2018 18th International Conference on Control, Automation and Systems (ICCAS 2018)
Oct. 17–20, 2018; YongPyong Resort, PyeongChang, GangWon, Korea

Path Planning using Multilayer Neural Network and Rapidly-exploring Random Tree

Takahiro Yamashita[1] and Takeshi Nishida[2*]

Department of Mechanical control Engineering
Kyushu Institute of Technology, 1-1 Sensui, Kitakyushu, Fukuoka, 804-8550, Japan
[1]Email: yamashita.takahiro610@mail.kyutech.jp
[2*]Email: nishida@cntl.kyutech.ac.jp

Abstract: We propose a path generation method that combines a multilayer neural network (MLN) and a rapidly-exploring random tree (RRT), which is a path-planning method that uses random numbers. Specifically, an MLN that recalls the midpoint of the path is constructed using the data set created from RRT. The recall of the midpoint using this neural network is repeated and the path is generated. Through this method, it is possible to generate a repeatable path by using MLN at high speed.

Keywords: MLN, RRT, path planning

1. INTRODUCTION

Numerous methods have been proposed for the generation of paths (path planning) for the movement of robots or for achieving postural changes while avoiding obstacles. For example, potential methods, methods based on random-number generation, optimization methods, etc. are well known. Among them, rapidly-exploring random tree (RRT) [1], which is based on random-number generation, is widely used owing to its high probability of path discovery. RRT is a metaheuristic algorithm designed to efficiently search high- dimensional regions, and it connects randomly sampled nodes to generate a path. In addition, it is particularly suited to problems of path planning with faults and constraints, and by applying smoothing processing to the generated path, it is possible to generate high-quality paths. However, as random numbers are used, there is a problem that the paths generated for the same pair of start and end points change each time a trial is performed.

Various methods for generating paths using machine learning have been proposed, and there are methods using neural networks（NNs）[2] and methods combining reinforcement learning and NN[3]. With these methods, it becomes possible to generate a path at high speed after learning. Further, unlike the above-described method, the same path is generated for the same pair of start and end points. However, with these methods, it is necessary to prepare a large number of annotation data sets for learning; moreover it is impossible to explicitly consider the physicality of the robot as prior knowledge.

To solve these problems, a path-planning method that combines metaheuristic and NN has been proposed[4]. In this method, an RRT and long short-term memory (LSTM) network are used. This network recalls the path to avoid obstacles in certain environments. Although this network is small in scale, high-speed path generation is possible. However, this method requires complicated hyperparameter adjustment and learning strategies. Thus, in this paper, we propose a simple method using a multilayer neural network (MLN) to demonstrate that path generation based on the same strategy is possible even with an NN with a different structure from that of an LSTM. In addition, we propose a hierarchical recall method of a midpoint as a path-recalling method and a network structure to realize it.

2. MULTILAYER NEURAL NETWORK

2.1 Dataset

Here, we describe the dataset used for MLN learning. First, let $\mathbb{C}$ be the configuration space to search for the path, for simplicity $\mathbb{C} \subset \mathbb{R}^2$, expressed as follows.

$$\mathbb{C} = \{\boldsymbol{x} \triangleq [x\ y]^T \big| x_l \leq x \leq x_u, x_l \leq y \leq y_u\} \tag{1}$$

The start points $\boldsymbol{x}_s^{(n)} \in \mathbb{C}$ and end points $\boldsymbol{x}_g^{(n)} \in \mathbb{C}$ of the paths used for learning are generated using random numbers. Here, $n = 1, \cdots, N$ represents the number of generated start points and end points. Subsequently, a path $\boldsymbol{p}^{(n)}$ is generated by RRT using pairs of start and end points.

$$\boldsymbol{p}^{(n)} \triangleq [\boldsymbol{x}_s^{(n)}\ \boldsymbol{x}_1^{(n)}\ \cdots\ \boldsymbol{x}_m^{(n)}\ \cdots\ \boldsymbol{x}_M^{(n)}\ \boldsymbol{x}_g^{(n)}] \tag{2}$$

Here, m is the number of the node, $\boldsymbol{x}_m^{(n)} \in \mathbb{C}$ is the node of the path generated by the RRT, and the sum of the Euclidean distances between the nodes is called the total distance of the path. The path in which the total distance of the generated paths largely deviates from the average value of the total distance of all paths is considered to be poor in quality and is excluded from the subsequent processing. Subsequently, the redundant paths are refined using the path- pruning method， shortcut method[5]，B-spline method[6], etc. for the generated path. Finally, the intermediate point $\boldsymbol{x}_a^{(n)}$ of the generated path， the intermediate point $\boldsymbol{x}_{b1}^{(n)}$ between the start point and the inter-

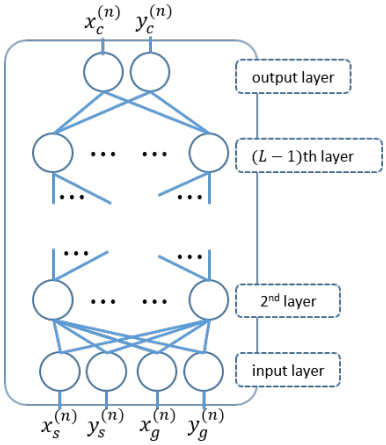


Fig. 1 Structure of MLN

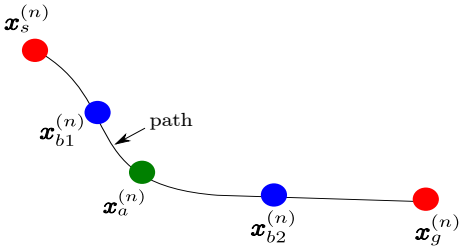


Fig. 2 Node of the recalling path

mediate point, and the intermediate point $\boldsymbol{x}_{b2}^{(n)}$ between the end point and the intermediate point are determined.

2.2 Learning MLN

The input vector $\boldsymbol{x}^{(n)}$ and the teacher signal $\boldsymbol{y}^{(n)}$ are expressed as follows.

$$\boldsymbol{x}^{(n)} = [\boldsymbol{x}_s^{(n)T}\ \boldsymbol{x}_g^{(n)T}]^T \tag{3}$$

$$\boldsymbol{y}^{(n)} = \boldsymbol{x}_a^{(n)} \tag{4}$$

Thus, the coordinates of the start point and the end point are considered as input, and the MLN is learned by using the coordinates of the midpoint of the path connecting the two points as the supervised data. An error backpropagation learning method is used for learning. The structure of the MLN is shown in Fig.1.

3. RETRIEVAL OF PATH BY MLN

3.1 Strategies for path generation

Regarding path generation targeting $\mathbb{C}$ including obstacles, in many examples, the midpoint of the path to be generated, and furthermore the midpoint are important; if they are determined, the path is often obtained. Therefore, in this study, we first determine the midpoint between the start point and the end point, and thereafter use a strategy to generate the path by generating the above point and the intermediate point between the start point and the end point. Fig.2 shows the outline of the generated path.

3.2 Structure of MLN recalling midpoint in multiple stages

The procedure for path generation using the proposed method of constructing and learning a layered MLN (L-MLN) by recalling the midpoint in multiple stages is shown below.

(1) Learn N_1 using dataset D_1.

$$D_1 = \{\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)} | n = 1, \cdots, N\} \tag{5}$$

This constitutes an MLN recalling the intermediate node $\hat{\boldsymbol{x}}_a^{(n)}$.

(2) An input signal having the intermediate node $\boldsymbol{x}_a^{(n)}$ as its end point and a node $\boldsymbol{x}_{b1}^{(n)}$ intermediate therebetween as a teacher signal.

$$\boldsymbol{x}_2^{(n)} = [\boldsymbol{x}_s^{(n)T}\ \boldsymbol{x}_a^{(n)T}]^T \tag{6}$$

$$\boldsymbol{y}_2^{(n)} = \boldsymbol{x}_{b1}^{(n)} \tag{7}$$

Using this dataset D_2, learn N_2.

$$D_2 = \{\boldsymbol{x}_2^{(n)}, \boldsymbol{y}_2^{(n)} | n = 1, \cdots, N\} \tag{8}$$

This constitutes N_2, which recalls $\hat{\boldsymbol{x}}_{b1}^{(n)}$.

(3) An input signal having the intermediate node $\boldsymbol{x}_a^{(n)}$ as its start point and a node $\boldsymbol{x}_{b2}^{(n)}$ intermediate therebetween as a teacher signal.

$$\boldsymbol{x}_3^{(n)} = [\boldsymbol{x}_a^{(n)T}\ \boldsymbol{x}_g^{(n)T}]^T \tag{9}$$

$$\boldsymbol{y}_3^{(n)} = \boldsymbol{x}_{b2}^{(n)} \tag{10}$$

Using this data set D_3, learn N_3.

$$D_3 = \{\boldsymbol{x}_3^{(n)} \boldsymbol{y}_3^{(n)} | n = 1, \cdots, N\} \tag{11}$$

This constitutes N_3, which recalls $\hat{\boldsymbol{x}}_{b2}^{(n)}$. This is the learning phase.

(4) The starting and ending points are input to N_1 and the three points $\hat{\boldsymbol{x}}_a^{(n)}$, $\hat{\boldsymbol{x}}_{b1}^{(n)}$, and $\hat{\boldsymbol{x}}_{b2}^{(n)}$ are recalled using the three learned MLNs. Finally, we create these paths by connecting these five points.

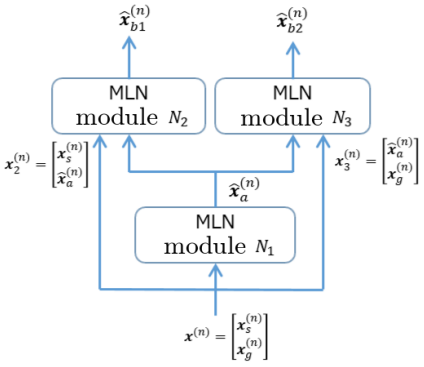


Fig. 3 Structure of L-MLN recalling midpoint in multiple stages

4. SIMULATION

4.1 Two-dimensional environment

The two-dimensional environment shown in Fig.4 was set as $\mathbb{C}_2$. In addition, $C_s \subset \mathbb{C}_2$ and $C_g \subset \mathbb{C}_2$ were set as areas to define the start and end points as follows.

$$C_s = \{[x_s\ y_s]^T \big| -1.0 \leq x_s \leq -0.5,\ 0.5 \leq y_s \leq 1.0\}$$
$$C_g = \{[x_g\ y_g]^T \big| 0.5 \leq x_g \leq 1.0,\ -0.5 \leq y_g \leq -1.0\}$$

Thus, consider the problem of generating a path connecting the start point and the end point arbitrarily set from each region. We used Chainer[7] as a framework for constructing and learning the L-MLN and verified it in the environment shown in Table 1. The RRT implemented in the open motion planning library（OMPL）[8] was used to generate the data set of the path used for learning.

4.1.1 Learning N_1

First, using uniform random numbers, we generated $\boldsymbol{x}_s^{(n)} \in C_s$ and $\boldsymbol{x}_g^{(n)} \in C_g$ with $n = 1100$. Subsequently, an RRT was applied to these sets to generate 1100 different paths. In the proposed method, the path is generated by connecting the three points recalled by the L-MLN, the start point, and the end point, but the smoothness of the path decreases because the number of nodes decreases.

When the path passing near the obstacle is approximated by a path with a small number of nodes, the possibility of collision is high, and hence, the generated path is restricted so as not to pass at a distance of 0.1 m from the obstacle. Specifically, the outer periphery of the obstacle was inflated by 0.1 m and the RRT was applied. By refining each path, $\boldsymbol{p}^{(n)}$ was generated to generate $\boldsymbol{x}_a^{(n)}$, $\boldsymbol{x}_{b1}^{(n)}$, and $\boldsymbol{x}_{b2}^{(n)}$. Further, 1000 sets out of 1100 sets of data, $n = 1, \cdots, 1000$, were used as training data and the remaining 100 groups were used as test data.

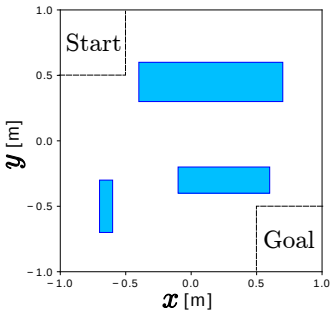


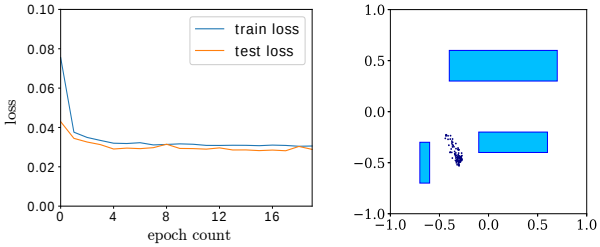
Fig. 4 Two-dimensional simulation environment

Table 1 Computer environment

Item	Details
OS	Ubuntu14.04 LTS(64bit)
CPU	Intel Core i7-6700 3.40GHz
RAM	DDR3-1600 8GB
Python Version	2.7.12

Table 2 Parameters of MLN

Item	Details
Number of input dimensions	4
Number of output dimensions	2
Number of intermediate layers	4
Number of middle layer units	30
Mini-batch size	50
Error function	mean squared error
Activation function	ReLU function
Adjustment of learning rate	Adam



(a) Error function transition (b) Recalled midpoint

Fig. 5 Result of recall of $\hat{\boldsymbol{x}}_a^{(n)}$ by N_1.

Subsequently, we learned the generated data set. Table2 shows the main parameters of the MLN used for learning. Fig.5 shows the decrease in error function with the progress of learning and the distribution of intermediate points recalled by inputting test data to N_1 after learning.

4.1.2 Learning N_2

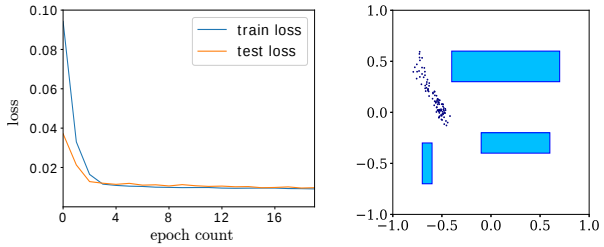
An input vector and an output vector were generated as shown in eq. (6) and eq. (7), respectively, with the start point of the path as $\boldsymbol{x}_s^{(n)}$ and the intermediate point $\boldsymbol{x}_a^{(n)}$ as the end point of the path. Subsequently, these data sets were learned by N_2. The network parameter of N_2 was set to be the same as N_1.

Fig.6 (a) shows that the error function decreased with the progress of learning. Fig.6 (b) shows the distribution of the recalled intermediate points $\hat{\boldsymbol{x}}_{b1}^{(n)}$ by the trained N_2.

4.1.3 Learning N_3

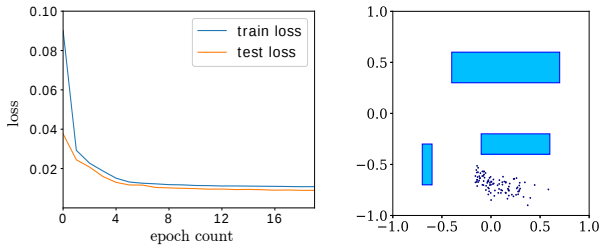
An input vector and an output vector were generated as shown in eq. (9) and eq. (10), respectively, with the intermediate point $\boldsymbol{x}_a^{(n)}$ as the start point of the path and $\boldsymbol{x}_g^{(n)}$ as the end point of the path. Subsequently, these data sets were learned by N_3. The network parameter of N_3 was set to be the same as N_1.

Fig.7 (a) shows that the error function decreased with the progress of learning. Fig.7 (b) shows the distribution of the recalled intermediate points $\hat{\boldsymbol{x}}_{b2}^{(n)}$ by the trained N_3.



(a) Error function transition (b) Recalled midpoint

Fig. 6 Result of recall of $\hat{\boldsymbol{x}}_{b1}^{(n)}$ by N_2



(a) Error function transition (b) Recalled midpoint

Fig. 7 Result of recall of $\hat{\boldsymbol{x}}_{b2}^{(n)}$ by N_3

4.1.4 Path generation in two-dimensional environment

The path generation was simulated using all the learned MNLs. An example of a path obtained by randomly generating a starting point and ending point from C_s and C_g and inputting them to the L-MNL is shown in Fig.8. It was confirmed that collision of obstacle did not occur for the starting and ending points of all 100 test data sets. In addition, the time required for learning by L-MLN was approximately 2.4 s, and the time required for path generation was approximately 4.0 ms to 4.5 ms per path.

4.2 Three-dimensional environment

Similar path-generating simulations were performed on the three-dimensional environment $\mathbb{C}_3$ shown in Fig. 9. A start area $C_s \subset \mathbb{C}_3$ and an end area $C_g \subset \mathbb{C}_3$ were set as follows:

$$C_s = \{[x_s\ y_s\ z_s]^T \big| 0 \leq x_s \leq 5,\ 15 \leq y_s,\ z_s \leq 20\},$$
$$C_g = \{[x_g\ y_g\ z_s]^T \big| 15 \leq x_g \leq 20,\ 0 \leq y_g,\ z_s \leq 5\}.$$

The main learning parameters of the L-MLN are shown in Table 3.

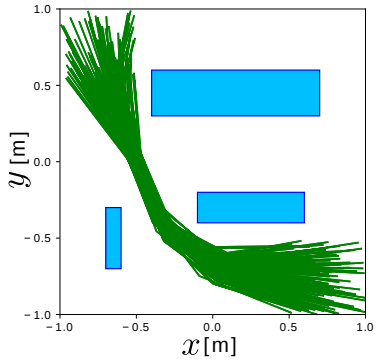


Fig. 8 Paths recalled by L-MLN in two dimensions

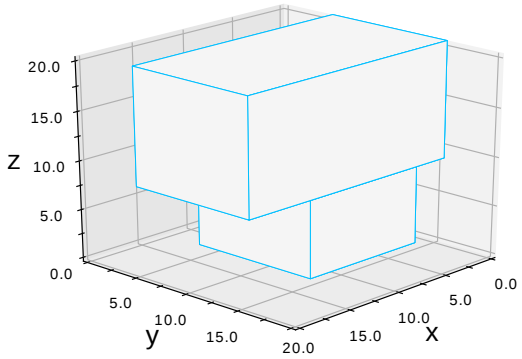


Fig. 9 Three-dimensional simulation environment

Table 3 Parameters of MLN

Item	Details
Number of input dimensions	6
Number of output dimensions	3
Number of intermediate layers	4
Number of middle layer units	30
Mini-batch size	50
Error function	mean squared error
Activation function	ReLU function
Adjustment of learning rate	Adam

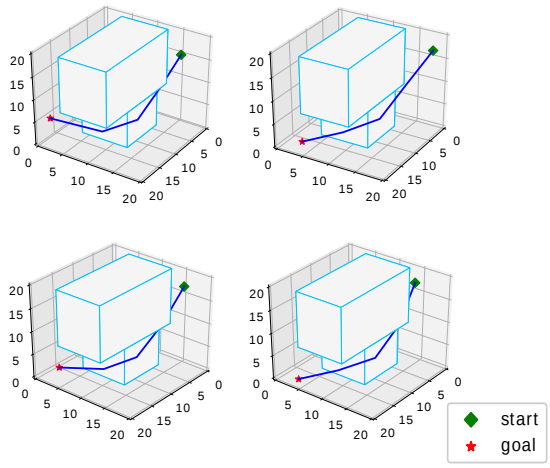


Fig. 10 Paths recalled by the L-MLN in $\mathbb{C}$

The path generation was executed by the L-MNL trained with the same procedure as previously described. Examples of paths generated by the L-MLN against the start and end point sets randomly extracted from C_s and C_g are shown in Fig.10. We conducted 100 path-generation tests, and it was confirmed that the path generated by any of them did not conflict with obstacles. In this simulation, the time required for the learning of 1000 paths by the L-MLN was 2.4 s, and the time required for path generation was approximately 7.0 ms to 7.5 ms per path.

5. CONCLUSION

We proposed a path-planning method that combines a metaheuristic method and an NN in an environment where obstacles exist. In this method, a path is generated by the L-MLN, which learns the path generated by an RRT. The proposed method avoids the limitation of metaheuristics i.e., the low reproducibility of path generation and the limitation of NN i.e., the amount of training data sets, and determines a quasi-optimal solution at high speed and high quality.

As an advantage of the proposed method, it is easy to set various restrictions, such as setting an evaluation function based on the motion characteristics of the robot when generating a learning path using the RRT. Moreover, it is possible to learn and recall a more precise path by increasing the hierarchy of the L-MLN. Furthermore, as the L-MLN has a simple structure, the learning time is short and it is easy to respond to relearning for environmental change. When actually using the proposed method for robot motion, it is possible to combine it with other optimization methods such as STOMP[9]. In the future, we will apply our method to a higher-dimensional configuration space and construct a network that can input environment information such as obstacles.

REFERENCES

[1] S. M. LaValle, J. J. Kuffner, “Rapidly-exploring random trees:Progress and prospects,” Algorithmic and Computational Robotics: New Directions, Wellesley, pp. 293–308, 2001.

[2] Simon X. Yang, M. Meng, “An efficient neural network approach to dynamic robot motion planning,” Neural Networks, 13.2, pp. 143–148, 2000.

[3] Gu, Shixiang, et al., “Deep Reinforcement Learning for Robotic Manipulation with Asynchronous OFF-Policy Updates,” arXiv preprint arXiv: 1610.00633, 2016.

[4] M. Inoue, T. Yamashita, T. Nishida, “Robot Path Planning by LSTM Network Under Changing Environment,” Proc. of The International Conference on Computational Sciences, Advanced Database and Computing, CS38, Thailand, 2017.

[5] K. Harada, “Optimization in Robot Motion Planning,” Journal of the Robotics Society of Japan（JRSJ）, Vol. 32, No. 6, pp. 508–511, 2014.

[6] K. Harada, S. Hattori, H. Hirukawa, M. Morisawa, S. Kajita, E. Yoshida, “Two-stage time-parameterized gait planning for humanoid robots,” IEEE/ASME Transactions on Mechatronics, 15.5, pp. 694–703, 2010.

[7] T. Seiya, et al., “Chainer. A next-generation open source framework for deep learning,” Proc. of workshop on machine learning systems in the twenty-ninth annual conference on neural information processing systems, 2015.

[8] I. A. Sucan, M. Moll, E. Kavraki Lydia, “The open motion planning library,” IEEE Robotics & Automation Magazine 19.4, pp. 72–82, 2012.

[9] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, S. Schaal, “Stomp: Stochastic trajectory optimization for motion planning,” in Robotics and Automation, IEEE International Conference on, pp.4569–4574, 2011.