2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)
Bari, Italy. October 6-9, 2019

Real-time Obstacle Avoidance using Supervised Recurrent Neural Network with Automatic Data Collection and Labeling

Shao-Hung Chan, Xiaoyue Xu, Ping-Tsang Wu, Ming-Li Chiang, Li-Chen Fu, *Fellow, IEEE*

Abstract— **In this paper, we propose an approach for real-time obstacle avoidance based on a supervised Recurrent Neural Network (RNN). As compared with conventional rule-based methods, fewer hyper parameters are needed to be tuned in the proposed system. On the other hand, as a data-driven system, our approach generates training data autonomously without manual labeling process. One of the main features of the proposed system is data generation, which can provide thousands of training data for supervised learning using simply 2D occupancy grid maps as input. To efficiently generate the path data, we utilize *A** algorithm as the initial guide for the autonomous training process of the RNN model. After that, the trained model will perform local path planning to avoid obstacles, which is tested in practical environments. With the proposed approach, we can effectively reduce the training time while maintaining satisfactory performance. Simulated experiments show that the proposed system not only exhibits the features of *A** algorithm in global aspect for path planning, but also performs obstacle avoidance in local aspect. As a by-product, the simulation results also show that the autonomously trained model can be successfully applied to many different scenarios.**

I. Introduction

For autonomous mobile robots, one of the crucial abilities is to navigate from points to points safely and efficiently. The so-called navigation task can be divided into two major sub-tasks: path planning, which is based on *a priori* global information, and obstacle avoidance, which relies on local information that the robot can obtain from the environment. However, while global path planning techniques can successfully predict a trajectory connecting start and goal points, the navigation task is prone to fail if some changes take place in the environment. Local obstacle avoidance prevents this by enabling the robot to avoid collision with obstacles in real time. Therefore, in this paper, we aim to design a real-time obstacle avoidance strategy for mobile robots. Numerous algorithms have already been introduced to deal with the obstacle avoidance problem, such as Artificial Potential Fields (APF) [1], grid-based Vector Field Histogram (VFH) [2][3], Elastic Band (EB) [4], and Dynamic Window Approach (DWA) [5]. In the following paragraph, we will briefly describe these technologies and their shortcomings.

The APF algorithm in [1] utilizes the vector summation of the attractive force generated by the goal point and the repulsive forces from different obstacles respectively to control the robot's mobility; however, this algorithm is known to converge to local minima, a sub-optimal solution. The VFH algorithm in [2] and [3] statistically represents the robot's environment through the so-called histogram grid, placing therefore great emphasis on dealing with uncertainty from sensor and modeling errors. The EB algorithm in [4] models the local path planning task as an elastic band which can be stretched to become a curve when an obstacle is encountered. The DWA algorithm in [5] generates possible local trajectories through sampling the angular and linear velocities at certain periods, which are a function of the surrounding obstacles, the relative distance to the goal and the robot orientation. Nevertheless, these rule-based algorithms require sophisticated approaches, even under the trivial scenario during the obstacle avoidance process, be taken into consideration. On top of that, numerous hyper parameters need to be tuned in order to fit the configuration of different robots.

Other than *a priori* knowledge based approaches, machine learning approaches or the data-driven approaches in the literature have been considered to be powerful techniques for navigation. In many cases, these data-driven approaches have proven to offer better solutions for complex problems which are not easily solved by heuristic algorithms. One of the most well-known machine learning structures is the Recurrent Neural Networks (RNN), which has been successfully applied on several time-oriented applications and sequential data analysis [6]. The advantage of applying RNN rather than regular Fully Connected Networks (FCN) is that the former is able to take historical data into account [7][8]. Thus, in this paper, we employ RNN structure as the system model for data-driven obstacle avoidance. In [9], a neural network-based method is proposed to deal with the robot navigation tasks. However, they only consider the current observation from the robot. Whereas in this work, we consider the RNN structure such that not only the current location and the nearby obstacles, but also the past few positions of the mobile robot are taken into account for future path prediction.

As one of the main challenges for data-driven approaches, a large amount of data is usually needed for convergence. Therefore, data collection and organization become important tasks. Data-driven approaches can be divided into two categories based on whether the data is labeled by a human or not, which are supervised learning and unsupervised learning. Supervised learning approaches, albeit well-established, require a considerable quantity of data pairs, which in turn makes manual labeling necessary. Unsupervised techniques on the other hand do not require manual data labeling. One of the most famous among them is the Deep Reinforcement Learning [10][11][12]. However, this approach requires either a well-constructed simulation environment or a time-consuming data collection process from different scenarios [12]. In this paper, one of the main contributions is that our RNN model, with the

Shao-Hung Chan, Xiaoyue Xu and Ping-Tsang Wu are with the Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan (e-mail: [r06921017, r07921094, r05921013] @ntu.edu.tw).

Ming-Li Chiang is with the Graduate Institute of Automation Technology, National Taipei University of Technology, Taipei, Taiwan (e-mail: minglichiang@ntu.edu.tw).

Li-Chen Fu is with the NTU Center for Artificial Intelligence & Advanced Robotics, Taipei, Taiwan (e-mail: lichen@ntu.edu.tw).

978-1-7281-4569-3/19/$31.00 ©2019 IEEE

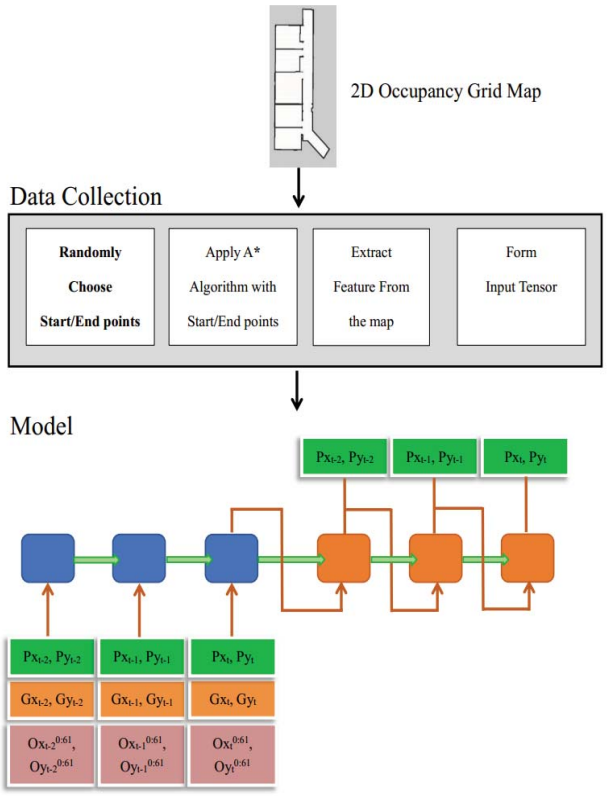


Fig. 1 System Architecture for Training

aid of the A^* algorithm, can be trained in an automatic manner instead of being trained by pair-wise, manually labeled data. Then, the system will generate feasible path data and execute regression learning. Hence, the only input of the proposed system are two-dimensional occupancy grid maps during the training stage, which means, the manually labeling process is no longer needed. After the training process, the system is able to perform obstacle avoidance in real time. In contrast to traditional data-driven approaches, our proposed system only requires two-dimensional occupancy maps which can be obtained from a variety of sources such as the well-established SLAM algorithms in [13].

II. System Architecture

The main objective of this paper is to propose a system for real-time obstacle avoidance with automatic data collection and labeling. The system can be separated into two modes: training and testing. In training mode, as shown in Fig. 1, the input of the system is a 2D-grid map which can be obtained from either the top view of the indoor environment or SLAM algorithms. During the data collection process, the system will randomly choose two points in the available space (for details, please refer to Section III) as start and end of a trajectory. Then, the system will apply the A^* algorithm [14] to find an admissible path between them. This path, along with the surrounding observed obstacles will form as the input tensor for the RNN model. On the other hand, in testing mode, the system will take sensor data and goal position as inputs. Then, it will extract features from sensor data and apply coordinate transformations to obtain these locations in global coordinates. Finally, these features in global coordinates become the input tensor as in training mode.

III. Methodology

In this section, the process of training mode and testing mode will be explained in detail.

A. Training Mode

At the beginning of the training process, the 2D occupancy grid map, which represents an environment by marking non-traversable paths as a binary matrix, is required to generate suitable navigation strategies. This type of matrix map is often utilized in mapping applications for integrating sensor information into a discrete binary map, with free spaces being 0 and obstacles being 1. In addition, it is useful in path planning for the purpose of finding collision-free paths and localizing robots in a known environment. Because of the above features, an occupancy grid map is suitable to apply the A^* algorithm, which is used as the training data for our RNN model. Equation (1) shows the mathematical representation of a certain grid value $m_{x,y}$ on the position (x, y) of the 2D occupancy grid map $\boldsymbol{M}$:

$$m_{x,y}\begin{cases}0 & \text{if grid (x,y) is a free space}\\ 1 & \text{if grid (x,y) is an obstacle}\end{cases}\in \boldsymbol{M} \tag{1}$$

In order to automatically decide global start and end points for the A^* algorithm to generate a path, the system will first define the admissible space in the map. According to this free space, two points will be decided following the criteria listed in (2):

$$\begin{aligned} &Dist(p_S, p_E) \geq d_{th}\\ &p_S, p_E \in \boldsymbol{A} \subset \{m_{x,y} = 0 \mid m_{x,y} \in \boldsymbol{M}\}\end{aligned} \tag{2}$$

where p_S and p_E denote global start and end points, respectively. The set $\boldsymbol{A}$ refers to the admissible space that the robot is able to reach, which is slightly narrower than the free space set of the grid map. The $Dist(\cdot)$ function calculates the Euclidean distance between two points on the map with the purpose of avoiding paths that are not long enough to generate acceptable training data. As the process starts, p_S and p_E will be generated randomly within all the admissible space. If the chosen points fulfill the criteria above, these will be chosen as the global start/end points.

Once the global points have been determined, the system will generate a path by applying the A^* algorithm with reference to the map coordinate, known as the global path. The role of the A^* algorithm in this work serves as an instructor that guides the robot to learn adequate obstacle avoidance behaviors. Because of its outstanding performance on path searching ability, the robot is expected to imitate the path planning strategy based on local information as observed from the on-board sensors. As a result, the only input to train the system is the 2D occupancy grid map and thus the manual labeling process is not needed.

The next step is to extract the information contained in the map. Since the key elements for path planning are the start point, the end point, and the obstacles in the environment, our objective is to determine these properties at each step so that

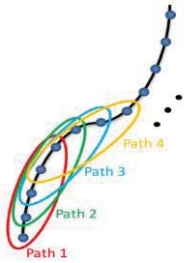
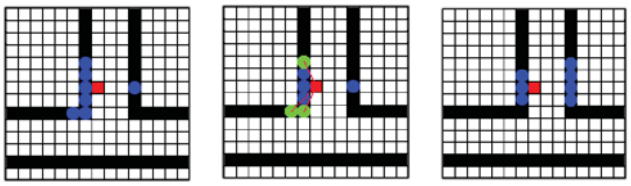


Fig. 2 The process of segmenting an A* path into a number of shorter paths. Every two consecutive paths will only shift forward by one step.

the system can imitate the A^* algorithm. Additionally, as the obstacle avoidance is a local behavior, all the information must be transferred from the global map coordinates to a local robot frame. We then segment the global path into a number of short paths in order for the robot to learn these behaviors from the local perspective. We update our path segmentation by moving one step further at each iteration. By implementing this technique, the system is able to generate the largest amount of training data from the limited available path. For every short path, the first and the last points will be considered the local start and end points respectively. Fig. 2 depicts a visual representation of the segmentation process.

In order to find obstacles, it is intuitive to apply a pixel-wise distance calculation due to the fact that the obstacles will be presented as binary value in 2D occupancy grid map. Then, a certain number of nearest pixels could be marked as obstacles as observed from the current position of the robot. An example of this process is shown in Fig. 3(a), where black squares represent obstacles in the 2D occupancy grid map, red squares represent the current position of the robot and blue dots represent candidate obstacles. However, applying only pixel-wise calculation may include obstacles which might not be detected at the robot's position in a real environment, shown as green points in Fig. 3(b).

To solve this quandary, inspired by the work of Bresenham on line algorithm in the field of image processing [15], we implement a method to overcome this shortcoming. Instead of drawing practical lines, our algorithm evaluates whether there exist additional obstacles between the current position and the candidate obstacle. The algorithm draws a line between each candidate obstacle and the current robot position. If there is any additional obstacle lying along the line, the algorithm will ignore this candidate obstacle. Algorithm 1 shows a pseudo-code of this algorithm. Once those obstacles which are impossible to detect in a real sensor setting have been excluded in Fig. 3(b), the final result of candidate obstacles is shown in Fig. 3(c).



(a) (b) (c)

Fig. 3 Collection of inaccurate obstacle information and its correction. Black squares denote obstacles, red square denotes the robot's current position, blue dots denote those obstacles that satisfy the pixel wise distance condition and green dots denote unsuitable obstacle candidates. (a) Candidate obstacles depends only on pixel-wise distance. (b) Using Bresenham's line algorithm to determine candidate obstacle validity. (c) The final set of obstacles.

The collecting features at each time step will then be formatted into an input tensor of the RNN model. For each position the robot takes in the trajectory, a concatenated vector will be generated. The composition of the vector is shown in Fig. 4. It is composed of three sections. The first section is the robot's current position, and this feature enables the robot to understand the relationship between the previous positions and the current position. The second part are the local goals, *i.e.* the last position of the current sub-path. It can be observed that the local goal position is repeated several times in the input tensor. This is used to give additional weight to the goal point, preventing the so-called unbalanced features problem. Finally,

Algorithm 1: Candidate Obstacles Selection

Input: 2D Occupancy Grid Map M
with value on $(x, y) = m_{x,y}$: {*free, occupied*} → {0,1}
Initialize: obstacles observed in p = *Obstacle Set* (p) = ϕ
p_S, p_E = ***random***$(\{(x, y) \mid m_{x,y} = 0\})$
Path = $\{ p \mid p \in A^*(p_S, p_E)\}$
for all p **in** *Path*:
  for all $p_{obstacle} \in \{(x,y) \mid m_{x,y} = 1\}$:
    line segment = ***Bresenham***$(p, p_{obstacle})$
    if *line segment* $\cap\, p_{obstacle} = \phi$:
      append $p_{obstacle}$ to *Obstacle Set*(p)

the last part contains the obstacle positions, which are obtained from the previous step as explained in the previous section.

In order to find the model which can best describe our problem, we have implemented different RNN models to test their performance:

a. Sequence-to-Sequence Learning (Seq2Seq) [7]

b. Sequence-to-Sequence with Peek Decoder [16][17]

c. Long Short-Term Memory (LSTM) [8]

The difference between *Sequence-to-Sequence with Peek Decoder* and regular Sequence-to-Sequence Learning is that the decoder gets a chance to peek at the context vector at every step [17]. The loss function is designed as shown in (3), which is composed of two parts: mean square error and cosine proximity:

$$L = \sum_{i=1}^{N} (w_{mse} \|y_i - \hat{y}_i\|_2^2 + w_{cos}(1 - \frac{y_i \cdot \hat{y}_i}{\|y_i\| \|\hat{y}_i\|})) \qquad (3)$$

in which the variable y_i is the ground truth data, $\hat{y}_i$ is the predicted values from the output of RNN model, w_{mse} and w_{cos} are hyper-parameters set to 10 and 0.01 respectively.

B. Testing Mode

In the proposed system, we assume the robot is equipped with a laser range finder with a 360-degree field-of-view

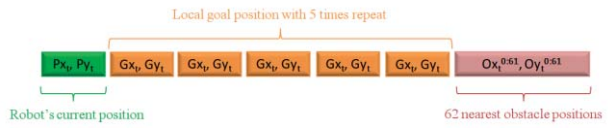
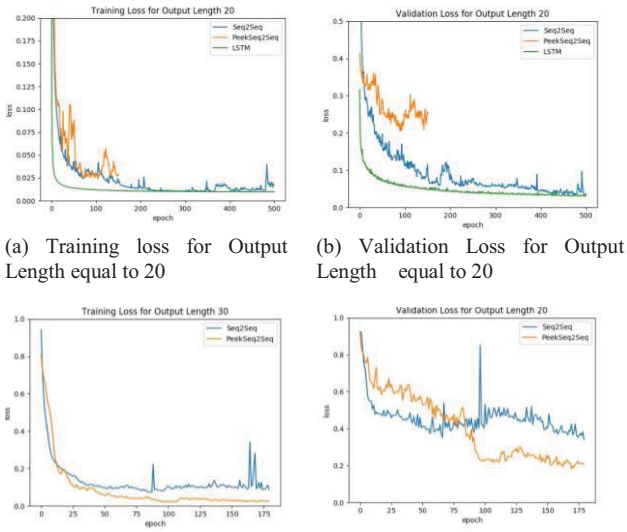


Fig. 4 The composition of feature vector



(a) Training loss for Output Length equal to 20

(b) Validation Loss for Output Length equal to 20

(c) Training Loss for Output Length equal to 30

(d) Validation Loss for Output Length equal to 30

Fig. 5 Training curves of different models

(FoV). Therefore, the sensor information will be distance values relative to the robot position. The global goal coordinate is given to the robot as *a priori* information. After processing the sensor data, the robot will extract the necessary features, namely local obstacle positions, and both start point position and end point position of the local path relative to the existing map. Later on, the system creates the input tensor as shown in Fig. 4. as an input to the RNN model. Note that here we repeat the local goal positions to solve the unbalanced feature problem; there are other options which can be considered such as weighting the features. The model then generates the predicted path for the system. The system will update its position first-in-first-out and repeat the process until it reaches the goal.

IV. Experimental Results

The experiments are performed using an Omnidirectional Wheeled Mobile Robot with 2D laser range finder running under simulated environments. The system is created using the Robot Operating System (ROS) with Kinetic distribution [18] and simulated with Gazebo 7 [19]. Results will be separated into three parts: training curves, testing on different maps, and adding new obstacles:

A. Training curves

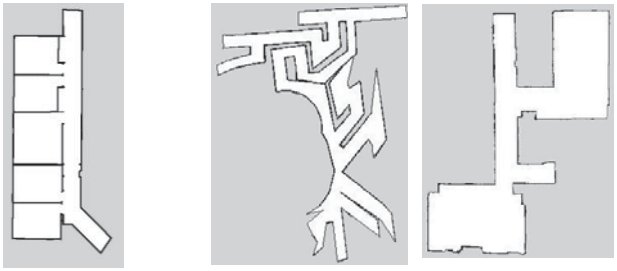
RMSProp [20] is chosen as the optimizer for our task with the learning rate being 0.0001. The training curves of different models with different pairs of input/output sequence length are shown in Fig. 5. Early stopping technique is applied during the training process with the hope of preventing overfitting. Besides, comparing the same model with different input/output sequence lengths will also lead to different loss values, which often increase as the output sequence length increases.

B. Testing on different maps

Analyzing the performance through training curves provides model candidates for suitable obstacle avoidance. On top of that, new paths in maps other than training are required in order to evaluate the robustness and accuracy of our system. A model for each different structure with the smallest validation loss is picked after the training process is done. Paths that are used as validation data are generated in the maps as shown in Fig. 6 (a). Paths coming from two new maps as shown in Fig. 6 (b) and Fig. 6 (c) are provided for testing these models. While the maps in Fig. 6 (b) and (c) are built in the simulated environment and real environment respectively, we show that the proposed system can be generalized to 2D grid maps from not only simulated environment but also the real world. In each map, the system first generates a path through A* algorithm soon after the start and end points are set. Then, it initializes its path data queue by adding the first segment of the path. In order to form the testing data, those points will be transformed into relative coordinates with respect to the current position. Additionally, the obstacles surrounding the robot will be utilized as training data for the RNN model. The temporary goal will be a certain point on the global path near the current position. The path predicted by the model will be pushed into the path queue, forcing the most previous data to be removed. The system will repeat the process until the robot reaches the final end point. If the predicted path encounters any obstacles on the way, the system will shift the temporary goal by a small amount, ranging around 0.01 meters, and predict the path once again. When the system is able to successfully reach the end point, performance is evaluated by measuring the relative distance error with respect to the path length generated through A^* algorithm, as in (4):

$$elative\ distance\ error = \frac{|Path_{pred} - Path_{A^*}|}{Path_{A^*}} \qquad (4)$$

TABLE I shows the mean relative distance error for different models with their respective maps with testing data, with Sequence-to-Sequence with Peek Decoder demoted as *Peak Seq2Seq*. the numbers after the name of the algorithm refer to the number of previous steps and the number of steps predicted relative to the current position of the robot. For instance, 30/20 means that the model uses 30 previous steps and outputs 20 the next 20 steps. On the other hand, there are some cases where the robot may be stuck on certain points or wander without reaching the goal point. Such cases will be marked as *Failed*. Comparing the performance among



(a) Map1: A 2D occupancy grid map used for training

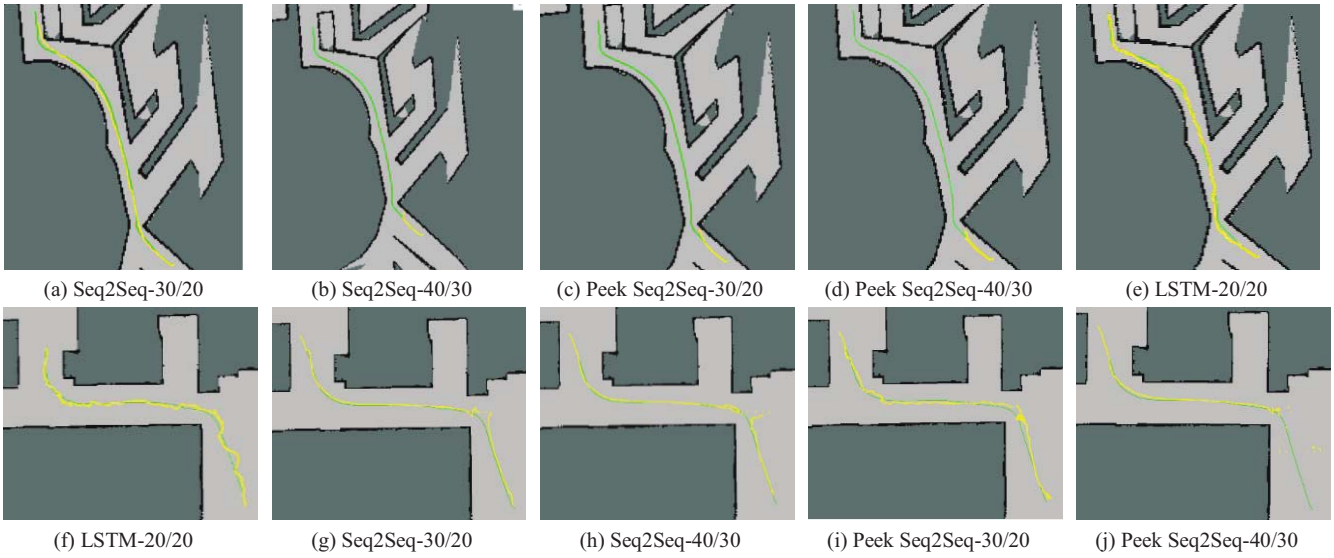
(b) Map2: A 2D occupancy grid map used for testing

(c) Map3: A 2D occupancy grid map used for testing

Fig. 6 The 2D occupancy grid maps used in the training and testing condition in the experiments. (a) and (b) are constructed within the simulated environments using SLAM algorithm. (c) is constructed from a real environment using SLAM algorithms

TABLE I. RELATIVE DISTANCE ERROR WITH RESPECT TO A* ALGORITHM

Model - Input / Output length	Map 1	Map 2	Map 3
Seq2Seq [7] - 30/20	**0.018**	**0.069**	**0.039**
Seq2Seq [7] - 40/30	0.128	Failed	Failed
Peek Seq2Seq [16][17] - 30/20	0.686	Failed	Failed
Peek Seq2Seq [16][17] - 30/20	Failed	Failed	0.870
LSTM [8] - 20/20	0.243	0.168	0.680



(a) Seq2Seq-30/20 (b) Seq2Seq-40/30 (c) Peek Seq2Seq-30/20 (d) Peek Seq2Seq-40/30 (e) LSTM-20/20

(f) LSTM-20/20 (g) Seq2Seq-30/20 (h) Seq2Seq-40/30 (i) Peek Seq2Seq-30/20 (j) Peek Seq2Seq-40/30

Fig. 7 Testing results of different RNN sampled from models. The upper figures are the result of the experiment on map2. The lower figures are the result of the experiment on map3. The green line and yellow line are the same representation as Fig. 8.

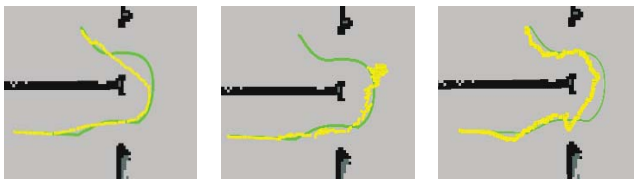
different models with the same output length of 20 in the testing paths generated in *Map 1*, the regular sequence to sequence model with input/output length equal to 30/20, denoted as *Seq2Seq - 30/20* in TABLE 1, reaches the end point successfully with the smallest relative error. On the contrary, *Peek Seq2Seq* tends to lose its track or become stuck, therefore it often fails to reach the goal point. As for *LSTM*, although it can complete the task, the relative error is larger than the sequence to sequence model.

Fig. 7 and Fig. 8 show some of the example results sampled from the different tested algorithms. Fig. 8 shows the application of different algorithms on the same map and path. While *Peek Seq2Seq* usually fails to achieve the goal, *Seq2Seq* and *LSTM* are both able to complete the task. Moreover, it is clear that the path that the Seq2Seq algorithm outputs stays closer to the path generated by *A** algorithm.

Fig. 7 shows the result of applying the trained algorithms to other maps that have never been seen by the algorithm during the training process. While some architectures will succeed some times and fail some others, *Seq2Seq - 30/20* is able to complete all tests successfully while at the same time being closest to the original A* path, as reflected in Table I.

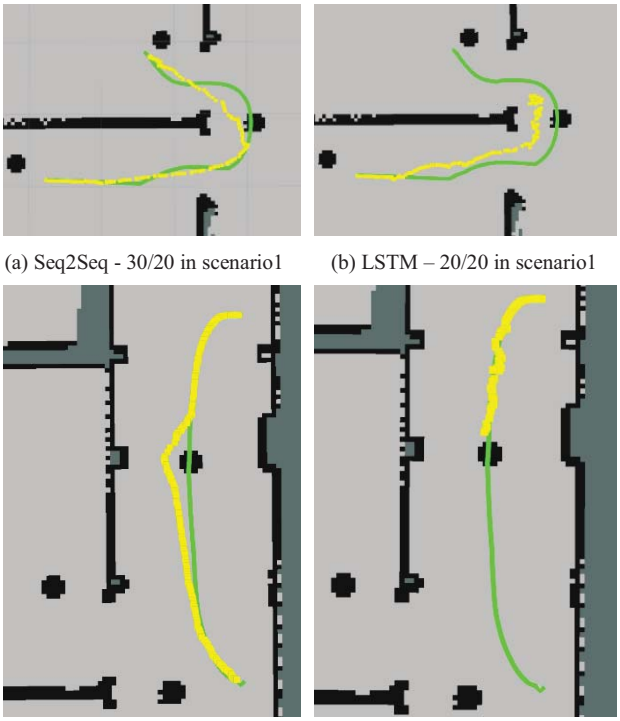
C. Adding new obstacles:

The purpose of adding new obstacles into map is to simulate whether sudden changes of the environment would affect the performance of the algorithm and whether it could successfully overcome them. That is, after a path has been generated using the *A** algorithm, there may be previously unseen obstacles that block its way. Therefore, the model should be able to act as a local path planner and avoid the robot collision with the additional obstacle. For this section, we only test those configurations with the best performance. As shown from the experiment conducted in the previous section, these are the sequence to sequence model with input/output length equal to 30/20 and the LSTM model with input/output length equal to 20/20. As shown in Fig. 9, LSTM starts to hover while extra obstacles appear in the map after the global path is planned. The Seq2Seq 30/20 configuration on the other hand, is able to complete the trajectory even with the new obstacles.



(a) Seq2Seq (b) Peek Seq2Seq (c) LSTM

Fig. 8 Testing examples on different models in the same environment and the same global goal. Green lines indicate the A* path planning result; yellow lines indicate the paths the robot travels during the obstacle avoidance process.



(a) Seq2Seq - 30/20 in scenario1 (b) LSTM – 20/20 in scenario1

(c) Seq2Seq - 30/20 in scenario2 (d) LSTM - 20/20 in scenario2

Fig. 9 The figures show the obstacle avoidance results for Sequence-to-Sequence model and LSTM model given two example scenarios. The green line and yellow line are the same representation as Fig. 8. From the results, we can see Sequence-to-Sequence model can successfully complete the task whereas LSTM model fails.

V. Conclusions

In this paper, we introduced a novel obstacle avoidance system that uses Recurrent Neural Networks with supervised learning. Instead of applying manually designed strategies in the traditional approaches, our system can learn and perform obstacle avoidance with fewer human efforts by automatically collecting and labelling the training data. Through several experiments, we have shown that the learning done on one environment can be generalized to different environments and successfully imitate the *A** behaviors. One thing worth mentioning is that while we used a 2D occupancy grid map built in the simulator for training, testing on a map built from a real environment can still be handled by the proposed method. The proposed system opens several possibilities to implement different additional features into the system, such as the presence of humans, objects or any other features that might affect the obstacle avoidance task. Since both the data collection and labeling are automatic, it is easy to fine tune the model into a specific environment. On the other hand, by the design flexibility of the feature vectors, there are myriad combinations which can be used to boost the predictive performance of the system, which is a great advantage in contrast to traditional obstacle avoidance approaches. As for the future work, the algorithm will be tested with physical robots to gauge its robustness in practical applications.

VI. Acknowledgment

This research was supported by the Joint Research Center for AI Technology and All Vista Healthcare under Ministry of Science and Technology of Taiwan, and Center for Artificial Intelligence & Advanced Robotics, National Taiwan University, under the grant numbers of 108-2634-F-002-016, 108-2634-F-002-017 and 108-2218-E-027-014.

References

[1] C. W. Warren, "Global Path Planning Using Artificial Potential Fields," in *Proc. 1989 IEEE Int. Conf. on Robotics and Automation (ICRA)*, 1989, pp. 316–321.

[2] J. Borenstein and Y. Koren, "The Vector Field Histogram - Fast Obstacle Avoidance for Mobile Robots," *IEEE Trans. Robot. Autom.*, vol. 7, no. 3, 1991, pp. 278–288.

[3] I. Ulrich and J. Borenstein, "VFH*: Local Obstacle Avoidance with Look-Ahead Verification," in *Procs. 2000 IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2000, pp. 2505–2511.

[4] S. Quinlan and O. Khatib, "Elastic Bands: Connecting Path Planning and Control," in *Proc. 1993 IEEE Int. Conf. on Robotics and Automation (ICRA)*, 1993, pp. 802–807.

[5] R. World, "The Dynamic Window Approach to Collision Avoidance," *IEEE Robot. Autom. Mag.*, vol. 4, 1997, pp. 23–33.

[6] A. Graves, A. -R. Mohamed, and G. Hinton, "Speech Recognition with Deep Recurrent Neural Networks," in *Procs. 2013 IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, 2013, pp. 6645–6649.

[7] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," in *Advances in Neural Information Processing Systems 27*, 2014, pp. 3104–3112.

[8] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, 1997, pp. 1735–1780.

[9] B. Ko, H. J. Choi, C. Hong, J. H. Kim, O. C.Kwon, and C. D. Yoo, "Neural network-based autonomous navigation for a homecare mobile robot," in *Procs. of the 2017 IEEE Int. Conf. on Big Data and Smart Computing (BigComp)*, 2017, pp. 403–406.

[10] A. Faust, H. -T. Chiang, N. Rackley, and L. Tapia, "Dynamic Obstacle Avoidance with PEARL: PrEference Appraisal Reinforcement Learning," in *Second Annual Machine Learning in Planning and Control of Robot Motion Workshop at 2015 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2015.

[11] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, F. -F. Li, and A. Farhadi, "Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning," in *2017 IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2017, pp. 3357–3364.

[12] S. -H. Hsu, S. -H. Chan, P. -T. Wu, K. Xiao, and L. -C. Fu, "Distributed Deep Reinforcement Learning based Indoor Visual Navigation," in *2018 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2018, pp. 2532–2537.

[13] G. Grisetti, C. Stachniss, and W. Burgard, "Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters," *IEEE Trans. Robot.*, vol. 23, no. 1, 2007, pp. 34–46.

[14] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. SSC-4, no. 2, pp. 100–107, 1968.

[15] J. Bresenham, "A Linear Algorithm for Incremental Digital Display of Circular Arcs," *Commun. ACM*, vol. 20, no. 2, pp. 100–106, 1977.

[16] K. Cho, B. v. Mërrieboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," in *Proc. 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1724–1734.

[17] F. Rahman, "Sequence to Sequence Learning with Keras." [Online]. Available: https://github.com/farizrahman4u/seq2seq.

[18] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System Morgan," in *ICRA Workshop on Open Source Software*, 2009.

[19] N. Koenig and A. Howard, "Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator," in *Proc. 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004, vol. 3, pp. 2149–2154.

[20] S. Ruder, "An overview of gradient descent optimization algorithms," 2016, pp. 1–14,.