Real-time Adaptation Technique to Real Robots: An Experiment with a Humanoid Robot

Shotaro Kamio   Hitoshi Iba
Graduate School of Frontier Science, The University of Tokyo,
7-3-1 Hongo, Bunkyo-ku, Tokyo, 113-8656, Japan.
{kamio,iba}@iba.k.u-tokyo.ac.jp

Abstract- We introduce a technique that allows a real robot to execute real-time learning, in which GP and RL are integrated. In our former research, we showed the result of an experiment with a real robot "AIBO" and proved the technique performed better than the traditional Q-learning method. Based on the proposed technique, we can acquire the common programs using a GP, applicable to various types of robots. We execute reinforcement learning with the acquired program in a real robot. In this way, the robot can adapt to its own operational characteristics and learn effective actions. In this paper, we show the experimental results in which a humanoid robot "HOAP-1" has been evolved to perform effectively to solve the box-moving task.

1 Introduction

We can use techniques of machine learning when applying a robot to achieve some task, while appropriate actions are unknown in advance. In such situations, the robot can learn appropriate actions in a try-and-error manner in a real environment. Well-known techniques for this purpose are Genetic Programming (GP) [11], Genetic Algorithm (GA) [14] and Reinforcement Learning (RL) [17].

GP can directly generate programs to control the robot. There are many studies in which GP is applied to control real robots [1, 19]. We can use a GA in the combination with a Neural Network (NN) to control robots [14]. The evaluation of real robots demands a significant amount of time because of their mechanical actions. Moreover, we have to repeat the evaluations of many individuals over several generations in both GP and GA. For example, Andersson et al. spent 15 hours for the evaluation of 111 generations to acquire a galloping behavior by GP [1], and Floreano et al. spent 10 days for 240 generations to evolve the motion of going toward a light source by GA with NN [5]. Therefore, in most of these studies, the learning was conducted through simulation and the results were applied to real robots.

RL is capable of finding optimal actions from interactions with the environment. To obtain optimal actions using RL, it is necessary to repeat learning trials time and again. The enormous learning time of 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 [10], or loading the results learned with a simulator into a real robot [2].

Learning by simulation requires the simulator to represent agents, the environment and their interactions precisely. However, there are many tasks that are difficult to make the simulator precise. Learning with an imprecise simulator does not lead to effective performance in a real environment. Furthermore, there are certain variations due to minor errors in the manufacturing process or to changes with time even if the real robots are modeled exactly the same way. The above approach, i.e., to learn with a simulator and to apply the result to a real robot, has limitations. Therefore, learning with a real robot is unavoidable in order to acquire optimal actions.

Now that various kinds of robots have been developed, it is very important to carry out a task by employing those different robots. In this case, it will take much longer if we allow each different robot has to learn the task independently. On the contrary, we can use some general knowledge for the common part of various robots and then fine-tune specific parameters for a particular robot, thereby establishing a more effective learning scheme.

In our previous paper [8], we have proposed a technique that allows a real robot to execute real-time learning in which GP and RL are integrated. Our technique does not need a precise simulator because learning is done with a real robot. In other words, the precision requirement can be easily met if the task is only expressed in a proper way. As a result of this concept, we can greatly reduce the cost to make the simulator highly precise. Since GP can generate programs, our technique has the possibility of producing more complex behaviors than the reactive behaviors of RL.

Based on the proposed technique, we can apply the program, which has been acquired with the same simulator via GP as described in [8], to different types of robots. The program will adapt to the operational characteristics of real robots and learn effective actions. In our former research, we used a robot "AIBO" (an entertainment four-legged robot by SONY) and proved the effectiveness of our proposed technique. In this research, we perform an experiment with a humanoid robot "HOAP-1" (manufactured by Fujitsu Automation Limited). We provide the experimental results to confirm the adaptation ability of our methodology.

This paper is organized as follows. The next section explains the task of the experiment. After that, Section 3 ex-

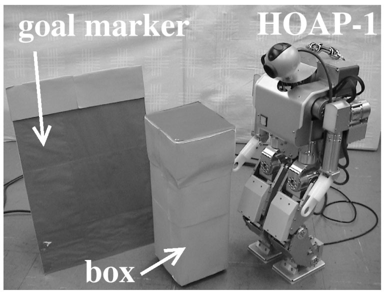


Figure 1: The robot “HOAP-1”, the box and the goal marker.

Table 1: The specification 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

plains our proposed technique and settings in this experiment. Section 4 presents an experimental result with a humanoid robot “HOAP-1” and Section 5 discusses results of comparison and future research. Finally, a conclusion is given in Section 6.

2 Task Definition

The task is the box-moving task, the goal of which is to move a box to a pre-defined destination area. It is the same as our previous works with AIBO [8].

The robot used in this paper is “HOAP-1” which is manufactured by Fujitsu Automation Limited. It is a humanoid of 20 degrees of freedom (Fig. 1). The specification of the robot is shown in Table 1. This robot makes an action from a command given by a host computer (RT-Linux operating system). It is also equipped with a CCD camera in its head, which provides image data for the purpose of environmental understanding.

The target object, i.e., the 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 the 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 this red marker, we regard the task has been achieved successfully.

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 its direction may be diverted unexpectedly with some disturbances. 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 the physical interactions or their frictions. It is very difficult to construct a precise simulator which expresses this movement. The robot must, therefore, learn these actions in a real environment.

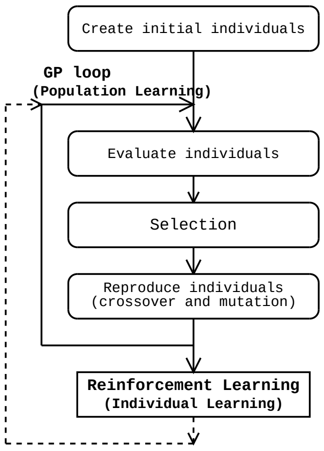


Figure 2: The flow of the algorithm.

3 Real-time Adaptation Technique

In our former research [8], we proposed a technique that integrates GP and RL. The algorithm is shown in Fig. 2. This technique enables us (1) to speed up learning in real robot and (2) to adapt to a real robot using the programs acquired in a simulator.

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

Step 1. Carry out GP in a simplified simulator, and formulate programs that have the standard actions required for executing a task.

Step 2. Conduct individual learning (= RL) after loading the programs obtained in Step 1 above.

In the first step above, the programs for the standard 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.

3.1 RL part conducted by the real robot

As you can well imagine, a humanoid robot is very different from the robot AIBO used in our former research. The difference includes not only the shapes of the robots (one has two legs and the other has four legs), but also the viewing angle of CCD camera and the dynamics of behaviors.

However, our technique 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. GP is applied by using the simple simulator so as to evolve such common programs (the details of GP are described in Sect. 3.2). At the stage of reinforcement learning, the effective adaptation is done according to the operational characteristics of a particular robot.

3.1.1 Action set

The robot can choose one of the seven actions: Forward (6 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 a little backward during the Right-turn or Left-turn. Thus, it is necessary for the robot to adapt the motion characteristics. As mentioned before, although the robot has an arm, its power is so weak that we cannot rely on it to move the box. The robot carries the box to the goal only by these seven actions.

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 the balance shift. Thus, even though the robot starts from the same position under the same conditions, it does not necessarily follow the same path.

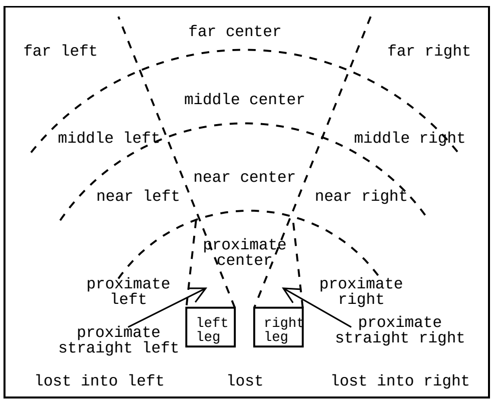
In addition, every action takes approximately ten seconds. It is, therefore, desirable that the learning time be as short as possible.

3.1.2 State Space

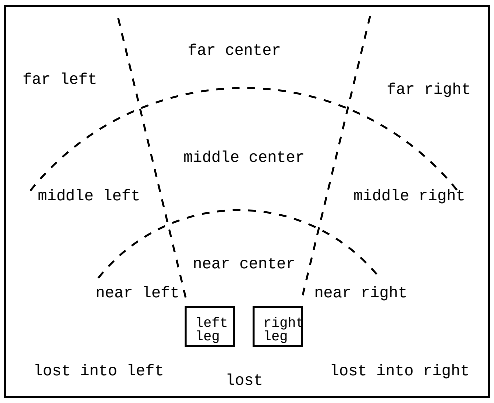
The state space is structured based on positions from where the box and the goal marker can be seen in the CCD image, as described in [8]. This recognition is performed after every action.

Figures 3(a) and 3(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 rough distances of objects. We used four levels of the distance for the box (i.e., “proximate”, “near”, “middle”, “far”) and three levels for the goal marker (“near”, “middle”, “far”).

We have to pay special attention to the position of the robot’s legs. Depending on the physical relationship between the box and legs of the robot, the box moves forward or deviates to a 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 have defined the “proximate straight left” and the “proximate straight right” states at the frontal positions of the front legs. These states do not exist in the simulation because the interaction between the legs of the robot and the box is very difficult to simulate.



(a) States of the box.



(b) States of the goal marker.

Figure 3: States in real robot. The front of the robot is upside of these figures.

The state is defined to be “lost” when the robot misses the box or the goal position. Remember that the CCD camera of our robot is fixed in a forward direction, not movable. As a result of this, the robot often misses the box or goal. To recover from this missing, we use the following strategy. The robot records the missing direction (i.e., left or right) when it has missed the box or goal. The robot can use this information for a certain number of steps so as to generate the “lost into left” or “lost into right” states, which means that the target had disappeared in either the left or right di-

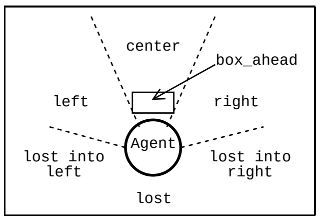


Figure 4: States for box and goal marker in the simulator. The area `box_ahead` is not the state but the place where `if_box_ahead` executes first argument.

rection in the last few steps.

As shown in Figs. 3(a) and 3(b), there are 17 states for the box and 14 states for the goal marker. The state of this environment is represented with their cross product. Hence, there are 238 states in total.

When the goal marker is "near center" and the box is near 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.

3.2 GP part conducted by the simulated robot

The simulator in our experiment uses a robot expressed as a circle on a two-dimensional plane as well as a box and a goal marker fixed on the plane. The task is completed when the robot has pushed the box forward to the goal marker [8].

We defined three actions ("move forward", "turn left", "turn right") in an action set. The state space in the simulator is simplified as shown in Fig. 4. While actions of the real robot are not ideal, the simulator actions are ideal, i.e., "move forward" action moves the robot truly straight-forward and "turn left" action purely turns left. Of all the states, the "lost into left" and "lost into right" states are similar to those used by a real robot. These are the states produced when the box or the goal is not in view and the preceding step is either at the left or at the right.

Such actions with a state division are similar to those for a real robot, but are not identical. In addition, physical parameters such as box weight and friction are not measured nor is the shape of the robot taken into account. Therefore, this simulator is so simple and it is possible to build it with low cost.

The operational characteristics of the box expressed in the simulator are as follows:

1. The box moves forward if it comes in contact with the front of the robot and the robot moves ahead[1].

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[2].

[1]This means the robot is able to push the box with the Forward action.

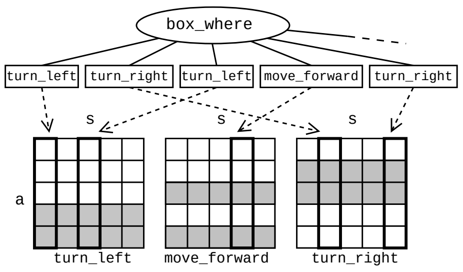


Figure 5: Action nodes pick up a real action according to the Q-value of a real robot's state.

We used a terminal set = { `move_forward`, `turn_left`, `turn_right` } and a function set = { `if_box_ahead`, `box_where`, `goal_where`, `prog2` }. The above terminal nodes correspond to the "move forward", "turn left", and "turn right" actions respectively 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 of the box and the goal marker as seen through the robot's eyes (Fig. 4). Further details of the setting of GP are described in [8].

3.3 Integration of GP and RL

Reinforcement learning is executed to adapt actions acquired via GP to the operational characteristics of a real robot. More precisely, this is aimed at revising the `move_forward`, `turn_left` and `turn_right` actions with the simulator so as to achieve their optimal actions in a real environment.

We applied $Q(\lambda)$-learning method to a real robot. $Q(\lambda)$-learning is a variant of Q-learning and can learn more efficiently 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 described in [17].

A Q-table, on which Q-values were listed, is allocated to each of the `move_forward`, `turn_left` and `turn_right` action nodes. The states used on the Q-tables are 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.

For the initialization of the Q-table, we allow a specific action to be chosen more frequently. For instance, in case of the Q-table for `move_forward`, the Forward action tends to be selected more often as are the Q-tables for `turn_left` and `turn_right`. However, when the box is

[2]This means that the robot is able to change the direction of the box only by Left-turn and Right-turn actions.

Table 2: Action nodes and their selectable real actions.

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 action which Q-table prefers to select.

[*2] Preferred actions if the box is in a state of "proximate center", "near center", "middle center", or "far center".

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 greater frequency. This maintains consistency with the simulator results. Because turn actions are always accompanied by a small backward motion, when the robot takes an action of the combination of turn and sidestep in one of these states, then the next state remains to be the same. In the implementation, 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 robot can learn optimal actions in this setting because Q-values converge regardless of the initial value [17]. The actions which are preferred to select on each action node are summarized in Table 2. In addition, 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.

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

As for $Q(\lambda)$-learning parameters, the reward was set to be 1.0 when the goal was achieved and 0.0 for other states. We chose the learning rate $\alpha = 0.3$, the discount factor $\gamma = 0.8$ and the trace-decay parameter $\lambda = 0.5$. These parameters are determined from preliminary experiments.

4 Experimental Results with a humanoid robot HOAP-1

An experiment was performed using a humanoid robot. In this experiment, the starting state was limited to an arrangement in which both the box and the goal position were visible. This was justified by the following reason. Even if learning is performed from an arrangement in which either the box or the goal is "lost", it cannot be predicted that the box or goal position will subsequently become visible. Thus, there will be substantial variations in state transition, and a long time will be required for the learning.

As a single trial, the learning was performed until the robot protruded from the field of the experiment or the predetermined number of steps (i.e., 30 steps) was exceeded. The learning in the real robot was performed in about six hours.

Just at the start of 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 characteristic differs from AIBO.

However, the robot took a long time in some situations. This proves that the acquired actions with the simulator are not always optimal in a real environment because of the differences between the simulator and the real robot. These differences necessitate the states added to the box 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 learning.

After six hours (after about 1800 steps) Noticeably improved actions were observed. The robot selected appropriate actions in the situations. Fig. 6 shows such a successful action sequence. It completed the task much faster than it did before learning.

The same improvement has been observed as described in our previous studies on AIBO [8]. 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, then it is applicable to both AIBO and HOAP-1.

5 Discussion

5.1 Measurement of Improvement

We performed the quantitative comparison so as to investigate how efficiently the robot performed after learning. For this comparison, we randomly selected six situations: four situations are selected from states added for the real robots ("box proximate straight left" and "box proximate straight right"), and the other two are selected from different states. We measured the number of steps in completing the task both before and after learning in these situations. These

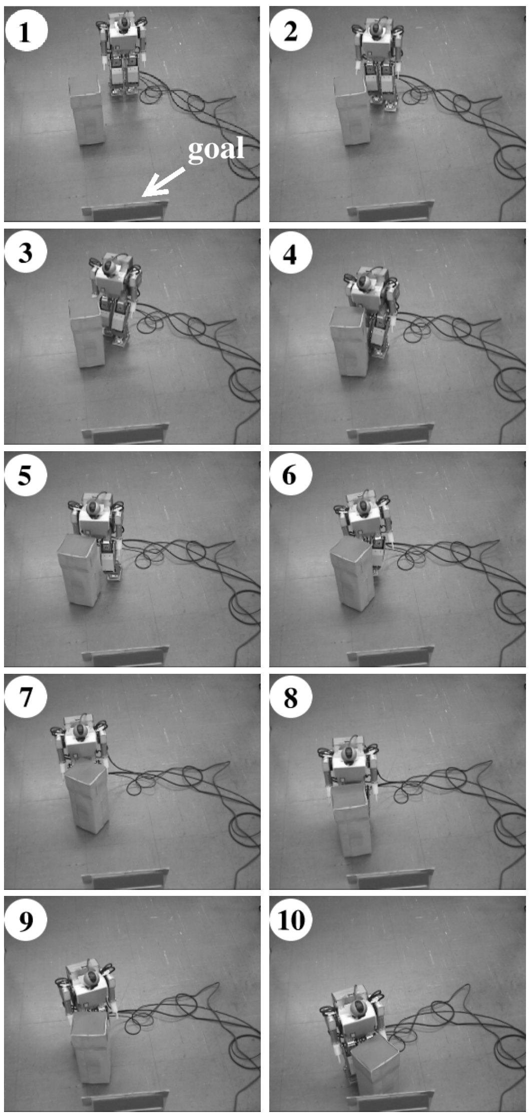


Figure 6: Successful action series. The goal is at the bottom center of each figure.

tests are executed in a greedy policy in order to insure the robot always selects the best action in each state.

Table 3 shows the results. After the learning, the robot completed the task more efficiently in four out of six situations (represented by bold font in the table) than before. In particular, great improvement was observed in situation #2. These results prove our technique worked very well and then the robots learned efficient actions.

There is another point to consider in terms of efficiency. We had to deal with the "state-action deviation" problem [2] when applying $Q(\lambda)$-learning to this experiment. This is the problem that optimal actions cannot be achieved due to the dispersion of state transitions. More precisely, if the state is composed only of the images, there are often no remarkable

Table 3: Comparison of the number of steps between before and after learning.

#. situation	state notation	before learning	after learning
1	Box: proximate straight right Goal Marker: middle right	**7.3**	8.5
2	Box: proximate straight right Goal Marker: middle left	10.7	**5.6**
3	Box: proximate straight left Goal Marker: middle right	16.3	**12.8**
4	Box: proximate straight left Goal Marker: far right	**14.0**	15.3
5	Box: near left Goal Marker: far center	14.7	**12.3**
6	Box: middle right Goal Marker: far center	11.3	**10.9**

differences in image values within an action. As a solution to this problem, the same action should be repeated until the current state changes. The Q-values are updated when the state changes (for details, see [8]).

In $Q(\lambda)$-learning as well as in the usual reinforcement learning, the agent learns optimal actions in order to maximize the sum of the discounted rewards which it receives until completing the task [17]. The sum, which is called as expected discounted return, can be written as follows:

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

where t is a current time step, T is the last time step, 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 $r = 0.0$. In the situation, the equation can be written simply as:

$$R_t = \gamma^T. \tag{2}$$

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

Table 4 shows the counts of state transitions in the situations. This table illustrates that the performance in situation #4 was improved after learning in terms of the counts of state transitions. This is evidence that the real-time learning process of our technique is very effective.

In situation #1, the 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.

Table 4: Comparison of the counts of state transitions.

#. situation	before learning	after learning
1	**7.0**	8.0
2	10.3	**4.0**
3	16.0	**12.5**
4	14.0	**13.5**
5	14.7	**8.3**
6	10.3	**9.9**

The number of steps did not necessarily become smaller after learning in situation #4, whereas the number of state transitions became 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 case of the incorrect state space division.

5.2 Related Works

There are many studies combining evolutionary algorithms and RL [13, 3]. Although the approaches differ from our proposed technique, we have see several studies in which GP and RL were combined [7, 4]. With these traditional techniques, Q-learning was adopted as a RL, and an individual of GP represented the structure of the state space to be searched. It is reported that its searching efficiency was improved in QGP method [7], compared to 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. A 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 with a real robot.

Noise in simulators are often essential to overcome the differences between a simulator and real environment [16]. The robot which learned with our technique, however, showed sufficient performance in the noisy real environment, even though the robot had learned 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 of the robustness produced by our technique with that by noisy simulators.

As described in Sect. 5.1, it seems that the division of the state space is not appropriate in some situations. It is difficult for the robot to improve the actions in such situations because the division is fixed in the learning process. Takahasi et al. [18] proposed two methods of segmenting a state space automatically. In the first method, the real robot moves in its environment and samples data. After that, it segments a state space 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 about ten seconds in one action. In this condition, uniform sampling is not reasonable because it takes an enormous amount of time. Although the second method segments the state space incrementally on-line, it also seems to require sampling many data to construct a sufficient state space. It may be time-consuming for a real robot, but it is still an interesting approach.

We used several pre-defined actions for the humanoid. This is a shortcut to investigate the applicability of evolutionary methods to such high-level functions as solving a task. In contrast, there are related studies to evolve low-level functions of a humanoid. Nordin et al. have developed a humanoid robot “ELVIS” [15]. The software of this robot is built mainly on GP. They experimented in the evolution of stereoscopic vision [6] and sound localization [9]. They also reported the result of a hand-eye coordination [12].

5.3 Future Researches

We chose only several discrete actions in this study. Although this is simple and easy to use, continuous actions will be more realistic in other applications. In that situation, for example, “turn left in 30.0 degrees” at the beginning of RL can be changed to “turn left in 31.5 degrees” 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 the multi-agent problem. It might be possible to handle a multi-agent task with heterogeneous robots by extending our approach. For this purpose, we use a simulation-based learning to acquire the common programs applicable to various types of robots. After that, real robots are supposed to learn their effective actions in spite of their different operational characteristics. This may be possible while they are carrying out a cooperative task in a real-world situation.

Another extension is to adapt the simulation parameters or modify the state space by using the information available from a real environment. The simulator tuning will require the feedback process as described in dotted lines in Fig. 2. This will enable GP to evolve more effective programs. Note that programs evolved by GP cannot be run without any state space. However, we can give a rough state space initially and then modify it gradually according to the robot’s characteristics, which will establish more effective learning scheme.

6 Conclusion

We have introduced a real-time adaptation technique to real robots. This approach is based on our previously proposed method with AIBO. We applied the same evolved programs to a humanoid robot. We confirmed that, after 6-hour learning, the effective adaptation was established according to the robots' operational characteristics so as to solve the task effectively. Our approach was successful in acquiring the common program, which will be applicable to heterogeneous robots.

Bibliography

[1] Björn Andersson, Per Svesson, Mats Nordahl, and Peter Nordin. On-line evolution of control for a four-legged robot using genetic programming. In Stefano Cagnoni et al., editors, *Real-World Applications of Evolutionary Computing*, LNCS 1803, pages 319–326. Springer-Verlag, 2000.

[2] Minoru Asada, Shoichi Noda, Sukoya Tawaratsumida, and Koh Hosoda. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, 23:279–303, 1996.

[3] Marco Dorigo and Marco Colombetti. *Robot Shaping: An Experiment in Behavior Engineering*. MIT Press, 1998.

[4] Keith L. Downing. Adaptive genetic programs via reinforcement learning. In *Proc. of the Third Annual Genetic Programming Conference*, 1998.

[5] D. Floreano and F. Mondada. Evolution of homing navigation in a real mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, 26(3):396–407, 1996.

[6] Christopher T.M. Graae, Peter Nordin, and Mats Nordahl. Stereoscopic vision for a humanoid robot using genetic programming. In Stefano Cagnoni et al., editors, *Real-World Applications of Evolutionary Computing*, LNCS 1803, pages 12–21. Springer-Verlag, 2000.

[7] Hitoshi Iba. Multi-agent reinforcement learning with genetic programming. In *Proc. of the Third Annual Genetic Programming Conference*, 1998.

[8] Shotaro Kamio, Hideyuki Mitsuhasi, and Hitoshi Iba. Integration of genetic programming and reinforcement learning for real robots. In *Proc. of the Genetic and Evolutionary Computation Conference (GECCO2003)*, 2003.

[9] Rikard Karlsson, Peter Nordin, and Mats Nordahl. Sound localization for a humanoid robot by means of genetic programming. In Stefano Cagnoni et al., editors, *Real-World Applications of Evolutionary Computing*, LNCS 1803, pages 65–76. Springer-Verlag, 2000.

[10] 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.

[11] John R. Koza. *Genetic Programming, On the Programming of Computers by means of Natural Selection*. MIT Press, 1992.

[12] William B. Langdon and Peter Nordin. Evolving hand-eye coordination for a humanoid robot with machine code genetic programming. In J. Miller et al., editors, *Proc. of 4th European Conference, EuroGP 2001*, LNCS 2038, pages 313–324, Lake Como, Italy, April 18-20 2001. Springer-Verlag.

[13] David E. Moriarty, Alan C. Schultz, and John J. Grefenstette. Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, 11:199–229, 1999.

[14] Stefano Nolfi and Dario Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. The MIT Press, 2000.

[15] J. P. Nordin and M. Nordahl. An evolutionary architecture for a humanoid robot. In *The Fourth International Symposium on Artificial Life and Robotics (AROB 4th 99)*, Oita Japan, 1999.

[16] Alan C. Schultz, Connie Loggia Ramsey, and John J. Grefenstette. Simulation-assisted learning by competition: Effects of noise differences between training model and target environment. In *Proc. of Seventh International Conference on Machine Learning*, pages 211–215, San Mateo, 1990. Morgan Kaufmann.

[17] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An introduction.* MIT Press in Cambridge, MA, 1998.

[18] 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.

[19] Kohsuke Yanai and Hitoshi Iba. Multi-agent robot learning by means of genetic programming: Solving an escape problem. In Y. Liu and et al., editors, *Evolvable Systems: From Biology to Hardware. Proceedings of the 4th International Conference on Evolvable Systems, ICES'2001, Tokyo, October 3-5, 2001*, pages 192–203. Springer-Verlag, Berlin, Heidelberg, 2001.