Incremental Learning of a Robot Controller by Means of Genetic Programming

Shotaro Kamio and 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*— The acquisition of effective behaviors for real robots requires learning not only by the simulation, but also within the real environment. The simulator is an important part of the learning system in that situation. This is because a useful simulation can accelerate the overall learning.**

In this paper, we propose an approach to the learning acceleration by the robot while behaving in the real environment. In this approach, the simulator is constructed based on data retrieved from the environment. The controller of the robot is trained with the acquired simulator. We present simulation results to show that the effective controller has been evolved by GP successfully.

I. Introduction

The learning of robots' behavior with the simulation has some limitations. Because the resultant behavior cannot easily be applied directly to a real environment. For this reason, other techniques are consulted for the learning by many researchers. Parker [8] described other two approaches: i.e., learning from the real environment, and learning by using the simulation and the real environment. The former approach takes too much time to complete the learning. The latter, therefore, is the reasonable approach.

We have proposed real-time learning method for a robot and showed experimental results in the prior researches [6], [4]. This method is aimed at acquiring the fundamental behavior by means of GP in a simple simulation, performing Q-learning using a real robot, and adapting the behavior to the real environment. In [4], we performed the "box moving" task with a humanoid robot (see Fig. 1). 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.

Parker proposed punctuated anytime learning [8]. 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.

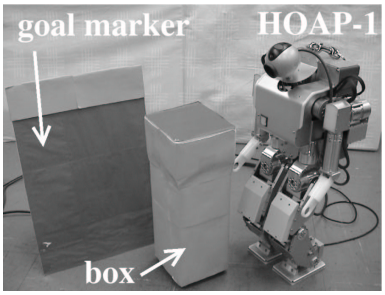


Fig. 1. The humanoid robot "HOAP-1", the box and the goal marker.

We should construct a model which is enough precise to learn the controller of the robot. Several studies dealt with the model construction based on the data from the real environment. Grefenstette et al. have proposed anytime learning [2]. In this framework, the agent creates its strategy using a simulation model. Because the agent acts in the environment, it is possible to bring the simulation model closer to the real environment using data acquired in the environment. However, their 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. Therefore, it is difficult to implement this system to the real robot. This problem arises because the simulation model is constructed from the human viewpoint.

The model used in these studies were applied only to simplified problems. For the purpose of a real robot task as shown in Fig. 1, we have to consult complex models with more realistic conditions. In this paper, we present a control architecture in order to incrementally acquire a simulator. The simulator is constructed using the data retrieved by the robot while behaving in the real environment. This architecture enables to generate the controller of the robot with the constructed simulator.

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 experiments and their results. Section VI discusses the effectiveness of our proposed architecture. Section VII explains related researches. Section VIII describes future issues, and section IX gives some conclusions.

II. Proposed Simulator and Control Architecture

The simulator is generally constructed in order to express physical interactions of robots and objects. If we focus the

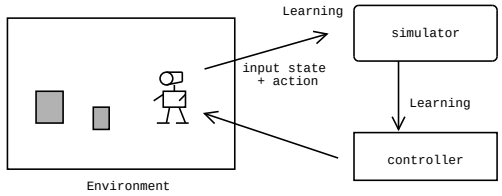


Fig. 2. The overall architecture of the proposed control system.

interaction of the robot and the environment, however, we can simply define what the simulator really is. Provided that the environment has the Markov property [9], the state of the environment ($\boldsymbol{s_t}$) and the action of the robot at that time (a_t) generate the resulting state by an action ($\boldsymbol{s_{t+1}}$). This is written in the following mathematical form:

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

Therefore, we can consider the simulator (i.e., the environmental model) as a function for predicting the effect of an action in a state. We can estimate this function from the generalization of experienced states. This estimation is done by a function approximation method (two methods were described in section III). If a simulator can be obtained, we can use any learning algorithm whatever on it. After the learning, its result (i.e., a controller of the robot) will be transferred to the robot and control it to achieve tasks. The overall architecture we propose is illustrated in Fig. 2.

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 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, 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). Note that these data were measured in the real environment and contained noise.

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 [7]. 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., a neural network and a clustering approximation method. We used a neural network instead of GP in the previous research [5].

A. Neural Network (NN)

The back propagation neural network can learn the relation of inputs and outputs of a system. Therefore, we can estimate the function f in eq.(1) by NN. The input of the NN is current input image data ($\boldsymbol{i}$) and selected action (a) at that time. The output is the difference of the input ($\dot{\boldsymbol{i}}'$) as a result of the action. We used 12 neurons for the input layer and five neurons for the output layer. Seven input neurons represent seven actions, respectively. Only one of them was given the value of 1.0 if the corresponding action is selected. The others were set to be the value of 0.0. We experimented with 10, 20 and 40 neurons for the hidden layer.

B. Clustering Approximation Method

We also used the method which automatically constructs the state space of Q-learning ("the second method" described in [10]). 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.

1) Model Construction [10]: The local approximation model for each cluster is a linear approximation of the data as follows:

$$\dot{\boldsymbol{i}}' = \boldsymbol{A}\boldsymbol{i} + \boldsymbol{b}. \tag{2}$$

The detailed algorithm of constructing the local model is described below, where d is a triplet data of $(a, \boldsymbol{i}, \dot{\boldsymbol{i}}')$ and a_i is ith 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 (2) 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 $C := C_2$.

Finally, each cluster has the coefficient $\boldsymbol{A}$ and the constant $\boldsymbol{b}$ which represent the local model (2).

TABLE I

PARAMETERS OF THE NN IN THE EXPERIMENT I.

Number of neurons in the hidden layer	10, 20 or 40
Rate of increasing number of training data set (per 1,000 iterations)	4, 5 or 10
Number of training iterations	350,000, 250,000 or 150,000
Learning rate	0.01

2) Prediction by the Local Approximation Model: We can predict the effect of an action in unobserved input data using the above local model. For this prediction, the nearest cluster to the current input i is searched for and used.

IV. EXPERIMNT I: SIMULATOR CREATION BY INCREMENTAL LEARNING

We performed experiments to create a simulator using the two methods. 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. In a real robot, the timing at which one set of data are acquired differs from one system to another. Moreover, the acquired data can be utilized immediately, or it may be possible to wait until a certain amount of data has been accumulated. This means the designer can decide the timing at which to increase the training data sets.

A. Results of NN

Each experiments was started from ten sets of training data, The rate of increasing number of training data set were tested in three ways: four sets, five sets or ten sets per 1,000 iterations of learning. The learning finished at the 350,000 iterations, the 250,000 iterations or the 150,000 iterations, respectively. The number of the training data sets was not increased past 1,210. The test data always contained 100 data sets different from training data sets. Table I shows the parameters we used.

Figure 3 shows the obtained results. The "h" in the figure means the number of neurons in the hidden layer and the "inc" means the rate of increasing number of training data sets. The horizontal axis is the number of training iterations. It should be noted that the vertical axis is the average prediction error in the test data (average of 10 trials). In all cases, the prediction error in the test data decreases along with the repetition of learning. As can be seen, it seems that the overfitting does not occur in the incremental learning. The final performance is not significantly affected by the rate of increasing the training data sets. This means the final performance does not depend upon the way in which the number of the training data increases.

B. Results of Clustering Approximation Method

The clustering approximation method is not an incremental learning. Hence, it was experimented with fixed training data sets. We used k-mean clustering as the clustering method. k-mean clustering needs the upper bound of the variance in a cluster for the termination condition. This

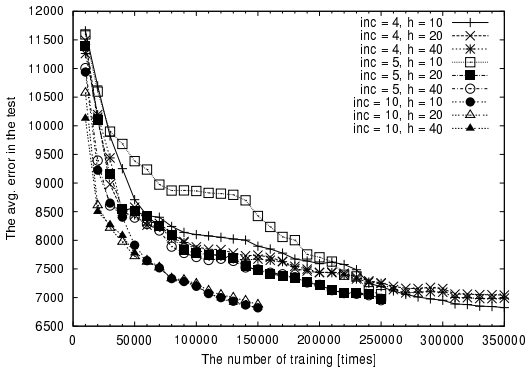


Fig. 3. The average prediction error by the NN in the experiment I.

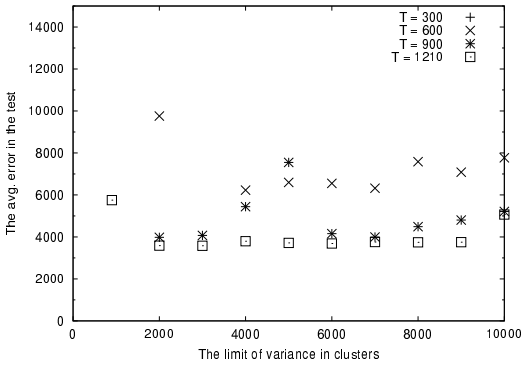


Fig. 4. The average prediction error by the clustering approximation method in the experiment I.

upper bound affects the performance (i.e., the granularity) of clustering. We tested with the following upper bounds: $1{,}000, 2{,}000, ...,$ 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 (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 are gradually increased. In the incremental learning, the overfitting was not found when a neural network was used. It may be because new data are continuously added, so that the selective pressure acts to generalize the results of learning.

With a large amount of training data set, the performance of the simulator by the clustering approximation method is better than that by NN. The final performance obtained by the NN is better than that by the GP in the previous research [5].

The above results would seem to indicate that the performance does not depend upon the rate at which the number of the training data is increased in the incremental

TABLE II

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 III

PARAMETERS OF GP FOR THE LEARNING OF THE CONTROLLER.

Population size	1000
Generations	50, 100, 150
Crossover rate	0.7
Mutation rate	0.1
Rate for introducing random individuals	0.1
Trials	10

learning. This means that we do not have to worry about setting the precise rate at which the number of the training data is increased when performing the learning using a real robot.

V. EXPERIMENT II: TRAINING A CONTROLLER USING THE INCREMENTALLY IMPROVED SIMULATOR

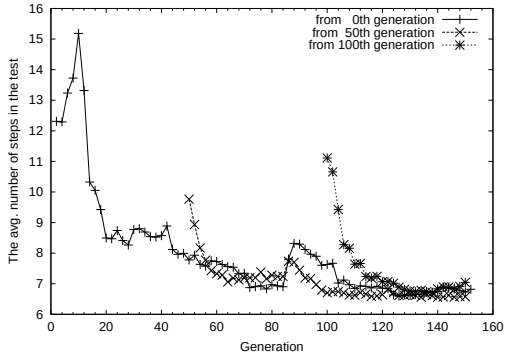
We performed experiments to see if the useful controller for a robot can be trained in our control architecture. The learning of the controller was executed with the simulator generated by the incremental learning. The box motion characteristics in the simulation may possibly change every time the simulator is improved. The controller of a robot, therefore, is expected to adapt to the simulator with learning. We tested GP and Q-learning for the purpose of learning the controller.

As the simulator, those obtained by the NN and the clustering approximation method were used. The training data used to acquire 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. The simulator by the NN 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.

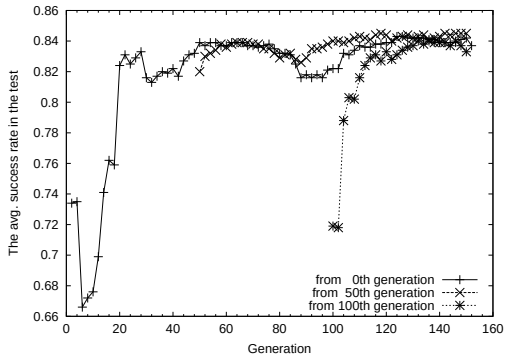
A. Learning the controller by GP

The used function and terminal nodes are shown in Table II. The parameters used are given in Table III. 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 of the camera. The fitness function is defined as follows:

$$\textit{fitness} = \frac{1}{N}\sum_{i}^{N} \text{steps}_i, \tag{3}$$



(a) The average steps to achieve the task.



(b) The average rate of successful trials.

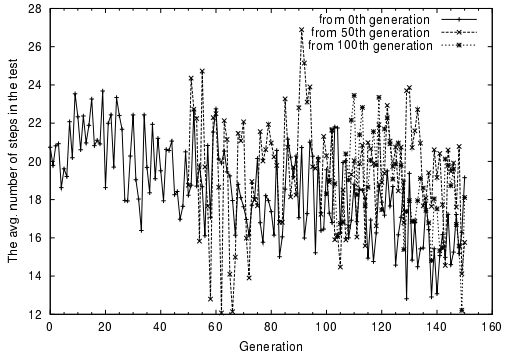
Fig. 5. The learning result of the controller by GP in the experiment II. The simulator was acquired by NN.

where steps_i means the number of steps to complete the ith trial and N is the number of sets of training data. If the task is not be accomplished, we assign $30 + (30 - \text{valid\_steps}_i)$ to steps_i as a penalty. In this definition, 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. The performance in test data is evaluated by the same fitness function.

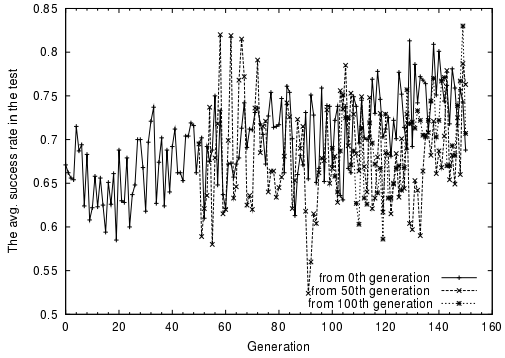
The controller learning began at each one of the 0th, 50th and 100th generations. We calculated the generation of the incrementally acquired simulator by the number of the learning iteration divided by 1,000 because the population size of the GP is 1,000. The training data were the same with those by 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 5 shows the results with the simulator by NN. The average steps were reduced to about seven and the success rates were about 84% in all cases at the final generation. Note that these results were obtained with the test data. This indicates that the effective controllers were evolved by GP successfully.

Figure 6 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,

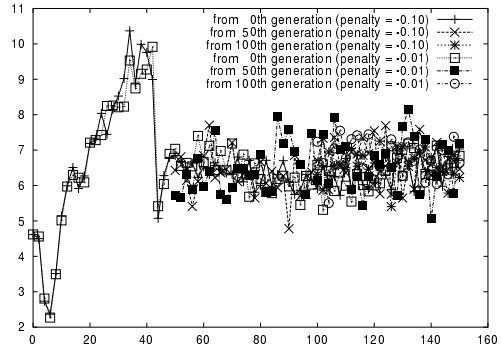


(a) The average steps to achieve the task.

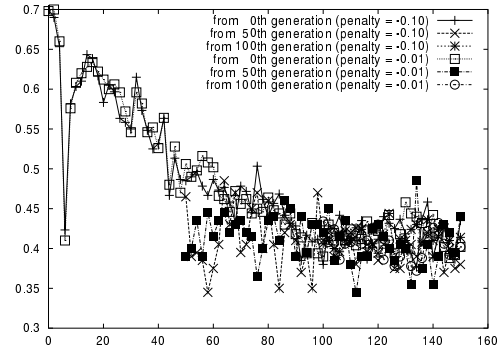


(b) The average rate of successful trials.

Fig. 6. The learning result of the controller by GP in the experiment II. The simulator was acquired by clustering approximation method.



(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 in the experiment II. The simulator was acquired by NN.

TABLE IV

PARAMETERS USED FOR Q-LEARNING IN THE EXPERIMENT II.

Learning rate	$0.05/(1.0 + (\text{total accumulated steps})/10^5)$
Discount rate	0.8
Reward (success)	1.0
Reward (failure)	$-0.01, -0.1$

its performance is supposed to change by every reproduction. These changes will affect the controller learning with the simulator.

B. Learning the controller by Q-learning

The state space of this Q-learning is the same as that used in our real-time adaptation study [4]. We cannot evaluate the controller using the same fitness function with GP (eq. (3)) 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 of -0.01 (or -0.1) for a failure. The discount rate γ was set to 0.8. In experiments, $1,000$ trials in Q-learning was treated equally as one generation in the GP because the population size of the GP is $1,000$.

Fig 7 shows the results with the simulator acquired by NN. The number of average steps were almost equal to that by GP controller. The success rate decreased at first and became to 0.4 at the final generation. This is because the simulator in the early stage present poor prediction performance and the training data set is small amount. Therefore, the learning of the controller was easy with the simulator. However, at the late stage of the learning, the situation changes. The training data set become large amount. Moreover, the motion characteristics of the box in the simulator changes as a result. It is difficult for the Q-learning to learn the change properly.

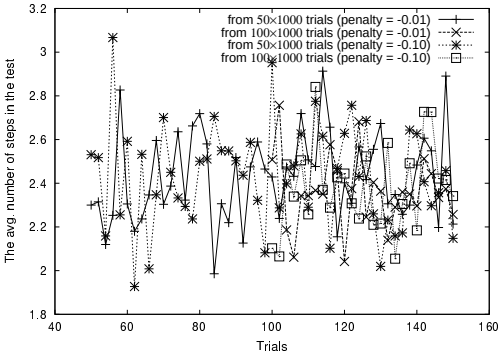
Fig. 8 presents the results with the simulator acquired by clustering approximation method. Low average steps and low success rate were observed in the figure. Furthermore, the lesser average steps were, the lower rate of the success was. This fact suggests that the acquired controller could only achieve the easy tasks which required low steps to complete.

Compared with the results of the GP and Q-learning, we can conclude that the performance of the controller by GP is better than that by Q-learning.

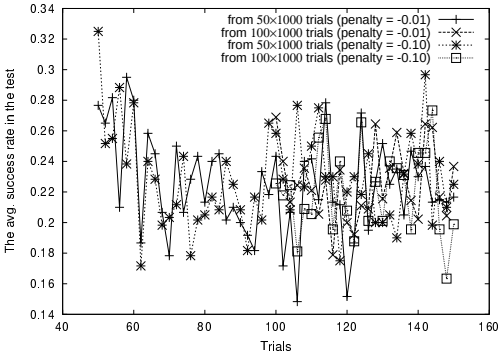
VI. DISCUSSION

Experiments were carried out to show the validness by using the proposed control architecture. It was found possible to train a controller by GP while improving the performance of simulation.

It was confirmed that an effective controller was acquired with the simulator constructed by NN. The controller achieved high success rate more than 80% from the 20th generation. The number of data sets available at the 20th generation is the same as the one acquired during the humanoid robot's 20 minutes learning (about 200 actions) in our real-time adaptation [4]. This is fewer generation



(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 in the experiment II. The simulator was acquired by clustering approximation method.

than that obtained result by the simulator evolved with GP in the previous study [5]. Even when the reinforcement learning is used, it is not possible to carry out the sufficient learning in that time. This fact indicates that our proposed approach is very useful for real robots.

VII. Related Researches

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. Q-learning does not necessarily result in the progress of the learning without rewards. For the progress, the successful trial must be repeated in early stage of the learning. For example, some learning scheme such as "Learning from Easy Mission" [1] will be required. However, Our approach utilizes non-rewarded data also for the learning of the simulator.

Sethu and Schaal developed Locally Weighted Projection Regression (LWPR) algorithm for high-dimensional data [11]. Their algorithm can incrementally construct the approximate function which expresses the inverse dynamics of a high-dimensional robot. However, they did not study the controller learning with their incremental model learning.

VIII. Future Research

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).

IX. Conclusion

We showed that the incremental learning of the robot controller is possible with the incrementally acquired simulator. As the results of the experiments with real data, GP evolved useful controllers successfully. Consequently, in spite of 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. We will plan to perform a verification by installing this method on a real robot and carrying out the real-time learning.

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 and Hitoshi Iba. Evolutionary construction of a simulator for real robots. In *Proceefings of the 2004 Congress on Evolutionary Computation (CEC2004)*, pages 2202–2209. IEEE Press, 2004.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] Sethu Vijayakumar and Stefan Schaal. Fast and efficient incremental learning for high-dimensional movement systems. In *Proc. of International Conference on Robotics and Automation (ICRA2000)*, 2000.