# Robot Path Planning by LSTM Network Under Changing Environment

Masaya INOUE, Takahiro YAMASHITA, and Takeshi NISHIDA

Kyushu Institute of Technology,
1-1 Sensui, Tobata, Kitakyushu, Fukuoka 804–8550, Japan

**Abstract.** We propose a novel robot path planning method that combines the rapidly-exploring random tree (RRT) and long short-term memory (LSTM) network. In this method, numerous and good paths are generated in the robot configuration space by the RRT method, a convolutional autoencoder and LSTM combination network is trained by them. The proposed method overcomes the difficulty of general methods with neural networks, i.e., "the acquisition of a large amount of training data." Moreover, the difficulty of general random based methods, i.e., "the reproducible path generation" is resolved with high-speed.

## 1 Introduction

Path planning is an important function for executing autonomous moving robots, and many path planning methods that satisfy various constraints, such as avoiding obstacles and energy efficiency, have been proposed. There are many random sampling algorithms widely used for robot path planning, such as the probabilistic roadmap [1], rapidly-exploring random tree (RRT) [2], and their improved algorithms. These methods are used for connecting nodes to create a path and create a strong feature so that the path can always be found under the condition that the starting point and goal point can be connected. Moreover, in situations where obstacles or other factors exist in the target area, or when the environment dynamically changes, it is difficult to set nodes to be prescribed in space. Therefore, it is known that these methods are more advantageous in such situations compared to Dijkstra's algorithm and A* method [3] that use fixed nodes. However, since these algorithms use random numbers, some problems occur, such as fluctuations of the found path for each search time and finding a redundant path when the search time is not sufficient.

On the other hand, path planning methods by machine learning such as methods using a neural network (NN) [4] and deep Q-network combining reinforcement learning [5] have been proposed. Although these methods require large amounts of learning beforehand, they can generate paths at high speed after learning. In addition, unlike the abovementioned methods, the same path is generated for the same starting point and goal point. Owing to the generalization ability of NN, path generation is possible even for conditions not used during learning. However, these methods cannot positively take into account the robot's physicality as prior knowledge, and trial and error is required to acquire a large amount of training data for learning. It is often impossible to perform many trial and error processes on site.

Therefore, we propose a new path planning method that combines the random sampling method and NN method to overcome "the fluctuation and redundancy of found path" with the random sampling algorithm and "the acquisition of a large amount of training datasets" for the NN method. In the proposed method, a large amount of high-quality paths are generated by RRT in the configuration space by considering the robot's physicality and environment, and learning by the NN is executed using these datasets. After learning, the NN can generate a high-quality path quickly and stably. This network also develops generalization ability.

## 2 Problem setting

First, let $C$ be the configuration space of a robot. In this study, to simplify the analysis we set $C \subset \mathbb{R}^2$ and the region is set as

$$C = \left\{ \boldsymbol{p} \triangleq [p_1\ p_2]^T \middle| p_{1l} \leq p_1 \leq p_{1u}, p_{2l} \leq p_2 \leq p_{2u} \right\}, \tag{1}$$

where $p_{1l}$, $p_{1u}$, $p_{2l}$, and $p_{2u}$ are constant. The starting points $\boldsymbol{p}_s$ and the goal points $\boldsymbol{p}_g$ of the paths are generated using random numbers in the specific regions. The regions of $C_s \subset C$ and $C_g \subset C$ are set as follows:

$$C_s = \left\{ \boldsymbol{p}_s \triangleq [p_{1s}\ p_{2s}]^T \middle| p_{1sl} \leq p_{1s} \leq p_{1su},\ p_{2sl} \leq p_{2s} \leq p_{2su} \right\},$$
$$C_g = \left\{ \boldsymbol{p}_g \triangleq [p_{1g}\ p_{2g}]^T \middle| p_{1gl} \leq p_{1g} \leq p_{1gu},\ p_{2gl} \leq p_{2g} \leq p_{2gu} \right\}.$$

We consider the problem of generating a path that connects arbitrarily set starting points $\boldsymbol{p}_s \in C_s$ and goal points $\boldsymbol{p}_g \in C_g$ without colliding with obstacles.

Next, the path generated using the starting point and goal point pair is expressed as follows:

$$P^{(n)} \triangleq \left\{ \boldsymbol{p}_s^{(n)}, \boldsymbol{p}_1^{(n)}, \cdots, \boldsymbol{p}_e^{(n)}, \cdots, \boldsymbol{p}_E^{(n)}, \boldsymbol{p}_g^{(n)} \right\}, \tag{2}$$

where $n = 1, \cdots, N$ represents the path number; $m = 1, \cdots, E$ is the node number; and $\boldsymbol{p}_e^{(n)} \in C$ is a node of the path. The sum of the Euclidean distances between the nodes is called the total length of the path.

In this research, we consider the problem based on $C \subset \mathbb{R}^2$; hence, we describe the configuration space as an image with obstacles. The image is represented as matrix $\boldsymbol{I}^{(r)} \in \mathbb{R}^{A \times B}$ composed of pixels, where $r = 1, \cdots, R$ is the index of the image, and $A$ and $B$ are the number of pixels of the height and width of the image, respectively. The obstacle area is represented by setting the value of matrix elements as 1, and there are no obstacles in $C_s$ and $C_g$. Also, the value of matrix elements representing the region where the obstacle does not exist is 0. The path $P^{(n)}$ used for training must be set so that there are no obstacles on the node and the line segment connecting them.

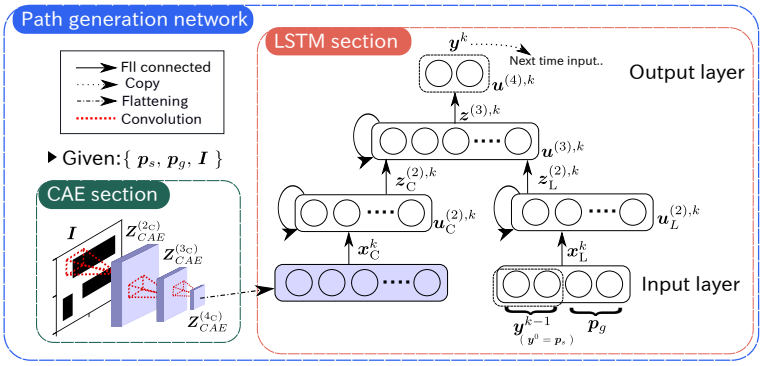



**Fig. 1.** Overview of the proposed network. This shows signal flow in the recalling phase.

# 3 Proposed Path Planning Method

## 3.1 Architecture of Proposed Network

The proposed network for path generation consists of the convolutional autoencoder (CAE) [7] and long short-term memory (LSTM) network [8]. The overview is shown in Fig. 1. The CAE section is integrated into the LSTM for feeding the environmental information, and the LSTM section executes the path generation. The proposed method has four phases as follows:

1. Training of the CAE section (CAE training phase).
2. Generating training data sets using trained CAE and RRT (Training data generating phase).
3. Path training by LSTM network (LSTM training phase).
4. Generating the new path (Recalling phase).

This process is illustrated in Fig.2.

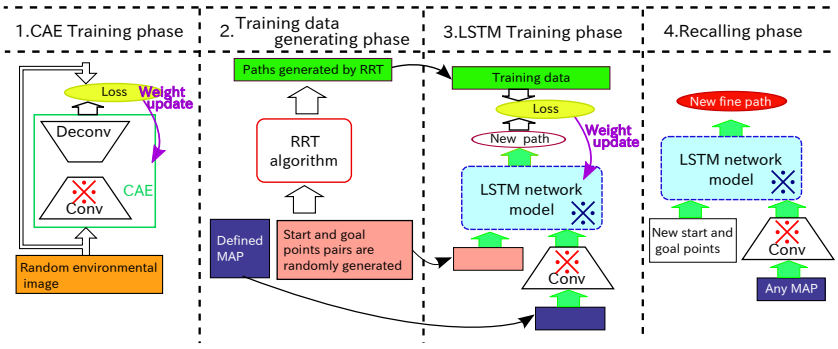



**Fig. 2.** Flow of the phases of the proposed method.

### 3.2 CAE Section

**Structure of CAE Section** In this section, the information of obstacles are extracted from $\boldsymbol{I}^{(r)}$ and the size is compressed by the CAE. Then, the output from the CAE is integrated with the path information in the input layer of the LSTM section. By compressing the image $\boldsymbol{I}^{(r)}$, the size of the LSTM section is reduced. Furthermore, by inputting information on the environment to the LSTM, it becomes possible to generate a path suitable for various environments. If there is no CAE section, the LSTM only learns the statistics of the paths without information on obstacles and avoids it. In this a case, it is impossible to acquire latent knowledge that obstacles must be avoided.

The CAE section is an $L_{\mathrm{C}}$ layered convolutional NN. Convolution processing is known to be very effective in image recognition. The convolution in the $l_{\mathrm{C}}$-th layer is executed against the element $z_{abd}$ of the output tensor $\boldsymbol{Z}_{CAE}^{(l_{\mathrm{C}}-1)} \in \mathbb{R}^{C\times V\times Q}$ using an element $h_{vqcd}$ of the filter set $\boldsymbol{H} \in \mathbb{R}^{V\times Q\times C\times D}$

$$u_{abd}^{(l_{\mathrm{C}})} = \sum_{c=1}^{C}\sum_{v=1}^{V}\sum_{q=1}^{Q} z_{sa+v,sb+q,c}^{(l_{\mathrm{C}}-1)} h_{vqcd}^{(l_{\mathrm{C}})} + b_{abd}^{(l_{\mathrm{C}})} \tag{3}$$

$$z_{abd}^{(l_{\mathrm{C}})} = \eta^{(l_{\mathrm{C}})}\left(u_{abd}^{(l_{\mathrm{C}})}\right) \tag{4}$$

where $V$ and $Q$ are the sizes in the horizontal and vertical axis direction of the filter, respectively; $c = 1, \cdots, C$ represents the channel number of input tensor; $d = 1, \cdots, D$ is the number of filters; $s$ is the pixel width scanned by the filter; $b_{abd}$ is a bias; and $\eta^{(l_{\mathrm{C}})}(\cdot)$ is called the *activation function*. The output tensor of the first layer corresponds to $\boldsymbol{I}$. In addition, each element $z$ corresponds to each pixel.

During training, restoration is performed using an $L_{\mathrm{C}}$ layered deconvolution layer, and the error backpropagation method is applied to the error between the restored image and original image. The deconvolution layer is the layer that performs convolution processing after padding target matrix components with arbitrary values [10]. It also has the function of restoring the compressed tensor by convolution layer.

**CAE Training Phase** The CAE section is trained using environmental datasets with the following procedure:

**(1)** Using randomly-positioned obstacles, a large amount of environmental images $\boldsymbol{I}^{(r)}$ are generated.
**(2)** The images $\boldsymbol{I}^{(r)}$ are divided into mini-batch sets, and CAE training is executed using these.
**(3)** Learning is aborted after sufficiently repeating (2).

**Training Data Generating Phase** The training dataset for LSTM section is generated with the following procedure:

**(1)** Give $\boldsymbol{I}^{(r)}$ $(r = 1, \cdots, R)$ and $N$ sets of start $\boldsymbol{p}_s^{(rn)} \in C_s$ and goal $\boldsymbol{p}_g^{(rn)} \in C_g$ $(n = 1, \cdots, N)$ for each image.

**(2)** Generate paths $P^{(rn)}$ using $\boldsymbol{p}_s^{(rn)}$ and $\boldsymbol{p}_g^{(rn)}$ in $\boldsymbol{I}^{(r)}$ by applying RRT.
**(3)** Modify the generated paths by the trajectory refinement method.
**(4)** Extract high-quality paths based on the number of nodes $E$.

It should be noted that in (4), the mode value of the number of nodes of generated paths is found; paths with number of nodes that deviate from this value are deleted. With these procedures, various paths for environmental images containing various obstacles are generated. Then, the image information $\boldsymbol{Z}_{CAE}^{(r)}$ compressed by the CAE and path $P^{(rn)}$ are combined to become the training datasets of the LSTM.

### 3.3 LSTM Section

Let us assume an LSTM network with $L_{\mathrm{L}}$ layers. There are two input layers and hidden layers that receive outputs from them in this network, they are integrated in the upper hidden layer and connected to the output layer. Specifically, in the simulation described later, the input of the input layer that accepts the CAE output is called $\boldsymbol{x}_{\mathrm{C}}^k \in \mathbb{R}^{108}$, while the input of the input layer that accepts information on the route is called $\boldsymbol{x}_{\mathrm{L}}^k \in \mathbb{R}^4$

$$\boldsymbol{x}_{\mathrm{C}}^k = \boldsymbol{z}_{CAE}^{(r)},\ \boldsymbol{x}_{\mathrm{L}}^k = \left[\boldsymbol{y}^{(k-1)T}\ \ \boldsymbol{p}_g^T\right]^T,\ \boldsymbol{y}^k = \boldsymbol{p}_k, \tag{5}$$

where $\boldsymbol{z}_{CAE}^{(r)}$ is a flattened vector of $\boldsymbol{Z}_{CAE}^{(r)}$ output of the CAE section, $\boldsymbol{y}^0 = \boldsymbol{p}_s$, and $k$ represents the recalling step corresponding to node number. When $\boldsymbol{x}^k$ is input into the LSTM network, the network recalls the next node of path $\boldsymbol{y}^k$. The number of cells in the input layers are 108 and 4, and they are fully connected to the corresponding hidden layers. The number of cells in each hidden layer is larger than the number of cells in the input layers (e.g., 120). The output of the hidden layers are integrated into the next hidden layer.

**LSTM Network** The input and output vectors of the $l_{\mathrm{L}}$-th layer at time $k$ are represented by

$$\boldsymbol{u}^{(l_{\mathrm{L}}),k} \triangleq \left[u_1^{(l_{\mathrm{L}}),k}\ \cdots\ u_j^{(l_{\mathrm{L}}),k}\ \cdots\ u_J^{(l_{\mathrm{L}}),k}\right]^T, \tag{6}$$

$$\boldsymbol{z}^{(l_{\mathrm{L}}),k} \triangleq \left[z_1^{(l_{\mathrm{L}}),k}\ \cdots\ z_j^{(l_{\mathrm{L}}),k}\ \cdots\ z_J^{(l_{\mathrm{L}}),k}\right]^T \tag{7}$$

where $j$ is the unit number, $J$ is the total number of units, and $J$ is not common to all layers. The input vector is $\boldsymbol{x}^k = \boldsymbol{u}^{(1),k}$ in the input layer, and the output vector is $\boldsymbol{y}^k = \boldsymbol{z}^{(L_{\mathrm{L}}),k}$ in the output layer. The unit number and total number of units of the $(l_{\mathrm{L}}-1)$-th layer are represented by $j^-$ and $J^-$, respectively. The input propagation weight from the $(l_{\mathrm{L}}-1)$-th layer to the $l$-th layer is represented by $\boldsymbol{W}^{(l_{\mathrm{L}})} \in \mathbb{R}^{J\times J^-}$. The recurrent weight in the $l_{\mathrm{L}}$-th layer $(l_{\mathrm{L}} = 2, \cdots, L_{\mathrm{L}}-1)$ is represented by $\boldsymbol{R}^{(l_{\mathrm{L}})} \in \mathbb{R}^{J\times J}$. The bias is represented by $\boldsymbol{b}^{(l)}$, and each component is represented by $w$, $r$, and $b$. Here, $j'$ is an arbitrary unit number corresponding to the output signal of the time $k-1$ of the $l_{\mathrm{L}}$-th layer The components of $\boldsymbol{u}^{(l_{\mathrm{L}}),k}$ are given by

$$u_j^{(l_\mathrm{L}),k} = \sum_{j^-}^{J^-} w_{jj^-}^{(l_\mathrm{L})} z_{j^-}^{(l_\mathrm{L}-1),k} + \sum_{j'}^{J} r_{jj'}^{(l_\mathrm{L})} z_{j'+}^{(l),k-1} + b_j^{(l_\mathrm{L})}. \tag{8}$$

The elements of the output vector of the $l$-th layer are represented by

$$z_j^{(l_\mathrm{L}),k} = \eta^{(l_\mathrm{L})} \left( u_j^{(l_\mathrm{L}),k} \right). \tag{9}$$

To summarize the discussion above, the output of the $l$-th layer is represented as

$$\boldsymbol{z}^{(l_\mathrm{L}),k} = \eta^{(l_\mathrm{L})} \left( \boldsymbol{W}^{(l_\mathrm{L})} \boldsymbol{z}^{(l_\mathrm{L}-1),k} + \boldsymbol{R}^{(l_\mathrm{L})} \boldsymbol{z}^{(l_\mathrm{L}),k-1} + \boldsymbol{b}^{(l_\mathrm{L})} \right), \tag{10}$$

and the output of the output layer is represented as

$$\boldsymbol{y}^k = \eta^{(L_\mathrm{L})} \left( \boldsymbol{W}^{(L_\mathrm{L})} \boldsymbol{z}^{(L_\mathrm{L}-1),k} \right). \tag{11}$$

**LSTM Unit** The LSTM network is constructed by LSTM units [9], and the following calculation is executed for each unit. First, the forget gate opening ratio of the $d$-th unit is calculated as

$$f_d^k = \sigma \left( \boldsymbol{w}_d^{fg} \boldsymbol{x}^k + \boldsymbol{r}_d^{fg} \boldsymbol{z}^{(k-1)} + b^{fg} \right), \tag{12}$$

where $\sigma(\cdot)$ is the logistic sigmoid function, $\boldsymbol{w}_d^{fg}$ is a weight vector corresponding to the input vector $\boldsymbol{x}^k$ from the previous layer, $\boldsymbol{r}_d^{fg}$ is a weight vector corresponding to the input vector $\boldsymbol{z}^{k-1}$ from previous time, and $b^{fg}$ is a bias.

Next, the opening ratio of the input gate is calculated as

$$i_j^k = \sigma \left( \boldsymbol{w}_j^{ig} \boldsymbol{x}^k + \boldsymbol{r}_j^{ig} \boldsymbol{z}^{(k-1)} + b^{ig} \right). \tag{13}$$

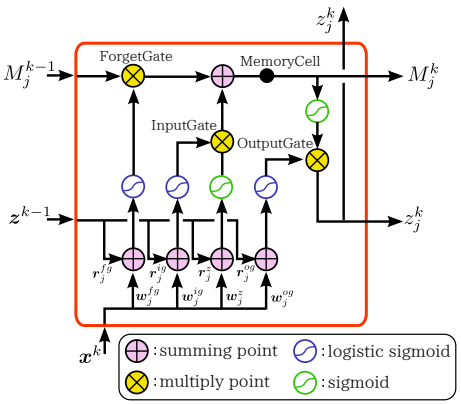



**Fig. 3.** Structure of a LSTM unit.

The signal through the input gate is

$$M_j^k = f_j^k \cdot M_j^{k-1} + i_j^k \cdot \tanh\left(\boldsymbol{w}_j^z \boldsymbol{x}^k + \boldsymbol{r}_j^z \boldsymbol{z}^{(k-1)} + b^z\right). \tag{14}$$

This value is transmitted as the internal state of the unit at the next time. Then, the opening ratio of the output gate is calculated as

$$o_j^k = \sigma\left(\boldsymbol{w}_j^{og} \boldsymbol{x}^k + \boldsymbol{r}_j^{og} \boldsymbol{z}^{(k-1)} + b^{og}\right), \tag{15}$$

and the output value $z_j^k$ is derived as

$$z_j^k = o_j^k \cdot \tanh(M_j^k). \tag{16}$$

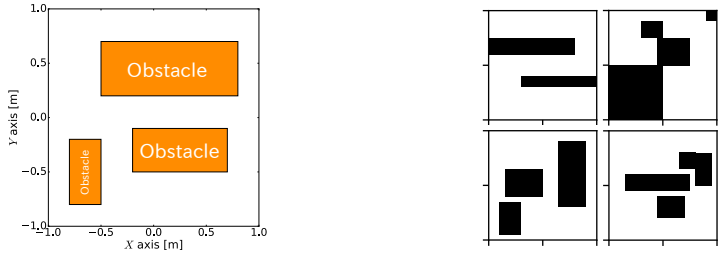
**LSTM Training Phase** The LSTM network is trained using the path dataset as follows:

**(1)** Shuffle the datasets $\{\boldsymbol{I}^{(r)}, P^{(rn)}\}$ and divide them into mini-batch sets.
**(2)** Execute the following processes on the first mini-batch.
1. $\boldsymbol{x}^k$ is constructed from the dataset and input into tge LSTM at time $k$.
2. $\boldsymbol{y}^k$ is recalled by forward propagation of $\boldsymbol{x}^k$.
3. Distance between $\boldsymbol{y}^k$ and $P^{(rn)}$ is accumulated as recalling error.
4. Construct $\boldsymbol{x}^{k+1}$ using $\boldsymbol{y}^k$, time step is replaced $k := k + 1$, and return to 2.
5. Loop until the final time.

**(3)** Modify connection weights in the network using backpropagation through time method [8].
**(4)** Execute the processing of (2) and (3) for the next mini-batch, and execute all mini-batches as well (This is called 1 epoch).
**(5)** Repeat processes (1)–(4) and execute a sufficient number of epochs.

### 3.4 Generating the new path

When a combination of an environmental image and a desired starting point and goal point is input into the network where learning has been completed, a path is generated. The network continues to generate nodes $\boldsymbol{y}^k$ until it approaches the goal point. Stop the path generation when it approaches the goal point beyond a certain threshold and combine it with the goal point.




(a) Two-dimensional region including obstacles.

(b) Examples of environmental images.

**Fig. 4.** Configuration space $C$ or environmental image $\boldsymbol{I}^{(r)}$.

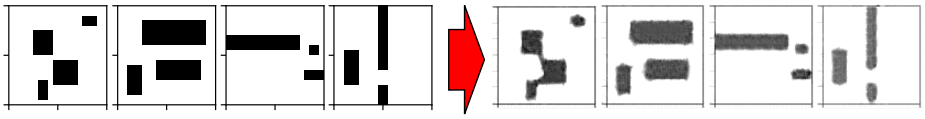


**Fig. 5.** Results of generalization test of CAE.

# 4 Simulation

## 4.1 Conditions

Examples of configuration space $C$ or environmental image $\boldsymbol{I}^{(r)}$ supposed in experiments are shown in fig. 4. The vertical and horizontal sizes of this image were set as $p_1 \in [-1, 1]$ and $p_2 \in [-1, 1]$, respectively.

## 4.2 Training

**Training of CAE** Table1 shows the parameters of the constructed CAE section. With the CAE section, the environmental information $\boldsymbol{I}^{(r)} \in \mathbb{R}^{100\times100}$ was compressed to $\boldsymbol{Z}^{(r)}_{\mathrm{CAE}}(A = 6, B = 6, D = 3)$ by three convolution processes.

The training of the CAE section was executed by restoring the compressed signal in the deconvolution layer and applying the error backpropagation method to the error from the original image. In order to confirm the generalization ability of the CAE section after the training phase, the environmental image which was not used for learning was restored after compression. The results of the generalization test, shown in Fig. 5, demonstrate that the CAE section acquired the ability to extract environmental features.

**Training Data Generation and Training of LSTM** Ten types (i.e., $r = 10$) of environmental images with different obstacle arrangements were prepared. We also set

**Table 1.** Parameters of the CAE section.

| Setting items | Detail |
|---|---|
| Size of environmental image $\boldsymbol{I}^{(r)} \in \mathbb{R}^{A\times B}$ | $A = B = 100$ |
| Number of convolutional layers $L_{\mathrm{C}}$ | 3 |
| Number of filters in each layer | 12, 6, 3 |
| Number of stride sizes in each layer $s$ | 1, 3, 5 |
| Filter size ($V \times Q$) | $V = Q = 5$ |
| Activation function $\eta(\cdot)$. | Rectifier function |
| Error function | Mean squared error |
| Mini-batch size | 200 |
| Initial weights | Random number in $[-0.1, 0.1)$ |
| Initial bias | None |
| Learning rate adjustment | Adam [12] |

**Table 2.** Parameters of the LSTM network.

| Setting items | Detail |
|---|---|
| Input dimension | 108(CAE), 4 |
| Output dimension | 2 |
| Number of hidden layers | 2 |
| Number of units in one hidden layer | 120 |
| Activation function of output layer | Identity function |
| Error function | Mean squared error |
| Mini-batch size | 200 |
| Initial weights | Random $[-0.1, 0.1)$ |
| Initial bias | None |
| Learning rate adjustment | Adam [12] |

$p_{1sl} = -1.0$, $p_{1su} = -0.5$, $p_{2sl} = 0.5$, and $p_{2su} = 1.0$ to set $C_s$, and $p_{1gl} = 0.5$, $p_{1gu} = 1.0$, $p_{2gl} = -1.0$, and $p_{2gu} = -0.5$ to set $C_g$. Next, we sampled $n = 3500$ combinations of start and goal points from $C_s$ and $C_g$, and generated paths $P^{(rn)}$ using RRT implemented in the Open Motion Planning Library [11]. We carried out processing to refine the generated paths using the path pruning method, shortcut method [13], and B-spline method [14]. Through these procedures, we constructed the training dataset $\{\boldsymbol{I}^{(r)}, P^{(rn)} | r = 10, n = 3500\}$.

The main parameters of the LSTM section are listed in Table 2.

### 4.3 Recalling of Path

**Path generation in trained environment** The network was trained in 10 environments, and the path generation ability of the trained network was investigated against arbitrary chosen starting and goal points in the trained environment. Examples of generated paths are shown in Fig.6. A route was generated by preparing 10,000 random pairs of starting points and goal points for each environment. Table 3 shows the success

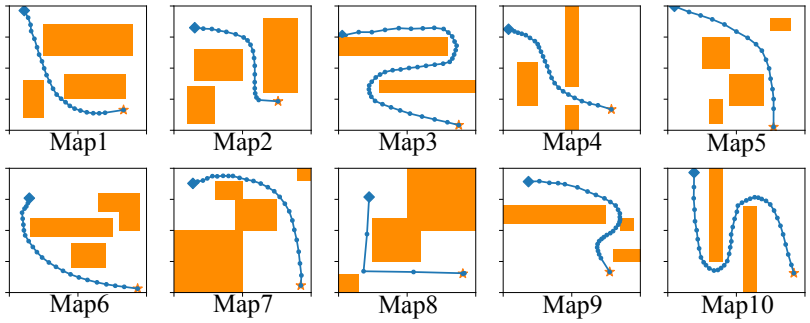



**Fig. 6.** Examples of the generated paths in trained environments. The symbol $\diamondsuit$ represents the starting point, and the symbol$\star$ represents the goal point.

**Table 3.** Success rates of path generation in trained environment.

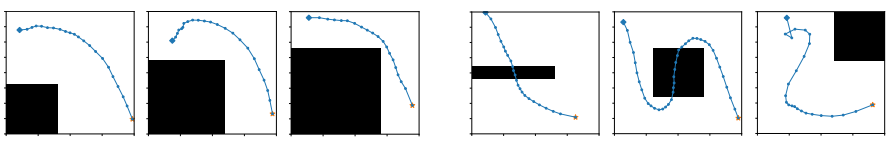
| environment | Map1 | Map2 | Map3 | Map4 | Map5 | Map6 | Map7 | Map8 | Map9 | Map10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Success rate [%] | 98.23 | 99.99 | 99.61 | 70.13 | 99.97 | 98.31 | 99.32 | 100 | 94.38 | 99.46 |

rates of path generation without collision with obstacles. The success rates were over 98% in 8 out of 10 environments.

**Path generation in unknown environment** The path generation ability of the trained network was investigated against arbitrary starting and goal points in an unknown environment. Examples of generated paths are shown in Fig.7. In the unknown environment, similar to the learned environment, the success rate was relatively high and path generation often failed when the environment was significantly different from the trained environment. Therefore, in order to verify the generalization ability of the trained network, the positions of all the obstacles in a specific trained environment were randomly fluctuated, and the change in the success rate of route generation was observed. The vertical and horizontal positions of all the obstacles in the environment shown in map 1 of Fig.6 were fluctuated from $\pm 0.01$ to $\pm 0.2$ [m] randomly and individually. The results shown in Fig.8 demonstrate that sufficiently high performance or generalization ability can be expected if fluctuations in the position of the obstacle are within 10%. It is found from these results that when the given environment is similar to the trained data set, the network realizes high path generation capability. Namely, since the network outputs the appropriate number of nodes and paths based on the training data set, it will be possible to improve the generalization ability by extending the training data set.

## 5 Conclusion

The proposed method has the following advantages: (1) In trained environments, paths can be generated without colliding with obstacles even if the starting and goal points change. (2) Even for unknown environments similar to trained environments, the method has the ability to generalize path generation. (3) In the composition of the dataset, path selection based on arbitrary evaluation index is possible. (4) It is possible to prevent the generation of poor quality or redundant trajectory caused by the trajectory generation method such as RRT. (5) The generated path does not fluctuate unlike with methods based on random numbers. (6) Path generation after training can be done quickly. (7) It is possible to configure the proposed method as hardware that operates at low power and high speed. We proposed a method to construct an LSTM network that recalls the



(a) Examples of success. (b) Examples of failure.

**Fig. 7.** Examples of the generated paths in unknown environments.

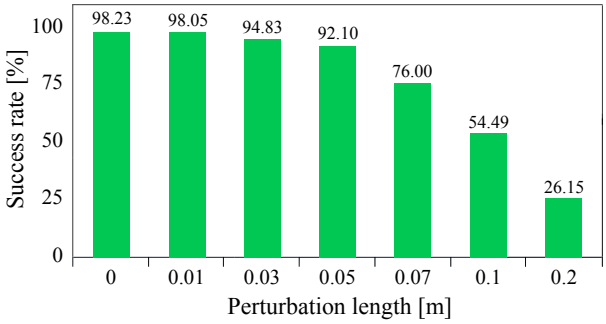



**Fig. 8.** Relationship between magnitude of environmental change and collision rate.

path of a robot by training with a large number of paths generated by RRT. The simulation results confirm that the proposed network achieves high learning and generalization abilities.

## References


1. Kavraki, L. E., Svestka, P., Latombe, J. -C., and Overmars, M. H., "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," IEEE Trans. on Robotics and Automation, 12.4, 566–580 (1996)
2. LaValle, S. M., Kuffner, J. J., "Rapidly-exploring random trees: Progress and prospects," Algorithmic and Computational Robotics: New Directions, Wellesley, 293–308 (2001)
3. Choset, H., Lynch, K., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L., Thrun, S., "Principles of Robot Motion: Theory, Algorithms, and Implementations," Cambridge: MIT Press (2005)
4. Yang, Simon X., and Max Meng. "An efficient neural network approach to dynamic robot motion planning," Neural Networks, 13.2, 143–148 (2000)
5. Gu, Shixiang, et al. "Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates." arXiv preprint arXiv:1610.00633 (2016)
6. Bin, N., Xiong, C., Liming, Z., Wendong, X., "Recurrent neural network for robot path planning," Parallel and Distributed Computing: Applications and Technologies. Springer Berlin Heidelberg, 188–191 (2004)
7. Masci, J., Meier, U., Cirean, D., Schmidhuber, J. , "Stacked convolutional auto-encoders for hierarchical feature extraction", Artificial Neural Networks and Machine LearningICANN 2011, 52–59 (2011)
8. Ilya, S., Training recurrent neural networks, University of Toronto (2013)
9. Gers, F. A., Schmidhuber, J., and Cummins, F., "Learning to forget: Continual prediction with LSTM," Neural computation, 12.10, 2451–2471, (2000)
10. Zeiler, M. D., Krishnan, D., Taylor, G. W., Fergus, R. , "Deconvolutional networks", In Computer Vision and Pattern Recognition , 2010 IEEE Conference on , 2528–2535 (2010)
11. Sucan, Ioan A., Mark Moll, and Lydia E. Kavraki. "The open motion planning library." IEEE Robotics & Automation Magazine 19.4, 72–82 (2012)
12. Diederik, D., Ba, J., "Adam: A method for stochastic optimization," arXiv preprint, arXiv:1412.6980 (2014)
13. Harada, K. "Optimization in Robot Motion Planning," JRSJ, 32.6, 508–511 (2014)
14. Harada, K., Hattori, S., Hirukawa, H., Morisawa, M., Kajita, S., Yoshida, E., "Two-stage time-parametrized gait planning for humanoid robots," IEEE/ASME Transactions on Mechatronics, 15.5, 694–703 (2010)