Navigation Behavior based on Spatial Representation Self-organized in Hierarchical Recurrent Neural Networks

Wataru Noguchi, Hiroyuki Iizuka and Masahito Yamamoto

Abstract— **Cognitive map is an internal model of external world, which has the positional representation of the space. Animals can navigate to the desired destination using the cognitive map while dealing with environmental uncertainty. We performed a mobile robot navigation experiment where obstacles are randomly placed using hierarchical recurrent neural network (HRNN) with multiple time-scales. HRNN was trained to navigate the mobile robot to the destination indicated by a snapshot image. As a result of the training, HRNN became able to successfully avoid the obstacles and navigate to the destination from any positions in the environment. Moreover, HRNN performed shortcut behavior. Analysis of the internal states of HRNN showed that module with fast time-scale handles obstacle avoidance and slow RNN controls goal-directed behavior using self-organized cognitive map-like representation. It indicates that training of goal-oriented navigation, namely the navigation motivated by the snapshot image of destination, results in the self-organization of cognitive map-like representation.**

I. Introduction

Animals perform navigation behaviors in natural environment. They have an internal model of the spatial representations of the external world as spatial representation, which is called a cognitive map [1]. They can navigate using the cognitive map, and it is considered that they remember not action sequences to destinations but where the destinations are in a space. That is why the rat can go to the food place by finding a new path to the memorized food position even if the trained familiar path is no longer available. In such navigation behaviors, animals have to deal with at least two different time scales. One is a fast time scale where they need to react quickly to the facing environmental events like wall avoidance and another is a slow time scale where they choose actions to achieve destinations. Actually, it is considered that these time scales are implemented in a hierarchical structure in mammals' brain [2].

The navigation ability has been modeled in various ways, and more complex ability can be achieved using advanced deep learning approach. A deep neural network that consists of recurrent neural network (RNN) and convolutional neural network (CNN) is trained to navigate in complex environment [3]. The model recognizes visual inputs with CNN and effectively navigates in the environment using memory stored in RNN. By modeling the navigation ability using the deep neural networks, no predefined function are needed for performing navigation and we can simulate the developmental process of the ability. However, the investigation about how the obtained function works and how such navigation ability was realized were not focused in their studies.

Graduate School of Information Science and Technology, Hokkaido University, Japan `noguchi@complex.ist.hokudai.ac.jp`

Paine and Tani simulated the self-organization of the hierarchy of time scales in a RNN [4]. The RNN was used as a robot controller for solving a navigation task in a simple maze. The weights of the RNN were optimized using genetic algorithm, and the wall avoidance and top-down control of goal-directed motion sequences were self-organized. Such a functional hierarchy of fast and slow time scales enabled the robot to navigate desired goals in the maze. However, because the navigation task starts from a fixed home position, cognitive map-based navigation was not required. The task can be achieved just by remembering the sequences of actions.

In this study, we show the navigation ability based on a self-organized spatial representation can be realized by the RNN with a hierarchical structure. We simulated the robot navigation in the environment where the staring position are not fixed and the obstacles are randomly placed. Furthermore, the robot can only receive subjective visuomotor inputs and objective information such as the position of robot, destination or obstacles are never provided. In such navigation task, the robot cannot solve the navigation task by remembering the sequences of actions. The hierarchical RNN (HRNN) are trained as the controller of the robot and it is shown that spatial representation is self-organized in the HRNN. The details of experiment are described following sections.

II. Navigation Model and Environment

In this study, a mobile robot performs navigation behaviors toward the destinations indicated by a snapshot image in a dynamic environment where the trained path is not necessarily available. The robot is required to obtain spatial representation from subjective visuo-motor experiences for the navigation. The robot equips a hierarchical recurrent neural network as a controller and we investigate how the hierarchical structure controls goal-directed behaviors while dealing with the dynamic environment.

A. Robot and Environment

The simulation environment is a square arena surrounded by the textured walls and it has grid coordinates. The size of arena is 20×20 grid cells. The walls at corners have characteristic textures, which can work as landmarks. The floor is also textured. In the arena, there are five objects as obstacles in addition to walls. These obstacles have the same texture. Two of them are static obstacles whose positions do not change, and the others are dynamic obstacles whose positions change in each trial of the navigation. The

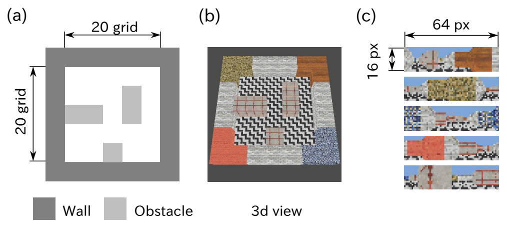

Fig. 1. (a) Overview of the environment. (b) 3d overview of the environment. (c) Examples of visual images captured by robot's camera.

simulated environment is organized as shown in Fig. 1. The static obstacles are placed between upper-left and lower-left corners and the straight path between them are unavailable.

The mobile robot is modeled as an agent that could move around the environment. The agent, which is equipped with an omnidirectional camera, is able to obtain an omnidirectional view while moving around the arena. The agent can move to adjacent 8 cells in one time step. The omnidirectional camera is used as a capturing system that consists of four cameras that cover the entire view around the agent. Each camera faced in a different direction (north, south, east, or west). The size of the captured visual image for each camera is 16×16, and each pixel of the image has three channels (RGB). Four captured images are used as a single image by combining them horizontally, and the size of the visual image inputs are then 64×16 with the three channels included. When capturing the images, the positions of cameras are perturbed by Gaussian noise with mean of zero and standard deviation of 0.1 where size of single grid cell is unit.

B. Navigation Task

The tasks for the agent are achieving the destination points indicated by the snapshot images captured at the destination points. The tasks are performed as described below. First, the agent freely moves around the arena as warm-up phase. The snapshot images for the destination point are given at a certain time abruptly. Once the agent received the snapshot image of the destination points, the navigation task starts and the agent is expected to generate the motion sequences to the destinations. The snapshot image is given once at the beginning of the navigation task. Moreover, the positions of three obstacles are changed every navigation trial as described above, and the agent has to avoid the obstacles for achieving destinations. Thus, the agent has to consider how to get to the destinations and what action should be taken to approach without stucking in obstacles. The overview of the navigation task is shown in Fig. 2.

III. Controller and Training

In this section, we build a controller for the agent to achieve the navigation task. In order to deal with functions with different time scales, namely interacting with the environment as fast time scale and generating goal-directed flow as slow time scale, the controller implements an explicit mechanism of multiple time scales. The controller

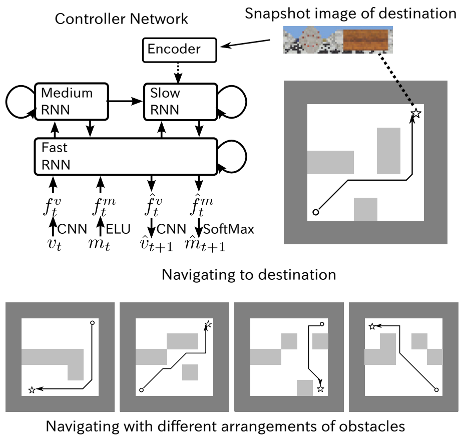

Fig. 2. The navigation task. Top: The controller network receives the photographs and navigates the robot in the environment using predictive motion output. Bottom: In the navigation task, the placement of obstacles, starting position, the destination are changed in each trial of navigation.

is a hierarchical recurrent neural network with multiple-time-scales.

Our model receives current vision v_t and motion m_t at each time step and predicts v_{t+1} and m_{t+1} as outputs. Objective information, such as spatial coordinates, is never provided. Our model is trained to predict the visuomotor sequences in teaching sequences and learns navigation in imitation manner. The motion output m_{t+1} is choice probability of each 8 actions and the agent choices the action of most highest probability. We describe the details about the controller bellow.

A. Hierarchical Recurrent Neural Network

Our model consists of three recurrent layers with multiple time scales and input/output processing layers for vision and motion. A schematic view of our model is shown in Fig. 2 (top-left). The model has a similar architecture to the model we previously proposed [5]. The three recurrent layers comprise fast, medium and slow RNNs according to the time scales. The fast RNN interacts with visuomotor sensory inputs and generates visuomotor outputs. The medium RNN interacts with fast RNNs and can extract the higher-level features of the visuomotor inputs. The slow RNN controls the generation flow of the fast RNN by setting the initial internal states using a snapshot image, as described later. The input and output layers are used for extracting visual and motion features and generating prediction of vision and motion. For the visual inputs and outputs, the CNN is used and our model can deal with complex visual images [6].

A RNN is an artificial neural network that has recurrent connections and can use historical information as it receives previous output vectors of itself, which are called an internal states. To implement the different time scales in this study, we used a Continuous-Time RNN (CTRNN) [7], [8]. A CTRNN has a time constant parameter τ, which determines the time scale. The potential of neurons $\boldsymbol{u}_t$ and the output

$\boldsymbol{h}_t$ of CTRNN at time t are calculated as follows:

$$\boldsymbol{u}_t = \left(1-\frac{1}{\tau}\right)\boldsymbol{u}_{t-1} + \frac{1}{\tau}\left(\mathbf{W_x}\boldsymbol{x_t} + \mathbf{W_h}\boldsymbol{h_{t-1}} + \boldsymbol{b}\right) \quad (1)$$
$$\boldsymbol{h}_t = \tanh(\boldsymbol{u}_t). \quad (2)$$

where $\mathbf{W_x}$, $\mathbf{W_h}$, and $\boldsymbol{b}$ are the connection weight matrix from the current input vector $\boldsymbol{x_t}$, the recurrent connection weight matrix from a previous output vector $\boldsymbol{h_{t-1}}$ (internal states) and bias, respectively. As indicated in (1), the larger τ is, the more slowly the potential $\boldsymbol{u}_t$ changes.

When the functions of the fast, medium and slow RNNs (modeled by CTRNNs), are expressed as $\mathrm{RNN^F}$, $\mathrm{RNN^M}$, and $\mathrm{RNN^S}$, respectively, the equations of these three layers in one-step processing is expressed as follows:

$$\boldsymbol{h}^F_t = \mathrm{RNN^F}((\boldsymbol{f}^v_t, \boldsymbol{f}^m_t, \boldsymbol{h}^M_{t-1}, \boldsymbol{h}^S_{t-1}), \boldsymbol{u}^F_{t-1}, \boldsymbol{h}^F_{t-1}), \quad (3)$$
$$\boldsymbol{h}^M_t = \mathrm{RNN^M}(\boldsymbol{h}^F_{t-1}, \boldsymbol{u}^M_{t-1}, \boldsymbol{h}^M_{t-1}), \quad (4)$$
$$\boldsymbol{h}^S_t = \mathrm{RNN^S}((\boldsymbol{h}^F_{t-1}, \boldsymbol{h}^M_{t-1}), \boldsymbol{u}^S_{t-1}, \boldsymbol{h}^S_{t-1}), \quad (5)$$

where $\boldsymbol{f}^v_t$ and $\boldsymbol{f}^m_t$ are the features of the visual $\boldsymbol{v}_t$ and motion $\boldsymbol{m}_t$ inputs, respectively. The first arguments of all RNN's functions are used as $\boldsymbol{x_t}$ in (1). The visual feature $\boldsymbol{f}^v_t$ and motion feature $\boldsymbol{f}^m_t$ are calculated as follows:

$$\boldsymbol{f}^v_t = \mathrm{CNN}^{rec}(\boldsymbol{v}_t), \quad (6)$$
$$\boldsymbol{f}^m_t = \boldsymbol{\phi}(\mathbf{W}_m\boldsymbol{m}_t + \boldsymbol{b}_m), \quad (7)$$

where CNN^{rec} is a CNN visual image recognizer, $\boldsymbol{\phi}$ is the activation function, and $\mathbf{W}_m$ and $\boldsymbol{b}_m$ are the weights matrix and bias vector, respectively. After the processing of the RNNs, the decoded features $\hat{\boldsymbol{f}}^v_t$ and $\hat{\boldsymbol{f}}^m_t$ are calculated in the same form as (7), but different weights and biases are used and $\boldsymbol{m}_t$ in (7) is replaced by $\boldsymbol{h}^F_t$. The visual output $\hat{\boldsymbol{v}}_{t+1}$ and motion output $\hat{\boldsymbol{m}}_{t+1}$ are generated as follows using $\hat{\boldsymbol{f}}^v_t$ and $\hat{\boldsymbol{f}}^m_t$:

$$\hat{\boldsymbol{v}}_{t+1} = \mathrm{CNN}^{gen}(\hat{\boldsymbol{f}}^v_t), \quad (8)$$
$$\hat{\boldsymbol{m}}_{t+1} = \mathrm{SoftMax}(\hat{\mathbf{W}}_m\hat{\boldsymbol{f}}^m_t + \hat{\boldsymbol{b}}_m), \quad (9)$$

where $\mathrm{CNN^{gen}}$ is the deconvolution of the CNN as a visual image generator [9]. SoftMax is the softmax function that normalizes the output vector and get probabilities of each discrete actions. We used an exponential linear unit (ELU) [10] as an activation function $\boldsymbol{\phi}$ for extracting features. The equation for ELU is the following:

$$\mathrm{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\ (\exp(x) - 1) & \text{if } x \le 0 \end{cases}, \quad (10)$$

where α is the parameter controlling the saturation value for the negative potentials. This activation characteristic of an ELU allows the recognized features to have useful sparse representation. α is set to 1.0 in our model.

B. Optimizing the initial states of a slow RNN

Our model has to generate the predicted motion sequences to visit the destination indicated by the snapshot images. Because the slow RNN has a slow time scale and does not directly connect raw-level input or output, it can deal with long-term dynamics, such as motion generation as a top-down process. It has been shown that generating goal-directed behaviors can be realized by optimizing the initial states of the RNN with a slow time scale with a genetic algorithm or backpropagation [4], [11]. In our previous model, the initial states are set by a vector value that is transformed from images of the destination by another encoder network. We take the same approach in this study. The encoder network receives the image of the destination and transforms it into a vector. The transformed vector has the same dimension as the internal states of the slow RNN and is used as the initial states of the slow RNN. At this time, the image of the destination is preprocessed by CNN^{rec} and the preprocessed image is transformed by the encoder network. The encoder network are optimized to minimize the errors between the predictions and teaching sequences in the same way as the other networks.

C. Training

Our model was trained as a controller in the navigation task. The objective of the learning was minimizing the errors between the predicted (or generated) and teaching sequences. In addition to the navigation sequences, our model was also trained to predict the inputs in the warm-up phase for recognizing the agents' position. For multimodal integration, our model learned to predict vision and motion, even when one of the visual or motion inputs was not given.

The backpropagation through time algorithm was used for training the parameters of our model. We implemented our model by using the PyTorch library (http://pytorch.org), and the training procedure was accelerated by GPU computation.

D. Training Settings

The training data for learning was constructed as follows. First, the agent moved to visit randomly selected positions in the arena as warm-up phase (snapshot image was not given). This warm-up phase last for 100 time steps. When the agent arrived at the destination before 100 time steps, another random destination is given. At 100 time steps, the target destination point were given by giving a snapshot image which is a visual image taken at the target destination point. The agent moved to visit the target destination. The agent's motion is controlled to trace the path founded by A* search algorithm. The visual images that were captured while moving were realized were stored for training. The motion sequences, stored visual images, and snapshot images were used as training data. Three hundred sequences were prepared for the warm-up phase. Ten different navigation sequences were created from the same starting position after a single sequence of the warm-up phase, which indicated that there were 3,000 (= 300 $\times$ 10) sequences for the navigation sequences.

Our model consisted of 128, 64, 32, and 32 neurons for the fast RNN, medium RNN, slow RNN, and encoder network, respectively. The time constant τ of the fast, medium, and slow RNNs was 5, 25, and 50, respectively. The number of dimensions for $\boldsymbol{f}^v_t$, $\boldsymbol{f}^m_t$, $\hat{\boldsymbol{f}}^v_t$, and $\hat{\boldsymbol{f}}^m_t$ was 64. The structure

TABLE I

THE STRUCTURE OF CNN^{rec} AND CNN^{gen}

CNN^{rec}

layer	type	size	channel	kernel size	stride	padding
1	input ($\boldsymbol{v}_t$)	64 x 16	3	-	-	-
2	conv	32 x 9	8	4 x 2	2 x 2	1 x 1
3	conv	16 x 5	16	4 x 2	2 x 2	1 x 1
4	conv	8 x 3	32	4 x 2	2 x 2	1 x 1
5	fully connected	1 x 1	64	-	-	-

CNN^{gen}

layer	type	size	channel	kernel size	stride	padding
1	input ($\hat{\boldsymbol{f}}^v_t$)	1 x 1	64	-	-	-
2	fully connected	8 x 3	32	-	-	-
3	deconv	16 x 6	16	4 x 2	2 x 2	1 x 0
4	deconv	32 x 10	8	4 x 2	2 x 2	1 x 1
5	deconv	64 x 16	3	4 x 2	2 x 2	1 x 2

of CNN^{rec} and CNN^{gen} are shown in Tab. I. In order to prevent our model from overfitting to the training sequences, a L2-norm of the model's parameters was added to minimize the objectives with coefficients of 10^{-3}.

IV. EXPERIMENT AND RESULTS

Our model was trained 10 times in all of the training sequences, and the abilities of the obtained model were evaluated.

A. Basic Navigation Behavior

In order to evaluate the navigation ability, the outputs of the trained model was used as the actual motion generation to move the agent. Fig. 3 (a) shows the example trajectories of agent's position in navigation. The agent navigates three different destinations from the same starting position by recognizing the image of destinations. The agent successfully achieved the destinations while avoiding the obstacles. It means that our model could recognize the existence of obstacles and selects motions properly to achieve the given destinations. Fig. 3 (b) shows other example trajectories. It shows the case of different starting points and the same goal. Our model also successfully navigates the specific goal from different starting points.

In order to investigate how the destination was represented in the internal states of our model, we collected the internal states while the agent navigates for different destinations and analyzed them. All obstacles except for the two static obstacles are not placed this time. First, we divide the grid arena into 5 $\times$ 5 cells in order to classify the internal states corresponding to the destinations. As the area which has the obstacle cannot be the destinations, the total number of categorized destinations became 23. In this experiment, the agent starts the navigation task from five different initial positions (four corners and center of the arena). The agent is required to navigate to the destinations designated by the snapshot images from the initial positions. Totally, we obtained $115 (= 23 \times 5)$ navigation behaviors and internal state changes for analysis. The internal states are visualized in two-dimensional space using a principal component analysis (PCA). The first and second components are used for visualization. Fig. 4 shows the visualized internal states of slow RNN colored by the categorized destinations.

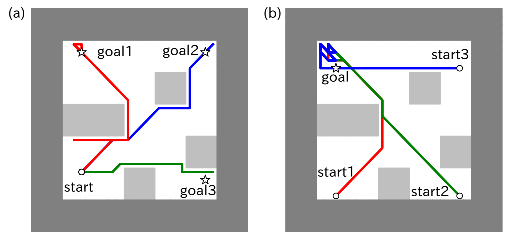

Fig. 3. (a) Examples of navigation behaviors from one starting position to different goals performed by trained model. (b) Examples of navigation behaviors from different starting positions to one goal performed by trained model.

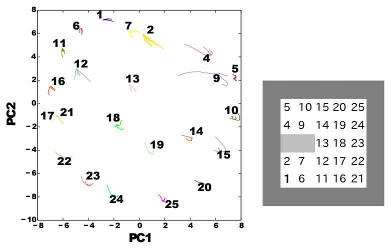

Fig. 4. The visualization of PCA analysis results with respect to the position of destination. The numbers shown together with the internal states of slow RNN indicate the position of destionations. The numbers correspond to the number shown in the map at the right side.

The states are numbered with respect to the position in the arena (corresponding positions are shown in the right figure). The internal states of fast and medium RNN seem not to organized by destination position (not shown). The initial states of slow RNN seem to align corresponding to the topological layout of positions in the arena. Moreover, the internal states keep distance from that for other destinations in the PCA space throughout the navigation. It supports the explanation that our model realized the navigation by remembering the position of destinations rather than remembering the sequence of action.

B. Shortcut Behavior

Next, we investigate how the agent behaves when shortcut path appears by removing the static obstacles which always exist during training. Our model is trained to go straight to the destinations because of A* algorithm, if there is no obstacle on the way. If our model well generalized these experiences during training and recognized the space of the arena as spatial representation (not remembering the action sequences to the destination), the agents would be able to take the newly appeared shortcut path. In this experiment, two static obstacles are removed, which means the shortcut path between upper-left and lower-left corners becomes available. All obstacles other than two static obstacles are not placed. The starting and destination points are set to the upper-left and lower-left corners, respectively. Fig. 5 shows that the trajectories of the agent when obstacles are removed. For comparison, the trajectories in the case that the obstacles are not removed are also shown. If the obstacles are not removed, the agent reached the destination by taking detour

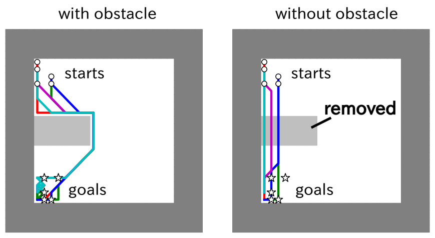

Fig. 5. The navigation behaviors in the cases that the static obstacles are removed or not removed. Five different behaviors are shown.

TABLE II

REACHING RATE AND PATH LENGTH

		Path Length avg.	s.d.	Reaching Rate
lower-left->upper-left	with obstacles	28.6	11.4	94%
	without obstacles	18.2	2.7	100%
upper-left->lower-left	with obstacles	22.3	2.6	100%
	without obstacles	15.6	1.3	100%

path. On the other hand, in the case that the obstacles were removed, the agent passed through the area where the static obstacles were placed.

In order to quantitatively evaluate the shortcut behavior, we performed the removing-obstacle experiments between upper-left and lower-left corners 100 times for each direction and calculated the reaching rate and path length. We set the starting and destination positions from the area of 4×4 size at upper-left and lower-left corners and assume that the agent reaches the destination when the agent enters the area including the destination position. The path length is calculated as accumulated Euclidean distance on the way to enter the destination area after the agent starts navigation. The reaching rate and path length are calculated for each path from upper-left to lower-left and vice versa. For comparison, these values are also calculated in the case that the obstacles are not removed. Table II shows the calculated results. The reaching rate is 100 % and it shows that the agent can successfully achieve the destination even in the unknown situation where the static obstacles are removed. The path length when the obstacles are removed is much shorter than that when the obstacles are not removed. This shortcut behavior indicates that our model realizes the obstacle avoidance and goal-directed behavior by not remembering the sequence of action but always selecting proper actions considering the environmental state and position of the desired destination.

We conduct the internal states analysis again for investigating how our model recognizes the environmental states and selects actions for achieving destination. We collected the internal states in the above experiment. This time, the agent navigates from each of 4×4 cells at lower-left corner to one destination point at upper-left corner. As a result, we obtained 16 trajectories of internal states of each layer of RNNs. The collected internal states are mapped to PCA space by the mapping function constructed for the previous PCA analysis above. We focus on the internal states of fast and slow RNN. Fig. 6 shows the results of the internal

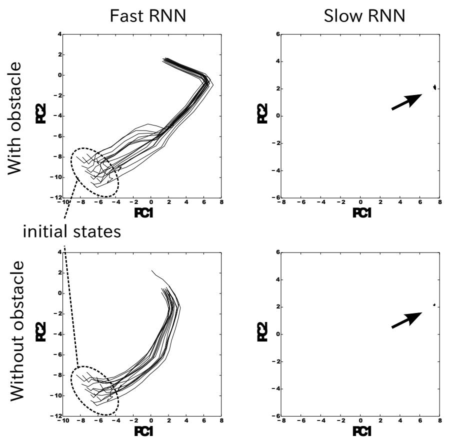

Fig. 6. The visualization of PCA analysis results. The internal states of fast and slow RNN are shown in both case that the obstacles are removed and not removed.

states analysis. Comparing the internal states of the fast RNN, they changed differently in the second half. Especially, there is a clear difference in the first component of the fast RNN. The values of the first component in the case with the static obstacles change more than in the case without them. It is considered that our model recognized whether the obstacles exist or not and fast RNN changes in response to the wall avoidance behaviors. If the obstacles do not exist, fast RNN does not need to change internal states and can keep the states for achieving the destination. On the other hand, there is no difference in the internal states of the slow RNN between conditions. The internal states were kept same regardless of the existence of the walls. It means that the slow RNN encodes the intentions to the destinations as top-down regulation to the behaviors.

C. Closed-loop Mental Simulation

By feeding back the prediction as intentions for the top-down regulations, our model can autonomously generate visuomotor sequences like mental simulation. In contrast to the above result of interactive motion generation, the agent behaves in the internal mental world rather than in the actual external environment. This experiment can clarify how the internal world is constructed inside of the agent. After the snapshot image is given, external visual inputs are shutout and the agent generates the visuomotor sequences in closed-loop manner. The starting point of navigation is at upper-left and destination is at lower-left. The other settings are the same as the above experiment. Fig. 7 shows an example of the simulated behaviors. For comparison, generated behaviors in interaction with external environment are shown for both case with and without obstacles. The mentally simulated trajectory passed on the obstacles and it is close to the trajectories in the case without obstacles. This result shows that our model assumes that there is no obstacle in the internal world. In other words, our model knows that, if the obstacles were removed, the agent can pass the area where the obstacles are placed, even though

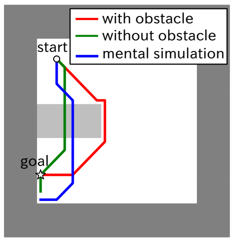

Fig. 7. An example of mentally simulated navigation behaviors.

our model have never entered the area during the training. This result is consistent with the results of the internal state analysis that the slow RNN controls the behaviors by position-based representation. Moreover it is considered that our model obtained the spatial structure of the external world like the cognitive map by generalizing the experiences.

V. Discussion

The network used in this study controls the robot behaviors by changing its neural dynamics. Especially, the slow dynamics of neurons control the fast dynamics in a top-down manner. Ito et al. previously studied network that controls humanoid robot's behaviors by neurons of slow dynamics called parametric bias [12]. When the robot's behavior is guided by a human supporter, the network changes the internal states of slow dynamics and the robot starts performing the guided behavior. It is more like bottom-up formation of the intentions. On the other hand, our model tries to find a way to achieve a goal regardless of environmental situations (i.e.. disappearing walls). It could be a top-down regulation of behaviors from the slow dynamics as intentions. Our experimental results showed that our model successfully changes its behaviors in response to the different initial positions and placements of the obstacles. It means that our model successfully recognizes the visual sensory inputs by CNN and uses the recognition for selecting actions in addition to top-down controlling signal from slow RNN.

Hwang et al. constructed the cognitive model that generates goal-oriented behaviors (grasping a specific object) from goal-oriented states of slow RNN [13]. In their experiment, the network has to generate different behaviors to achieve a task with same internal states of slow dynamics in response to the various situations. The distinct point of this study from Hwang's study is that our model controls spatial navigation. In the spatial navigation, there are spatial relationships between goals. As shown in the experimental results, our model learned the spatial relationships (Fig. 4) and used the relationships for navigation and performed the shortcut behavior (Fig. 5). We considered that this spatial recognition results from the setting of the navigation task. In the navigation task, there are numerous behaviors that our model should generate because our model has to navigate to any goals from any starting points. Thus the acquired representation, in which the goals which are spatially close in the environment are also placed closely, is efficient for navigation, because similar behavior is required for close goals. We considered the acquired spatial recognition has similar concepts to the cognitive map of rats in Tolman's experiment.

VI. Conclusion

This study investigated how the navigation ability is self-organized in a hierarchical neural network with multiple-time scales. For achieving destination where the placement of obstacles were randomly changed, the fast RNN successfully selected action for achieving destination and controlled the obstacle avoidance, and the slow RNN generated the goal-directed flow in a top-down manner. Moreover, even though our model did not have a priori knowledge about the spatial structure, our model obtained cognitive map-like representation in slow RNN. It is considered that the cognitive map-like representation resulted from the generalization of training experiences. We considered that the goal-oriented initial states rather than sequence-oriented initial states made our model to find the general structure of the space, where there are many paths for achieving destination. This is because our model was trained to generate the goal-directed behavior to the same destination, which is indicated by the initial states, by different paths.

References

[1] E. C. Tolman, "Cognitive maps in rats and men.," *Psychological review*, vol. 55, no. 4, p. 189, 1948.

[2] S. J. Kiebel, J. Daunizeau, and K. J. Friston, "A hierarchy of time-scales and the brain," *PLoS Comput Biol*, vol. 4, no. 11, p. e1000209, 2008.

[3] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, pp. 1928–1937, 2016.

[4] R. W. Paine and J. Tani, "How hierarchical control self-organizes in artificial adaptive systems," *Adaptive Behavior*, vol. 13, no. 3, pp. 211–225, 2005.

[5] W. Noguchi, H. Iizuka, and M. Yamamoto, "Hierarchical recurrent neural network model for goal-directed motion planning using self-organized cognitive map," in *Proceedings of the Twenty-Second International Symposium on Artificial Life and Robotics 2017 (AROB 22nd 2017)*, pp. 73–78, 2017.

[6] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[7] R. D. Beer, "On the dynamics of small continuous-time recurrent neural networks," *Adaptive Behavior*, vol. 3, no. 4, pp. 469–509, 1995.

[8] Y. Yamashita and J. Tani, "Emergence of functional hierarchy in a multiple timescale neural network model: a humanoid robot experiment," *PLoS Comput Biol*, vol. 4, no. 11, p. e1000220, 2008.

[9] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, "Deconvolutional networks," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 2528–2535, IEEE, 2010.

[10] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," in *International Conference on Learning Representations (ICLR 2016)*, 2016.

[11] R. Nishimoto, J. Namikawa, and J. Tani, "Learning multiple goal-directed actions through self-organization of a dynamic neural network model: A humanoid robot experiment," *Adaptive Behavior*, vol. 16, no. 2-3, pp. 166–181, 2008.

[12] M. Ito, K. Noda, Y. Hoshino, and J. Tani, "Dynamic and interactive generation of object handling behaviors by a small humanoid robot using a dynamic neural network model," *Neural Networks*, vol. 19, no. 3, pp. 323–337, 2006.

[13] J. Hwang, M. Jung, J. Kim, and J. Tani, "A deep learning approach for seamless integration of cognitive skills for humanoid robots," in *Development and Learning and Epigenetic Robotics (ICDL-EpiRob), 2016 Joint IEEE International Conference on*, pp. 59–65, IEEE, 2016.