



An empirical evaluation of Q-learning in autonomous mobile robots in static and dynamic environments using simulation

Ee Soong Low, Pauline Ong^{*}, Cheng Yee Low

Faculty of Mechanical and Manufacturing Engineering, Universiti Tun Hussein Onn Malaysia (UTHM), 86400 Parit Raja, Batu Pahat, Johor, Malaysia

ARTICLE INFO

Keywords:

Autonomous mobile robot
Real world experiment
Static environment
Dynamic environment
Path planning
Q-learning

ABSTRACT

Path planning plays a crucial role in the navigation of mobile robots. Among the various path planning techniques, Q-learning (QL) has gained popularity as a reinforcement learning approach that exhibits the ability to learn without significant prior knowledge of the environment. However, despite the introduction of enhanced versions of Q-learning, specifically distance metric and moving target Q-learning (DMMTQL) and distortion and optimization Q-learning (DOQL), the validation of these algorithms in real-world scenarios is incomplete. In this study, we conduct real-world experiments to assess the performance of DMMTQL and DOQL in two distinct environments: one featuring static and another with dynamic obstacles. Our investigation involves comparing the real-world results obtained from DMMTQL and DOQL with those obtained from QL and contrasting them with simulation results. The findings from our real-world experiments demonstrate that both DMMTQL and DOQL outperform QL in terms of path planning effectiveness. Both improved QL algorithms are able to analyse and decide the optimal path with free collision for the mobile robots. When comparing the improvements achieved by DMMTQL and DOQL with simulation results, we observe similar outcomes in most aspects, with the exception of the time taken and distance travelled metrics.

1. Introduction

Autonomous mobile robots (AMRs) have a wide range of applications in various domains, including self-driving vehicles [1,2], automated guided vehicles (AGVs) for logistics automation in factories [3,4], and cleaning robots [5,6]. In the field of AMR, path planning plays a vital role as it enables the robot to navigate from one location to another while avoiding obstacles and preventing collisions with objects [7]. An effective path design ensures faster and safer transportation of goods, thereby enhancing overall operational efficiency. Consequently, path planning has emerged as a fundamental research area for scholars and practitioners alike. The environment in which path planning occurs can be broadly classified into two categories: (i) static environments, where obstacle positions remain fixed throughout the path planning process, and (ii) dynamic environments, where the navigation environment comprises moving obstacles or obstacles that appear spontaneously over time [7].

A wide range of methods and algorithms have been developed to facilitate collision-free path planning for AMRs [8–13]. These approaches can be categorized into two main groups: classical approaches and artificial intelligence (AI) approaches [14–18]. The classical approaches commonly employed include cell decomposition (CD) [19–21], roadmap (RM) [22–24], and artificial potential field (APF) [25–27]. On the other hand, artificial intelligence approaches, which

have gained popularity, include artificial neural network (ANN) [28–30], fuzzy logic (FL) [31–34], metaheuristic optimization algorithms [35–37], and reinforcement learning (RL) [38–40].

Classical approaches were traditionally utilized to address path planning challenges; as artificial intelligence approaches were not yet established. However, classical approaches exhibit certain limitations, such as an inability to effectively handle uncertainty [14–16], high computational demands [7,14–16,41], reliance on complete environmental information for operation [16], time-consuming processes [36,42], significant memory requirements [36,41], and susceptibility to local minimum trapping [7,42]. Despite efforts to enhance classical approaches for unknown environments, the results have fallen short of achieving optimal performance [14]. In light of the limitations and bottlenecks encountered by classical approaches, researchers have turned their attention towards a more promising alternative. AI approaches have gained significant favour among researchers when compared to classical methods. This is primarily due to the superior capabilities of AI approaches in handling environmental changes [14,15] and operating in unknown environments [14,16].

Among the various AI approaches, Q-learning (QL), initially introduced by Watkins [43], holds great potential for addressing complex optimization problems in path planning. Q-learning offers notable advantages, such as being model-free and possessing the ability to learn

^{*} Corresponding author.

E-mail addresses: low_es@hotmail.com (E.S. Low), ongp@uthm.edu.my (P. Ong), cylow@uthm.edu.my (C.Y. Low).

and update knowledge through continuous interaction with the environment [44]. Despite the demonstrated effectiveness of Q-learning, it is not without its limitations. The drawbacks of Q-learning include slow learning speed and issues related to memory size requirements [45,46].

2. Recent development of QL

Significant efforts have been devoted to addressing the inherent limitations of QL. As QL comprises several components, such as the reward function, action selection function, Q-table and, Q-formula function, various enhancements have been introduced to improve the overall performance of Q-learning. These improvements involve modifying different aspects of QL by incorporating ANN, optimization algorithms, distance information, the A^* algorithm, and other techniques. These modifications aim to augment the capabilities of QL and enhance its effectiveness in solving complex path planning problems.

Researchers have made various modifications to the Q-table of QL, which is responsible for storing and retrieving information collected by the agent for each state–action pair. In [45], the Q-table was initialized using the flower pollination algorithm (FPA). The results demonstrated improved learning time and higher consistency compared to traditional QL approaches. Another improvement to the Q-table was introduced by Cruz and Yu [47], who utilized kernel smoothing and ANN in an unknown environment. Kernel smoothing was employed to convert the discrete state representation of the Q-table into a continuous state representation, while unknown states were estimated using ANN. This enhancement enabled multiple AMRs to navigate effectively in unknown environments. In [46], the size of the Q-table was reduced by assigning states and actions to the vertices of obstacles instead of the positions on the grid. Additionally, the Q-table was initialized using a reward function based on distance information. Moreover, modifications were made to the reward function to assign higher rewards to states with better coordination, i.e., states that could avoid collisions and achieve shorter distances. As a result, the proposed QL approach exhibited enhanced learning capabilities, produced shorter path lengths, and required less time for computation.

The action selection function plays a crucial role in determining the chosen action at a given state. In a study by Wang et al. [48], two ANNs were integrated into the Deep-Q Network (DQN) for action selection. Each network focused on two important aspects: target approaching and obstacle avoiding. The results demonstrated the generation of smooth and safe paths in real-time applications. Another approach, presented in [18], involved partially basing the action selection function on the direction of the target, while replacing the Q-table with an ANN model. This proposed strategy exhibited a faster convergence rate, required fewer steps to reach the target, and demonstrated adaptability in unknown environments. Similarly, Yan et al. [38] proposed a similar approach where an ANN was used to replace the Q-table for Q-value evaluation and target direction was incorporated in the action selection process. Remarkable improvements were observed in terms of success rate, path distance, learning ability, and safety compared to other algorithms. The introduced enhancements in terms of success rate, path distance, learning ability, and safety achieved by these approaches surpassed those of other algorithms, highlighting their significant contributions in path planning and decision-making for autonomous systems.

Upon taking an action, the reward function plays a vital role by providing feedback to the agent, assisting in the assessment of the quality of the selected action. In [17], the distance between the current state and the target position was incorporated into the reward function for an autonomous ship. The findings demonstrated that the proposed algorithm achieved enhanced learning effectiveness and exhibited manoeuvring capabilities comparable to human operators when compared to the previous method.

The Q-formula function plays a crucial role in calculating the Q-values before they are stored in the Q-table. In [40], the Q-formula

function was modified to facilitate the implementation of an asynchronous multi-agent system. To achieve this, a global Q-formula was introduced to receive and calculate Q-values from all agents. The proposed algorithm exhibited significant improvements in terms of learning rate and convergence rate compared to QL methods. Additionally, Das et al. [49] made modifications to the Q-formula function along with reducing the size of the Q-table. Instead of storing Q-values for all actions, only the Q-value of the best action was retained. Moreover, a Boolean variable called “lock” was incorporated into the Q-table. Different Q-formulas were utilized based on the lock status and the distance information between the current, next, and goal states. Only the Q-values of selected states, determined by an optimization algorithm, were updated. Consequently, states with the highest Q-value were assigned as the global and local best particles for the optimization algorithm. This approach resulted in achieving minimal path distance and time taken, as well as smaller turning angles in multi-robot path planning scenarios.

In recent advancements of QL, two improved versions of QL have been introduced in studies [50,51]. In [50], the integration of a distance metric and a moving target enhances the learning speed of QL, leading to the development of the improved QL referred to as distance metric and moving target Q-learning (DMMTQL) in this study. On the other hand, the study in [51] improves QL by incorporating the distortion concept and an optimization algorithm, resulting in an enhanced efficiency of QL in avoiding dynamic obstacles. This improved QL is referred to as distortion and optimization Q-learning (DOQL). While the simulation results of DMMTQL and DOQL exhibit promising outcomes, this study aims to conduct real-world experiments to ascertain the feasibility of these algorithms. The contributions of this study are as follows:

- (i) Real-world experiments are conducted to validate the feasibility of the QL, DMMTQL, and DOQL algorithms for analysing and determining the optimal path with free collision and the shortest distance in practical scenarios.
- (ii) The performance of DMMTQL and DOQL is compared to that of QL in terms of path planning, time taken, distance travelled, and success rate. Both DMMTQL and DOQL are expected to outperform QL in these evaluations.
- (iii) The results obtained from the real-world experiments are compared with the simulation results presented in [50,51] in terms of path planning, time taken, distance travelled, and success rate.

The paper follows a structured outline, which is presented as follows. Section 3 provides a comprehensive discussion on the problem formulation of QL. In Section 4, the experiment setup is detailed, highlighting the specific methodologies employed. The obtained results are thoroughly analysed and discussed in Section 5. Finally, the paper concludes the work in Section 6 by summarizing the key findings and their implications.

3. Problem formulation and QL

3.1. Configuration space

In the context of path planning, the real-world environment undergoes simplification through the utilization of various map representation techniques. This simplification results in the creation of a configuration space, also known as C-space. For the purpose of representing the navigation environment in this study, a grid-based C-space approach was employed. Specifically, a two-dimensional navigation environment within a Cartesian plane was considered, with the dimensions defined as a length u and a width v . Each individual grid within this grid-based C-space corresponds to a state within the QL algorithm and is denoted by its x -coordinate and y -coordinate values. Fig. 1 provides an illustration of a grid-based C-space configuration, depicting a 20×20 dimensional space containing various obstacles, an AMR, a starting point, and a target position. It is important to note the following assumptions made in this study:

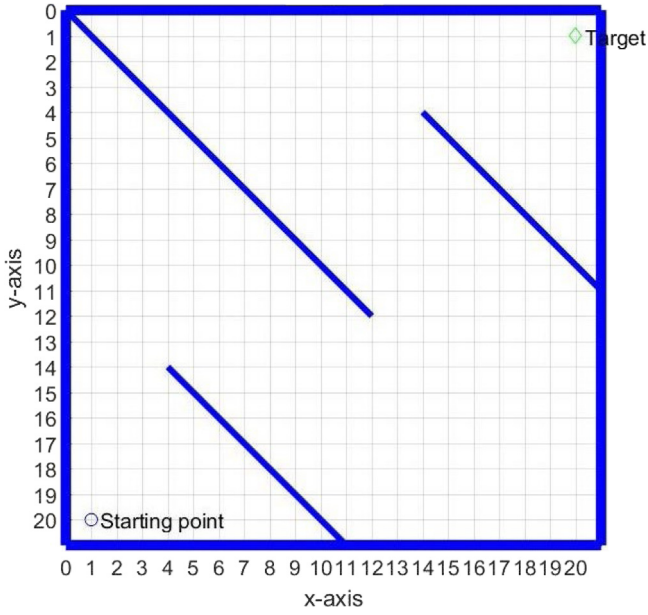


Fig. 1. An illustrative instance of a grid-based C-space configuration is presented, featuring multiple obstacles, an Autonomous Mobile Robot (AMR), a starting point, and a target position. This depiction serves to provide a visual representation of the grid-based C-space, highlighting the layout of the environment in relation to the AMR's navigation.

- (i) The information regarding the environment, including the precise locations of the AMR, the target position, and the obstacles, is known and available.
- (ii) The maximum allowable movement for the AMR is restricted to one-unit grid within each time frame.

3.2. Q-learning

QL [52] is a widely utilized model-free reinforcement learning algorithm. The fundamental components of QL are depicted in Fig. 2, encompassing the agent, environment, action, reward, and state. Through interaction with the environment, the agent acquires knowledge and adapts its actions to optimize the reward function associated with specific states.

Let us denote the state as $s \in S$ and the action as $a \in A$. At a specific state s , the agent takes an action a and receives a reward $r(s, a)$ as feedback. The reward serves as an evaluative signal, with positive rewards indicating progress towards the goal and negative rewards indicating suboptimal actions. As the agent transitions to a new state

s' , a Q-value $Q(s, a)$ is computed and assigned to the state-action pair (s, a) using the following formula:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r(s, a) + \gamma \max_{a' \in A} Q(s', a') - Q(s, a) \right] \quad (1)$$

Here, $Q(s, a)$ represents the Q-value of the state-action pair (s, a) , $\max_{a' \in A} Q(s', a')$ represents the highest Q-value among all possible actions in the next state s' , r represents the received reward, α represents the learning rate, and γ represents the discount factor. The Q-table, also known as the state-action table, is employed to store and update the Q-values for each state-action pair. Subsequently, when the agent revisits a particular state, the Q-table serves as a reference for selecting the best action based on the associated Q-value.

In this study, the environmental representation employs a grid-based approach, where each state is represented in the format of (x, y) . The state set, denoted as S , is defined as follows:

$$S = \{ s | (x, y) \} \quad (2)$$

The action set, denoted as A , comprises a collection of permissible actions that allow the agent to transition from the current state to the subsequent state. In this study, the agent has the capability to move in eight distinct directions from the current state, as specified below:

$$A = \left\{ \begin{array}{l} \text{forward, forward left, forward right, backward,} \\ \text{backward left, backward right, left, right} \end{array} \right\} \quad (3)$$

Among the feasible actions, the action with the maximum Q-value is chosen at the current state. In this study, the reward function R is defined as follows:

$$\text{Reward, } R = \begin{cases} 1, & \text{when reached the target} \\ -1, & \text{when met obstacle} \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

The flowchart of the QL algorithm is depicted in Fig. 3. Initially, the Q-table is initialized. If the current episode is less than the maximum episode, the initialization of the starting and target points takes place. Otherwise, the flow terminates. An action with the highest Q-value among all available actions in the current state is selected. Following the execution of the chosen action, the agent transitions to a new state. A reward is received upon reaching the new state, and an update to the corresponding Q-value is performed. If the new state is not the target point, the process repeats by selecting a new action. However, if the new state is the target point, a new episode commences, and the current episode is checked.

3.3. Distance metric and moving target Q-learning (DMMTQL)

The DMMTQL [50] is an improved QL, which has been enhanced through the integration of three key features: the introduction of a distance metric, the modification of the Q-function, and the incorporation

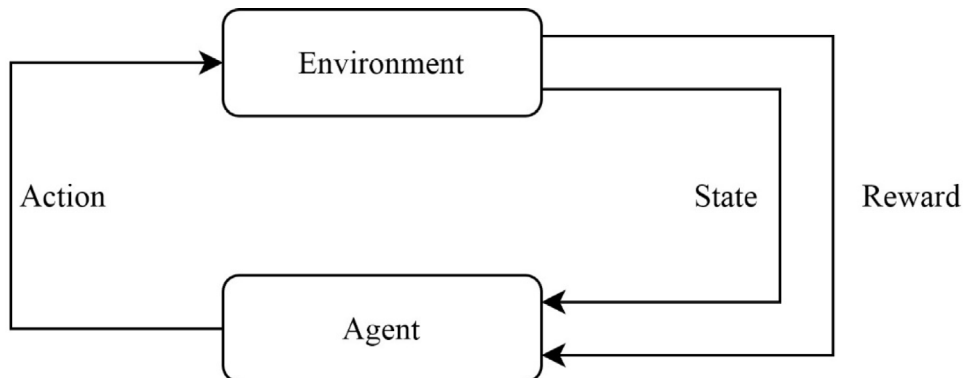


Fig. 2. Relationship of agent and environment.

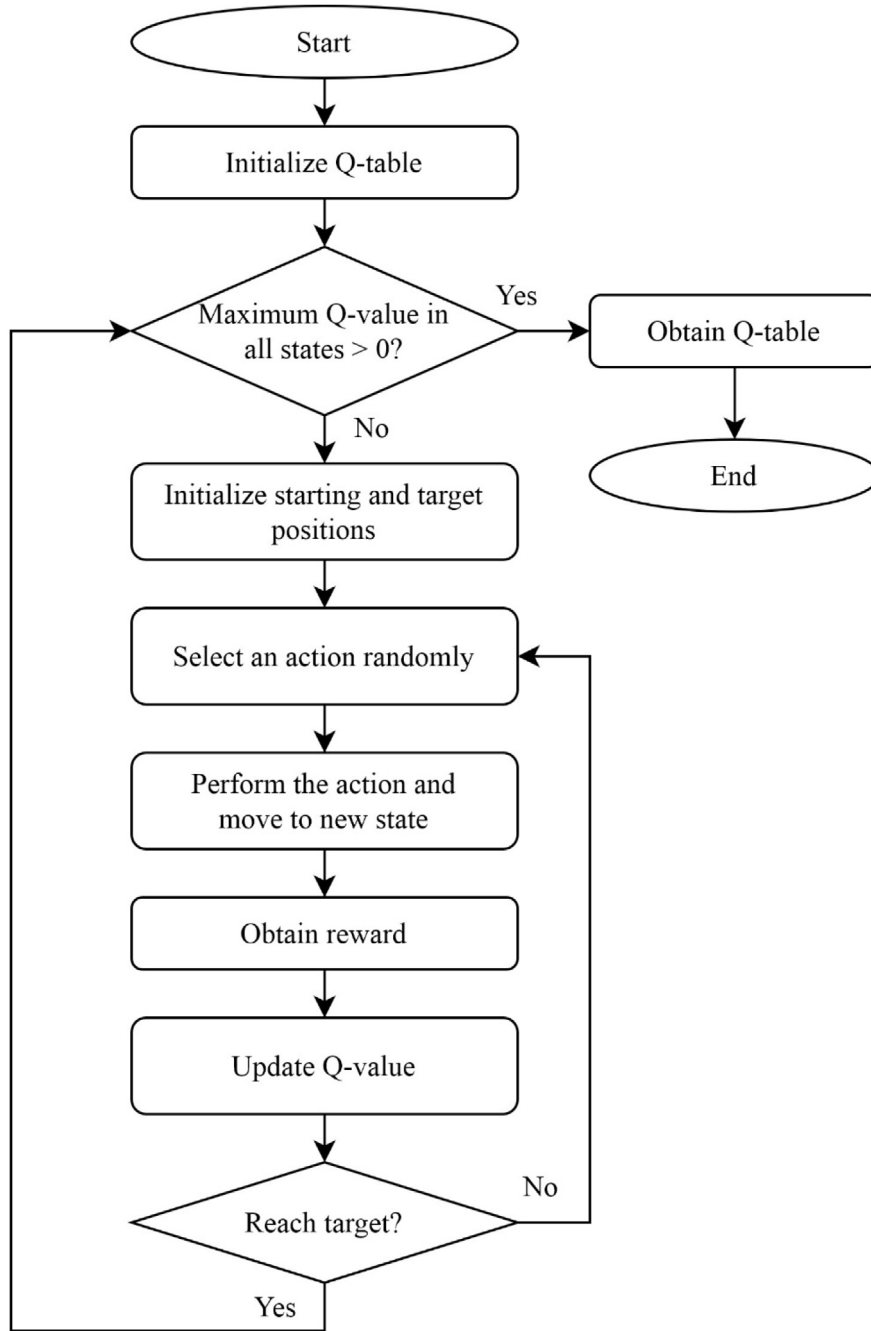


Fig. 3. Flowchart of QL.

of a virtual target. The distance metric is employed to provide guidance to an AMR in navigating towards the desired target direction. By calculating the distance between subsequent states and the target, the metric assists in determining the optimal path for the AMR. However, the implementation of the distance metric gives rise to the issue of dead-ends. To effectively address this problem, modifications are made to the Q-function, which is responsible for estimating the expected rewards associated with various actions in Q-learning algorithms. The refined Q-function aims to facilitate improved exploration and exploitation of the state-action space, enabling the AMR to navigate around dead-ends more efficiently. Additionally, a virtual target is introduced to overcome dead-ends and generate a smoother path. This virtual target serves as a reference point, enabling the robot to identify alternative paths when faced with dead-ends, thereby ensuring continuous

progress towards the ultimate target. Further details of DMMTQL can be referred to [50].

3.4. Distortion and optimization Q-learning (DOQL)

DOQL [51] is developed as a solution for operating in environments with dynamic obstacles. The framework comprises three distinct modes: normal mode, distortion mode, and optimization mode. Under normal mode, which is engaged when a dynamic obstacle is not detected, DOQL functions as a standard QL algorithm, where states with the highest Q-values are selected. Conversely, distortion mode is triggered upon the detection of a dynamic obstacle. In this mode, the Q-values of states surrounding the dynamic obstacle are intentionally distorted or reduced to discourage the AMR from moving towards them. This distortion ensures that the AMR avoids colliding with the

dynamic obstacle. However, it is important to note that this approach may lead to local minima in certain distorted states, where the AMR gets trapped. To overcome this challenge, an optimization mode is activated, which aims to guide the AMR out of local minima and towards more favourable regions of the state-action space. The optimization mode ensures the continuous progression of the AMR towards its objective, even in the presence of dynamic obstacles. More details of DOQL are omitted here for brevity and can be referred to [51].

4. Experiment setup

In the context of real-world environments, three evaluation metrics are employed to assess the performance of path planning algorithms: time taken, distance travelled and success rate. In these evaluations, if the AMR collides with obstacles, it is deemed a failure. Since the real-world environment experiments serve as validation, all path planning algorithms are executed once on the given navigation map. The distance travelled metric is determined by the following equation:

$$distance = \sum_{i=1}^{n-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \quad (5)$$

where (x_i, y_i) represents the current position of the AMR, (x_{i+1}, y_{i+1}) represents the next position of the AMR, i is the number of iterations, and n represents the last iteration.

The success rate is determined as follows:

$$success\ rate = \text{number of successful runs} / \text{total number of runs} \quad (6)$$

4.1. Hardware used and setup

The hardware configuration for the real-world experiment setup is as follows:

- (i) A smartphone with built-in Wi-Fi, camera, and Droidcam application.
- (ii) A laptop with the following specifications: Intel® Core™ i7-5500U CPU @ 2.40 GHz, 8 GB RAM, and Windows 8.1 Pro 64-bit operating system. The laptop is installed with MATLAB R2017a and MakerCode for micro:bit software.
- (iii) Two Micro:bit microcontrollers.
- (iv) Three Tiny:bit smart robot cars.

The Micro:bit microcontroller is equipped with a 32-bit 16 MHz ARM Cortex-M0 processor, along with 25 LED lights and two buttons. Additionally, the microcontroller features built-in sensors such as a compass and accelerometer. It can be powered either through the USB connector or the battery socket. General-purpose input/output (GPIO) pins are responsible for handling input and output signals, with support from the 3 V power supply and ground pins. Wireless communication is facilitated through Bluetooth low energy (BLE) and radio wave technologies. To program the microcontroller, the MakerCode for microbit software is utilized.

In this study, one Tiny:bit smart robot car is designated as the AMR, while the other two Tiny:bit smart robot cars function as dynamic obstacles. The Tiny:bit smart robot car is a pre-assembled robotic vehicle featuring various components. It includes an ultrasonic sensor module, line sensors, a buzzer, two LED car lights, two programmable RGB LEDs, and a microphone. Control of the Tiny:bit smart robot car is achieved through the integration of a Micro:bit microcontroller, which can be easily attached and connected to the Tiny:bit using a readily available pin slot.

The Tiny:bit smart robot car operates as a differential drive wheeled mobile robot, with two rubber wheels located at the rear and a universal wheel at the front. The two rubber wheels are driven by N20 1:10 micro gear DC motors. The motor's gear ratio of 1:10 enables a torque capability of 0.11 kg cm and a speed of 10.3167 rotations per second (rps). The power supply for the Tiny:bit smart robot car is derived

from an 18650 lithium battery with a voltage of 3.7 V. The diameter of the wheels measures 0.04 m. Through calculation, the maximum speed achievable by the Tiny:bit smart robot car is determined to be 1.2966 m/s, as demonstrated by the following calculation:

$$\begin{aligned} speed &= \text{perimeter} \times \text{rps} \\ &= 0.04\pi \times 10.3167 \\ &= 1.2966 \text{ m/s} \end{aligned}$$

The experimental setup involves the placement of a navigation map or game field on the floor, with a smartphone positioned above it to capture a planar view. The smartphone's camera captures images of the game field, which are then transmitted to a laptop via a Wi-Fi connection using the Droidcam application. The received images are processed using MATLAB software running on the laptop. A path planning algorithm is applied to determine the optimal path based on the processed images.

To facilitate communication and control, a Micro:bit microcontroller connected to the laptop via a USB serial port serves as the signal sender. This Micro:bit is responsible for transmitting control signals between the laptop and the Micro:bit on the Tiny:bit smart robot car. Communication between the Micro:bit microcontrollers is established using radio waves. Upon receiving the control signal, the Tiny:bit smart robot car performs the corresponding action. The flow of the signal is illustrated in the flowchart depicted in Fig. 4.

4.2. Image modification setup and map processing

Before conducting the real-world experiment, a setup for image modification is performed to enhance the quality and alignment of the obtained images through the Droidcam application. It is crucial to modify the images as they may exhibit imperfections such as curvature and misalignment. These imperfections are primarily caused by the camera's positioning, which may not be perfectly horizontal or centred with respect to the game field. To address these issues, the obtained images undergo a series of modifications. Firstly, the images are straightened to correct any curvature and ensure a more accurate representation of the game field. Additionally, the images are rotated to align with the orientation of the maps used in the simulated environment. This adjustment enables consistent and standardized processing across different image frames. Furthermore, the modified images undergo a cropping process to exclude unwanted objects, including staircases, doors, and unused areas of the game field. This step is essential to isolate the relevant components of the game field, allowing for accurate allocation of static obstacles, dynamic obstacles, the target position, and the AMR. By performing these image modifications, the final image serves as the foundation for subsequent steps in the experimental procedure, facilitating the proper identification and allocation of various elements within the game field.

In the final image, the locations of obstacles are determined through manual pinpointing. To ensure safety and account for the representation of the AMR in the simulated environment, the size of the obstacle area is enlarged to approximately match the size of the AMR. This enlargement provides a safety margin for the AMR's navigation. Subsequently, the image of the obstacle is reduced in size to 37×29 pixels, enabling its implementation in the path planning process using various path planning algorithms.

The reduction in size is a crucial step to maintain consistency between the navigation map's size and the path planning algorithms. Different algorithms may be impacted differently by changes in the map size, which can lead to significant discrepancies between the results of simulated experiments and real-world experiments. Therefore, resizing the obstacle image ensures compatibility with the chosen path planning algorithms. Similar procedures are employed to determine the location of the target. However, in the case of the target, there is no need for an enlargement of its area since it does not require the same safety margin as the obstacles.

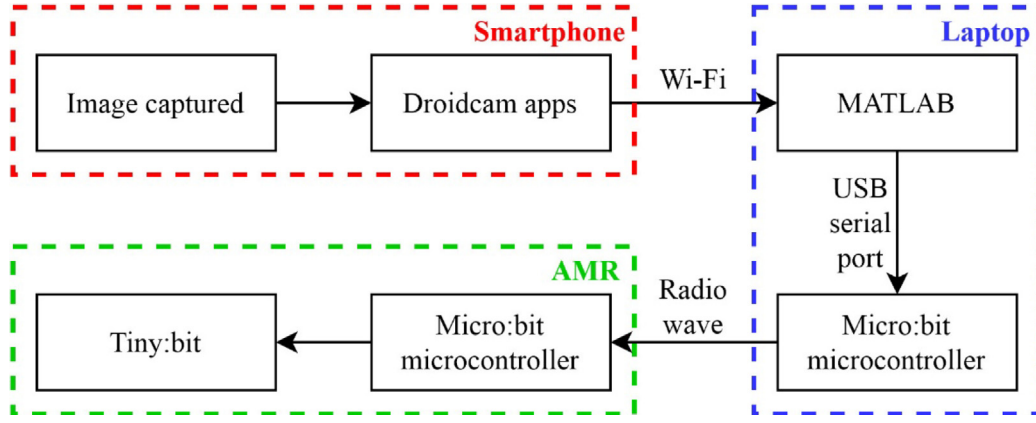


Fig. 4. Flowchart for signal's flow.

Table 1

Colour assigned to programmable LEDs for AMR and dynamic obstacles.

	Left LED	Right LED
AMR	Yellow	Blue
Dynamic obstacle 1	Indigo	Green
Dynamic obstacle 2	Indigo	Green

4.3. Determination and tracking of locations of AMR and dynamic obstacles

To accurately determine the positions of the AMR and dynamic obstacles, the two programmable RGB LEDs on the Tiny:bit smart robot are utilized. The assigned colours for these LEDs corresponding to the AMR and dynamic obstacles are outlined in Table 1. Furthermore, a moving object tracking approach based on the methodology described in [53] is employed.

During the initialization stage of the real-world experiment, the positions of all programmable LEDs on the AMR and dynamic obstacles are manually pinpointed. These programmable LEDs serve as tracking markers, and their movements are tracked using the FPA [54] throughout the experiment.

For each pinpointed position, a 10-pixel image is cropped, and subsequently converted to the Hue Saturation Value (HSV) format to construct a colour histogram. This cropped image, along with its corresponding colour histogram, serves as the target object for the moving object tracking algorithm.

During the experiment, a range of positions within a designated search area are selected using FPA. Each of these selected positions undergoes the same preprocessing steps, including cropping and conversion to HSV histograms. These histograms are subsequently compared with the histogram of the target object.

The similarity between two histograms is assessed using the Bhattacharyya distance [53], which is formulated as follows:

$$\text{Bhattacharyya distance} = \sqrt{1 - \frac{\sum_n \sqrt{H_i(n) H_t(n)}}{\sqrt{H_i H_t N^2}}} \quad (7)$$

where N represents the total number of histogram bins. H_t denotes the histogram of the target object, H_i represents the histogram of solution i , and n denotes the distribution domain ranging from 0 to 255. $H_i(n)$ and $H_t(n)$ represent the pixel count of HSV value n for the solution and target object, respectively. The Bhattacharyya distance serves as the fitness function for the optimization algorithm.

The optimization algorithm identifies the population with the best fitness, which corresponds to the position of the programmable LED. Once the positions of the programmable LEDs on both the AMR and dynamic obstacles are determined, their respective positions can be established as well. The flowchart illustrating this process is summarized in Fig. 5. The parameters of the tracking method and FPA algorithm can be found in Table 2.

Table 2

Parameters of tracking system and FPA.

Parameter	Value
Width of search area	0.05*image width
Height of search area	0.05*image height
Width of cropped image	10 pixels
Height of cropped image	10 pixels
FPA maximum iteration	15
FPA number of variable	2
FPA number of population	20
FPA switch probability, p	0.6

Table 3

Parameters of PID controller for AMR.

Variable	Value
K_p	20
K_i	0
K_d	3
Motor initial	100

4.4. Controlling of AMR and dynamic obstacles

Once a path is determined by the path planning algorithm, it is necessary to scale the path back to match the size of the final image. An illustration of the enlarged path planning, indicated by a black-coloured line, can be observed in Fig. 6. In order to enable the AMR to effectively follow the planned path, a set of 24 virtual sensors is strategically positioned around the AMR. Each virtual sensor is assigned a unique number, beginning with number one at the bottom-left sensor, and subsequently numbering the remaining sensors in a clockwise direction. To ensure smooth and controlled motion of the AMR along the path, a PID controller is implemented.

A variable named "sensor value" is introduced to enable the implementation of the PID controller. The calculation of sensor values for all virtual sensors is based on the equation:

$$\text{sensor value} = |\text{detection} * \text{sensor numbering} - 12| \quad (8)$$

where detection equals to 1 if the sensor detected the path or else 0 and sensor numbering is the number allocated to the sensor.

In the subsequent step, the sensor with the lowest sensor value is selected. Consequently, the error is defined as:

$$\text{error} = \text{selected sensor numbering} - 12 \quad (9)$$

where $\text{selected sensor numbering}$ is the number allocated to the selected sensor.

The parameters for the PID controller applied to the AMR are presented in Table 3. These parameter values are determined through iterative experimentation. The parameter K_p governs the stability of

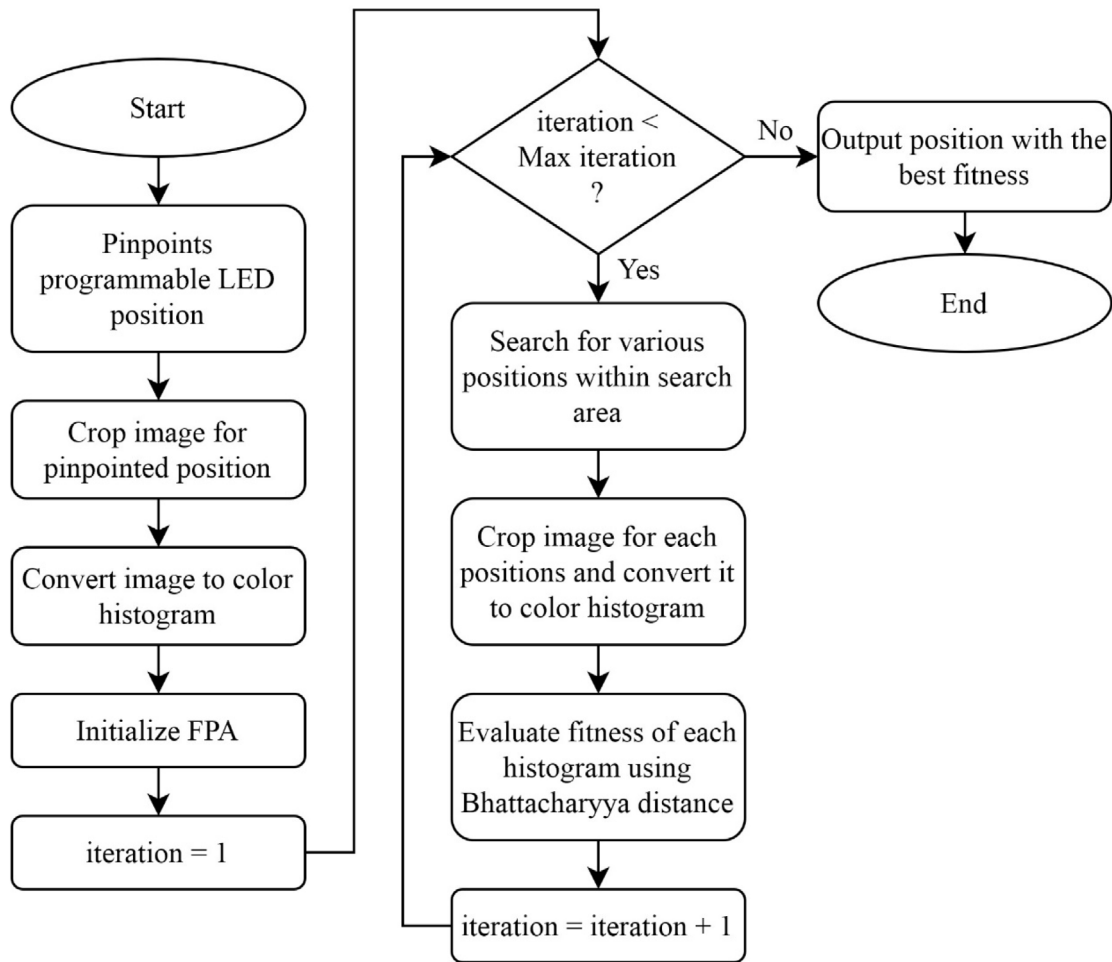


Fig. 5. Flowchart of tracking process using FPA.

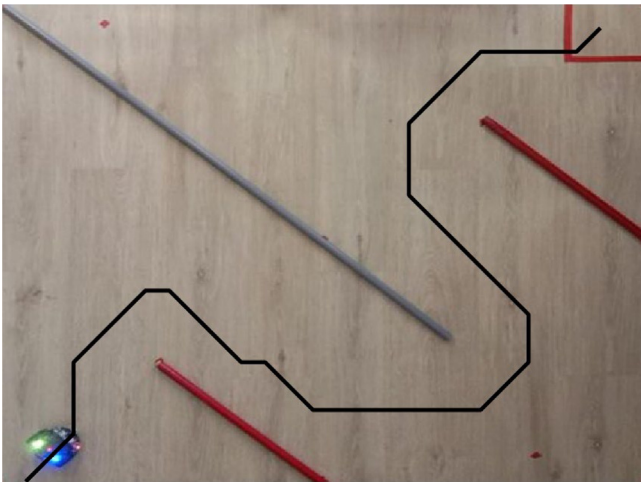


Fig. 6. An example of enlarged path planned and virtual sensor on AMR.

I , and D values are calculated as follows:

$$P = \text{error}$$

$$I = I + \text{error} \quad (10)$$

$$D = \text{error} - \text{previous error}$$

where *previous error* is the error in the previous iteration.

The overall PID value is expressed as:

$$PID \text{ value} = K_p * P + K_i * I + K_d * D \quad (11)$$

The speed signal assigned to the left and right motors of the AMR are formulated as:

$$\begin{aligned} \text{motor left} &= \text{motor initial} + PID \text{ value} \\ \text{motor right} &= \text{motor initial} - PID \text{ value} \end{aligned} \quad (12)$$

The tuning process involves iteratively adjusting the proportional (K_p) and derivative (K_d) gain parameters, while keeping the integral gain (K_i) constant at zero, to achieve precise and stable motion control. The tuning process commences with setting K_p to 1 and K_d to 0 while maintaining K_i at zero, as a slight deviation from the path is permissible. Subsequently, the AMR's response to changes in K_p is evaluated. If K_p is too low, the AMR exhibits sluggish response during turns, leading to deviations from the desired path. Conversely, an excessively high K_p induces zigzag motion during straight-line travel. Accordingly, K_p is incrementally adjusted until an appropriate value is attained. Upon identifying an optimal K_p , the tuning process proceeds to fine-tune K_d . With a suitable value of K_p and zero K_d , the AMR demonstrates improved path-following capabilities, as the risk of sliding out of the path is mitigated. However, the AMR may experience post-turn oscillations

the AMR, with lower values ensuring smoother changes in direction and vice versa. K_i is set to zero, as a slight offset is acceptable for our specific application. K_d influences the speed at which the AMR executes turns, with higher values leading to faster responses and vice versa. The “motor initial” parameter determines the baseline speed of the AMR, with higher values resulting in increased speed and vice versa. The P ,

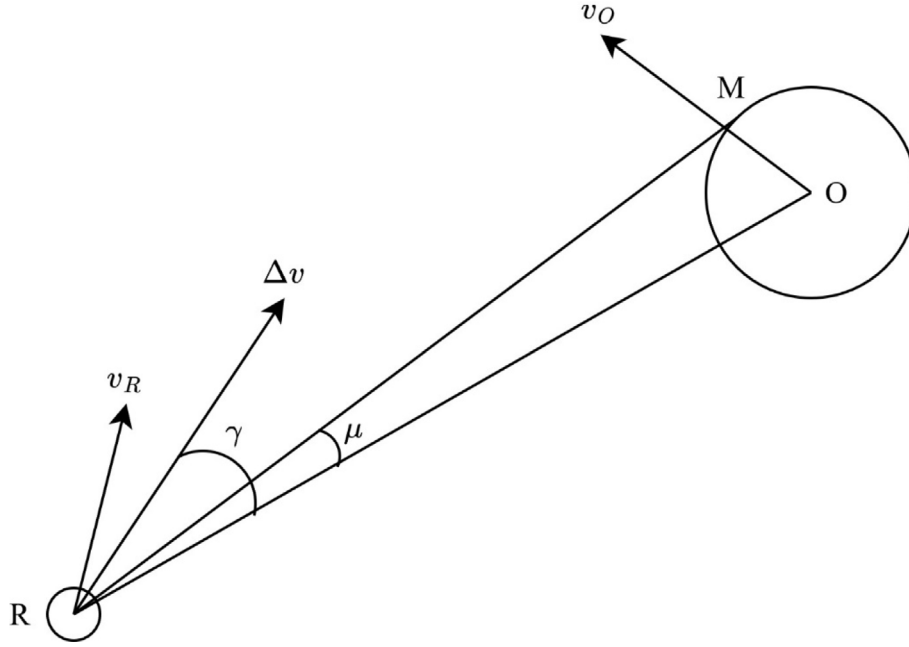


Fig. 7. The motion model [55].

due to an excessive response to errors. By introducing a non-zero K_d , these oscillations are mitigated. Initially, K_d is set to 1, and similar to K_p , it is incrementally adjusted based on the AMR's behaviour. A low K_d value results in post-turn oscillations, while a high K_d induces high-frequency oscillations during straight-line motion. Optimal navigation performance is achieved when both K_p and K_d are fine-tuned to ideal values. With this configuration, the AMR exhibits smooth navigation along straight paths and maintains stability during turns, promptly re-centring itself without oscillatory behaviour.

To control the movement of dynamic obstacles, a similar approach is employed. Paths for dynamic obstacles are manually pinpointed and depicted as red-coloured lines on the final image. In contrast to the AMR, dynamic obstacles are equipped with six virtual sensors since their paths tend to be relatively straighter. These virtual sensors are positioned in a linear arrangement along the front section of the dynamic obstacle, with numbering starting from the left side as sensor number 1 and progressing towards the right.

To indicate the endpoints of the dynamic obstacle's path, horizontal lines (green lines) are added at both ends of the path. These horizontal lines possess different variable values compared to the path of the dynamic obstacle. When the virtual sensors on the dynamic obstacle detect these lines, the dynamic obstacle is programmed to perform a turnaround maneuver. Similar to the AMR, a PID controller is employed to regulate the motion of the dynamic obstacle. However, there are slight differences in the parameters and calculations of the PID controller for the dynamic obstacle, as it predominantly moves in a straight line and has only six virtual sensors.

The sensor value for each virtual sensor of the dynamic obstacle is determined by the following calculation:

$$\text{sensor value} = \text{detection} * \text{sensor numbering} \quad (13)$$

The error is defined by the following formulation:

$$\text{error} = \frac{\text{total sensor value}}{\text{total detection}} - 3.5 \quad (14)$$

The *total sensor value* represents the sum of all sensor values, while the *total detection* corresponds to the sum of detections across all sensors.

The PID controller parameters for the dynamic obstacle are provided in Table 4. The calculation of P , I , D , PID value, and speed signal remains the same as that of the AMR.

Table 4

Parameters of PID controller for dynamic obstacle.

Variable	Value
K_p	20
K_i	0
K_d	3
Motor initial	70

4.5. Mathematical modelling for dynamic obstacle avoidance

Fig. 7 depicts the motion model, wherein R represents the AMR, and O signifies the obstacle encountered. The obstacle's radius r_i is expanded by the summation of its original radius and the AMR's radius. The line MR represents the tangent line to the obstacle, and the minimum safe angle u denotes the angle between this tangent line and RO . The velocities of the AMR and obstacle are represented as v_R and v_O respectively. The relative velocity of the AMR with respect to the obstacle is designated as Δv and γ denotes the angle between the relative velocity vector Δv and the line RO .

Fig. 8 introduces a global coordination system, where angles α and β are measured positively from the x -axis to the directions of v_R and v_O respectively, in an anti-clockwise manner. A local coordination system is established for Δv with point R serving as the origin, and the x -axis aligned with the direction from R to O . Consequently, Δv can be decomposed into two components using the local coordination system: v_r along the x -axis and v_θ along the y -axis. The ensuing equations are formulated based on the vector relationships depicted in Fig. 8 [55]:

$$\begin{aligned} v_r &= v_a \cos(\alpha - \theta) - v_b \cos(\beta - \theta) \\ v_\theta &= v_a \sin(\alpha - \theta) - v_b \sin(\beta - \theta) \end{aligned} \quad (15)$$

To ensure collision avoidance with an obstacle, it is imperative that the parameter γ be greater than u as illustrated in Fig. 7. Consequently, the angle $\Delta\gamma$ must comply with several mathematical relationships. By referring to both Figs. 7 and 8, the ensuing formula can be derived [55]:

$$\tan \gamma = \frac{v_\theta}{v_r} = \frac{v_a \sin(\alpha - \theta) - v_b \sin(\beta - \theta)}{v_a \cos(\alpha - \theta) - v_b \cos(\beta - \theta)} \quad (16)$$

Under the assumption of $f(v_a, \alpha, v_b, \beta) = \tan \gamma$, the first derivative of the function can be expressed as derived in Ref. [55]:

$$\gamma = \tan^{-1} f(v_a, \alpha, v_b, \beta)$$

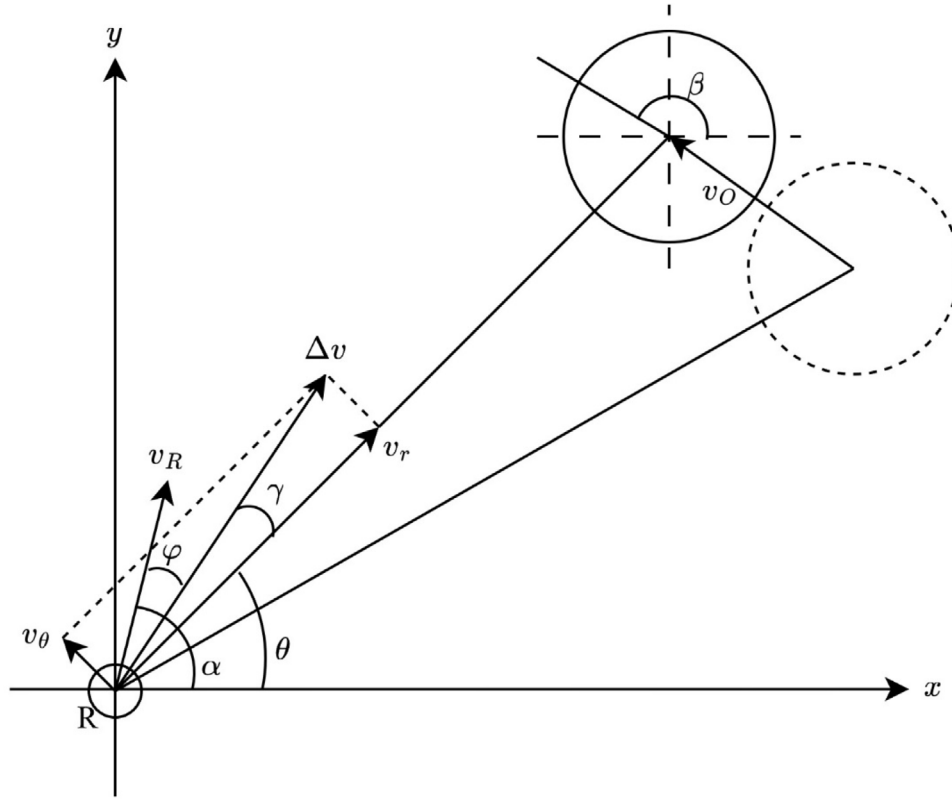


Fig. 8. The motion model decomposition [55].

$$d\gamma = \frac{1}{1+f^2} df \quad (17)$$

$$\frac{1}{1+f^2} = \frac{K^2}{v_a^2 + v_b^2 - 2v_a v_b \cos(\alpha - \beta)}$$

$$K^2 = [v_b \cos(\beta - \theta) - v_a \cos(\alpha - \theta)]$$

$$df = \frac{\partial f}{\partial v_a} dv_a + \frac{\partial f}{\partial \alpha} d\alpha + \frac{\partial f}{\partial v_b} dv_b + \frac{\partial f}{\partial \beta} d\beta \quad (18)$$

Given the negligible variation of direction angle and velocity over a short time interval, both are approximated as zero. Consequently, Eq. (18) can be expressed accordingly in Ref. [55]:

$$df = \frac{\partial f}{\partial v_a} dv_a + \frac{\partial f}{\partial \alpha} d\alpha$$

$$\frac{\partial f}{\partial v_a} = \frac{-v_b \sin(\alpha - \beta)}{K^2}$$

$$\frac{\partial f}{\partial \alpha} = \frac{v_a [v_a - v_b \cos(\alpha - \beta)]}{K^2}$$

Derived from the equations presented above, the following equations are obtained [55]:

$$\begin{aligned} d\gamma &= \frac{-v_b \sin(\alpha - \beta) dv_a + v_a [v_a - v_b \cos(\alpha - \beta)] d\alpha}{v_a^2 + v_b^2 - 2v_a v_b \cos(\alpha - \beta)} \\ \Delta\gamma &= \frac{-v_b \sin(\alpha - \beta) \Delta v_a + v_a [v_a - v_b \cos(\alpha - \beta)] \Delta\alpha}{v_a^2 + v_b^2 - 2v_a v_b \cos(\alpha - \beta)} \end{aligned} \quad (19)$$

The mathematical relationship involving v_a , v_b and Δv in Eq. (19) can be simplified with the aid of Fig. 9. By referring to Fig. 9, we can formulate a system of equations to describe the interdependence between these variables [55]:

$$v_b \sin(\alpha - \beta) = \Delta v \sin \varphi$$

$$v_a - v_b \cos(\alpha - \beta) = -\Delta v \cos \varphi$$

$$v_a^2 + v_b^2 - 2v_a v_b \cos(\alpha - \beta) = \Delta v^2 \quad (20)$$

Consequently, Eq. (19) undergoes simplification, yielding the following expression [55]:

$$\Delta\gamma = \frac{1}{\Delta v} (-\Delta v_a \sin \varphi - v_a \Delta\alpha \cos \varphi) \quad (21)$$

4.6. Map with static obstacles

Map 2 from the Ref. [50] is chosen for the purpose of evaluating and comparing the performance of DMMTQL with QL in both simulation and real-world environments.

4.7. Map with dynamic obstacles

To assess the performance of the algorithms in both simulation and real-world scenarios, Map 11 from [51] is chosen and implemented. The setup is similar to the map with static obstacles, but in this case, two Tiny:bit smart robot cars are utilized as dynamic obstacles.

5. Results and discussion

5.1. Environment with static obstacle

5.1.1. Path planned

The paths generated by QL and DMMTQL algorithms in the real-world environment with static obstacles are summarized in Table 5. The result displays the position of the AMR at i th iteration. The paths planned by the algorithms are represented by black lines, as indicated in Table 5. Subsequently, the actual paths travelled by the AMR in subsequent iterations are denoted by red lines.

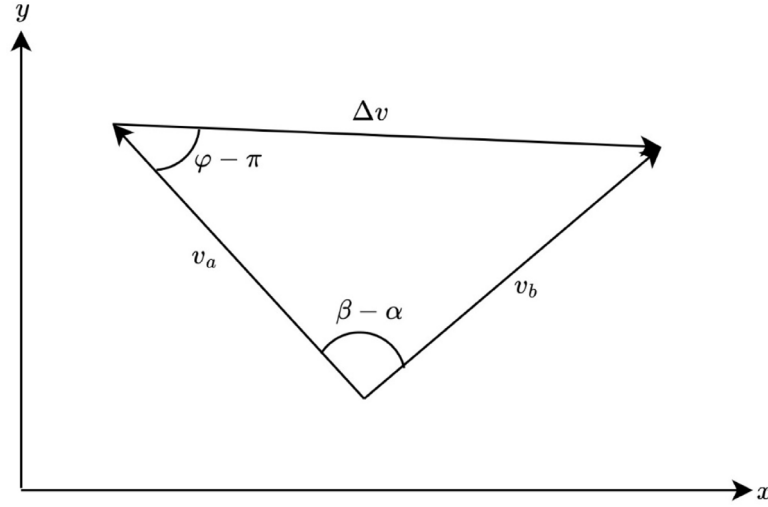


Fig. 9. Velocity relation [55].

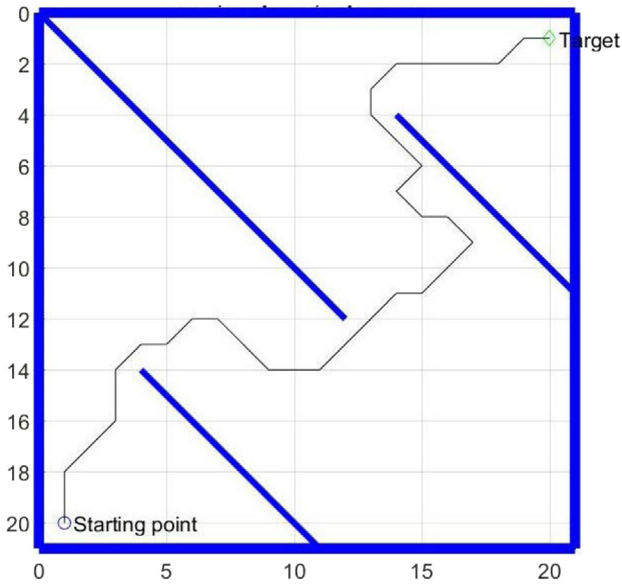


Fig. 10. The path planned by QL in simulated environment with static obstacle [50].

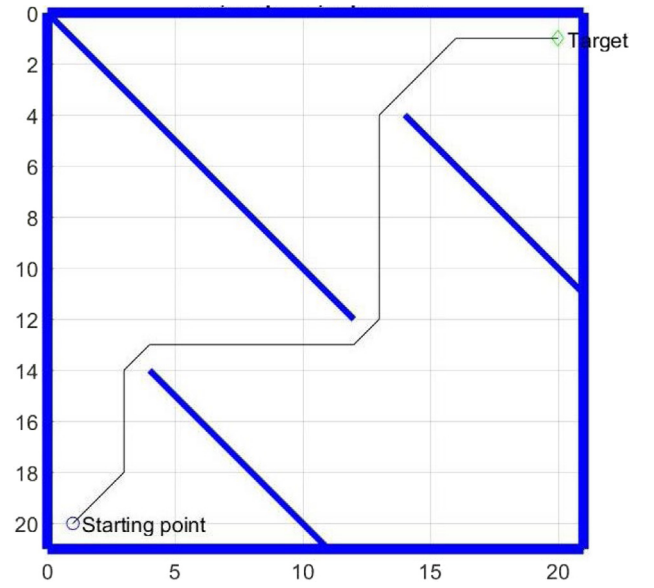


Fig. 11. The path planned by DMMTQL in simulated environment with static obstacle [50].

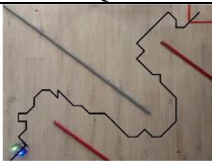
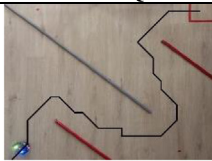
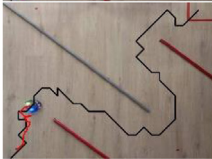
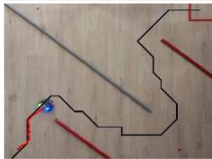
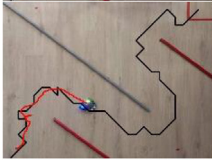
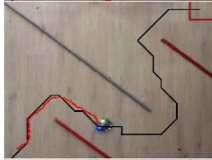
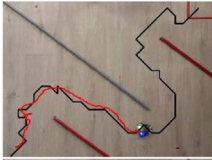
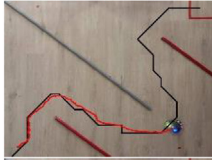
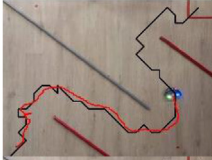
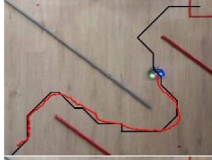
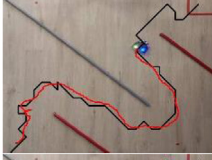
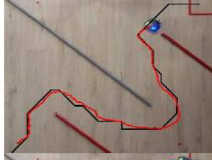
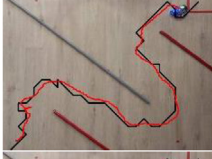
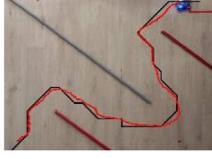
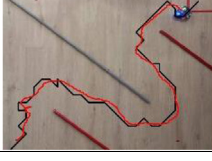

In the initial iteration, the contrasting characteristics of the paths generated by QL and DMMTQL algorithms are evident. The path planned by QL exhibits a curvy trajectory, particularly at the beginning, whereas the path planned by DMMTQL appears significantly straighter. By the 50th iteration, the limitations of QL become apparent, as the AMR struggles to follow the planned path. This is evident from the considerable gaps between the planned path and the actual path of the AMR, as well as the back-and-forth movements observed. Conversely, DMMTQL demonstrates better performance without encountering similar difficulties. By the 100th iteration, QL continues to lag behind due to its previous curvy path. Both algorithms deviate from the planned path, resulting in significant gaps between the planned path and the actual path followed by the AMR. At the 150th iteration, DMMTQL successfully navigates around the corner of the second obstacle, while QL still struggles to catch up. By the 200th iteration, both QL and DMMTQL complete the corner of the second obstacle; however, a noticeable gap between the planned path and the actual path of the AMR indicates overshooting during the cornering manoeuvre. After overshooting, DMMTQL exhibits an oscillatory behaviour by swinging back in the opposite direction. At the 250th iteration, DMMTQL encounters the corner of the final obstacle. Following the previous

oscillation, DMMTQL closely adheres to the planned path. QL, although catching up from behind, also exhibits oscillations after navigating the corner of the second obstacle. By the 300th iteration, DMMTQL completes its run, finishing the corner of the last obstacle with a significant deviation from the planned path. QL also successfully passes the corner of the last obstacle, albeit with minor deviations. At the 350th iteration, QL reaches the target area; however, oscillations are observed near the target area due to a 90-degree turn in the planned path just before reaching the target. Nevertheless, both algorithms involved in this experiment successfully complete the path by arriving at the target area.

Fig. 10 presents the path planned by QL in a simulated environment with static obstacles. Upon comparing the simulated path planned by QL with the actual path travelled by QL in the real world, it can be observed that both paths exhibit sharp 90-degree turns and a curvy trajectory. However, the path travelled by QL in the real world demonstrates a smoother profile, attributed to the AMR's adherence to the path using the PID controller. Additionally, the real-world path includes a back-and-forth section in the front part.

Fig. 11 displays the path planned by DMMTQL in a simulated environment with static obstacles. A comparison between the simulated

Table 5
The path planned by QL and DMMTQL in real environment with static obstacle. Black line is path planned by algorithms while red line is real paths travelled by AMR.

Iteration	Path planned	
	QL	DMMTQL
1		
50		
100		
150		
200		
250		
300		
350		

path planned by DMMTQL and the actual path travelled by DMMTQL in the real world reveals notable differences. The simulated path exhibits a smoother and straighter trajectory, in contrast to the real-world path, which involves more turns and a sharp 90-degree turn just before reaching the target location.

5.1.2. Time taken

The time required by QL and DMMTQL, along with a comparison of DMMTQL in a real-world environment with static obstacles, is summarized in Table 6. When examining the time taken, the differences between DMMTQL and QL are minimal. DMMTQL demonstrates a superior performance, exhibiting a 3.53% improvement over QL. The increased time taken by QL can be attributed to its significantly curvier path compared to DMMTQL. Despite the higher curviness of QL’s path, the time taken by QL exceeds that of DMMTQL by only 3.53%. This

suggests that while a curvier path may slightly prolong the completion time in the real world, the difference is relatively small.

The average time required by QL and DMMTQL, along with a comparison in a simulated environment with static obstacles, is summarized in Table 7. In the simulated environment, DMMTQL demonstrates a significantly shorter time taken compared to QL, with a difference of 91.55%. It is noteworthy that the time taken in the simulated and real-world environments differs substantially. QL required 146.02 s and 0.9295 s to complete the real-world and simulated environments, respectively, while DMMTQL required 140.88 s and 0.785 s for the same environments. This significant disparity in time taken between the simulated and real-world environments arises from the fact that these experiments measure different aspects. In the simulated environment, the time taken primarily reflects the efficiency and functioning of the algorithm, without considering the physical constraints of the AMR. On

Table 6

The time taken for QL and DMMTQL and comparison with DMMTQL in real world environment with static obstacles.

Algorithm	Time taken (s)	Improvement (%)
QL	146.02	3.53
DMMTQL	140.88	0.00

Table 7

The average time taken for QL and DMMTQL and comparison with DMMTQL in simulated environment with static obstacles.

Algorithm	Time taken (s)	Improvement (%)
QL	0.9295	91.55
DMMTQL	0.0785	0.00

Table 8

The distance travelled for QL and DMMTQL and comparison with DMMTQL in real world environment with static obstacles.

Algorithm	Distance travelled (unit)	Improvement (%)
QL	1326	8.41
DMMTQL	1214	0.00

Table 9

The average distance travelled for QL and DMMTQL and comparison with DMMTQL in simulated environment with static obstacles.

Algorithm	Distance travelled (unit)	Improvement (%)
QL	48.80	28.39
DMMTQL	34.94	0.00

the other hand, the real-world experiment measures the time taken for the AMR to traverse the final path planned by the algorithms, taking into account factors such as distance and path smoothness.

5.1.3. Distance travelled

The distance travelled by QL and DMMTQL, along with a comparison in the real-world environment with static obstacles, is presented in Table 8. DMMTQL achieved a shorter distance travelled than QL, with a difference of 8.41%. This difference is relatively substantial compared to the difference in time taken between DMMTQL and QL in Table 6, which is 3.53%. The longer distance travelled by QL can be attributed to the curviness of its path, as observed in Table 5. This indicates that a curvy path has a lesser impact on the time taken to complete the path but a higher impact on the overall distance travelled.

The average distance travelled by QL and DMMTQL, along with a comparison in the simulated environment with static obstacles, is presented in Table 9. In the simulated environment, DMMTQL achieved a shorter distance travelled than QL, with a difference of 28.39%. When comparing the distance travelled in the real-world and simulation environments, the improvement of DMMTQL is lower in the real-world experiment, where the improvement is 8.41%, while in the simulation, it is 28.39%. This can be attributed to the curviness of the path planned by DMMTQL in the real-world experiment compared to the simulation. Furthermore, the implementation of the PID controller in the AMR helped to smoothen out the curvy path, significantly reducing the difference between QL and DMMTQL in the real-world experiment.

5.1.4. Success rate

The success rate of QL and DMMTQL in the real-world environment with static obstacles is presented in Table 10. Both algorithms achieved a success rate of one, indicating that the paths planned by them are executable by the AMR and feasible for real-world application.

The success rate of QL and DMMTQL in the simulated environment with static obstacles is presented in Table 11. Both algorithms achieved a success rate of one, demonstrating their capability to successfully navigate the simulated environment with static obstacles.

Table 10

Success rate for QL and DMMTQL in real world environment with static obstacles.

Algorithm	Success rate
QL	1
DMMTQL	1

Table 11

Success rate for QL and DMMTQL in simulated environment with static obstacles.

Algorithm	Success rate
QL	1
DMMTQL	1

5.2. Environment with dynamic obstacle

5.2.1. Path planned

The paths planned by QL and DOQL in the real-world environment with dynamic obstacles are depicted in Table 12, showing the position of AMR at i th iteration. The path planned by the algorithms is represented by a black line, while the actual path travelled by the AMR is indicated by a red line.

At the initial iteration, the starting positions of the AMR and dynamic obstacles are observed, along with the paths planned by QL and DOQL. Both QL and DOQL paths exhibit similarities, including sharp 90-degree turns and a curvy section. At iteration 50, both algorithms closely follow their planned paths. QL shows slight deviations, while DOQL has larger deviations when executing the right turn. By iteration 100, QL follows its planned path with moderate deviation. However, QL does not encounter the first dynamic obstacle since its planned path avoids it. In contrast, DOQL closely adheres to the planned path and demonstrates collision awareness by executing a quick right turn to avoid a head-on collision with the first dynamic obstacle. At iteration 150, QL deviates from its planned path by navigating towards the east instead of the northeast direction. This diversion is a result of QL detecting the presence of the dynamic obstacle and manoeuvring away from it. On the other hand, DOQL performs a swift U-turn to return to its planned path after avoiding the first dynamic obstacle. By iteration 200, QL continues its journey without adhering to the planned path, choosing to travel alongside the second static obstacle. DOQL, on the other hand, roughly follows its planned path but with a significant gap between the planned path and the actual path travelled by the AMR.

At iteration 250, QL avoids the second dynamic obstacle by gradually moving in a northward direction. In contrast, DOQL's planned path already avoids the second dynamic obstacle, thus preventing any interaction. However, DOQL displays a large deviation from the planned path. By iteration 300, QL approaches the target area by following the path planned after avoiding the second dynamic obstacle. However, there is a significant deviation between the planned path and the actual path travelled by the AMR. On the other hand, DOQL reaches the target area and successfully completes the experiment but with a considerable deviation from the planned path. At iteration 350, QL arrives at the target area, successfully completing the experiment.

Fig. 12 depicts the simulated environment with dynamic obstacles, showcasing the paths planned by QL and DOQL. In this simulated setting, QL and DOQL exhibit minimal differences. Both algorithms traverse curvy paths with several sharp 90-degree turns, as depicted in Fig. 12(e). Furthermore, both algorithms demonstrate their capability to avoid dynamic obstacles, as demonstrated in Fig. 12(c).

In the real-world environment with dynamic obstacles, DOQL requires fewer iterations compared to QL to complete the designated path. Conversely, in the simulated environment, both algorithms reach path completion at the same iteration (iteration 32).

Table 12

The path planned by QL and DOQL in real environment with dynamic obstacle. Black line is path planned by algorithms while red line is real paths travelled by AMR.

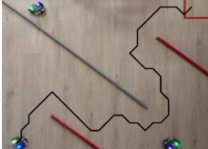
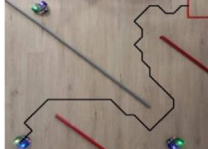
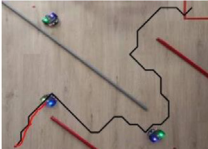
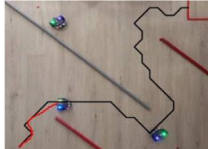
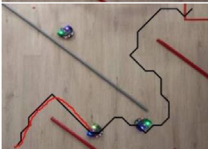
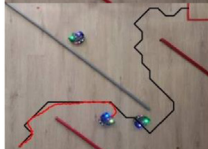
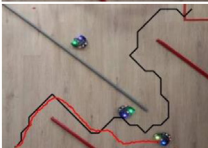

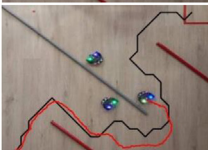

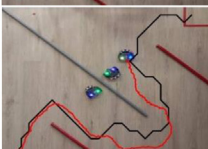

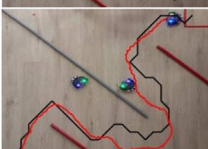

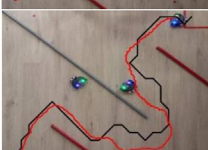

Iteration	Path planned	
	QL	DOQL
1		
50		
100		
150		
200		
250		
300		
350		

Table 13

Time taken for QL and DOQL and their comparison with DOQL in real world environment with dynamic obstacles.

Algorithm	Time taken (s)	Improvement (%)
QL	307.30	8.83
DOQL	280.16	0.00

5.2.2. Time taken

Table 13 presents the time taken by QL and DOQL, along with a comparison of DOQL's performance in a real-world environment with dynamic obstacles. DOQL exhibits an 8.83% improvement in terms of time taken compared to QL. The relatively slower performance of QL can be attributed to its frequent large-angle turns required to avoid dynamic obstacles. QL encounters two instances of such turns for both

dynamic obstacles, whereas DOQL encounters only one such turn for the first dynamic obstacle. The substantial time consumed during these large-angle turns is due to the necessary slowdown of the AMR before executing the turn.

Table 14 presents the average time taken by QL and DOQL, along with a comparison of DOQL's performance in a simulated environment with dynamic obstacles. In the simulated environment, DOQL demonstrates a significantly shorter time taken compared to QL, with a difference of 99.00%. The disparity in time taken between the simulated and real-world environments is substantial. QL requires 307.30 s and 0.856 s to complete the real-world and simulated environments, respectively, while DOQL requires 280.16 s and 0.0008 s. These significant differences, again, can be attributed to the experiments conducted in the real-world and simulation environments measure different aspects of performance, as observed in the obtained results for static obstacles.

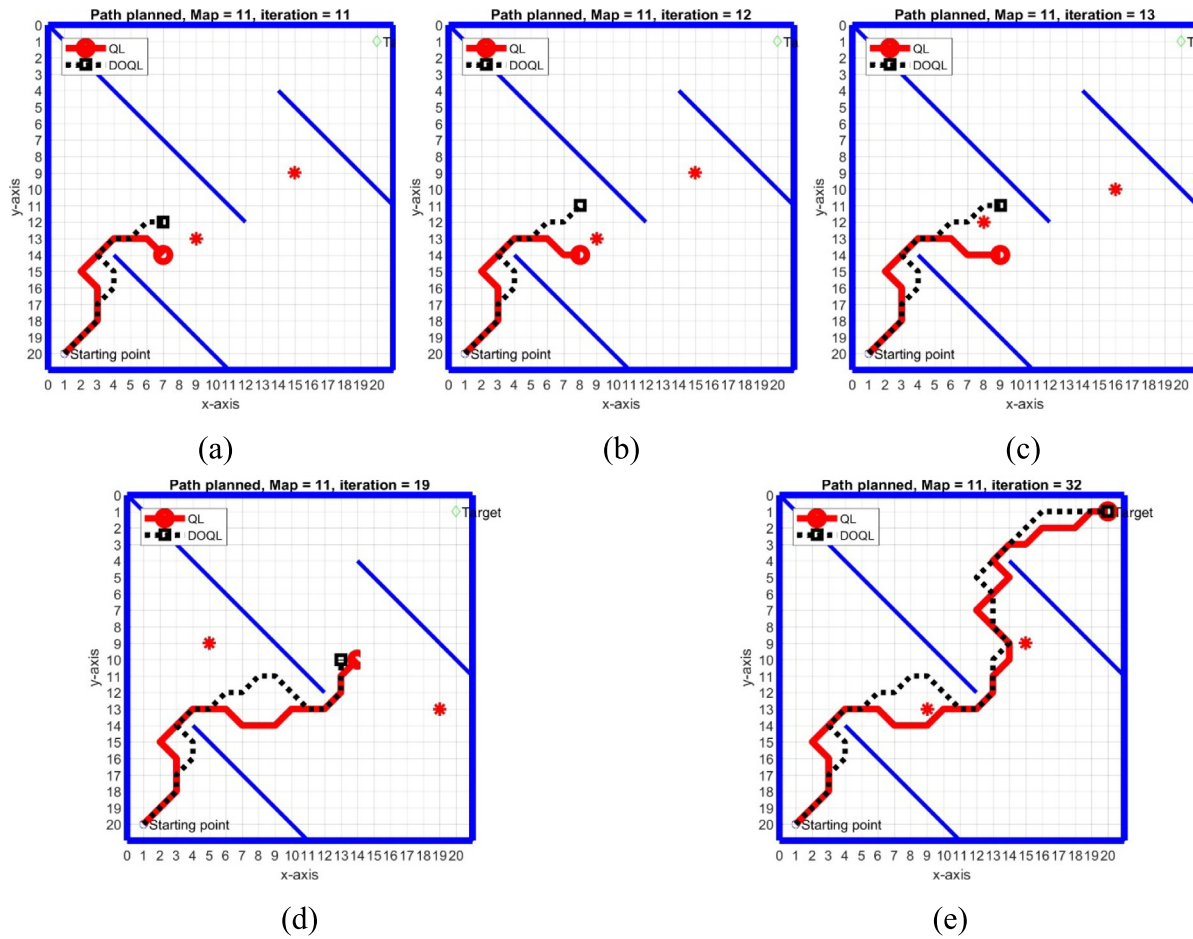


Fig. 12. The path planned by QL and DOQL in simulated environment with dynamic obstacle at: (a) iteration 11; (b) iteration 12; (c) iteration 13; (d) iteration 19; and (e) iteration 32.

Table 14

Average time taken for QL and DOQL and their comparison with DOQL in simulated environment with dynamic obstacles.

Algorithm	Time taken (s)	Improvement (%)
QL	0.0856	99.00
DOQL	0.0008	0.00

5.2.3. Distance travelled

Table 15 presents the distance travelled by QL and DOQL, along with a comparison of DOQL's performance in a real-world environment with dynamic obstacles. When comparing DOQL with QL, DOQL exhibits a slightly longer distance travelled, with a marginal difference of 1.00%. Both algorithms follow quite similar paths, making a significant swing around the corner of the second static obstacle. However, DOQL's path becomes curvier when approaching the third static obstacle, resulting in a slightly longer distance travelled compared to QL. Although the increase in distance is minimal, it is attributed to the curvy section of DOQL's path.

Table 16 presents the average distance travelled by QL and DOQL, along with a comparison of DOQL's performance in a simulated environment with dynamic obstacles. Comparing the results from Tables 15 and 16, it can be observed that the improvements achieved by DOQL are nearly identical. In both the real-world and simulated environments, DOQL demonstrates similar improvements, with -0.72% in the real-world environment and -1.00% in the simulated environment. These consistent results highlight the effectiveness of DOQL in optimizing the distance travelled, regardless of the environment.

Table 15

Distance travelled for QL and DOQL and their comparison with DOQL in real world environment with dynamic obstacles.

Algorithm	Distance travelled (unit)	Improvement (%)
QL	1308	-0.72
DOQL	1318	0.00

Table 16

Average distance travelled for QL and DOQL and their comparison with DOQL in simulated environment with dynamic obstacles.

Algorithm	Distance travelled (unit)	Improvement (%)
QL	39.75	-1.00
DOQL	40.32	0.00

5.2.4. Success rate

As shown in Table 17, both algorithms demonstrate a success rate of one, indicating their ability to safely navigate towards the target in the presence of dynamic obstacles. This underscores the effectiveness of both algorithms in successfully avoiding dynamic obstacles and highlights their capability in real-world scenarios.

Table 18 displays the success rate of QL and DOQL in a simulated environment with dynamic obstacles. In this simulated environment, DOQL demonstrates a higher success rate compared to QL. This indicates that DOQL exhibits superior capability in avoiding dynamic obstacles and successfully completing the designated path in simulation. It should be noted that the success rate observed in the real-world environment, with only one run conducted, may not accurately represent the overall and comprehensive success rate of the algorithms.

Table 17

Success rate for QL and DOQL in real world environment with dynamic obstacles.

Algorithm	Success rate
QL	1
DOQL	1

Table 18

Success rate for QL and DOQL in simulated environment with dynamic obstacles.

Algorithm	Success rate
QL	0.87
DOQL	1.00

6. Conclusion

To validate the feasibility of the QL, DMMTQL, and DOQL algorithms, real-world experiments were conducted. The performance of these algorithms was compared in terms of path planning, time taken, distance travelled, and success rate, in both environments with static obstacles and environments with dynamic obstacles. The results obtained from the real-world experiments revealed that DMMTQL and DOQL outperformed QL in all evaluation criteria for both static and dynamic obstacle environments, except for the distance travelled in the dynamic obstacle environment. Comparing the real-world results with the simulation results, significant disparities were observed in the improvements achieved by DMMTQL and DOQL in terms of time taken for both static and dynamic obstacles, as well as the distance travelled for static obstacles. However, it is noteworthy that the differences between the real-world and simulation results were minimal when considering the improvements made by DMMTQL and DOQL.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article

Acknowledgements

The authors would like to express the deepest appreciation to the Ministry of Higher Education Malaysia, for funding this project through the Fundamental Research Grant Scheme (FRGS/1/2018/ICT02/UTHM/02/2 Vot K070). Additional support from Universiti Tun Hussein Onn Malaysia (UTHM) in the form of GPPS Vot H034 is also gratefully acknowledged.

References

- [1] X. Du, K.K. Tan, Comprehensive and practical vision system for self-driving vehicle lane-level localization, *IEEE Trans. Image Process.* 25 (2016) 2075–2088.
- [2] Z. Ouyang, J. Niu, Y. Liu, M. Guizani, Deep CNN-based real-time traffic light detector for self-driving vehicles, *IEEE Trans. Mob. Comput.* 19 (2019) 300–313.
- [3] I. Draganjac, D. Miklić, Z. Kovačić, G. Vasiljević, S. Bogdan, Decentralized control of multi-AGV systems in autonomous warehousing applications, *IEEE Trans. Autom. Sci. Eng.* 13 (2016) 1433–1447.
- [4] E. Cardarelli, V. Digani, L. Sabatini, C. Secchi, C. Fantuzzi, Cooperative cloud robotics architecture for the coordination of multi-AGV systems in industrial warehouses, *Mechatronics* 45 (2017) 1–13.
- [5] V. Prabhakaran, M.R. Elara, T. Pathmakumar, S. Nansai, Floor cleaning robot with reconfigurable mechanism, *Autom. Constr.* 91 (2018) 155–165.
- [6] A.V. Le, P.T. Kyaw, R.E. Mohan, S.H.M. Swe, A. Rajendran, K. Boopathi, et al., Autonomous floor and staircase cleaning framework by reconfigurable sTetro robot with perception sensors, *J. Intell. Robot. Syst.* 101 (2021) 1–19.

- [7] F.H. Ajeil, I.K. Ibraheem, M.A. Sahib, A.J. Humaidi, Multi-objective path planning of an autonomous mobile robot using hybrid PSO-MFB optimization algorithm, *Appl. Soft Comput.* 89 (2020) 106076.
- [8] R.K. Mandava, M. Katla, P.R. Vundavilli, Application of hybrid fast marching method to determine the real-time path for the biped robot, *Intell. Serv. Robot.* 12 (2019) 125–136.
- [9] R.K. Mandava, K. Mrudul, P.R. Vundavilli, Dynamic motion planning algorithm for a biped robot using fast marching method hybridized with regression search, *Acta Polytech. Hung.* 16 (2019) 189–208.
- [10] S. Pradhan, R.K. Mandava, P.R. Vundavilli, Development of path planning algorithm for biped robot using combined multi-point RRT and visibility graph, *Int. J. Inf. Technol.* 13 (2021) 1513–1519.
- [11] D. Parappagoudar, R.K. Mandava, P.R. Vundavilli, B. Betadur, An efficient path planning algorithm for the biped robot in a static environment using fast sweeping method, *Proc. Inst. Mech. Eng. C* 236 (2022) 7417–7425.
- [12] J. Kim, D. Mavris, Flight data clustering for offline evaluation of real-time trajectory optimization framework, *Decis. Anal. J.* 7 (2023) 100221.
- [13] M. Apurba, R. Arindam, M. Samir, M. Sukumar, N. Izabela Ewa, A multi-parent genetic algorithm for solving longitude-latitude-based 4D traveling salesman problems under uncertainty, *Decis. Anal. J.* (2023) 100287.
- [14] T.T. Mac, C. Copot, D.T. Tran, R. De Keyser, Heuristic approaches in robot path planning: A survey, *Robot. Auton. Syst.* 86 (2016) 13–28.
- [15] B. Patle, A. Pandey, D. Parhi, A. Jagadeesh, A review: On path planning strategies for navigation of mobile robot, *Def. Technol.* 15 (2019) 582–606.
- [16] M.N. Ab Wahab, S. Nefti-Meziani, A. Atyabi, A comparative review on mobile robot path planning: Classical or meta-heuristic methods? *Annu. Rev. Control* (2020).
- [17] C. Chen, X.-Q. Chen, F. Ma, X.-J. Zeng, J. Wang, A knowledge-free path planning approach for smart ships based on reinforcement learning, *Ocean Eng.* 189 (2019) 106299.
- [18] L. Jiang, H. Huang, Z. Ding, Path planning for intelligent robots based on deep Q-learning with experience replay and heuristic knowledge, *IEEE/CAA J. Autom. Sin.* (2019).
- [19] T.T. Mac, C. Copot, D.T. Tran, R. De Keyser, A hierarchical global path planning approach for mobile robots based on multi-objective particle swarm optimization, *Appl. Soft Comput.* 59 (2017) 68–76.
- [20] D.C. Guastella, L. Cantelli, G. Giammello, C.D. Melita, G. Spatino, G. Muscato, Complete coverage path planning for aerial vehicle flocks deployed in outdoor environments, *Comput. Electr. Eng.* 75 (2019) 189–201.
- [21] S.W. Cho, H.J. Park, H. Lee, D.H. Shim, S.-Y. Kim, Coverage path planning for multiple unmanned aerial vehicles in maritime search and rescue operations, *Comput. Ind. Eng.* (2021) 107612.
- [22] G. Chen, N. Luo, D. Liu, Z. Zhao, C. Liang, Path planning for manipulators based on an improved probabilistic roadmap method, *Robot. Comput.-Integr. Manuf.* 72 (2021) 102196.
- [23] B.B.K. Ayawli, A.Y. Appiah, I.K. Nti, F. Kyeremeh, E.I. Ayawli, Path planning for mobile robots using morphological dilation voronoi diagram roadmap algorithm, *Sci. Afr.* 12 (2021) e00745.
- [24] H. Niu, Y. Lu, A. Savvaris, A. Tsourdos, An energy-efficient path planning algorithm for unmanned surface vehicles, *Ocean Eng.* 161 (2018) 308–321.
- [25] U. Orozco-Rosas, O. Montiel, R. Sepúlveda, Mobile robot path planning using membrane evolutionary artificial potential field, *Appl. Soft Comput.* 77 (2019) 236–251.
- [26] S.M.H. Rostami, A.K. Sangaiah, J. Wang, X. Liu, Obstacle avoidance of mobile robots using modified artificial potential field algorithm, *EURASIP J. Wireless Commun. Networking* 2019 (2019) 1–19.
- [27] Y. Rasehipour, A. Khajepour, S.-K. Chen, B. Litkouhi, A potential field-based model predictive path-planning controller for autonomous road vehicles, *IEEE Trans. Intell. Transp. Syst.* 18 (2016) 1255–1267.
- [28] I. Sung, B. Choi, P. Nielsen, On the training of a neural network for online path planning with offline path planning algorithms, *Int. J. Inf. Manage.* 57 (2021) 102142.
- [29] K. Wu, M.A. Esfahani, S. Yuan, H. Wang, TDPP-Net: Achieving three-dimensional path planning via a deep neural network architecture, *Neurocomputing* 357 (2019) 151–162.
- [30] Y. Peng, S.-W. Li, Z.-Z. Hu, A self-learning dynamic path planning method for evacuation in large public buildings based on neural networks, *Neurocomputing* 365 (2019) 71–85.
- [31] A. Pandey, D.R. Parhi, Optimum path planning of mobile robot in unknown static and dynamic environments using fuzzy-wind driven optimization algorithm, *Def. Technol.* 13 (2017) 47–58.
- [32] M. Fakoor, A. Kosari, M. Jafarzadeh, Humanoid robot path planning with fuzzy Markov decision processes, *J. Appl. Res. Technol.* 14 (2016) 300–310.
- [33] B. Sun, D. Zhu, S.X. Yang, An optimized fuzzy control algorithm for three-dimensional AUV path planning, *Int. J. Fuzzy Syst.* 20 (2018) 597–610.
- [34] S. Maji, S. Maity, D. Giri, O. Castillo, M. Maiti, A multi-path delivery system with random refusal against online booking using type-2 fuzzy logic-based fireworks algorithm, *Decis. Anal. J.* 6 (2023) 100151.

- [35] A. Bakdi, A. Hentout, H. Boutami, A. Maoudj, O. Hachour, B. Bouzouia, Optimal path planning and execution for mobile robots using genetic algorithm and adaptive fuzzy-logic control, *Robot. Auton. Syst.* 89 (2017) 95–109.
- [36] A. Tharwat, M. Elhoseny, A.E. Hassanien, T. Gabel, A. Kumar, Intelligent Bézier curve-based path planning model using Chaotic Particle Swarm Optimization algorithm, *Cluster Comput.* 22 (2019) 4745–4766.
- [37] J. Liu, J. Yang, H. Liu, X. Tian, M. Gao, An improved ant colony algorithm for robot path planning, *Soft Comput.* 21 (2017) 5829–5839.
- [38] C. Yan, X. Xiang, C. Wang, Towards real-time path planning through deep reinforcement learning for a UAV in dynamic environments, *J. Intell. Robot. Syst.* 98 (2020) 297–309.
- [39] B. Wang, Z. Liu, Q. Li, A. Prorok, Mobile robot path planning in dynamic environments through globally guided reinforcement learning, *IEEE Robot. Autom. Lett.* 5 (2020) 6932–6939.
- [40] X. Zhao, S. Ding, Y. An, W. Jia, Asynchronous reinforcement learning algorithms for solving discrete space path planning problems, *Appl. Intell.* 48 (2018) 4889–4904.
- [41] Z. Zeng, K. Sammut, L. Lian, F. He, A. Lammas, Y. Tang, A comparison of optimization techniques for AUV path planning in environments with ocean currents, *Robot. Auton. Syst.* 82 (2016) 61–72.
- [42] P.K. Das, H.S. Behera, B.K. Panigrahi, A hybridization of an improved particle swarm optimization and gravitational search algorithm for multi-robot path planning, *Swarm Evol. Comput.* 28 (2016) 14–28.
- [43] C.J.C.H. Watkins, *Learning from Delayed Rewards*, 1989.
- [44] S.H. Klidbary, S.B. Shouraki, S.S. Kourabaslou, Path planning of modular robots on various terrains using Q-learning versus optimization algorithms, *Intell. Serv. Robot.* 10 (2017) 121–136.
- [45] E.S. Low, P. Ong, K.C. Cheah, Solving the optimal path planning of a mobile robot using improved Q-learning, *Robot. Auton. Syst.* 115 (2019) 143–161.
- [46] A. Maoudj, A. Hentout, Optimal path planning approach based on Q-learning algorithm for mobile robots, *Appl. Soft Comput.* 97 (2020) 106796.
- [47] D.L. Cruz, W. Yu, Path planning of multi-agent systems in unknown environment with neural kernel smoothing and reinforcement learning, *Neurocomputing* 233 (2017) 34–42.
- [48] J. Wang, Z. Wu, S. Yan, M. Tan, J. Yu, Real-time path planning and following of a gliding robotic dolphin within a hierarchical framework, *IEEE Trans. Veh. Technol.* 70 (2021) 3243–3255.
- [49] P. Das, H. Behera, B. Panigrahi, Intelligent-based multi-robot path planning inspired by improved classical Q-learning and improved particle swarm optimization with perturbed velocity, *Eng. Sci. Technol. Int. J.* 19 (2016) 651–669.
- [50] E.S. Low, P. Ong, C.Y. Low, R. Omar, Modified Q-learning with distance metric and virtual target on path planning of mobile robot, *Expert Syst. Appl.* 199 (2022) 117191.
- [51] E.S. Low, P. Ong, C.Y. Low, A modified Q-learning path planning approach using distortion concept and optimization in dynamic environment for autonomous mobile robot, *Comput. Ind. Eng.* (2023) 109338.
- [52] C.J. Watkins, P. Dayan, Q-learning, *Mach. Learn.* 8 (1992) 279–292.
- [53] K.M. Ong, P. Ong, C.K. Sia, E.S. Low, Effective moving object tracking using modified flower pollination algorithm for visible image sequences under complicated background, *Appl. Soft Comput.* 83 (2019) 105625.
- [54] X.-S. Yang, Flower pollination algorithm for global optimization, in: *International Conference on Unconventional Computing and Natural Computation*, 2012, pp. 240–249.
- [55] H.-Q. Min, J.-H. Zhu, X.-J. Zheng, Obstacle avoidance with multi-objective optimization by PSO in dynamic environment, in: *2005 International Conference on Machine Learning and Cybernetics*, 2005, pp. 2950–2956.