Journal of Computer Science and Cybernetics, V.33, N.2 (2017), 107–118
DOI 10.15625/1813-9663/33/2/9822

AN EFFICIENT NAVIGATION FRAMEWORK FOR AUTONOMOUS MOBILE ROBOTS IN DYNAMIC ENVIRONMENTS USING LEARNING ALGORITHMS

XUAN-TUNG TRUONG, HONG TOAN DINH, CONG DINH NGUYEN

Department of Automation and Computer Engineering, Faculty of Control Engineering, Le Quy Don Technical University, Vietnam
xuantung.truong@gmail.com, dhtoan.hn@gmail.com, dinhnc@mta.edu.vn

Abstract. In this paper, we propose an efficient navigation framework for autonomous mobile robots in dynamic environments using a combination of a reinforcement learning algorithm and a neural network model. The main idea of the proposed algorithm is to provide the mobile robots the relative position and motion of the surrounding objects to the robots, and the safety constraints such as minimum distance from the robots to the obstacles, and a learning model. We then distribute the mobile robots into a dynamic environment. The robots will automatically learn to adapt to the environment by their own experience through the trial-and-error interaction with the surrounding environment. When the learning phase is completed, the mobile robots equipped with our proposed framework are able to navigate autonomously and safely in the dynamic environment. The simulation results in a simulated environment show that, our proposed navigation framework is capable of driving the mobile robots to avoid dynamic obstacles and catch up dynamic targets, providing the safety for the surrounding objects and the mobile robots.

Keywords. Autonomous mobile robot, mobile robot navigation, reinforcement learning, q-learning.

1. INTRODUCTION

The ability to autonomously navigate in dynamic environments, such as urban and terrain environments, museums, airports, offices and homes, and shopping malls, is crucial for mobile robots. If we wish to deploy the autonomous mobile robots in such environments, the first and most important issue is that, the mobile robots must avoid obstacles in their vicinity, while navigating safely towards a given goal. In order to archive that, several mobile robot navigation systems have been proposed in the recent years [1, 8, 11, 16].

The conventional mobile robot navigation frameworks can be roughly classified into two categories according to the techniques utilized to develop the motion planning systems: (i) model-based methods and (ii) learning-based approaches. In the first category, the navigation systems utilize available models, such as artificial potential field, vector field histogram, dynamic window approach, velocity obstacles, randomized kinodynamic planning, inevitable collision states, reciprocal velocity obstacles techniques to develop the motion planning system. In the second category of the methods, the machine learning techniques, such as inverse

reinforcement learning, are used to enable the mobile robots to navigate autonomously in dynamic environments.

Although the model-based approaches [2, 15, 17, 19, 20] have been evaluated such that, the navigation systems are capable of driving the mobile robots to navigate autonomously and safely towards a given goal, they still suffer essential weaknesses that seriously hinder the robot capabilities to navigate in dynamic environments. For example, in these papers, the authors have to hand-craft all the features of the surrounding environments and then incorporate them into the robot navigation system. In addition, several parameters are empirical set by the authors experiences for a specific environment. Moreover, this parameter set often need to be tuned individually, and can vary significantly for different environments.

In order to overcome the aforementioned drawbacks, recently, a few machine learning techniques-based navigation systems have been proposed to enable the mobile robots to navigate autonomously and safely in dynamic environments [4, 6, 21]. In these papers, the authors utilized the inverse reinforcement learning technique to develop the navigation systems of the mobile robot. Using the inverse reinforcement learning-based autonomous mobile robot navigation systems, the mobile robots are taught using the demonstrations of the human experts. The mobile robots will learn and get the policy of each action they have learnt. The trained mobile robots then can work in the same situation. Although these methods have enabled the mobile robots to deal with dynamic environments, and the mobile robots can learn to adapt to the surrounding environments. However, the environments are very dynamic, unknown, clustered, and unstructured. Therefore, the authors cannot teach the mobile robots by hand in each single situations.

To overcome the above-mentioned drawbacks, we propose an efficient navigation framework for autonomous mobile robots in dynamic environments using a reinforcement learning algorithm [18]. Because, the reinforcement learning is a useful way for robots to learn control policies. In addition, the main advantage of the reinforcement learning is the complete independence from human labeling. In other words, using the reinforcement learning algorithms, the mobile robots can learn without expert supervision. Furthermore, reinforcement learning is an online learning algorithm. Thus it offers to robotics a framework, which enables a mobile robot to autonomously discover an optimal action through trial-and-error interactions with surrounding environment [7].

The remainder of the paper is organized as follows. Section 2 presents the proposed navigation framework for autonomous mobile robot in dynamic environments. Section 3 provides the simulation results of the proposed model in a simulated environment. We conclude this paper in Section 4.

2. THE PROPOSED FRAMEWORK

2.1. Efficient mobile robot navigation framework

Our primary objective is to develop a navigation system that enable a mobile robot to navigate safely and autonomously in dynamic environments. To achieve that goal, in this paper, we propose an efficient mobile robot navigation system, as shown in Fig. 1. The efficient navigation system is developed based on the conventional navigation scheme

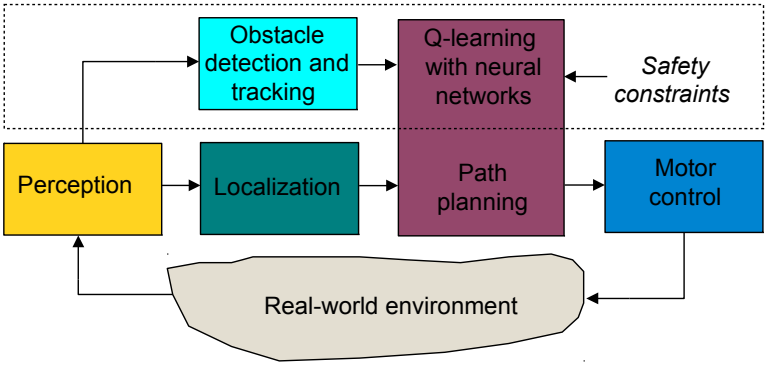

Figure 1. An efficient navigation framework for autonomous mobile robots in dynamic environments using a learning approach.

introduced by Siegwart et al. [16], and consists of two major parts: (i) the conventional navigation scheme, and (ii) the extended part (in the dash line box). In the first part, the conventional navigation scheme is based on the composition of four typical functional blocks: perception, localization, motion planning, and motor control. In the extended part, the navigation framework aims at extracting the obstacles information, including their position and the motion in the robots vicinity. These obstacles information, and the safety constraints such as the minimum distance from the robot to the surrounding obstacles, are then used as inputs of the q-learning algorithm with neural networks. The output of this model is the optimal action of the mobile robot. This optimal action is then combined with a path planning technique to allow the mobile robot to navigate autonomously and safely in the dynamic environments.

2.2. Q-learning with neural networks-based mobile robot navigation system

2.2.1. Reinforcement learning algorithms

Reinforcement learning [18], is a type of machine learning techniques, which allows agents and machines to automatically determine an optimal behavior within a specific context, to maximize their performance. Figure 2 shows a typical framework of the reinforcement learning algorithm, in which at each step, the agent executes an action, receives an observation (new sate), and receives a reward; while the environment receives an action, emits an observation (new sate), and emits a reward.

Reinforcement learning is a useful way for robotics to learn control policies [7]. The main advantages of the reinforcement learning algorithms are the complete independence from human labeling and the potential of automating design of the data representations. Therefore, reinforcement learning abstracted considerable attentions in the recent years. Conventional reinforcement learning methods are normally utilized to improve the controller performances in path-planning of robot-arms [22] and controlling of helicopters [10], but they are rarely applied to autonomous mobile robots.

A typical model of the reinforcement learning techniques is the q-learning algorithm [5], which is a model-free reinforcement learning technique. The main idea of q-learning

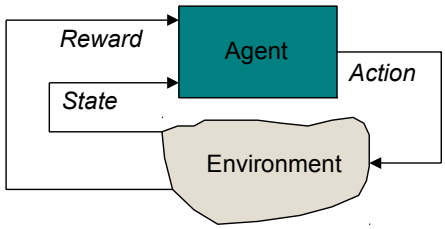

Figure 2. A typical framework of the reinforcement learning algorithm. An agent takes an action in an environment. The agent then transits to a new state and gets a reward, which is fed into the agent

algorithm is that, we can iteratively approximate the q-function using the Bellman equation, as presented in Eq. 1.

$$Q(s,a) = r + \gamma \max_{a'} Q(s',a'). \tag{1}$$

In the simplest case, the q-function is implemented as a table, with states as columns and actions as rows. A pseudocode of the q-learning algorithm is presented in Algorithm 1. The inputs of the q-learning algorithm are the set of states S, set of possible actions A, learning rate α, discount factor γ. The output is the table $Q(s,a)$, whose column size U is the number of the states, and row size V is the number of the possible actions. Thus the size of the table $Q(s,a)$ is $U \times V$. The table $Q(s,a)$ is then used to select an optimal action of this system based on the current states of the system. A detailed description of the q-learning algorithm was given in [5].

Algorithm 1: Q-learning algorithm

input : Set of states S, set of possible actions A, learning rate α, discount factor γ
output: $Q(s,a)$, state $s \in S$, action $a \in A$

```
begin
    Initialize Q(s,a) arbitrarily for s ∈ S, a ∈ A
    for (each episode):
        Initialize state s
        while (s is not a terminal state):
            Choose a from s using policy derived from Q
            Take action a, observe s', compute reward r
            Q(s,a) += α (r + γ max_a' Q(s',a') - Q(s,a))
            s = s'
        end while
    end for
```

Q-learning algorithm has been applied to real mobile robot platforms and has achieved considerable success [7]. However, the limitation of the original q-learning in robotics is that, the state space is so large, that combines with all possible actions, hence there is an exhaustive exploration of all state-action pairs. In other words, it is not feasible to store

every element of table $Q(s, a)$ separately. For example, lets take a robot manipulator with seven degrees-of-freedom, a representation of the robots state would consist of its joint angles and velocities for each of its seven degrees of freedom as well as the Cartesian position and velocity of the end effector. That accounts for $2 \times (7 + 3) = 20$ states and 7-dimensional continuous actions. If we assume that each dimension of the state-space is discretized into ten levels, we have 10 states for a one-dimensional state-space. Therefore, we will have 10^{20} unique states. In addition, in general, the states space of the the robot is continuous (the information from the sensors). In order to tackle these issues, function approximation is made use of in this paper.

2.2.2. *Q*-learning with neural networks

In this study, the neural networks-based function approximation is utilized instead of using a table $Q(s, a)$ from the original q-learning algorithm. Because, the neural networks offer many advantages, such as quality of the generalization, limited memory requirement for storing the knowledge, and continuous state space of the system. In this case, the memory is used to store only the weights of the neural networks. Therefore, the memory size required by the system to store the knowledge is defined by the number of connections of the neural networks. And it is independent of the number of explored state-action pairs.

Like the original q-learning model, this new model will accept a state and an action, and spit out a value of a state-action pair. Importantly, however, unlike the lookup table $Q(s, a)$, the state-action pair is the output of a neural networks. In addition, the neural networks also has a bunch of parameters associated with it. These are the architecture of the neural networks and its weights θ. Therefore, our Q function actually looks like this $Q(s, a, \theta)$, where θ is a vector of parameters. Moreover, in the training process, instead of iteratively updating values in the table $Q(s, a)$, the system will iteratively update the weights θ of the neural networks, therefore it learns to provide us with better estimation of the value of the state-action pairs.

In this study, a procedure of the q-learning with neural networks used to develop the autonomous mobile robot navigation system is presented in Algorithm 2. The inputs of the algorithm are the laser data $X = (x_1, x_2, ..., x_N)$, robots pose (position, orientation and motion of the mobile robot), learning rate α, discount factor γ, epsilon-greedy policy ϵ, and safety constraints. The output of the algorithm is the $Q(s, a, \theta)$ value, which allows the mobile robot to select an optimal action based on the current states of the robot. The possible actions of the mobile robot consist of move forward, left, right, or backward. The safety constraint is the safety distance from the robot to the obstacles, and are used to compute the reward of the q-learning with neural networks model.

In the q-learning algorithm, the mobile robot learns a q-function that can be used to determine an optimal action. To accomplish that, there are two things that are useful for the robot to do: (i) *exploit* the knowledge that it has found for the current state s by doing one of the actions a that maximizes $Q(s, a, \theta)$ (line 12 of the Algorithm 2); (ii) *explore* in order to build a better estimate of the optimal q-function. That is, it should select a different action from the one that it currently thinks is the best (line 11 of the Algorithm 2). There have been a number of suggested ways to trade off between the exploration and exploitation. In this

Algorithm 2: Q-learning with neural networks-based navigation framework for autonomous mobile robots.

input : Laser data $X = (x_1, x_2, ..., x_N)$, learning rate α, discount factor γ, epsilon-greedy policy ϵ, robots pose, safety constraints.
output: $Q(s, a; \theta)$, state $s \in S$, action $a \in A$, weights θ

1 **begin**
2 Initialize replay memory D to capacity N
3 Initialize $Q(s, a; \theta)$ with random weights θ
4 Initialize $Q'(s, a'; \theta')$ with weights θ'=θ
5 **for** $episode = 1, M$ **do**
6 Randomly set the robots pose in the scenario
7 Observe initial states of robot s
8 **for** $t = 1, T$ **do**
9 Select an action a_t
10 with probability ϵ select a random action a_t
11 otherwise select $a_t = \arg\max_{a'} Q(s_t, a'; \theta)$
12 Execute action a_t, observe state s_{t+1}, compute reward R_t
13 Store transition (s_t, a_t, R_t, s_{t+1}) in replay memory D
14 Sample random minibatch of transitions (s_j, a_j, R_j, s_{j+1}) from D
15 Calculate the predicted value $Q(s_j, a_j; \theta)$
16 Calculate target value for each mimibatch transition
17 if s_{j+1} is terminal state then $y_j = R_j$
18 otherwise $y_j = R_j + \gamma \max_{a'_j} Q'(s'_j, a'_j; \theta')$
19 Train neural networks using $(y_j - Q(s_j, a_j; \theta))^2$ as loss function
20 **end for**
21 **end for**

paper, the $epsilon - greedy$ policy ϵ is used to select the greedy action (one that maximizes $Q(s, a, \theta)$), where $0 < \epsilon < 1$. It is possible to change ϵ value through time. Intuitively, early in the life of the robot it should select a more random strategy to encourage initial exploration and, as time progresses, it should act more greedily.

In order for the mobile robot to perform well in long-term, it need to take into account not only the immediate rewards, but also the awards it is going to get in the future. In addition, because environments are stochastic, the robot can never be sure, it will get the same rewards the next time when it perform the same actions. The more into the future the robot moves, the more it may diverge. For that reason, in this study, we use a future discounted reward. The future discounted factor R_t (line 13 of the Algorithm 2) return at time t is defined as follows

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... + \gamma^{T-t} r_T, \quad 0 \leq \gamma \leq 1 \tag{2}$$

where, r_t is the immediate reward, and T is the time-step at which the robot action terminates. Using Eq. 2, the more into the future the reward is, the less the robot takes it into consideration. If we set the discount factor $\gamma = 0$, then our strategy will be short-sighted

and we rely only on the immediate reward. If we want to balance between immediate and future rewards, we should set discount factor to something like $\gamma = 0.9$. If our environment is deterministic and the same actions always result in same rewards, then we can set discount factor $\gamma = 1$. The goal of the robot is to interact with the environment by selecting actions in a way that maximizes future rewards. R_t indicates how well the robot is doing at step t. The robot's job is to maximise cumulative reward reinforcement learning based on the reward hypothesis.

It has been known that, the type of correlation between successive sample is bad news for any iterative optimizer. In order to solve this issue, the experience replay [13] is utilized. The main idea of this technique is that, we simply keep old events in a memory and replay them back as training examples in a random order. To do that, during the robot navigation all the experiences (s_t, a_t, R_t, s_{t+1}) are stored in a replay memory D (line 14 of the Algorithm 2). In the training process of the neural networks, random samples from the replay memory are used instead of the most recent transition (line 15 of the Algorithm 2). This breaks the similarity of subsequent training samples, which otherwise might drive the network into a local minimum. Also experience replay makes the training task more similar to usual supervised learning, which simplifies debugging and testing the algorithm. One could actually collect all those experiences from human expert and train the neural networks on these.

For updating of weights θ of the neural networks, we first sample random minibatch of transitions from replay memory D (line 15 of the Algorithm 2), with the minibatch size is set to be N. For each given minibatch transition (s_j, a_j, R_j, s_{j+1}), the algorithm does the following steps: (1) do a feedforward pass the neural networks for the current state s_j to get predicted value $Q(s_j, a_j; \theta)$ (line 16 of the Algorithm 2); (2) if the sampled transition is a collision sample, the evaluation for this (s_j, a_j) pair is directly set as the termination reward (line 18 of the Algorithm 2). Otherwise, do a feedforward pass the neural networks for the next state s', calculate maximum overall network outputs $\max_{a'_j} Q'(s'_j, a'_j; \theta')$, and compute the target for action using the *Bellman* equation $(r + \max_{a'_j} Q'(s'_j, a'_j; \theta'))$ (line 19 of the Algorithm 2). For all other actions, set the target value to the same as originally returned from step 1, making the error 0 for those others. (3) Finally, the loss function (squared error) is defined as follows

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{n} (y_j - Q(x_j, a_j; \theta))^2. \tag{3}$$

Using the loss function $L(\theta)$, the weights θ of the neural networks will be updated through back-propagation and stochastic gradient descent (line 20 of the Algorithm 2). When the training process is completed, the mobile robot will save the trained neural networks into it brain and will use it in the future testing and working processes.

3. SIMULATION

In order to demonstrate the effectiveness of the proposed mobile robot navigation framework, in this section, we conduct experiments in a simulated environment.

3.1. Simulation setup

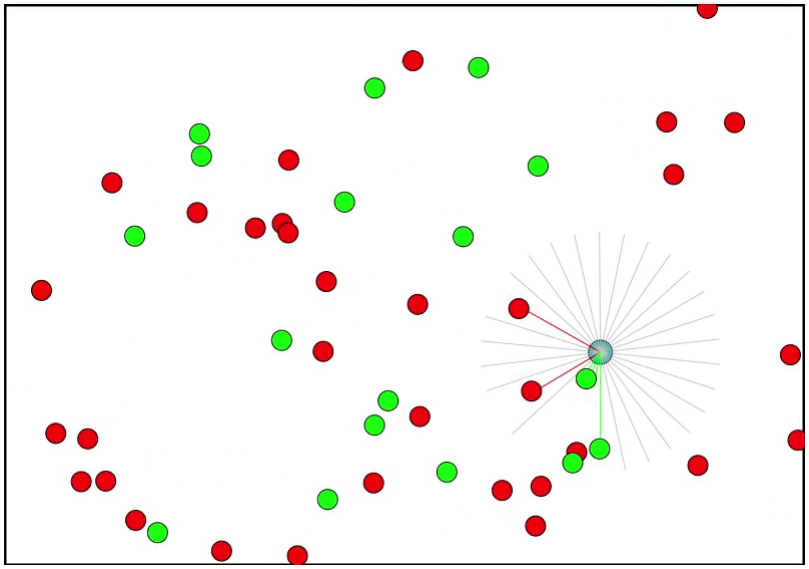

Figure 3. A simulated office-like scenario with walls, dynamic obstacles (red circles), dynamic targets (blue circles), and a mobile robot. The mobile robot is equipped with a laser rangfinder, providing angular field of view of 360^o, and requested to approach the dynamic targets while avoiding dynamic obstacles.

To narrow the gap between the simulated and real-world environments we have created a simulated office-like scenario with walls, dynamic obstacles (red circles), dynamic targets (blue circles), and a mobile robot for testing the proposed navigation framework, as shown in Fig. 3. The mobile robot is requested to catch up dynamic targets while avoiding dynamic obstacles.

3.1.1. Mobile robot model

In this paper, we chose a mobile robot model, which can only do four possible actions, including move to the left, right, forward or backward. The obstacles and targets bounce around the scenario. The mobile robot is equipped with a simulated laser rangfinder, providing the angular field of view of 360°, and the resolution of 12°. In other words, the mobile robot has 30 eyes pointing out in all directions. We assume that, in each direction the robot can observes 5 variables in its vicinity: (1) the range from the robot to the walls, obstacles and targets; the type of sensed objects including (2) targets, and (3) obstacles; and the velocity of the sensed objects including (4) targets, and (5) obstacles. In addition, the mobile robot's proprioception includes two additional sensors for its own speed in both directions v_x and v_y. Therefore, there is a total of $(30 \times 5) + 2 = 152$ dimensional state spaces, which is very high-dimensional. This indicates that, using a neural networks model-based function approximation to approximate the $Q(s, a)$ value is an appropriate choice.

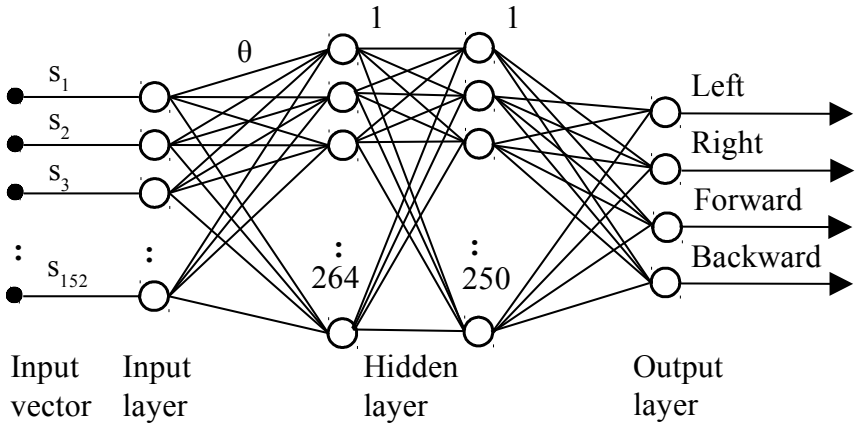

Figure 4. The topology of the used back-propagation neural networks. The inputs are the states of the robot, the outputs are the q-value corresponding to each possible action of the mobile robot.

3.1.2. Neural network settings

In this study, a back-propagation neural networks (BPNN) is used for the function approximation. The key of the back-propagation is a method for calculating the gradient of the error with respect to the weights for a given input by propagating error backwards through the network. More detailed information of the training process is available in [12]. In the previous section, the dimension of the state spaces of the mobile robot is 152. This vector is used as an input vector of the BPNN. The dimension of BPNN output layer is 4, one for each of the possible actions of the mobile robot (forward, backward, left, right). By guesswork and experience, we choose the topology of BPNN to be 152-264-250-4, as shown in Fig. 4. The activation function, which is used in our proposed method, is log-sigmoid. Thus, the output value of the network is constrained between 0 and 1.

When the training phase is completed, we save the trained model as a brain of the mobile robot. Then, whenever we distribute the trained mobile robot in a dynamic environment, it will load this trained model and utilize it to navigate in the environment. Using the trained model, the mobile robot will choose a appropriate action, which is corresponding to the maximum value of the output of the back-propagation neural networks. In addition, in order to evaluate the performance of the proposed mobile robot navigation framework, the collision index proposed by Truong et al. [20] is used. The immediate reward awarded to the mobile robot is +1 for catching up a target and -1 for making contact with any obstacle. The parameters of the Algorithm 2 are set including the discount factor $\gamma = 0.9$, the epsilon-greedy policy $\epsilon = 0.2$, and the learning rate $\alpha = 0.005$.

3.2. Simulation results

In this section, using the trained model, we conduct two experiments in two different scenarios to verify the performance of the proposed mobile robot navigation framework. The simulation results are shown in Fig. 5.

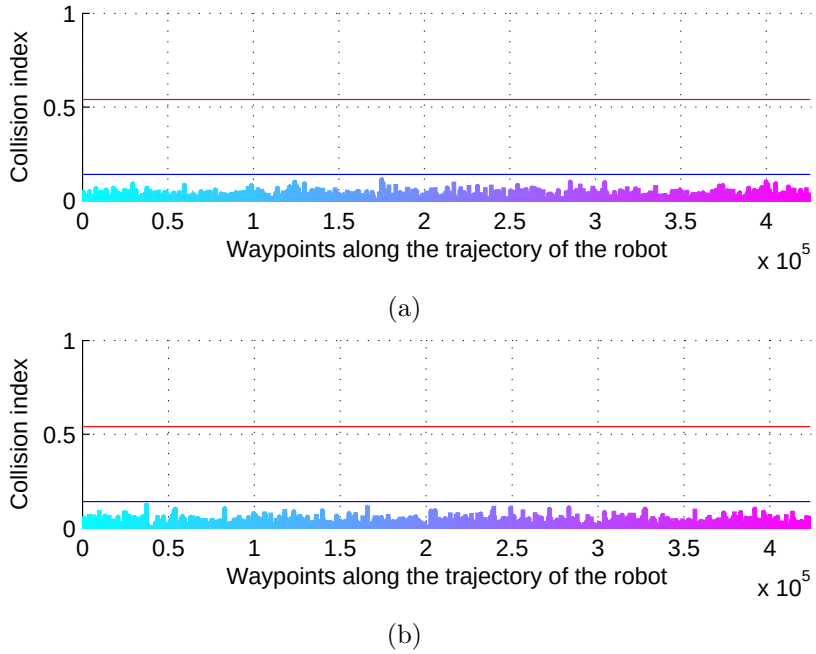

Figure 5. Simulation results of the proposed mobile robot navigation framework in two different scenarios: (a) the stationary scenario, and (b) the dynamic scenario.

3.2.1. Experiment 1 – stationary scenario

In the first experiment, a mobile robot and sixty targets and obstacles randomly distributed in the scenario. The robot is equipped with our proposed navigation framework, but the targets and obstacles are stationary. As can be seen in the video, the mobile robot is able to catch up the static targets (green circles) and avoid the static obstacles (red circles). In addition, Fig. 5(a) indicates that, the mobile robot always keeps a safety distance to the stationary obstacles. In other words, the mobile robot is capable of autonomously and safely navigating in the stationary environment.

3.2.2. Experiment 2 – dynamic scenario

In this experiment, a mobile robot, and fifty targets and obstacles are randomly distributed in the scenario. The mobile robot is equipped with our proposed navigation framework, however the targets and obstacles are randomly move around the scenario. As can be seen in the video, the mobile robot is capable of catching up the dynamic targets and avoiding the moving obstacles. Moreover, Fig. 5(b) shows that, the mobile robot always keeps a safety distance to the dynamic obstacles in the robots vicinity. In other words, the mobile robot is capable of autonomously and safely navigating in the dynamic environment.

Overall, the proposed efficient navigation framework enables an autonomous mobile robot to catch up dynamic targets and avoid moving obstacles in the dynamic environments, providing the safety for the mobile robot and the surrounding objects.

4. CONCLUSION

We have presented an efficient navigation framework for autonomous mobile robots in dynamic environments using q-learning algorithm with neural networks. The main idea of the proposed algorithm is to provide the mobile robots the relative position and motion of the obstacles to the robots, the safety constraints such as minimum distance from the robot to the obstacles, and a learning model. We then distribute the mobile robots into a dynamic environment. The robots will automatically learn to adapt to the environment by their own experienced through the trial-and-error interaction with the surrounding environment. The simulation results in a simulated environment show that, our proposed framework is capable of guiding the mobile robots to navigate autonomously and safely in the dynamic environments.

In the future, we will verify our proposed navigation framework with dynamic actions of mobile robot (the actions relative to the dynamic model of the mobile robot), instead of using only four discrete actions. In addition, we will implement the proposed framework in our mobile robot platform for real-world experiments. Furthermore, deep neural networks [14], deep reinforcement learning techniques [9], and continuous actions (acceleration and steering) [3] should also be considered to improve the learning efficiency and navigation task of the mobile robot.

REFERENCES

[1] M. Bennewitz, *Mobile Robot Navigation in Dynamic Environments.* Germany: PhD Thesis, Department of Computer Science, University of Freiburg, 2004.

[2] B. Daman and J. Van den Berg, "Generalized reciprocal collision avoidance," *The International Journal of Robotics Research*, vol. 34, no. 12, pp. 1501–1514, 2015.

[3] S. Gu, T. P. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," *arXiv:1603.00748v1*, 2 Mar 2016. [Online]. Available: https://arxiv.org/abs/1603.00748

[4] K. Henrik, S. Markus, S. Christoph, and B. Wolfram, "Socially compliant mobile robot navigation via inverse reinforcement learning," *The International Journal of Robotics Research*, pp. 1289–1307, 2016.

[5] L. P. Kaelbling, M. L. Littman, and A. P. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[6] B. Kim and J. Pineau, "Socially adaptive path planning in human environments using inverse reinforcement learning," *International Journal of Social Robotics*, pp. 1–16, 2015.

[7] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, Sep. 2013.

[8] T. Kruse, A. K. Pandey, R. Alami, and A. Kirsch, "Human-aware robot navigation: A survey," *Robotics and Autonomous Systems*, vol. 61, pp. 1726–1743, 2013.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou,

H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236

[10] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, "Autonomous inverted helicopter flight via reinforcement learning," in *The 9th International Symposium on Experimental Robotics*, 2006, pp. 363–372.

[11] J. Rios-Martinez, A. Spalanzani, and C. Laugier, "From proxemics theory to socially-aware navigation: A survey," *International Journal of Social Robotics*, September 2014.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1, pp. 318–362, 1986.

[13] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv:1511.05952v4*, 25 Feb 2016. [Online]. Available: https://arxiv.org/abs/1603.00748

[14] j. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[15] M. Shiomi, F. Zanlungo, K. Hayashi, and T. Kanda, "Towards a socially acceptable collision avoidance for a mobile robot navigating among pedestrians using a pedestrian model," *International Journal of Social Robotics*, vol. 6, no. 3, pp. 443–455, 2014.

[16] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*. The MIT Press, February 2011.

[17] C. Sungjoon, K. Eunwoo, and O. Songhwai, "Real-time navigation in crowded dynamic environments using gaussian process motion control," in *IEEE International Conference on Robotics and Automation*, May 2014.

[18] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[19] P. Trautman, J. Ma, R. Murray, and A. Krause, "Robot navigation in dense human crowds: the case for cooperation," in *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2013, pp. 2153–2160.

[20] X. T. Truong and T. D. Ngo, "Dynamic social zone based mobile robot navigation for human comfortable safety in social environments," *International Journal of Social Robotics*, vol. 8, no. 5, pp. 663–684, 2016.

[21] D. Vasquez, B. Okal, and K. O. Arras, "Inverse reinforcement learning algorithms and features for robot navigation in crowds: An experimental comparison," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, September 2014, pp. 1341–1346.

[22] C. Xie, S. Patil, T. M. Moldovan, S. Levine, and P. Abbeel, "Model-based reinforcement learning with parametrized physical models and optimism-driven exploration," *arXiv:abs/1509.06824*, 2015. [Online]. Available: http://arxiv.org/abs/1509.06824

Received on May 17, 2017
Revised on December 03, 2017