A Method for Learning Macro-Actions for Virtual Characters Using Programming by Demonstration and Reinforcement Learning

Yunsick Sung* and Kyungeun Cho**

Abstract—The decision-making by agents in games is commonly based on reinforcement learning. To improve the quality of agents, it is necessary to solve the problems of the time and state space that are required for learning. Such problems can be solved by Macro-Actions, which are defined and executed by a sequence of primitive actions. In this line of research, the learning time is reduced by cutting down the number of policy decisions by agents. Macro-Actions were originally defined as combinations of the same primitive actions. Based on studies that showed the generation of Macro-Actions by learning, Macro-Actions are now thought to consist of diverse kinds of primitive actions. However an enormous amount of learning time and state space are required to generate Macro-Actions. To resolve these issues, we can apply insights from studies on the learning of tasks through Programming by Demonstration (PbD) to generate Macro-Actions that reduce the learning time and state space. In this paper, we propose a method to define and execute Macro-Actions. Macro-Actions are learned from a human subject via PbD and a policy is learned by reinforcement learning. In an experiment, the proposed method was applied to a car simulation to verify the scalability of the proposed method. Data was collected from the driving control of a human subject, and then the Macro-Actions that are required for running a car were generated. Furthermore, the policy that is necessary for driving on a track was learned. The acquisition of Macro-Actions by PbD reduced the driving time by about 16% compared to the case in which Macro-Actions were directly defined by a human subject. In addition, the learning time was also reduced by a faster convergence of the optimum policies.

Keywords—Reinforcement Learning, Monte Carlo Method, Behavior Generation Model, Programming By Demonstration, Macro-Action, Multi-Step Action

1. INTRODUCTION

In order to apply reinforcement learning to complicated game environments involving real-time strategies, it is necessary to identify a method such as hierarchical reinforcement learning (HRL) to divide a problem into its sub-problems to solve it [1]. However, an enormous amount of learning time is required even after dividing a problem into its several parts. An approxima-

※ This work was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology (2011-0011266)
Manuscript received October 6, 2011; accepted February 9, 2012.
Corresponding Author: Kyungeun Cho
* Dept. of Game Engineering, Graduate School, Dongguk University-Seoul, Seoul, Korea (sung@dongguk.edu)
** Dept. of Multimedia Engineering, Dongguk University-Seoul, Seoul, Korea (cke@dongguk.edu)

tion function [2], modular learning [3], or Macro-Actions [4] to reduce the learning time can satisfy such a requirement. An approximation function estimates the value function of a unlearned intermediate state between two learned states. In modular learning, a state space is divided and learned, and primitive actions are selected on the basis of learning. Finally, Macro-Actions are combinations of primitive actions. With the use of the Macro-Actions, agents can reduce the number of policy decisions and hence the learning time.

A Macro-Action can be defined in one of two ways. According to one method, a Macro-Action is consecutive primitive actions that are manually defined and that consist of the same primitive actions [4]. A second method defines a Macro-Action by learning [5]. Although the latter method can automatically generate Macro-Actions, it leads to increasing the learning time [5] and the size of the state space [6] that are required to learn Macro-Actions. These problems can be solved with the use of Programming by Demonstration (PbD) [7]. PbD is a method that allows agents to learn by analyzing the actions of a predecessor. Studies on PbD have shown that it can generate the tasks of robots [8] or virtual agents [9] from the actions of a predecessor with a small amount of training.

In this paper, we propose a method to solve the problems that hinder the generation of Macro-Actions owing to the time and state space required for learning in complicated environments such as in a game environment. In this method, the Macro-Actions of virtual characters are generated by using PbD and are executed by using reinforcement learning. In a verification experiment, we applied the proposed method to a car simulation in a virtual environment. We then describe a series of processes to generate and execute the Macro-Actions for driving a car.

The paper is organized as follows: in Section 2, we provide an overview of studies on Macro-Actions and PbD. In Section 3, we describe how to generate and execute Macro-Actions, and in Section 4, we explain the processes to apply the proposed method to a car simulation. Finally, we summarize the proposal in Section 5.

2. RELATED WORK

Here, we review studies of reinforcement learning on Macro-Actions. We also present studies on PbD, which can be applied when generating Macro-Actions.

2.1 Reinforcement Learning with Macro-Actions

A Macro-Action has been defined as a combination of primitive actions and is executed by Q-learning [4]. Agents select a primitive action or a Macro-Action to be executed in each state. After selecting a Macro-Action, it successively executes the primitive actions defined within it. Nash Q-learning is also used to execute Macro-Actions in multi-agent environments [10]. In conventional Macro-Actions, the primitive actions are executed in discrete time. Therefore, the difficulty of executing Macro-Actions in continuous time has been addressed [11].

In one class of studies, only the same primitive actions were manually combined and then they were executed [4]. It is difficult to define a variety of Macro-Actions as combinations of the same primitive actions. In another class of studies, Macro-Actions have been generated by learning [5]. For example, new Macro-Actions are generated by crossover and mutation using a genetic algorithm [5]. The generated Macro-Actions are then executed by Q-learning. In this study, the learning time is increased by iterative Macro-Action generation and policy learning.

2.2 Task Learning Based on Programming by Demonstration

In PbD studies, robots learn the tasks to be executed by observing the motions of a predecessor [12]. It then executes the learned tasks. In other study, virtual agents learn tasks by analyzing the consecutive actions of a virtual agent controlled by a human subject [9]. This method derives all possible consecutive actions that can be defined as tasks from the observed actions. The tasks to be executed by a virtual agent are then selected by the *Maximin Selection* algorithm. This method is appropriate when the number of tasks to learn is predefined. Similarly, in a study on task learning in robots, the behaviors are observed to generate tasks [8]. Consecutive behaviors are grouped based on the variance and are selected as the tasks of the robot. Finally, in some studies, the tasks to be executed by a robot are learned from the tasks that are executed repetitively by a human subject [13]. The tasks are then defined by a behavior network.

In this paper, we propose a method to generate Macro-Actions by PbD and execute generated Macro-Actions by reinforcement learning.

3. Generation and Execution of Macro-Actions for Virtual Characters

In this section, we introduce a model and relevant algorithms to automatically generate and execute the Macro-Actions of virtual characters.

3.1 Overview of the Macro-Action Model

We describe the relationship between the proposed model and a virtual environment. A human subject directly controls a virtual character to execute the primitive actions as shown in Fig. 1. The executed actions are delivered to the proposed model, and the Macro-Actions are generated by learning. Furthermore, the policy to execute Macro-Actions in a virtual environment is learned by reinforcement learning.

The proposed model consists of a *Macro-Action Generation Stage* and a *Macro-Action Exe-*

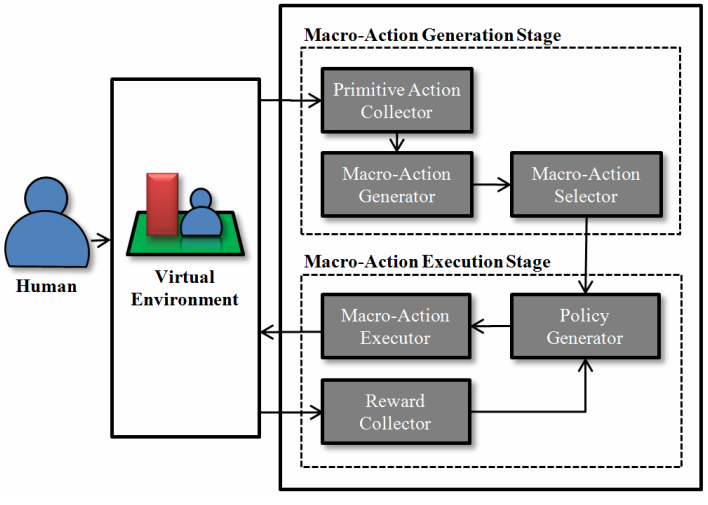


Fig. 1. Proposed two-stage model

cution Stage. In the former, which contains the *Primitive Action Collector*, *Macro-Action Generator*, and *Macro-Action Selector*, primitive actions are collected to generate Macro-Actions. The policy to execute the generated Macro-Actions is learned during Macro-Action Execution Stage, which contains the *Policy Generator*, *Macro-Action Executor*, and *Reward Collector*.

3.2 Macro-Action Generation Stage

In this stage, Macro-Actions are generated by the collection of a series of primitive actions in a virtual environment as follows: the Primitive Action Collector receives the primitive actions executed by a virtual character in the virtual environment. The n*th* primitive action consists of multiple movements and duration d_n, as shown in Eq. (1). Here, m_n^1 is the first movement comprising the primitive action a_n. After the control of a virtual character by a human subject, all collected primitive actions are transferred to the Macro-Action Generator.

$$a_n = [m_n^1 \cdot m_n^2 \cdot \ldots, d_n] \tag{1}$$

The Macro-Action Generator receives the consecutive primitive actions and derives the Macro-Action candidates. As shown in Eq. (2), all consecutive primitive actions from the certain positions of u to v among the z primitive actions that have been collected are defined as Macro-Action candidates. Here, c_j is the Macro-Action candidate of the j*th* order. However, Macro-Action candidates with durations greater than $\varepsilon_{Duration}$ are eliminated to control the calculation amount. All Macro-Action candidates are transferred to the Macro-Action Selector.

$$c_j = a_u \cdot a_{u+1} \cdot \ldots \cdot a_v,\ 1 \le u \le v \le z \tag{2}$$

The Macro-Action Selector sets $\varepsilon_{Macro\text{-}Action}$ as the number of Macro-Actions to be selected in advance. The $\varepsilon_{Macro\text{-}Action}$ Macro-Actions that are to be executed by a virtual character are selected from among multiple Macro-Action candidates by using the Maximin Selection algorithm[9]. This algorithm selects the representative Macro-Actions to minimize the sum of differences between a group of Macro-Action candidates and a selected group of representative Macro-Actions. The difference between two Macro-Actions is calculated by the function Diff, as shown in Eq. (3).

$$\mathrm{Diff}(e_p, b_k) = \left(\sum_{t=1}^{\min(d_p, d_k)} \sum_{i=1}^{\varepsilon_{Movement}} \left(m_{p,t}^{\,i} - m_{k,t}^{\,i} \right)^2 \right)^{\frac{1}{2}} \tag{3}$$

Here, b_k is the selected representative Macro-Action of the k*th* order; $m_{k,t}^{\,i}$ is the i*th* movement at time t comprising the k*th* Macro-Action; d_k is the duration of primitive actions comprising the k*th* Macro-Action; and $\varepsilon_{Movement}$ is the predefined number of movements. The function Diff calculates the difference by comparing the movement of each action that the Macro-Actions are comprised of. Two Macro-Actions could have different durations. Thus, the movements remaining after the comparison are calculated separately as shown in Eq. (4).

$$\mathrm{Diff}(e_p, b_k) = \left(\sum_{t=1}^{\min(d_p,d_k)} \sum_{i=1}^{\varepsilon_{\mathrm{Movement}}} \left(m_{p,t}^{\,i} - m_{k,t}^{\,i}\right)^2 + \sum_{t=\min(d_p,d_k)+1}^{d_p} \sum_{i=1}^{\varepsilon_{\mathrm{Movement}}} \left(m_{p,t}^{\,i}\right)^2 + \sum_{t=\min(d_p,d_k)+1}^{d_j} \sum_{i=1}^{\varepsilon_{\mathrm{Movement}}} \left(m_{k,t}^{\,i}\right)^2 \right)^{\frac{1}{2}} \quad (4)$$

Macro-Actions are selected by using Eq. (5) as follows: initially, the first Macro-Action candidate c_1 is selected as the first representative Macro-Action b_1, as shown in Eq. (5).

$$b_1 = c_1 \quad (5)$$

Subsequently, the Macro-Action candidate with the greatest value for the function Diff compared to the first representative Macro-Action b_1 is selected as the representative Macro-Action b_2, as shown in Eq. (6).

$$b_2 = \max_c \mathrm{Diff}(b_1, c) \quad (6)$$

From the third selection onward, each Macro-Action candidate is compared with all of the selected representative Macro-Actions. The minimum value of Diff (b, c_h) is calculated for each comparison. The Macro-Action candidate with the largest minimum value is then selected, as shown in Eq. (7).

$$b_k = \max_{c_h} \min \mathrm{Diff}(b, c_h) \quad (7)$$

After the $\varepsilon_{\text{Macro-Action}}$ Macro-Actions are selected, all processes in the Macro-Action Generation Stage are completed. The selected Macro-Actions are transferred to the Macro-Action Execution Stage.

3.3 Macro-Action Execution Stage

In this stage, the way in which to execute all of the Macro-Actions that were selected in the previous stage is learned. The result of learning this execution is recorded in the Q-values. Initially, the Policy Generator receives no reward. Therefore, all Q-values are initialized to 0. The initialized Q-table, which contains all of the Q-values, and the Macro-Actions are transferred to the Macro-Action Executor.

The Macro-Action Executor proceeds as follows: if no Macro-Action is being executed at present, the Macro-Action Executor selects the Macro-Action with the maximum Q-value of the state of a virtual character. It then executes the first primitive action of the Macro-Action in the virtual environment. If the virtual character is executing a Macro-Action at present and the execution time of the current primitive action is greater than or equal to its duration, the next primitive action of the Macro-Action is executed. If there is no further primitive action to be executed,

another Macro-Action is selected and its first primitive action is begun. In the conventional Macro-Actions, the virtual agent executes primitive actions or Macro-Actions. However, the Macro-Action Executor only executes Macro-Actions.

The Reward Collector collects the rewards which are generated in the virtual environment and transfers them when the Policy Generator generates the policies. The Policy Generator receives the rewards and updates the Q-table. The Q-table can be updated in various ways depending on the reinforcement learning algorithm. The Policy Generator updates the Q-values as shown in Eq. (8) using Monte Carlo methods. The cumulative average reward is assigned to the Q-value.

$$Q(s_t, b_t) \leftarrow \text{average}(\text{Returns}(s_t, b_t)) \tag{8}$$

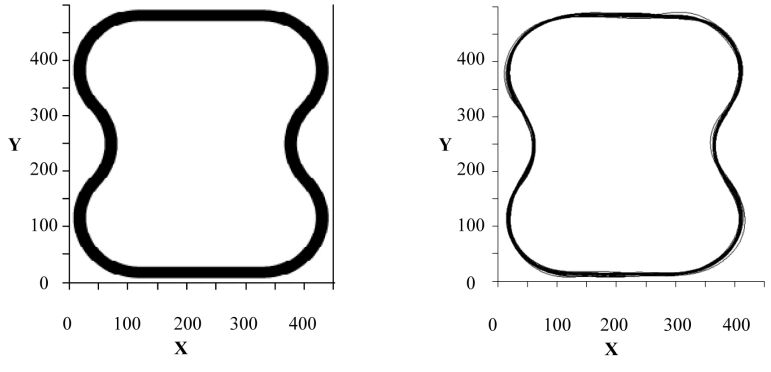
The updated Q-table is transferred to the Macro-Action Executor, which in turn executes the Macro-Actions using a new policy.

4. EXPERIMENT

To verify the proposed model, *E-Track 5* from *The Open Racing Car Simulation* (TORCS) was adopted. The diagrams in Fig. 2 show the track and all of the paths driven by a human subject, respectively. For E-Track 5, two laps constitute one race. The data from a total of 10 races were collected in the experiment. As shown in Eq. (9), the primitive actions in this experiment consist of using the driving wheel m_n^w, accelerator m_n^a, and brake m_n^b.

$$a_n = [m_n^w \cdot m_n^a \cdot m_n^b, d_n] \tag{9}$$

Fig. 3 shows the driving record of the first race. For m_n^w, which responded sensitively, the range of the measured values was 40–60. That is, the m_n^w values were neither smaller than 40 nor larger than 60. On the other hand, the range of values for m_n^a and m_n^b was 50–100. This graph indicates that the driver continuously changed speeds by using the accelerator and brake. In the track, there were two straight-line sections. However, examining the driving records



(a) E-Track 5 Track (b) Driving Path

Fig. 2. Racing track

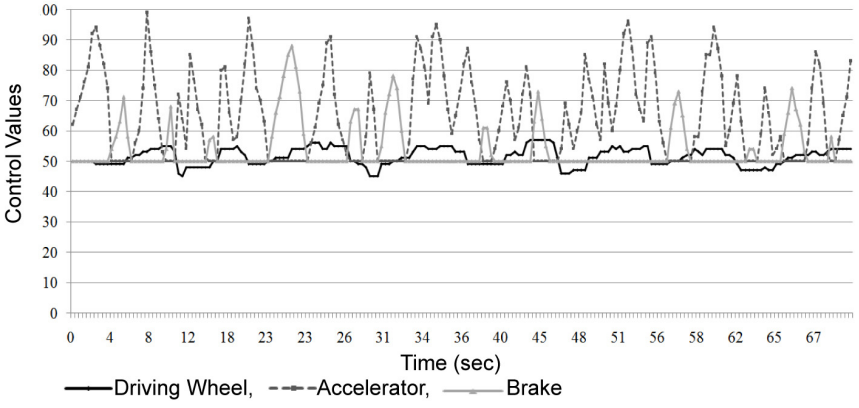


Fig. 3. Driving record showing the primitive actions for the first race

shows that there were many movements of the driving wheel, accelerator, and brake even in the straight-line section. Moreover, the accelerator and brake were used in alternation.

TORCS delivered the primitive actions to the Primitive Action Collector. When the one-time race was completed, all of the actions were delivered to the Macro-Action Generator, which in turn drew 2,034 Macro-Action candidates with diverse combinations of consecutive primitive actions. The Macro-Action Selector selected 40 representative Macro-Actions with the Maximin Selection algorithm from the Macro-Action candidates. Fig. 4 shows 16 of the 40 Macro-Actions generated by the Macro-Action Generation Stage.

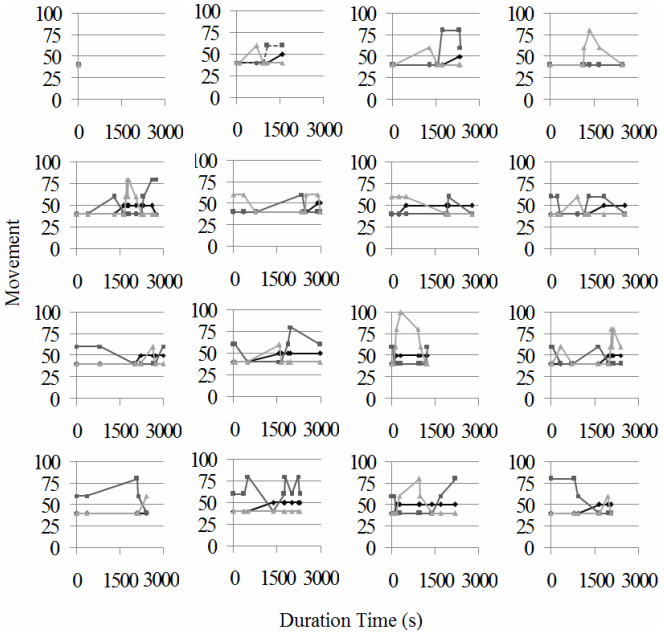


Fig. 4. Generated Macro-Actions

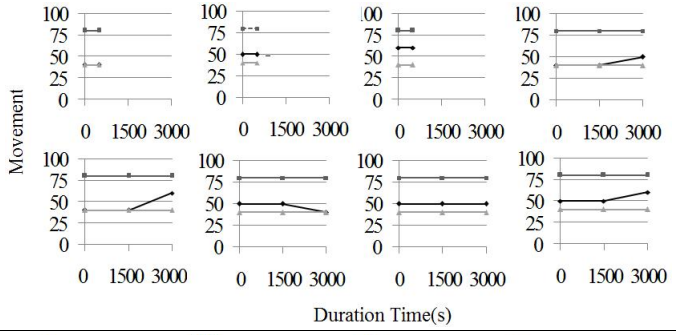
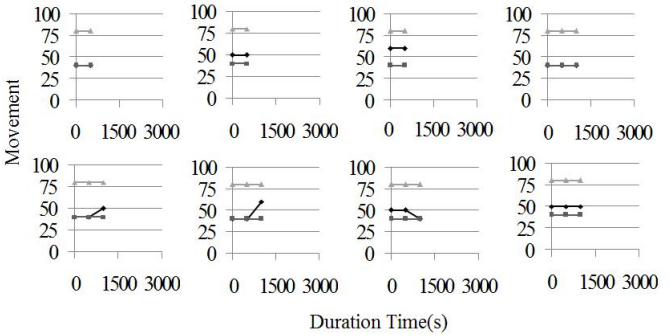
The experiment compared the race results of 40 Macro-Actions that were defined by a human subject and those of the Macro-Actions that were generated by the proposed model. The user-defined Macro-Actions consisted of 12 primitive actions defined by a human subject as shown in Table 1. The 28 Macro-Actions enabled acceleration and 12 Macro-Actions enabled deceleration. A subset of the defined Macro-Actions is shown in Table 2.

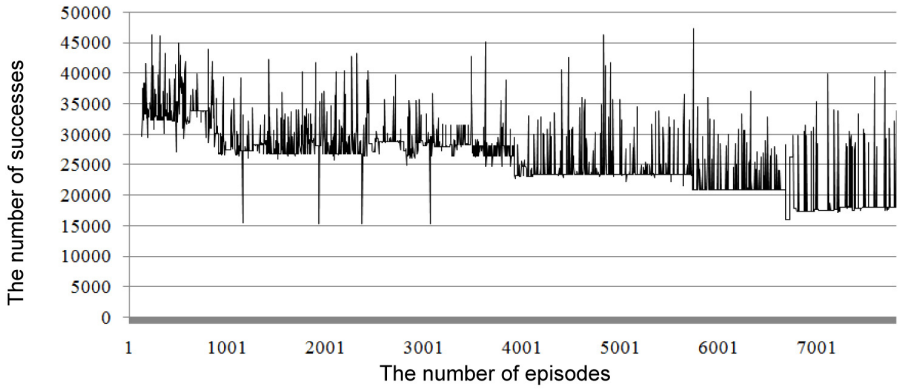
Running times were compared after each group of Macro-Actions was learned 8,000 times in

Table 1. 12 human-defined actions

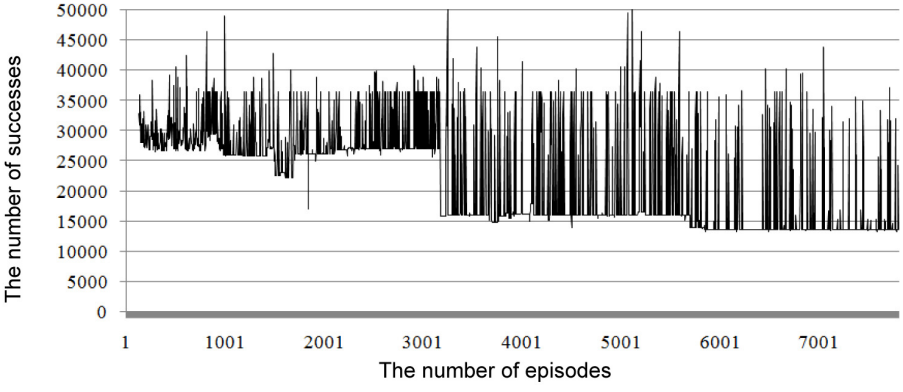
	w_1	c_1	b_1	d_1
a_1	40	80	40	1,500
a_2	50	80	40	1,500
a_3	60	80	40	1,500
a_4	40	80	40	1,000
a_5	50	80	40	1,000
a_6	60	80	40	1,000
a_7	40	80	40	500
a_8	50	80	40	500
a_9	60	80	40	500
a_{10}	40	40	80	500
a_{11}	50	40	80	500
a_{12}	60	40	80	500

Table 2. human-defined Macro-Actions

	Macro-Actions
28 Accelerating Macro-Actions	
12 Decelerating Macro-Actions	



(a) Result of learning with a human subject defined Macro-Actions



(b) Result of learning with generated Macro-Actions

Fig. 5. Running time with user-defined and learned Macro-Actions

a section of the track that required 15 sec for a human subject to drive a car. Fig. 5 shows the running time for each completed race. Fig. 5(a) shows the running time when the Macro-Actions were directly defined by a human subject. The further the learning progressed, the more the running time was reduced. After 8,000 learning episodes, the virtual character took 16.032 sec to run the section that required 15 sec for a human subject. Fig. 5(b) shows the results with Macro-Actions that were acquired by the proposed method. The initial running time of the proposed method was faster than that of the Macro-Actions that had been defined by a human subject. The running time did not decrease until 3,044 learning episodes. On the 3,045*th* episode, the running time fell to 15.966 sec. Thereafter, the running time decreased to a minimum of 13.545 sec. Therefore, the virtual character drove faster than the human subject.

The speed of a car can be easily calculated from the time required to drive a fixed distance. However, the amount to which a car can be driven as naturally as by a human subject cannot be determined. Therefore, the paths driven by a human subject were compared with those that were driven by Macro-Actions. A human subject drove along the track of E-Track 5. The Macro-Actions defined by a human subject drove the car all the way around the track. However, the

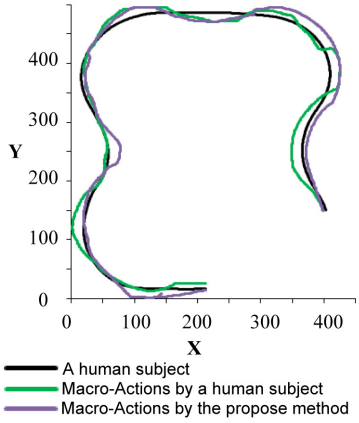


Fig. 6. Comparison of driving courses

path of the Macro-Actions defined by a human subject frequently veered off the track, and the car sometimes wandered. The Macro-Actions generated by the proposed method made the car wander slightly and zigzag along straight sections. However, the car is driven more quickly and naturally by the generated Macro-Actions than the user-defined Macro-Actions.

5. CONCLUSION

In this paper, we proposed a model for virtual characters to learn Macro-Actions and a policy by using PbD and reinforcement learning, respectively. The Macro-Action Generation Stage, which includes the Primitive Action Collector, Macro-Action Generator, and Macro-Action Selector, was defined by the process of learning Macro-Actions after collecting the primitive actions in a virtual environment. When learning the Macro-Actions, the Primitive Action Collector collects the primitive actions and transfers them to the Macro-Action Generator, which then defines the Macro-Action candidates by combining consecutive primitive actions. The Macro-Action Selector selects the representative Macro-Actions from among the Macro-Action candidates with the Maximin Selection algorithm. After determining the Macro-Actions, the primitive actions to be executed in the Macro-Action Control Stage are determined and transferred to a virtual character. The Macro-Action Executor selects the Macro-Actions with the highest Q-value in the present state based on the policies learned by the Policy Generator. The primitive actions that make up the selected Macro-Actions are executed in turn. When a virtual character learns, the Reward Collector collects the rewards from the Macro-Actions and transfers them to the Policy Generator to determine the next policy.

In the verification experiment, 40 Macro-Actions were generated with the proposed method and a virtual character was trained with the Monte Carlo method to execute the generated Macro-Actions. To analyze the 40 Macro-Actions that were generated, the running times were compared to those obtained with 40 Macro-Actions that were defined by a human subject. In accordance with the experiment results, the car that generated Macro-Actions with the proposed method was about 16% faster than one driven with human-defined Macro-Actions and about 10% faster than one driven by a human subject.

In the proposed method, the Macro-Actions and policy are not learned iteratively. Accordingly, the method could reduce the learning time required for a series of processes to generate Macro-Actions and to learn a policy. The Macro-Actions learned and generated with the proposed method can be used to reduce the learning time of reinforcement learning in complicated environments such as those in a game environment.

REFERENCES

[1] R. Tobi, Hierarchical Reinforcement Learning on the Virtual Battlefield. University of Amsterdam; 2007.

[2] F. S. Melo, and M. I. Ribeiro, "Q-learning with Linear Function Approximation," Proceedings of the 20th Annual Conference on Learning Theory, Sandiego, CA, June, 2007, Lecture Notes in Artificial Intelligence (LNAI), Vol.4539, 2007, pp.308-322.

[3] N. Ono, and K. Fukumoto, "Multi-agent Reinforcement Learning: A Modular Approach," Proceedings of the Second International Conference on Multiagent Systems, Kyoto, December, 1996, pp.252-258.

[4] A. Mcgovern, R. S. Sutton, and A. H. Fagg, "Roles of Macro-Actions in Accelerating Reinforcement Learning," Proceedings of Grace Hopper Celebration of Women in Computing, San Jose, CA, 1997, pp.13-18.

[5] T. Tateyama, S. Kawata, and T. Oguchi, "Automatic Generation of Macro-actions Using Genetic Algorithm for Reinforcement Learning," Proceedings of the 41st SICE Annual Conference, Osaka, August, 2002, Vol.1, pp.286-289.

[6] J. Randløv, Learning Macro-actions in Reinforcement Learning. University of Copenhagen; 1999.

[7] A. Cypher, Watch What I Do: Programming by Demonstration, MIT Press, 1993.

[8] N. Koenig, and M. J Matarić, "Behavior-based segmentation of demonstrated task," Proceeding of IEEE 10th International Conference on Development and Learning, Bloomington, May, 2006.

[9] Y. Sung, and K. Cho, "An Actions Generation Method of Virtual Character using Programming by Demonstration," Journal of Korea Game Society, Vol.11, No.2, 2011, pp.141-149.

[10] Y. Sung, K. Cho, and K. Um, "A Performance Improvement Technique for Nash Q-learning using Macro-Actions," Journal of Korea Multimedia Society, Vol.11, No.3, 2008, 353-363.

[11] R. Schoknecht, and M. Riedmiller, "Speeding-up Reinforcement Learning with Multi-step Actions," Proceedings of the Twelfth International Conference on Artificial Neural Networks, Aug, 2002, Lecture Notes in Computer Science (LNCS), Vol.2415, 2002, pp.813-818.

[12] M. J Matarić, Imitation in Animals and Artifacts : Sensory-Motor Primitives as a Basis for Imitation: Linking Perception to Action and Biology to Robotics, MIT Press, 2000, pp.391-422.

[13] M. N. Nicolescu, and M. J Matarić, "Natural Methods for Robot Task Learning: Instructive Demonstrations, Generalization and Practice," Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems, Melbourne, July, 2003, pp.241-248.

Yunsick Sung

He received his B.S. degree in division of electrical and computer engineering from Pusan National University, Busan, Korea, in 2004, his M.S. degree in computer engineering from Dongguk University, Seoul, Korea, in 2006., and his Ph.D. degree in game engineering from Dongguk University, Seoul, Korea, in 2012. He was employed as a member of the researcher at Samsung Electronics between 2006 and 2009. He was the plural professor at Shinheung College in 2009 and at Dongguk University in 2010. His main research interests are many topics in brain-computer Interface, programming by demonstration, ubiquitous computing and reinforcement learning.

Kyungeun Cho

She received her B. Eng. Degree in Computer Science in 1993, her M. Eng. and her Dr. Eng. Degrees in Computer Engineering in 1995 and 2001, respectively, all from Dongguk University, Seoul, Korea. Since September 2003, she has been a Professor at Department of Multimedia Engineering, Dongguk University, Seoul, Korea. During 1997-1998 she was a research assistant at the Institute for Social Medicine at the Regensburg University, Germany, and a visiting researcher at the FORWISS Institute at TU-Muenchen University, Germany. Her current research interests are focused on the areas of intelligence of robot and virtual characters and real-time computer graphics technologies. She has led a number of projects on robotics and game engines and also has published many technical papers in these areas.