# Evolutionary Construction of a Simulator for Real Robots

Shotaro Kamio Hitoshi Iba
Graduate School of Frontier Science, The University of Tokyo,
5-1-5 Kashiwa-no-ha, Kashiwa-shi, Chiba, 277-8561, Japan.
{kamio,iba}@iba.k.u-tokyo.ac.jp

*Abstract*— **In order to acquire useful motions of a real-world robot, it is necessary to carry out learning in a real environment. However, learning is difficult within a real environment. In addition, the acceleration of learning is required for a practical execution.**

**In this paper, we propose an approach to the learning acceleration using data retrieved from the real environment. This consists of the method of automatically constructing the simulator from real data and of learning a robot controller with the simulator. The experimental results suggest that our GP-based technique enables the effective controller learning.**

## I. Introduction

In order to automatically acquire the motion of a robot that moves in a real environment, it is important to select the effective learning method for the robot. There are three conceivable learning methods [10]: learning by simulation, learning from the real environment, and learning by using a combination method of these two. All of these methods have their own trade-offs. There is a difference between simulation and a real environment, and it is not possible to guarantee that the results obtained by simulation will work in a real environment. However it is troublesome to move a robot in a real environment, and it takes too much time to complete the learning only in a real environment. To overcome this, it is necessary to accelerate the learning process by using simulation.

Several methods that utilize not only simulation but also motion in a real environment have been proposed. Nordin et al. carried out research into a method of controlling a robot by acquiring a world model using GP [8]. The learning algorithm consists of memorizing, learning and planning. First, when the robot moves, the result of this motion is memorized as one event. The state value of the environment is assigned to this event by using a pre-determined evaluation function. In the learning process, a memorized event is used to generate a function that predicts the state value of an inexperienced state using GP. This prediction function is considered as the world model (i.e., environmental model).

Grefenstette et al. have proposed anytime learning [2]. They used this method for agent learning in a pursuit problem. The learning system contains an environmental simulation model. The agent's strategy is created using this simulation model. In addition, the simulation model is refined when a difference between the behavior of the target in the real environment and the behavior in the simulation model is detected. In this way, it is possible to bring the simulation model closer to the real environment using the behaviors in the environment. However, all of this research was done by computer simulation. The input data that can be acquired by a robot in a real environment consist of image data, and so on, hence processing to obtain the desired data is generally complicated. Consequently, it is difficult even to detect the difference between the behavior of the target in the environment and the behavior in the simulation model. This problem arises because the simulation model is constructed from the human viewpoint.

Parker et al. proposed punctuated anytime learning [9], [10]. In this method, learning was done by an evaluation function in the computer. However, at every several generations, some of individuals were evaluated in the real environment, and the results thereof were incorporated in the evaluation function in the computer. Consequently, it was possible to acquire behavior that adapted to changes in the environment while reducing the evaluation time in the real environment. Although this method is very interesting, the model for the evaluation and environment used has been greatly simplified. Thus, it may be difficult to apply the model to other robots without change.

We have proposed an real-time learning method for a robot and showed experimental results in the prior researches [5], [4]. This method acquires the fundamental behavior using GP in a simple simulation, performs Q-learning using a real robot, and adapts the behavior to the real environment. In [4], we used a humanoid robot for the experiment. This robot took about 10 seconds to perform one motion. For this reason, it is not reasonable to carry out learning until Q-learning has converged completely.

We can collect real data concerning the environment because the robot behaves in the real environment. Therefore, we should use these data. A simulation model should be constructed using these data. This approach reduces the difference between the real environment and the simulation model.

In this paper, methods of automatically constructing a simulator are studied. First, a method of constructing a simulator based on data obtained from the real environment is investigated. Next, the resulting simulator is used to carry out training of the robot controller.

The organization of this paper is as shown below. The next section describes the construction of the simulator used in this research. Section III describes the setting of the target task performed in this research. Sections IV and V express the tests and their results. Section VI describes future issues, and section VII gives some conclusions.

## II. Construction and Utilization of a Simulator

By moving the robot in a real environment, it is possible to acquire data related to that environment. Provided that the environment has the Markov property [11], the data are expressed using the state of the environment ($\boldsymbol{s_t}$), the action of the robot at that time ($a_t$), and the resulting state of this action ($\boldsymbol{s_{t+1}}$), as follows:

$$(\boldsymbol{s_t}, a_t) \mapsto \boldsymbol{s_{t+1}} \tag{1}$$

The environmental model (i.e., simulator) is a function for predicting the effect of an action in an inexperienced state. In other words, it is equivalent to the following function $f$ in a mathematical form:

$$\boldsymbol{s_{t+1}} = f(\boldsymbol{s_t}, a_t). \tag{2}$$

We can estimate this function from the generalization of the currently possessed data. In this paper, we propose to use a function approximation method (described in section III) for acquiring of this function (simulator). If a simulator can be obtained, we can use any learning algorithm whatever on it.

We can regard Q-learning as an approximation method of the function $f$ based on a sampling by an agent. However, the model in Q-learning (i.e., the Q-value) is closely connected to the reward. Although Q-learning cannot reuse the model for other tasks, our approach can reuse the simulator only with a change of the subject function. Moreover, the Markov property of the environment is not necessarily assumed for our approach (it depends on the setting of the simulator).

In addition, our method has the following advantage. Each time the robot moves, data are obtained, making it possible to train the simulator with an increasing amount of those data. This is what we call an incremental learning. Data may well contain real noise due to the real environment. The overfitting, therefore, is avoided by utilizing these characteristics of the data, as can be seen in the later section.

## III. Task setting

The target task is the “box moving” task for the “HOAP-1” humanoid robot (Fig. 1), which was performed in our prior research [4]. The goal of this task is to have the robot move a box to the front of the designated mark (goal marker). The robot uses a CCD camera installed on its head to recognize the box and the goal marker. The strength of the robot’s arms is weak. Thus, the robot does not move the box with its arms, but instead pushes it with its knees. Because the robot walks on two legs, complex motion is sometimes involved in moving the box. It is difficult to create a simulator that can accurately express this situation.

In this research, we dealt with real data acquired when we performed a learning of the task using the robot in the real environment for six hours. These data consist of input image data ($\boldsymbol{i}$), the action selected at that time ($a$), and next input data after the action ($\boldsymbol{i'}$), recorded as one set. The action is selected from seven actions: i.e., 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. The recorded input image data ($\boldsymbol{i}$) consist of the position of the geometrical center of the box in the image, the position of the geometrical center of the goal marker, and the horizontal width of the goal marker (used to obtain the approximate distance to the goal marker). Fig. 2 shows a sketch of the input image data. In this research, for the sake of simplicity, we do not use the data acquired when the robot lost sight of the box or the goal marker. These data are obtained by performing measurement in the real environment. Note that it contains noise.

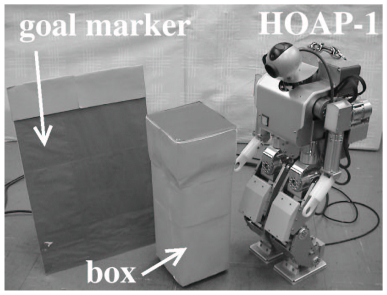


Fig. 1. The humanoid robot “HOAP-1”, the box and the goal marker.

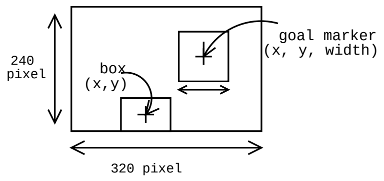


Fig. 2. Input image data.

By using the function approximation method, the input data after the robot moves ($\boldsymbol{i'}$) are predicted based on the current data $\boldsymbol{i}$ and $a$. The data prediction error is evaluated as the squared error $E = |\boldsymbol{i'}_{\text{real}} - \boldsymbol{i'}_{\text{predict}}|^2$. While learning takes place based on this prediction error, the generality of the resulting function is important. For this reason, it is necessary not only to carry out learning using training data, but also to evaluate the resulting function with the test data [6]. The number of training data sets provided was 1,210. For the test, 100 sets different from the training data were used.

In this research, two function approximation methods for acquiring a simulator were used for the comparison: i.e., Genetic Programming (GP) and a clustering approximation method.

### A. GP

To acquire the simulator, we have to estimate the function $f$ in (2). This is a so-called regression problem.

Table I shows functions and terminals of the GP we used. This is a strongly-typed GP [7]: i.e., each function and terminal has a specified type. In addition, each function’s argument is constrained to have a specific type. The function `if-`

TABLE I

GP FUNCTIONS AND TERMINALS IN THE EXPERIMENT I IN BNF NOTATION.

| An individual of the GP ::= PREDICTION |
|---|
| PREDICTION ::=<br>'(' `if-action-eq-then-else` ACTION PREDICTION PREDICTION ')' \|<br>'(' `box-moves` COORD COORD ')' \|<br>'(' `goal-moves` COORD COORD COORD ')' \|<br>'(' `prog2` PREDICTION PREDICTION ')' \| `nop` |
| ACTION ::=<br>'(' `or` ACTION ACTION ')' \| `action-any` \|<br>`action-fwd` \| `action-turn-l` \| `action-turn-r` \|<br>`action-step-l` \| `action-step-r` \|<br>`action-turn-step-l` \| `action-turn-step-r` |
| COORD ::=<br>'(' `if-lt-then-else` COORD COORD COORD COORD ')' \|<br>'(' `if-gt-then-else` COORD COORD COORD COORD ')' \|<br>'(' + COORD COORD ')' \| '(' - COORD COORD ')' \|<br>'(' * COORD COORD ')' \| '(' / COORD COORD ')' \|<br>`box-x` \| `box-y` \| `goal-x` \| `goal-y` \| `goal-width` \|<br>`0.5` \| `1.0` \| `2.0` \| `3.0` \| `4.0` \| `5.0` \| `6.0` \| `7.0` \| `8.0` \| `9.0` |

`action-eq-then-else` executes its second argument if the selected action at the time matches with its first argument (this can be a logical expression using `or`). Otherwise, it executes its third argument. The function `if-lt-then-else` (`if-gt-then-else`) executes its third argument if the value of its first argument is smaller (larger) than that of its second argument. Otherwise, it executes its fourth argument. Functions `box-moves` and `goal-moves` output their prediction values. Two arguments of `box-moves` express changes in the position (x, y) of the box after one action. Three arguments of `goal-moves` express changes in the position (x, y) and the horizontal width of the goal marker. Terminals in COORD (`box-x`, `box-y`, `goal-x`, `goal-y`, `goal-width`) are values of the coordinate (x, y) of the box, the coordinate (x, y) of the goal marker and the horizontal width of the goal marker at the time, respectively.

### B. Clustering Approximation Method

We also use another method which automatically constructs the state space of Q-learning ("the second method" described in [12]). This method divides data acquired from the real environment into several clusters and constructs the approximation model for each cluster. We call this method as the clustering approximation. In our study, however, we exclude reward information from the approximation model. This is because our purpose is to construct the simulator according to the environment.

*1) Model Construction [12]:* Data $d$ acquired by the robot from the environment are defined as follows:

$$d = < a, \boldsymbol{i}, \dot{\boldsymbol{i}'} > . \tag{3}$$

We can define a local approximation model when these data are given. This local approximation model is a linear approximation of the data as follows:

$$\dot{\boldsymbol{i}'} = \boldsymbol{A}\boldsymbol{i} + \boldsymbol{b}. \tag{4}$$

The detailed algorithm of constructing the local model is described below, where $a_i$ is $i$th action in an action set.

1) Let $C$ be a set of all $d$ which contains the action $a_i$.
2) Apply the weighted linear regression method so as to fit the local model (4) to the above set $C$.
3) If the unbiased variance of its residual exceeds a certain threshold, then
   a) Divide $C$ into two clusters ($C_1$, $C_2$) using the clustering method with the weighted Euclidean norm as its similarity.
   b) Go back to step 2) with $C := C_1$.
   c) Go back to step 2) with $C := C_2$.

This method divides the data into several clusters. Each cluster has the coefficient $\boldsymbol{A}$ and the constant $\boldsymbol{b}$ which represent the local model (4).

*2) Prediction by the Local Approximation Model:* We can predict the effect of an action in an unknown state (i.e., unobserved input data) using the above local model as the simulator. For the prediction, the nearest cluster to the current input $\boldsymbol{i}$ is searched for. The distance between an input and a cluster is defined by the distance between the input and the most similar data to the input which belongs to that cluster.

## IV. EXPERIMNT I: SIMULATOR CREATION BY INCREMENTAL LEARNING

We performed experiments to create a simulator using the abovementioned two methods. In a real robot, the timing at which data are acquired differs from one system to another. In one robot, it may be one set of data per an action, while in another acquiring many data may be possible even within an action. In addition, the acquired data can be utilized immediately, or it may be possible to wait until a certain amount of data has been accumulated. In other words, the designer can decide the timing at which to increase the training data.

With GP, tests were started from ten sets of training data, and were then performed in three ways: (1) the case where the training data were increased by four sets up to 350 generations, (2) the case where they were increased by five sets up to 250 generations, and (3) the case where they were increased by 10 sets up to 150 generations, every generation. However, the number of the training data were not increased past 1,210. The test data always contained 100 sets of data. The clustering approximation method is not an incremental learning. Thus, it was experimented with fixed training data sets.

### A. Results of GP

In order to perform the incremental learning using GP, it is considered necessary to maintain a certain diversity. Accordingly, in each generation, randomly generated individuals were introduced into the population (indicated as "new" parameter in the later graph). The parameters used are shown in Table II.

The results are shown in Fig. 3. The horizontal axis is the generation, and the vertical axis is the average prediction error in the test data indicated by the best individuals in each generation. The mark "inc" in the figure is the number of sets of training data that increases in one generation. "cr" means the crossover rate and "mu" means the mutation rate. The

TABLE II

PARAMETERS OF GP IN THE EXPERIMENT I

| Population size | 1000 |
|---|---|
| Generations | 150, 250, 350 |
| Crossover rate | 0.7, 0.8 |
| Mutation rate | 0.10, 0.05 |
| Rate for introducing random individuals | 0.1 |
| Trials | 10 |

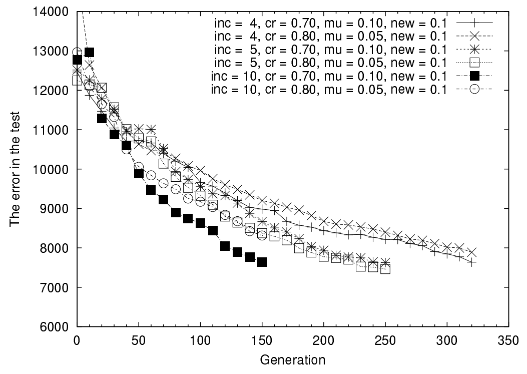


Fig. 3. The average prediction error by GP in the experiment I.

TABLE III

GP FUNCTIONS AND TERMINALS FOR THE LEARNING OF THE CONTROLLER IN EXPERIMENT II.

| |
|---|
| an individual of GP ::= ACTION |
| ACTION ::= ‘(’ `if-lt-then-else` COORD COORD ACTION ACTION ‘)’ \| ‘(’ `if-gt-then-else` COORD COORD ACTION ACTION ‘)’ \| `action-fwd` \| `action-turn-l` \| `action-turn-r` \| `action-step-l` \| `action-step-r` \| `action-turn-step-l` \| `action-turn-step-r` |
| COORD ::= ‘(’ `+` COORD COORD ‘)’ \| ‘(’ `-` COORD COORD ‘)’ \| ‘(’ `*` COORD COORD ‘)’ \| ‘(’ `/` COORD COORD ‘)’ \| `box-x` \| `box-y` \| `goal-x` \| `goal-y` \| `goal-width` \| `0.5` \| `1.0` \| `2.0` \| `3.0` \| `4.0` \| `5.0` \| `6.0` \| `7.0` \| `8.0` \| `9.0` |

TABLE IV

PARAMETERS OF GP FOR THE LEARNING OF THE CONTROLLER IN THE EXPERIMENT II.

| Population size | 1000 |
|---|---|
| Generations | 50, 100 |
| Crossover rate | 0.7 |
| Mutation rate | 0.1 |
| Rate for introducing random individuals | 0.1 |
| Trials | 10 |

prediction error continues to decrease with generations, and overfitting has not occurred.

The figure shows each final value of prediction error is similar to others. This means the final performance depends upon parameters of the genetic operator, but does not depend upon the way in which the number of the training data increases.

## B. Results of Clustering Approximation Method

We used k-mean clustering as the clustering method in this study. k-mean clustering needs the upper bound of the variance in a cluster for the termination condition. This upper bound affects the performance (i.e., the granularity) of clustering. We performed this experiment with the following upper bounds: $1{,}000, 2{,}000, \ldots,$ and $10{,}000$. The numbers of training data used were 300, 600, 900 and 1,210.

Figure 4 shows the result. The horizontal axis is the upper bound of the variance. With respect to the number of training data, the error is the largest with 300 sets of the training data

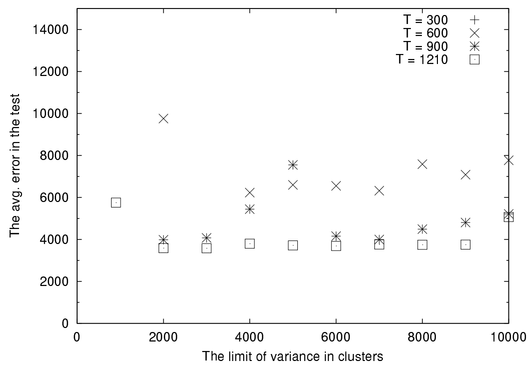


Fig. 4. The average prediction error by the clustering approximation method in the experiment I.

(all of them were out of the range). However, the more training data were used, the better results were achieved. The average prediction error became the smallest with 1,210 data sets. This proves that the clustering approximation method is effective if sufficient data are available for the training.

## C. Summary

It was confirmed that a simulator could be acquired for the incremental learning in which the training data were gradually increased. With a large amount of training data set, the performance of the simulator by the clustering approximation method is better than that by GP.

The overfitting was not found by GP. This may be because new training data are continuously added, so that the selective pressure acts for the sake of generalizing the results of learning.

The above results in GP would seem to indicate that the performance does not depend upon the rate at which the number of the training data is increased. This means that we do not have to worry about setting the precise rate at which the number of the training data is increased.

## V. EXPERIMENT II: TRAINING A CONTROLLER USING THE ACQUIRED SIMULATOR

A test was carried out to see whether or not a useful controller for a robot could be trained using an automatically generated simulator. In the same way as the tests carried out in the previous section, the overfitting does not occur when the incremental learning is carried out, and it is expected that the prediction performance of the simulator will be improved with learning. Accordingly, the following two methods are possible:

1) Experiment II-1: Method in which the controller is trained using the best simulator obtained at a certain generation (i.e., the simulator is fixed).

2) Experiment II-2: Method in which the controller is trained while an improved simulator is used (i.e., the best simulator for that generation is used).

The former method is the same as the normal learning using a fixed fitness function. In the latter method, the box and goal marker motion characteristics may possibly change every time the simulator is improved.

As the simulator, those obtained by GP and the clustering approximation method were used. The training data used to train the controller were originally the same as those used when training the simulator. However, in these experiments noise was added so that the data were different from the original ones.

In these experiments, we used GP for learning the controller. The used function and terminal nodes are shown in Table III. The parameters used are given in Table IV. A trial ends when the robot either moves 30 steps or the coordinate of the box or the goal marker gets out of the visible zone shown in Fig. 2. The fitness function is defined as follows:

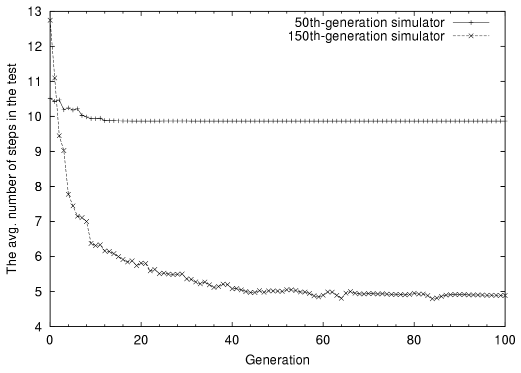
$$fitness = \frac{1}{N}\sum_{i}^{N} \text{steps}_i, \tag{5}$$

where $\text{steps}_i$ means the number of steps to complete the $i$th trial and $N$ is the number of sets of training data. If the task are not be accomplished, we assign $30 + (30 - \text{valid\_steps}_i)$ to $\text{steps}_i$ as a penalty. In this definition, $\text{valid\_steps}_i$ is the number of steps to the point where the robot loses sight of the box or the goal marker. The larger the number of steps until the coordinate of the box or the goal marker gets out of sight, the better is the fitness value. Equation (5) is also used to evaluate the performance based on the test data with letting N be the number of test data. The motion characteristics of the box will change when the simulator is improved. For this reason, a different fitness value will be assigned to an individual every generation, even if the same training (test) data are used to evaluate the same individual.

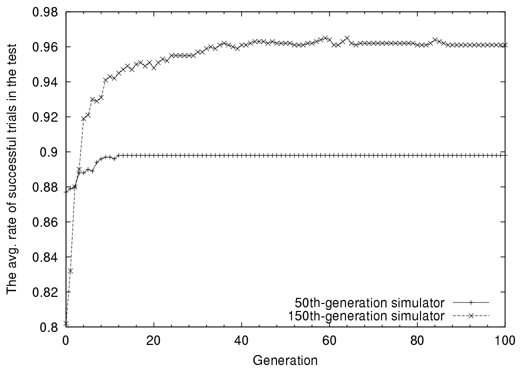
## A. Experiment II-1: Training the Controller with a Fixed Simulator

We performed the experiment to acquire the controller with a fixed simulator. For the simulator evolved with GP, the result of a typical single trial (Sect. IV-A) was used. We chose two simulators from a typical run: i.e., the 50th-generation simulator and the 150th-generation simulator. For the simulator by clustering approximation method, we used two results acquired with 510 and 1,210 training data. The training data for the learning of the controller were produced by adding some noise to the data available from the simulator production.

The results are shown in Figs. 5-6. The vertical axis indicates the average steps to achieve the task or the average rate of successful trials. As can be seen from Fig. 5, the success ratio of the learning is relatively small by the 50th-generation simulator. This may be because the performance of the simulator is not sufficient for the learning. In contrast, regarding what was learned by the 150th-generation simulator,



(a) The average steps to achieve the task.



(b) The average rate of successful trials.

Fig. 5. The result of learning a controller by GP with the acquired simulator. The simulator was evolved with GP in the experiment I. The performance was shown for the test data.

TABLE V
PARAMETERS USED FOR Q-LEARNING IN THE EXPERIMENT II-1.

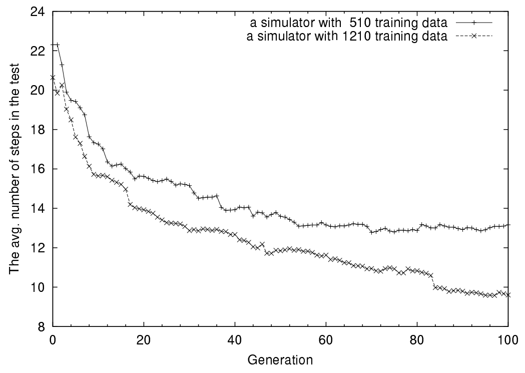
| | |
|---|---|
| Learning rate | $0.05/(1.0 + \text{(total accumulated steps)}/10^6)$ |
| Discount rate | 0.8 |
| Reward (successful trial) | 1.0 |
| Reward (failed trial) | -0.01, -0.1 |

the number of steps was reduced to five, and its success rate exceeded 96%. Therefore, it proved that an effective controller has been acquired with this simulator.

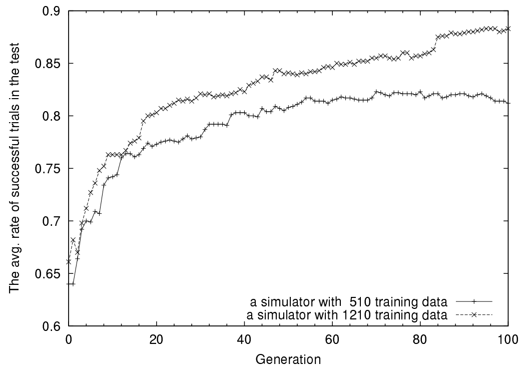
Results with the simulator by the clustering approximation method shows some improvement of the performance with generations (Fig. 6). However, final results were not as good as that with the simulator by GP in terms of both the number of steps and the rate of the success.

For the sake of comparison, we have also experimented in learning the controller with Q-learning. The state space of this Q-learning is the same as that used in our prior study [4]. Table V shows parameters we chose in this experiment. We cannot evaluate the controller using the same fitness function (eq. (5)) with GP. This is because Q-learning evaluates an agent by the reward. Thus, we used the reward value of $1.0$ for a successful trial and the reward (penalty) of $-0.01$ (or $-0.1$) for a failure.

Figs. 7 and 8 show the results of Q-learning with the simu-

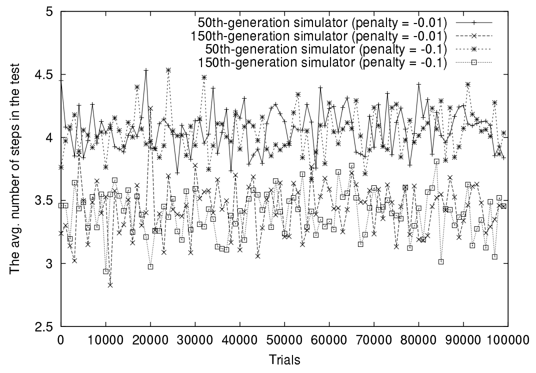


(a) The average steps to achieve the task.

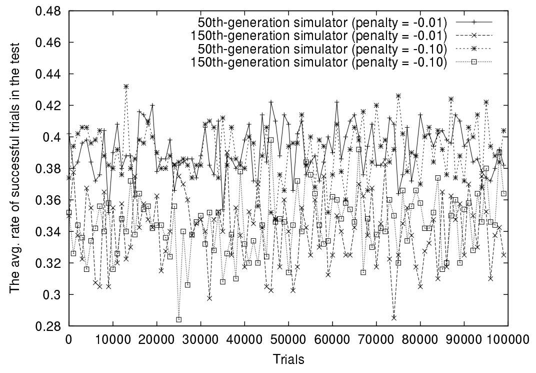


(b) The average rate of successful trials.

Fig. 6. The result of learning a controller by GP with the acquired simulator. The simulator was produced by the clustering approximation method in the experiment I. The performance was shown for the test data.



(a) The average steps to achieve the task.



(b) The average rate of successful trials.

Fig. 7. The learning result of the controller by Q-learning with the simulator. The simulator was evolved by GP in the experiment I. The performance was shown for the test data.

lator acquired by GP and by clustering approximation method, respectively. Both results present relatively low average steps and low success rate. Moreover, the higher rate of the success was, the larger average steps were required. This indicates that the acquired controller could only achieve the easy tasks which required low steps to complete. Therefore, we can conclude that the performance of the controller by GP is better than that by Q-learning.

In the above experiment, the Q-learning acquired appropriate actions only within easy situations. This is because Q-learning needs the effective scheduling of the learning (e.g. "Learning from Easy Mission" [1]). This scheduling may be applicable when we use 2-D simulators which are manually made for the learning. Then, we will be able to find easy situations where both the robot and the box are near to the goal marker. However, without a priori knowledge of the environment, we cannot always distinguish the situations based on input image data of objects. Therefore, the scheduling is not possible in case of this type of learning.

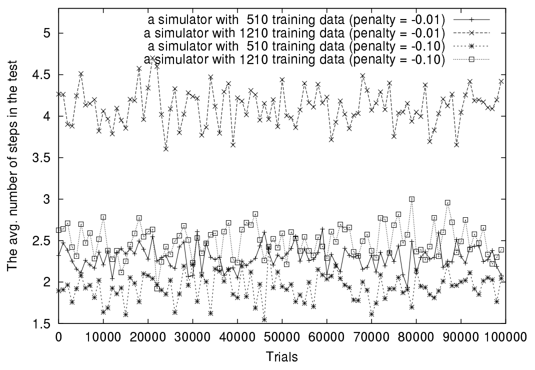
## *B. Experiment II-2: Training a Controller with the Incrementally Improved Simulator*

Next, we performed experiments in acquiring a controller with the improved simulator. The simulator by GP is based on the resulting simulator of a typical single trial (Sect. IV-A). The simulator by the clustering approximation, which is not an incremental learning method, was reproduced every time the number of the training data was increased.

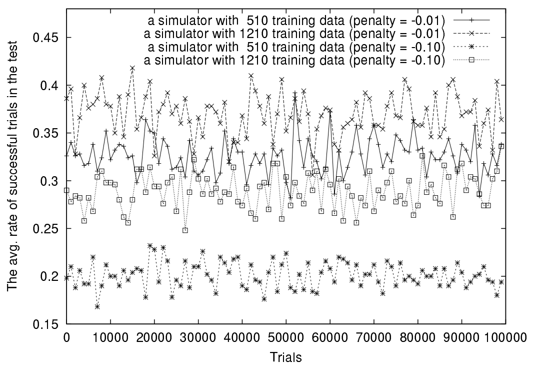
The controller learning by GP began from each simulation of the 0th, 50th and 100th generations. The training data were the same with those by the simulator acquisition; i.e., ten data sets were available in the initial generation and the number of the data set was increased by ten every generation.

Figure 9 shows the results with the simulator by GP. The average steps were reduced to about five and the success rates were at least 0.9 in all cases at the final generation. This indicates that the effective controllers were evolved successfully. A large peak was observed around the 130th generation.

Figure 10 shows the performance of the simulator we used. A large change in the performance (about 500) was observed around the 130th generation. This change affects the fitness values of individuals. It seems that the large peak in Fig. 9 was caused by the performance change of the simulator. Although another large change occurred around the 80th generation, we cannot observe the corresponding change in Figure 9. Furthermore, if we use different simulators, the results of the controller learning will be varied. We were sometimes

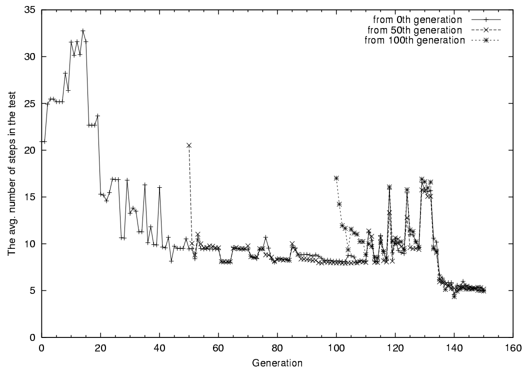


(a) The average steps to achieve the task.

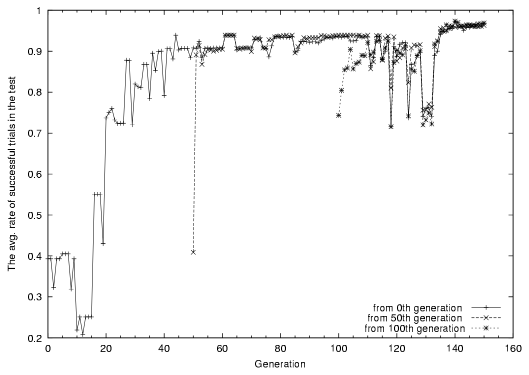


(b) The average rate of successful trials.

Fig. 8. The result of learning the controller by Q-learning with the simulator. The simulator was acquired by clustering approximation method in the experiment I. The performance was shown for the test data.



(a) The average steps to achieve the task.



(b) The average rate of successful trials.

Fig. 9. The result with the simulator by GP in the experiment II-2. The performance was shown for the test data.

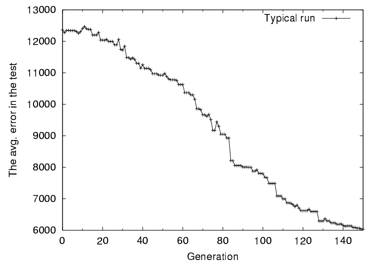


Fig. 10. The performance of the simulator by GP used in the experiment II-2.

not able to acquire the effective controller. Therefore, further investigations are needed about the relation of the performance between the simulator and the controller learning.

Figure 11 shows the results with the simulator by the clustering approximation method. Many increases and decreases were observed in the results. They may be caused by the performance of the simulator, which was reproduced due to increased training data sets every generation. Hence, its performance is supposed to change every reproduction. These changes will affect the controller learning with the simulator.

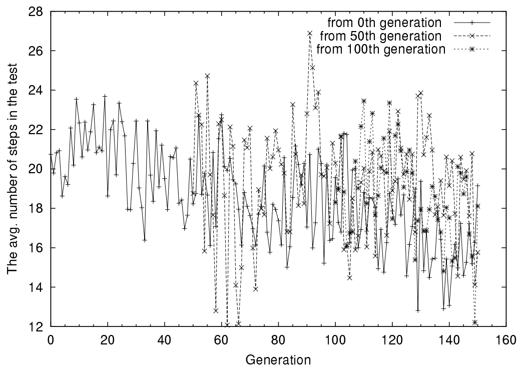
## C. Summary

It has been confirmed that training a controller is possible by the automatically acquired simulator. Experiments were carried out to show the validness using the two possible methods.

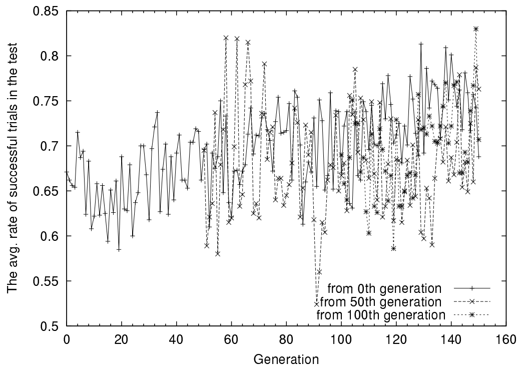
In the experiment II-1, we made experiments in the controller learning with GP and Q-learning. The performance of the controller with GP was superior to that with Q-learning. The effective controller, which can complete the task with a small amount of steps and with high success rate, was obtained with a large amount of the training data by GP.

Based on the above observation, we can conclude that a good choice would be to evolve the controller by GP with a large amount of training data set if we use the fixed simulator for the controller learning. In addition, the simulator by GP seems more suitable for the learning than the one by the clustering approximation method.

In the experiment II-2, it was found possible to train a controller while improving the performance of simulation. We compared the simulator by GP and the one by the clustering approximation method. With the simulator evolved by GP, it was possible to obtain a useful controller with a success rate of more than 90% at the 50th generation, even from the initial generation of the simulator. The number of data sets at the 50th generation is the same as the one acquired during the humanoid robot's 1-hour learning (about 700 actions) in our



(a) The average steps to achieve the task.



(b) The average rate of successful trials.

Fig. 11. The result with the simulator by clustering approximation method in the experiment II-2 (the performance in the test data).

prior research [4]. Even when the reinforcement learning is used, for example, it is not possible to carry out the sufficient learning. This fact implies that our learning approach is very useful. However, the learning performance is affected by the training situation, so it is necessary to conduct a more detailed investigation concerning the learning of the simulator.

## VI. Future Research

The results of this research demonstrated the effectiveness of simulator construction based on the data acquired from a real robot. We would like to perform a verification by installing this method on a real robot and carrying out the learning in real time. The use of the acquired simulator is not limited to the target task by which the simulator has been produced. When the target task is changed, we may use the simulator with necessary modification of the evaluation functions for the new task if the environment is the same as before. This will enable the robot to acquire the controller for the new task immediately. We will plan to conduct the experiment to confirm this idea.

In this research, it was assumed that all of the past data could be used as training data. However, the question of how much data should be retained remains an important problem. If all of the data are accumulated, long learning time will be required to construct the simulator. It should be possible to efficiently select data based on the information criterion, such as by preferentially accumulating data with large prediction errors.

We are working on the extension of the proposed method for the sake of applying to multi-agent environment, in which one agent has to predict actions of other agents. This means that the agent has to construct a model to predict others' actions in observing their past actions. We have some preliminary experimental results in the multi-humanoid robot environment (see [3] for details).

## VII. Conclusion

An attempt was made to construct a simulator of the real environment from sensor data (e.g. input image) obtained by a robot moving in such a real environment. Two methods were studied for that construction. In the incremental learning that assumed a real situation, there were no signs of overfitting when GP was used. Thus, it is expected that the prediction performance of the simulator will progressively be improved every time the learning is repeated.

As a result of the learning experiments using the acquired simulator, GP evolved useful controllers successfully. Consequently, based on a relatively small amount of experiences in a real environment, we were able to carry out the effective controller learning, and the learning has been accelerated satisfactorily.

## References

[1] 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.

[2] John J. Grefenstette and Connie Loggia Ramsey. An approach to anytime learning. In *Proc. of the Ninth Int. Machine Learning Workshop*, pages 189–195, San Mateo, CA, 1992. Morgan Kaufmann.

[3] Y. Inoue, T. Tohge, and H. Iba. Cooperative transportation by humanoid robots – learning to correct positioning –. In *Proc. of the Hybrid Intelligent Systems (HIS2003)*, pages 1124–1133, 2003.

[4] Shotaro Kamio and Hitoshi Iba. Real-time adaptation technique to real robots: An experiment with a humanoid robot. In *Proc. of 2003 Congress on Evolutionary Computation (CEC2003)*, pages 506–513, Canberra, Australia, December 8-12 2003. IEEE Press.

[5] 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.

[6] Ibrahim Kushchu. Learning, evolution and generalisation. In *Proc. of 2003 Congress on Evolutionary Computation (CEC2003)*, pages 506–513, Canberra, Australia, December 8-12 2003. IEEE Press.

[7] David J. Montana. Strongly typed genetic programming. Technical Report #7866, 10 Moulton Street, Cambridge, MA 02138, USA, 7 1993.

[8] J. P. Nordin, W. Banzhaf, and M. Brameier. Evolution of world model for a miniature robot using gentic programming. *International Journal of Robotics and Autonomous systems*, 1998.

[9] Gary B. Parker. Co-evolving model parameters for anytime learning in evolutionary robotics. *Robotics and Autonomous Systems*, 33(1):13–30, 2000.

[10] Gary B. Parker. Punctuated anytime learning for hexapod gait generation. In *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2002)*, pages 2664–2671, 2002.

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

[12] 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.