Autonomous Robots (2021) 45:51–76
https://doi.org/10.1007/s10514-020-09947-4

Reinforcement based mobile robot path planning with improved dynamic window approach in unknown environment

Lu Chang[1] · Liang Shan[1] · Chao Jiang[2] · Yuewei Dai[1,3]

Received: 1 November 2019 / Accepted: 8 September 2020 / Published online: 30 September 2020

Abstract

Mobile robot path planning in an unknown environment is a fundamental and challenging problem in the field of robotics. Dynamic window approach (DWA) is an effective method of local path planning, however some of its evaluation functions are inadequate and the algorithm for choosing the weights of these functions is lacking, which makes it highly dependent on the global reference and prone to fail in an unknown environment. In this paper, an improved DWA based on Q-learning is proposed. First, the original evaluation functions are modified and extended by adding two new evaluation functions to enhance the performance of global navigation. Then, considering the balance of effectiveness and speed, we define the state space, action space and reward function of the adopted Q-learning algorithm for the robot motion planning. After that, the parameters of the proposed DWA are adaptively learned by Q-learning and a trained agent is obtained to adapt to the unknown environment. At last, by a series of comparative simulations, the proposed method shows higher navigation efficiency and successful rate in the complex unknown environment. The proposed method is also validated in experiments based on XQ-4 Pro robot to verify its navigation capability in both static and dynamic environment.

Keywords Robot navigation · Path planning · DWA · Q-learning · Evaluation function

1 Introduction

In the fields of industrial and agricultural production, intelligent logistics, space exploration and emergency assistance, the application of mobile robots has become more widespread. Robots can carry different tools such as robotic arms, rangefinders, fire extinguishers to accomplish different tasks. The basis for completing the task is that the robot can move autonomously and adaptively.

Path planning is one of the key technologies of mobile robots (Durrant 1994), which is described as finding a collision-free path connecting the starting point and the target in the working environment. The optimization indicator of the path can be selected as shortest length, minimum travel time, minimum collision probability, passing through specific locations, and so on. According to the environment awareness and planning scope, the path planning algorithms can be divided into (1) global path planning, where the robot finds a collision-free path connecting the starting point with the target under the known global environment map, which is usually conducted once and obtains a reference global path; (2) local path planning, where the robot moves according to the real-time information acquired by range sensors such as

This work is supported by the Natural Science Foundation of Jiangsu Province (BK20191286) and the Fundamental Research Funds for the Central Universities (30920021139).

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s10514-020-09947-4) contains supplementary material, which is available to authorized users.

✉ Liang Shan
shanliang@njust.edu.cn

Lu Chang
116110001162@njust.edu.cn

Chao Jiang
chao.jiang@uwyo.edu

Yuewei Dai
dywjust@163.com

[1] School of Automation, Nanjing University of Science and Technology, Nanjing, Jiangsu, People's Republic of China

[2] Department of Electrical and Computer Engineering, University of Wyoming, Laramie, WY 82071, USA

[3] School of Electronic and Information Engineering, Nanjing University of Information Science and Technology, Nanjing, Jiangsu, People's Republic of China

lidar (Zhang and Singh 2017), camera (Pinto et al. 2016), ultrasonic (Kim and Kim 2010), which is generally applied for obstacle avoidance and performed at each time step. In an actual workspace, the surrounding environment of the robot is relatively easy to be detected by these sensors. For example, the lidar obtains the nearest obstacle distance in each direction, camera obtains the image of a certain angle of view, and ultrasonic measures the obstacle speed. These information will help to model the surrounding environment. However, due to the lack of understanding of the whole workspace and some ambiguous factors like moving objects or pedestrians, the complete global map is difficult to be obtained. Therefore, the global path planning is hard to conduct in an unknown environment and the local path planning is a better method to realize the mobile robot autonomous navigation based on the surrounding environment.

To illustrate the application scenario of this paper, the robot task in a warehouse is taken as a motivating example. The warehouse robot needs to transport goods from the warehouse entrance to a certain position in the warehouse every day, but the distribution of other goods in the warehouse is unknown. The robot can only perceive its location in the warehouse, the position of the target and the surrounding obstacles (pedestrians or other goods) within perception range. In such circumstances, only the static global map acquired offline is known to the robot, and the robot moves mainly base on the local information it perceives in real time. Therefore, this paper aims to provide an effective local path planning approach to solve the robot navigation problem in unknown environments.

In recent years, machine learning (ML) techniques have been extensively exploited in autonomous robots which are endowed with adaptive learning ability based on accumulated knowledge. Reinforcement learning (RL) is an important branch of ML, which continuously updates the agent's action policies according to the feedback from the environment by trial-and-error. Benefiting from the development of simulation technology and the accumulation of measured data, the samples for training are increasingly available. RL is showing promising results in the field of mobile robot navigation for its powerful learning ability in complex environments. The structure of path planning algorithms is refined and the parameters of these algorithms are optimized by the richer data. In addition, early studies also used ML to enhance the awareness of the environment which is vast, dynamic or partially non-structured (Das et al. 2016).

Focusing on the local path planning algorithm, this work aims to improve the original dynamic window approach (DWA) to enhance the global navigation capability, and proposes a Q-learning based method to learn the optimal parameter adaptation in DWA. Q-learning algorithm (Watkins 1992) is one of the RL algorithms, which has many features that makes it suitable for robot path planning. Firstly, the Q-learning algorithm does not require any prior information of the environment, but accumulates the awareness of environment through interacting with it. Similarly, the robot does not have any information of the global map, but moves only based on the action policy and the real-time perception of the surrounding environment. Secondly, as an offline method, Q-learning algorithm makes past experience reusable for non-real-time learning, and the action to update the Q-table does not have to be taken. This learning method can effectively save the training cost and improve the training efficiency for mobile robots. Thirdly, the training data of Q-learning is unlabeled and not every single action is given an immediate reward from the environment. Most of the time, the overall reward of an action sequence is obtained after a complete interaction is finished. Similarly, the environment may not tell the robot whether a single action is right, but give a reward after each episode, that is, the end of each path planning (arriving at target, colliding with obstacles, being forced to stop or other termination conditions). Finally, the main goal of Q-learning algorithm is to train a general model for the same set of tasks. Similarly, the trained model (agent Q) on a map containing complex conditions can also obtain satisfactory planning results in other environments.

Our contribution in this paper is two-fold. Firstly, we improve the three evaluation functions in DWA by modifying the insufficiency of the original functions and adding two new functions to take into account special situations. These five evaluation functions enhance the efficiency of the robot navigation to the target under normal conditions, and solve the problem of local optimum caused by the lack of the global information. Secondly, DWA parameters adaptive tunning method based on Q-learning is proposed. To the best of our knowledge, our work in this paper is the first work to combine the DWA with Q-learning and adjust the weights of each evaluation function in DWA. Unlike other literatures (to be discussed in Section 2) where the robot speed is directly generated by Q-learning, in this paper, the Q-learning is used to adjust the parameters in DWA and the speed taken by the robot is still generated by DWA, which ensures the coherence and realizability of the path. The parameters adjusted dynamically are the weights of each evaluation function and the forward simulation time, which constitutes the action space of the agent Q. The state space and reward function are designed based on the robot pose and its relationship between the obstacle and target.

The rest of the paper is organized as follows. Section 2 introduces the related achievements on the common path planning algorithms, and analyzes their merit and demerit. Section 3 presents the definition and assumptions of the problem. Section 4 proposes the improved DWA by modifying and adding its evaluation functions, and implements the simulation to verify its effectiveness. Section 5 discusses the algorithm designed for parameter tunning in DWA based

on Q-learning, demonstrates the training process and the simulation results that shows the superiority of the trained agent. Section 6 conducts the hardware experiments to validate the navigation performance of the proposed method in real unknown environments. Section 7 concludes this paper.

2 Related works

The content of this paper covers the path planning algorithm of mobile robot and its combination with RL. A summary of the relevant works and results in these areas are as follows.

Over the past few decades, the path planning technology has been studied by many scholars. The methods based on different ideals and map descriptions have been proposed, such as potential field method (PFM), grid-based method, sampling-based planning (SBP), intelligent algorithm, DWA, RL, etc. In this section, we briefly introduce other kinds of path planning algorithms, then focus on the DWA and RL and analyses their shortcomings.

2.1 Overview of path planning algorithms

Grid-based method is a classic global path planning algorithm that divides the map into the two-dimensional grids. If there are obstacles close enough to a certain grid, this gird is regarded as an unreachable grid. Otherwise, it is a reachable grid. A path connecting the starting point and the target will be found among the reachable grid. Representative algorithms include A* (Hart et al. 1968), REA* (Zhang et al. 2016), limited-damage A*(Bayili and Polat 2011), D*(González et al. 2017), D Lite (Likhachev et al. 2008), Focused D*(Stentz 1995), etc. Among them, the A*algorithm and its improved methods are mainly used for the shortest path calculation on the static map, while the D* algorithm and its improved methods are for the situation where the obstacle and target are alterable. The improvement directions of the grid-based method generally are speeding up the computation (Zhao et al. 2018), increasing the searching direction to shorten the path (Xin et al. 2014) or ensuring the safety and feasibility of the trajectory (Chang et al. 2019). The grid-based method often depends on the known global map and clear obstacle position, and the result of this method is a reference path, which still remains the path tracking algorithm to guide the robot motion.

PFM (Khatib 1986) can be used in both global and local path planning. Its basic principle is to establish a virtual potential field where the target establishes a gravitational field while the obstacle establishes a repulsive one. The robot moves along the negative gradient direction of the superimposed field until it reaches the target. An enhanced PFM (Li and Chou 2016) combining with Levenberg-Marquardt (LM) algorithm and k-trajectory algorithm was proposed to overcome the inherent oscillation problem of the basic PFM. In order to assist the human-robot interaction, an online local PFM was defined by adapting animal motion attributes, where the improved potential field enabled the robot to move along the edge of the obstacle (Bence et al. 2016). The local optimum and the goal non-reachable with obstacles nearby (GNRON) are common problems of PFM. To solve these problems, a stronger attractive function was proposed to ensure that the robot reaches the target successfully and a rotational force was introduced to allow the robot to escape from the deadlock positions (Azzabi and Nouri 2019). PFM is proved to be an effective path planning algorithm where the effect of the environment is expressed as the action of force, and the desired velocity is calculated by the resultant force. The velocity can serve as the command signal of the motor. However, PFM requires the positions or velocities of all the obstacles, which is hard to be detected in real time. Solutions of the common problems like GNRON and trajectory oscillation highly depend on the global map.

The biological or physical laws have been summarized from the nature to serve various fields, which is called the intelligent algorithm. Some latest research about the intelligent algorithm applied in the robot path planning are as bellow. A DEQPSO method combining with the differential evolution (DE) algorithm and quantum-behaved particle swarm optimization (QP- SO) was proposed, and a safe and flyable path was generated in the presence of different threat environments for the unmanned aerial vehicle (UAV) by this method (Fu et al. 2013). To reduce the time of reaching the optimal path, a generalized intelligent water drops algorithm (IWD) was proposed based on fuzzy local. This method divides the graph into equal sections, compares the paths on them with a fuzzy inference system and determines the worth of each solution by the comparison (Monfared and Salmanpour 2015). The self-organising map (SOM) algorithm was proposed as a solution to the multi-robot path planning problem for active perception and data collection tasks. (Best et al. 2017). An evolutionary approach to solve the mobile robot path planning problem was proposed, which combined the artificial bee colony (ABC) algorithm as a local search procedure and the evolutionary programming (EP) algorithm to refine the feasible path found by a set of local procedures (Contreras-Cruz et al. 2015). An application of the bacterial foraging optimization (BFO) to the problem of mobile robot navigation was explored to determine the shortest feasible path to move from any current position to the target position in an unknown environment with moving obstacles (Hossain and Ferdous 2015). Intelligent algorithms may obtain the best result on a particular map by iterative processes and complex calculations, but are hard to execute when the global map is unknown or dynamic. Besides this, Intelligent algorithms usually only obtain reference paths.

Sampling based planning (SBP) is unique in the fact that planning occurs by sampling the configuration space (C-space). In a sense SBP attempts to capture the connectivity of the C-space by sampling it (Elbanhawi and Simic 2014). The main SBP algorithm includes probabilistic roadmap method (PRM), randomized potential planner (RPP), rapidly-exploring random trees (RRT), exploring/exploiting tree (EET), expansive space trees (EST), etc. The potential function based-RRT* incorporating the artificial potential field algorithm in RRT* was proposed to decrease the iterations number, and then leaded to more efficient memory utilization and an accelerated convergence rate (Qureshi and Ayaz 2016). The notion of flexible PRM was introduced to solve the problem of planning paths in the workplaces containing obstacles and regions with preferences expressed as degrees of desirability (Khaled et al. 2013). The EET planner deliberately trades probabilistic completeness for computational efficiency. When the available information captures the structure, the planner became increasingly exploitative. Otherwise, the planner increased local configuration space exploration (Rickert et al. 2008). Although some achievements have been made to solve the uncertainty (Agha-Mohammadi et al. 2014) or kinodynamic (Karaman and Frazzoli 2011) in active environments, there is still much room for improvement of SBP algorithm amongst moving uncontrollable obstacles and under stochastic dynamic and sensing conditions.

2.2 Path planning with dynamic window approach

Local path planning and obstacle avoidance problem was firstly formulated as constrained velocity space optimization problem and was solved by curvature velocity method (CVM) algorithm (Simmons 1996). On the basis of CVM, the more complete dynamic window approach (DWA) was proposed which considered the robot physical constraints, environmental constraints and current speed (Fox et al. 2002). DWA algorithm first obtains a sampling window of the speed based on the kinematics model and current speed of the robot, then generates the trajectory for each set of speed within the window, finally evaluates these trajectories by the evaluation function to find the optimal speed at the next moment. According to the sensor and robot pose information, the evaluation functions often consider three factors of speed, route angle and distance form obstacle. This method can directly obtain the desired linear and angular speed while considering the robot kinematics model, which makes the trajectory smoother and suitable for robot motion. Viewing the DWA as a model predictive control method, a version of DWA which is tractable and convergent was proposed using the control Lyapunov function (CLF) framework (Ogren and Leonard 2005). A method based on the integration of focused D* and DWA with some adaptations providing efficient avoidance of moving obstacles was proposed to enhance the navigation capability in partially unknown environments (Seder and Petrović 2007). To improve the vulnerable performance of DWA facing the trap situations, a global dynamic window approach (GDWA) with a scalar-valued function representing the distance from the goal point was proposed. (Kiss and Tevesz 2012; Maroti et al. 2013). An energy efficient local path planner in dynamic environments was presented, which extended the DWA to incorporate a cost function based on energy consumption predicted using a linear regression (LR) model (Henkel et al. 2016). To improve the acceptance of robot navigation and adapt the trajectory to mimic human behavior, the biomimetical DWA (BDWA) was proposed whose reward function was extracted from real traces of different motor disabilities navigating in a hospital environment (Ballesteros et al. 2017). A FGM-DWA algorithm was proposed to achieve the safe, smooth and fast navigation where the follow the gap method (FGM) was to guide the robot globally through the optimum gap and the DWA was to avoid the oscillations or collisions caused by the robot dynamics (Aykut and Volkan 2018).

However, the existing DWA is still unideal in two aspects. First, the mechanism of the existing evaluation function is not reasonable enough, which may lead to a better trajectory associated with a lower score. The number of evaluation functions is inadequate to deal with some special situations where the robot is very close to the target point or faces the spiral obstacle distribution. Second, in the path planning process, the degree of demand for each evaluation function is time-varying, which changes with the positional relationship between the robot, obstacle and target. However, the fixed weights will not be able to adapt to the dynamic situation, that is, the robot cannot decide which evaluation function should be taken into special consideration under different circumstances.

These two aspects may cause some unsatisfactory conditions. For example, the speed of the robot is too low when it is near the target, or the robot is too close to the obstacle when avoiding it. When the obstacle is in the spiral distribution, the robot may be trapped inside it and cannot escape (called local minima problem). When facing dense obstacles, the robot may not choose the shorter path between them but bypasses these areas, which makes the overall path longer.

For the first aspect, GDWA was proposed to eliminate the previously mentioned local minima problem by defining a navigation function (NF) (Brock and Oussama 1999). NF is a scalar valued function defined on the reachable location in global map and has exactly one minimum, namely the target position. The NF values at each reachable position were obtained by global path planning algorithm like wavefront propagation (WP) (Kiss and Tevesz 2012) or Dijkstra (Maroti et al. 2013). The evaluation function of GDWA is designed based on the defined NF using its value and gra-

dient, which is still a form of weighted summation. GDWA eliminates the local minima problem in most cases, but may fail to converge to the target position or still get trapped due to the unreasonable weights or zero velocity. More importantly, the definition of NF highly depends on the accurate global map, and the ideal of GDWA is similar to functional optimization. In this paper, we know less about the global map or the obstacle in it, so there is no such NF to optimize. We only use the real-time detection of the surrounding obstacle.

For the second aspect, a version of GDWA changed the evaluation function into a no-weighted from. This function only considers the position of the robot and the value of NF, which makes it contain only one item (Kiss 2012). Another kind of method is adjusting each weight separately. For example, the weight of evaluation subfunction considering the velocity became adaptive according to the distance from the obstacle (Wang et al. 2019), or the weight of one subfunction was adjusted when other weights were fixed (Xu 2017). Without the global information or NF serving as the basis for optimization, the existing optimization algorithms are hard to apply. Adjusting weights separately splits the relationship between them, which may miss the best weights combination. In contrast, our method adjusts all weights jointly and choses different combinations for different conditions.

2.3 Path planning with reinforcement learning

One of the most interesting approaches applied in path planning is RL. In the process of RL, the robot learns the best strategy of path planning by interacting with the environment, just like some living creatures. In the proposed approach, the robot path planning was solved with Q-learning, which was first introduced for learning with delayed rewards (Kröse 1995). One of the first application of Q-learning in robot navigation utilized the example trajectories to bootstrap the value function approximation and split the learning into two phases (Smart and Kaelbling 2002). In order to improve the hit rate and reduce the size of the Q-table, the number of the states was limited based on a new definition for the states space (Jaradat et al. 2011). A new methodology was proposed for Q-learning with improved particle swarm optimization (IPSO) to reduce the complexity of the classical Q-learning and the robot's energy consumption by saving turning angles and path length (Das et al. 2015).

In recent years, there has been a new trend of combining Q-learning with deep learning. Under the circumstance where the global information is available, a new approach using Q-learning and a neural network planner was proposed to solve the problem of autonomous movement in environment containing both static and dynamic obstacles (Duguleana and Mogan 2016). A model-based path planning method was proposed with Q-learning, whose Q-value was approximated with a neural network named deep Q-learning and the reward was calculated from the grid map (Sharma et al. 2017). A log-based reward function was introduced in deep Q-learning to increase the success rate of obstacle avoidance for the wheeled mobile robot (Mohanty et al. 2017). Q-learning was also applied in collaborative path planning system with holonic multi agent architecture (Lamini et al. 2015). Combining with Boltzmann policy to avoid trapping in local optimum, Q-learning can remarkably improve the efficiency of the multi-robot system, reducing the number of explorations and converging the process (Wang et al. 2014).

In the aforementioned literatures that applied Q-learning in the robot navigation, they optimized the generation mode of Q-values or the selection of reward functions to reduce the operating cost of programs and robots, or promoted its application objects to multiple agents. However, the state apace only considers the position relationship of robot with the obstacle or target, but ignores the orientation and velocity of the robot, which may not fully describe the robot state. In addition, the action apace is relatively simple, generally are turning right or left, moving one step at several directions (especially in a grid map). These blunt actions tend to make the path too incoherent to track and some better paths be ignored. In contrast, Q-learning applied in our proposed method is to enhance the performance of DWA, which preserves the advantage of feasible and superior path of DWA.

2.4 Summary and motivation

As mentioned, most of the existing path planning algorithms rely heavily on known static global maps. If the map has dynamic obstacles, their state (location, shape or velocity) needs to be highly understood. Therefore, these algorithms may not perform well in the environment with unknown static and dynamic obstacles. In addition, they often obtain the reference path rather than velocity command, so more sensors or methods should be adopted to track the path, which increases the complexity of the navigation and computation for the mobile robot.

In this paper, the navigation scenario contains only the outline of the global map but not the obstacles in it, and the obstacles are detected in real time by the sensors on the robot, so the path planning algorithm can only use the information of surrounding obstacles. Meanwhile, to simplify the navigation framework, the algorithm is better to obtain the desired velocity directly instead of the desired path. Therefore, we find that DWA meets these requirements and is suitable as the path planning algorithm. As for the two defects of DWA that analyzed above, we correct them by enhancing the evaluation functions and parameter self-adaption based on Q-learning.

3 Problem definition and assumptions

Similar to the task in a warehouse above, we define the navigation problem considered in this paper in general. The initial position of the robot and target are preset, and the robot only knows the outline of the global map (like the walls of the warehouse), which means the obstacles inside the map is completely unknown. The task of robot is to reach the target while keeping the safe distance from the obstacles. The robot can reach the target with any velocity and does not need to stop at the target.

Beside the outline of the global map, all the information the robot can get are obtained by the various sensors equipped (gyroscope, lidar, encoder, etc.). At each time step, the information required by the proposed algorithm can be obtained based on these sensors. Concretely, we make the following assumptions about the navigation problem.

Assumption 1 The location, orientation and velocity of the robot in the global map are known at each time step.

Assumption 2 The position of the target in the global map is known at each time step.

Assumption 3 The distance of the nearest obstacle in each direction of a circle is known at each time step (realized by the lidar on the robot). The structure of these data is expressed as $[\Theta\ D]$, where $\Theta = [\theta_1\ \theta_2\ \cdots\ \theta_n]^T$ are angles in the range $[0, 2\pi)$ with a certain resolution; $D = [d_1\ d_2\ \cdots\ d_n]^T$ are the distances of the nearest obstacle in the i^{th} angle, d_i is not greater than the detection range; n is the angle resolution of the lidar, meaning that n data pairs of angle and distance will be obtained after the lidar scanning a circle.

The robot in this paper is differential driven, and we only consider its forward and rotational motion. The kinematics model of the robot is shown in Eq. (1).

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = \omega \\ \dot{v} = a \\ \dot{\omega} = \alpha \end{cases} \tag{1}$$

where $[x, y]$ is the position of the robot; θ is the orientation of the robot; v and ω are the linear velocity and angular velocity of the robot; a and α are the linear acceleration and angular acceleration of the robot.

4 Improved DWA algorithm

4.1 Basic DWA algorithm

DWA algorithm transforms the path planning problem into the constrained optimization problem of the velocity space, and controls the robot motion by outputting the real-time optimal speed.

The speed set (v, ω) is restricted to the following restrictions:

(1) The velocity of robot cannot exceed the maximum.

$$\begin{cases} 0 \le v_t \le v_{\max} \\ -\omega_{\max} \le \omega_t \le \omega_{\max} \end{cases} \tag{2}$$

where v_t and ω_t are the linear speed and angular speed of the robot at time t; $v_{\max}$ and $\omega_{\max}$ are the maximum velocity.

(2) The acceleration of robot cannot exceed the maximum.

$$\begin{cases} v_t - a_{\max}\Delta t \le v_{t+1} \le v_t + a_{\max}\Delta t \\ \omega_t - \alpha_{\max}\Delta t \le \omega_{t+1} \le \omega_t + \alpha_{\max}\Delta t \end{cases} \tag{3}$$

where v_{t+1} and ω_{t+1} are the linear speed and angular speed of the robot at time t+1; $a_{\max}$ and $\alpha_{\max}$ are the maximum linear acceleration and angular acceleration of the robot; Δt is the time step.

The feasible speed sets are obtained by sampling the velocity space which matches the kinematics model, and the predicted trajectories are generated by these speed sets and the trajectory prediction time period T. These trajectories, which will be some arcs, are scored by the evaluation function to find the best one and the corresponding speed set. The original evaluation function is shown in Eq. (4).

$$\begin{aligned} J(v,\omega) = \sigma[&w_1 \cdot heading(v,\omega) + w_2 \cdot obdist(v,\omega) \\ &+ w_3 \cdot velocity(v,\omega)] \end{aligned} \tag{4}$$

Equation (4) contains three evaluation subfunctions and their coefficients w_1, w_2, w_3; σ denotes the normalization process.

Concretely, function $heading(v, \omega)$ calculates the angle θ between the orientation angle of the robot at the end of the predicted trajectory and that of the robot position to the target. This function evaluates the degree of the trajectory being toward the target. The schematic diagram of angle θ is shown in Fig. 1a. The smaller the angle is, the higher the score will be.

Function $obdist(v, \omega)$ calculates the minimum distance from each point of the predicted trajectory to the obstacle. This function evaluates the degree of the trajectory being away from the obstacle. The schematic diagram of finding the minimum distance is shown in Fig. 1b. Taking three points as an example, d_2 is the minimum distance. The farther the distance is, the higher the score will be. If the minimum distance of a trajectory is less than the safe distance, this trajectory will be discarded directly and removed from the sampling space.

Function $velocity(v, \omega)$ evaluates the linear and angular velocity of the robot, which favors fast and straight move-

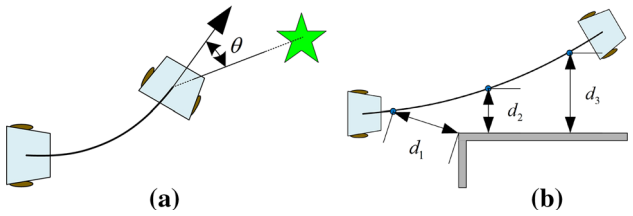


Fig. 1 Schematic diagram of **a** the angle θ, **b** the minimum distance d

ment. The higher the linear speed and the smaller the angular velocity are, the higher the score will be.

4.2 Improved DWA algorithm

The basic DWA algorithm only contains three kinds of evaluation functions, which is not adaptable to unknown environments. The robot may hesitate in front of the obstacle for a long time and be easy to fall into the spiral obstacle, caused by the lack of the number of the existing evaluation functions and the mechanism of these function being not quite appropriate. In this section, we modify the existing evaluation functions to improve the score of better trajectories in each specific subfunction and add two new subfunctions to consider more complex situations.

4.2.1 Modified subfunction *heading'* (v, ω)

This modification is to reduce the negative effects that function $heading(v, \omega)$ may cause when the robot approaches the target, whose method is changing the reference position used to calculate the angle.

For the predicted trajectory to be prospective enough, the trajectory prediction time period T is generally dozens of times of the time step, that is, $T = (10\text{–}30)\Delta t$. Function $heading(v, \omega)$ calculates the angle based on the final robot position of the predicted trajectory, which is the position the robot can reach by moving at the current speed for the whole time period T. However, the speed will be re-selected after Δt, which means that the robot will only move along a short section of the whole predicted trajectory.

Figure 2 shows the comparison of the two predicted trajectories. Obviously, trajectory A is more close to the ideal one, which means the velocity corresponding to trajectory A is better. So the score of trajectory A should be higher than that of trajectory B. If the reference position is at the end of the predicted trajectory, we can see from Fig. 2 that $\theta_{B,o} < \theta_{A,o}$, which makes the trajectory A get the lower score. If the reference position is set somewhere close to the current position, we can see from Fig. 2 that $\theta_{A,n} < \theta_{B,n}$. It means trajectory A will get the higher score, which makes the robot turn to the target in time.

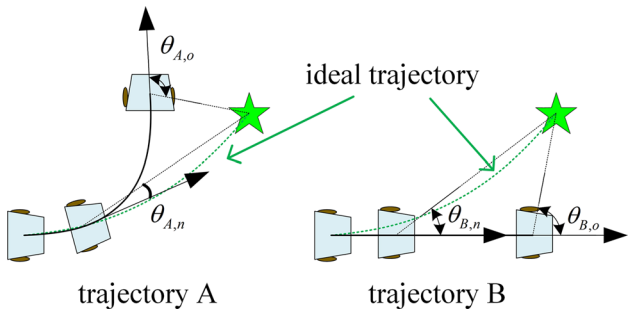


Fig. 2 Trajectory comparison for function $heading(v, \omega)$. Notations: 1) The black arc means the predicted trajectory. 2) The green arc means the ideal trajectory. 3) The green star means the target. 4) The grey rectangle (if there is) means the obstacle. 5) The ideal trajectory, target and obstacle are the same in the comparison. These notations are also applicative below (Color figure online)

The distance from the current position, which is called the travelled distance, is directly determined by the number of time steps. In practice, the travelled distance d should be set first, and the time steps which are required to move such distance can be estimated by the current speed. Generally, when d is short enough, the robot trajectory can be considered a straight line, so the number of the time steps can be estimated by Eq. (5).

$$n_{\Delta t} = fix(d/v) \tag{5}$$

where v is the current linear speed; $fix(\cdot)$ is the rounding function.

When operating the function $heading(v, \omega)$, the reference position of the robot is the position after $n_{\Delta t}$ time steps in the predicted trajectory.

Finally, function $heading(v, \omega)$ returns an angle between the orientation of the new reference position and the line pointing from the new reference position to the target.

4.2.2 Modified subfunction *obsdist'* (v, ω)

This modification is to reduce the negative effects that function $obsdist(v, \omega)$ may cause in the environment with many obstacles, whose method is similar to that described in Section 4.2.1.

Figure 3 shows the comparison of two predicted trajectories. Obviously, trajectory A is more close to the ideal one, which means the velocity corresponding to trajectory A is better. In terms of the original function, as can be seen from Fig. 3, trajectory A will be discarded since $d_{A,o}$ is less than the safe distance. Actually, all the trajectory similar to trajectory A will suffer the risk of being discarded while the trajectory similar to trajectory B can get a higher score, which makes the navigation poor. Function $obsdist(v, \omega)$ has two effects: trajectory scoring and discarding. The trajectory discarding

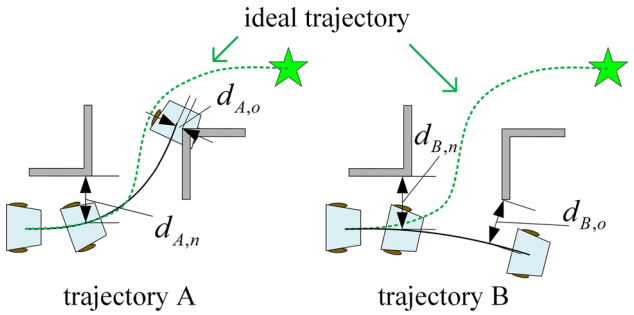


Fig. 3 Trajectory comparison for function $obsdist(v, \omega)$

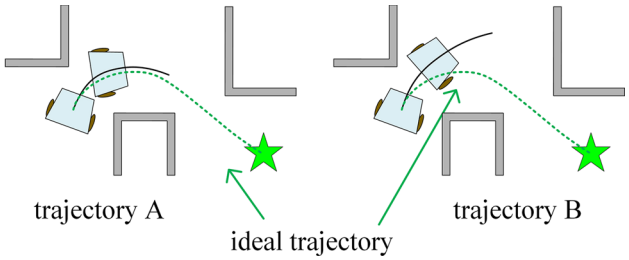


Fig. 4 Trajectory comparison for function $velocity(v, \omega)$

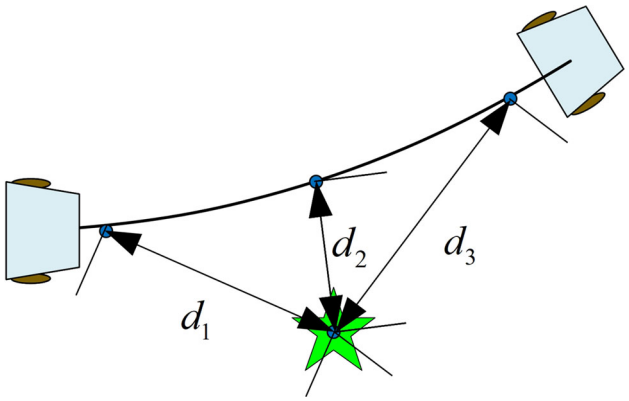


Fig. 5 Schematic diagram of the minimum distance d

part should be more cautious as it removes a possible speed set directly which will not be evaluated by other subfunctions, and the speed is re-selected at each time step. So the trajectory similar to trajectory A that will not collide with the obstacle temporarily should not be discarded. The trajectory scoring part remains unchanged. When discarding the trajectory, the minimum distance from the obstacle should be calculated from the beginning of the trajectory to the position after several time steps. As can be seen from Fig. 3, $d_{A,n}$ is similar to $d_{B,n}$, and both are more than the safe distance, so that the two trajectories can be retained and evaluated by other subfunctions.

The estimation of the number of time steps is similar to the Eq. (5). Since the purpose of this function is to avoid obstacles, the value of the travelled distance d should be slightly larger than that of the function $heading(v, \omega)$.

Finally, function $obsdist(v, \omega)$ returns the shortest distance between the predicted trajectory and obstacles.

4.2.3 Modified subfunction *velocity'*(v, ω)

This modification is to reduce the negative effects that function $velocity(v, \omega)$ may cause when the robot need to move slowly and circuitously.

The score of this function is determined by the linear and angular speed of the trajectory, which is to allow the robot to move as fast as possible and avoid unnecessary turning. Figure 4 shows the comparison of the two predicted trajectories that share the same linear speed, while the angular speed of trajectory A is greater than that of trajectory B. Obviously, trajectory A is better, but trajectory B will get the higher score using the original function where the scores of the two speeds are independently calculated and simply added. Under the same situation, the higher the linear speed is, the faster the robot reaches the target. Therefore, the scoring function of the linear speed does not need to change. In the environment with many obstacles, it is more reasonable to move circuitously with slower linear speed and larger angular speed, and the larger angular speed will not increase the risk of collision with obstacles too much when the linear speed is slower. Therefore, we modify the scoring function of the angular speed as Eq. (6).

$$\omega' = \omega_{\max} - k\frac{v}{v_{\max}}\omega \tag{6}$$

where the term $\frac{v}{v_{\max}}$ is added to measure the current linear speed and k is a parameter.

Finally, function $velocity(v, \omega)$ returns the score of the current speed which is calculated by Eq. (7), where σ denotes the normalization of the two speeds.

$$velocity'(v, \omega) = \sigma(\omega' + v) \tag{7}$$

4.2.4 Added subfunction *goaldist*(v, ω)

Function $goaldist(v, \omega)$ calculates the minimum distance from each point of the predicted trajectory to the target. This function is to evaluate the degree of the trajectory being close to the target. The weight of this function is w_4. Figure 5 shows the method to find the minimum distance. Taking three points as an example, d_2 is the minimum distance. The closer the distance is, the higher the score will be.

This function is added to enhance the motion trend towards the target point especially when there are obstacles around it. Figure 6 shows the situation to compare the two predicted trajectories. Obviously, trajectory A is more close to the ideal

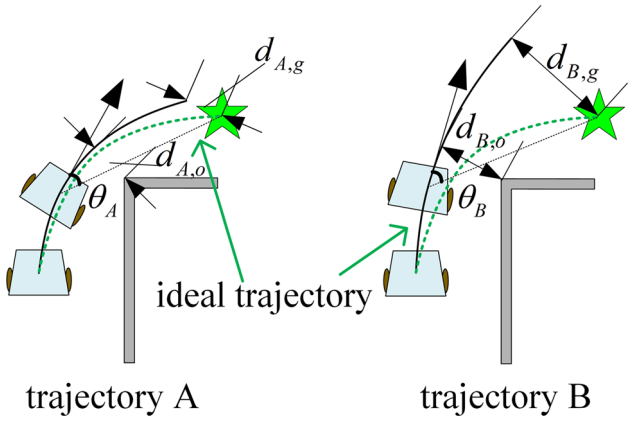


Fig. 6 Trajectory comparison for function $goaldist(v, \omega)$

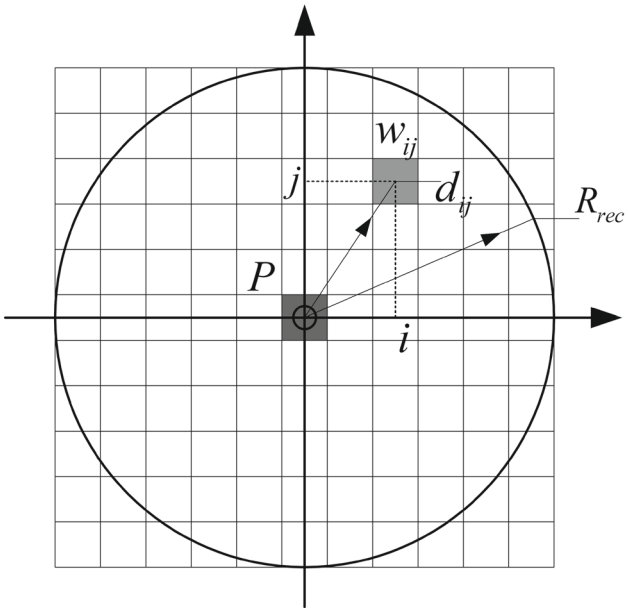


Fig. 7 Updating method of the cell cost

one, which means the velocity corresponding to trajectory A is better. In terms of the original evaluation functions, we can see from Fig. 6 that $\theta_A < \theta_B$ and $d_{A,o} < d_{B,o}$ which results in the scores of the two trajectories being almost the same and it is hard to choose the better one. After adding function $goaldist(v, \omega)$, we can find in Fig. 6 that $d_{A,g} < d_{B,g}$, which helps trajectory A to get the higher score.

Note that the subfunction $heading(v, \omega)$ is also to make the robot move toward the target, but keeping these two subfunctions is still necessary as they have different effects and applicable conditions. Though the reference position in subfunction $heading(v, \omega)$ has been modified, it is still possible that this position in the trajectory overshoots the target, especially when the robot is very close to the target, which will adversely impact the navigation effect. At this time, subfunction $goaldist(v, \omega)$ can help the robot turn to the target with a steady speed. On the other hand, if the robot is still far away from the target, the minimum distances to the target of all the trajectories will be large and similar, which greatly limits the discrimination of subfunction $goaldist(v, \omega)$. At this time, the robot mainly depend on the subfunction $heading(v, \omega)$ to navigate to the target. Actually, the subfunction $goaldist(v, \omega)$ is designed only to work when the minimum distance to the target is less than $2m$. In general, these two subfunctions contribute to the robot moving to the target from the view of orientation and distance, and work in different conditions, so they are both indispensable.

Finally, function $goaldist(v, \omega)$ returns the shortest distance between the predicted trajectory and the target.

4.2.5 Added subfunction *oscillation*(v, ω)

Function $oscillation(v, \omega)$ is to evaluate the proximity of the predicted trajectory and the historical trajectory. The weight of this function is w_5. We describe the map as a two-dimensional cell group H with a certain resolution, which is similar to the grid map, to store the information of historical trajectory. Each cell has a cost to describe how close the historical trajectory is to this cell. The higher the value of the cell is, the easier it is for the robot to stagnate or wander near this position. The cell group H is updated after each robot movement. The cost of the cell on or near the robot position will be updated with the method shown in Fig. 7, where P is the robot position at time t and R_{erc} is the radius of the circular area affected by the robot where cell cost should be updated.

We establish the coordinate system Ω_P with the origin P, where d_{ij} means the distance from the cell with coordinates (i, j) to the origin P and w_{ij} means the cost should be increased on the cell (i, j) at time t which is calculated by Eq. (8).

$$w_{ij} = \frac{(R_{rec} - d_{ij})v_t}{R_{rec}v_{max}} \tag{8}$$

Adding the speed term to the cost calculation can avoid a rapid increase in the cost of the surrounding cell when the robot speed is slower.

Figure 8a shows the trajectory of the robot obtained by the improved evaluation function in the map with simple obstacles, and the cell group H generated by this trajectory. The color of each cell represents its cost, i.e. the darker the color is, the higher the cost is. The score of the function $oscillation(v, \omega)$ of a trajectory is the total cost of the cell swept over by this trajectory. For example, as for the trajectory shown in Fig. 8b, its score is the sum of the cost of all the red cell.

This function is added to prevent the robot from returning to the place travelled or going around in circle, which is espe-

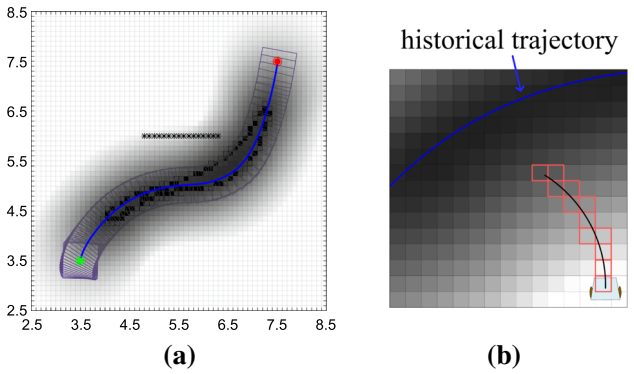


Fig. 8 **a** Trajectory of the robot with its cell group H, **b** Cells to calculate the $oscillation(v, \omega)$ value

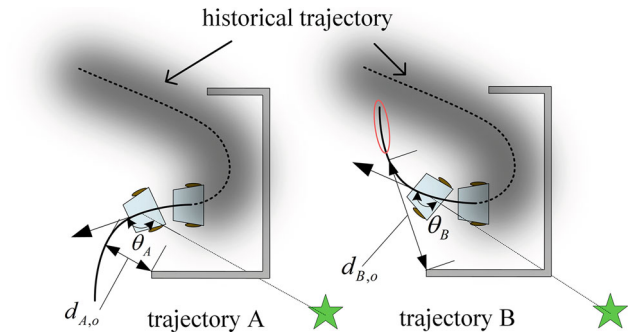


Fig. 9 Trajectory comparison for function $oscillation(v, \omega)$

cially important when the obstacle is in spiral distribution. Figure 9 shows the comparison of the two predicted trajectories. Obviously, the robot should move along trajectory A rather than trajectory B. In terms of the original evaluation functions, we can see from Fig. 9 that $\theta_A < \theta_B$ and $d_{A,o} < d_{B,o}$, which results in the scores of the two trajectories being almost the same and it is hard to choose the better one. After adding function $oscillation(v, \omega)$ we can find in Fig. 9 that the gird cost in the red circle of trajectory B is larger while trajectory A overlaps little with the historical trajectory. This difference means the proximity of trajectory B and historical trajectory is significantly higher than that of trajectory A, which helps trajectory A to get the higher score.

Finally, function $oscillation(v, \omega)$ returns the total cost of all the cell swept by the predicted trajectory.

After modifying and adding the evaluation subfunctions, the improved evaluation function is shown in Eq. (9).

$$\begin{aligned} J'(v,\omega) &= \sigma[w_1 \cdot heading'(v,\omega) + w_2 \cdot obdist'(v,\omega) \\ &\quad + w_3 \cdot velocity'(v,\omega) + w_4 \cdot goaldist(v,\omega) \\ &\quad + w_5 \cdot oscillation(v,\omega)] \end{aligned} \tag{9}$$

Table 1 Parameters of the robot

Parameter name	Parameter value
Safety radius R	0.4 m
$v_{\max}$	1 m/s
$\omega_{\max}$	2π rad/s
$a_{\max}$	0.5 m/s^2
$\alpha_{\max}$	6π rad/s^2
Resolution of linear speed	0.01 m/s
Resolution of angular speed	$\frac{\pi}{36}$ rad/s

4.3 Simulation analysis

In this section, five simulations in different scenarios are carried out to prove the effectiveness of the improved DWA algorithm, and the performance is compared with the original one. These simulations are implemented in smaller environments which are similar to the ones in Sect. 4.2 to show the effect of the improved evaluation functions, and in larger environments with discrete or spiral obstacle distributions to show the overall performance of the proposed algorithm.

In order to make the simulation results more practical, the kinematic model of the robot is set the same as the real robot used in the hardware experiments. The resolution of speed balances the navigation performance and computation cost. Considering the real radius of the robot, the indicator of reaching the target is the distance between the robot and target being less than $0.05m$. The speed and orientation of the robot are set to zero at the beginning. The robot parameters are shown in Table 1.

The parameters in the proposed algorithm are also set to ensure the comparability, as are shown in Table 2. Weights of the namesake evaluation subfunctions (w_1, w_2 and w_3) are the same. Some parameters in these subfunctions (travelled distance in $heading'(v, \omega)$ and $obsdist'(v, \omega)$) and weights of the added evaluation subfunctions (w_4 and w_5) are unique to the improved DWA. Two travelled distances are chosen according to the analysis in Sects. 4.2.1 and 4.2.2. Safety is considered as the primary standard of robot motion, so the weight of the $obsdist(v, \omega)$ (w_2) is twice as much as other evaluation functions, and weights of other evaluation functions are the same. All the simulations in this subsection share the same parameters.

4.3.1 Simulation scenario 1

The first simulation scenario is carried out in the environment similar to Fig. 3, which is to show the effect of the modified subfunction $obsdist'(v, \omega)$. The parameters of the map are shown in Table 3. Figure 10a, b respectively shows the trajectory of the original and improved DWA algorithm.

Table 2 Parameters of DWA

Parameter name	Parameter value
w_1	1
w_2	2
w_3	1
w_4	1
w_5	1
Travelled distance in $heading(v, \omega)$	0.5 m
Travelled distance in $obsdist(v, \omega)$	0.8 m

Table 3 Parameters of simulation scenario 1

Parameter name	Parameter value
Map size	9 m × 6 m
Starting position	(1 m, 1.5 m)
Target position	(8 m, 5.5 m)
Initial orientation	0 rad

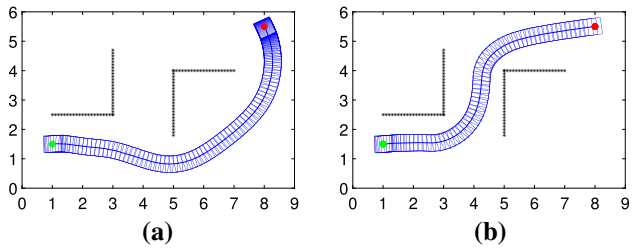


Fig. 10 Trajectory obtained by the **a** original DWA, **b** improved DWA in simulation scenario 1

Obviously, the improved trajectory is better since the robot moves through the obstacles and directly to the target. If there are more obstacles under the target, the original trajectory may become even more sub-optimal. The main reason for this is that the modified subfunction $obsdist'(v, \omega)$ changes the way of discarding the trajectory, so that trajectories through the obstacles can be remained.

4.3.2 Simulation scenario 2

The second simulation scenario is carried out in the environment similar to Fig. 4, which is to show the effect of the modified subfunction $velocity'(v, \omega)$. The parameters of the map are shown in Table 4. Figure 11a, b respectively shows the trajectory of the original and improved DWA algorithm. One can find that the improved trajectory go through the obstacles with a shorter path and a faster speed. It is mainly because of the modified subfunction $obsdist'(v, \omega)$, and the modified subfunction $velocity'(v, \omega)$ which enables the robot to consider the linear and angular speed together. Meanwhile, dangerous trajectories have been discarded by subfunction $obsdist'(v, \omega)$ so the safety is secured.

Table 4 Parameters of simulation scenario 2

Parameter name	Parameter value
Map size	8 m × 5 m
Starting position	(2 m, 1.5 m)
Target position	(7.5 m, 1 m)
Initial orientation	$\pi/2$ rad

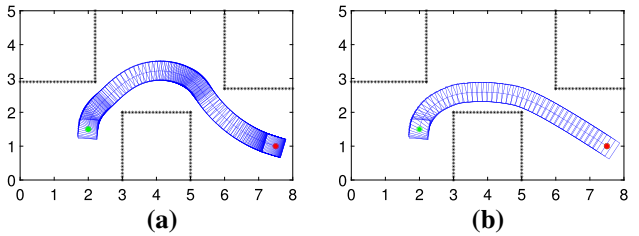


Fig. 11 Trajectory obtained by the **a** original DWA, **b** improved DWA in simulation scenario 2

Table 5 Parameters of simulation scenario 3

Parameter name	Parameter value
Map size	4 m × 6 m
Starting position	(1 m, 1m)
Target position	(3.5 m, 5.5 m)
Initial orientation	$\pi/2$ rad

4.3.3 Simulation scenario 3

The third simulation scenario is carried out in the environment similar to Fig 6, which is to show the effect of the modified subfunction $goaldist(v, \omega)$. The parameters of the map are shown in Table 5. Figure 12a, b respectively shows the trajectory of the original and improved DWA algorithm. It is obvious that the improved trajectory reaches the target with a shorter length while ensures the safety margin. The main reason is that the added subfunction $goaldist(v, \omega)$ enhances the motion trend towards the target when there are obstacles around it, while the safety is secured by subfunction $obsdist'(v, \omega)$.

4.3.4 Simulation scenario 4

The forth simulation scenario is carried out in the environment with discrete obstacles generated randomly at integer coordinates within certain ranges, which is to show the overall performance of the proposed algorithm. The parameters of the map are shown in Table 6.

Figure 13a, b respectively shows the trajectory of the original and improved DWA algorithm. Obviously, the improved trajectory is smoother. Figure 13c, d respectively shows the linear speed curve of the original algorithm and the improved

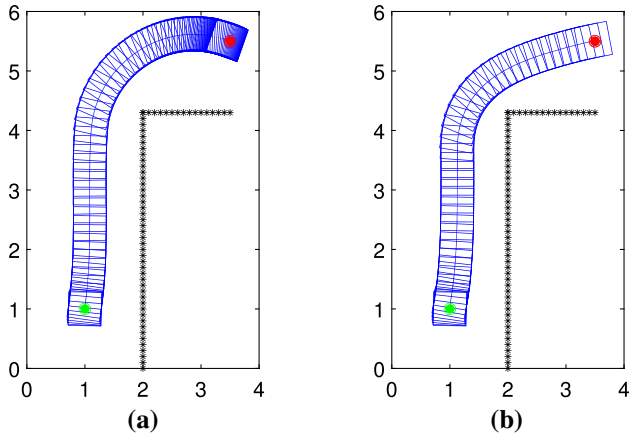


Fig. 12 Trajectory obtained by the **a** original DWA, **b** improved DWA in simulation scenario 3

algorithm. The improved DWA takes 19.65 s to reach the target while the original one takes 33.95 s, which proves that the improved algorithm significantly shortens the moving time and improves the traffic efficiency. According to Fig. 13c, it can be found that the original algorithm has lower traffic efficiency in the position near the obstacles or corners (around 9 s, 21 s and 27 s) and target (around 33 s).

Three reasons can account for the low traffic efficiency of the original DWA. Firstly, the original $obsdist(v, \omega)$ function calculates the minimum distance from the obstacle by the whole trajectory, which causes the predicted trajectory with larger linear speed to be more likely to collide with obstacle. These trajectories will get low score or be discarded directly when the robot is near the obstacle. Secondly, the trajectories with larger linear speed and slower angular speed will get lower score using the original $velocity(v, \omega)$ function, which reduces the traffic efficiency of the robot when moving slowly and circuitously. Thirdly, the original $heading(v, \omega)$ function calculates the orientation angle by the final position of the trajectory, which causes the trajectory with larger linear speed to get lower score due to its end overshooting the target when the robot is near the target. Also, it delays the robot reaching to the target which can be found in the close-up window of Fig. 13a. In contrast, the improved DWA avoids these three problems and significantly improves the traffic efficiency by enhancing the evaluation function.

Table 6 Parameters of simulation scenario 4

Parameter name	Parameter value
Map size	14 m × 14 m
Starting position	(1 m, 1 m)
Target position	(13 m, 13 m)
Initial orientation	0 rad

4.3.5 Simulation scenario 5

The fifth simulation scenario is carried out in the environment with spiral obstacle located at the center of the map. This scenario is mainly to prove the effect of function $oscillation(v, \omega)$. The parameters of the map are shown in Table 7

Figure 14a, b respectively shows the trajectories of the original and improved DWA algorithm. Note that the robot

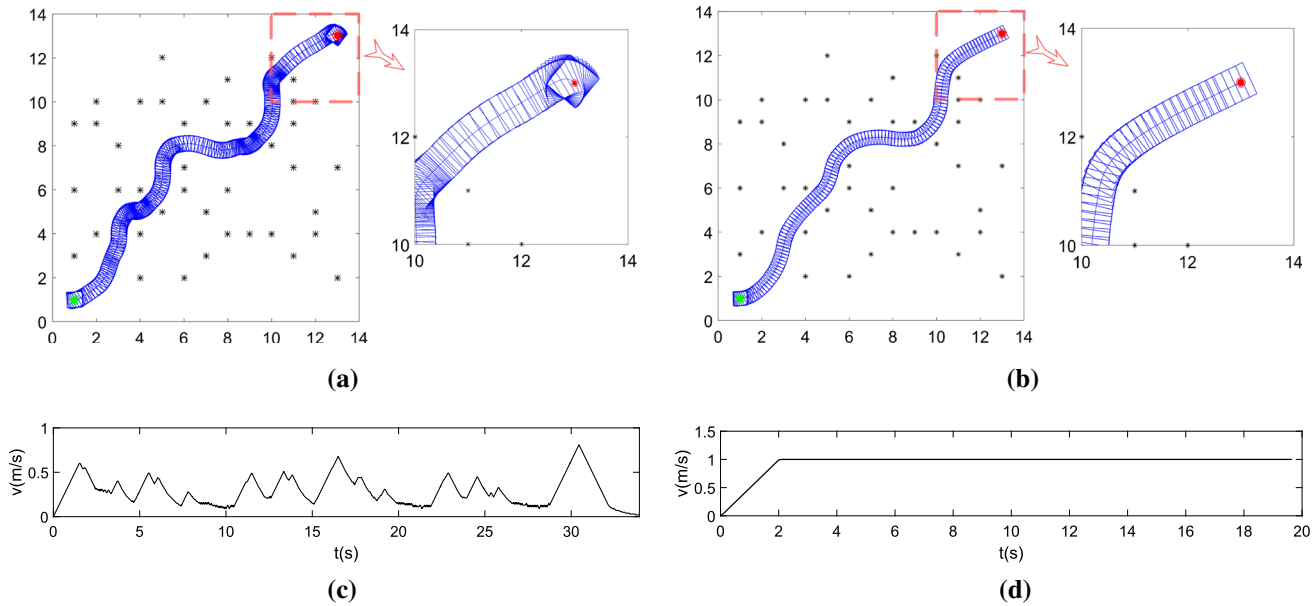


Fig. 13 Results of in simulation scenario 4. **a** Trajectory obtained by the original DWA and its close-up window, **b** Trajectory obtained by the improved DWA and its close-up window, **c** Linear speed obtained by the original DWA, **d** Linear speed obtained by the improved DWA

Table 7 Parameters of simulation scenario 5

Parameter name	Parameter value
Map size	11 m × 11 m
Starting position	(1.5 m, 1.5 m)
Target position	(10 m, 10 m)
Initial orientation	0 rad

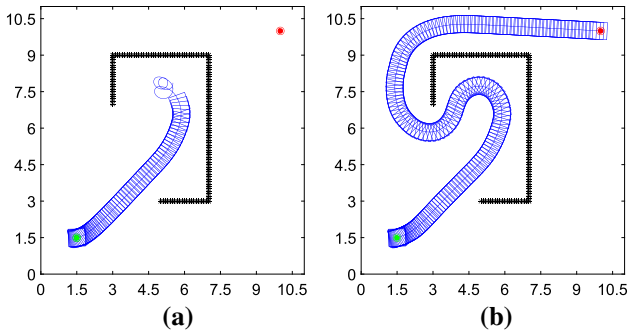


Fig. 14 Trajectory obtained by the **a** original DWA, **b** improved DWA in simulation scenario 5

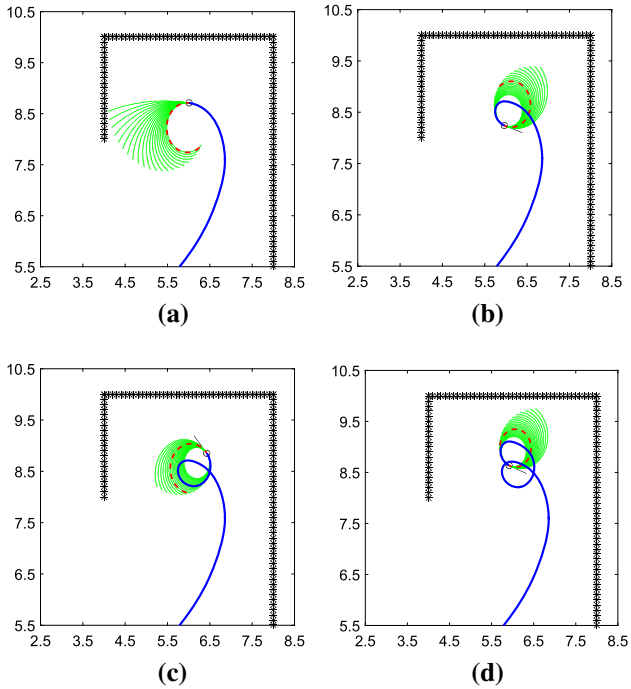


Fig. 15 Snapshots of the position in the trajectory obtained by the original DWA in simulation scenario 5. **a**–**d** Respectively shows the robot position in chronological order

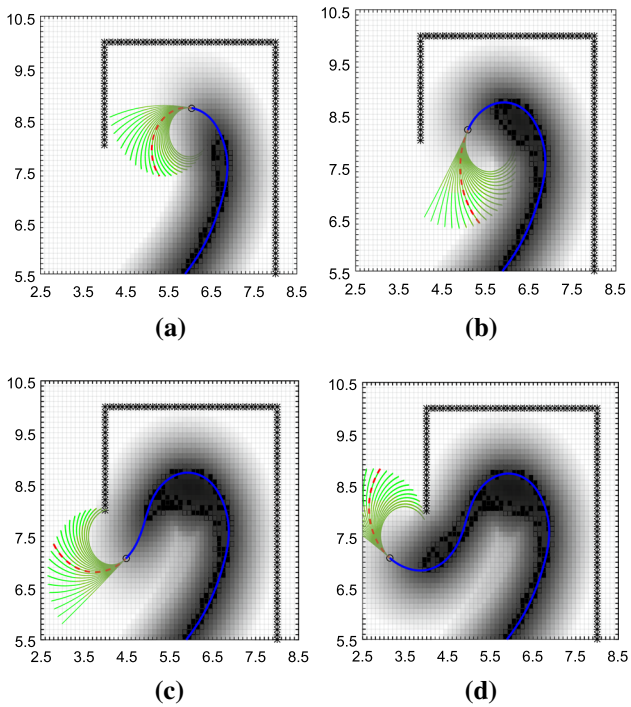


Fig. 16 Snapshots of the position in the trajectory obtained by the improved DWA in simulation scenario 5. **a**–**d** Respectively shows the robot position in chronological order

does not know the global distribution of obstacles, so it cannot go around the obstacle area directly. It is obviously that the improved trajectory escapes from the spiral obstacle and successfully reaches the target, while the original trajectory falls into local minima and keeps turning around. Note that the function $goaldist(v, \omega)$ is designed to work only when the minimum distance of all the predicted trajectories to target is less than 2 m, so it is function $oscillation(v, \omega)$ that causes the different performance of the two trajectories.

Comparing Fig. 14a, b, it can be found that the robot moves straight (see the initial section of the trajectory) when there is no obstacle around, and the original or improved DWA has the same effect on the trajectory generation. The snapshots of the positions when the two trajectories differ from each other are shown in Figs. 15 and 16. The green arcs in front of the robot are all the predicted trajectories in the sampling window, and the trajectory marked with dotted red line is the highest trajectory to be adopted at the current time. In Fig. 16, the cell group H to record the historical trajectory is shown with the similar method in Fig. 8. Observing Figs. 15a and 16a, at this time, the historical trajectory and all predicted trajectories of the robot are basically the same. Owing to function $oscillation(v, \omega)$, the improved trajectory avoids the predicted trajectories that are heavily biased towards the historical trajectory. In contrast, the original trajectory "forgets" the travelled position and always enters the spiral obstacle. Observing the subfigures b, c and d of Figs. 15 and 16, the improved trajectory escapes from the spiral obstacle and avoids the travelled position at all times, while the original trajectory cannot escape.

From the comparison of the two methods in five simulations, we can find that the improved DWA shortens the trajectory length, improves the capability and efficiency of reaching the target in different environments with complex obstacles.

5 DWA parameter adaptive tunning algorithm based on reinforcement learning

The weight terms of each evaluation subfunction were considered difficult to choose (Kiss and Tevesz 2012). In this section, we first analyze the parameters needed to be dynamically adjusted in DWA, and propose an adaptive tunning algorithm based on Q-learning to adjust them, finally conduct the simulation to verify the proposed method.

5.1 Parameter analysis in DWA

Because of the complex real working environment, the demand for each evaluation function is time-varying. For example, when the robot approaches the obstacle, it should turn or decelerate at once, so the effect of function $obsdist(v, \omega)$ needs to increase while other functions decrease to prevent collisions. When the robot approaches the target, it can arrive quickly in any direction regardless of the approaching angle, so the effect of function $heading(v, \omega)$ can decrease and function $goaldist(v, \omega)$ increase. When the current direction deviates from the target severely, it is possible that the robot avoids obstacles overly or goes around in circle, so the effect of function $heading(v, \omega)$ and $oscillation(v, \omega)$ should increase. It is hard to consider all situations only by designing functions, so the weights of each function need to be self-adaptive with the change of the surrounding environment and the robot state.

In addition, all evaluation functions will use the time period T. Considering the situation in Fig. 17a, there is a spiral obstacle near the target, and the robot is still away from the obstacle in a certain distance. If T is shorter, all the sampling trajectories will be shorter. In terms of the existing five evaluation functions, trajectory A will get the highest score. However, after the robot moves along the trajectory for a while, all trajectories will collide with obstacles which makes the robots stop and fail to navigate. In contrast, if T is longer, the trajectory will be longer, so the further obstacle can be perceived. Trajectory B may get the highest score, so that the robot can adjust its heading in advance. However, considering the situation in Fig. 17b, there are many small obstacles around the robot. A longer T results in a longer predicted trajectory. Most trajectories will collide with obstacles, which reduces the discrimination of the function $obsdist(v, \omega)$ and discards some feasible direction like trajectory A. Meanwhile some trajectories may be toward the target but too close to obstacles, which leads to misjudgment of function $goaldist(v, \omega)$ like trajectory B. These two kinds of situations are both not ideal. Therefore, the time period T should also be self-adaptive.

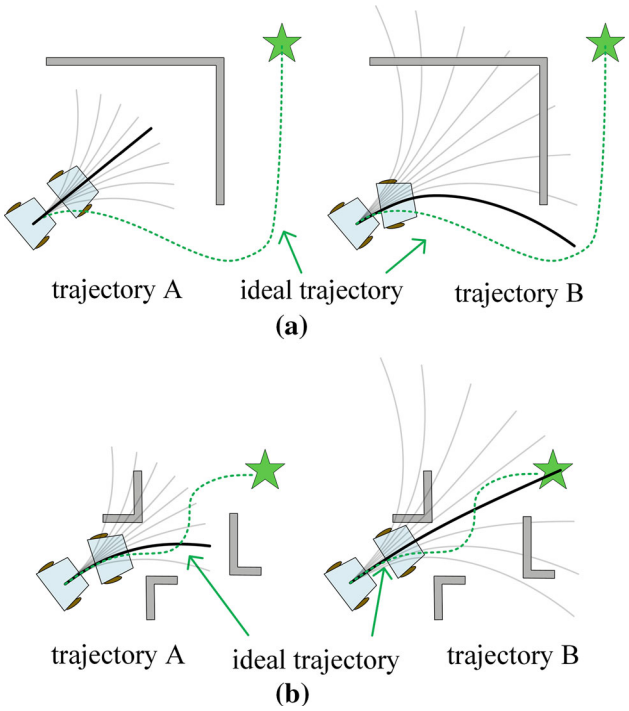


Fig. 17 Trajectory analysis for time period T

The time period T essentially determines the distance traveled from the original position. During the considerable time period, the trajectory can no longer be seen as a straight line. Once the travelled distance d is determined, we can obtain the time period T by the geometric relationship in Fig. 18 and Eq. (10). Concretely, if the maximum distance between the trajectory and current position is larger than d, T is obtained by the trajectory whose end position is d meters away from the original position. Otherwise, T is set where the trajectory moves half a cycle.

$$T = \begin{cases} \frac{2\arcsin\left(\frac{d}{2r}\right)}{\omega} & 2r > d \\ \frac{\pi}{\omega} & 2r \leq d \end{cases} \tag{10}$$

where $r = \frac{v}{\omega}$ is the radius of the predicted trajectory; v and ω are the linear and angular speed of the robot.

According to the analysis above, six parameters need to be dynamically adjusted during the navigation process, which are: the weights of each evaluation function w_1–w_5 and the travelled distance d. The algorithm for parameter adaptation is illustrated in Sect. 5.2.

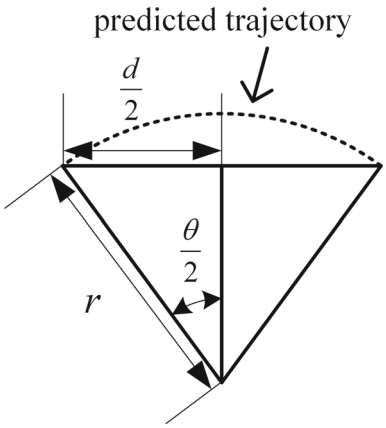


Fig. 18 Calculation method of time period T

5.2 Parameter adaptive algorithm in DWA based on Q-learning

5.2.1 Application method of Q-learning in DWA

The tabular Q-learning establishes its agent as a $m \times n$ matrix Q where m is the state dimension and n is the action dimension. The agent executes an action A_j at state S_i, the reward R from the environment will be superimposed on the position (i, j) of the matrix Q. The action is selected according to the current state, matrix Q and action selection strategy. The updating rule of matrix Q is written by Eq. (11).

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha[R(s,a) + \gamma Q(\tilde{s},\tilde{a})] \quad (11)$$

where $\alpha \in (0, 1)$ is the learning rate. The larger of α means the faster learn and converge rate, but may lead to overfitting; $\gamma \in (0, 1)$ is the discount factor. The larger of γ means the long-term interests is paid more attention while the smaller means the current interests; $R(s, a)$ is the reward from the environment obtained by the current state and action; $Q(\tilde{s}, \tilde{a})$ is the maximum value of Q among all the actions that the next state corresponds to.

With the fundamental framework of Q-learning, customized algorithm design is needed for the application in DWA, which includes: definition of the state space, the action space and the reward function. The design of the Q-learning based parameter learning algorithm is presented in subsequent subsections.

5.2.2 Definition of the state space

In the process of robot motion, the state information depends on the heading and speed of the robot, the surrounding obstacle and target. In order to apply the agent to different environments, the state cannot represent the absolute coordinates and orientation on a specific map. Therefore, some general and relative features need to be extracted from the complex environment to constitute the state space, and each state corresponds to a particular situation. The robot's state is uniquely described at any time in a given map. Also, the state space should not be too large, otherwise it will lead to "the curse of dimensionality".

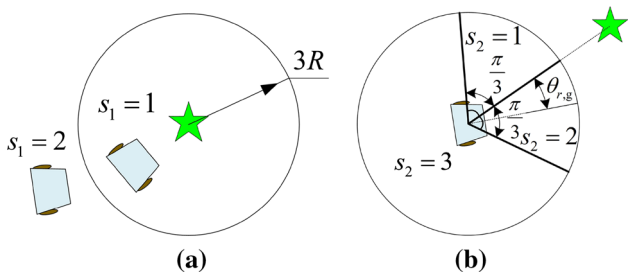


Fig. 19 Dimension sketch of **a** s_1, **b** s_2

The score of the trajectory in the evaluation function determines the robot speed at the next time step, so the definition of state space should be closely related to the evaluation function. Form the previous analysis for the evaluation function, we can find that some particular features are frequently considered as the scoring basis, such as: the distance between the robot and target, the speed and orientation of the robot and the surrounding obstacle, which means that these features play a pivotal role in the operation of DWA. Therefore, we use these features to constitute the dimensions of state space S as Eq. (12).

$$S = [s_1\ s_2\ s_3\ s_4]^T \quad (12)$$

where s_1–s_4 respectively are the various dimensions in S to be explained in detail below.

(1) Dimension s_1 is to represent whether the robot is very close to the target whose value is designed as Eq. (13) and dimension sketch is shown in Fig. 19a.

$$s_1 = \begin{cases} 1 & d < 3R \\ 2 & d \geq 3R \end{cases} \quad (13)$$

where d is the distance between robot and target; R is the safety radius of the robot.

(2) Dimension s_2 is used to represent whether the robot is moving towards the target and the approximate deviation. The value of dimension s_2 is designed as Eq. (14) and dimension sketch is shown in Fig. 19b.

$$s_2 = \begin{cases} 1 & \theta_{r,g} \in \left[0, \frac{\pi}{3}\right) \\ 2 & \theta_{r,g} \in \left[-\frac{\pi}{3}, 0\right) \\ 3 & \theta_{r,g} \in else \end{cases} \quad (14)$$

where $\theta_{r,g}$ is the angle between the orientation of the robot and the line pointing from the robot position to the target.

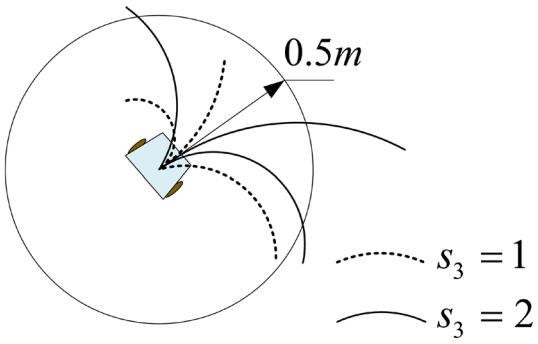


Fig. 20 Dimension sketch of s_3

(3) Dimension s_3 is to represent whether the position where the robot moves with its current speed for $1s$ is too far away from its current position, which also reflects the coverage of all the sampling trajectories. The travelled distance d between the two positions is calculated by Eq. (15).

$$d = \begin{cases} \frac{2v}{\omega} & \omega > \pi \\ \frac{2v}{\omega} \sin\left(\frac{\omega}{2}\right) & \omega \leq \pi \end{cases} \tag{15}$$

where v and ω are the linear speed and angular speed of the robot.

The value of s_3 is designed as Eq. (16) and dimension sketch is shown in Fig. 20.

$$s_3 = \begin{cases} 1 & d \leq 0.5\,\text{m} \\ 2 & d > 0.5\,\text{m} \end{cases} \tag{16}$$

(4) Dimension s_4 is to represent whether there are obstacles in a certain range around the robot, and the approximate relationship between the obstacle and the robot motion direction. Dimension s_4 describes the approximate distribution of the surrounding obstacles with a single angle, which is called dominant orientation of obstacles. Laser sensors are widely used in the field of robotic perception, so the method of laser scanning can be applied to detect the obstacles around. We divide the surrounding of the robot into several angles evenly, and measure the distance of the nearest obstacle on each angle. If the distance exceeds a given range $5R$, it will be set to zero. In this way, the angle of the nearest obstacle θ_{on} can be found which reflects the distribution of the obstacle around to a certain extent, but it is not an ideal feature as it abandons most of the information. Here we present a concept called the average orientation of obstacles θ_{oa}. Generally speaking, the smaller the distance is, the more dangerous the obstacle is, and the larger the proportion in the average orientation should be. Therefore, the distance reciprocal of each angle can be used as the weight for obtaining the average angle, and the average orientation of obstacles is weighted sum of the angles, which is calculated by Eq. (17).

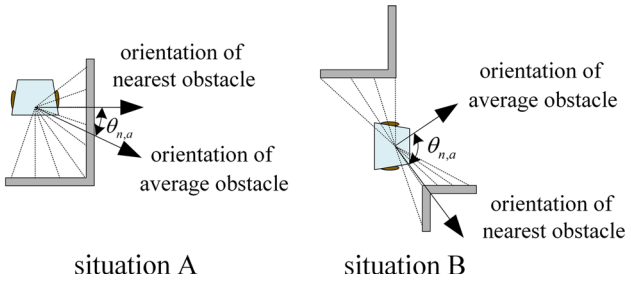


Fig. 21 Analysis for the dominant orientation of obstacles

$$\theta_{oa} = \frac{\sum_{i=1}^{n} \frac{\theta_i}{d_i}}{\sum_{i=1}^{n} \frac{1}{d_i}} \tag{17}$$

where n is the angle resolution; θ_i is the ith angle; d_i is the distance of the nearest obstacle on the ith angle.

However, the average position is not always accurate. Considering two situations in Fig. 21, $\theta_{n,a}$ is the angle between the orientation of the nearest and average obstacles. In situation A, $\theta_{n,a}$ is smaller and the average orientation can reflect the approximate distribution of the surrounding obstacle more comprehensively. Conversely, obstacles in situation B are located on both sides of the robot and there are no obstacles in the direction of the average orientation, which makes the average orientation fail to reflect the obstacle distribution. The reason for this failure is that the obstacles are scattered or discontinuous, so the average orientation may be the average of several angles with short distance, which is probably the discontinuity. At this time, $\theta_{n,a}$ is generally larger. Considering that θ_{on} also has certain reference value, the dominant orientation of obstacles θ_o is set as Eq. (18).

$$\theta_o = \begin{cases} \theta_{oa} & \theta_{n,a} < \frac{\pi}{4} \\ \theta_{on} & \theta_{n,a} \geq \frac{\pi}{4} \end{cases} \tag{18}$$

Now, the value of s_4 can be designed as Eq. (19) and the dimension sketch is shown in Fig. 22.

$$s_4 = \begin{cases} 1 & \theta_{r,o} \in \left[0, \frac{\pi}{3}\right) \\ 2 & \theta_{r,o} \in \left[-\frac{\pi}{3}, 0\right) \\ 3 & \theta_{r,o} \in else \\ 4 & \text{no obstacle around} \end{cases} \tag{19}$$

where $\theta_{r,o}$ is the angle between the orientation of the robot and the dominant orientation of surrounding obstacles.

So far, four kinds of features have been extracted. Single feature can describe some explicit and measurable information, while the relationship between each feature can deduce some hidden information, such as: the relationship between target and obstacles, the extent needed of obstacle avoidance,

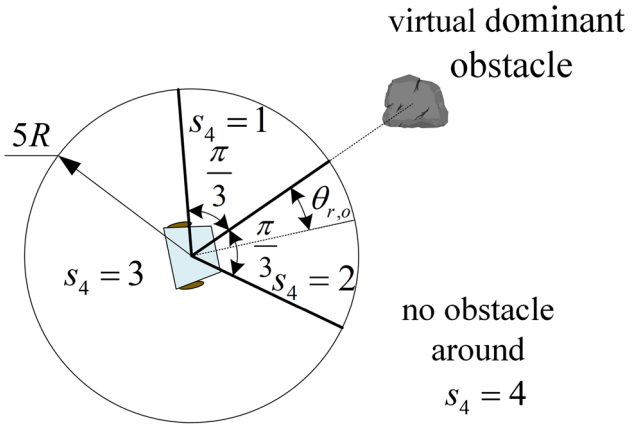


Fig. 22 Dimension sketch of s_4

the way the robot moves toward target, etc. Both the measurable and hidden information are conducive to the parameter selection in DWA. According to these four features, the state space is divided into 48 substates, which constitutes the state dimension of the matrix Q.

5.2.3 Definition of the action dimensions

The trajectory adopted by the robot has the highest total score in the sampling space, and the total score is the weighted sum of each evaluation function. According to the previous analysis, the time period T is also an important parameter affecting the score, which is determined by the travelled distance d and the speed of the robot. The real-time speed is known, so d can determine T. Therefore, each dimension in the action space includes six parameters: the weights of each evaluation function w_1–w_5 and the travelled distance d. Since each evaluation function has been normalized before participating in the general evaluation, their importance can be uniquely represented by the weights. The base value of each weight is set to 1, and the weight of the evaluation function requiring more consideration is set 2 or 3. For example, if there are closer obstacles on the motion direction, the weight of the function $obsdist(v, \omega)$ should increase. The travelled distance is divided into three grades, and the range of value for each parameter is designed in Eq. (20).

$$\begin{cases} w_1, w_2, w_3, w_4, w_5 \in \{1, 2, 3\} \\ d \in \{1, 1.5, 2\} \end{cases} \tag{20}$$

To reduce the dimension of the action space, we do not use all the combinations of the possible parameter values, but select a representative part of them, namely the five weights are chosen from $\{1, 2\}$ or $\{1, 3\}$. After removing the equal proportion weights and considering the three-grade travelled distance, 93 combinations from the weights set $\{1, 2\}$ are obtained by $(1 + C_5^1 + C_5^2 + C_5^3 + C_5^4) \times 3$, and 90 combinations are obtained from the weights set $\{1, 3\}$ by $(C_5^1 + C_5^2 + C_5^3 + C_5^4) \times 3$. In summary, 183 combinations are selected to constitute the action dimension of the matrix Q.

5.2.4 Definition of the reward function

The reward function is to calculate the reward from the environment of a specific action on a given state, which is an indicator of how good or bad the action is. The definition of reward function should consider different aspects and rely on the actual movement whose goal is to reach the target quickly and avoid obstacles. We refer to some Atari game environments which are commonly applied in RL research to motivate the reward values, and design some minor rewards for each change of the state to avoid the reward being too sparse. Concretely, the principle of the reward function is as follows. If the robot reaches the target, it gets a larger positive reward ($+5000$). If the robot collides with obstacles, it gets a larger negative reward (-200). If the robot is closer to the target ($+10$) or farther from the obstacle ($+5$), it gets a certain positive reward, otherwise, it gets a certain negative reward (-10 or -5). In addition, the robot gets smaller negative reward (-2) at each time the state changes before reaching the target. Generally, the robot will experience more states during the path which costs more time, so we hope this reward can help the robot experience few states to obtain a faster path.

Concretely, when the robot executes an action A on the previous state S_{pre} and enters the current state S_{curr} ($S_{pre} \neq S_{curr}$), the reward R of A from the environment is calculated with Algorithm 1.

5.2.5 Training process of the proposed method

The application of Q-learning in DWA is different from that in other field such as labyrinth, which is mainly reflected in two aspects. First, in the real environment, the real state of the robot changes continuously while the 48 states consisting of the features are discrete, so not each action will change the state and get the reward. Note that the previous action should be maintained if the state is not changed, so that the investigation time of an action can be guaranteed. The time of updating the matrix Q is when the state changes, and the position of updating in the matrix Q is determined by the state before this change and the action causing this change. Second, because of the large dimension of the matrix Q, when selecting a new action, especially in the initial stage of training, there will be many zeros in the matrix Q, which causes some states to have many actions with the same maximum Q value. At this time, selecting an action randomly is not appropriate, since it makes the action change too frequently. The investigation time of an action is not long enough, and

Algorithm 1: Algorithm for calculating the reward R from the environment

Input:
Nearest distance to the obstacle at S_{pre}: $d_{o,pre}$;
Nearest distance to the obstacle at S_{curr}: $d_{o,curr}$;
Distance to the target at S_{pre}: $d_{t,pre}$;
Distance to the target at S_{curr}: $d_{t,curr}$.
Output:
Reward: R.

```
R = 0;
if S_curr collides with obstacles then
    R = R − 200;
    return R;
if S_curr reaches the target then
    R = R + 5000;
    return R;
if d_o,curr > d_o,pre then
    R = R + 5;
else
    R = R − 5;
if d_t,curr > d_t,pre then
    R = R − 10;
else
    R = R + 10;
R = R − 2;
return R;
```

Algorithm 2: Algorithm for parameter adaptation in the improved DWA based on Q-learning

Input:
Information of the map and robot, e.g. stating point, target, kinematic model of the robot, etc;
Parameter in this algorithm, e.g. maximum training time N, learning rate α, greed factor γ, etc.
Output:
Trained agent: Q.

```
1  i = 1;
2  Q = zero matrix of 48 × 183;
3  while i ≤ N do
4      Initialize robot pose;
5      //an episode where S_pre and S_curr are the previous and current
       state, A_pre and A_curr are the previous and current action
6      j = 1;
7      while True do
8          S_curr = state perceived;
9          if j = 1 then
10             A_curr = [1 1 1 1 1 1.5]^T;
11         else
12             if S_pre = S_curr then
13                 A_curr = A_pre;
14             else
15                 Calculate R by Algorithm 1;
16                 Update Q by Eq. (11);
17                 if S_curr meet the end indicator of an episode then
18                     break
19                 else
20                     if strategy is exploitation then
21                         if A_pre ∈ A_max Q then
22                             A_curr = A_pre;
23                         else
24                             A_curr = random action in A_max Q;
25                     else
26                         A_curr = random action;
27         S_pre = S_curr;
28         A_pre = A_curr;
29         Robot moves with A_curr for one time step;
30         j = j + 1;
31     i = i + 1;
32 return Q;
```

finally, the learning results are highly random. It should be checked whether the action set with the maximum Q value $A_{\max Q}$ contains the previous action, and if does, the previous action should be executed to keep its continuity, otherwise, select one randomly. The end indicator of a navigation (say an episode) is the robot reaching the target, forced to stop or collision with obstacles.

Algorithm 2 illustrates the training process of the proposed method, where we assume that the initial robot pose does not meet the end indicator of an episode. Concretely, i defined in line **1** is the counter of the episode, and the training ends when i exceeds the maximum times N. j defined in line **6** is the counter of the time steps in one episode by which we can calculate the time required for a real robot to finish this episode. The weights chosen in line **10** are the default weights.

5.3 Simulation analysis

The simulations are carried out in this section to prove implement the training process and show the training results of the proposed method using MATLAB. To enhance the training effect and make the result adapt to different environments, we design the map shown in Fig. 23. In this figure, the distribution of obstacles is various and there are many feasible paths connecting the starting point and the target. Using such a complex map is to enrich the states perceived (row of the matrix Q) and the actions chosen (column of the matrix Q) during the training, which will help to train the agent Q fully and make it applicative to other maps. Note that although the distribution of the obstacles is determined, the robot is set to use the information of the obstacles within a certain range, which reflects the unknown characteristic of the global environment. This simulation contains three parts: the first part is the training process to obtain the trained agent, the second part is the test result to show the superiority of the trained agent compared with the fixed weight, and the third part is the comparisons with other algorithm in different environments.

Table 8 Parameters of the training process

Parameter name	Parameter value
Map size	16 m × 16 m
Starting position	(2.5 m, 2.5 m)
Target position	(14.5 m, 14.5 m)
Initial orientation	$\pi/4$ rad
α	0.5
γ	0.5
ε in $\varepsilon - greedy$	0.02

5.3.1 Training process

The parameters of the training are set by convention. The initial speed of the robot is set to zero. Without loss of generality, the initial orientation of the robot is set to $\pi/4\ rad$ to make the trajectories be distributed in both sides of the map diagonal. The kinematic model of robot is the same as before. The parameters of the training process are shown in Table 8.

The training is implemented on an Intel Core i7-8750H 2.2 GHz CPU with 16 GB memory and takes about 6.3 h. After the training, 359 successful paths were obtained among the 5000 episodes, which are shown in Fig. 23a. It is clear that multiple paths have been explored by the robot. To embody the training effect more clearly, all the episodes are divided in to 50 groups orderly and each group contains 100 episodes, which are used for statistics. Figure 24 shows the number of arriving paths and the average time consuming of the arriving paths of each group respectively. With the agent Q continuously updates, the successful rate of arrival presents a rising tendency while the average time consumption presents a declining tendency, which shows that the proposed method conduces to the robot avoiding obstacles and finding a faster path. A trained agent is obtained after the training. Figure 23b shows the trajectory of the robot adopting this trained agent whose time consumption is $21.5s$. The robot successfully reaches the target with relatively less time-consuming path, which also reflects the effectiveness of the proposed met- hod.

5.3.2 Test results

To further verify the performance of the trained agent, it should be adopted in different environment. Note that the information that the robot can perceive includes its pose and speed, the relative position to the target and obstacles within a certain range. These information will vary with different initial states including positions and speeds. In addition, the agent does not record any information about the distribution of obstacles during the training. Therefore, the performance

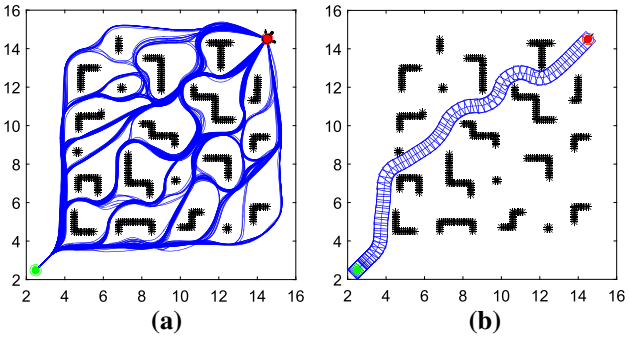


Fig. 23 **a** Arriving paths among training, **b** Trajectory adopting the trained agent Q

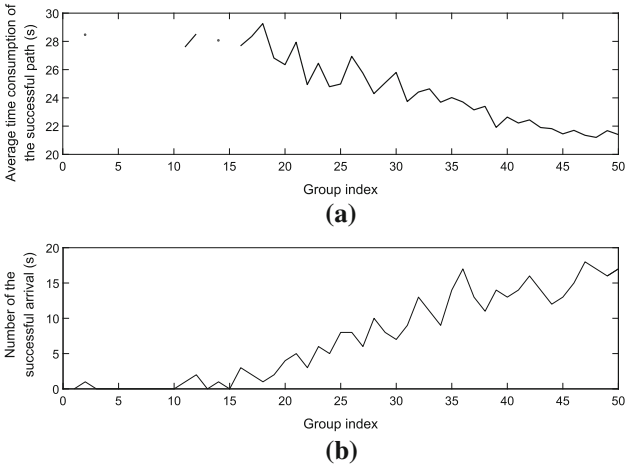


Fig. 24 The statistical results of each group. **a** Average time consumption of the successful path (s), **b** Number of successful arrival (s), where the discontinuity of the curve means no arriving paths in the corresponding group

of the trained agent can be thoroughly evaluated with adequate number of different initial states in the same map.

We randomly set 700 initial states, specifically: the starting position is randomly distributed at the L-shaped area of the lower left portion of the map, which is highlighted with green rectangle in Fig. 25; starting linear speed is randomly chosen in $[0, 0.8v_{\max}]$; starting angular speed is 0; starting orientation is randomly chosen in $[0, 2\pi)$.

Figure 25 shows the navigation results of random initial state adopting the trained agent Q and three different fixed parameter sets respectively. The parameter set I $[1\ 1\ 1\ 1\ 1\ 1.5]^T$ contains the default parameters. This set has the same weights of each evaluation subfunction which means each of them shares the same importance in different states, and the travelled distance $1.5m$ which is the medium grade in Eq. (20). The parameter set II $[1\ 2\ 1\ 1\ 1\ 2]^T$ is the same as the simulation in Sect. 4.3, which favors the subfunction of obstacle avoidance. The parameter set III $[2\ 1\ 1\ 1\ 1\ 1.5]^T$ favors the subfunction of navigation to the target. In Fig. 25, the length and orientation of line segments

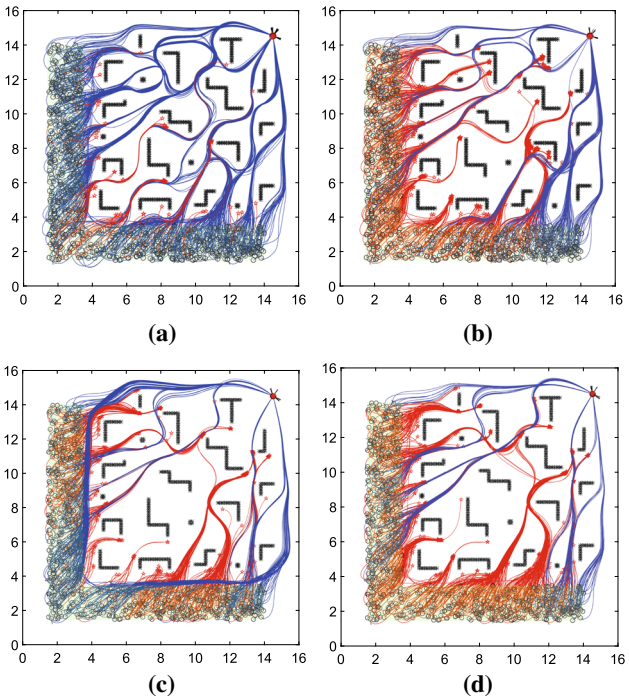


Fig. 25 Navigation result of random initial state adopting **a** trained agent Q, **b** parameter set I, **c** Parameter set II, **d** Parameter set III

at the initial position of each path means the initial linear speed and orientation of the robot. The arrival path is blue while the failure path is red, and the failure points where robot is forced to stop or collides with obstacles are marked with stars.

One can find from Fig. 25a that the trained agent enables the robot to move flexibly between obstacles and reach the target in most case. In contrast, the arriving paths of Fig. 25b mostly reach the target from the edge or outside of the obstacle area, and many paths with the starting position that need the path to go through the middle of the obstacle area fail. In Fig. 25c, the trend of bypassing the whole obstacle area is even more obvious as the subfunction of obstacle avoidance has the largest weight in parameter set II. The weight of subfunction of navigation to the target is set higher than other weights in parameter set III. As a result, the willing of going through the obstacle is strong but the least paths successfully reach the target in 25d. One can find from these subfigures that the fixed parameter set will make the robot always show a specific preference of navigation but fail to consider the different states to change this preference, which leads to low success rate. The comparison results are shown in Table 9. Under the same randomly-chosen initial states, the success rate of the trained agent is significantly higher than that of the other fixed parameter sets.

This test further proves that the proposed method with trained agent Q can find the best action at different states by learning, and have significant effects without long-term training.

5.3.3 Comparisons with other algorithms

In addition, we test the proposed method on different environments by comparison with other local path planning algorithms. The application scenario in this paper does not include prior knowledge of obstacle in the global map, so the comparison algorithm can only use the obstacle within the detection range for real-time planning.

Rolling RRT algorithm is an improved version of RRT which uses the rolling detection window instead of the accurate global map for planning. This planning pattern is very similar to our proposed method. Figure 26a is the result of rolling RRT which is also smoothed by the Bezier curve (Li et al. 2016). We build the same environment and test our method with the trained agent Q in this environment, which is shown in Fig. 26b. Obviously our path is smoother and shorter without any smoothing process.

Bug algorithm is a stress based algorithm which avoids the obstacle by rotating around it. In general, Bug algorithm does not include the global map such as Bug2 (Lumelsky and Stepanov 1987), VisBug (Lumelsky and Skewis 1990) or E-Bug (Lynda 2015), but some versions still consider it as K-Bug (Langer et al. 2007) or TangentBug (Ishay et al. 1998). Figure 27a is the result of a set of Bug algorithms (Lynda 2015). We build the same environment and test our method with the trained agent Q in this environment, which is shown in Fig. 27b. We can find that our trajectory is similar to the best path (E-Bug) of the Bug family while being smoother and safer.

Table 9 Comparison of navigation result with fixed parameters and trained agent

	Number of navigations	Number of arrival	Success rate (%)
Trained agent	700	636	90.86
Parameter set I	700	184	26.29
Parameter set II	700	266	38.00
Parameter set III	700	120	17.14

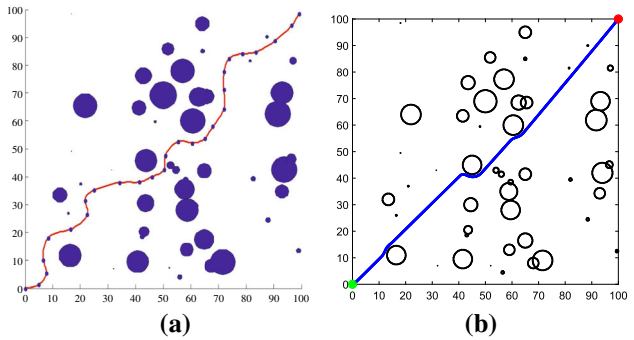


Fig. 26 Navigation result of **a** rolling RRT (Li et al. 2016), **b** our method with trained agent Q

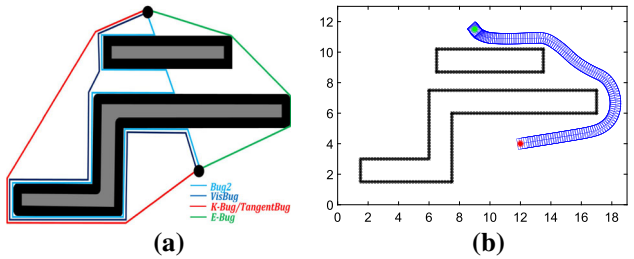


Fig. 27 Navigation result of **a** Bug family (Lynda 2015), **b** our method with trained agent Q

Moreover, comparison algorithms only obtain the path. In contrast, our paths are actually trajectories, which means that no path tracking algorithm is needed to follow the path. This is the advantage of DWA and our method preserves it.

The comparisons prove that our proposed method adapts to different environments and shows better effects than other algorithms. Another different environment is built in Sect. 6.2.1.

6 Hardware experiments

6.1 Experimental framework

The experiments are carried out on the XQ-4 Pro two-wheeled mobile robot with Intel Core i7-4500U 1.8 GHz CPU, 8 GB memory and Robot Operating System (ROS) based on Ubuntu 14.04, which is shown in Fig. 28. This robot is equipped with many sensors such as gyroscope, lidar, infrared, camera, etc. The robot's kinematic model is the same as the simulation in Sect. 5.3. We use a workstation to remotely login and operate the XQ-4 Pro through the wireless network, and the monitoring and control on the workstation is based on Rviz software.

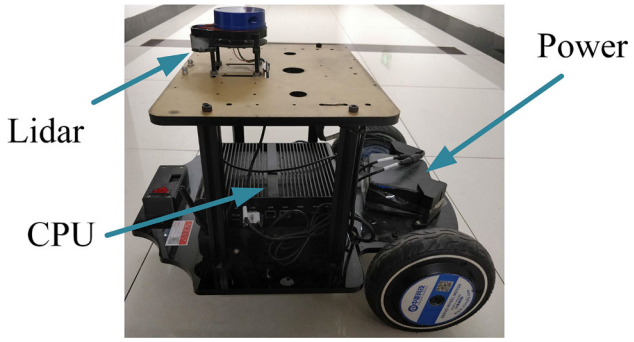


Fig. 28 The XQ-4 Pro robot

In ROS, the autonomous navigation tasks involve many nodes, which are divided into four categories by different functions: sensor node, localization node, path planning node (named *move_base*) and motor control node. The proposed algorithm in this paper are implemented in the *move_base* node through C++. The framework of autonomous navigation is shown in Fig. 29. The thin dashed box shows the process of original path planning, where the A* global path planner receives the information of the global map and target, and obtains a reference path which needs to be tracked by the original DWA local path planner. However, when the global map is less known or dynamic, a feasible reference path is hard to obtain. In contrast, our new process shown in the thick dashed box empowers the DWA algorithm with the capability of global navigation. As it can be seen in the thick dashed box, the improved DWA algorithm does not need the global map, but only the target. The trained agent with optimal weights obtained by the Algorithm 2 is deployed on the *move_base* node. Note that the parameters of the training in Sect. 5.3 are set the same as those in the experiment, so the trained agent form the simulation can be adopted in the real robot. The action with the maximum Q-value, which contains the optimal weighs of each evaluation function and the traveled distance, is selected according to the trained agent and the current state calculated by the perception. Finally,

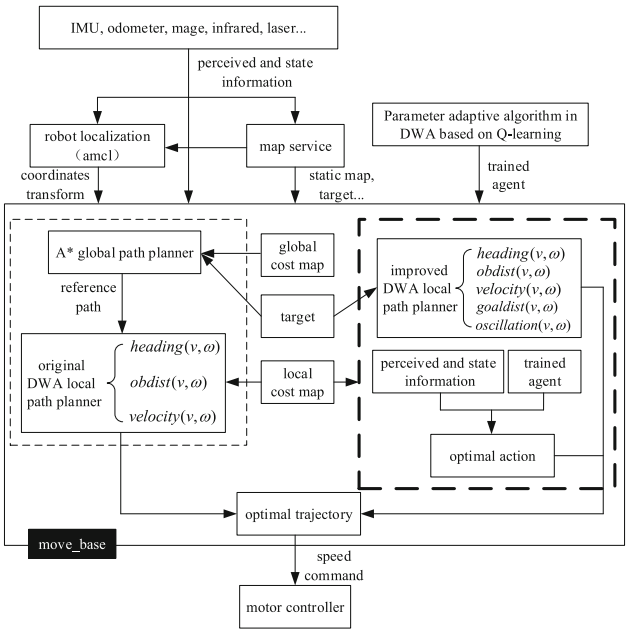


Fig. 29 Framework of autonomous navigation

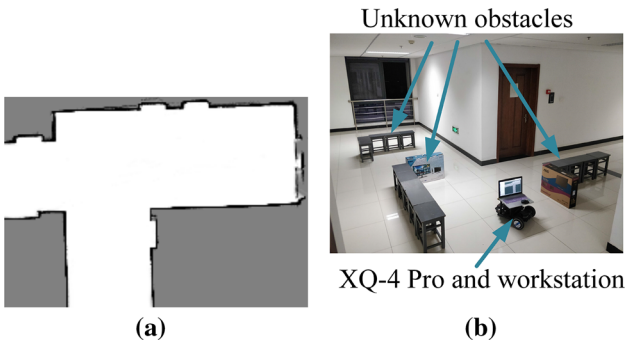


Fig. 30 **a** Original map of experiment scenario 1, **b** real experiment scenario 1

the best trajectory is obtained and the corresponding speed command is released to the motor controller.

In addition, due to the existence of unknown obstacles, the reference path obtained by the global path planner may be found unfeasible during the navigation. At this time, the original path planning will rerun the A* global path planner to get a new reference path considering the obstacles perceived. This process is time-consuming and inefficient, especially when there are many unknown obstacles. In contrast, our new process can navigate and deal with unknown obstacles without the reference path.

The supplementary materials of this paper include two videos named "dwa_static" and "$dwa\_dynamic$". These videos are the screen recordings of the workstation during the two experiments respectively. The agent Q deployed in the experiments is obtained by the training in Sect. 5.3.1, so the successful arrivals in the videos validate the effectiveness of the proposed method. The screenshots of the videos are shown and analyzed in Sect. 6.2.

6.2 Experimental results

The experiments are conducted in an unknown environment with static or dynamic obstacles to verify the performance of the proposed method. All the experimental environments are set up in the corridor of an office building. Note that we still operate the global path planner to show the difference between the reference path generated by the global path planner and the real trajectory obtained by our proposed algorithm. However, the reference path plays no role in the robot navigation.

6.2.1 Experiment scenario 1

The first experiment scenario is carried out in the environment with unknown static obstacles. To reflect the characteristic of "unknown", the original scenario contains no obstacles, and its 2D map built by simultaneous localization and mapping (SLAM) is shown in Fig. 30a. However, as can be seen in Fig. 30b, the actual environment for navigation is equipped with obstacles such as stools and boxes. The robot has no prior knowledge of the obstacles and can only perceive the obstacles during the navigation. The spatial distribution of these obstacles is set to significantly intercepts the trajectory of the robot towards the target.

Figure 31 shows the temporal sequence of the snapshots of robot navigation visualized in ROS Rviz and the corresponding real robot position in experiment 1. Concretely, Fig. 31a shows the starting position, the target and the reference path. We can find that the reference path only considers the original obstacle (the wall). Figure 31b–d show the movements of robot avoiding the equipped obstacles and Fig. 31e shows the real trajectory as the robot successfully reaches the target. It can be seen that the robot moves flexibly between the obstacles and the trajectory is quite different from the reference path, which proves that the proposed method can navigate the robot in an unknown environment.

6.2.2 Experiment scenario 2

The second experiment scenario is carried out in the environment with one moving pedestrian as the dynamic obstacle. Note that although the dynamic obstacles are not included in our training process, the original DWA is well improved for better navigation performance, and the trained agent is obtained by the complex environment containing various states defined in Sect. 5.2.2, so the proposed method can deal with the scenario with simple dynamic obstacles. The map of the original scenario built by SLAM is shown in Fig. 32a and the actual environment for navigation is shown in Fig. 32b. In the experiment, a pedestrian walks randomly in

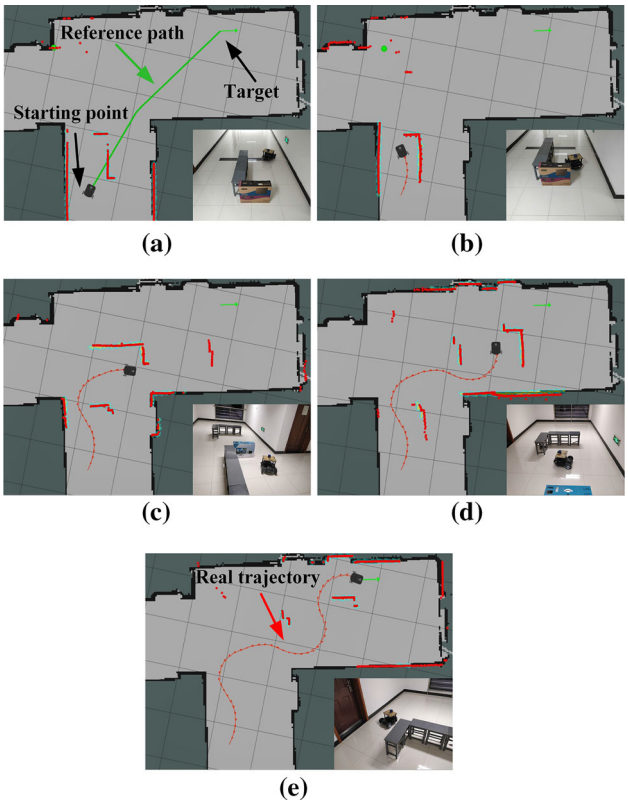


Fig. 31 The process of robot experiment 1. **a**–**e** Respectively shows the robot position in chronological order

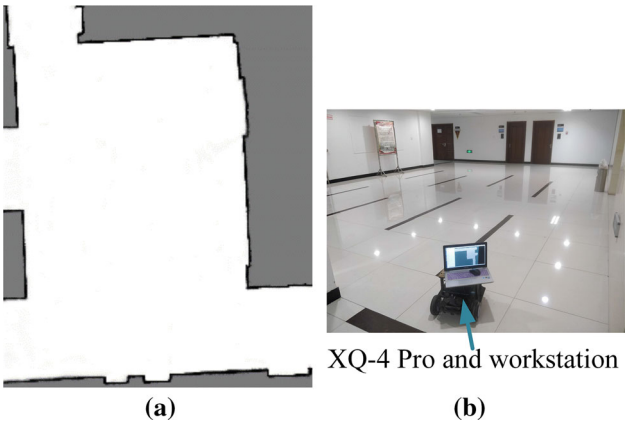


Fig. 32 **a** Original map of experiment scenario 2, **b** real experiment scenario 2

the scenario who significantly intercepts the trajectory of the robot towards the target twice.

Figure 33 shows the temporal sequence of the snapshots of the robot navigation visualized in ROS Rviz and the corresponding real robot position in experiment 2. Concretely, Fig. 33a shows the starting position, the target and the reference path. Since there is no obstacle in the original map, the reference path goes straight to the target. Figure 33b–g show the movements of robot avoiding the moving pedes-

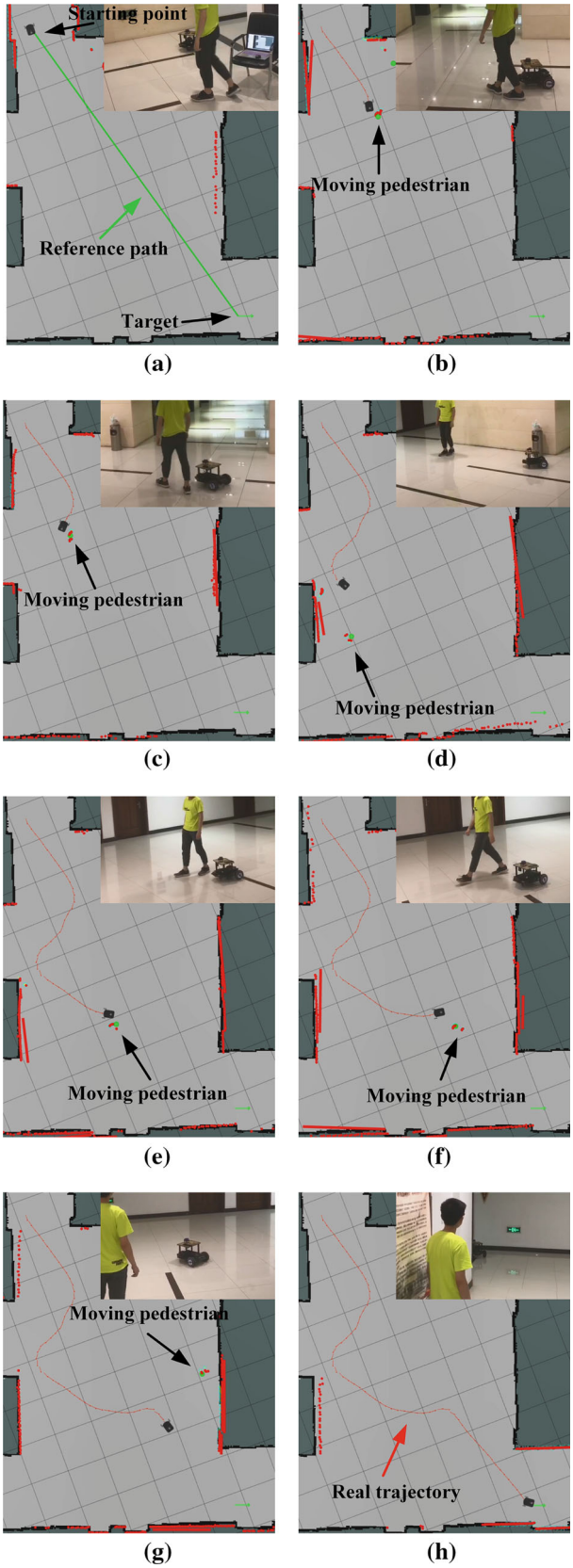


Fig. 33 The process of robot experiment 2. **a**–**h** Respectively shows the robot position in chronological order

trian. When a pedestrian hinders the robot motion, the robot steers away from this pedestrian (see Fig. 33b, c, e, f). When the pedestrian is far from the robot, the robot returns to the target direction (see Fig. 33d, g). Figure 33e shows the real trajectory as the robot successfully reaches the target. It can be seen that the real trajectory avoids the moving pedestrian and is quite different from the reference path, which proves that the proposed method can effectively navigate the robot in the simple dynamic environment.

7 Conclusion

As a classic local path planning algorithm, DWA has little capability of global search and relies on a reference path, which makes the robot navigation hard to achieve when the global environment is unknown. In this paper, a modified method for mobile robot navigation in unknown environments is proposed using the improved DWA combined with Q-learning. We improve the original DWA by modifying and extending the original evaluation functions, whose calculation method are redesigned and number of the functions is increased to enhance the navigation capability of DWA. In addition, the weights of each evaluation function were difficult to choose, so the Q-learning is applied to adaptively tune these weights. In order to apply Q-learning to robot navigation in an unknown environment, the state space, action space and reward function are defined. These definitions try to mimic the human perception, reasoning and handling of the unknown environment. Meanwhile they balance navigation performance and computation cost.

Series of simulations are carried out to verify the performance of the improved DWA and its combination with Q-learning. The first part of simulations shows that the efficiency of navigation of the improved DWA is significantly better than that of the original DWA in both environments with discrete and spiral obstacle. In the second part of simulations, the training process shows the trend of decreasing time consumption and increasing success rate, the test result shows that the success rate of the DWA with the trained agent is significantly higher than that of DWA with the fixed parameters, and the comparisons prove the superiority of our method against other local planners. The proposed method is validated by the hardware experiments based on XQ-4 Pro robot. The experimental results show that the improved DWA with trained agent is able to navigate the robot in both static and dynamic unknown environment.

As future work, dynamic obstacles will be considered in the training process and more complex obstacles will be set in the experiment. Also, we may study pedestrian behaviors avoiding robot to promote the robot obstacle avoidance.

References

Agha-Mohammadi, A. A., Chakravorty, S., & Amato, N. M. (2014). Firm: Sampling-based feedback motion-planning under motion uncertainty and imperfect measurements. *The International Journal of Robotics Research*, *33*(2), 268–304.

Azzabi, A., & Nouri, K. (2019). An advanced potential field method proposed for mobile robot path planning. *Transactions of the Institute of Measurement and Control*,. https://doi.org/10.1177/0142331218824393.

Ballesteros, J., Urdiales, C., Velasco, A. B. M., & Ramos-Jimenez, G. (2017). A biomimetical dynamic window approach to navigation for collaborative control. *IEEE Transactions on Human–Machine Systems*, *47*(6), 1123–1133.

Bayili, S., & Polat, F. (2011). Limited-damage A*: A path search algorithm that considers damage as a feasibility criterion. *Knowledge-Based Systems*, *24*(4), 501–512.

Best, G., Faigl, J., & Fitch, R. (2017). Online planning for multi-robot active perception with self-organising maps. *Autonomous Robots*, *42*(4), 715–738.

Brock, O., & Oussama, K. (1999). High-speed navigation using the global dynamic window approach. In *1999 IEEE international conference* (pp. 341–346).

Chang, L., Shan, L., Li, J.,& Dai, Y. W. (2019). The path planning of mobile robots based on an improved A* algorithm. In *2019 IEEE 16th international conference on networking, sensing and control (ICNSC)* (pp. 257–262).

Contreras-Cruz, M. A., Ayala-Ramirez, V., & Hernandez-Belmonte, U. H. (2015). Mobile robot path planning using artificial bee colony and evolutionary programming. *Applied Soft Computing*, *30*(2015), 319–328.

Das, P. K., Behera, H. S., & Panigrahi, B. K. (2015). Intelligent-based multi-robot path planning inspired by improved classical Q-learning and improved particle swarm optimization with perturbed velocity. *Engineering Science and Technology, an International Journal*,. https://doi.org/10.1016/j.jestch.2015.09.009.

Das, P. K., Behera, H. S., & Panigrahi, B. K. (2016). Intelligent-based multi-robot path planning inspired by improved classical q-learning and improved particle swarm optimization with perturbed velocity. *Engineering Science and Technology, an International Journal*, *19*(1), 651–669.

Duguleana, M., & Mogan, G. (2016). Neural networks based reinforcement learning for mobile robots obstacle avoidance. *Expert Systems with Applications*, *62*(2016), 104–115.

Durrant, W. H. (1994). Where am I? A tutorial on mobile vehicle localization. *Industrial Robot*, *21*(2), 11–16.

Elbanhawi, M., & Simic, M. (2014). Sampling-based robot motion planning: A review. *IEEE Access*, *2*(2014), 56–77.

Fox, D., Burgard, W., & Thrun, S. (2002). The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, *4*(1), 23–33.

Fu, Y., Ding, M., Zhou, C., & Han, H. (2013). Route planning for unmanned aerial vehicle (UAV) on the sea using hybrid differential evolution and quantum-behaved particle swarm optimization. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *43*(6), 1451–1465.

González, R., Jayakumar, P., & Iagnemma, K. (2017). Stochastic mobility prediction of ground vehicles over large spatial regions: A geostatistical approach. *Autonomous Robots*, *41*(2), 311–331.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2), 100–107.

Henkel, C., Bubeck, A., & Xu, W. (2016). Energy efficient dynamic window approach for local path planning in mobile service robotics *. *IFAC PapersOnLine*, *49*(15), 32–37.

Hossain, M. A., & Ferdous, I. (2015). Autonomous robot path planning in dynamic environment using a new optimization technique inspired by bacterial foraging technique. *Robotics and Autonomous Systems*, *64*(2015), 137–141.

Ishay, K., Elon, R., & Ehud, R. (1998). TangentBug: A range-sensor-based navigation algorithm. *The International Journal of Robotics Research*, *17*(9), 934–953.

Jaradat, M. A. K., Al-Rousan, M., & Quadan, L. (2011). Reinforcement based mobile robot navigation in dynamic environment. *Robotics and Computer-Integrated Manufacturing*, *27*(1), 135–149.

Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, *30*(7), 846–894.

Khaled, B., Froduald, K., & Leo, H. (2013). Randomized path planning with preferences in highly complex dynamic environments.*Robotica*, *31*(8), 1195–1208.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, *5*(1), 90–98.

Kim, S., & Kim, H. (2010). Optimally overlapped ultrasonic sensor ring design for minimal positional uncertainty in obstacle detection. *International Journal of Control, Automation, and Systems*, *8*(6), 1280–1287.

Kiss, D. (2012). A receding horizon control approach to navigation in virtual corridors. *Applied Computational Intelligence in Engineering and Information Technology*, *1*(2012), 175–186.

Kiss, D., & Tevesz, G. (2012). Advanced dynamic window based navigation approach using model predictive control. *International conference on methods & models in automation & robotics* (pp. 148–153).

Kovács, B., Szayer, G., Tajti, F., Burdelis, M., & Korondi, P. (2016). A novel potential field method for path planning of mobile robots by adapting animal motion attributes. *Robotics and Autonomous Systems*, *82*(C), 24–34.

Kröse, Ben J. A. (1995). Learning from delayed rewards. *Robotics and Autonomous Systems*, *15*(4), 233–235.

Lamini, C., Fathi, Y., & Benhlima, S. (2015). Collaborative Q-learning path planning for autonomous robots based on holonic multi-agent system. In *International conference on intelligent systems: theories & applications* (pp. 1–6).

Langer, R. A., Coelho, L. S., & Oliveira G. H. C. (2007). K-Bug, a new bug approach for mobile robot's path planning. Control applications. In *2007 IEEE international conference on control applications* (pp. 403–408).

Li, G., & Chou, W. (2016). An improved potential field method for mobile robot navigation. *High Technology Letters*, *22*(1), 16–23.

Li, M., Song, Q., Zhao, Q. J., & Zhang, Y. L. (2016). Route planning for unmanned aerial vehicle based on rolling RRT in unknown environment. In *2016 IEEE international conference on computational intelligence and computing research* (pp. 1–4).

Likhachev, M., Ferguson, D., Gordon, G., Stentz, A., & Thrun, S. (2008). Anytime search in dynamic graphs. *Artificial Intelligence*, *172*(14), 1613–1643.

Lumelsky, V. J., & Skewis, T. (1990). Incorporating range sensing in the robot navigation function. *IEEE Transactions on Systems, Man and Cybernetics*, *20*(5), 1058–1069.

Lumelsky, V. J., & Stepanov, A. A. (1987). Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, *2*(1), 403–430.

Lynda, D. (2015). E-Bug: New bug path-planning algorithm for autonomous robot in unknown environment. In *2015 international conference on information procession, security and advanced communications* (pp. 1–8).

Maroti, A., Szaloki, D., Kiss, D., & Tevesz, G. (2013). Investigation of dynamic window based navigation algorithms on a real robot. In *2013 IEEE 11th international symposium on applied machine intelligence and informatics (SAMI)*(pp. 95–100).

Mohanty, P. K., Sah, A. K., Kumar, V., & Kundu, S. (2017). Application of deep Q-learning for wheel mobile robot navigation. *International conference on computational intelligence & networks* (pp. 88–93).

Monfared, H., & Salmanpour, S. (2015). Generalized intelligent water drops algorithm by fuzzy local search and intersection operators on partitioning graph for path planning problem. *Journal of Intelligent & Fuzzy Systems*, *29*(2), 975–986.

Ogren, P., & Leonard, N. E. (2005). A convergent dynamic window approach to obstacle avoidance. *IEEE Transactions on Robotics*, *21*(2), 188–195.

Özdemi, A., & Sezer, V. (2018). Follow the gap with dynamic window approach. *International Journal of Semantic Computing*, *12*(01), 43–57.

Pinto, A. M., Moreira, E., Lima, J., Sousa, J. P., & Costa, P. (2016). A cable-driven robot for architectural constructions: a visual-guided approach for motion control and path-planning. *Autonomous Robots*, *41*(7), 1487–1499.

Qureshi, A. H., & Ayaz, Y. (2016). Potential functions based sampling heuristic for optimal path planning. *Autonomous Robots*, *40*(6), 1079–1093.

Rickert, M., Brock, O., & Knoll, A. (2008). Balancing exploration and exploitation in motion planning. In *IEEE international conference on robotics & automation* (pp. 2812–2817).

Seder, M. & Petrović, I. (2007). Dynamic window based approach to mobile robot motion control in the presence of moving obstacles. In *Proceedings of the 2007 IEEE international conference on robotics and automation* (pp. 1986–1991).

Sharma, A., Gupta, K., Kumar, A., Sharma, A., & Kumar, R. (2017). Model based path planning using Q-Learning. In *2017 IEEE international conference on industrial technology (ICIT)* (pp. 837–842).

Simmons, R. (1996). The curvature-velocity method for local obstacle avoidance. In *IEEE international conference on robotics & automation* (pp. 3375–3382).

Smart, W. D., & Kaelbling, L. P. (2002). Effective reinforcement learning for mobile robots. In *IEEE international conference on robotics and automation* (pp. 3404–3410).

Stentz A. (1995). The focussed D* algorithm for real-time replanning. In *International joint conference on artificial intelligence* (pp. 1662–1669).

Wang, Y. X., Tian, Y. Y., Li, X., & Li, L. H. (2019). Self-adaptive dynamic window approach in dense obstacles. *Control and Decision*, *34*(02), 34–43.

Wang, Z., Shi, Z., Li, Y., & Tu, J. (2014). The optimization of path planning for multi-robot system using Boltzmann Policy based Q-learning algorithm. In *2013 IEEE international conference on robotics and biomimetics (ROBIO)* (pp. 1199–1204).

Watkins. (1992). Technical note: Q-learning. *Machine Learning*, *8*(3–4), 279–292.

Xin, Y., Liang, H. W., Du, M., Mei, T., Wang, Z. L., & Jiang, R. (2014). An improved A* algorithm for searching infinite neighbourhoods. *Robot*, *36*(5), 627–633.

Xu, Y. L. (2017). Research on mapping and navigation technology of mobile robot based on ROS. M.A. Thesis. Harbin: Harbin Institute of Technology.

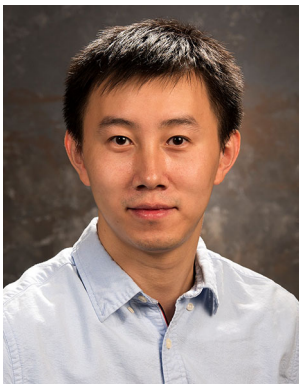
Zhang, A., Chong, C., & Bi, W. (2016). Rectangle expansion A* pathfinding for grid maps. *Chinese Journal of Aeronautics*, *29*(5), 1385–1396.

Zhang, J., & Singh, S. (2017). Low-drift and real-time lidar odometry and mapping. *Autonomous Robots*, *41*(2), 401–416.

Zhao, X., Wang, Z., Huang, C. K., & Zhao, Y. W. (2018). Mobile robot path planning based on an improved A* algorithm. *Robot*, *40*(06), 137–144.

Lu Chang received his B.S. degree in Electrical Engineering from the Jiangsu University of Science and Technology, Zhenjiang, P.R. China in 2016, and the M.S. degree in System Engineering from the Nanjing University of Science and Technology. He is currently a Ph.D. student in Control Science and Engineering at the Nanjing University of Science and Technology. His research interests include robot path planning and SLAM algorithm, machine learning and human–robot interaction.

Liang Shan received his B.S. degree in Electrical Engineering and the Ph.D. degree in Control Science and Control Engineering from the Nanjing University of Science and Technology, Nanjing, P.R. China in 2002 and 2007, respectively. He is currently an Associate Professor with the Nanjing University of Science and Technology. His research interests include mobile robot technology, system detection and modeling, and intelligence artificial algorithm.



Chao Jiang received his B.S. degree in Measuring and Control Technology and Instrumentation from Chongqing University, Chongqing, China, in 2009, and his Ph.D. degree in Electrical Engineering from Stevens Institute of Technology, Hoboken, NJ, USA, in 2019. He worked as a Research Assistant in the Key Laboratory of Optoelectronic Technology and Systems (Chongqing University), Ministry of Education, China, from 2009 to 2012. He joined the Department of Electrical and Computer Engineering at the University of Wyoming, Laramie, WY in 2019, where he is currently an Assistant Professor. His main research interests include autonomous robots, human–robot interaction, robotic learning, deep reinforcement learning, and multi-robot cooperative control. He is the recipient of the Innovation & Entrepreneurship Doctoral Fellowship at Stevens Institute of Technology from 2012 to 2016 and the recipient of the Outstanding Doctoral Dissertation Award in Electrical Engineering at Stevens Institute of Technology in 2019. He is a member of IEEE.

Yuewei Dai received his B.S. and M.S. degrees in system engineering from the East China Institute of Technology, Nanjing, P.R. China, in 1984 and 1987, respectively, and the Ph.D. degree in control science and engineering from the Nanjing University of Science and Technology, in 2002. He is currently a professor with the School of Automation, Nanjing University of Science and Technology. His research interests are in multimedia security, system engineering theory, and network security.