Robotica (2019) volume 37, pp. 445–468. © Cambridge University Press 2018
doi:10.1017/S026357471800111X

A Q-learning approach based on human reasoning for navigation in a dynamic environment

Rupeng Yuan, Fuhai Zhang*, Yu Wang, Yili Fu and Shuguo Wang

State Key Laboratory of Robotics and System, Harbin Institute of Technology, Harbin 150001, China. E-mails: yuanrupeng1991@163.com, yuwang_hit@163.com, meylfu_hit@163.com, wangxy_hit@163.com

(Accepted October 2, 2018. First published online: October 30, 2018)

SUMMARY

A Q-learning approach is often used for navigation in static environments where state space is easy to define. In this paper, a new Q-learning approach is proposed for navigation in dynamic environments by imitating human reasoning. As a model-free method, a Q-learning method does not require the environmental model in advance. The state space and the reward function in the proposed approach are defined according to human perception and evaluation, respectively. Specifically, approximate regions instead of accurate measurements are used to define states. Moreover, due to the limitation of robot dynamics, actions for each state are calculated by introducing a dynamic window that takes robot dynamics into account. The conducted tests show that the obstacle avoidance rate of the proposed approach can reach 90.5% after training, and the robot can always operate below the dynamics limitation.

KEYWORDS: Autonomous navigation, Mobile robot, Dynamic environment, Q-learning

1. Introduction

At present, more and more autonomous robots are built to liberate human from doing tedious and dangerous work. Generally speaking, an autonomous robot is designed to complete specific tasks using robot-mounted sensors.[1] There are also some cases where a robot has to cooperate with other intelligent agents to make joint decisions.[2] Two major problems are required to be solved before robots can autonomously navigate in dynamic environments: localization and path planning.

In the process of localization, a robot has to determine its current position and orientation in the environment, and distinguish the places of interest, for example, the target and obstacles. A variety of sensors are capable of locating a robot, such as laser range finders,[3] ultrasonic sensors,[4] Wi-Fi,[5] etc. Many techniques are now available to solve the problem of localization and one of the most widely used is simultaneous localization and mapping (SLAM).[6]

After determining the current location, the robot has to find a collision-free path to the goal.[7] A great number of path planning algorithms have been proposed, attempting to navigate the robot in unknown dynamic environments. For the past decades, the potential field method[8] and its variants[9], [10] are widely used in commercial robots. The principle of the method is to place a potential field in the robot environment and the robot is 'attracted' by the target and 'pushed away' by its surrounding obstacles. At any time, the robot has to calculate the position with the minimal repulsive force and move towards it. Then, the robot repeats the process until reaching the goal. Some improved methods based on the potential field method are introduced. Chen *et al.* [11] provided a method used for obstacle avoidance for a low-speed vehicle using a barrier function. The method creates an 'avoidable set'. When the robot is likely to collide with obstacles, the barrier function controller will intervene, using a mixed integer program to ensure safety with minimal control effort based on the distance to an 'avoidable set'. Rapidly-exploring random tree (RRT)[12] and its variants are considered as Monte-Carlo methods bias

* Corresponding author. E-mail: zfhhit@hit.edu.cn, meylfu_hit@163.com

https://doi.org/10.1017/S026357471800111X Published online by Cambridge University Press

search into the largest Voronoi regions of a graph in a configuration space. RRT is able to determine a collision-free path from the initial position to the goal without the local minimum problem. Mixed-integer linear programming (MILP)[13] is a special case of a linear program where some variables are constrained to take only integer values. The MILP form of the trajectory optimization problems is linear throughout, and so the method is immune to issues of local minima. The method has been successfully used to plan a spacecraft trajectory with avoidance constraints. Dubins curve[14] is also a popular method to navigate a robot safely. The Dubins curve typically refers to the shortest curve that connects two points in the two-dimensional Euclidean plane with a constraint on the curvature of the path and with prescribed initial and terminal tangents to the path. Also, the robot is assumed to only travel forward. A common problem in Dubins curve is the Dubins interval problem where the tangent direction of the path at initial and final points is constrained to lie within the specified intervals.

Recently, there is a shift towards artificial intelligence methods to navigate a robot through learning from past experiences.[15] In general, artificial intelligence methods are easier to implement and converge quickly, which means better computational efficacy.[16] The deep learning method with computer vision has now drawn more and more attention in autonomous navigation. In ref. [17], a deep network method is used for model-less obstacle avoidance. The method shows the effectiveness of a hierarchical structure that fuses a convolutional neural network (CNN) with a decision process. The network structure uses raw depth images as input, and generates control commands output. Findi *et al.* [18] provided a method for solving the problem of determining a smooth and collision-free path with maximum possible speed for a mobile robot. The method contains a genetic network programming with reinforcement learning (GNP-RL) based on predicting collision positions for mobile robot navigation in a dynamic environment. The combination between features of the proposed collision prediction and that of GNP-RL provides safe navigation in a dynamic environment, smooth movement, and reduces the obstacle avoidance latency time.

Q-learning, a reinforcement learning method, seems to be appealing to a great number of researchers since it has high computational efficiency and does not have the need for *a priori* environmental model. The idea of Q-learning can be traced back in 1989.[19] Q-learning algorithms combining with other methods can be applied in different application domains.[20] In the field of mobile robot, Q-learning algorithms have several advantages for autonomous navigation. First of all, the Q-learning algorithm is model free,[21] which means no previous information about the environment is required. The learning agent gets information about the environment by interacting with it. Secondly, the Q-learning agent is trained on-line, in addition, the reward and the punishment for each action it takes will be store in a table. By updating the table using Q-learning algorithms, the 'experiences' of the learning agent begin to accumulate. In ref. [22], Q-learning algorithms are applied for obstacle avoidance. However, the time for navigation to the target is quite long and the hit rate is not high enough using the algorithm. A hybrid method using both Q-learning and neural network is proposed in ref. [23]. Discrete instead of continuous Q-learning is used in the method. Kohonen neural network is adopted to transform the continuous states into discrete ones. The result shows that the method has a better convergence rate than continuous Q-learning. However, this method can only be used in a static environment. In ref. [24], the researchers use hand-crafted features to approximate Q value. A general collision avoidance system is developed through off-line learning and this system can be used to unknown environment. In ref. [25], the researchers give a Q-learning approach to navigate in dynamic environments, in which the states are defined to represent a dynamic environment and the actions are defined as going forward, turning left and right. The result shows that the Q-learning algorithm proposed in ref. [25] is capable of navigating a robot in dynamic environments. However, the trajectory of the robot is not smooth enough since only three actions are allowed during the navigation. Also, since the dynamics of the robot is not taken into the consideration, velocities may not be reachable in certain conditions.

In this paper, a new Q-learning approach gives a solution to navigate a mobile robot in dynamic environments. The state space and actions in the algorithm are defined by imitating human reasoning. The proposed approach specifically takes the dynamics of robot into consideration by introducing the dynamic window; therefore, the proposed method will produce a smooth trajectory for navigation within the robot dynamics. Also, the proposed approach has the ability to solve the local minimal problem. This paper is organized as follow. Section 2 presents the methodology for the Q-learning approach including the definition for states, actions and reward function as well as the training procedure and the policy for navigation. Section 3 presents the simulation in MATLAB. The comparisons between the proposed approach and other approaches are carried out. Section 4 describes

https://doi.org/10.1017/S026357471800111X Published online by Cambridge University Press

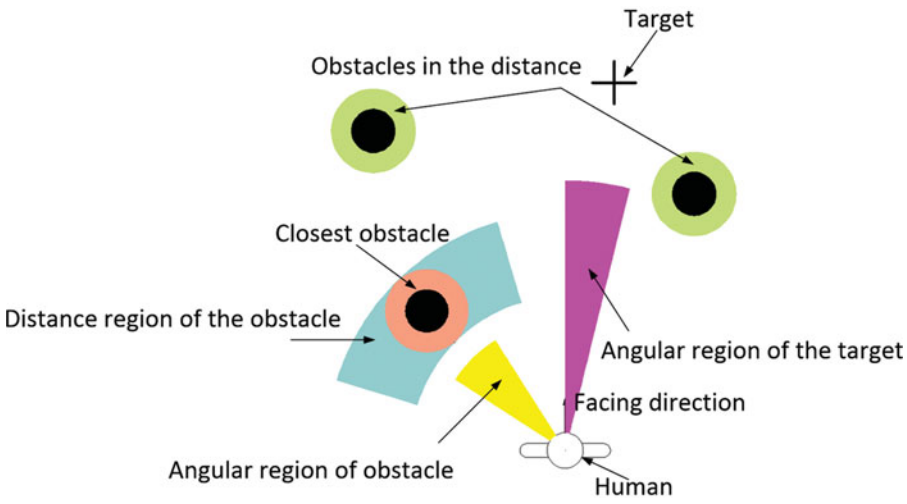


Fig. 1. Human perception of the environment.

the robot system, demonstrates the training in Gazebo and experiments conducted in the real world. Section 5 presents a brief conclusion for the proposed approach.

2. Methodology

2.1. Defining states by imitating human perception

In this paper, the surrounding objects of the robot can be divided into two categories: obstacles and goals. Some of the obstacles are dynamic, while others remain static. The first step to apply Q-learning approach is defining states. Before giving the new definition of states, the way how people avoid obstacles to reach their goals in dynamic environments should be analysed first. The way people perceive the environment is shown in Fig. 1. People who try to walk safely in a dynamic environment tend to deal with obstacles ahead of them one at a time. Therefore, what they care about is the closest obstacle in front of them. Also, people pay no attention to the specific distance or angle between them and the places of interest, for example, obstacles and goals. What they care about is the approximate regions where these places are located. Moreover, it is unnecessary for people to measure the exact speed when they walk. What they care about is the approximate velocity range where the current velocity is in. In the proposed algorithm, the definition of states, which is comprised of the approximate regions where the closest obstacle, goal and velocity are located, is quite similar to human perception.

2.1.1. The description of the robot coordinate system. The robot is considered to be the centre of the environment. In order to simply describe the position and orientation of obstacles and goals relative to the robot, the robot coordinate system is placed in a fixed coordinate system. The robot coordinate system that contains the goal and obstacles is shown in Fig. 2. θ_R is the angle between the robot heading direction and X-axis, θ_T is the angle between the robot to target line and X-axis, θ_o is the angle between the robot to obstacle line and X-axis and D is the distance between the robot and the obstacle.

The forward area (FA) is the region that the robot heading direction points to. The origin of the FA is the centre of the robot. The angle of FA ranges from $\theta_R - \pi/2$ to $\theta_R + \pi/2$. By this definition, obstacles in the environment are either located in FA or not, as shown in Fig. 2. The proposed Q-learning algorithm only takes the closest obstacle located in FA into account.

2.1.2. The definition of obstacle states. In the proposed algorithm, obstacles are regarded as 'spots' without size or shape. For example, an obstacle can be regarded as a single 'spot' in one scan of a laser range finder. At any time, the origin of FA is $P = [\,P_x \quad P_y\,]^{\mathrm{T}}$, and the closest obstacle located in FA is $P_{\mathrm{obs}} = [\,P_{\mathrm{obs}-x} \quad P_{\mathrm{obs}-y}\,]^{\mathrm{T}}$. The position and direction of the obstacle located in FA can be completely defined by distance D and angle θ, respectively, as shown in Fig. 3(a). The distance D between the

https://doi.org/10.1017/S026357471800111X Published online by Cambridge University Press

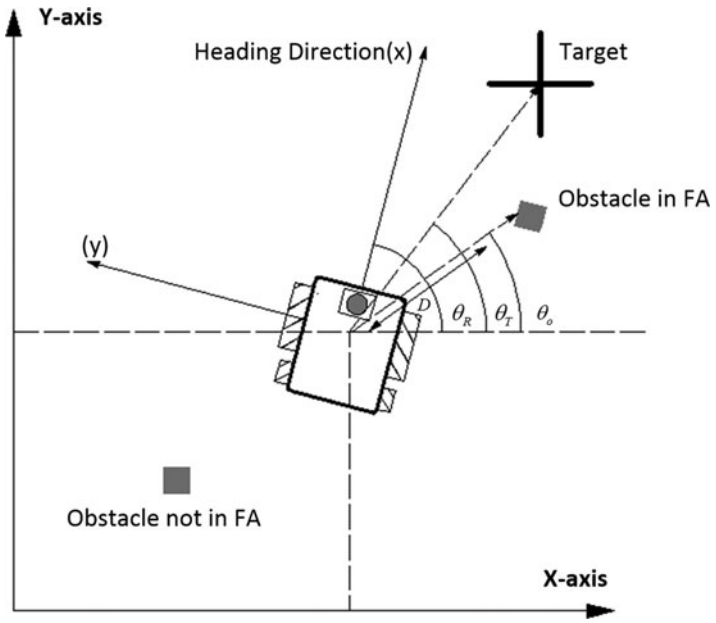


Fig. 2. The description of the robot coordinate system.

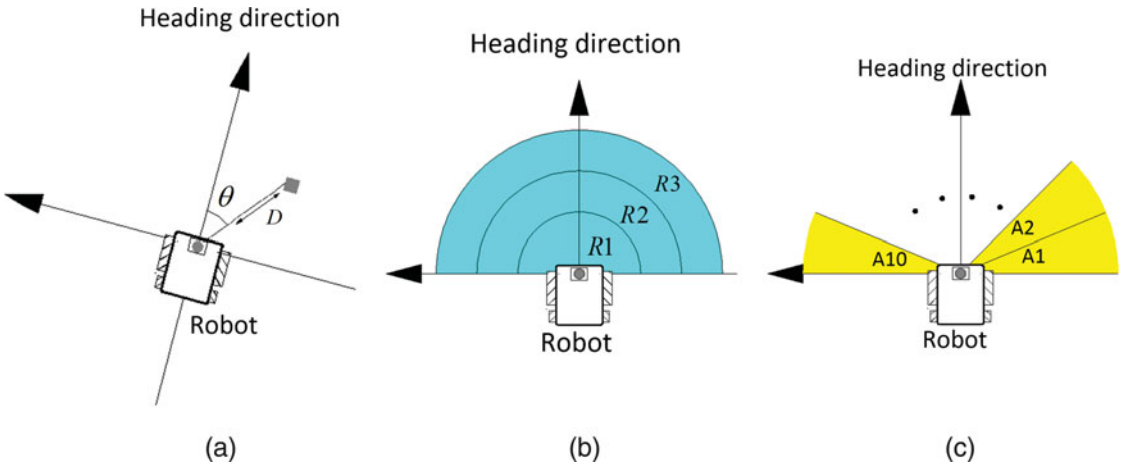


Fig. 3. Distance D and angle θ and their corresponding approximate regions.

origin of FA and the obstacle in FA is given by the following equation:

$$D = \|P_{\text{obs}} - P\| \tag{1}$$

The angle θ between the robot to obstacle line and heading direction is given by following equation:

$$\begin{cases} \theta_o = \arccos\left(\left(P_{\text{obs}-x} - P_x\right)/D\right) \\ \theta = \theta_o - \theta_R, \theta \in [-\pi/2, \pi/2] \end{cases} \tag{2}$$

In order to divide FA into safe region (SR) and non-safe region (NSR), $D_{\max}$ is introduced as a constant. An obstacle with $D > D_{\max}$ is considered to be located in an SR. If the closest obstacle in FA is located in SR, the robot can simply move closer to the goal according to a strategy which will be discussed later. An obstacle with $D < D_{\max}$ is considered to be located in an NSR. The NSR can be further divided into three sub regions:

$$R = \begin{cases} R1, & D \in [0, D_{\max}/3) \\ R2, & D \in [D_{\max}/3, 2D_{\max}/3) \\ R3, & D \in [2D_{\max}/3, D_{\max}) \end{cases} \tag{3}$$

https://doi.org/10.1017/S026357471800111X Published online by Cambridge University Press

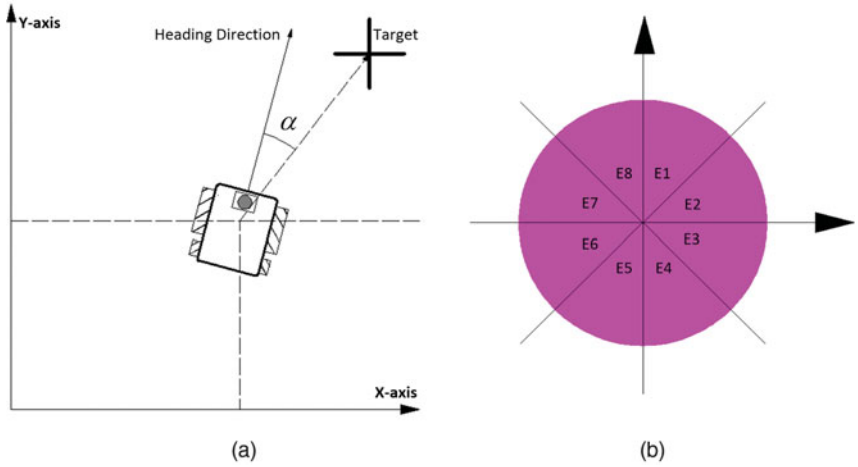


Fig. 4. Angle α and its corresponding approximate regions.

In Eq. (3) R is the approximate distance region where the closest obstacle is located. $R1$ to $R3$ are the possible values for the region. The three regions R_1, R_2, R_3 are shown in Fig. 3(b).

In order to complete the description for the location of obstacle, it is necessary to add the approximate angle region where the closet obstacle is located. FA can then be divided into 10 sub regions according to angleθ:

$$A = \begin{cases} A1 & \theta \in [-\pi/2, -\pi/2 + res) \\ A2 & \theta \in [-\pi/2 + \text{res}, -\pi/2 + 2\text{res}) \\ \vdots & \vdots \\ A10 & \theta \in [-\pi/2 + 9\text{res}, \pi/2] \end{cases} \tag{4}$$

The angular resolution for each region is $\text{res} = \pi/10$.

In Eq. (4) A is the approximate angle region where the obstacle is located, and $A1$ to $A10$ are the possible values for the region. These 10 regions are shown in Fig. 3(c).

2.1.3. The definition of target states. Now that the approximate region where the closest obstacle is located has been described thoroughly, the description for the target region should now be given to define states. The location of the target is $P_{\text{tar}} = [\, P_{\text{tar}-x} \quad P_{\text{tar}-y} \,]^{\text{T}}$. D_t is the distance between the robot and the target. As Fig. 4(a) shows, angle α that measures deviation of robot heading direction from the target is given by following equation:

$$\begin{cases} D_t = \|P_{\text{tar}} - P\| \\ \theta_T = \arcsin\left(\left(P_{\text{tar}-y} - P_y\right)/D_t\right) \\ \alpha = \theta_R - \theta_T, \alpha \in [0, \pi] \end{cases} \tag{5}$$

The range of angle α is the interval $[0, 2\pi)$. To give the approximate region where the target is located, the interval is divided into eight angular regions:

$$E = \begin{cases} E1, & \alpha \in [0, \pi/4) \\ E2, & \alpha \in [\pi/4, \pi/2) \\ E3, & \alpha \in [\pi/2, 3\pi/4) \\ E4, & \alpha \in [3\pi/4, \pi) \\ E5, & \alpha \in [\pi, 5\pi/4) \\ E6, & \alpha \in [5\pi/4, 3\pi/2) \\ E7, & \alpha \in [3\pi/2, 7\pi/4) \\ E8, & \alpha \in [7\pi/4, 2\pi) \end{cases} \tag{6}$$

https://doi.org/10.1017/S026357471800111X Published online by Cambridge University Press

In Eq. (6) E is the approximate angle region where the target is located, and $E1$ to $E8$ are the possible values for the region. The eight regions are shown in Fig. 4(b).

2.1.4. The definition of velocity states. So far, approximate regions where the closest obstacle and goal are located have completely been described. The description for the approximate velocity region should be given to complete the definition of state space. V_l and V_a are the linear velocity and angular velocity of the mobile robot, respectively. $V_{l\max}$ and $V_{a\max}$ are the maximum linear and angular velocity allowed by the mobile robot, respectively. In order to reduce the state space, resolution for V_l and V_a is $V_{l\max}/4$ and $V_{a\max}/5$, respectively. Therefore, approximate linear velocity region and angular velocity region can be represented the following equations:

$$VL = \begin{cases} VL1 & V_l \in [0, V_{l\max}/4) \\ VL2 & V_l \in [V_{l\max}/4, 2V_{l\max}/4) \\ VL3 & V_l \in [2V_{l\max}/4, 3V_{l\max}/4) \\ VL4 & V_l \in [3V_{l\max}/4, V_{l\max}) \end{cases} \tag{7}$$

$$VA = \begin{cases} VA1 & V_a \in [-V_{a\max}, -V_{a\max} + V_{a\max}/5) \\ VA2 & V_a \in [-V_{a\max} + V_{a\max}/5, -V_{a\max} + 2V_{a\max}/5) \\ \vdots & \vdots \\ VA10 & V_a \in [-V_{a\max} + 9V_{a\max}/5, V_{a\max}) \end{cases} \tag{8}$$

In Eq. 7 and Eq. 8, VL and VA are the approximate linear velocity region and approximate angular velocity where V_l and V_a are in, respectively. $VL1$ to $VL4$ are the possible values for the linear velocity region. $VA1$ to $VA10$ are the possible values of the angular velocity region.

2.1.5. The complete definition of states. Using all the definitions above, the state at any time instant can be completely defined as a combination of the approximate regions of the closest obstacle ahead, the target and the linear and angular velocity:

$$s_t = (R, A, E, VL, VA) \tag{9}$$

where s_t is the state of robot at time instant t, R is the distance region of obstacle, A is the angular region of obstacle, E is the angular region of the goal and VL and VA are the velocity region of linear velocity and angular velocity, accordingly.

In fact, in the definition of obstacle states, target states and velocity states, the number of sub-regions that divides a continuous value is chosen in a proper way. The larger the number is, the finer the value can be divided. The robot may perform better if the number of division is increased. At the same time, however, the number of states will also become larger which may cause insufficient training in every one of the states. Therefore, the number of states is the compromise of the fineness of state division and the number of the training scenarios that can sufficiently train the robot.

2.2. Defining actions by introducing a dynamic window

Since all states space have been defined completely, allowed actions for each state can be specified. Action is defined as the *index* from which the pair of linear and angular velocity can be calculated.

At any time, velocities that a robot can reach are in a dynamic window,[26] which restricts the admissible velocities to those that can be reached within a short time interval when the limited accelerations of the robot are given.

As shown in Fig. 5, the current velocity of the robot is $V_r = (V_l, V_a)$.V_l and V_a are the linear and angular velocity, respectively. The dynamic window V_d for the current velocity can be described as follow:

$$V_d = \{(v, \omega) \,|\, [V_l - \dot{v} \cdot \Delta t, V_l + \dot{v} \cdot \Delta t] \wedge [V_a - \dot{\omega} \cdot \Delta t, V_a + \dot{\omega} \cdot \Delta t], 0 < v < v_{\max}, \omega < \omega_{\max}\} \tag{10}$$

where (v, ω) is the set of admissible linear and angular velocity, $\dot{v}$ and $\dot{\omega}$ are the maximum linear and angular acceleration allowed by the robot, Δt is the time interval and $v_{\max}$ and $\omega_{\max}$ are the maximum linear and angular velocity allowed by the robot, respectively.

https://doi.org/10.1017/S026357471800111X Published online by Cambridge University Press

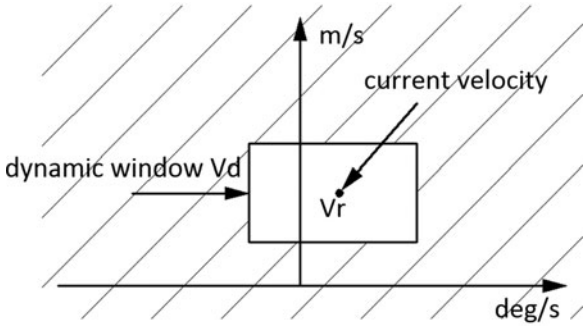


Fig. 5. Dynamic window.

The minimum linear velocity $V_{l\min}$ in V_d is calculated as $V_{l\min} = \max((V_l - \dot{v} \cdot \Delta t), 0)$.
The minimum angular velocity $V_{a\min}$ in V_d is calculated as $V_{a\min} = \max((V_a - \dot{\omega} \cdot \Delta t), -\omega_{\max})$
The maximum linear velocity $V_{l\max}$ in V_d is calculated as $V_{l\max} = \min((V_l + \dot{v} \cdot \Delta t), v_{\max})$.
The maximum angular velocity $V_{a\max}$ in V_d is calculated as $V_{l\max} = \min((V_a + \dot{\omega} \cdot \Delta t), \omega_{\max})$.

In this paper, each state has 40 actions to choose. In other words, the index has the range from 1 to 40. As angular and linear velocities in V_d are continuous, the resolution of velocity should be introduced. Resolutions for linear and angular velocity are $\text{res}_l = (V_{l\max} - V_{l\min})/4$ and $\text{res}_a = (V_{a\max} - V_{a\min})/10$, respectively.

The linear velocity V_{nl} and angular velocity V_{na} calculated from the action taken, i.e. index n, $n \in [1, 40]$ are defined as follow:

$$\begin{cases} V_{nl} = V_{l\min} + \text{floor}\left((n-1)/4\right) \cdot \text{res}_l \\ V_{na} = V_{a\min} + ((n-1) \bmod 10) \cdot \text{res}_a \end{cases} \tag{11}$$

2.3. Defining reward function using human evaluation

In Q-learning algorithms, reward functions are used to give evaluation for the taken action at a given state. For people walking in a dynamic environment, they tend to keep away from the obstacle, head towards the target and try to reach their goals quickly. The reward function proposed by this paper simulates human evaluation by taking into account the distance from the closest obstacle, the angle between the heading direction and robot to target line and the linear velocity calculated from action.

2.3.1. The calculation of robot movement and the introduction of virtual trajectory. Let $[\, P_{xt} \quad P_{yt} \quad \theta_{Rt} \,]^{\mathrm{T}}$ denotes the position and heading direction of robot at time t in a global coordinate system. Let $[\, v_t \quad \omega_t \,]^{\mathrm{T}}$ denote the pair of linear and angular velocity calculated from the action taken at time t. The position and heading direction of robot $[\, P_{x(t+1)} \quad P_{y(t+1)} \quad \theta_{R(t+1)} \,]^{\mathrm{T}}$ at time $t+1$ can be calculated as follow:

$$\begin{bmatrix} P_{x(t+1)} \\ P_{y(t+1)} \\ \theta_{R(t+1)} \end{bmatrix} = \begin{bmatrix} P_{xt} \\ P_{yt} \\ \theta_{Rt} \end{bmatrix} + \begin{bmatrix} \cos(\theta_{Rt}) & 0 \\ \sin(\theta_{Rt}) & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v_t \\ \omega_t \end{bmatrix} \cdot \Delta t \tag{12}$$

In Eq. (12), Δt is the time interval between $t+1$ and t. As the time interval is quite short, the movement of robot is not quite noticeable for evaluation. In order to properly define a reward function, the result of a taken action should be 'magnified'. Therefore, a virtual trajectory should be introduced using simulation time t_s. The virtual trajectory is defined as a hypothetical route which the robot generates in simulation time t_s by the pair of linear and angular velocity $[\, v_t \quad \omega_t \,]^{\mathrm{T}}$. The position and heading direction of robot$[\, P_{xv} \quad P_{yv} \quad \theta_{Rv} \,]^{\mathrm{T}}$ at the end of virtual trajectory can be achieved by iterating Eq. (19). The pseudo code that illustrates the iterative process is given by Fig. 6.

2.3.2. The evaluation for the taken action. The robot location at the end of the virtual trajectory is $P_v = [\, P_{xv} \quad P_{yv} \,]^{\mathrm{T}}$, and the closest obstacle located in FA is $P_{\text{obs}} = [\, P_{\text{obs}-x} \quad P_{\text{obs}-y} \,]^{\mathrm{T}}$. The distance D_v between the virtual robot location and the closest obstacle in FA is given by the following equation:

$$D_v = \|P_v - P_{\text{obs}}\| \tag{13}$$

https://doi.org/10.1017/S026357471800111X Published online by Cambridge University Press

```
function ITERATIVE-ALGORITHM(P_xt, P_yt, θ_Rt, v_t, ω_t)
    inputs: P_xt, P_yt, θ_Rt, the robot's position and orientation at time t
            v_t, ω_t, the robot's velocity at time t
    outputs: P_xv, P_yv, θ_Rv, the robot's hypothetical position and orientation
    persistent: Δt, the time interval between time t and time t + 1
                t_s, the simulation time to generate a virtual trajectory
    for counter = 1 to t_s/Δt do
        P_xt ← P_xt + v_t × cos(θ_Rt) × Δt
        P_yt ← P_yt + v_t × sin(θ_Rt) × Δt
        θ_Rt ← θ_Rt + ω_t × Δt
    end for
    P_xv ← P_xt
    P_yv ← P_yt
    θ_Rv ← θ_Rt
    return P_xv, P_yv, θ_Rv
end function
```

Fig. 6. Pseudo code for the iterative algorithm.

Angle α_v that measures the deviation of robot heading direction from the target is given by the following equation:

$$\alpha_v = \theta_{Rv} - \theta_T \tag{14}$$

where θ_{Rv} is the robot heading direction at the end of the virtual trajectory. θ_T is the angle between the robot to target line and X-axis.

2.3.3. The reward function. To define the reward function, two transient states and two terminal states should be introduced. All these states are defined as the transition state S.

Two transient states are defined as

- The safe states (SS) if the robot is located in SR.
- The non-safe states (NS) if the robot is located in NSR.
- Two terminal states are defined as
- The winning states (WS) if the robot reaches the goal.
- The failure states (FS) if the robot has collided with the obstacle.

In order to train the robot navigating safely in the environment, the reward function $R(s_t, a_t)$ can be defined as follow:

- The reward for moving from an NS to SS is $R(s_t, a_t) = 1$.
- The reward for moving from an NS to FS is $R(s_t, a_t) = -1$.
- The reward for moving from an NS to WS is $R(s_t, a_t) = 2$.

The reward for moving from one NS to another NS should be calculated as follow:

$$R(s_t, a_t) = a \cdot D_v + b \cdot \alpha_v + c \cdot v_t \tag{15}$$

where a, b and c are the weight coefficients that determine the importance of D_v, α_v and v_t in evaluation. s_t and a_t are the state and the action that the robot takes at time t. To summarize, the reward function can be rewritten as

$$R(s_t, a_t) = \begin{cases} 1, & S \subset NS \rightarrow SS \\ -1, & S \subset NS \rightarrow FS \\ 2 & S \subset NS \rightarrow WS \\ a \cdot D_v + b \cdot \alpha_v + c \cdot v_t & S \subset NS \rightarrow NS \end{cases} \tag{16}$$

https://doi.org/10.1017/S026357471800111X Published online by Cambridge University Press

2.4. The value function

The value function of Q-learning algorithms is stored in a table. The rows of the table represent different states the robot passes through, and the columns are the actions performed by the robot. All the Q-values stored in the table are initially set to zero. During the training process, each entry in the table is filled by updating Q-values of state and action pairs. The update equation for Q-learning algorithm is as follows:

$$Q(s_t, a_t) = Q(s_t, a_t) + \beta \left(R(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right) \tag{17}$$

where s_t and a_t are the state and action that robot takes at time t. β is the learning factor. γ is the discount factor. $\max_a Q(s_{t+1}, a)$ is the maximum Q-value calculated for taking all possible actions in the new state at time $t + 1$. By exposing the robot in different training scenarios, the Q-table of the robot is updated. After training the robot, it is able to navigate in a real environment using information stored in the table.

2.5. The policy

The policy for navigation in real world can be achieved by training the robot in different testing environments. In SR, the robot simply turns to the target and moves closer. In NSR, however, the robot has to follow the policy when navigating. In this case, the robot takes the action with the highest Q-value from the Q-table for the current state, and moves forward with the pair of linear and angular velocity calculated from the action taken.

2.5.1. Training procedure for the Q-learning algorithm. Training the robot in different environments can update the Q-value stored in the Q-table. Every one of the training environments is called a scenario. Since the location of the closest obstacle is not given in advance, the robot is obliged to perceive and choose the closest obstacle ahead of it through sensors (e.g. laser range finders).

The flow chart of training procedure for the Q-learning algorithm is shown in Fig. 7(a). The training procedure contains different scenarios. Each scenario begins with calculating the robot current state. The location of the goal is defined in the global coordinate system and set to the robot. The location of obstacles and velocity of robot are achieved through the sensors equipped on the robot. Therefore, the state at time t, $s_t = (R, A, E, VL, VA)$ can be calculated at this stage.

Now that the state has been determined, it is easy to find out whether the robot is located in SR by determining the approximate region where the closest obstacle in FA is located. If the robot is located in SR and if the angle α defined in Eq. (5) is relatively large, then it will move closer to the target by a pair of linear and angular velocity $[\, v_f \quad \omega_f \,]^{\mathrm{T}}$ or $[\, v_f \quad -\omega_f \,]^{\mathrm{T}}$ depending on which pair of velocity can turn the robot to the target faster, and if the angle α is relatively small, then the robot will move closer to the target by a pair of linear and angular velocity $[\, v_f \quad 0 \,]^{\mathrm{T}}$. v_f and w_f are the fixed linear and angular velocity. If the robot reaches the goal, then current training scenario will finish and a new scenario will initiate. Otherwise, the robot calculates the state at time $t + 1$ and repeats the process.

If the robot is located in NSR, it has to retrieve the specific row in the Q-table according to the state calculated, and choose the action that will make it move closer to the goal while avoiding the closest obstacle. As mentioned before, the action chosen is actually an index. The robot can calculate the pair of linear and angular velocity according to the index using Eq. (11). After moving towards the goal with the calculated velocity pair, the robot calculates the reward for the current transition state and updates the Q-table using Eqs. (16) and (17) accordingly. Then the robot will check if it has collided with the obstacle. If the collision happens, then robot will initiate a new scenario. Otherwise, the robot will check if it has reached the goal. If it has reached its goal, a new scenario will initiate, or the robot will repeat the current training process by calculating the state in a new time instant $t + 1$.

2.5.2. Navigation with the learned policy. After training the robot in different scenarios, the resulting policy is able to navigate the robot in dynamic environments. The flow chart in Fig. 7(b) shows the behaviour of robot when following the policy. Once the robot has determined its current state, it is able to find out whether it is located in SR or not. If the robot is located in SR, it will move closer to the goal through a strategy described above. If the robot is located in NSR, it will retrieve the row in the Q-table according to its state and choose the action with the highest Q-value. After the pair of linear and angular velocity is calculated according to the chosen action, the robot moves forward with

https://doi.org/10.1017/S026357471800111X Published online by Cambridge University Press

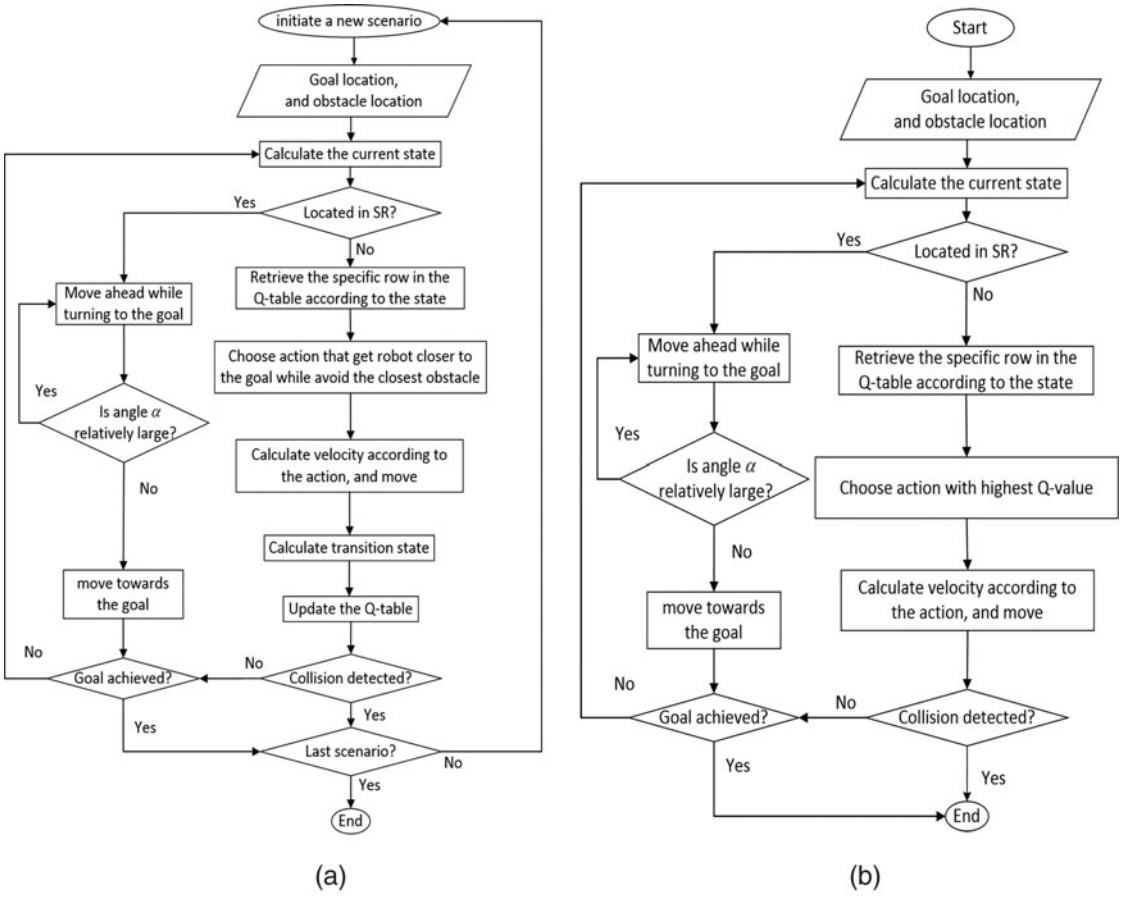


Fig. 7. (a) Training procedure for Q-learning algorithm. (b) Navigation with learned policy.

that velocity pair. The robot will recalculate its current state and repeat the process until it reaches its goal or collides with an obstacle.

3. Simulation

The simulation of the proposed Q-learning algorithm is carried out in MATLAB. In the training procedure, different scenarios are designed to train the robot. After the training procedure, the robot is placed in several dynamic environments to evaluate the effectiveness of the proposed Q-learning algorithm. In this section, the proposed algorithm is compared with the algorithm proposed in ref. [22] and the potential field method.

The simulation conducted in MATLAB contains two aspects: the training procedure and the navigation tests using the learned policy. In the training procedure, the robot is exposed to different scenarios. At the beginning, in order to conduct more targeted trainings, the robot is trained in specific conditions where each obstacle is remained static at a required location. Dynamic obstacles are gradually added into the scenarios in order to simulate the real environment. The training results are stored in a Q-table which is updated while training. In navigation using the learned policy, the robot is exposed to testing scenarios where both dynamic obstacles and static obstacles exist. These testing scenarios are used to assess the resulting Q-table trained in the training procedure.

For all the simulations conducted in MATLAB, the scale for one meter is one; the robot is initially located at point $[0 \quad 0]^{\mathrm{T}}$; the constant $D_{\max}$ is 2.1; and the initial linear and angular velocity of the robot is zero. The fixed linear and angular velocities, v_f and w_f, for navigation in SR are 0.3 m/s and 0.26 rad/s, respectively.

https://doi.org/10.1017/S026357471800111X Published online by Cambridge University Press

Table I. Parameters for the training procedure.

Parameter name	Parameter symbol	Parameter value
Initial Heading direction	θ_R	$\pi/2$
Maximum linear acceleration	$\dot{v}$	0.2 m/s
Maximum angular acceleration	$\dot{\omega}$	0.87 rad/s
Maximum linear velocity	$v_{\max}$	0.8 m/s
Maximum angular velocity	$\omega_{\max}$	0.35 rad/s
Minimum linear velocity	$v_{\min}$	0 m/s
Minimum angular velocity	$\omega_{\min}$	−0.35 rad/s
Weight coefficient	a	0.05
Weight coefficient	b	0.2
Weight coefficient	c	0.1
Simulation time	t_s	3 s
Time interval	Δt	0.1 s
Learning factor	β	1
Discount factor	γ	1

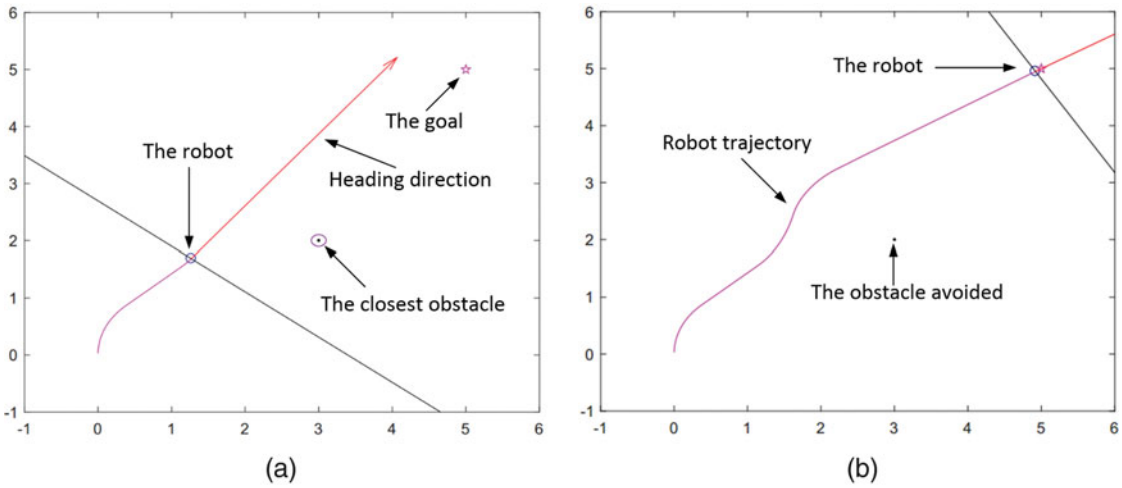


Fig. 8. (a) The robot located in an NSR. (b) The robot had reached the target.

3.1. The training procedure

The training procedure in MATLAB contains a great number of scenarios. The procedure trains the robot gradually. At first, the robot is trained in several scenarios with only one static or dynamic obstacle. Then the robot is exposed in scenarios with a few static obstacles. Finally, the robot is exposed in scenarios with both static obstacles and dynamic ones. At this stage, training scenarios are very similar to the real environments. In order to demonstrate the training procedure in detail, four typical kinds of training scenario are analysed. Parameters used for the training procedure can be seen in Table I.

3.1.1. Training scenarios that contain one static obstacle. The environments in these scenarios are simple. The objective of these scenarios is to train the robot in specific condition, and to be prepared for the more complex scenarios in the future. The target is located at point $[5 \quad 5]^{\mathrm{T}}$. One of these scenarios can be seen in Fig. 8. At first, the robot was located in an SR. It turned around and moved towards the target. Then it reached an NSR with the obstacle located at $[3 \quad 2]^{\mathrm{T}}$, as shown in Fig. 8(a). It calculated its current state and started to learn using the Q-learning Algorithm. Figure 8(b) shows the entire trajectory of the robot reaching its target after learning.

3.1.2. Training scenarios that contain one dynamic obstacle. The environments in these scenarios are a little more complicated than the previous ones. The objective of these scenarios is to train the robot in a simple dynamic environment. The target is located at point $[5 \quad 5]^{\mathrm{T}}$. One of these scenarios can be seen in Fig. 9. The obstacle is located at point $[2 \quad 3]^{\mathrm{T}}$. The velocity of the obstacle is $[0 \quad -0.2]^{\mathrm{T}}$.

https://doi.org/10.1017/S026357471800111X Published online by Cambridge University Press

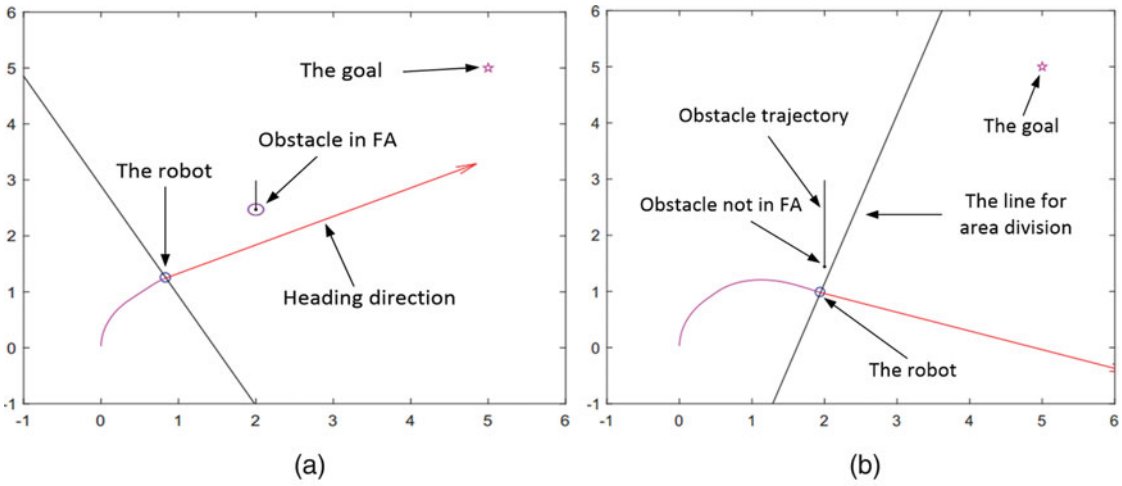


Fig. 9. (a) The robot moved into an NSR. (b) The robot reached an SR.

At first, the robot turned around and moved towards the target, until it moved into an NSR, as shown in Fig. 9(a). The robot calculated the state and chose the action using the Q-table trained in the previous scenarios. The robot finally avoided the obstacle and reached an SR, as shown in Fig. 9(b).

3.1.3. Training scenarios that contain multiple static obstacles. In these scenarios, the robot is exposed to more complicated environments. The target is located at point $[20 \quad 20]^{\mathrm{T}}$. One of these scenarios can be seen in Fig. 10. The positions where static obstacles were located can be seen from the picture. At first, the robot was located in an SR; it turned around and moved towards to the target. Then it moved into an NSR. Since the Q-table of the robot had been updated in the above two kinds of scenarios, the robot was able to navigate safely by choosing the action that would make it move closer to the goal while avoiding the closest obstacle. The robot performed quite well at first until it encountered the obstacle located at $[7 \quad 11]^{\mathrm{T}}$. The robot thought that heading towards the target seemed to be a good option, but it actually collided with the obstacle (Fig. 10(a)). The reward function for this failure is -2. Then the robot learns which action to choose through updating Q-table. As shown in Fig. 10(b), after suffering a couple of failures, the robot had finally learned how to avoid that obstacle and navigate safely to the target.

3.1.4. Training scenarios that contain multiple static and dynamic obstacles. In these scenarios, the robot is exposed to more complicated environments that contain both static and dynamic obstacles. The objective of these scenarios is to train the robot in environments similar to the real world. The target is located at point $[20 \quad 20]^{\mathrm{T}}$. The velocity of dynamic obstacles ranges from 0.2 m/s to 0.4 m/s. In order to train the robot more effectively, an obstacle does not move unless the distance between the robot and the obstacle is less than 5 meters. One of the training scenarios can be seen in Fig. 11. At first, the robot located in an SR can move towards the target in a smooth trajectory. Since the Q-table had been updated quite many times in the previous training scenarios, the robot is quite able to perform obstacle avoidance in NSR. After avoiding both static and dynamic obstacles, the robot safely reached its target. At this stage, the robot has learned quite enough to navigate safely in a dynamic environment.

3.2. Tests using trained policy

In this subsection, different testing scenarios are designed to assess the capability of navigation, using the resulting Q-table after the training procedure. In some testing scenarios, obstacles are randomly located, while others are located at required positions to make comparison with other navigation algorithms. In all testing scenarios, the robot is initially located at $[0 \quad 0]^{\mathrm{T}}$.

3.2.1. Testing in a dynamic environment. One typical testing scenario where obstacles are randomly located can be seen in Fig. 12. The target is located at point $[20 \quad 20]^{\mathrm{T}}$. The velocity of obstacle ranges from 0.2 m/s to 0.4 m/s. When the robot reached the NSR, it calculated its current state and

https://doi.org/10.1017/S026357471800111X Published online by Cambridge University Press

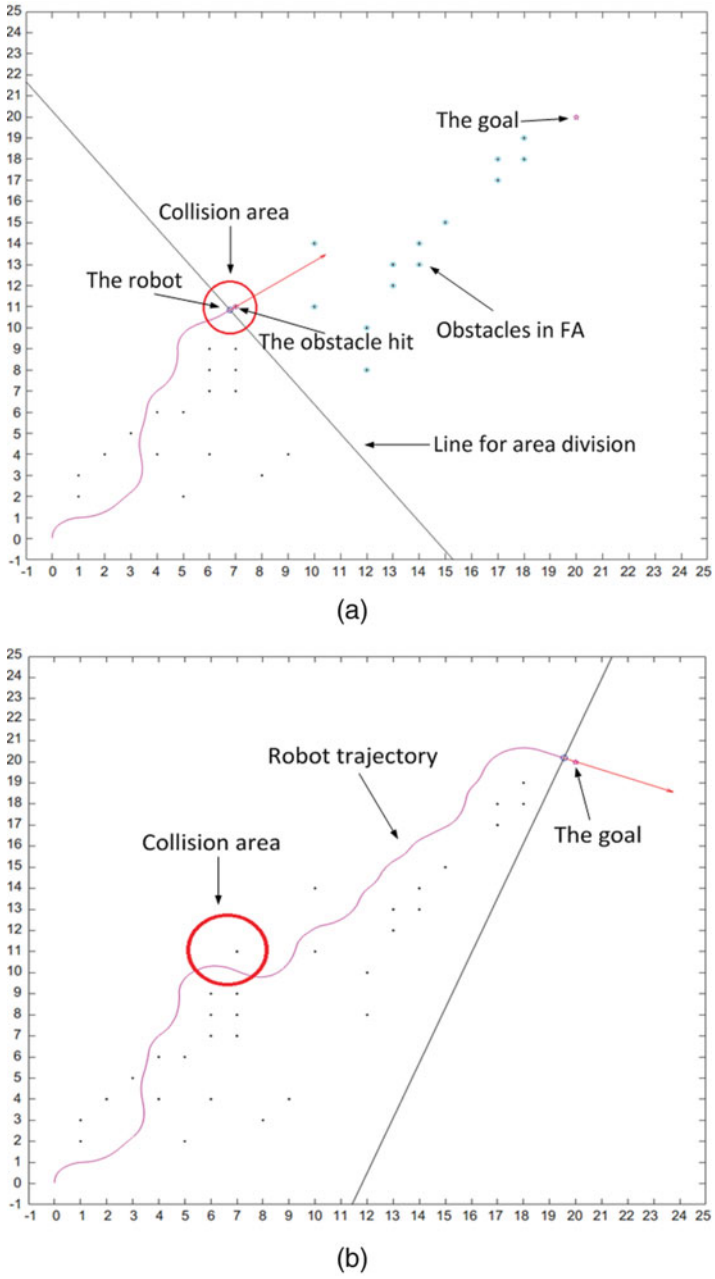


Fig. 10. (a) Robot collided with the obstacle. (b) Robot avoided the obstacle.

retrieved the specific line in the Q-table according to the state. By taking the action with the highest Q-value, the robot was able to navigate safely in the environment.

3.2.2. Comparison with the potential field method. In this paper, the proposed algorithm is compared with the potential field method, one of the most widely used path planning algorithms.[8] The potential field method assumes that the robot is driven by effect of the forces of two potential fields. The goal generates the attractive potential field which adds attractive force F_{attr} to 'pull' the robot moving towards it, and obstacles generate the repulsive potential field which adds repulsive force F_{rep} to push the robot away from them. And the robot is driven by the total virtual force $F_{\text{total}} = F_{\text{attr}} + F_{\text{rep}}$.

The potential field method suffers from the local minimum problem when the robot, the target and the obstacle are on the same line, as shown in Fig. 13(a). As the obstacle gets closer to the robot, the repulsive force gradually becomes equals to or greater than the attractive force. Therefore, the robot

https://doi.org/10.1017/S026357471800111X Published online by Cambridge University Press

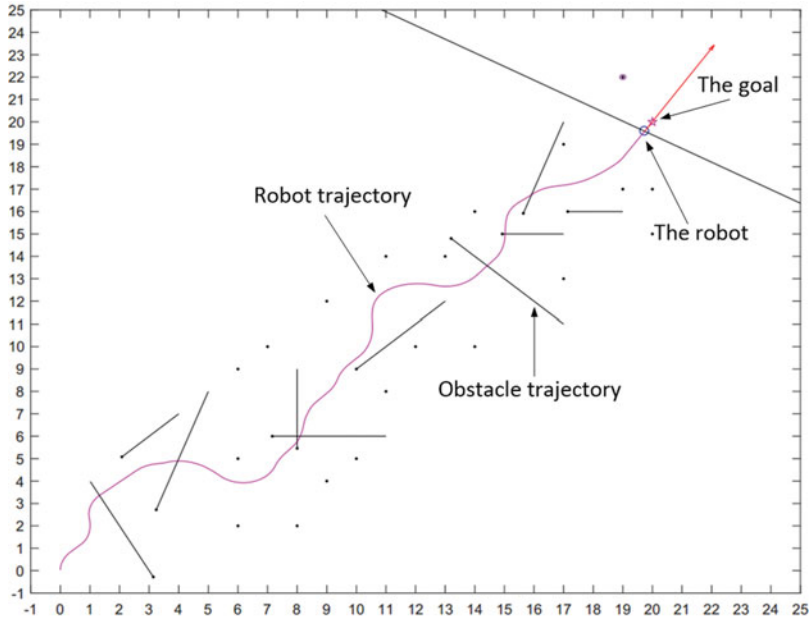


Fig. 11. The robot navigating safely in a dynamic environment.

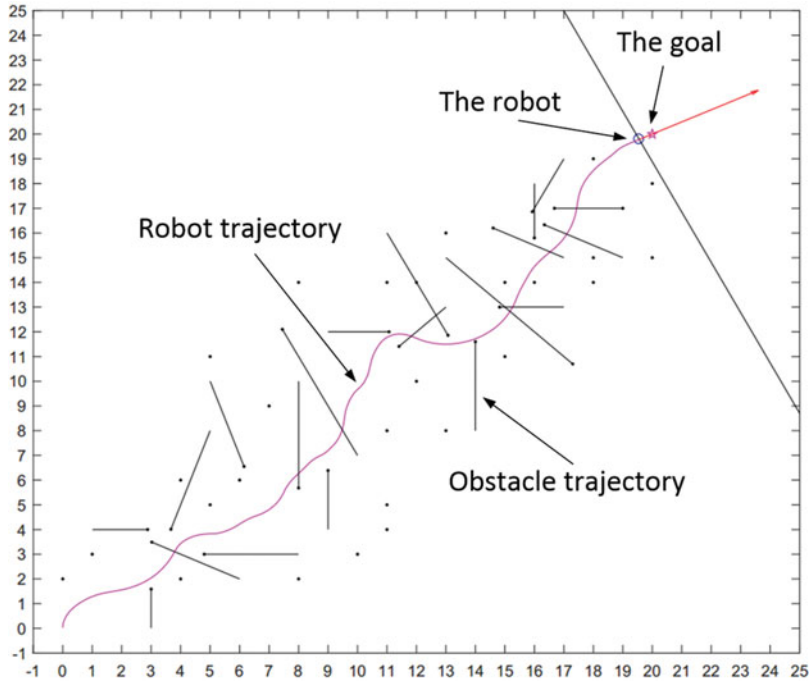


Fig. 12. The robot navigating using learned policy in a complicated environment.

will inevitably stop or move backward instead of avoiding the obstacle. As a result, the robot will never reach the goal unless the obstacle moves away.

The objective of testing scenarios in Fig. 13 is to verify the capacity of the proposed algorithm to solve the local minimum problem. The heading direction of the robot is $\pi/4$. The target is located at $[5 \quad 5]^{\mathrm{T}}$. In Fig. 13(a), the obstacle is initially located at $[4 \quad 4]^{\mathrm{T}}$ and the velocity of the obstacle is $[-0.2 \quad -0.2]^{\mathrm{T}}$. In Fig. 13(b), the obstacle is initially located at $[1 \quad 1]^{\mathrm{T}}$ and the velocity of the obstacle is $[0.2 \quad 0.2]^{\mathrm{T}}$. In the training process, the robot updated its Q-table through interacting with the environment. Through the experience from the past, the robot avoided the obstacle by increasing

https://doi.org/10.1017/S026357471800111X Published online by Cambridge University Press

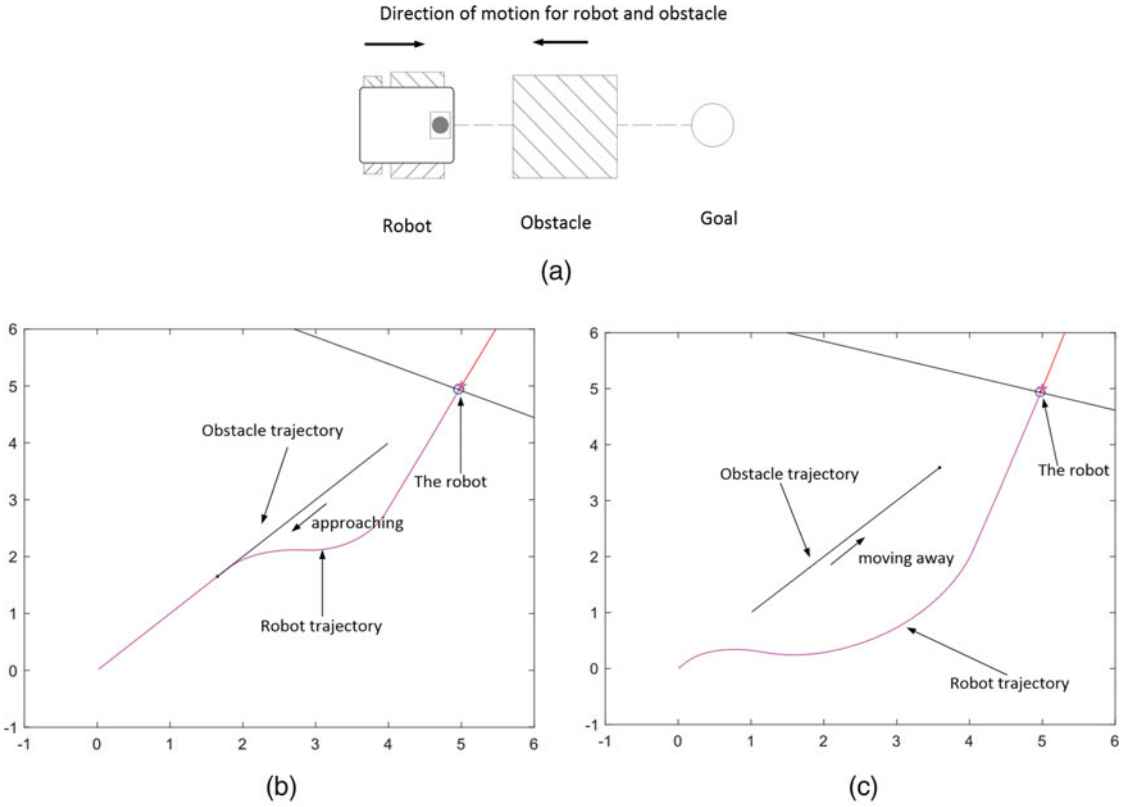


Fig. 13. (a) The local minimum problem. (b) Comparison with the potential field method.

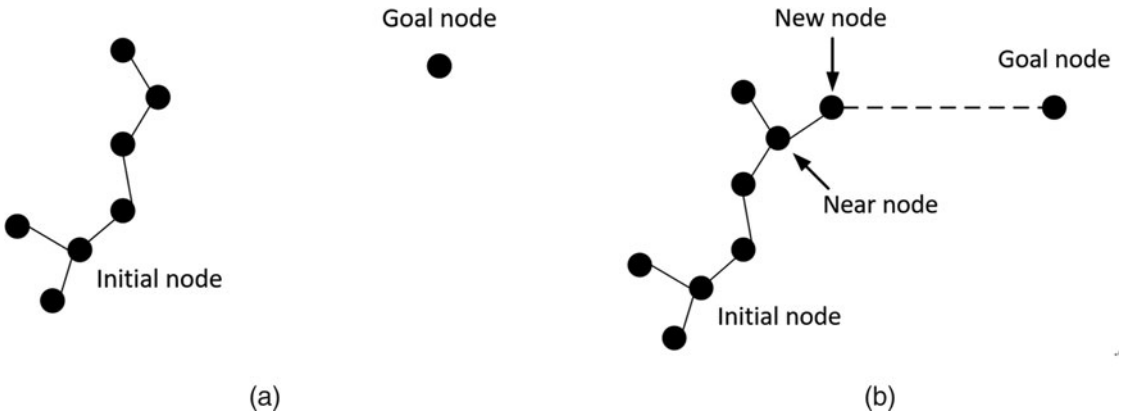


Fig. 14. A brief process of a rapidly-exploring random tree.

its angular velocity. The test shows that the algorithm proposed by this paper has an advantage in solving the local minimum problem.

3.2.3. Comparison with rapidly-exploring random tree. In this paper, the proposed algorithm is compared with the RRT, which also has the ability to solve the local minimum problem. RRT is a sampling-based algorithm for solving path planning problem, which searches the possible route in the state space using a tree structure.

Figure 14 shows a brief process of using RRT to find a path from the initial pose to the goal which can be illustrated as follows: At first, the random tree has only one root, the initial node. It adds new nodes to the tree through sampling the space. A new node is the child node of the nearest node on the random tree. It is not completely randomly added into the random tree; instead, each new node has the possibility to grow towards the goal or grow randomly into the space. If the new node hits

https://doi.org/10.1017/S026357471800111X Published online by Cambridge University Press

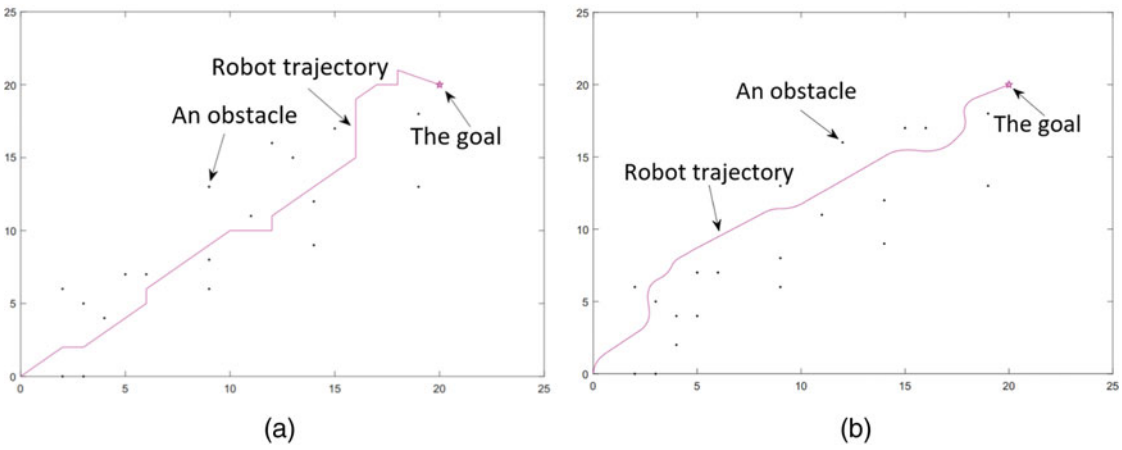


Fig. 15. Comparison with rapidly-exploring random tree.

the obstacle, it stops growing. Otherwise, the node will continue grow until the distance between its child node and the goal node is less than a certain threshold. As a Monte Carlo method, RRT adds random nodes into the state without considering robot dynamics; therefore, RRT will create a jagged trajectory which may not smoothly navigate a mobile robot.

The objective of testing scenarios in Fig. 14 is to verify that the proposed Q-learning algorithm can create a smoother trajectory than RRT. The heading direction of the robot is $\pi/4$. The target is located at $[\,20 \quad 20\,]^{\mathrm{T}}$. Since whether obstacles are static or not will not have a notable effect on the smoothness of resulting trajectory, all obstacles remain static. Figure 15(a) shows a trajectory created by RRT. The maximum number of samples to produce a trajectory is 100, the distance between a new node and its near node is 1 meter and the probability to select a node towards the goal is 0.1. Figure 15(b) shows a trajectory created by the algorithm proposed by this paper. A smoother trajectory means that the algorithm can navigate a robot under its dynamic limitation. The trajectory created by RRT is less smooth. Therefore, the robot will inevitably stop and rotate at the inflection point on the trajectory.

3.2.4. Comparison with another Q-learning algorithm. In this paper, the proposed algorithm is also compared with the Q-learning algorithm proposed in ref. [25] where action is defined as moving forward, turning left and turning right. The resulting trajectory using the algorithm proposed in ref. [25] and the algorithm proposed by this paper can be seen in Fig. 16(a) and (b), respectively. In both figures, the robot is initially located at $[\,0 \quad 0\,]^{\mathrm{T}}$, the target is located at $[\,5 \quad 5\,]^{\mathrm{T}}$ and the obstacle is located at $[\,2 \quad 3\,]^{\mathrm{T}}$ with the velocity of $[\,0 \quad -0.1\,]^{\mathrm{T}}$. The heading direction of the robot in Fig. 16(b) is $\pi/4$. In Fig. 16(a), the robot can only avoid the closest obstacle by turning left and right to a fixed angle. The robot is unable to move to the target with a smooth trajectory. Moreover, since the dynamics of the robot is not included in the algorithm, the robot is assumed to be able to turn to a fixed angle at any time which is not applicable to all robots due to the physical limitation of motors and drivers. In Fig. 16(b), the robot trajectory is smooth and the velocity of the robot at any time is reachable due to the consideration of robot dynamics.

3.3. The obstacle avoidance rate

In this paper, the obstacle avoidance rate is introduced as an indication to evaluate the effectiveness of training to obstacle avoidance. The obstacle avoidance rate is the fraction of all trials to avoid one static obstacle successfully. Since the Q-learning is a method based on the accumulation of past experiences, it is expected that the obstacle avoidance rate will enhance as the number of training scenarios grows.

The robot is firstly trained in different training scenarios, and then it is exposed in 200 different testing scenarios where the static obstacle is randomly located between the robot and the goal. As shown in Fig. 17, the obstacle avoidance rate increases as the training scenarios increases. At first, the robot was quite unlikely to avoid the obstacle and the obstacle avoidance rate was less than 5%, since the most part of the Q-table had not been trained. Then the obstacle avoidance rate increased,

https://doi.org/10.1017/S026357471800111X Published online by Cambridge University Press

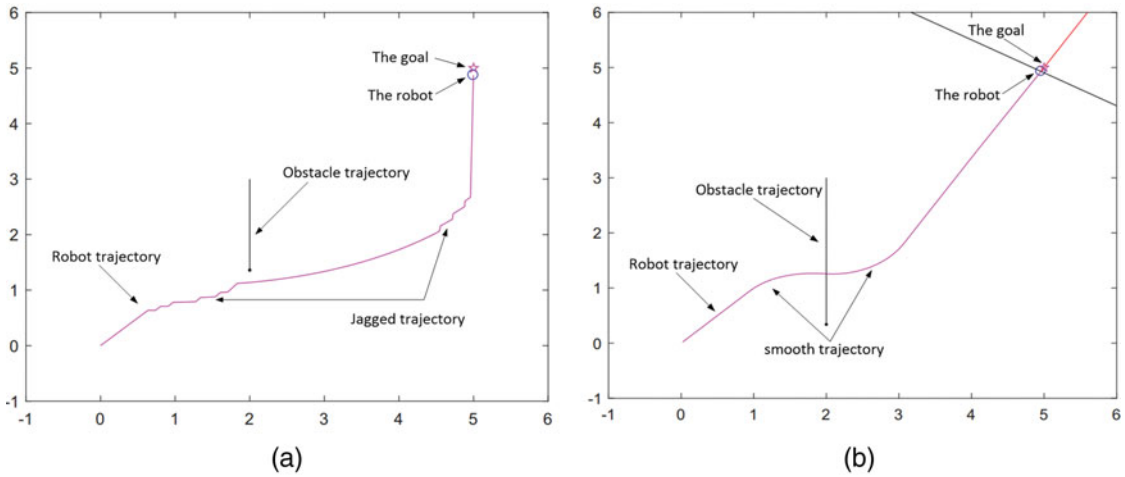


Fig. 16. Comparison with Q-learning method proposed in ref. [25].

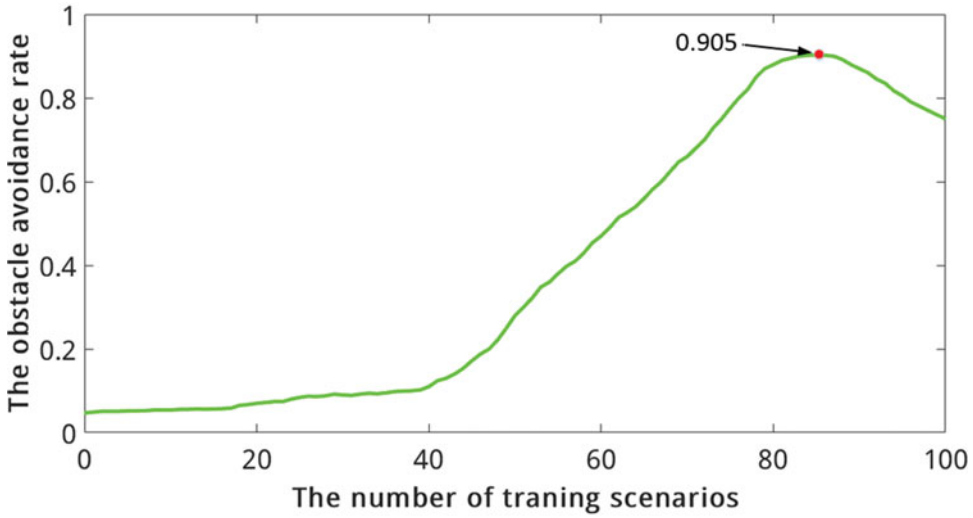


Fig. 17. The obstacle avoidance rate of the robot.

and finally it reached 90.5%. After that, there was a decrease in the obstacle avoidance rate which results from the overtraining the robot that caused the robot converges to specific actions.

4. Experiment

4.1. System description

Before conducting experiments, it is necessary to present a brief description of the important components in the robot system and the robot model in a virtual environment. As seen in Fig. 18, the robot is a self-made non-omnidirectional robot with two powered front wheels and two omnidirectional back wheels. Two brushless DC motors drive the robot. The maximum linear and angular velocity of the robot is 0.8 m/s and 0.35 rad/s, accordingly. Linear velocity of each motor is obtained through the hall sensors coaxial with motors. Angular velocity of the robot is obtained through a high resolution inertial measurement unit (IMU). The maximum linear and angular acceleration of the robot is 0.2 m/s and 0.87 rad/s, accordingly. The sensor used to perceive the environment is a SICK TIM551 laser range finder. A minicomputer with robotic operation system (ROS) is used to control the robot. Information is transmitted through a wireless router to a laptop to monitor the state of robot.

In order to train the robot in Gazebo, a simulator that takes kinematics, dynamics and other physical parameters into account should be applied, and the model of the robot should be built in Gazebo. First,

https://doi.org/10.1017/S026357471800111X Published online by Cambridge University Press

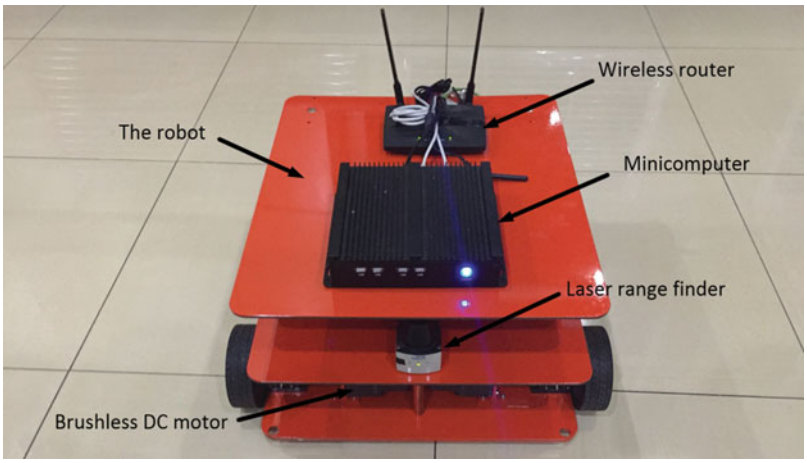


Fig. 18. The robot system.

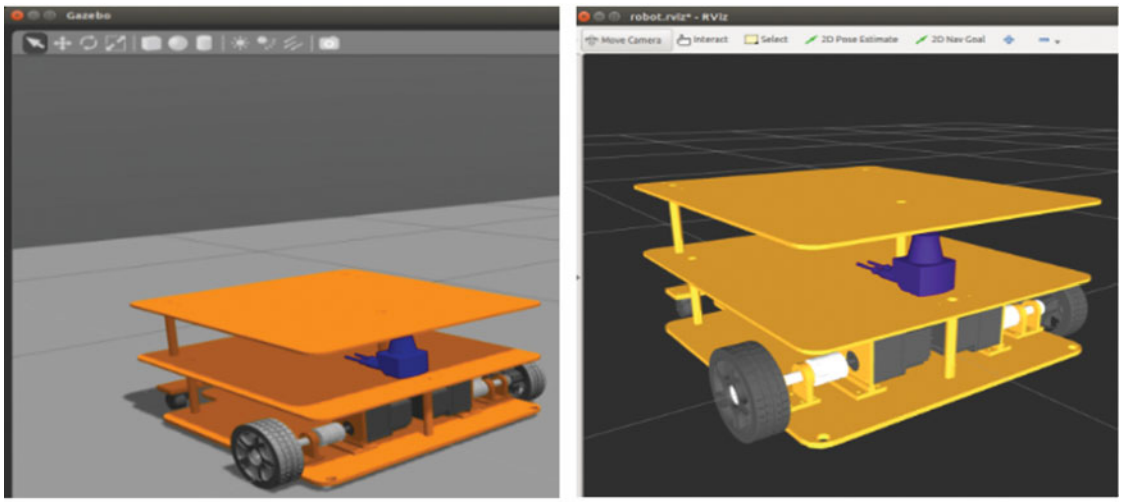


Fig. 19. Model in Gazebo and RVIZ.

the model was built in SOLIDWORKS, a widely used CAD software. Second, an STL format file was achieved from the model. After that, a Unified Robot Description Format (URDF) file, which is used to describe the robot or spatial information in ROS, was created and the resulting STL format file was added in the URDF file. Third, as can be seen in Fig. 19(a), the robot model in Gazebo can be achieved using that URDF file. ROS can load the robot model and demonstrate it in the RVIZ, a visualization environment in ROS. The model demonstration in RVIZ can be seen in Fig. 19(b).

4.2. Training the robot in Gazebo

The experiments are conducted in Gazebo and the real world. The method for localization is SLAM. Since training procedure of the Q-table requires many training scenarios, and the robot will inevitably collide with obstacles, it is unlikely to conduct the training procedure in a real environment. As the robot model in Gazebo is built from the real robot, Q-table trained in Gazebo can be used for the robot navigation in real world.

Similar to the training procedure in MATLAB, the robot is first exposed in simple environments and then environments with more obstacles. The Q-table is updated through the training. Four typical kinds of training scenarios can be seen in Fig. 20 corresponding to the training scenarios in Section 3.

https://doi.org/10.1017/S026357471800111X Published online by Cambridge University Press

Table II. Environment model and sensor parameter.

Parameter name	Parameter value	unit
Width of the experimental site	7	m
Length of the experimental site	10.5	m
IMU angular velocity resolution	0.1	rad/s
Laser range finder detecting range	10	m
Laser range finder angular resolution	0.0015	rad
Frequency of receiving sensor data	10	hz

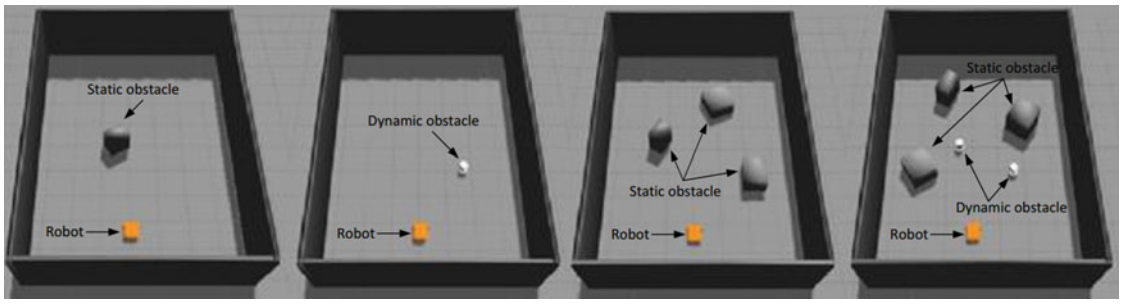


Fig. 20. Training scenarios in Gazebo.

4.3. Testing the robot in the real world

After the training procedure, the trained Q-table can be used to navigate the robot in real world. Several experiments of different kinds of testing scenarios had been carried out. This paper intends to demonstrate one of the most representative testing scenarios containing both static and dynamic obstacles.

The environment model and the parameter used for the experiment can be seen in Table II. The position and heading direction of the robot was determined using SLAM, as mentioned. As the detecting range and resolution of the laser range finder, as well as the IMU angular velocity resolution is good enough for the experiment, the robot position and orientation are assumed to be accurate throughout the experiment. Static obstacles are boxes of different shapes, and the dynamic obstacle is another self-made mobile robot covered by a paper cylinder.

4.3.1. Comparison with RRT. The main purpose of this experiment is to compare the resulting trajectory produced by the proposed algorithm and RRT. Without considering the robot dynamics, the trajectory produced by RRT should be less smooth than the proposed algorithm, as mentioned in the previous section. In this section, statistical indicators are introduced to qualitatively analyse the smoothness of trajectory.

As shown in Fig. 21(a), the initial position of the robot in experiment was (0, 0), and the direction of the robot was 0 degree. The position of the target was (0, 7.6). Figure 21(b) and (c) shows the trajectory produced by the proposed algorithm and RRT, correspondingly.

The resulting trajectory produced by RRT is composed of 11 discrete line segments, with 7 inflection points. The robot moves along the lines and rotates at inflection points. The angular variation of each infection point can be seen in Fig. 22(a). The resulting trajectory produced by the proposed algorithm has 312 inflection points, which can be seen in Fig. 22(b). As shown in Fig. 22, although trajectory produced by RRT contains less inflection points, the angular variation of each inflection points is much larger than the proposed algorithm.

Table III shows the statistical indicators of the comparison in angular variation of trajectory between the proposed algorithm and RRT. The average angular variation value and median angular variation value of RRT are 157 times and 27 times that of the proposed algorithm. The large difference in variance of angular variation between the proposed algorithm and RRT shows that the change of the trajectory of the proposed algorithm is gentler. All the statistical indicators in Table III verify that the proposed algorithm can produce a smoother trajectory for robot to navigate under its dynamic limitation.

https://doi.org/10.1017/S026357471800111X Published online by Cambridge University Press

Table III. The comparison in angular variation of trajectory between the proposed algorithm and RRT.

Statistical indicator	Proposed algorithm	RRT	Unit
Maximum angular variation	0.033937	1.5708	Rad
Minimum angular variation	−0.03491	−0.7854	Rad
Average angular variation	0.001992	0.31416	Rad
Median angular variation	0.016968	0.463	Rad
Variance of angular variation	0.000602	0.888269	rad^2

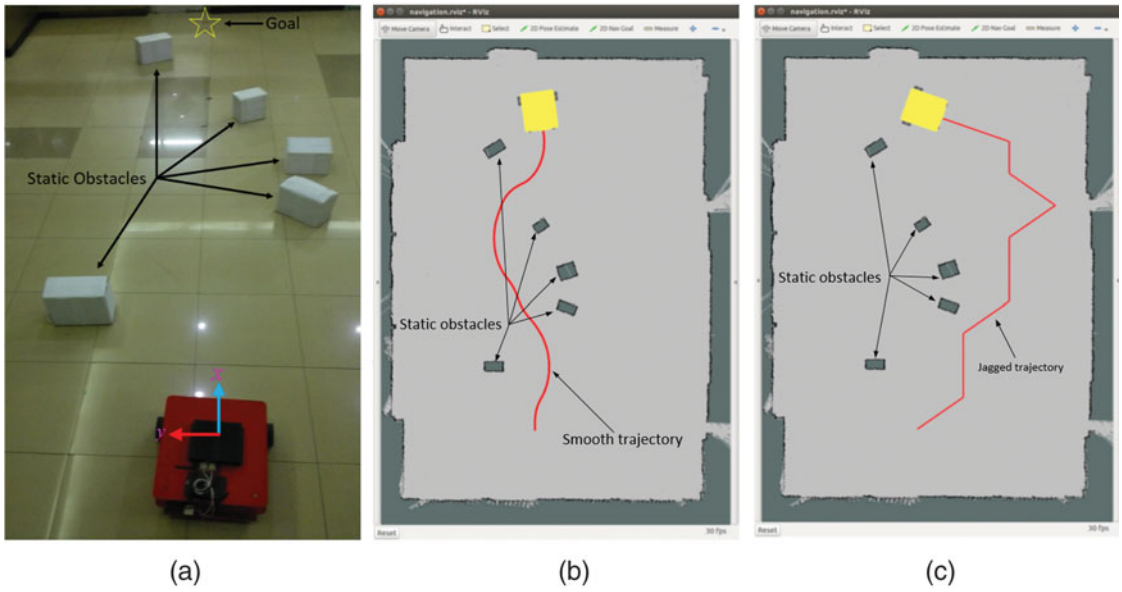


Fig. 21. Contrastive experiment of the proposed algorithm and RRT.

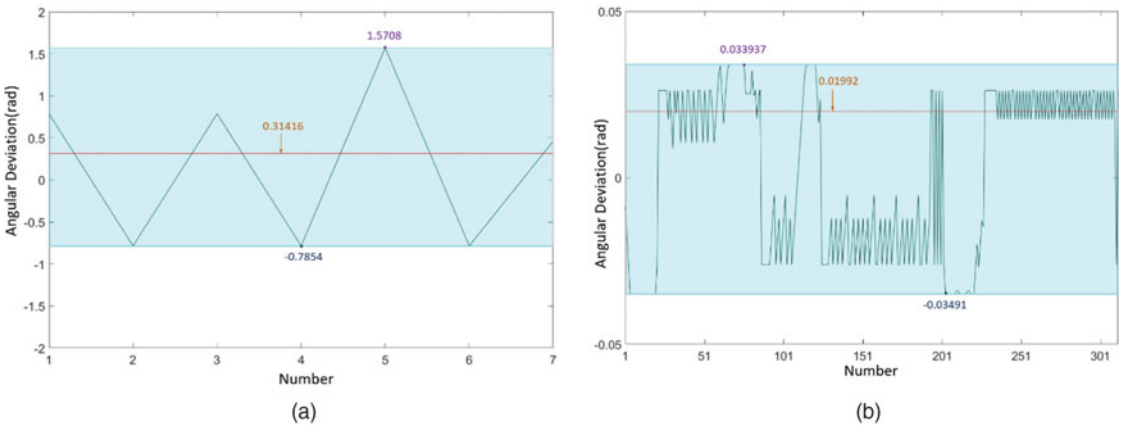


Fig. 22. Angular variation of infection points on the trajectory.

4.3.2. Testing the robot in a dynamic environment. The initial position of the robot in experiment was (0, 0), and the direction of the robot was 0 degree. The position of the target was (0, 7.2). The robot calculated its current state and chose the action with the highest Q-value to avoid the obstacle.

At first, the robot was in an SR; it moved towards the goal until it encountered the first static obstacle. It calculated its current state and chose the action with the highest Q-value to avoid the obstacle. Then it avoided the second and third static obstacle. After avoiding the dynamic obstacle, the robot finally reached the goal. The navigation process in real world is shown from Fig. 23(a) to (f). The corresponding trajectory of the robot is demonstrated in RVIZ, as shown in Fig. 23(g)

https://doi.org/10.1017/S026357471800111X Published online by Cambridge University Press

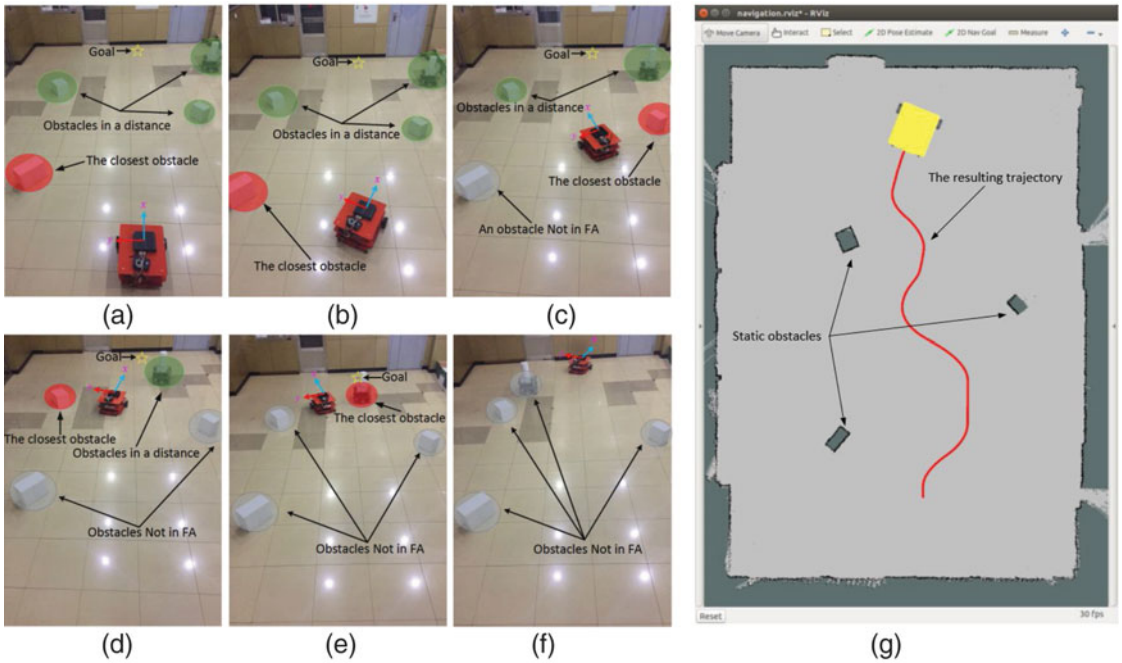


Fig. 23. The robot navigation in dynamic environment.

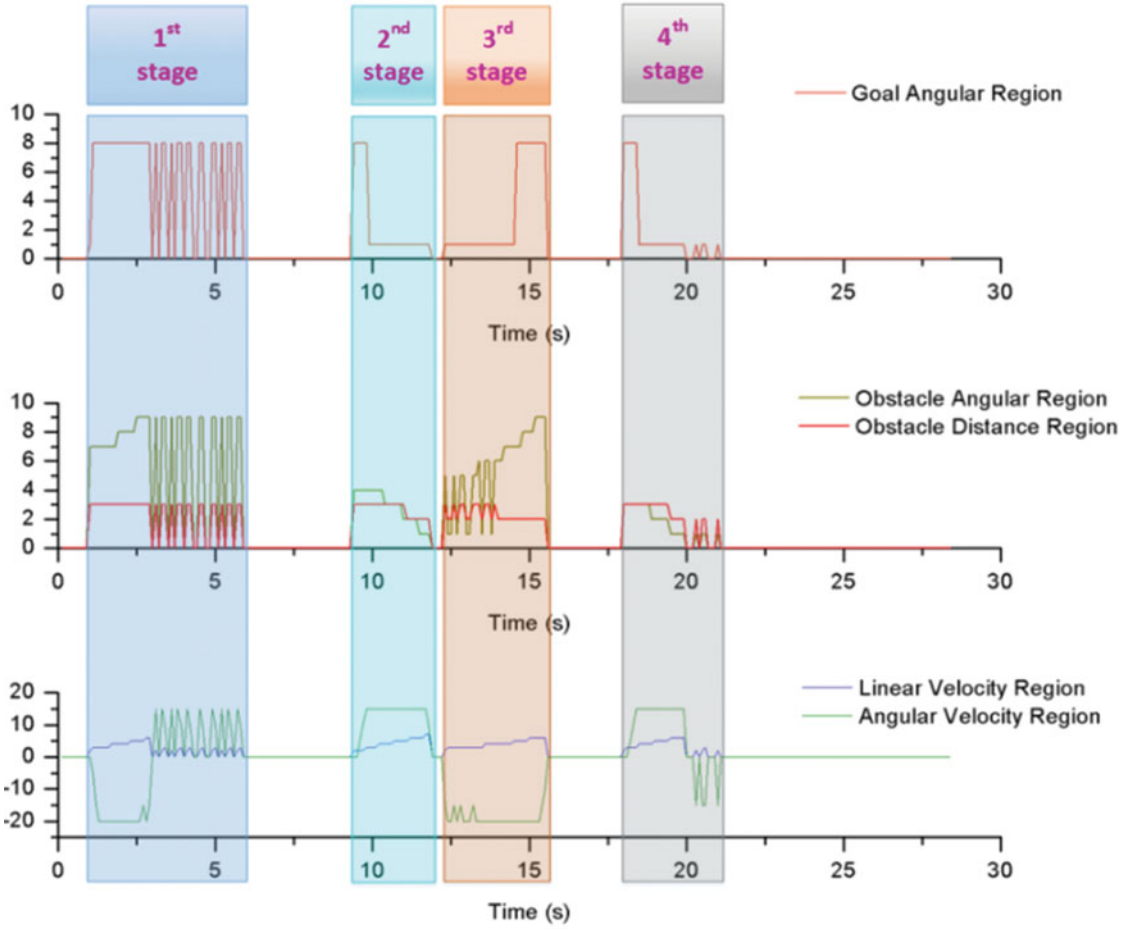


Fig. 24. Different states the robot passing through.

https://doi.org/10.1017/S026357471800111X Published online by Cambridge University Press

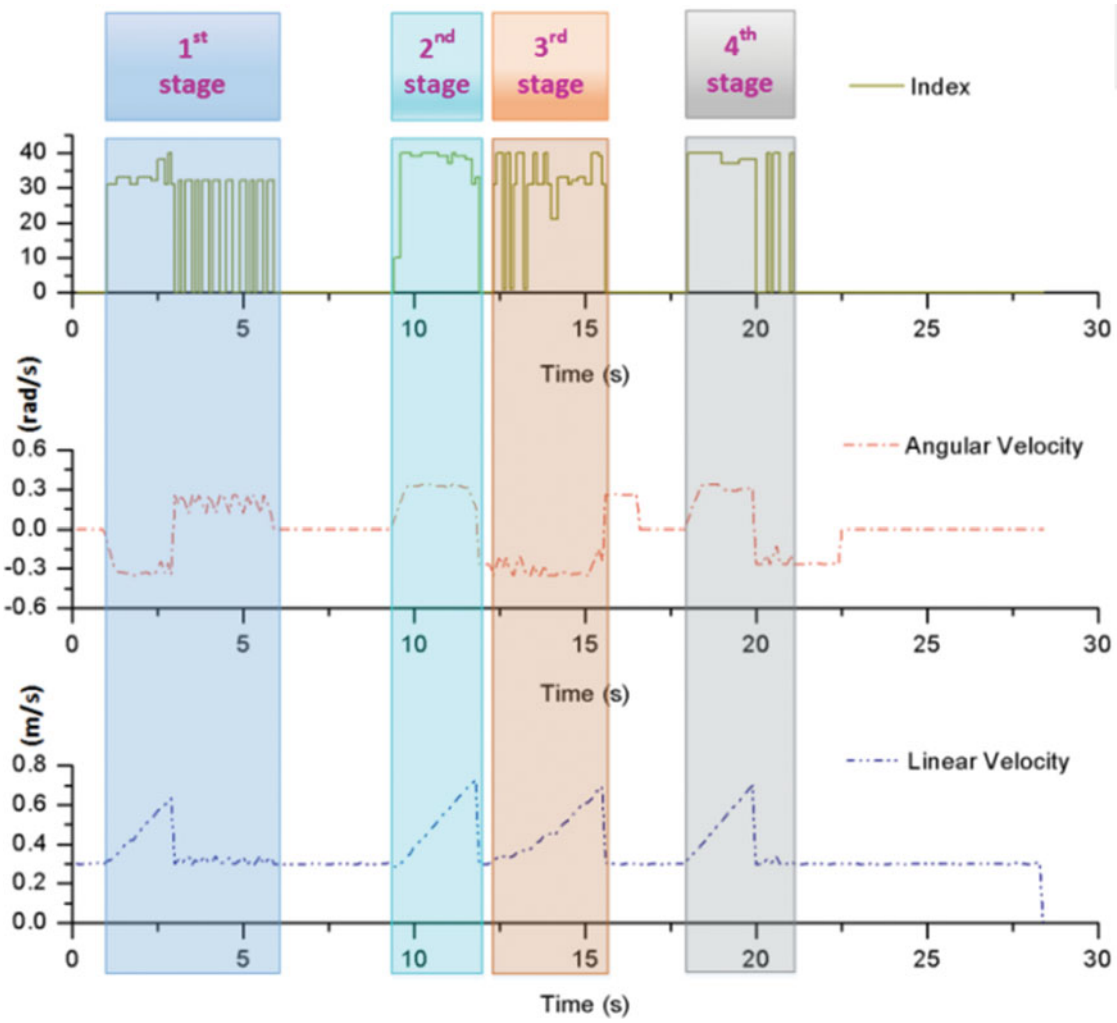


Fig. 25. The robot's index and the corresponding velocity order.

The state where the robot is located is shown in Fig. 24. The action of the robot is an index, as mentioned before. The index and the linear and angular velocity calculated from the index is shown in Fig. 25. If the robot is located in SR, the value of the state and index is 0.

The entire experiment process contained four stages of obstacle avoidance. The first stage started from 0.9 s to 5.8 s. At this stage, the robot encountered its first obstacle and was trying to avoid it. During the period from 3.1 s to 5.8 s, the robot frequently switched its state between safe and unsafe states, since the obstacle frequently appeared in the marginal area of FA. The second stage started from 9.4 s to 11.8 s. At this stage, the robot encountered its static obstacle. Shortly after the second stage, the robot encountered its third static obstacle. The third stage started from 12.3 s to 15.5 s. The fourth stage started from 18 s to 21 s. At this stage, the robot encountered its dynamic obstacle. After avoiding the dynamic obstacle, the robot finally reached the goal.

In each stage, there was a dramatic increase in linear velocity and the robot maintained high angular velocity, which shows the robot tends to choose high angular and linear velocity to avoid the chosen obstacle after training.

5. Conclusion

In this paper, a new Q-learning approach is introduced to navigate a robot in dynamic environments using human perception and evaluation. As a Q-learning approach is a model-free method, the robot does not require the information about the environment in advance. In the proposed Q-learning

https://doi.org/10.1017/S026357471800111X Published online by Cambridge University Press

approach, the definitions of state and reward function are given by imitating human reasoning, and actions are defined considering robot dynamics by introducing the dynamic window, which allows a robot to navigate under its dynamic restriction. The conducted simulations and experiments prove that the maximum obstacle avoidance rate can reach 90.5% after training. Moreover, the proposed algorithm is able to navigate a robot smoothly in dynamic environments and has successfully avoided the local minimum problem. Since the dynamics of robot is taken into consideration, all actions the robot takes are within its physical limitation.

As a matter of fact, using one navigation algorithm alone to guarantee absolute safety is pretty challenging. No matter how many training scenarios a robot has been through, applying the proposed algorithm to navigate a robot in real world may require some additional safety measures, for example, the robot must stop if the sensor perceives an obstacle that is too close to the robot. Also, if the goal is surrounded by obstacles which is therefore unreachable. The robot will stop trying to reach its goal after several attempts.

Acknowledgements

This work is supported by the National Natural Science Foundation of China (Grant No. 61673134), and the Natural Science Foundation of Heilongjiang Province of China (Grant No. LC2017022).

References

1. J. Minguez and L. Montano, "Sensor-based robot motion generation in unknown, dynamic and troublesome scenarios," *Robot. Auton. Syst.* **52**(4), 290–311 (2005).
2. E. Xidias, P. Zacharia and A. Nearchou, "Path planning and scheduling for a fleet of autonomous vehicles," *Robotica* **34**(10), 2257–2273 (2016).
3. L. Zhang, "Self-adaptive Monte Carlo localization for mobile robots using range sensors," *Robotica* **30**(2), 229–244 (2009).
4. X. Chen, Y. Xu, Q. Li, J. Tang and C. Shen, "Improving ultrasonic-based seamless navigation for indoor mobile robots utilizing EKF and LS-SVM," *Measurement* **92**, 243–251 (2016).
5. Y. Zhuang, Z. Syed, Y. Li and N. El-Sheimy, "Evaluation of Two WiFi positioning systems based on autonomous crowdsourcing of handheld devices for indoor navigation," *IEEE Trans. Mob. Comput.* **15**(8), 1982–1995 (2016).
6. C. Cadena *et al*., "Simultaneous localization and mapping: Present, future, and the robust-perception age," *IEEE Trans. Robot.* **30**(6), 1309–1332 (2016).
7. X. Hu, L. Chen, B. Tang, D. Cao and H. He, "Dynamic path planning for autonomous driving on various roads with avoidance of static and moving obstacles," *Mech. Syst. Signal Process.* **100**, 482–500 (2018).
8. O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Int. J. Robot. Res.* **5**(5), 500–505 (1986).
9. S. S. Ge and Y. J. Cui, "New potential functions for mobile robot path planning," *IEEE Trans. Robot. Autom.* **16**(5), 615–620 (2000).
10. S. S. Ge and Y. J. Cui, "Dynamic motion planning for mobile robots using potential field method," *Auton. Robots* **13**(3), 207–222 (2002).
11. Y. Chen, H. Peng and J. Grizzle, "Obstacle avoidance for low-speed autonomous vehicles with barrier function," *IEEE Trans. Control Syst. Technol.* **26**(1), 194–206 (2018).
12. S. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," *Res. Report* **1**, 293–308 (1998).
13. Arthur Richards *et al.*, "Spacecraft trajectory planning with avoidance constraints using mixed-integer linear programming," *J. Guidance Control Dynamics* **25**(4), 755–764 (2012).
14. Lin Yucong and S. Saripalli, "Path planning using 3D Dubins Curve for Unmanned Aerial Vehicles," *Proceedings of the International Conference on Unmanned Aircraft Systems IEEE*, Orlando, FL, USA (2014) pp. 296–304.
15. M. Duguleana and G. Mogan, "Neural networks based reinforcement learning for mobile robots obstacle avoidance," *Expert Syst. Appl. Int. J.* **62**, 104–115 (2016).
16. M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science* **349**(6245), 255–260 (2015).
17. L. Tai, S. Li and M. Liu, "A Deep-Network Solution Towards Model-Less Obstacle Avoidance," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, South Korea (Oct. 2016) pp. 2759–2764.
18. A. H. M. Findi, M. H. Marhaban, R. Kamil and M. K. Hassan, "Collision prediction based genetic network programming-reinforcement learning for mobile robot navigation in unknown dynamic environments," *J. Electr. Eng. Technol.* **12**, (2017).
19. C. J. C. H. Watkins, "Learning from delayed rewards," *Robot. Auton. Syst.* **15**(4), 233–235 (1989).
20. X. Xu, L. Zuo and Z. Huang, "Reinforcement learning algorithms with function approximation: Recent advances and applications," *Information Sci*. **261**, 1–31 (2014).

https://doi.org/10.1017/S026357471800111X Published online by Cambridge University Press

21. D. Gu and H. Hu, "Teaching robots to plan through Q-learning," *Robotica* **23**(2), 139–147 (2005).
22. W. D. Smart and L. P. Kaelbling, "Effective Reinforcement Learning for Mobile Robots," *Proceedings of the* IEEE International Conference on Robotics and Automation (ICRA), **4**, Washington, DC, USA (May 2002) pp. 3404–3410.
23. K. Macek, I. Petrovic and N. Peric, "A Reinforcement Learning Approach to Obstacle Avoidance of Mobile Robots," *Proceedings of the International Workshop on Advanced Motion Control*, Maribor, Slovenia (2002) pp. 462–466.
24. J. Lee, T. Kim and H. J. Kim, "Autonomous Lane Keeping based on Approximate Q-learning," *Proceedings of the International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, Jeju, South Korea (July 2017) pp. 402–405.
25. M. A. K. Jaradat, M. Al-Rousan and L. Quadan, "Reinforcement based mobile robot navigation in dynamic environment," *Robot. Comput.-Integr. Manuf.* **27**(1), 135–149 (2011).
26. D. Fox, W. Burgard and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robot. Autom. Mag.* **4**(1), 23–33 (1997).

https://doi.org/10.1017/S026357471800111X Published online by Cambridge University Press