE GP SYSTEM

Terminal set	`move-forward` , `turn-left`, `turn-right`
Function set	`if-box-ahead`, `box-where`, `goal-where`, `prog2`
The representation of the individuals	Tree-based GP
Maximum tree depth	No limitation
Maximum total number of nodes	150 nodes
Initialization method	Grow method (maximum number of nodes is 20)
Crossover operator	One point crossover
Crossover rate	0.85
Mutation operator	Grow method (maximum number of nodes is 15)
Mutation rate	0.15
Selection method	Tournament selection and elite selection (tournament size = 5, elite rate = 0.05)
Population size	1,000
Termination criterion	Maximum number of generation is 50

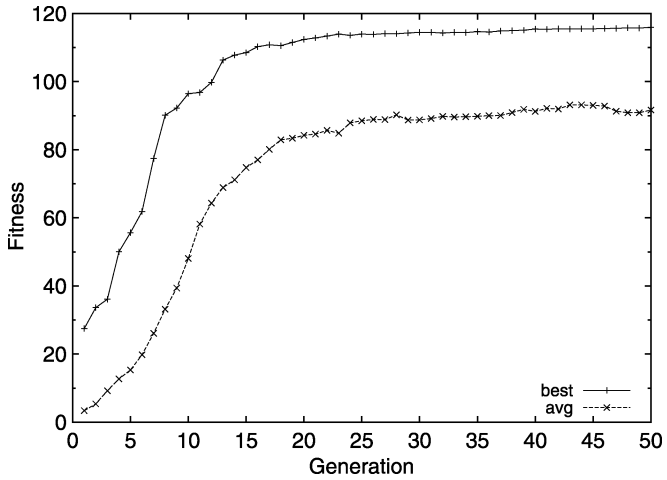


Fig. 3. Averaged fitness values for results of GP.

```
(box-where (turn-left) (turn-left) (turn-right)
(turn-left) (if-box-ahead (goal-where (turn-left)
(turn-left) (turn-right) (turn-left) (move-forward)
(turn-right) ) (move-forward) ) (turn-right) )
```

Fig. 4. The best program evolved through GP.

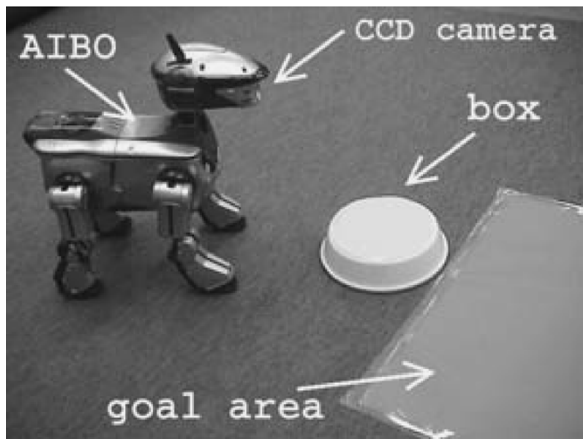


Fig. 5. Photograph of AIBO, the box, and the goal area.

equipped with an Athlon XP 1800+. Fig. 3 shows the fitness results from the GP. Each point is the average value of ten runs. As seen in Fig. 3, the GP successfully evolved programs over the generations.

We applied the best individual to the experiments in later sections. The best evolved individual program is shown in Fig. 4. This is one of the shortest programs to complete successfully the box-moving task.

IV. EXPERIMENT 1: FOUR-LEGGED ROBOT, AIBO

A. Experimental Environment

We used SONY's "AIBO ERS-220" (Fig. 5) as the real robot in this experiment. AIBO's development environment is freely available for noncommercial use, and we can program it using C++ [17]. AIBO has a CCD camera on its head and is equipped with an image processor. It is able to recognize easily objects of specified colors in a CCD image at a high speed.

One of the difficulties with this task is that the robot has four legs. As a result, when the robot moves forward with the box, the box is sometimes moved ahead or deviates to one side, depending on the physical relationship between the box and the AIBO's legs. It is extremely difficult, in fact, to create a precise simulator that accurately expresses the movement of this box. Therefore, adaptation with the real robot is necessary.

B. RL Part Conducted by the Real Robot

1) Action Set: We prepared six selectable robot actions: "forward," "backward," "left-turn," "right-turn," "backward + left-turn," and "backward + right-turn." These actions are far from being ideal, e.g., the "forward" action not only moves the robot straight forward but also has some deviation from side to side, and the "left-turn" action not only turns left but also moves the robot slightly forward. The robot has to learn and compensate for these characteristics in its actions.

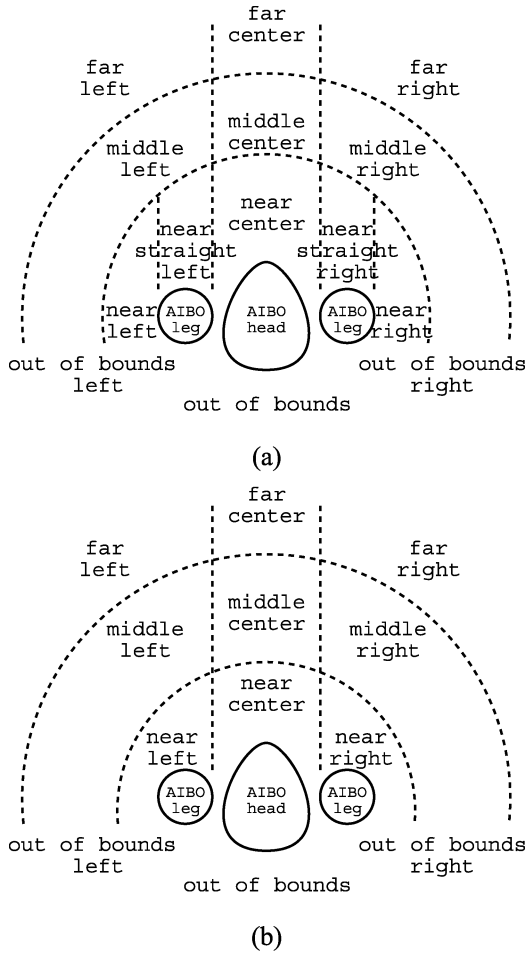


Fig. 6. Defined states for the box and goal area of the real robot "AIBO." The front of the robot faces up in the figure. (a) States for the box in the real robot. (b) States for the goal area in the real robot.

Every action takes approximately four seconds plus four additional seconds to include the swinging of the head, described below; therefore, it is desirable to keep the learning time as short as possible.

2) State Space: The state-space structure is based on positions from which the box and the goal area can be seen in the CCD image, as described in [10]. The viewing angle of AIBO's CCD camera is so narrow that in most cases the box or the goal area cannot be seen well utilizing only unidirectional images. To avoid this difficulty, we added a mechanism to fill in the surrounding images by swinging AIBO's head after each action so that the state recognition can work effectively. This head-swinging operation was uniformly performed throughout the experiment, as it was not an element to be learned.

Fig. 6(a) is the projection of the box's states on the ground surface. The "near center" position is where the box fits between the two front legs. If the robot pushes it forward in this state, the box can be moved. The box remains at the same "near center" position after the robot turns left or right in this state because the robot holds the box between its two front legs. The state with the box not being in view was defined as "out of bounds"; the state with the box not being in view with the preceding step to either the left or right was defined as "out of bounds left" or "out of bounds right."

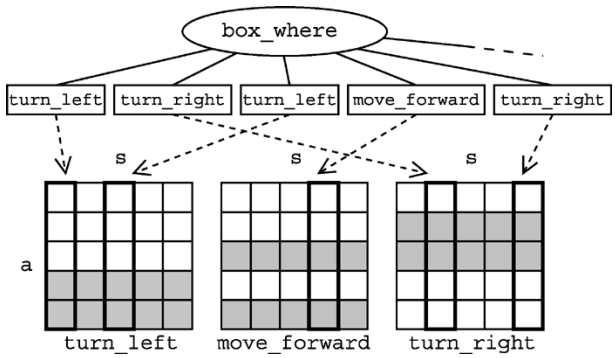


Fig. 7. Action nodes determine the real action according to the Q-value of a real robot's state.

TABLE II
ACTION NODES AND THEIR SELECTABLE REAL ACTIONS FOR "AIBO"

action node	real actions which Q-table can select.
move-forward	"Forward"*, "Backward + left-turn", "Backward + right-turn"
turn-left	"Left-turn"*, "Backward + left-turn", "Backward"
turn-right	"Right-turn"*, "Backward + right-turn", "Backward"

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

We have to pay special attention to the position of the robot's legs. Depending on the physical relationship between the box and AIBO's legs, the movement of the box varies from moving forward to deviating to one side. If an appropriate state space is not defined, then the Markov property of the environment, which is a premise of RL, cannot be met; therefore, optimal actions cannot be found. Consequently, we defined "near straight left" and "near straight right" states at the frontal positions of the front legs. These states do not exist in the simulation because the interaction of the legs and the box is very difficult to simulate.

We defined 14 states for the box in the simulation. We similarly defined states for the goal area, except for "near straight left" and "near straight right" which do not exist [Fig. 6(b)]. There are 14 states for the box and 12 for the goal area; hence, this environment has a total number of states equal to their cross product, 168 states.

3) Integration of GP and RL: Q-learning is conducted in the real robot. Each action node of the GP is assigned Q-tables for Q-learning (see Section II-C2). The real action is selected from Q-tables of the action nodes according to the state of the real robot. Fig. 7 illustrates this situation.

For adaptation, we arranged a preferred action and two variations for each action node; see Table II. The states "near straight left" and "near straight right," which exist only in a real robot, are translated into a "center" state in the functional nodes of GP.

Q-learning's reward parameter is set to 1.0 when the task is achieved and to 0.0 for all other states. We set the learning rate $\alpha = 0.3$ and the discount factor $\gamma = 0.9$. These values were determined from preliminary experiments.

C. Experimental Results With AIBO

We performed the experiment using a real robot. Trials began by placing the robot and the box at random positions. The trial

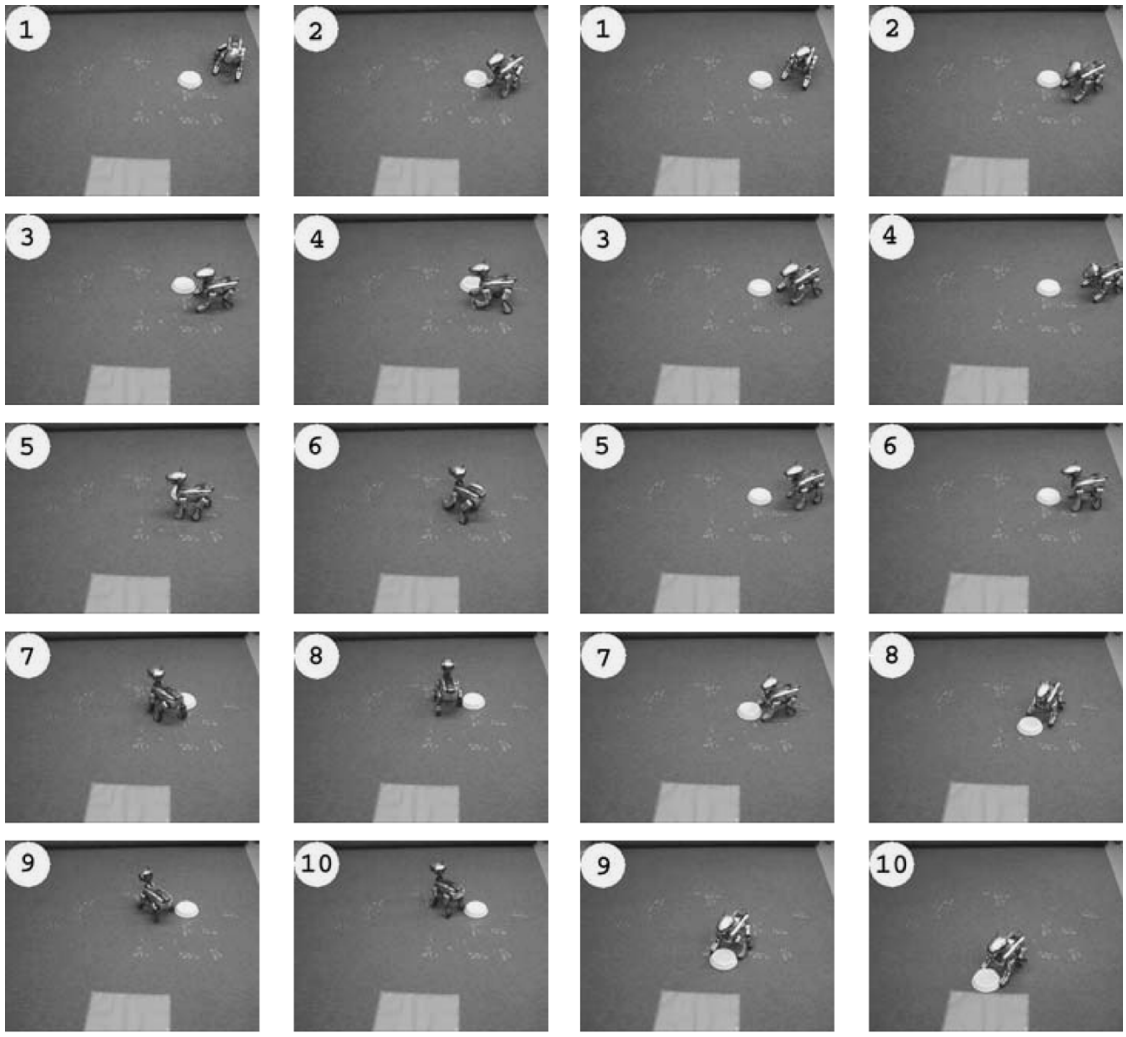


Fig. 8. Two typical series of actions. (a) Failed actions, i.e., losing the box—at the beginning of on-line learning. (b) Successful actions—after ten hours of on-line learning.

was stopped and restarted manually in the following two situations: 1) the robot went out of the field or 2) the number of actions exceeded 90. This experiment was carried out for 10 hours; the robot executed about 4000 actions during that time.

1) Just After Initiating Learning: The real robot succeeded in completing the task after Q-learning started by using the above technique. This was because the robot could perform actions by taking advantage of the results learned via GP.

In situations where the box was placed near the center of the robot, in line with the robot's movements, the robot always achieved the task. On the other hand, 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 would repeatedly turn right to face the box, but would continue moving in vain because it did not have a small enough turning radius, unlike the actions in the simulator. Fig. 8(a) shows a typical series of actions. In some situations, the robot turned right but could not face the box, and its state became "out of bounds" [bottom of Fig. 8(a)].

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

2) After Ten Hours (About 4000 Actions): We observed effective actions, as shown in Fig. 8(b). The robot selected the "backward" or "backward + left (right)-turn" action in situations where it could not complete the task at the beginning of Q-learning [see third and fourth pictures in Fig. 8(b)]. As a result, the robot was able to face the box and push the box forward to the goal, thereby completing the task.

Learning effects were also found in another place. As the robot smoothly approached the box, the number of occurrences of "out of bounds" was reduced. This means that the robot was acting more efficiently than at the beginning of on-line learning.

D. Result of the Adaptation

We show the resultant action from the RL stage in this experiment. Fig. 9(a) and (b) presents the resulting Q-table for `turn_left` with the box state "near left" and the result for

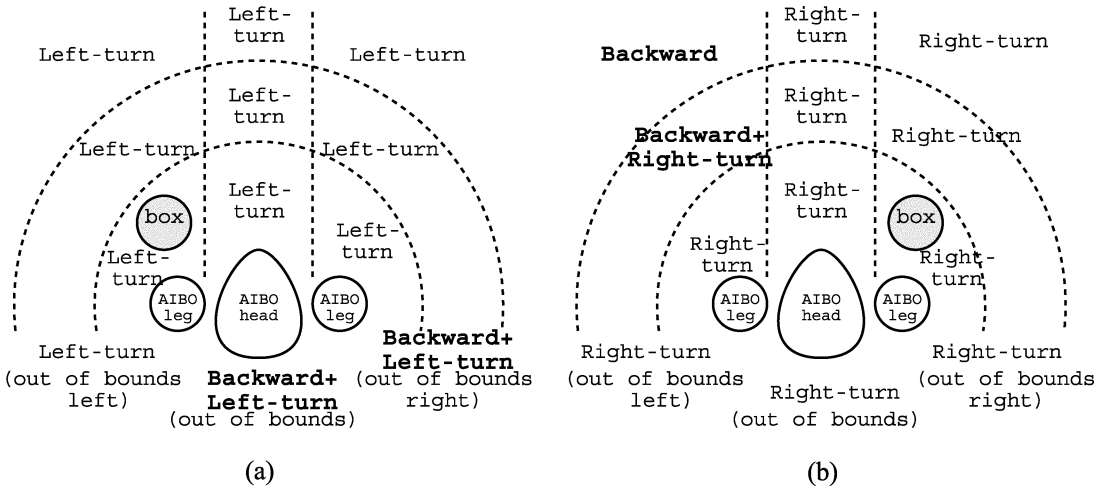


Fig. 9. Resultant actions are in the Q-tables. The state of the box is fixed in each figure. Each action name, in the figures, indicates the action of the highest Q-value for each state of the goal area [see Fig. 6(b) for the state of the goal area]. The bold font indicates that the learned action differed from the initial action for the state. (a) The resulting Q-table for `turn−left`. The state of the box is "near left." (b) The resulting Q-table for `turn−right`. The state of the box is "near right."

`turn−right` with the box state "near right." These are symmetric situations for the ideal robot. However, the motion of the real robot "AIBO" was not symmetric between the "left-turn" and "right-turn" actions. As can be seen from the figures, the acquired knowledge presented different actions in these two situations. Although there is no change in the result of Q-table for `turn−left` [Fig. 9(a)] except for the "out of bounds" states, there are changes of "far left" and "middle left" states in the resulting Q-table for `turn−right` [Fig. 9(b)]. The adoption of "backward," "backward + left-turn" or "backward + right-turn" actions in the result was because the real robot did not have a small enough turning radius for succeeding with the normal turning action. Therefore, this result supports the successful adaptation to the operational characteristics of the real robot using our technique.

E. Experimental Comparison With Traditional Methods

We have compared our proposed technique with two traditional methods: the traditional Q-learning method and the method proposed in [18]. In the Q-learning method, learning is initially done in the simulator and then in an on-line adaptation of the real world; we call this method "$\mathrm{RL}+\mathrm{RL}$." The second method we call "policy transfer."

In the $\mathrm{RL}+\mathrm{RL}$ method, the resulting Q-table from simulation is transferred to the adaptation process in the real robot. The Q-values of states which did not exist in the simulation, i.e., "near straight left" and "near straight right," are filled in by copying those of "near center." Note that this is identical to the mapping of the "center" states in our technique (see Section IV-B3). The Q-values of actions which exist only in the real robot were initialized to 0.0.

Using the policy transfer method, we can obtain a policy, i.e., the best action in each state, through simulation and apply the policy to a real robot. The real robot always behaves according to the policy, updating its Q-table in the Q-learning manner. If the success rate of the task is lower than simulated, the robot executes the exploration process by a means of Q-learning and relearns the states (surrounding the states of failure) (see [18] for further details). In this study, we calculate the success rate by averaging the last four trials. For this calculation, we define the following two situations as "failure": 1) the robot is out of bounds or 2) the number of actions exceeds 40, and the task cannot be achieved within a few more steps, which is decided manually. In the exploration process, the action of relearning states will be chosen from all the real actions based on the Q-values in Q-learning. We set the initial Q-values of real actions, i.e., the ones additionally introduced for the real robot to 0.0. The policy of states only used in the real robot, i.e., "near straight left" and "near straight right," was created by copying the policy of "near center." This is the same mapping for our proposed technique (see Section IV-B3) and the first traditional method $(\mathrm{RL}+\mathrm{RL})$.

For both the $\mathrm{RL}+\mathrm{RL}$ method and the policy transfer method, we used a more detailed state space in the simulation than the one chosen for our technique, in order to carry out Q-learning in the simulator. This is because Q-learning requires detailed states to obtain optimal actions. For example, when the robot is in a "center" state in the simulator (see Section III-B1), it has to distinguish the state of the box, whether it is in front of the robot or far from the robot; GP programs can distinguish these states using the `if−box−ahead` node. Therefore, we introduced distance, i.e., "far," "middle," and "near," in each state. The state space was similar to the real world but not necessarily equal. This is because we have to compare the methods' adaptability to the real world. We did not use the sub-goals with Q-learning in the simulation. Instead, we performed learning for a satisfactory period of time until the Q-values converged in the simulation.

As treatment for the "state-action deviation" problem, we used the solution of Asada *et al.* [10] because these methods are basically normal Q-learning at the stage of on-line learning.

1) Comparison of Performance: For this comparison, we have selected ten typical task situations. The tasks proved to be difficult to complete at the beginning of Q-learning because of the gaps between the simulation and the real robot. We measured the action efficiency after ten hours of Q-learning for each of these ten situations. These tests were executed using

TABLE III
COMPARISON OF PROPOSED TECHNIQUE ($\mathrm{GP} + \mathrm{RL}$), Q-LEARNING ($\mathrm{RL} + \mathrm{RL}$), AND POLICY TRANSFER METHOD

situation #	state notation		GP+RL			RL+RL			Policy Transfer		
			avg. # of actions	lost box	lost goal	avg. # of actions	lost box	lost goal	avg. # of actions	lost box	lost goal
1	Box: near left,	Goal: middle center	**19.7**	0	1	20.0	0	1	19.5	0	1
2	Box: near right,	Goal: middle right	**14.7**	0	0	53.0	2	2	29.0	0	1
3	Box: near left,	Goal: far left	**24.0**	0	1	26.7	0	1	26.0	0	1
4	Box: near straight right,	Goal: far center	**10.3**	0	0	11.0	0	0	10.5	0	0
5	Box: near right,	Goal: far right	**21.7**	0	0	88.0	3	3	41.0	1.3	1.3
6	Box: far left,	Goal: far left	13.5	0	0	**10.5**	0	0	10.0	0	0
7	Box: near straight left,	Goal: far right	26.7	0	1	**26.0**	0	1	27.5	0	1
8	Box: far center,	Goal: far right	23.0	0	1	**13.0**	0	0	23.0	0	1
9	Box: middle right,	Goal: far left	21.5	0	0	**10.5**	0	0	12.5	0	0
10	Box: near left,	Goal: lost	**13.5**	0	0	29.0	0	1	30.0	0	1

a greedy policy so that the robot could always select the best action in each state.

Table III shows the obtained performance of the three methods: our proposed technique ($\mathrm{GP} + \mathrm{RL}$), the Q-learning method ($\mathrm{RL} + \mathrm{RL}$) and the Policy Transfer method. This table represents the average numbers of actions to complete the task ("avg. # of actions" in the table) and the number of occurrences where the robot lost the box or the goal area in completing the task ("lost box," "lost goal" in the table). These data were retrieved from a single run of learning. Each test of a situation was repeated three times.

After comparing our technique ($\mathrm{GP} + \mathrm{RL}$) and Q-learning ($\mathrm{RL}+\mathrm{RL}$), we can confirm that our technique performed much better than $\mathrm{RL}+\mathrm{RL}$ in six situations (shown in bold in Table III). $\mathrm{RL} + \mathrm{RL}$ performed better than the proposed technique in the other four situations, based on the average number of actions. In addition, the robot utilizing the proposed technique lost the box and the goal area less often than $\mathrm{RL} + \mathrm{RL}$. This proves that our proposed technique results in more efficient actions being learned than with $\mathrm{RL} + \mathrm{RL}$.

In comparison, the policy transfer method performed better than our technique in only two situations, based on the average number of actions (underlined numbers in Table III). Our technique outperformed the policy transfer method in five situations; there appears to be no significant difference in situations #1, #4, and #8. In addition, our technique resulted in better performance than the policy transfer method in terms of the number of occurrences of "lost box" and "lost goal." Consequently, we can conclude that the proposed technique is more effective than the policy transfer method.

2) Comparison of the Speed of Learning: Fig. 10 shows the changes in Q-values, for the three methods, in a typical trial with updating during the Q-learning in the real robot. The absolute values of the changes represent how far Q-values are from the convergence. We can observe in Fig. 10 that large changes occurred more frequently with the $\mathrm{RL} + \mathrm{RL}$ method than with our proposed technique. This means that the $\mathrm{RL} + \mathrm{RL}$ method requires more time to converge to the optimal Q-values. One of the reasons seems to be that $\mathrm{RL}+\mathrm{RL}$ has to make on-line adaptation to the optimal Q-values, starting from the ones which had already been learned with the simulator. Thus, we can conclude that our proposed technique results in faster learning than the $\mathrm{RL} + \mathrm{RL}$ method.

The policy transfer method gave rise to infrequent changes in Q-values [see Fig. 10(c)]. However, it may be too soon to conclude which is better based solely on the above comparison. This is because the policy transfer method does not fully explore the search space in order to establish the best actions. We will discuss this aspect later in detail.

3) Conclusions of the Comparisons: The traditional Q-learning method ($\mathrm{RL} + \mathrm{RL}$) had to re-learn in order to adapt to the real world after using the simulation results. The Q-values change as $\mathrm{RL}+\mathrm{RL}$ searches for the optimal Q-values. Because there are differences between the simulation and the real world, the state transitions also differ. In addition, the real robot adaptation used states and actions which do not exist in the simulator. These caused significant changes in the Q-values during learning and took considerable time to converge. Our technique used the value of 0.0001 as Q-values for the preferred actions and the value of 0.0 for the others. Moreover, our technique does not necessarily use all states of the Q-tables because the program, evolved with GP, selects which ones to use during Q-learning. As a result, our technique was not greatly affected by the changes in the state transition when learning Q-values. This enabled our technique to adapt to the real world faster than $\mathrm{RL} + \mathrm{RL}$.

The policy transfer method does not pursue the optimality of the resulting actions. In this method, the robot behaves under the policy obtained in the simulation. The exploration process is executed only when the success rate for task achievement decreases. Therefore, there is no guarantee that the optimal actions can be acquired by this method. In our experiment, we rarely observed complete failure situations in the real world when using the policy acquired by the simulation. Whenever the robot lost the box or the goal area, the robot was able to recover from the failure situation by repeating unidirectional turns. As a result, although it necessitated many extra action steps, the robot achieved the task completely in most cases. In addition, with the

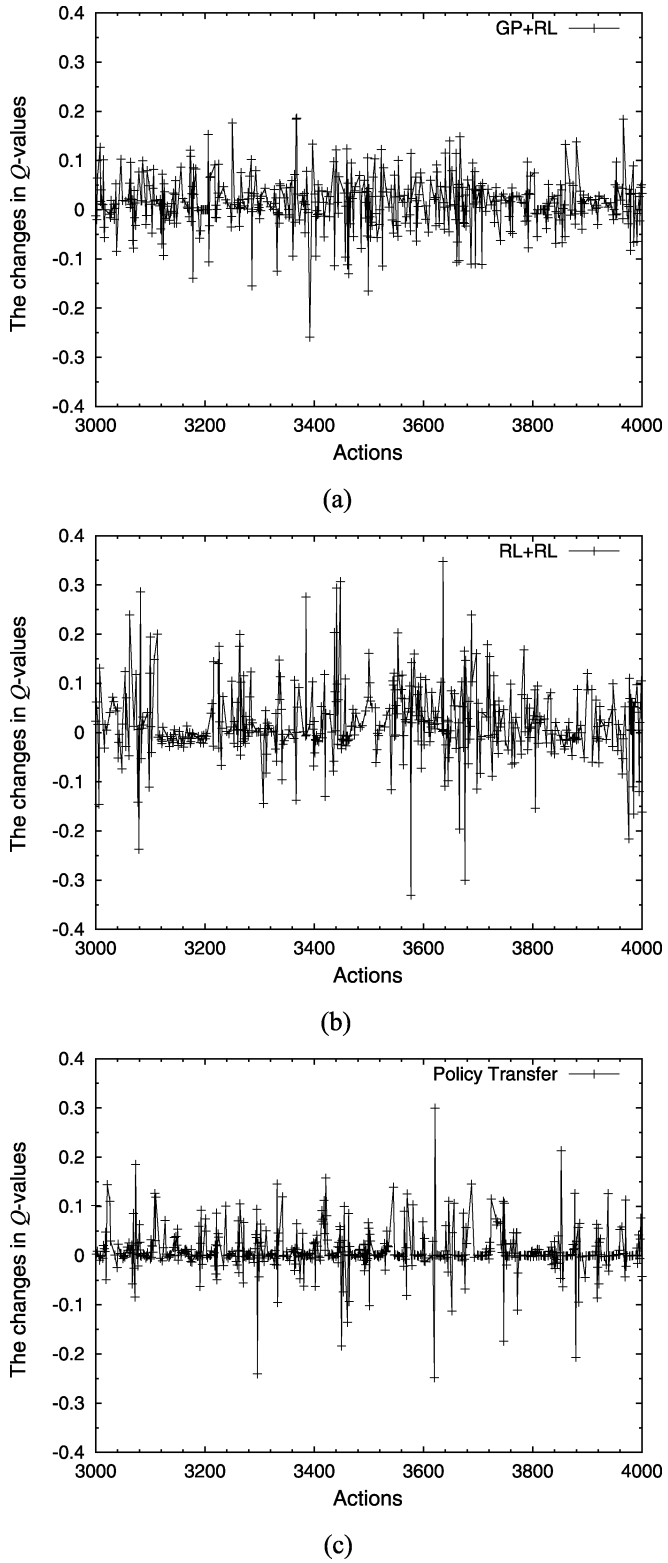


Fig. 10. Comparison of changes in Q-values from about eight to ten hours of Q-learning in a real robot. (a) Proposed technique $(\mathrm{GP} + \mathrm{RL})$. (b) Q-learning $(\mathrm{RL} + \mathrm{RL})$. (c) Policy transfer method.

policy transfer method, the exploration occurs at states proximal to the previous failure states. Thus, if only a slight modification around the failure states is enough to increase the success rate, then the method stops the exploration so that it does not escape from a local optimum. In the above experiments, the success rate was easily recovered by exploring neighboring states around the failure states where the robot either went out of bounds or took more than 40 actions without achieving the task. This exploration resulted in the local optimal solution. This will explain the reason why Fig. 10(c) shows relatively few changes in the Q-values. The method seems to have terminated its explorations at some local optimal actions and stabilized the Q-values used frequently in its policy. On the other hand, the proposed technique explores more effective actions globally so that we can observe a significant difference in performance.

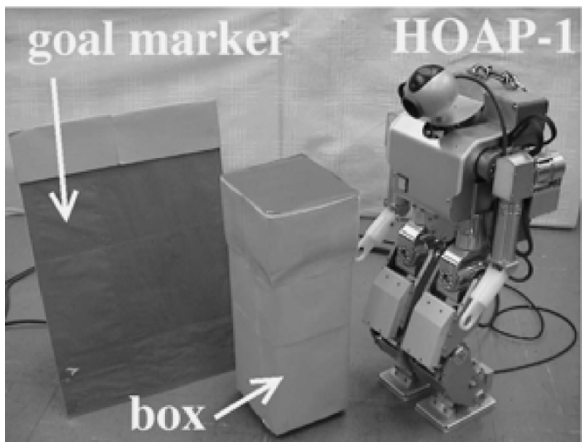


Fig. 11. Robot "HOAP-1," the box, and the goal marker.

TABLE IV
SPECIFICATIONS OF THE ROBOT "HOAP-1"

Height	about 48cm
Weight	about 6kg, including 0.7kg of battery
Joint Mobility	6DOF/foot $\times 2$, 4DOF/arm $\times 2$
Sensors	Joint angle sensor 3-axis acceleration sensor 3-axis gyro sensor Foot load sensor

V. EXPERIMENT 2: HUMANOID ROBOT, HOAP-1

A. Experimental Environment

The robot used in this experiment is "HOAP-1," which is manufactured by Fujitsu Automation Limited. It is a humanoid with 20 degrees of freedom (Fig. 11). The specifications of the robot are shown in Table IV. This robot acts on commands given by a host computer (RT-Linux operating system). It is equipped with a CCD camera in its head, which provides image data for the purpose of environmental understanding.

The target object, a box, has wheels on the bottom so that it can be easily pushed. Note that the force power of a humanoid's arm is so weak that it has difficulty carrying something heavy. Thus, we have chosen to fix the arm position and use a push action for the sake of simplicity. The goal position is marked with a red marker. When the humanoid has pushed the box in front of the red marker, we regard the task as successfully completed.

Moving the box to an arbitrary position is generally difficult. The moving behavior is achieved when the robot pushes the box

with its knees while walking. A humanoid robot is quite different from AIBO in the sense that it stands on one foot while walking, during which time its direction may be diverted by some unexpected disturbance. In particular, if the box is in front of one leg, the box may be pushed forward or sometimes moved outside of the leg area. It is unpredictable because of these physical interactions and friction. It is very difficult to construct a precise simulator which expresses this movement. Therefore, the robot must learn these actions in the real environment.

B. RL Part Conducted by the Real Robot

As you can well imagine, a humanoid robot is very different from an AIBO robot. The differences include not only the shapes of the robots (one has two legs and the other has four legs), but also the viewing angle of the CCD cameras and the dynamics of behaviors.

However, with our technique we can treat the programs for both AIBO and HOAP-1 in the same manner because the fundamental actions, i.e., forward moving and turning, required for the task are common to both of them. Those actions are represented in our simplified simulator. Thus, we have managed to apply the GP program, used for the AIBO experiment, to the humanoid robot. At the stage of RL, the program is expected to adapt to the operational characteristics of the humanoid robot.

1) Action Set: The robot can choose one of the following seven actions: "forward" (six steps), "left-turn," "right-turn," "left-sidestep" (one step to the left), "right-sidestep" (one step to the right), the combination of "left-turn" and "right-sidestep," and the combination of "right-turn" and "left-sidestep." However, these actions are far from ideal. For instance, the robot tends to move slightly backward during the "right-turn" or "left-turn." Thus, it is necessary to adapt to the motion's characteristics. As mentioned above, although the robot has an arm, its power is so weak that we cannot rely on it to move a box.

As is often the case with a real robot, any action gives rise to some error. For example, it is inevitably affected by the slightest roughness of the floor or the friction change due to a balance shift. Thus, even though the robot starts from the same position under the same conditions, it does not necessarily follow the same path.

In addition, every action takes approximately ten seconds. Therefore, it is desirable to keep the learning time as short as possible.

2) State Space: The state-space structure is based on positions from which the box and the goal marker can be seen in the CCD image. This recognition is performed after every action.

Fig. 12(a) and (b) are the projections of the box state and the goal marker state on the ground surface. These state spaces are constructed from rough directions and qualitative distances of objects. We used four levels of distance for the box ("proximate," "near," "middle," "far") and three levels for the goal marker ("near," "middle," "far"). Additionally, we have defined "proximate straight left" and "proximate straight right" states at the frontal positions of the robot's legs.

The state is defined to be "out of bounds" when the robot misses the box or the goal position. Remember that the CCD camera of our robot is fixed in a forward direction and is not movable. As a result of this, the robot often misses the box or goal marker. To recover from the misses, we use the following strategy. The robot records the missing direction, i.e., left or right, when it has missed the box or goal marker. The robot can use this information for a certain number of action steps in order to generate the "out of bounds left" or "out of bounds right" states, which means that the target had disappeared in either the left or right direction within the last few action steps.

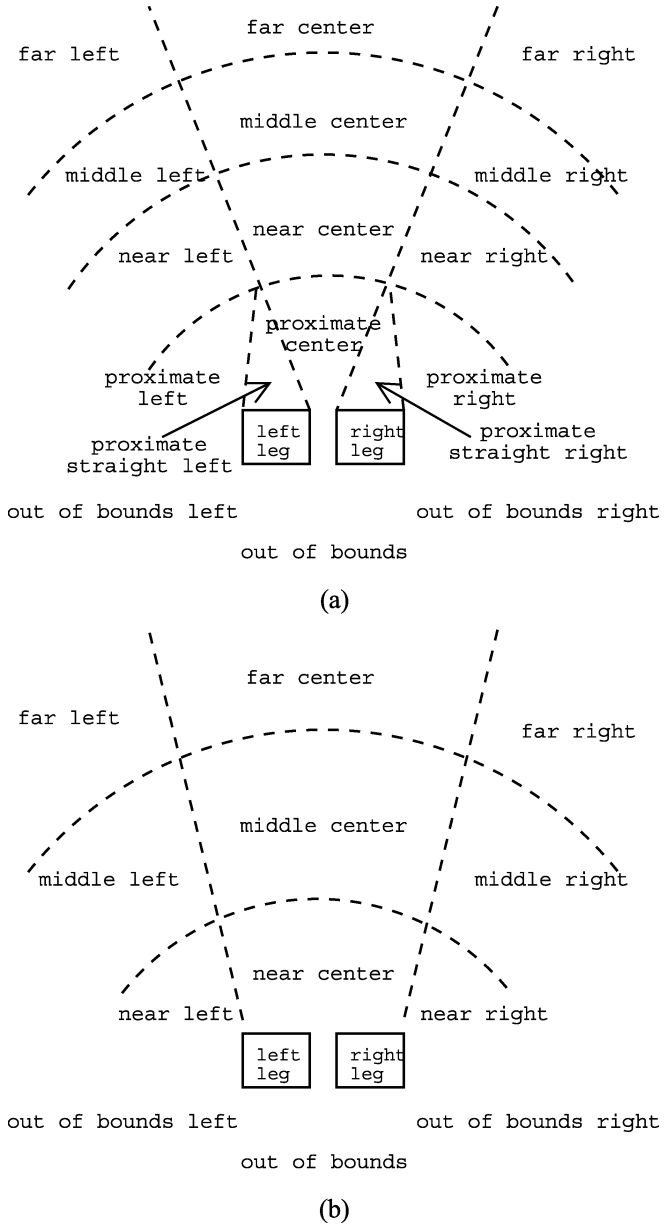


Fig. 12. Defined states in the real robot "HOAP-1." The front of the robot faces up in the figures. (a) States of the box. (b) States of the goal marker.

As shown in Fig. 12(a) and (b), there are 17 states for the box and 14 states for the goal marker. The number of states over the whole environment is represented by their cross product. Hence, there are 238 states in total. When the goal marker is "near center" and the box is near the front of the robot, i.e., one of the states in "box proximate center," "box proximate straight left," "box proximate straight right," and "box near center," the robot recognizes that it has completed the task.

TABLE V
ACTION NODES AND THEIR SELECTABLE REAL ACTIONS FOR "HOAP-1"

action node	real actions which Q-table can select.
`move-forward`	"Forward"*1, "Left-sidestep", "Right-sidestep"
`turn-left`	"Left-turn"*1, "Left-turn + sidestep"*2, "Left-sidestep"
`turn-right`	"Right-turn"*1, "Right-turn + sidestep"*2, "Right-sidestep"

*1 The preferred action which Q-table tends to select initially.

*2 The preferred action if the box is in a state of "proximate center", "near center", "middle center", or "far center".

3) Integration of GP and RL: We applied the $Q(\lambda)$-learning method to a real robot in this experiment. $Q(\lambda)$-learning is a variant of Q-learning and is more efficient than normal Q-learning. This method records the trace of visited states and actions taken. When a temporal difference (TD) error occurs, Q-values involved in the trace are assigned credit or blame for the error. We implemented "naive $Q(\lambda)$" with a replacing trace as described in [5].

We defined a preferred action for each Q-table of the terminal node. For instance, in the case of the Q-table for `move_forward`, the forward action tends to be selected more often than the Q-tables for `turn_left` and `turn_right`. However, when the box is in a state of "proximate center," "near center," "middle center," or "far center," then the actions for the combination of left (right)-turn and sidestep are chosen with a greater frequency. This maintains consistency with the simulator's results. In other words, when the robot performs a combination of turn and sidestep in the state of "proximate center," then the next state remains "proximate center." The preferred action and its variations are summarized for each action node in Table V.

Some translations of states are required to run a GP individual on a real robot. We translated the states "proximate straight left" and "proximate straight right," which exist only in the real robot, into a "center" state for the function nodes of the GP. If the box is in the "proximate center" state for the real robot, then the `if_box_ahead` node executes its first argument.

The reward is set to 1.0 when the task is achieved and to 0.0 for all other states. As for other $Q(\lambda)$-learning parameters, we chose the learning rate $\alpha = 0.3$, the discount factor $\gamma = 0.8$ and the trace-decay parameter $\lambda = 0.5$. These parameters were determined from preliminary experiments.

C. Experimental Results With a Humanoid Robot HOAP-1

In this experiment, using the humanoid robot, the starting state was limited to an arrangement in which both the box and the goal marker were visible. This was justified for one reason; even if learning is performed from an arrangement in which either the box or the goal is "out of bounds," it cannot be predicted that the box or goal position will subsequently become visible. Thus, there will be substantial variations in state transitions, and a long period of time will be required for learning.

For a single trial, learning was performed until the robot moved out of bounds or the predetermined number of action steps, i.e., 30 action steps, was exceeded. Learning in the real robot was performed for six hours.

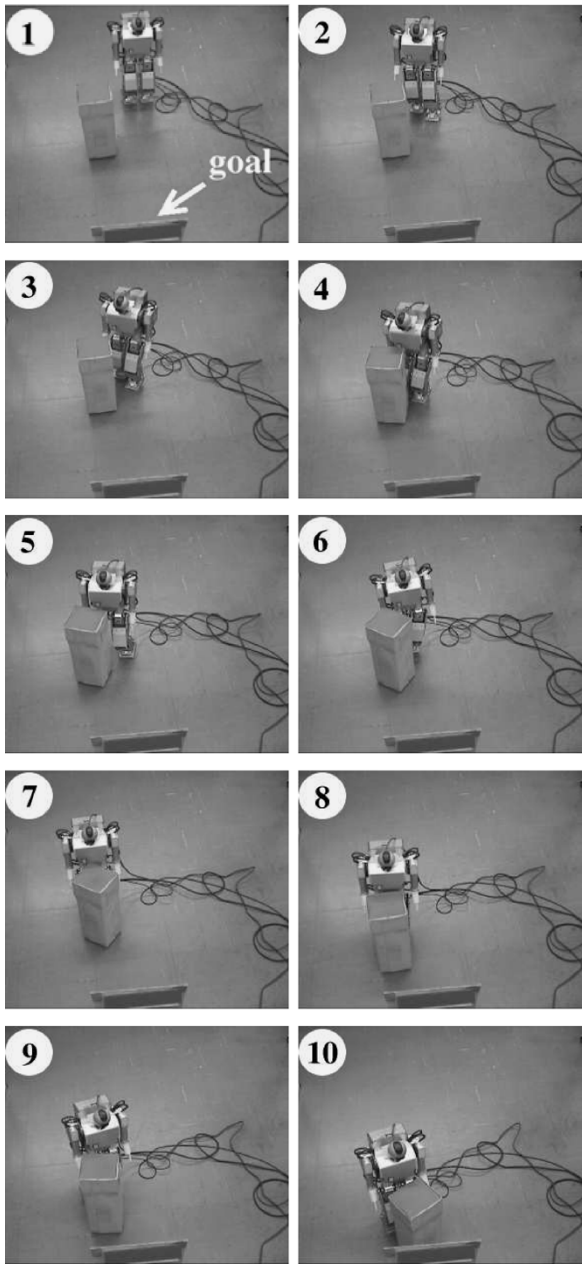


Fig. 13. One successful action series. The goal is at the bottom center of each figure.

a) Just After Initiating Learning: The robot succeeded in completing the task in many situations. This is because the robot acted relatively well using the program evolved with the simulator, although the operational characteristics differ from AIBO.

However, the robot took a long time in some situations. This proves that the actions acquired from the simulator are not always effective in a real environment because of the differences between the simulator and the real robot. These differences necessitate two added box states for the real robot, i.e., "proximate straight left" and "proximate straight right." The operational characteristics in these states are unknown to the robot before on-line learning.

b) After Six Hours (About 1800 Actions): Noticeably improved actions were observed. The robot selected very appropriate actions in the situations. Fig. 13 shows one successful

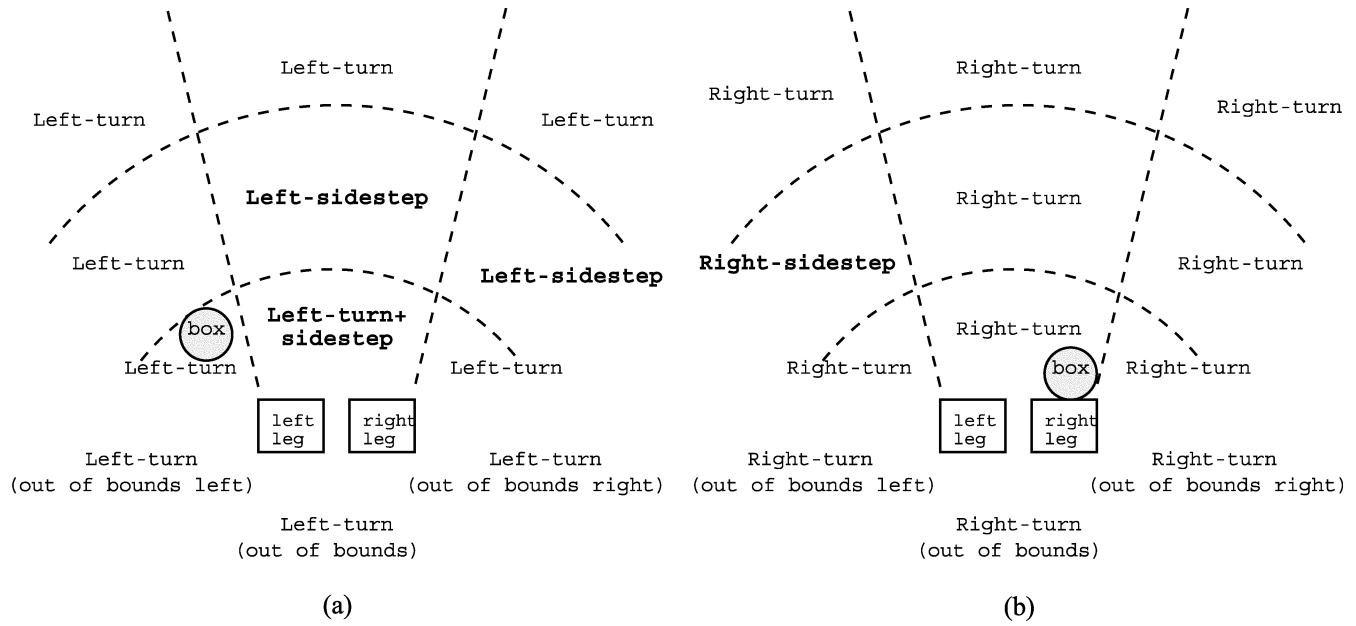


Fig. 14. Resultant actions are in the Q-tables. The state of the box is fixed in each figure. Each action name, in the figures, indicates the resultant action for each state of the goal marker [see Fig. 12(b) for the state space of the goal marker]. The bold font indicates that the learned action differed from the initial action for the state. (a) The resulting Q-table for `turn−left`. The state of the box is "near left." (b) The resulting Q-table for `turn−right`. The state of the box is "proximate straight right."

action sequence. It completed the task much faster than it did before on-line learning.

The same improvement was observed as described in the experiment on AIBO (Section IV-C). This underscores the effectiveness of our approach. Since GP succeeded in learning some general knowledge, in the sense that its usage is not limited to a particular robot, it is applicable to both AIBO and HOAP-1.

D. Result of the Adaptation

We present the acquired Q-tables as the result of the adaptation by our technique. Fig. 14(a) and (b) are the resulting Q-tables for the `turn−left` and `turn−right` action nodes with the box state of "near left" and "proximate straight right," respectively. The robot learned to use the "sidestep" action in situations where the box is near to the robot. This is because the real robot does not have a small turning radius. The real robot has difficulty in approaching the box using normal turning actions, while the simulated robot does not. Therefore, the "sidestep" action was required for adaptation from the simulated robot to the real robot.

E. Measurement of Improvement

We performed a quantitative comparison to investigate how efficiently the robot performed after learning. For this comparison, we randomly selected six situations: four situations were selected from states added just for the real robot ("box proximate straight left" and "box proximate straight right") and the other two were selected from other states. We measured the number of actions for completing the task both before and after on-line learning. The tests were executed using a greedy policy in order to insure that the robot always selects the best action in each state.

TABLE VI
COMPARISON OF THE NUMBER OF ACTIONS BOTH BEFORE AND AFTER LEARNING

situation #	state notation	avg. number of actions	
		before Q-learning	after Q-learning
1	Box: proximate straight right Goal Marker: middle right	**7.3**	9.3
2	Box: proximate straight right Goal Marker: middle left	10.7	**5.7**
3	Box: proximate straight left Goal Marker: middle right	16.3	**14.7**
4	Box: proximate straight left Goal Marker: far right	**11.7**	13.0
5	Box: near left Goal Marker: far center	14.7	**10.7**
6	Box: middle right Goal Marker: far center	11.3	**10.9**

Table VI shows the results. This test was carried out using the result from a single run of learning. These data are the average of three values, each situation was repeated five times and the largest and the smallest values were removed. After learning, the robot completed the task more efficiently in four out of six situations (represented in bold in the table). In particular, great improvement was observed in situation #2. These results prove that our technique worked very well and that the robots learned efficient actions.

There is another point to consider in terms of the efficiency. We had to deal with the "state-action deviation" problem when applying $Q(\lambda)$-learning to this experiment. As a solution to this problem, the same action should be repeated until the current

TABLE VII
COMPARISON OF THE NUMBERS OF STATE TRANSITIONS

	avg. number of state transitions	
situation #	before learning	after learning
1	**7.0**	8.7
2	10.3	**4.0**
3	16.0	**14.7**
4	11.7	**10.7**
5	14.7	**6.7**
6	10.3	**9.9**

state changes (see Section II-C2). The Q-values are updated when the state changes.

In $Q(\lambda)$-learning, as well as in the usual RL, the agent learns optimal actions in order to maximize the sum of the discounted rewards, which it receives until completing the task [5]. The sum, which is called the expected discounted return, can be written as follows:

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1} \tag{6}$$

where t is a current time step, T is the last time step, and γ is the discount factor $(0 \leq \gamma \leq 1)$. r_{t+k+1} means a reward received $k+1$ time steps in the future. In this experiment, the reward is defined as $r = 1.0$ only when the task is completed; otherwise $r = 0.0$. The equation can be written simply as

$$R_t = \gamma^T. \tag{7}$$

This forces the agent to minimize T, i.e., the number of steps, in achieving the task. Each step is a state transition because of the treatment of the "state-action deviation." Therefore, we also have to compare the number of state transitions in completing the task so as to investigate how efficiently the robot behaves.

Table VII shows the number of state transitions in the selected situations. This table illustrates that the performance in situation #4 was improved after learning, in terms of the number of state transitions. This is evidence that the learning process of our technique in the real environment is effective.

In situation #1, unpredictable movement of the box was observed many times. This unpredictability resulted in the longer convergence, which means that it took much longer to learn.

In situation #4, the number of actions was not decreased after learning; however, the number of state transitions did become smaller. One reason seems to be that the division of the state space is not appropriate. Since the division is fixed during $Q(\lambda)$-learning, we cannot expect much improvement in cases with incorrect state-space division.

VI. DISCUSSION

A. Related Works

There are many studies combining evolutionary algorithms and RL [19], [20]. Although these approaches differ from our proposed technique, we have seen several studies in which GP and RL were combined [14], [15]. With these traditional techniques, Q-learning was adopted as RL, and an individual of GP represented the structure of the state space to be searched. It is reported that searching efficiency was improved in the QGP method [14], compared with the traditional Q-learning. The techniques used in these studies, however, are also a type 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. 1(b). An inordinate amount of time would be required for learning if the whole process was directly applied to a real robot. As a result, no studies using these techniques have been reported using a real robot.

Noise in simulators is often essential in overcoming the differences between a simulator and real environment [11]. One real robot, which learned with our technique, showed a sufficient performance in the noisy real environment even though the robot had learned while in an ideal simulator. One reason seems to be that the coarse state division can absorb the image processing noise. We plan to conduct a comparative experiment on the robustness produced by our technique with that of noisy simulators.

As described in Section V-E, it seems that the division of the state space is not appropriate in some situations. It is difficult for the robot to improve its actions in certain situations because the division is fixed in the learning process. Takahasi *et al.* proposed two methods of segmenting a state space automatically [22]. In the first method, the real robot moves in its environment and samples data. After that, the state space is segmented, constructing local models of inputs. They pointed out that the method requires uniformly sampled data to construct an appropriate state space. The robot in our experiment takes ten seconds per action. Under this condition, uniform sampling is not reasonable because it would take an enormous amount of time. Although the second method segments the state space incrementally on-line, it also seems to require sampling of a lot of data to construct a sufficient state space. It may be time-consuming for our robots, but it is still an interesting approach.

We used several predefined actions for AIBO and HOAP-1. This is a shortcut to investigate the applicability of evolutionary methods to such a high-level function as solving a task. In contrast, there are related studies that evolve the low-level functions of robots. For instance, Ziegler *et al.* evolved the gaits of a small humanoid robot with a physical simulation by means of GP [23]. Endo *et al.* evolved the walking behavior of a humanoid robot "PINO" with GA and tested it in a real robot [24]. They also experimented with the coevolution of the morphology and the controller of a humanoid robot in the simulation. There are many other researches approaching real robots by evolutionary methods in simulation. However, the application to real robots is a challenging area. Hornby *et al.* evolved gaits of a prototype version of AIBO with GA [25]. Nordin *et al.* developed a humanoid robot "ELVIS" [26]. The software of this robot was built mainly on GP. They experimented on the evolution of stereoscopic vision [27] and on hand-eye coordination [28].

B. Future Researches

We chose only several discrete actions in this study. Although this simplified matters and made the task easier to handle, studying continuous actions would be more realistic in other applications. For example, "turn left in 30.0°" at the beginning of RL can be changed to "turn left in 31.5°" after learning, depending on the operational 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 multiagent tasks. One example is the cooperation task. There are researches on cooperation tasks with evolutionary methods. For example, Uchibe *et al.* obtained a cooperative behavior from independent robots in a simple soccer task by means of GP [29]. One of the authors experimented successfully with a cooperative transportation task of two humanoid robots with Q-learning using master–slave cooperation [30]. Taking these results into account, it would be possible to handle a multiagent task by extending our approach. The control programs of multiple robots are acquired in the simulation. The adaptation to real robots may be possible while they are carrying out a cooperative task in a real-world situation. We are currently working on this task for future research.

Another extension is to adapt the simulation parameters or modify the state space by using the information available from a real environment. Simulator tuning would require a feedback process, as described in the dotted lines in Fig. 1(a). This would enable GP to evolve more effective programs. Note that programs evolved by GP cannot be run without a state space. A rough state space can be given initially and then modified gradually, according to the robot's characteristics, which will establish a more effective learning scheme.

The amount of time needed for learning in a real robot is still a big problem for our technique. It is important to make the simulation model more detailed and construct more effective controllers by GP. We are conducting research on the incremental refinement of the simulation model [31]. It constructs a simulation model based on the data retrieved from a real environment. The controller of the robot is trained from the acquired simulation model. The trained controller is transferred to the robot and is responsible for achieving tasks. Whenever the robot moves in the real environment, we acquire additional data about the environment. Using that data, we refine the simulation model and train the controller simultaneously. Hence, controller learning can be accelerated. We confirmed from simulation experiments that the method successfully refined the model and evolved effective controllers with less data from the real world. Therefore, the problem of time can be improved by just such an appropriate method.

VII. Conclusion

In this paper, we proposed a technique for adapting the actions of a robot to its real environment based on the integration of GP and RL techniques; we then experimentally verified its effectiveness.

At the initial stage of Q-learning with AIBO, we sometimes observed unsuccessful displacements of the box due to a lack of data concerning the real robot's characteristics, which had not been reproduced by the simulator. The technique, however, adapted to the operating characteristics of the real robot through a ten hours learning period.

We applied the same evolved programs to a humanoid robot HOAP-1. We confirmed that, after six hours of learning, effective adaptation was established according to the robot's operational characteristics so as to solve the task effectively. Our approach was successful in acquiring the common program, which is applicable to heterogeneous robots.

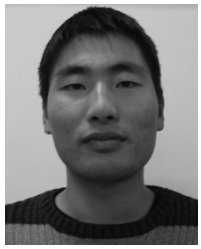
Acknowledgment

The authors would like to thank H. Mitsuhashi for his support with the experiment using AIBO.

References

[1] J. R. Koza, *Genetic Programming, On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press, 1992.
[2] D. P. Muni, N. R. Pal, and J. Das, "A novel approach to design classifiers using genetic programming," *IEEE Trans. Evol. Comput.*, vol. 8, no. 2, pp. 183–196, Apr. 2004.
[3] S. Nolfi and D. Floreano, *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. Cambridge, MA: MIT Press, 2000.
[4] M. V. Butz, T. Kovacs, P. L. Lanzi, and S. W. Wilson, "Toward a theory of generalization and learning in XCS," *IEEE Trans. Evol. Comput.*, vol. 8, no. 1, pp. 28–46, Feb. 2004.
[5] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.
[6] B. Andersson, P. Svesson, M. Nordahl, and P. Nordin, "On-line evolution of control for a four-legged robot using genetic programming," in *Real-World Applications of Evolutionary Computing*. ser. Lecture Notes in Computer Science, S. Cagnoni *et al.*, Eds. Berlin, Germany: Springer-Verlag, 2000, vol. 1803, pp. 319–326.
[7] K. Yanai and H. Iba, "Multi-agent robot learning by means of genetic programming: solving an escape problem," in *Proc. 4th Int. Conf. Evolvable Syst. ICES'2001, Evolvable Syst. From Biol. Hardware.*, Y. Liu *et al.*, Eds., Tokyo, 2001, pp. 192–203.
[8] D. Floreano and F. Mondada, "Evolution of homing navigation in a real mobile robot," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 26, no. 3, pp. 396–407, 1996.
[9] H. Kimura, T. Yamashita, and S. Kobayashi, "Reinforcement learning of walking behavior for a four-legged robot," in *Proc. 40th IEEE Conf. Decision Control*, 2001, pp. 411–416.
[10] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda, "Purposive behavior acquisition for a real robot by vision-based reinforcement learning," *Mach. Learn.*, vol. 23, pp. 279–303, 1996.
[11] N. Jacobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," in *Lecture Notes in Artificial Intelligence*, vol. 929, Proc. ECAL'95, 1995, pp. 704–720.
[12] [Online]. Available: http://www.sony.net/Products/aibo/
[13] [Online]. Available: http://www.automation.fujitsu.com/en/products/products07.html
[14] H. Iba, "Multi-agent reinforcement learning with genetic programming," in *Proc. 3rd Annu. Genetic Program. Conf.*, 1998, pp. 167–175.
[15] K. L. Downing, "Adaptive genetic programs via reinforcement learning," in *Proc. Genetic Evol. Comput. Conf. (GECCO-2001)*, L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, Eds., San Francisco, CA, 2001, pp. 19–26.
[16] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming—An Introduction*. San Mateo, CA: Morgan Kaufmann, 1998.
[17] OPEN-R Programming Special Interest Group, "Introduction to OPEN-R programming," (in Japanese), Impress Corporation, LOCATION, 2002.
[18] T. Minato and M. Asada, "Environmental change adaptation for mobile robot navigation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 1998, pp. 1859–1864.

[19] D. E. Moriarty, A. C. Schultz, and J. J. Grefenstette, "Evolutionary algorithms for reinforcement learning," *J. Artif. Intell. Res.*, vol. 11, pp. 199–229, 1999.
[20] M. Dorigo and M. Colombetti, *Robot Shaping: An Experiment in Behavior Engineering*. Cambridge, MA: MIT Press, 1998.
[21] A. C. Schultz, C. L. Ramsey, and J. J. Grefenstette, "Simulation-assisted learning by competition: Effects of noise differences between training model and target environment," in *Proc. 7th Int. Conf. Mach. Learn.*, 1990, pp. 211–215.
[22] Y. Takahashi, M. Asada, S. Noda, and K. Hosoda, "Sensor space segmentation for mobile robot learning," in *Proc. ICMAS'96 Workshop Learn. Interaction Organ. Multiagent Environ.*, 1996.
[23] J. Ziegler, J. Barnholt, J. Busch, and W. Banzhaf, "Automatic evolution of control programs for a small humanoid walking robot," in *Proc. 5th Int. Conf. Climbing Walking Robots (CLAWAR)*, 2002, pp. 109–116.
[24] K. Endo, F. Yamasaki, T. Maeno, and H. Kitano, "Co-evolution of morphology and controller for biped humanoid robot," in *Lecture Notes in Computer Science*, vol. 2752, Proc. RoboCup 2002, G. A. Kaminka, P. U. Lima, and R. Rojas, Eds., 2003, pp. 327–341.
[25] G. S. Hornby, M. Fujita, S. Takamura, T. Yamamoto, and O. Hanagata, "Autonomous evolution of gaits with the Sony quadruped robot," in *Proc. Genetic Evol. Comput. Conf.*, vol. 2, Orlando, FL, Jul. 1999, pp. 1297–1304.
[26] J. P. Nordin and M. Nordahl, "An evolutionary architecture for a humanoid robot," presented at the 4th Int. Symp. Artif. Life Robotics, Oita, Japan, 1999.
[27] C. T. Graae, P. Nordin, and M. Nordahl, "Stereoscopic vision for a humanoid robot using genetic programming," in *Real-World Applications of Evolutionary Computing*. ser. Lecture Notes in Computer Science, S. Cagnoni *et al.*, Eds. Berlin, Germany: Springer-Verlag, 2000, vol. 1803, pp. 12–21.
[28] W. B. Langdon and P. Nordin, "Evolving hand-eye coordination for a humanoid robot with machine code genetic programming," in *Lecture Notes in Computer Science*, vol. 2038, Proc. 4th Eur. Conf. EuroGP 2001, J. Miller *et al.*, Eds., Lake Como, Italy, Apr. 18–20, 2001, pp. 313–324.
[29] E. Uchibe, M. Nakamura, and M. Asada, "Cooperative behavior acquisition in a multiple mobile robot environment by co-evolution," in *RoboCup-98: Robot Soccer World Cup II*. ser. Lecture Notes in Computer Science, H. K. M. Asada, Ed. Berlin, Germany: Springer-Verlag, 1999, vol. 1604, pp. 273–285.
[30] Y. Inoue, T. Tohge, and H. Iba, "Cooperative transportation by humanoid robots—learning to correct positioning," in *Proc. Hybrid Intell. Syst.*, 2003, pp. 1124–1133.
[31] S. Kamio and H. Iba, "Evolutionary construction of a simulator for real robots," in *Proc. Congr. Evol. Comput.*, 2004, pp. 2202–2209.



Shotaro Kamio received the B.Tech. degree in information and communication engineering from the University of Tokyo, Tokyo, Japan, in 2002 and the M.Sc. degree from the Graduate School of Frontier Sciences, University of Tokyo, Tokyo, Japan, in 2004. He is currently working towards the Ph.D. degree in frontier informatics at the Graduate School of Frontier Sciences, University of Tokyo.

He is interested in intelligent behaviors of real robots with machine learning methods.

Hitoshi Iba (M'99) received the Ph.D. degree from University of Tokyo, Tokyo, Japan, in 1990.

From 1990 to 1998, he was with the ElectroTechnical Laboratory (ETL), Ibaraki, Japan. He has been with the University of Tokyo, Tokyo, Japan, since 1998. He is currently a Professor at the Graduate School of Frontier Sciences, University of Tokyo. His research interest includes evolutionary computation, genetic programming, bioinformatics, foundation of artificial intelligence, and robotics.

Dr. Iba is an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and the *Journal of Genetic Programming and Evolvable Machines* (GPEM).