Sensors and Materials, Vol. 33, No. 2 (2021) 825–842
MYU Tokyo

S & M 2498

Enhanced Autonomous Navigation of Robots by Deep Reinforcement Learning Algorithm with Multistep Method

Xiaohong Peng,[1] Rongfa Chen,[1*] Jun Zhang,[1] Bo Chen,[2]
Hsien-Wei Tseng,[3] Tung-Lung Wu,[3] and Teen-Hang Meen[4**]

[1]School of Mathematics and Computer Science, Guangdong Ocean University, Zhanjiang, 524088, China
[2]College of Information Engineering, Lingnan Normal University, Zhanjiang, 524048, China
[3]School of Mathematics and Information Engineering, Longyan University, Longyan, Fujian, 364012, China
[4]Department of Electronic Engineering, National Formosa University, Huwei 632, Yunlin, Taiwan

(Received July 27, 2020; accepted November 24, 2020)

Keywords: autonomous navigation, obstacle avoidance, deep reinforcement learning, navigation planning

In this paper, we propose a new method to improve the autonomous navigation of mobile robots. The new method combines a multistep update method with a double deep Q-network (MS-DDQN) to realize reinforcement learning (RL) to enhance the navigation ability of mobile robots. The proposed MS-DDQN gives two types of rewards for taking actions: terminal and non-terminal rewards. These rewards are subdivided into several different ones including rewards for arrival and collision (terminal rewards) and rewards for distance, orientation, and danger (non-terminal rewards). With the subdivided reward system, the new method trains mobile robots more effectively to increase their autonomous navigation ability. In the experimental process of this study, a laser range finder was used as the sensor for the mobile robot to perceive the distance information of the obstacle. Experiment results validated the new method's enhanced ability, showing higher success rates (97% on average) than those of other methods such as the double deep Q-network (DDQN), prioritized deep Q-network (DQN), and prioritized DDQN. The higher success rates originated from the sophisticated reward system as the total reward of the proposed method was 7–94% higher than those of the other methods in simulations in five different environments. The learning speed was also improved, reducing the learning time, as the new method completed the learning in fewer episodes. The results of the new model suggest that MS-DDQN enables mobile robots to have higher learning efficiency and generalization ability than conventional deep reinforcement (DRL)-based methods and allows them to navigate autonomously even in unknown complex environments.

1. Introduction

Autonomous navigation ability is essential for mobile robots. To improve the navigation ability, many navigation planning methods have been proposed for mobile robots that work effectively in different environments.[1,2] A representative method is the potential field method

*Corresponding author: e-mail: rongfachen@163.com
**Corresponding author: e-mail: thmeen@nfu.edu.tw
https://doi.org/10.18494/SAM.2021.3050

(PFM),[3] which has been widely used owing to its simplicity, high safety, and fast computation time. However, PFM has the following disadvantages: a robot is likely to fall into local minima, oscillate in narrow passages, and wander in the presence of obstacles and between closely spaced obstacles.[4] To overcome these disadvantages, Borenstein and Koren proposed a vector field histogram (VFH) algorithm that finds the best path of a robot in a locally established polar histogram. [5] VFH* and VFH$^+$ have also been developed as modifications of VFH.[6,7] Simultaneous localization and mapping (SLAM) is another navigation algorithm. SLAM needs a map with complete information,[8] which enables autonomous navigation with continuous constructions and updates of the map. The algorithm of SLAM mainly uses an extended Kalman filter[9] and a particle filter[10] to collect information in a working environment for mobile robots. The information of a working environment can be added using cameras in a SLAM algorithm.[11,12] Although SLAM performs well in various navigation tasks, it still needs a huge amount of memory and computation. Therefore, most SLAM algorithms are applied in static environments.

In many cases, mobile robots are required to work in environments without previous information. Thus, traditional navigation algorithms have limited applicability to mobile robots. Recently, the development of artificial intelligence (AI) has allowed supervised and self-supervised learning methods to be applied to mobile robots even for navigation planning. For example, Chen *et al.* used a deep neural network to train mobile robots for autonomous navigation in crowded environments,[13] and Pfeiffer *et al.* proposed an end-to-end autonomous navigation model based on a deep convolutional neural network (CNN).[14] The model with CNN used two-dimensional (2D) laser scanner data and the relative positions of targets to execute steering commands. Tai and Liu also used CNN to train an obstacle avoidance model in corridors for the end-to-end control of mobile robots. In this case, CNN inputs raw depth image data, and output discrete commands were used to control the robots.[15] These deep-learning navigation algorithms learn strategies using raw information in multiple dimensions. However, they also have shortcomings for real applications. For example, the performance of a robot with such an algorithm depends on training data sets that are collected manually.

Thus, reinforcement learning (RL) algorithms are widely used. RL enables mobile robots to directly learn about the interactive environment. However, it is challenging for mobile robots to learn a good strategy in a large space. In addition, although RL guides and trains mobile robots to learn navigation strategies in unknown environments, the strategy is effective only in a fixed time period in the environments. This limits the application of RL in a complex environment with multiple dimensions. Thus, many new technologies and architectures of RL have been developed to improve the efficiency and performance in various tasks, examples of which are hierarchical RL (HRL) and deep RL (DRL).[16–19]

HRL decomposes a complex main task into several subtasks. Then, through the strategy of 'divide and conquer', the subtasks are completed one by one to fulfill the main task. DRL combines the advantages of deep learning and RL, achieving remarkable successes in many challenging tasks. DRL uses CNN to process multidimensional raw information and approximate the value function of RL. DRL is divided into temporal-difference (TD) learning and Monte Carlo (MC) learning. In TD, a mobile robot updates the Q table of states every

time it interacts with the environment. TD includes the deep Q-network (DQN),[20] double deep Q-network (DDQN),[21] and prioritized DQN.[22] DQN uses CNN to directly process the multidimensional information of raw images and approximate the best value function.[23] Although TD has a high convergence speed and high learning efficiency, it is less stable than other learning algorithms and sometimes converges to an incorrect solution. In MC, a mobile robot creates the Q learning table of states after it has completed one episode of interaction with the environment. The policy gradient algorithm is an example of MC.[24] As MC needs to first complete an episode of interaction with the environment and then update the Q table, its convergence speed and learning efficiency are relatively low.

Most DRL algorithms use TD to perform the navigation planning of mobile robots. For example, Tai *et al.* proposed a DRL network trained by an asynchronous DRL algorithm that was based on input and output control commands for 10-dimensional distances measured by a laser.[25] Vinyals *et al.* designed an A3C network for rescue missions of a mobile robot.[26] Al-Nima *et al.* used human driving data as the input to a DRL network to train the autopilot ability of a vehicle.[27] TD learns rapidly in general, but it only calculates the reward from the next state, which restricts the ability of the mobile robot to perceive obstacles. MC improves a mobile robot's ability to perceive obstacles but the learning is still slow.

Therefore, we use a DRL algorithm to solve the autonomous navigation task of a mobile robot in an unknown environment. This method is a machine learning algorithm that combines the advantages of RL and deep learning algorithms. A mobile robot based on this method can directly process the original high-dimensional input information, and can also learn by directly interacting with the environment. In order to overcome the shortcomings of traditional DRL directly applied to autonomous tasks of a mobile robot and improve the ability of the mobile robot to quickly perceive obstacles, we propose a DRL algorithm using a multistep update method and a continuously combined reward function. On the basis of the method and the reward function, we use hierarchical DRL, where each layer of a neural network has a clear learning goal. The multistep method is different from TD and MC. Without reducing the training efficiency, the multistep method predicts the impact of multiple states in the future to obtain the reward in the current state. DRL with the multistep method is expected to enhance the navigation ability of mobile robots by improving their ability to sense and avoid obstacles in advance. The multistep method also ensures that mobile robots have high training and learning efficiencies.

In this study, we simulated obstacle environment information and a laser range finder (LRF), and the mobile robot measured the distance information of obstacles through the simulated LRF. Only 36-dimensional distance information in the forward direction of the mobile robot was used as the input information of the deep neural network, and then the deep neural network outputs control instructions for controlling the mobile robot. To validate the proposed method in this study, we carried out simulations and compared the results with those of other methods. It is expected that an autonomous navigation algorithm based on the results of this study will improve the autonomous navigation ability of mobile robots in unknown and complex environments.

2. Methods

2. 1 RL

The navigation problem of a mobile robot in an unknown environment can be expressed as an RL problem in which the mobile robot interacts with the environment E in a fixed time step. At each time step t, the mobile robot obtains the state information $s_t \in S$ of the environment through sensors, where S is the state space. The mobile robot selects an action a_t from all possible action sets A according to the acquired state information. After the action is completed, the mobile robot transitions to the next environment state s_{t+1} and gets a reward r_t. In this process, the state is altered according to the state transition probability $P(s_{t+1}/s_t, a_t)$. This defines the possibility with which the robot takes action a_t in state s_t and then transfers to state s_{t+1}. In the traditional RL algorithm, the return function G_t is defined as the sum of discounts obtained in state s_t and all rewards in the future states as follows:

$$G_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i} = r_t + \gamma G_{t+1}, \tag{1}$$

where $\gamma \in (0, 1)$ is a discount factor defining the impact of future rewards on the current state. The goal of the mobile robot is to find the best strategy to maximize the future rewards by training actions to take in the current state s_t that is a mapping function from the state sets S to the action sets A. Under a given strategy π, the state-action value function $Q^{\pi}(s, a)$ is the expected reward of the robot when it takes an action at a state with the expectation function $E_{\pi}(\cdot)$.

$$Q^{\pi}(s,a) = E_{\pi}\left[G_t / s_t = s, a_t = a\right] \tag{2}$$

Depending on the update method of the state-action value table, the RL algorithm is divided into the MC and TD methods. The MC method updates the state-action value table as

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots - Q(s_t,a_t)\right], \tag{3}$$

where $\alpha \in (0, 1)$ is the learning rate and the polynomial $(r_t + \gamma r_{t+1} + \gamma^2 r_{t+1} + ...)$ is equal to the return function G_t. Thus, $r_t + \gamma r_{t+1} + \gamma^2 r_{t+1} + ...$ is replaced by the return function G_t, and Eq. (3) is simplified as

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[G_t - Q(s_t,a_t)\right]. \tag{4}$$

The MC method requires the mobile robot to complete an episode of interaction with the environment before updating the state-action value table. If the mobile robot takes a long time to finish an episode of training, the update of the state-action value table becomes slow. Therefore, the training and learning efficiencies of the MC method are relatively low.

Unlike the MC method, the TD method only considers the impact of the reward obtained by the next state based on the current state's reward. The standard Q learning algorithm is an example of a TD method.[29,30] The TD method updates the state-action value table as

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[r_t + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t) \right], \tag{5}$$

where $\alpha \in (0, 1)$ is the learning rate.

When $G_t^{(1)} = r_t + \gamma \max_a Q(s_{t+1},a)$ and $G_t^{(1)}$ is defined as the return function of the TD method, Eq. (5) is transformed as

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[G_t^{(1)} - Q(s_t,a_t) \right]. \tag{6}$$

The return function of the TD method is approximated as the return function G_t. The rest of the return function G_t is $\gamma r_{t+1} + \gamma^2 r_{t+1} + ...$, which is replaced by $\gamma \max_a Q(s_{t+1},a)$ after time step t. The TD method does not need to update the state-action value table after one episode of interaction with the environment, but it updates the state-action value table at every step of training the robot. Therefore, the TD method has higher efficiencies in training and learning. However, as the TD method only considers the return of the next state, it does not predict the future returns, so it is relatively short-sighted.

The multistep method is different from the TD and MC methods and is one of the RL algorithms.[1,28] In some tasks, RL algorithms improve the performance of the TD algorithm and the learning efficiency of the MC algorithm by considering the reward of the next states. The multistep method is very similar to the TD method, and the only difference is the return function. The return function of the multistep method is defined as

$$G_t^{(n)} = \sum_{i=0}^{i=n-1} \gamma^i r_{t+i} + \gamma^n \max_a Q(s_{t+n},a). \tag{7}$$

The return function of the multistep method truncates the traditional return function G_t after n steps, and the remaining items are replaced by $\gamma^n \max_a Q(s_{t+1},a)$. The multistep method enables the mobile robot to overcome the limitation of a single time step and focus on the reward of a longer time interval. Therefore, the multistep method updates the state-action value table as

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[G_t^{(n)} - Q(s_t,a_t) \right]. \tag{8}$$

2. 2 DRL

When a mobile robot works in a very complex environment with a large number of states and actions, the traditional RL algorithm encounters a dimension problem. A good solution to this problem is to use a neural network to approximate the state-action value function in the

algorithm. Therefore, the state-action value function at this time is related not only to state s and action a, but also to the weight parameter θ in a neural network. A method combining deep neural networks and RL is called a DRL method. By optimizing the parameter θ in the network to approximate the state-action value function, the loss function of the neural network is defined as

$$L(\theta) = E(s,a,r,s')\left[\left(y^{Q} - Q(s,a;\theta)\right)^2\right], \tag{9}$$

where $y^{Q} = r + \gamma \max Q(s', a; \theta)$.

In practical applications, owing to the strong correlation between the collected state data, the RL algorithm does not converge or even diverge as it directly uses the approximate Q value function of the neural network.[29] To solve this problem, the DQN algorithm was proposed.[17] This algorithm uses a double neural network structure with an evaluation network and a target network. The parameter θ in the evaluation network is assigned randomly at the beginning, and the parameter θ' in the target network is copied periodically. Therefore, y^{Q} of the DQN algorithm is changed into

$$y^{DQN} = r + \gamma \max_{a} Q(s',a;\theta') . \tag{10}$$

To break the strong correlation between the training data, the DQN algorithm replays its experience. By storing the experience data $e_t = (s_t, a_t, r_t, s_{t+1})$ in the replay memory D, the mobile robot remembers and reuses the experience from the past. In the process of training, two tuples, $e_{t1} \in D$ and $e_{t2} \in D$, are weakly correlated due to random sampling. The random extraction of training data in small batches from D helps to break the strong correlation between training sample data and ensures the stability of the DRL system. Thus, the loss function of the DQN algorithm is defined as

$$L(\theta) = E_{(s,a,r,s')\sim U(D)}\left[\left(y^{DQN} - Q(s,a;\theta)\right)^2\right]. \tag{11}$$

In the DQN algorithm, the target network uses the MAX method to estimate the state-action value, which leads to an overestimation problem. The DDQN method solves this problem. The MAX operator in the target is broken down into two parts, and the parameter θ is used to select the action while θ' is used for the estimation of the state-action value function. Therefore, y^{DQN} of the DQN algorithm becomes y^{DDQN}, where

$$y^{DDQN} = r + \gamma Q\left(s', \arg\max Q(s',a,\theta);\theta'\right) . \tag{12}$$

2. 3 Proposed algorithm

We applied the multistep update method to DDQN to train a mobile robot for navigation planning. We named this method the multistep deep Q-network (MS-DDQN). By combining

Eqs. (7) and (12), y^{DDQN} is transformed into $y^{MS\text{-}DDQN}$ as follows.

$$y^{MS-DDQN} = \sum_{i=0}^{i=n-1} \gamma^i r_{t+i} + \gamma^n Q\left(s_{t+n}, \arg\max Q(s_{t+n}, a, \theta); \theta\right) \tag{13}$$

The loss function of the MS-DDQN method is defined as

$$L(\theta) = E_{(s,a,r,s_{t+n})\sim U(D)}\left[\left(y^{MS-DDQN} - Q(\mathrm{s}, a; \theta)\right)^2\right], \tag{14}$$

where $U(D)$ represents a function that randomly extracts the experience data from memory D, and the experience data stored in the replay memory becomes $e_t(s_t, a_t, r_t, s_{t+n})$. The algorithm of the MS-DDQN training method is as follows.

Algorithm 1: MS-DDQN

Initialize experience replay memory D
Initialize parameters θ randomly
Initialize state-action value function Q'
Initialize parameters of target network $\theta' = \theta$
Initialize hyperparameter n
For episode = 1 to M do
Set/reset simulated environment
Observe s_t
$T \leftarrow \infty$
Reset four empty arrays S_t, A_t, R_t, S_{t+1}
For t = 1, 2, ... do
If $t < T$, then
With probability ε select a random action a_t, otherwise select
a_t = arg max $Q'(s_t, a; \theta)$
Execute action a_t in emulator and observe reward r_t and s_{t+1}
Store s_t in S_t, store r_t in R_t, store a_t in A_t, store s_{t+1} in S_{t+1}
If s_{t+1} is terminal, then
$T \leftarrow t + 1$
$\tau \leftarrow t - n + 1$
If $\tau \geq 0$, then
If $\tau + n < T$, then

$$r_\tau = \sum_{i=\tau}^{i=\tau+n-1} \gamma^{i-\tau} r_i \qquad r_i \in R_t$$

Else

$$r_\tau = \sum_{i=\tau}^{i=T} \gamma^{i-\tau} r_i \qquad r_i \in R_t$$

Store transition $(s_\tau, a_\tau, r_\tau, s_{\tau+n})$ in D, $s_\tau \in S, a_\tau \in A_t, s_{t+n} \in S_{t+1}$
Sample random mini-batch of transition $(s_\tau, a_\tau, r_\tau, s_{\tau+n})$ from D
Set $y_i = r_i + \gamma^n Q(s_{i+n}, \arg\max Q(s_{i+n}, a; \theta); \theta)$
Perform a gradient descent step on $(y_i - Q'(s_t, a_t; \theta))^2$ with respect to network parameter θ
Until $\tau = T - 1$
End

2. 4 Hierarchical DRL framework

To find the best navigation planning in a complex environment, we adopted a 'divide and conquer' strategy. In this study, we adopted a hierarchical DRL framework, and we divided a navigation task into two submodules: an avoidance module and navigation module. The avoidance module guides a mobile robot to avoid obstacles. With the input of distance data by a laser scan and the relative positions of the mobile robot, the module outputs a discrete command to control the robot. The navigation module guides the mobile robot to reach the target position faster. It also outputs a discrete command with the input information of the relative positions of the mobile robot. Either of the two modules finds the nearest distance between the mobile robot and the obstacles. The hierarchical DRL framework we proposed is shown in Fig. 1. The deep neural network architecture of the avoidance module and the global navigation module is shown in Fig. 2.

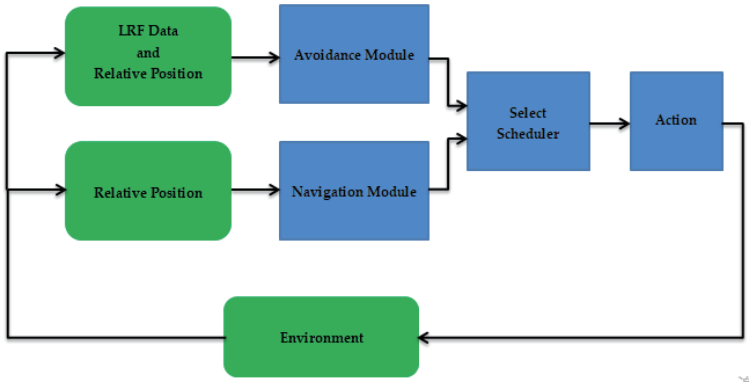


Fig. 1. (Color online) DRL framework in this study. (Blue boxes are for internal components and green boxes are the external environment).

Avoidance Module

LRF Date	Fully	S1	Fully	S2	Fully	S3	Fully	Action
and	Connect	256	Connect	128	Connect	32	Connect	5
Relative position	ReLU	dim	ReLU	dim	ReLU	dim	ReLU	dim

Navigation Module

	Fully	S1	Fully	Action
Relative position	Connect	32	Connect	5
	ReLU	dim	ReLU	dim

Fig. 2. (Color online) Deep neural network structure of obstacle avoidance module and global navigation module. The deep neural network of the obstacle avoidance module consists of three hidden layers containing 256, 128, and 32 neurons. The global navigation module contains only one hidden layer containing 30 neurons. These two modules both output the commands to control the mobile robot.

2. 5 Reward function

The reward function gives the reward value of the mobile robot moving from the current state to the next state and indicates how well the action is taken in the current state. Generally, the reward function of the RL method gives 0 for a failed action and 1 for a completed action. This simple reward function provides sparse rewards, which are not conducive to the convergence of the algorithm. To solve this problem and accelerate the convergence of the algorithm, a new combined reward function is needed.

We divide the rewards into two types, depending on the end of the current training episode. The first reward is a 'terminal reward', which is given when the mobile robot reaches a target position or collides with an obstacle. The second reward is a 'non-terminal reward', which is given when the mobile robot is moving towards a target position. The terminal reward is divided into an arrival reward and a collision reward.

When the mobile robot reaches the target position, the reward given is

$$r_{arr} = 100; \text{ if } d_{r\text{-}t} \le d_{win}, \tag{15}$$

where $d_{r\text{-}t}$ is the Euclidean distance from the mobile robot to the target position and d_{win} is the threshold of the mobile robot's collision with the obstacle.

When a mobile robot collides with an obstacle, the reward becomes

$$r_{col} = -100; \text{ if } d_{r\text{-}o} \le d_{col}, \tag{16}$$

where $d_{r\text{-}o}$ is the Euclidean distance between the mobile robot and the nearest obstacle and d_{col} is the threshold of the mobile robot's collision with the obstacle. Thus, the terminal reward is $r_{arr} + r_{col}$.

Non-terminal rewards consist of three types: a positive reward, a danger reward, and an orientation reward.

When the mobile robot moves towards the target position, it gets a positive reward of

$$r_{t\_goal} = c_r[d_{r\text{-}t}(t) - d_{r\text{-}t}(t-1)], \tag{17}$$

where $C_r \in (0, 1)$ is a coefficient. This reward guides the mobile robot toward the target position.

As the distance of the mobile robot from obstacles becomes shorter, the danger reward decreases and is defined as

$$r_{dang} = \beta * 2^{d_{min}}, 0 \le r_{dang} \le 1, \tag{18}$$

where d_{min} is the distance of the mobile robot from the obstacle.

In addition, we designed an 'orientation reward'. The orientation reward is given according to the angle between the vector of the mobile robot's forward direction and the vector from the current coordinate of the mobile robot to the coordinate of the target position. When the

angle is greater than 18° and less than 72°, the reward is 0.3. In other cases, the reward is 0. The orientation reward is defined as

$$r_{ori} = \begin{cases} 1 & \text{if } a_{ori} \le \pm 18° \\ 0.3 & \text{if } 18° \le a_{ori} \le 72° \\ 0.3 & \text{if } -72° \le a_{ori} \le -18° \\ 0 & \text{otherwise} \end{cases}, \tag{19}$$

where a_{ori} is the angle between the vector of the mobile robot's forward direction and the vector from the current coordinate of the mobile robot to the coordinate of the target position. The final non-terminal reward is defined as $r_{t\text{-}goal} + r_{dang} + r_{ori}$.

This combination of the rewards solves the problem of sparsity so that the mobile robot gets corresponding rewards at each step of the training process. The combination also enables the mobile robot to learn a strategy allowing it to reach the target position faster along a shorter path.

3. Results and Discussion

3.1 Simulated environment

To demonstrate and evaluate the proposed method in this study, a 2D environment for simulations was designed as shown in Fig. 3. In Fig. 3, gray ellipses, circles, and polygons represent various types of obstacles. The red circle represents the mobile robot and blue lines represent laser beams. When the mobile robot starts a new episode of training, its starting position is randomly distributed in the light yellow rectangular area at the bottom of Fig. 3, marked with an orange circle. The target positions are randomly distributed in the light cyan rectangle at the top of Fig. 3, marked with a black circle. This layout ensures that the mobile robot must pass through a large number of obstacles and safely avoid them before reaching the

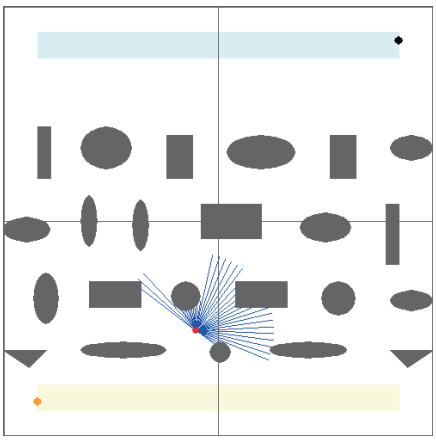


Fig. 3. (Color online) Training environment (Env-1) of size 500 × 500.

target position, and avoid invalid training due to a too short distance. The size of the training environment (Env-1) is 500 × 500.

The mobile robot senses its surroundings through a simulated LRF with a field of view (FOV) of 180°. Distance is measured by the mobile robot with an angular resolution of 5°. The output of the simulated LRF is a 36-dimensional vector o_i, which is normalized by the maximum effective range o_{max}. The first element of o_t always indicates a measurement in the horizontal left direction, followed by a measurement in the clockwise direction. In addition, the mobile robot also obtains the relative 2D coordinate information p_t of the robot's current position relative to the target position, as well as the Euclidean distance information Ω_t of the current position relative to the target position, and the relative angle information A_t of the target point relative to the robot's forward direction. The robot moves at a constant speed. The command set for controlling the mobile robot is composed of five discrete commands: turn left, turn left 30°, forward, turn right 15°, and turn right 30°. In the 2D environment, the mobile robot needs to continuously sense the surrounding environment to avoid obstacles on the way to the target position.

3.2 Training results

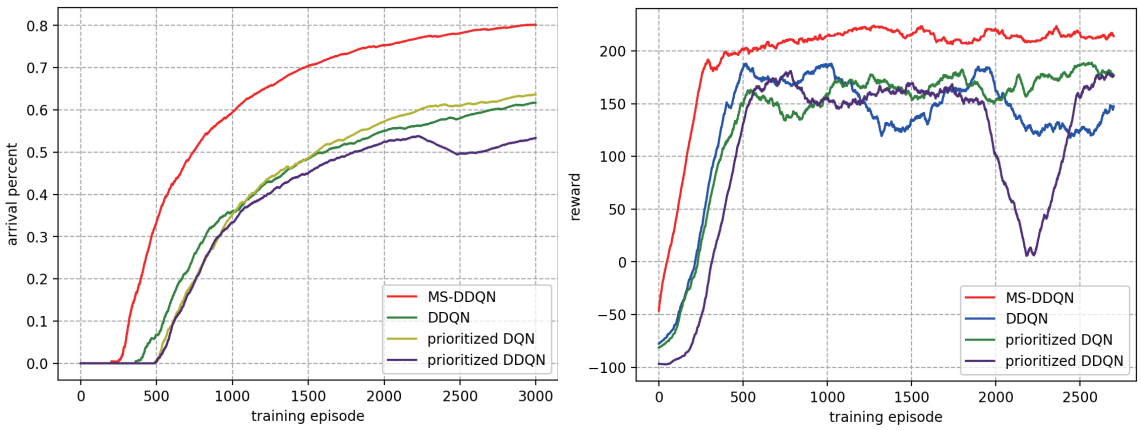
MS-DDQN trained the autonomous navigation capabilities of the mobile robot in the environment Env-1. During the training process, the starting position of the mobile robot and the target position were randomly initialized at the beginning of each training episode. When the mobile robot collided with an obstacle or reached the target position, a new episode of training started. The network models were built with TensorFlow and implemented on a single GIGABYTE 2070 GPU. The simulated environment was run on an Intel i7-6800k CPU. The training parameters are given in Table 1.

To verify the effectiveness of the multistep DRL method, we compared the MS-DDQN algorithm with the DDQN, prioritized DQN, and prioritized DDQN algorithms in terms of training the navigation ability of the mobile robot in Env-1. We used the same network structure and the same software and hardware platforms for the training. The success rate indicates the probability of the mobile robot successfully reaching the target position. Reward curves represent the sum of the rewards obtained by the mobile robot in each episode of the training. We used a sliding average method to process the curves with a sliding window size of 300. The average reward is the mean of the rewards for the mobile robot in 3000 episodes.

Table 1
Training parameters.

Parameter	Value
Learning rate	0.001
Discount factor	0.9
Replay buffer size	15000
Mini-batch size	32
Number of steps n	5
Replacement	300

Figures 4(a) and 4(b) show that the success rate curve of MS-DDQN rises faster than those of the other three methods, indicating that MS-DDQN has the highest learning efficiency. After 3000 episodes of training, MS-DDQN has a much higher success rate than the other algorithms. The success rate with MS-DDQN is 80.133%, while those with DDQN, prioritized DQN, and prioritized DDQN are 61.7, 63.633, and 53.366%, respectively. This indicates that the mobile robot trained by MS-DDQN avoided obstacles far better owing to its improved navigation capabilities. The average reward value of MS-DDQN is 185.072, while those of DDQN, prioritized DQN, and prioritized DDQN are 130.064, 132.067, and 101.650, respectively. This also proves that the mobile robot with MS-DDQN had stronger navigation capabilities. A lower reward value means many negative rewards, implying that the mobile robot had more collisions. Figure 4(b) shows that the reward with MS-DDQN remained above 200 after 500 episodes of training, while that with the other methods showed greater fluctuation. The navigation model learned by MD-DDQN has higher stability than the other methods.



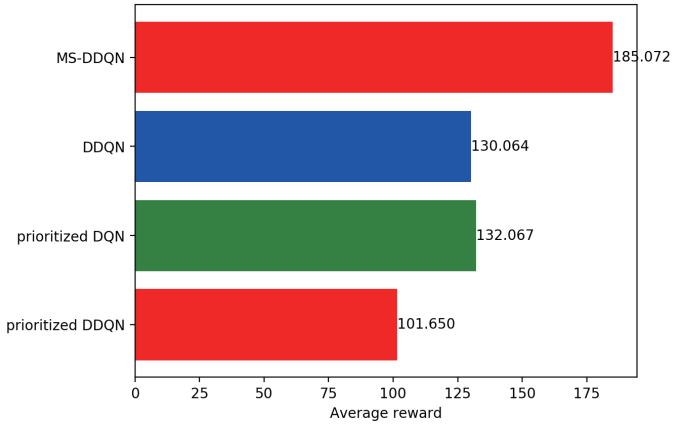


Fig. 4. (Color online) (a) Success rate of the mobile robot reaching the target position during the training process, (b) total reward obtained by the mobile robot per episode of training, and (c) average reward obtained after the mobile robot has performed 3000 episodes of training.

3.3 Test results

The navigation model was tested 200 times in Env-1. In each test, the starting position and target position of the mobile robot were randomly assigned. The performance of each algorithm was measured on the basis of the success rate and the average reward of the mobile robot reaching the target position in the 200 episodes. A higher success rate and average reward indicate that the navigation model provides a better strategy. The results are shown in Table 2. After 3000 episodes of training, the mobile robot trained by the four algorithms in the table had learned how to avoid obstacles and reach the target position in Env-1. MS-DDQN had a success rate of 100% and the highest average reward. The navigation trajectory of the MS-DDQN-trained mobile robot is shown in Fig. 5.

To evaluate the performance of the proposed method in this study, we designed four test environments that differed from the training environment. The sizes of the four test environments were 500 × 500 (Env-2), 600 × 600 (Env-3), 700 × 700 (Env-4), and 800 × 800 (Env-5). In these environments, the starting and target positions of the mobile robot were randomly initialized (light yellow and light cyan shaded areas in Fig. 6, respectively). The navigation models based on MS-DDQN, DDQN, prioritized DQN, and prioritized DDQN were tested 200 times in each of the four test environments. The navigation trajectories of the mobile robot with MS-DDQN in the different environments (Fig. 6) show that the navigation model trained by MS-DDQN had high generalization ability, adapting well to the new unknown environments.

Table 2
Results of MS-DDQN, DDQN, prioritized DQN, and prioritized DDQN on unseen test environment Env-1.

Environment	Approach	Success rate (%)	Average reward
Env-1	MS-DDQN	100	243.996
	DDQN	88	203.829
	prioritized DQN	94	228.829
	prioritized DDQN	91	215.293

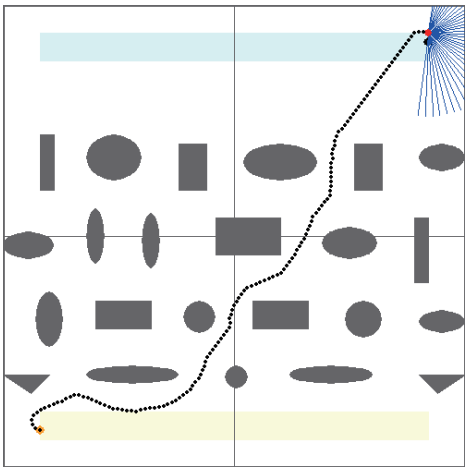


Fig. 5. (Color online) Navigation trajectory of the mobile robot in Env-1.

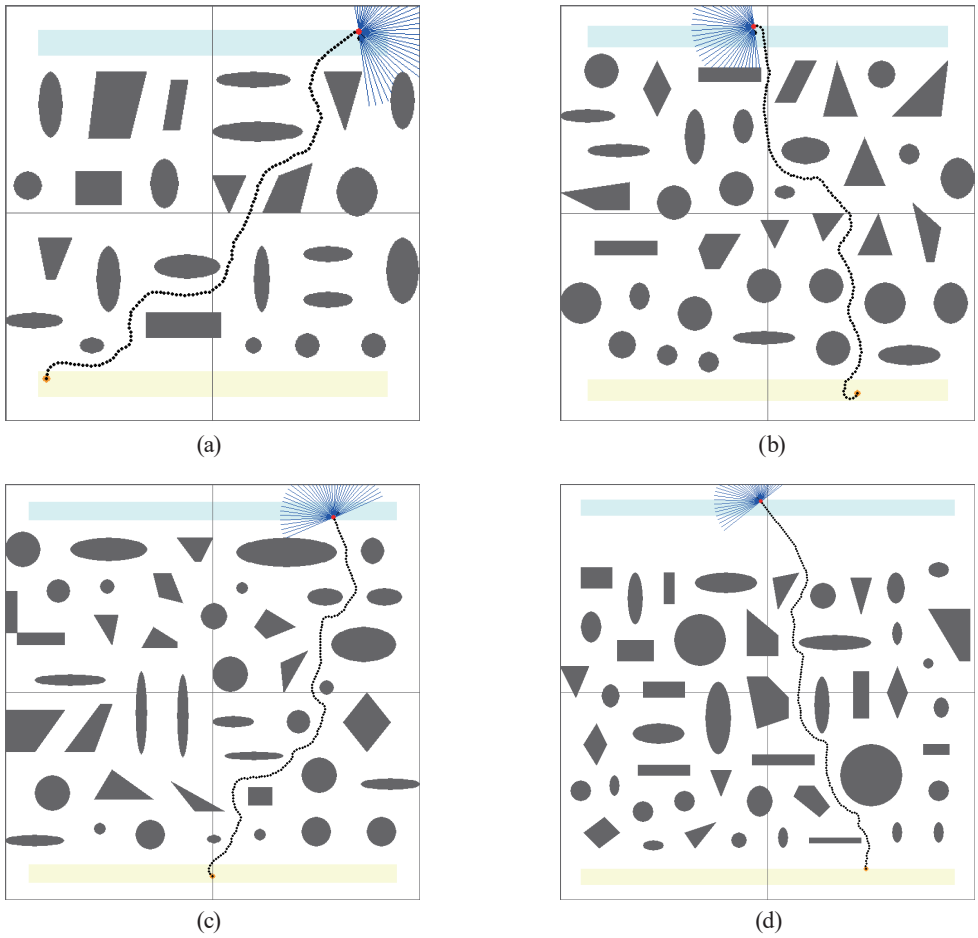


Fig. 6. (Color online) Navigation trajectories of MS-DDQN-based mobile robot in four different test environments. (a) Env-2, (b) Env-3, (c) Env-4, and (d) Env-5.

According to Table 3, the success rates of the navigation model trained by MS-DDQN were 97% (Env-2), 91% (Env-3), 94% (Env-4), and 96% (Env-5). However, the navigation models trained by the other three algorithms did not reach a success rate of 90% in the test environments. The success rate of DDQN in Env-3 was only 46%. This demonstrates that the MS-DDQN-based navigation model successfully solved the navigation problem of the mobile robot in a new environment. The test results also confirm that the navigation strategy with MS-DDQN is more stable than the other methods. The generalization ability of prioritized DDQN and prioritized DQN was better than that of DDQN because the two methods performed targeted training and learning on collision training samples during the training process.

The above experiments revealed that the performance of the navigation model trained by MS-DDQN was better than that of the navigation models trained by DDQN, prioritized DQN, and prioritized DDQN. The reason is that MS-DDQN integrated the multistep update method in the RL into DDQN. When training and learning, MS-DDQN calculates the impact of the rewards obtained in the next few steps on the current state. Therefore, the mobile robot

Table 3
Results of MS-DDQN, DDQN, prioritized DQN, and prioritized DDQN on unseen test environments Env-2, Env-3, Env-4, and Env-5.

Environment	Method	Success rate (%)	Average reward
Env-2	MS-DDQN	97	223.306
	DDQN	58.5	126.012
	prioritized DQN	61.5	140.081
	prioritized DDQN	86.5	189.762
Env-3	MS-DDQN	91	239.777
	DDQN	46	123.682
	prioritized DQN	64.5	173.851
	prioritized DDQN	82	209.123
Env-4	MS-DDQN	94	281.952
	DDQN	51	167.767
	prioritized DQN	63	211.320
	prioritized DDQN	78.5	232.045
Env-5	MS-DDQN	96	333.441
	DDQN	69.5	243.474
	prioritized DQN	76.5	280.121
	prioritized DDQN	76.5	260.025

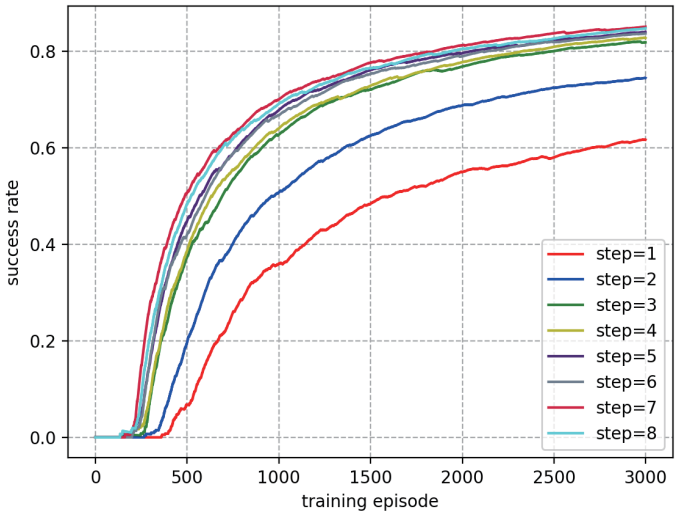


Fig. 7. (Color online) Success rate of the model reaching the target position for different n values.

navigation model trained by this method has greater ability to sense obstacles, and the mobile robot can devise avoidance strategies in advance.

In MS-DDQN, the number of steps n is a sensitive hyperparameter. We examined how different values of n affected the performance with MS-DDQN. The larger the value of n, the greater the impact of future rewards. We conducted 3000 episodes of training for $n \in (1, 8)$ in Env-1. The training results are shown in Fig. 7. The experimental results show that a larger n has a higher learning efficiency with MS-DDQN. When $n = 1$, MS-DDQN is degraded to DDQN, and the learning efficiency is the lowest. However, the success rates of $n = 3$, 4, and 5 are

similar, which indicates that the model with MS-DDQN already has the best strategy to avoid obstacles in advance with $n = 3$. Thus, even though increasing n further increases the success rate, the degree of improvement is not high. For example, when $n = 6$, 7, and 8, the difference in the success rate is less than 3% from that when $n = 5$. Moreover, as n further increases, MS-DDQN gradually degenerates into the MC method. Therefore, we set n as 5 in this study with MS-DDQN.

3.4 Discussion

There are three novelties in this study as follows:

1. We propose the MS-DDQN algorithm, which combines the multistep update method in RL with the DDQN algorithm. This method was applied for the first time to solve the problem of the autonomous navigation of mobile robots in unknown environments. This method can enable a mobile robot to perceive obstacles in advance, thereby improving its autonomous obstacle avoidance and navigation capabilities.
2. We construct two neural network structures, the local obstacle avoidance neural network structure and the global navigation neural network structure. The advantage of the DRL architecture is that each deep neural network has a clear training/learning goal. Local obstacle avoidance is mainly used by the mobile robot to avoid obstacles at a static distance, and global navigation is mainly used to quickly reach the target position.
3. We propose a new combined reward function that is divided into terminal rewards and non-terminal rewards. Terminal rewards include rewards for reaching the target position and rewards for collisions; non-terminal rewards include a positive reward, a danger reward, and an orientation reward. Through the combined reward function, not only can the convergence speed of the DRL algorithm be increased, but also the mobile robot can be guided to learn the strategy of reaching the target position with a shorter path. In Sect. 3.2, we compared the proposed MS-DDQN algorithm with the DDQN,[17] prioritized DQN,[20] and prioritized DDQN algorithms. It was found that MS-DDQS has higher learning efficiency as well as greater ability to avoid obstacles.

4. Conclusions

To solve the navigation problem of a mobile robot in an unknown environment, we proposed a multistep DRL method (MS-DDQN) that applies a multistep method of RL to the DDQN network. In the process of training and learning, we tested the subdivided reward system comprising terminal rewards (arrival, collision) and non-terminal rewards (distance, orientation, danger). With these rewards, the mobile robot was able to learn and improve its navigation ability. The learning efficiency of MS-DDQN in this study was higher than that of other methods with a success rate in the training of 100%, while the success rates of DQN, prioritized DDQN, and prioritized DQN were 88, 94, and 91%, respectively. In four test environments that were completely different from the training environment, the success rate of MS-DDQN was 91–97%, compared with 46–70% for DDQN, 62–77% for prioritized DQN, and 77–87% for

prioritized DDQN. The well-established reward system enabled such results as the total reward of MS-DDQN was 7–94% higher than those of the other methods. The learning speed was also higher, as MS-DDQN achieved the highest reward with the fewest episodes and just five steps. This proves that MS-DDQN has high generalization ability, stability, and efficiency, and a mobile robot using the method can perform real-time autonomous navigation in an unknown environment with only 36-dimensional laser detection distance information. In future work, we will consider using a deterministic policy gradient algorithm to improve the algorithm in this study and solve more general and realistic navigation problems such as navigation in a continuous motion space and multi-mobile robot navigation control.

References

1 P. A. Veatch and L.S. Davis: Comput. Gr. Image Process. **50** (1990) 50. https://www.sciencedirect.com/science/article/abs/pii/0734189X90900676?via%3Dihub
2 S. S. Ge and Y.J. Cui: Auton. Robots **13** (2002) 207. https://link.springer.com/article/10.1023/A:1020564024509
3 O. Khatib: Proc. 1985 IEEE Int. Conf. Robotics and Automation (IEEE, 1985) 90–98. https://doi.org/10.1109/ROBOT.1985.1087247
4 S. E. Nadira, R. Omar, and C.K.N.A. Hailma: ARPN J. Eng. Appl. Sci. **11** (2016) 10801. https://www.researchgate.net/publication/313389747_Potential_field_methods_and_their_inherent_approaches_for_path_planning
5 J. Borenstein and Y. Koren: IEEE Trans. Robot. **7** (1991) 278. https://ieeexplore.ieee.org/document/88137
6 I. Ulrich and J. Borenstein: Proc. IEEE Int. Conf. Robotics and Automation (IEEE, 1998) 1572–1577. https://doi.org/10.1109/ROBOT.1998.677362
7 I. Ulrich and J. Borenstein: Proc. IEEE Int. Conf. Robotics and Automation (IEEE, 2000) 2505–2511. https://doi.org/10.1109/ROBOT.2000.846405
8 H. F. Durrant-Whyte and T. Bailey: IEEE Robot. Autom. Mag. **13** (2006) 99. https://ieeexplore.ieee.org/document/8202312
9 M. G. Dissanayake, P. Newman, S. Clark H. F. Durrant-Whyte, and M. Csorba: IEEE Trans. Robot. **17** (20010) 229–241. https://doi.org/10.1109/70.938381
10 M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. Proc. 2002 AAAI National Conf. Artificial Intelligence (AAAI, 2002) 593–598. https://www.researchgate.net/publication/224773156_FastSLAM_A_Factored_Solution_to_the_Simultaneous_Localization_and_Mapping_Problem
11 A.J. Davison, I.D. Reid, N.D. Molton, and O. Stasse:. IEEE PAMI **29** (2007) 1052. https://ieeexplore.ieee.org/document/4160954
12 I. K. Jung and Lacroix: Proc. 2003 IEEE Int. Conf. Computer Vision (IEEE, 2003) 946–951. https://doi.org/10.1109/ICCV.2003.1238450
13 Y. F. Chen, M. Everett, M. Liu, and J. P. How: Proc. 2017 IEEE/RSJ Int. Conf. Intelligent Robot and Systems (IEEE, 2017) 1343–1350. https://doi.org/10.1109/IROS.2017.8202312
14 M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena: Proc. 2017 IEEE Int. Conf. Robotics and Automation (IEEE, 2017) 1527–1533. https://doi.org/10.1109/ICRA.2017.7989182
15 L. Tai and M. Liu: https://arxiv.org/abs/1610.01733
16 R. Parr and S. Russell: Proc. 1997 Advances in Neural Information Processing Systems (NIPS, 1998) 1043–1049. https://dl.acm.org/doi/10.5555/302528.302894
17 R. S. Sutton, D. Precup, and S. Singh: Artif. Intell. **112** (1999) 181. https://doi.org/10.1016/S0004-3702(99)00052-1
18 T. G. Dietterich: J. Artif. Intell. **13** (2000) 227. https://arxiv.org/abs/cs/9905014
19 K. Rohanimanesh and S. Mahadevan: Adv. Neural Inf. Process. Syst. **15** (2003) 1651. https://www.researchgate.net/publication/2939840_Learning_to_Take_Concurrent_Actions
20 V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller: Proc. 2013 Annu. Conf. Neural Information Processing Systems (NIPS, 2013) 1–9. https://arxiv.org/abs/1312.5602
21 H. Hasselt, A. Guez, and D. Silver: Proc. AAAI Conf. Artificial Intelligence (AAAI, 2015). https://arxiv.org/abs/1509.06461
22 T. Schaul, J. Quan, I. Antonoglou, and D. Silver: Proc. Annu. Conf. Neural Information Processing Systems (NIPS, 2015) 1–21. https://arxiv.org/abs/1511.05952

23 V. Mnih, K. Kavukcuoglu, D. Silver, A. 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: Nature **518** (2015) 529. https://doi.org/10.1038/nature14236
24 P. S. Thomas and E. Brunskill: https://arxiv.org/abs/1706.06643
25 L. Tai, S. Li, and M. Liu: Proc. 2016 IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS, 2016) 2759–2764. https://doi.org/10.1109/IROS.2016.7759428
26 O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J.P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver: Nature **575** (2109) 350. https://doi.org/10.1038/s41586-019-1724-z
27 R. R. O Al-Nima, T. Han, and T. Chen: Proc. 2019 Int. Conf. Computer Recognition Systems (CORES, 2019) 106–116. https://doi.org/10.1007/978-3-030-19738-4_12
28 J. N. Tsitsiklis and B. V. Roy: IEEE Trans. Automat. Control. **42** (1997) 674. https://doi.org/10.1109/9.580874
29 Z. Zang, D. Li, J. Wang, and D. Xia: Knowledge-Based Syst. **40** (2013) 58. https://doi.org/10.1016/j.knosys.2012.11.011
30 C.J. Watkins and P. Dayan: Mach. Learn. **8** (1992) 279. https://doi.org/10.1023/A:1022676722315