Action Acquisition with Memory Reinforcement Learning using Prior Knowledge

Anonymous submission

***Abstract*— It is difficult for humans to obtain new knowledge, but we can acquire new knowledge by imitating others behaviors. Inspired by such human characteristics, we propose a deep reinforcement learning method called *memory reinforcement learning*. Our approach leverages the technique of experience replay and a replay buffer architecture. We manually create stable action transition sequences (prior knowledge) and store these transitions in the replay buffer at the beginning of training. These hand-crafted transition sequences enable us to avoid random action selections and a local optimal policy. Consequently, our method can acquire stable control efficiently. Experimental results on a lane-changing task of autonomous driving indicate that the proposed method can efficiently acquire stable control.**

I. Introduction

Humans have the ability to obtain new knowledge and skills by using past experiences and practice with many failures. For instance, when we practice riding a bike, we fall off our bikes at the beginning and accumulate experiences. After leveraging those experiences, we finally acquire the skills to ride a bike. In the reinforcement learning community, we aim to model such human ability and embed it into any autonomous system.

Reinforcement learning has been studied over several decades. In recent years, thanks to the rapid development of deep neural networks, various deep reinforcement learning methods have been proposed [1]–[7]. Unlike conventional (i.e., non-deep) reinforcement learning methods, deep reinforcement learning methods can deal with tasks of higher dimensional state space, which enables us to solve more complex tasks such as playing Go [8], controlling autonomous system [9]–[11], and grasping objects by using hand manipulators [12].

Although deep reinforcement learning methods have exhibited higher performance in various application fields, there are two major problems making it difficult to acquiring desirable controls. One problem is the difficulty of updating network parameters. For example, with a reinforcement learning method using a neural network such as a deep Q network (DQN) [1], [2], an agent observes the current state and inputs this state into a network. As an output of the network, a control is obtained. In this case, by using a multi-layered network, practical controls on complex and higher dimensional environments can be obtained. However, such a deep network causes learning difficulty due to the increase in the number of parameters and vanishing/exploding gradients.

The other problem is derived from the characteristics of reinforcement learning. At the beginning of training, we randomly initialize parameters in a network and start to train. This is equivalent to training from random action selections. When acquiring time series actions, it may be difficult to acquire stable controls because a current action selection depends on the past agent's behavior. Experience replay [13] has been proposed to address this issue. In experience replay, experienced action transitions are stored in a replay buffer. Then, the stored transitions are randomly sampled and used for training. However, learning from random initial values accumulates action selections that do not contribute to learning, so it requires a large amount of time to select an optimal control. Furthermore, if we apply deep reinforcement learning on a large-scale state space, a solution easily falls into a local optimum. Therefore, it is difficult to obtain the desired action.

To avoid the above two problems and achieve higher performance, transfer learning is widely used in computer vision. In transfer learning, a convolutional neural network (CNN) is trained using a large-scale image dataset for general object recognition tasks such as ImageNet [14]. Then, we use the trained parameters as initial parameters and further update the parameters for another task. Transfer learning is applied on the assumption that features extracted from an image should be similar for different tasks. However, introducing transfer learning into a reinforcement learning framework is difficult because a state space and input significantly differ for each task.

In case of humans, pre-train model can be considered as knowledges of other persons. There are two ways which a human obtains new knowledge or skill. One is repeating trial and error on their own, which would require much time to acquire knowledges. This is corresponding to the conventional reinforcement approach. The other is imitating the behaviors of other persons. In this way, prior knowledges through other persons would be an important clue to acquire new knowledges efficiently.

Regarding humans, a pre-train model can be considered as knowledge of other people. There are two ways that a human obtains new knowledge or skills. One is through trial and error, which requires much time to acquire knowledge. This corresponds to the conventional reinforcement approach. The other is imitating the behaviors of other people. In this way, prior knowledge from other people can be important in acquiring new knowledge efficiently.

We propose a reinforcement learning method, called *memory reinforcement learning*, to efficiently acquire stable controls. We introduce the concept of prior knowledge mentioned above into a reinforcement learning architecture. Our approach leverages experience replay and a replay buffer.

We manually create action transitions representing stable human-like behaviors, which we call *prior knowledge*. We store prior knowledge into a replay buffer before the beginning of training. During the training, prior knowledge is randomly used for training by using experience replay. Consequently, our method enables us to acquire stable controls from the early stage of training. To seek an effective way to acquire stable controls, we introduce two approaches for storing hand-crafted transitions. Experimental results by using an autonomous driving simulator for lane-changing tasks demonstrated that our method can efficiently acquire stable vehicle controls.

II. Related work

Deep reinforcement learning methods to efficiently acquire stable controls have been proposed. Peng *et al.* [15] proposed a hierarchical reinforcement learning method to obtain bipedal locomotion skills. They defined two control systems: high- and low-level controllers. A high-level controller finds long-term controls to reach a destination and a low-level controller controls local walking motions. In contrast, our method leverages hand-crafted action transitions at the beginning of training and acquires stable controls.

A memory network [16] has been introduced in deep reinforcement learning architecture to deal with a higher-dimensional state space and leverage past behaviors of an agent efficiently. Parisotto *et al.* [17] proposed Neural Map, which is a structured memory designed for reinforcement learning agents in 3D environments. The adaptive write operation and a bias for the writing operation suppress the computational cost for writing past behaviors into an external memory and required the external memory size. Thanks to the external memory, this method enables us to store past long-term agent behaviors in a three-dimensional environment. However, our method stores hand-crafted stable action transitions instead of past behaviors.

Price *et al.* [18] introduced imitation learning [19] as a policy obtained in advance. They roughly estimate desirable behaviors (i.e., a policy) from human motions by using an imitation learning framework then integrate with a standard reinforcement learning framework. This improves performance and convergence. However, obtaining stable behavior of an agent through imitation learning is also challenging. If we cannot obtain desirable agent behavior in the imitation learning step, the next reinforcement learning step will be difficult. Our method involves stable action transition as prior knowledge and enables us to avoid unstable action acquisition.

III. Proposed method

Deep learning has attracted attention for various tasks of automatic driving. For instance, semantic segmentation recognizes traffic lanes by considering the context in the image, and object detection determines the possibility of understanding the position of the pedestrian and other vehicles. A mechanism for determining the action of the vehicle is also necessary to move in a complex environment without any traffic accidents. A classical approach is a rule-based method that is based on human knowledge, but it is difficult to describe controls for all possible cases in a complicated environment where other vehicles are driving. Therefore, we assume a lane-changing task in an environment in which several other vehicles are running and use our deep reinforcement learning method to acquire stable control efficiently. Deep reinforcement learning makes it possible to acquire an action even when it is difficult to define complex environments and number of states that are difficult to describe in rules. To use the results of the aforementioned computer vision techniques in a reinforcement learning framework, we adopt a bird's-eye view image to describe the surrounding environment as an input (see Figure 1). Bird's-eye view images can be obtained using semantic segmentation, distance estimation, and object detection for on-vehicle camera images and Lidar.

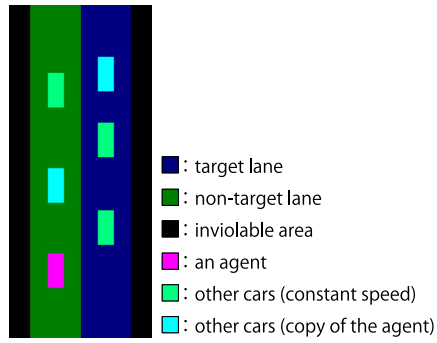


Fig. 1. Example of our autonomous driving simulator. In this environment, an agent tries to change lane from left lane to right lane.

A deep reinforcement learning approach based on Q-learning [20] inputs an image and outputs the value of each selectable action in that scene. By taking the highest value action, an agent should be close to a goal. To select better actions to reach a goal, a reward is an important cue, and we train an agent to obtain a higher reward. An action-value function (i.e., Q-function) is approximated as operations of a CNN, which can be defined as $Q(s, a; \theta_i)$, where s is a state (i.e., an observation), a is an action, and θ_i are network parameters at the ith training iteration. During training, the θ_i are updated using a loss function. Let r be a reward value to be obtained by the selected a and s' be the next state of s. In the case of a DQN [1], given an action transition $e = (s, a, r, s')$, the loss function $L_i(\theta_i)$ is defined using a temporal-difference error [21] as

$$L_i(\theta_i) = \mathbb{E}\left[\frac{1}{2}\left(r+\gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\right)^2\right], \tag{1}$$

where a' is an action at s' and γ is a discount factor. The θ_i are updated using the following gradient of the loss function:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}\left[\left(r+\gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\right)\nabla_{\theta_i} Q(s, a; \theta_i)\right]. \tag{2}$$

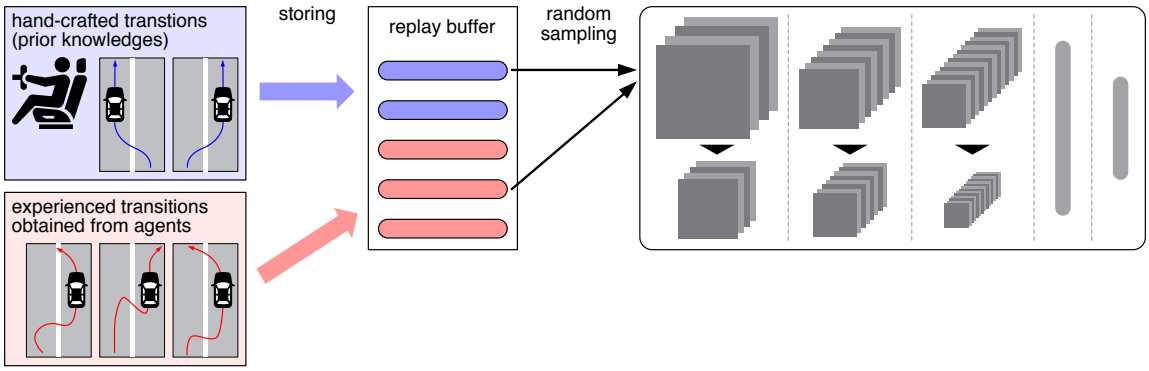


Fig. 2. The structure of a replay buffer with prior knowledge A. Blue cells in a replay buffer are stable action transitions manually created by human (i.e., prior knowledge) and red cells are action transitions obtained from past agent behaviors. In prior knowledge A, we store the hand-crafted transitions in part of replay buffer and the experienced transitions are stored in the remaining buffer. Because there stored transitions are randomly selected during training and suppress random action selections, an agent can introduce prior knowledge to improve controls throughout training.

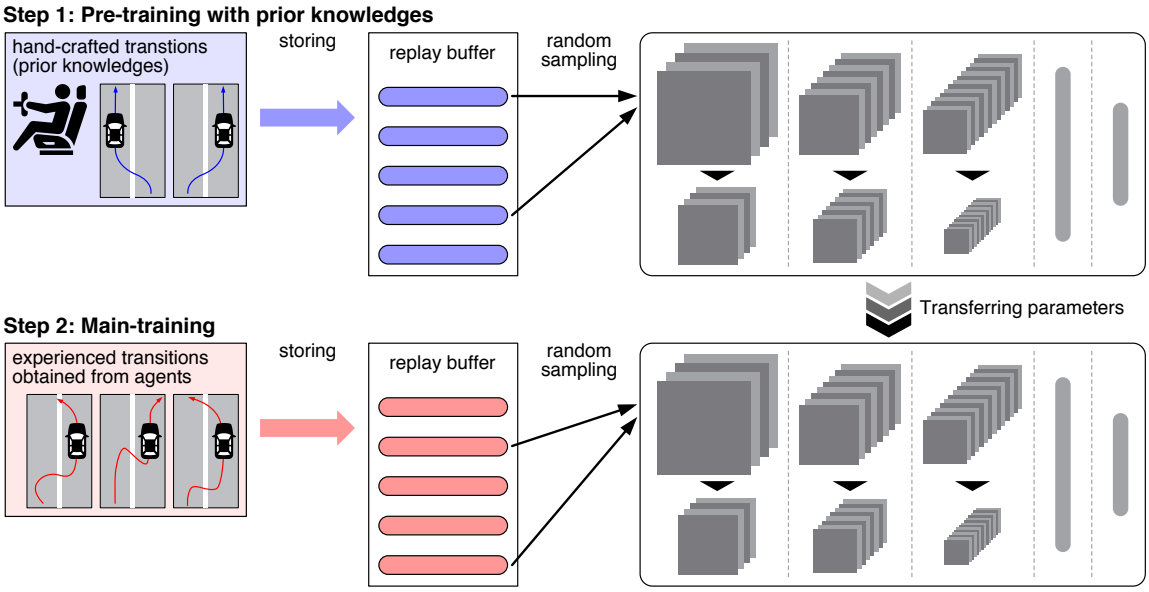


Fig. 3. The structure of a replay buffer with prior knowledge B. Blue cells in a replay buffer are stable action transitions manually created by human (i.e., prior knowledge) and red cells are action transitions obtained from past agent behaviors. Training with prior knowledge B consists of two steps. At the first step, we fill a replay buffer with the hand-crafted transitions and train an agent with the replay buffer. At the second step, we use the network parameters obtained in the first step as initial parameters and we further trains the agent with a regular deep reinforcement learning. This approach acquires the same vehicle control as human and improves the controls.

For a stable gradient update, *experience replay* is frequently used. In experience replay, a set of action transitions $D = \{e_1, \ldots, e_n\}$ is stored into a replay buffer. Then, transitions are randomly selected from the replay buffer and used for training. A replay buffer is empty at the beginning of training, and we first store action transitions in the replay buffer then start training. Because most of these stored transitions contain random actions in the early state of training, they require a large amount of time to select behaviors that would contribute to obtaining stable controls.

We, therefore, our method uses prior knowledge acquired by humans, and call the method *memory reinforcement learning*. With the proposed method, we create action transitions that represent human behaviors. Specifically, we record the states and actions when people control an agent vehicle. These manually created action transitions implicitly contain knowledge and skills that humans have acquired. Hence, we call such a transition *prior knowledge*. Such prior knowledge is stored in a replay buffer and used for training. The behaviors of prior knowledge are more stable than those of an agent at the beginning of training. Moreover, prior knowledge might be ideal behaviors that we want an agent to acquire. Because such desirable behaviors are used for training, it enables us to learn agents by suppressing random action selections. To use prior knowledge efficiently, we use the two approaches introduced in the previous section.

A. Prior knowledge A

Figure 2 shows the structure of a replay buffer with prior knowledge A. This approach constantly stores prior knowledge in 10% of a replay buffer. In the remaining replay buffer, we store past behaviors of an agent, as with conven-

tional experience replay. In this approach, prior knowledge stored in a replay buffer is constantly used during training. The prior knowledge suppresses random action selection that does not contribute to obtaining stable controls. Action transitions of the past agent behaviors are also randomly used for training. These explores new actions to acquire better controls.

B. Prior knowledge B

Figure 3 shows the structure of a replay buffer with prior knowledge B. This approach consists of two training steps. At the first step, we store prior knowledge in the entire replay buffer and train a network with the filled replay buffer until the behaviors approximately converge into stable controls. Then, we use the network parameters obtained in the first training step as initial network parameters of the second training step. At the second step, we apply conventional training.

In this approach, we obtain the same controls as prior knowledge at the first training step. Then, at the second training step, we further improve the network parameters to obtain more stable controls.

IV. Experimental results

To evaluate our method, we adopted a lane-changing task of an autonomous driving simulator. In this task, an agent (i.e., an autonomous car) moves from the current traffic lane to the neighboring traffic lane. Because other cars exist in this environment, the agent needs to change the lane while avoiding collisions. Moreover, there is no clear goal location in this task, and it is necessary to acquire long-term controls. Consequently, in this lane-changing task, it is difficult to acquire optimal and stable vehicle controls without prior knowledge.

A. Experimental settings

Figure 1 shows an example of a driving simulator of a lane-changing task. During training, the number of other cars is set as 5. In the experiment, we randomly set the number of other cars from 0 to 5. We defined two behaviors for the other cars. One was going straight ahead at constant speed, which is shown with the green car in Figure 1. The other is copying the past trained agents behaviors, which is shown with the light blue car in the figure. The reason we used two types of cars is that using only constant-speed cars might be too easy. In a practical environment, the behaviors of cars differ depending on their drivers driving skills. Our simulator takes into account such a realistic environment and uses the past trained agents behaviors.

As an input for a network, we trimmed 84×84 pixels centering at the agent and converted the trimmed patches to grayscale. We inputted four successive gray scale patches by channel-wise concatenation, as shown in Figure 4, to consider the continuous movement of an agent. Actions that an agent can take consist of combinations of speed and curvature. We define speed as ± 0, $+0.1$, and -0.1 m/s and define curvature as 0, $+0.01$, -0.01 rad/m. Hence, an agent takes nine different actions. These are controllable values for actual cars in the real world.

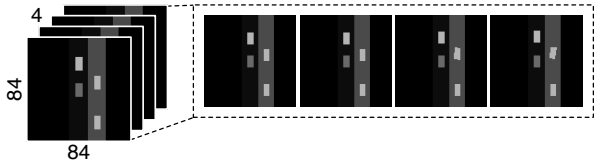


Fig. 4. Examples of input data. 84×84 pixels centered at an agent are trimmed from the autonomous driving simulator shown in Figure 1. Then, the trimmed patches are converted to grayscale patches. In our experiments, we use the grayscale patches of the last four frames as inputs by concatenating channel-wisely.

TABLE I

The detailed architecture of CNN.

	processing	details
input	size	$84 \times 84 \times 4$
conv1	activation func.	ReLU
	filter size	$3 \times 3 \times 16$
	maxpooling	2×2
conv2	activation func.	ReLU
	filter size	$3 \times 3 \times 16$
	maxpooling	2×2
conv3	activation func.	ReLU
	filter size	$3 \times 3 \times 16$
	maxpooling	2×2
fc	activation func.	ReLU
	size	256
output	size	9

TABLE II

Reward setting and learning details

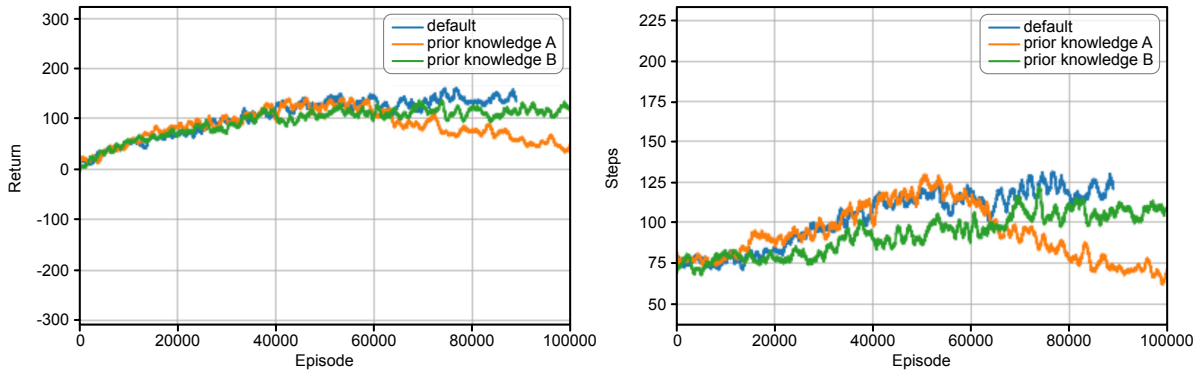
Reward	collision with other cars or wall: -5 center of a lane: +1 stop: -1 target lane: +1 non-target lane: -2
Termination criterion	collision or 300 action steps
# of training episodes	100,000 episodes
Discount factor γ	0.95
Exploration	Linear decay epsilon greedy
Replay buffer size	$1e+5$

As a deep reinforcement learning architecture, we adopted a double DQN [3]. Table I lists the detailed network architecture of the CNN we used and Table II shows the reward setting and learning details.

B. Baselines and evaluation metrics

We compared our method (with prior knowledge A and B) with the following two baselines. One was *Default*, which trained an agent by using a naïve double DQN. For fair comparison, we used the same network architecture as for the proposed method (see Table I). The other was *Human*, which involved several people manually controlling an agent car. In this experiment, six people controlled an agent to change the lane. Each person controlled an agent 100 times.

We used the following two scores as evaluation metrics. The first was *survival score*, which is the number of episodes in which an agent did not collide with other cars or obstacles during 300 action steps. The second was *goal score*, which is



(a) Mean return

(b) Mean survival steps

Fig. 5. Mean return and survival steps at each training episode. Horizontal axis shows the number of training episodes and vertical axis shows (a) mean return value and (b) mean survival steps.

TABLE III

SURVIVAL AND GOAL SCORES

	initial lane	survival score	goal score
default	random	53	50
	non-goal	33	29
prior knowledge A	random	49	37
	non-goal	37	10
prior knowledge B	random	**67**	**66**
	non-goal	**51**	**48**
human	random	50.7	50.3

the number of episodes an agent successfully changed into the target lane within 300 action steps. Each method was tested for 100 episodes, that is, 100 is the best score and 0 is the worst score. Because each person controlled 100 episodes with regards to Human, the scores were averages over all six people.

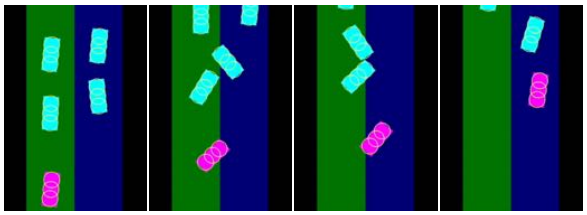
C. Results

Figure 5 shows the mean returns and mean survival steps during training. Here, *survival steps* are the action steps until collision with other cars or obstacles. Note that the survival step is different to the survival score introduced in the previous section. For our method with prior knowledge A, both the mean return and mean survival steps decreased from 50,000 episodes. At the end of training (i.e., 100,000 episode), our method with prior knowledge A had the lowest values compared with the other methods. Our method with prior knowledge B successfully trained without decreasing those values. However, Default, which does not use prior knowledge, achieved the highest return and number of survival steps.

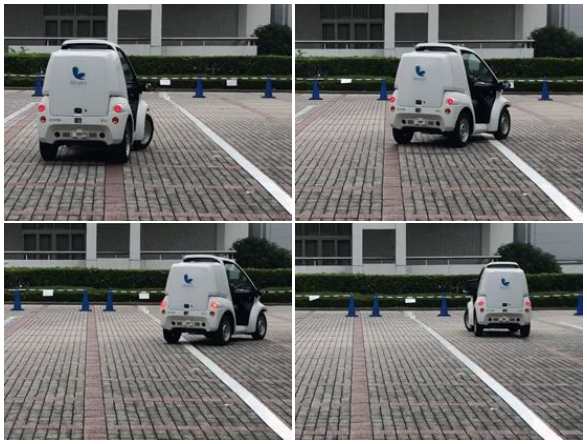
Table III shows the evaluation scores for each training method. Initial lane in this table shows the location of an agent at the beginning of an episode. *Random* means that we randomly set the initial lane as the target or non-target lane. *Non-goal* means that we set an agent to a non-target lane for all episodes. Although Default achieved better results during training, as shown in Figure 5, its evaluation scores were lower than those of our method with prior knowledge B. This means that training without prior knowledge plunges into local optimum solution, and Default acquired unstable controls. The survival scores of prior knowledge A is at the same level as default, while the goal score becomes lower than default. In prior knowledge A, the hand-crafted transitions are always stored in the part of a replay buffer and these transitions are constantly used during the training. Therefore, the training relies on the prior knowledges and an agent does not explore new actions. The scores of our method with prior knowledge B were the highest. For our method with prior knowledge B, we first trained a network with only the hand-crafted transitions and found a more optimal solution by using action transitions obtained from past agent behaviors. Hence, agent behavior that should be learned from prior knowledge is already acquired in the first training step. At the second training step, apart from the controls acquired from prior knowledge, exploring new actions enables us to acquire better vehicle controls. Moreover, our method with prior knowledge B outperformed Human. Consequently, our method acquired stable controls from prior knowledge and acquired better actions suitable for the lane-changing task. Our method with prior knowledge B was especially efficient for leveraging human prior knowledge.

D. Operation result in a real environment

We conducted a running test using an actual vehicle in the real world. The action of a vehicle is determined by a trained agent with prior knowledge B. Given the speed and curvature of a vehicle in a driving simulator, the degrees of accelerator and steering of an actual vehicle are determined by Proportional-Integral-Differential (PID) control to reproduce the same vehicle control as the driving simulator. The parameters of PID control were experimentally set as $P = 2000$, $I = 100$, and $D = 0$. We set them in a real environment so that the distance between cars and lane distance correspond to those in the simulator environment. Figure 6 shows snapshots of the running test. In this running test, there are no other cars to avoid accidents. However, we assumed that there are several cars in the target lane and that an agent moves from the non-target lane to the target lane. As a result, the vehicle successfully changed lanes without



(a) Example of lane change in simulator by agent



(b) Example of lane change in real world by agent

Fig. 6. Snapshots of running test using actual vehicle in real world

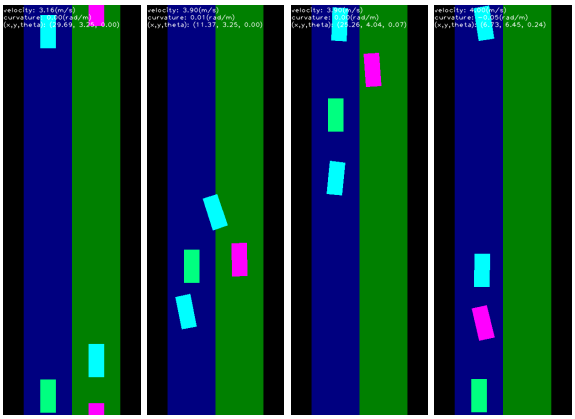
any collisions or overshooting of the target lane.

E. Discussion

We give examples of lane-changing results for our method with prior knowledge B in Figure 7 for qualitative evaluation. As shown in Figure 7(a), an agent changed the lane from right to left in an environment with three other cars. In this situation, the agent did not accelerate and followed the car in front of it from initial to step 30. After the other car changed the lane at step 65, the agent smoothly moved to the large space between other cars at step 95. As shown in Figure 7(b), there were three other cars in the target lane from initial to step 70. After the agent passed the other cars and ensured the safety margins, the agent changed the lane. These results indicate that an agent trained with the proposed method takes the same actions as a human and that the proposed method can obtain stable controls with prior knowledge.

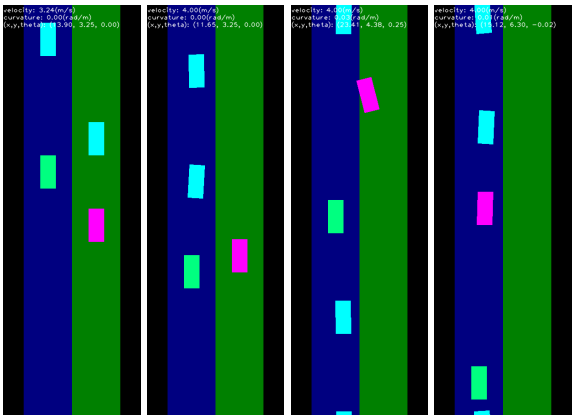
V. Conclusion

We proposed a deep reinforcement learning method, called memory reinforcement learning, to acquire stable controls efficiently. Our method manually creates action transitions, i.e., prior knowledge, that represent actual human behaviors, which we store into a replay buffer. Using the stored prior knowledge suppresses random action selection and enables us to acquire stable controls from the early stage of training. To acquire more stable controls, we introduced two approaches to use the prior knowledge. In the experiments involving a lane-changing task of an autonomous driving



Initial step 30 step 65 step 95 step

(a) An example to change lane after following another car



Initial step 70 step 175 step 230 step

(b) An example to change lane with passing another car

Fig. 7. Examples of lane-changing results for our method with prior knowledge B. Details of colored objects are given in Figure 1. Each row shows one trial of lane changing. The number of action steps corresponding to each image is shown under that image.

simulator, our method outperformed conventional deep reinforcement learning methods without prior knowledge. Moreover, the proposed method, with which we train a network with only prior knowledge then use the trained parameters for regular reinforcement learning, performed the best. In a running test using an actual vehicle in the real world, the vehicle successfully changed lanes with our method.

Our future work includes seeking more effective approaches to use prior knowledge and a reward design by considering such knowledge.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," in *NIPS Workshop on Deep Learning*, 2013.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 02 2015.

[3] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *National Conference on Artificial Intelligence*, 2016.

[4] Z. Wang, N. de Freitas, and M. Lanctot, "Dueling network architectures for deep reinforcement learning," in *International Conference on Machine Learning*, 2016.

[5] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. D. Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, S. Legg, V. Mnih, K. Kavukcuoglu, and D. Silver, "Massively parallel methods for deep reinforcement learning," in *ICML workshop on Deep Learning*, 2015.

[6] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learningr*, 2016.

[7] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," in *International Conference on Learning Representations*, 2017.

[8] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 01 2016.

[9] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *International Conference on Learning Representations*, 2016.

[10] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in *International Conference on Robotics and Automation*, May 2017, pp. 285–292.

[11] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," in *International Conference on Intelligent Robots and Systems*, Sept 2017, pp. 1343–1350.

[12] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," in *International Symposium on Experimental Robotics*, 2016.

[13] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, no. 3, pp. 293–321, May 1992.

[14] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition*, June 2009, pp. 248–255.

[15] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Transactions on Graphics*, vol. 36, no. 4, 2017.

[16] S. Sukhbaatar, a. szlam, J. Weston, and R. Fergus, "End-to-end memory networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 2440–2448.

[17] E. Parisotto and R. Salakhutdinov, "Neural map: Structured memory for deep reinforcement learning," in *International Conference on Learning Representations*, 2018.

[18] B. Price and C. Boutilier, "Implicit imitation in multiagent reinforcement learning," in *International Conference on Machine Learning*, 1999.

[19] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys*, vol. 50, no. 2, 2017.

[20] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992.

[21] G. Tesauro, "Temporal difference learning and td-gammon," *Commun. ACM*, vol. 38, no. 3, pp. 58–68, Mar. 1995.