APPLICATIONS OF
MODELLING AND SIMULATION
http://arqiipubl.com/ams eISSN 2600-8084 VOL 9, 2025, 389-403

A Comparative Analysis of Reinforcement Learning-Based Navigation for Autonomous Mobile Robot

Al-Mahdi Sallam[1], Norhaliza Abdul Wahab[1,*], Muhammad Zakiyullah Romdlony[2], Mohd Saiful Azimi Mahmud[1] and Yeong Che Fai[3]

[1]Department of Control and Mechatronics Engineering, Faculty of Electrical Engineering, Universiti Teknologi Malaysia, 81310 UTM Johor Bahru, Johor, Malaysia
[2]School of Electrical Engineering, Telkom University, Bandung, Indonesia
[3]DF Automation & Robotics Sdn. Bhd., Johor, Malaysia

[*]Corresponding author: aliza@fke.utm.my

Submitted 01 October 2025; Revised 18 November 2025; Accepted 24 November 2025; Available online 26 November 2025.

Abstract: Mobile robots have been widely used in many industries including manufacturing, healthcare and warehouse automation. To ensure efficiency and safety of the robots, it is crucial to design effective control strategies that can adapt to changing environments. This paper presents reinforcement learning (RL) algorithms including Q-learning, Deep Q-Learning (DQN), and Double Deep Q-Learning (DDQN) for autonomous navigation using the Turtle-Bot3 Waffle Pi in a Gazebo-simulated environment. Three progressively complex training stages were designed to evaluate the algorithms: (1) static obstacles with predefined goals, (2) randomized goals with static obstacles, and (3) dynamic obstacles with moving goals. Performance metrics, including success rates, collision avoidance, and reward stability, were analyzed to compare algorithm effectiveness. Key results highlight DDQN's superiority in handling complex navigation tasks. In the most challenging stage, DDQN achieved a 100% success rate and zero collisions, outperforming DQN, which attained an 88% success rate with higher collision rates. Q-learning performed well only in simple environments, as it cannot easily handle continuous state spaces. This study demonstrates the scalability of RL-based navigation systems for autonomous mobile robots. The findings provide a foundation for future advancements in dynamic and real-world robot navigation.

Keywords: Autonomous navigation; Deep Q-Learning; Double Deep Q-Learning; Gazebo simulation; Reinforcement learning; TurtleBot3.

1. INTRODUCTION

Autonomous navigation is one of the most critical challenges in mobile robotics, as robots must safely and efficiently navigate environments filled with obstacles and uncertainties. For autonomous navigation in unfamiliar environments, continuously collecting sensor data is acquired for mobile robots (MRs) to simultaneously create a map of its environment [1], [2]. In industrial automation, the MRs can efficiently navigate factory floors to transport materials, conduct inspections, and collaborate with human workers, enhancing efficiency and safety [3]. In logistics and warehousing, autonomous robots can optimize operations by navigating efficiently to pick up and deliver items, reducing labor costs and increasing throughput [4]. Moreover, robots in healthcare can assist in hospitals by delivering medications, transporting patients, or disinfecting rooms, improving patient care and reducing the workload on healthcare professionals [5]. For exploration and search-and-rescue application, robots equipped with advanced navigation capabilities can operate in hazardous or unknown environments to gather critical information, locate missing persons, or conduct rescue missions [6].

Traditional robot navigation methods such as simultaneous localization and mapping (SLAM) and rule-based path planning depend on pre-existing maps or environmental knowledge face significant challenges in unfamiliar or dynamic settings. The methods demonstrated effectiveness in static environments but face significant limitations in dynamic and unstructured settings [7]. SLAM builds a map of the environment while simultaneously determining the robot's position within it, achieving notable success in structured and static environments. However, it struggles in dynamic settings with moving obstacles or significant environmental changes [8]. Classical rule-based path planning methods, such as Dijkstra's algorithm and rapidly exploring random tree (RRT), depend on deterministic algorithms and predefined conditions to address navigation challenges [9]. In contrast, heuristic approaches offer alternatives to traditional methods by integrating problem-specific knowledge to guide search strategies, with examples including A*, D*, timed elastic band (TEB), and dynamic window approach (DWA) algorithms [10], [11]. The A* and D* algorithms are employed for global path planning, while the TEB and DWA algorithms are used for local path planning and real-time obstacle avoidance [12]. Similarly, path planning algorithms,

including A* and Dijkstra's, require predefined conditions and often struggle with sudden changes, such as moving obstacles or environmental shifts [13], [14]. Although effective in predictable environments, these methods often require extensive manual tuning and lacks adaptability in unstructured or unfamiliar settings [7].

The limitations of the traditional methods have motivated the exploration of intelligent method for an autonomous navigation. In recent years, RL algorithm has emerged as a promising alternative, enabling robots to learn navigation policies by interacting with their environment and receiving rewards for desirable behaviors, such as reaching a goal or avoiding collisions [6], [15]. RL serves as an adaptive and scalable solution for robot navigation, enabling robots to learn navigation strategies through trial and error without the need for extensive prebuilt maps or environmental models [16]. It provides an agent with the ability to learn optimal navigation policies by interacting with the environment and receiving rewards for desirable actions, making it particularly effective in scenarios with continuous environmental changes. Among RL algorithms, Q-learning has gained recognition for its simplicity and effectiveness in discrete state-action spaces. It is a foundational model-free approach where an agent learns by updating a Q-table that maps state-action pairs to expected rewards. While Q-learning is simple and effective in structured environments, it faces limitations in high-dimensional state spaces due to the size and complexity of the Q-table and struggles to generalize in dynamic scenarios with unpredictable state transitions [7], [17]. Its scalability to high-dimensional tasks remains a limitation [18]-[20].

To address these challenges, deep Q-Learning network (DQN) extends Q-learning by employing deep neural networks to approximate Q-values, enabling RL applications in high-dimensional state spaces [21]. The overestimation bias in DQN arises from using the same network for both action selection and value estimation, leading the algorithm to consistently over-estimate Q-values particularly in states with high action-value variance [22]. This bias becomes especially problematic in dynamic environments where the robot must make rapid decisions with incomplete information, potentially causing the agent to select suboptimal actions that appear artificially promising. The DQN architecture includes a neural network component that interacts with the environment, a policy for action selection, and mechanisms for stabilizing training, such as experience replay and the target network. DQN has demonstrated success in dynamic navigation tasks, including obstacle avoidance and goal-oriented navigation [21]. Recent studies [6], [17] have shown that in navigation tasks with moving obstacles, DQN's over-estimation can cause the robot to underestimate collision risks, leading to aggressive path planning that compromises safety. The DQN introduced stability through techniques such as experience replay and a target network. However, it suffers from overestimation biases, leading to suboptimal learning outcomes in certain cases [17]. Double deep Q-Learning networks (DDQN) addressed this limitation by decoupling the selection of actions from their evaluation, specifically using the online network to select actions and the target network to evaluate them, effectively breaking the positive feedback loop that causes over-estimation [22]. This architectural modification has been shown to reduce Q-value over-estimation by 20-30% in robotic navigation tasks [17], leading to more conservative and safer navigation policies. These advancements have positioned the Deep Reinforcement Learning (DRL) as a compelling choice for mobile robot navigation, particularly in environments requiring dynamic obstacle avoidance and goal-directed navigation [17], [23]. Despite these advancements, applying RL-based methods in real-world navigation remains challenging. Issues such as transferring learned policies from simulation to real robots, handling moving obstacles, and ensuring safety in dynamic environments require further exploration. Additionally, bridging the gap between simulation results and real-world scenarios, addressing the high computational demands of advanced RL algorithms, and designing reward functions that balance exploration and exploitation for optimal policy learning are ongoing areas of research [6], [15], [22]. Sensor fusion plays a critical role in enhancing environmental perception and decision-making in MR navigation. By integrating data from multiple sensors, such as LiDAR and odometry, sensor fusion mitigates the limitations of individual sensors and improves robustness [1], [24]. LiDAR provides high-precision environmental measurements, while odometry offers real-time motion updates, enabling more accurate localization and obstacle detection [7]. Simulation environments, such as Gazebo, also play a vital role in the development and evaluation of RL-based navigation systems. Gazebo's ability to simulate complex physical interactions and sensor behaviors provides a controlled platform for training RL agents, with integration into ROS further streamlining the training and evaluation process [7]. Despite these advancements, several challenges remain in RL-based navigation. Ensuring RL agents can adapt to continuous changes in the environment without retraining is a significant challenge [17].

This paper advances engineering knowledge by introducing a three-stage progressively complex training methodology designed to rigorously test the scalability and adaptation limits of value-based DRL algorithms Q-learning, DQN, and DDQN for autonomous navigation. Unlike previous comparative studies that evaluate algorithms in isolated scenarios, this work systematically escalates environmental complexity across three distinct stages: (1) static obstacles with predefined goals, (2) randomized goals with static obstacles, and (3) dynamic obstacles with moving goals. This progressive design uniquely enables the quantitative identification of algorithm-specific breaking points where traditional approaches fail, and advanced methods become necessary.

The key engineering contribution lies in establishing robust design principles for industrial deployment: our experimental framework quantitatively proves that for complex, non-static scenarios, DDQN's bias-correction mechanism is not merely beneficial but a fundamental requirement to achieve the high success rates (100% vs. 88% for DQN) and collision-free performance (0% vs. 8% collision rate) needed for real-world deployment. This finding directly addresses the critical gap between simulated testing and industrial application requirements, providing actionable guidelines for practitioners on when to justify the computational overhead of DDQN over simpler alternatives. Furthermore, the study introduces a comprehensive reward shaping framework tailored for multi-stage training that balances exploration, safety, and efficiency a practical contribution that can be directly applied to other robotic platforms. The scope encompasses algorithm development, including the design and implementation of Q-learning, DQN, and DDQN algorithms for the MR navigation tasks such as obstacle avoidance and goal-directed navigation. The simulation environment utilizes robot operating system (ROS) and the Gazebo simulator to create realistic and customizable virtual environments with progressively complex obstacle configurations. Sensor

integration involves the TurtleBot3 Waffle Pi's LiDAR sensor, enabling the robot to perceive its surroundings, detect obstacles, and make informed navigation decisions. Training and evaluation are conducted across three stages of increasing complexity: static obstacles with predefined goals, randomized goals with static obstacles, and dynamic environments with moving obstacles and randomized goals. Performance is analyzed using metrics such as success rate, collision avoidance, cumulative reward, and time to reach goal.

The paper is organized as follows: Section 2 provides a problem background of MRs robot navigation, examining classical methods and RL-based approaches, and identifies research gaps. Section 3 details the methodology, including the design and implementation of Q-learning, DQN, and DDQN algorithms and the experimental setup in the Gazebo simulation environment. Section 4 presents the simulation and experimental analysis, providing comprehensive discussions on the performance evaluation of the algorithms. Finally, Section 5 provides the conclusions and future directions of this research

2. PROBLEM STATEMENT

The problem statement briefly described the MR navigation, RL including static and dynamic environments and simulation environment for MR navigation.

2.1 Mobile Robot Navigation

MR navigation is a multifaceted challenge that encompasses several key components: perception, localization, path planning, and control [1], [16]. Perception involves sensing the environment through various sensors, such as LiDAR, cameras, and ultrasonic sensors, to detect obstacles and map surroundings [17]. Localization enables the robot to determine its position and orientation within the environment, a critical step for effective navigation [8]. Path planning generates a trajectory for the robot to follow, guiding it from its current location to a desired goal while avoiding obstacles [19]. Finally, control ensures that the robot accurately executes the planned path and responds to environmental changes [20]. Despite advancements in MR navigation, significant challenges remain in adapting to dynamic environments and achieving optimal decision-making. Classical methods like SLAM and path planning struggle with environmental changes and require extensive feature engineering, while the RL offers a promising alternative by learning directly from interactions.

Recent advances in RL-based navigation have explored various approaches to improve adaptability and safety. Studies by Miranda et al. [6] demonstrated that reward shaping significantly impacts generalization capabilities in dynamic environments, while [17] provided a comprehensive review highlighting that most existing DRL navigation systems are evaluated in simplified scenarios that do not adequately test algorithm robustness under realistic conditions. Furthermore, Pak et al. [7] identified that field evaluations often reveal performance degradation of 15-25% compared to simulation results, emphasizing the need for progressively complex training environments that better approximate real-world challenges. These findings motivate our three-stage training approach, which systematically escalates environmental complexity to identify algorithm limitations before real-world deployment.

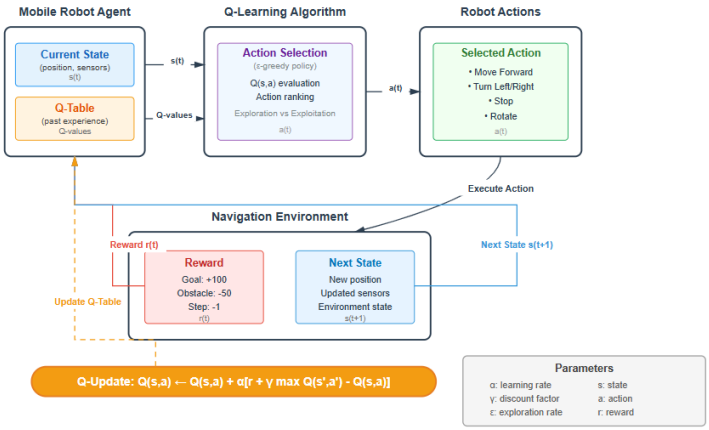
2.2 Reinforcement Learning Environment Representation

Unlike classical approaches that rely on prebuilt maps or predefined rules, RL enables robots to learn navigation behaviors through interaction with their surroundings, making decisions based on real-time feedback [16], [17]. This adaptability positions RL as a key enabler for MR navigation in scenarios where environmental conditions are unpredictable or continuously evolving. This study explores Q-learning as a foundational RL algorithm for structured environment and transitions to its advanced extensions such as DQN and DDQN for unstructured environment. Q-learning is designed to enable agents to learn optimal policies for decision-making through interaction with their environment. Figure 1 shows the Q-learning and deep Q-learning network architectures, respectively. The DQN represents a significant advancement over Q-learning by utilizing deep neural networks to approximate Q-values, enabling the algorithm to handle high-dimensional state spaces effectively, hence allows robots to learn from raw sensor data without requiring manual feature engineering [18]. Meanwhile, DDQN addresses the limitations of DQN by decoupling action selection and value estimation. This separation reduces overestimation bias, resulting in more stable and reliable training [22].

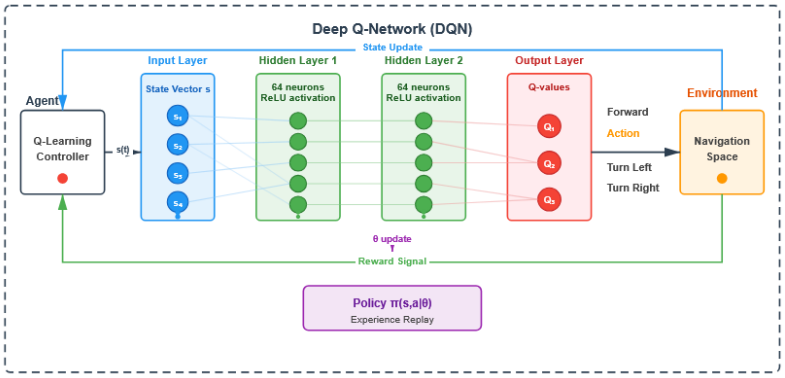
Three essential components need to be considered to ensure effectiveness of the operational strategies and the RL algorithm of the MR navigational framework including state space, action space and reward function.

2.2.1 State Space

The state space in Q-learning represents the robot's interpretation of its environment at any given time. In this study, the state space was designed by discretizing the continuous sensor data obtained from the TurtleBot3 Waffle's LiDAR sensor. The LiDAR provides a 360-degree scan of the environment, delivering precise distance measurements to nearby obstacles [17], [19]. To optimize computational efficiency, only a subset of this data, specifically a 150-degree arc in front of the robot was utilized. This arc was divided into five equal sectors, with the minimum distance measurement within each sector being selected as a feature. Hence, the state space was condensed into a 5-dimensional vector, where each dimension represented the minimum distance to an obstacle in a corresponding sector [24]. Additionally, the robot's orientation relative to its goal was included in the state representation. The angular difference between the robot's heading and the direction to the goal was discretized into a finite number of bins, further refining the state space [14]. The state space is denoted by s.



(a)



(b)

Figure 1. Reinforcement learning network architectures: (a) Q-Learning; (b) Deep Q-Network.

The choice of a 150-degree forward-facing arc represents an engineering trade-off between comprehensive environmental awareness and computational efficiency. This configuration was selected based on pilot studies showing that obstacles outside this frontal region have minimal impact on immediate navigation decisions for goal-directed tasks, while the 5-sector discretization provides sufficient spatial resolution to distinguish between left, center, and right obstacle configurations without overwhelming the Q-table size. Alternative approaches using full 360-degree scans with finer discretization were tested but resulted in state-space explosion (over 10^6 states) that prevented Q-learning convergence within reasonable training times. The 5-dimensional representation maintains the state space at approximately 10^4 states, enabling Q-learning to converge within 300 episodes while still capturing essential obstacle proximity information. This design aligns with recommendations from [9] for balancing state-space complexity with learning efficiency.

2.2.2 Action Space

The action space in Q-learning defines the set of possible movements the robot can execute at each time step. There are mainly three types of actions in the RL-based MR navigation, including discrete moving actions, motor speed commands and continuous velocity commands (angular and linear velocities of the MR). The discrete actions of turning left, turning right and moving forward are implemented. The action space was intentionally kept discrete and simple, with constant linear velocity and fixed angular velocities to facilitate efficient learning and decision-making. The action space is denoted by *a*.

The discrete action space with constant linear velocity (0.1 m/s) and fixed angular velocities (±0.5 rad/s) was designed to facilitate initial learning while maintaining realistic robot dynamics. More granular control with variable velocities was intentionally avoided in this study for two reasons: (1) it exponentially increases the action space dimensionality, slowing convergence for comparative analysis purposes, and (2) it introduces additional complexity that obscures the fundamental differences between Q-learning, DQN, and DDQN that this study aims to highlight. The chosen velocities were validated through preliminary tests to ensure the TurtleBot3 Waffle Pi could execute turns safely without overshooting or instability. This simplified action space represents a standard approach in RL navigation research [14], [19], [20], allowing our results to be directly comparable with existing literature while maintaining practical relevance. Future work will explore continuous action spaces using actor-critic methods to optimize velocity control for real-world deployment.

2.2.3 Reward Function

The reward function is a critical component of the Q-learning algorithm, as it provides feedback to the robot about the desirability of its actions at different states. The reward function is designed to incentivize the robot to reach its goal while avoiding obstacles. This reward function encourages the robot to move forward and increase its distance from obstacles while penalizing collisions and unnecessary turns. The reward for moving away from obstacles is proportional to the change in

distance, providing a stronger incentive for the robot to move towards open spaces. The large negative reward for collisions serves as a strong deterrent, ensuring that the robot prioritizes safety. The reward function is denoted by *r* and is formulated as given in Table 1.

Table 1. Reward function of the Q-learning algorithm.

Condition	Reward	Description
crash is True	-100	The robot collided with an obstacle. This is a significant penalty to discourage collisions.
action is 0 (move forward)	+1.0	The robot successfully moved forward without a collision. This encourages the robot to explore the environment and move towards its goal.
action is 1 or 2 (turn left or right)	-0.7	The robot turned, which is penalized to encourage straight movement when possible.
weighted sum of distance changes	+0.5	The robot is moving away from obstacles, which is a desirable behavior for obstacle avoidance.
weighted sum of distance changes	-0.5	The robot is moving closer to obstacles, which is undesirable and should be avoided.
prev_action and action represent opposite turns (e.g., left followed by right or vice-versa)	-1	This penalizes sudden changes in direction, encouraging smoother trajectories and preventing the robot from getting stuck in oscillating movements.

2.3 Simulation Environment

Simulation environments enable RL agents to interact with realistic virtual worlds, facilitating the development of navigation policies under various scenarios. The system architecture developed in this study integrates the Robot Operating System (ROS) and the Gazebo simulator to facilitate seamless interaction between the simulation environment, the RL algorithms, and the TurtleBot3 Waffle robot. The architecture is designed to accommodate the training and testing of RL algorithms: Q-learning, DQN, and DDQN, with adjustments made for each stage of training. In this simulation, the RL agent implements Q-learning, DQN, and DDQN as ROS nodes, processing data from the robot's 360° LiDAR sensor and odometry to compute state-action pairs. The reward mechanism is designed with custom functions to encourage adaptive navigation behaviors, such as reaching goals and avoiding collisions.

2.3.1 Q-learning Environment Setup

A predefined 5 m × 5 m hexagonal Gazebo world shown in Figure 2 serves as the testbed for Q-learning. The environment consists of five static cubic landmarks and nine smaller cylindrical obstacles placed in semi-random configurations. This setup allows the Q-learning algorithm to navigate around static obstacles and reach randomly generated goals.

2.3.2 Deep Q-Network and Double Deep Q-Network Environment Setup

Three progressively training stages were designed to evaluate the DQN and DDQN algorithms including static obstacles with predefined goals, and randomized goals with static obstacles, and dynamic obstacles with moving goals, as shown in Figure 3 (a-c). The final stage (Stage 3) presents a dynamic environment combining static and moving obstacles following predefined trajectories. Goals move continuously within a predefined range during episodes, requiring real-time navigation policy adjustments to simulate real-world scenarios.

3. METHODOLOGY

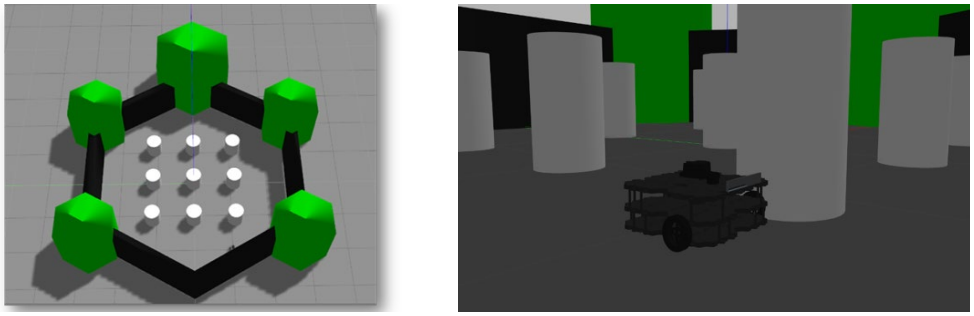
This section describes the methodology for implementing and evaluating the RL-based MR navigation. The simulation environment of Q-learning, DQN, and DDQN are carried out using the open-source ROS system and Gazebo physics simulator. The RL agent selects actions from a predefined discrete action space consisting of turning left, turning right and moving forward. The moving forward action allows the robot to move forward at a constant speed, while turning left and turning right enable the robot to rotate left and right by a fixed angle, respectively. All experiments are run on a virtual machine with Intel Core i7 7th Gen, Ubuntu 20.04 LTS, Python 3.8.10 and 16 G RAM. Turtlebot3 Waffle Pi model in simulated Gazebo environment is used in our experiments. The simulations demonstrate the capabilities of DRL algorithms, guiding the future development of safer, more efficient robots.

3.1 Q-Learning Algorithm

Q-learning is a model-free RL algorithm that updates a Q-table based on state-action pairs. The Q-learning algorithm operates by iteratively updating the Q-values according to the Bellman equation in Equation (1).

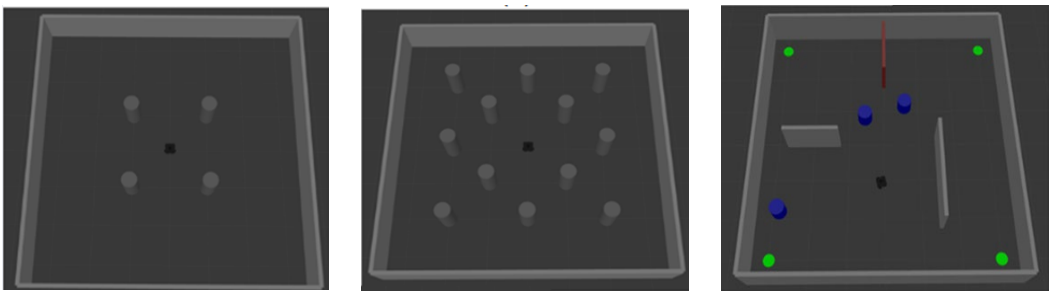
$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \underset{a'}{max} Q(s',a') - Q(s,a) \right] \quad (1)$$

where $Q(s,a)$ is the Q-value for state-action pair (s,a), α is the learning rate, γ is the discount factor, r is the reward, and $\max_{a'} Q(\acute{s},\acute{a})$ represents the best possible Q-value in the next state $\acute{s}$. The reward structure for Q-learning, which outlines the rewards and penalties for various conditions given in Table 2. The Q-learning algorithm is given in Algorithm 1.



(a) (b)

Figure 2. Pre-built Gazebo World for Q-learning: (a) Pre-built Gazebo World for Q-learning; (b) TurtleBot 3 waffle placed in the Gazebo world.



(a) (b) (c)

Figure 3. Simulation environment of DQN/DDQN training stages: (a) Stage 1 - static obstacles with predefined goals; (b) Stage 2 - randomized goals with static obstacles; (c) Stage 3 - dynamic obstacles with moving goals.

Table 2. Q-learning reward system.

Condition	Reward	Explanation
Goal reached	+100	Task completion incentive
Distance reduction	Proportional	Encourages efficient navigation
Collision	-100	Penalizes unsafe behavior
Timeout	-50	Discourages prolonged tasks
Oscillation	-1.0	Discourages abrupt changes

Algorithm 1 Q-learning

Require:
Q-Learning algorithm
Reward function $r : S \times A \rightarrow \mathbb{R}$
Initialize Q-table Q(s,a) with zeros
Initialize learning rate α, discount factor γ, exploration rate ε
Initialize max episodes M and max steps T
Ensure: Trained navigation policy
for e = 1 to M **do**
Reset environment
Place robot at initial position
for t = 1 to T **do**
Observe state st (LiDAR scan + goal orientation)
Select action at using ε-greedy policy:
at ← random action with probability ε
at ← argmax_a Q(st, a) with probability 1-ε
Execute action at and observe reward rt, next state st+1
Update Q-table using Q(st, at)
if st ← st+1 **then**
Decay exploration rate: $\varepsilon \leftarrow \max(\varepsilon_{min}, \varepsilon \times \varepsilon_{decay})$

Table 3. DQN and DDQN reward system.

Condition	Reward	Explanation
Goal reached	+200	Strong incentive for task completion
Reducing distance to goal	Proportional	Encourages progress
Collisions	-150	Strong penalty to prioritize safety
Timeout	-50	Penalizes inefficiency
Time efficiency bonus	+50	Rewards faster task completion
Oscillation penalty	-2.0	Discourages erratic movements

3.2 DQN and DDQN Algorithms

DQN enhances Q-learning by using a deep neural network to approximate Q-values, allowing it to handle large state spaces. The DQN loss function is computed as:

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

where θ represents the neural network weights and θ^- are the target network weights, updated periodically to stabilize training. The DDQN further refines DQN by decoupling action selection and value estimation, reducing overestimation bias. The DDQN target update is:

$$y = r + \gamma Q(s', \underset{a'}{argmax} Q(s', a'; \theta); \theta^-) \tag{3}$$

The reward structure for DQN and DDQN, which includes rewards and penalties for various conditions is given in Table 3. The algorithm is given in Algorithm 2.

3.3 Goal Representation and Distance Calculation

The goal is represented as a 2D coordinate (x_g, y_g) in the simulation environment. The robot's state consists of its current position (x_r, y_r) and orientation θ_r. The Euclidean distance to the goal is computed as:

$$d_g = \sqrt{(x_g - x_r)^2 + (y_g - y_r)^2} \tag{4}$$

The heading angle to the goal is:

$$\theta_g = \arctan\left(\frac{y_g - y_r}{x_g - x_r}\right) \tag{5}$$

The relative orientation error is:

$$\theta_{\text{error}} = \theta_g - \theta_r \tag{6}$$

where θ_{error} guides the agent in selecting appropriate turning actions.

Algorithm 2 DQN/DDQN

Require:
Deep Q-Learning algorithm DQN/DDQN
Reward function r : S × A → $\mathbb{R}$
Initialize Q-network Q(s,a;θ) and target network $\hat{Q}$(s,a;θ⁻)
Initialize replay buffer D, max episodes M, max steps T
Initialize $\varepsilon_0 = 1.0$, $\varepsilon_{min} = 0.05$, batch size B = 64
Ensure: Trained navigation policy
for e = 1 to M **do**
Reset environment
ε = max(ε_{min}, ε_0 × decay^e)
for t = 1 to **T do**
Observe state s_t (LiDAR + goal orientation)
Select action a_t using ε-greedy policy from Q(s_t, a; θ)
Execute a_t and observe reward r_t, next state s_{t+1}
Store transition (s_t, a_t, r_t, s_{t+1}) in D
for each transition (s_j, a_j, r_j, s_{j+1}) **do**
Update θ using loss L = (1/B) Σ(y_j - Q(s_j, a_j; θ))²
if e mod 100 = 0 **then**
Update target network: θ⁻ ← θ

3.4 Training and Evaluation

Training involves three key phases: initialization, episode iteration, and termination. During initialization, the simulation is loaded, and hyperparameters such as the learning rate ($\alpha\alpha$), discount factor ($\gamma\gamma$), and exploration rate ($\epsilon\epsilon$) are defined. In the episode iteration phase, the agent executes actions, receives rewards, and updates the Q-table or neural network. Training terminates upon goal achievement, collision, or reaching the maximum step limit. Algorithm-specific training strategies include:

- Q-learning: Uses a tabular approach with an epsilon-greedy strategy.
- DQN/DDQN: Uses a replay buffer of size 10,000 and mini-batch updates with 64 samples.
- DDQN: Employs a dual-network approach to mitigate Q-value overestimation.

Performance is assessed using metrics such as success rate, time to goal, collision rate, and cumulative reward. These metrics provide a comprehensive evaluation of the algorithms' effectiveness in different stages of the simulation environment.

4. SIMULATION AND EXPERIMENTAL ANALYSIS

This section presents a comprehensive analysis of the performance of the RL-based MR navigation design environments. The experiment designs were conducted in Gazebo, utilizing the TurtleBot3 Waffle Pi platform. Three progressively environment configurations were implemented to evaluate the RL algorithms: (1) Q-Learning environment (Phases 1 - 2) - A hexagonal arena with static obstacles, focusing on collision avoidance and discrete state space based on LiDAR data; (2) Intermediate DQN/DDQN environment (Stages 1 - 2) - A rectangular arena with increased complexity, featuring static obstacles with random positioning and goal-oriented navigation tasks; (3) Advanced DQN/DDQN Environment (Stage 3) - A dynamic environment with moving obstacles and adaptive goal positions, simulating more advanced navigation scenarios. The results are organized into three main parts: simulation setup and parameters, performance analysis of each algorithm, and a comparative analysis of their effectiveness in different environments.

4.1 Training and Evaluation

The algorithms were configured with specific hyperparameters to optimize their performance. Table 4 and Table 5 illustrate the list of parameters and their values for Q-learning and DQN/DDQN respectively, which are used in this experiment.

Table 4. Q-Learning parameters

Parameter	Value	Parameter	Value
Initial Position	Random	Min Epsilon	0.05
Max Episodes	300	Max LiDAR Distance	1.20 m
Max Steps per Episode	500	Collision Distance	0.20 m
Learning Rate (α)	0.5	Zone 0 Length	0.60 m
Discount Factor (γ)	0.9	Zone 1 Length	1.00 m
Initial Epsilon	0.9	Linear Speed	0.100 m/s
Epsilon Decay Rate	0.96	Angular Speed	0.500 rad/s

Table 5. DQN/DDQN parameters.

Parameter	Stage 1	Stage 2	Stage 3	Parameter	Stage 1	Stage 2	Stage 3
Max Episodes	1300	2500	1300	Epsilon Decay Rate	0.995	0.998	0.997
Max Steps per Episode	500	500	600	Minimum Epsilon	0.05	0.02	0.05
Batch Size	64	64	64	Memory Size	100,000	100,000	200,000
Learning Rate	0.0001	0.00005	0.00002	Target Network Update	5	5	10
Discount Factor (γ)	0.99	0.99	0.99	State Size	28	28	28
Initial Epsilon (ε)	1.0	0.2	0.15	Action Size	5	5	5

4.2 Performance Analysis

4.2.1 Q-Learning

The Q-learning algorithm was evaluated in two phases within a simulated Gazebo environment: an initial training Phase 1 (100 episodes) and an extended training Phase 2 (300 episodes). In the initial training phase, the environment was designed to present a moderate level of difficulty, with a fixed number of obstacles strategically placed to challenge the robot's navigation capabilities. The primary goal of this phase was to allow the Q-learning algorithm to learn the fundamental principles of obstacle avoidance and develop a basic understanding of the environment's spatial layout. The hyperparameters shown in Table 4 were chosen to encourage exploration and initial learning.

Figures 4(a) and 4(b) show the average reward per episode for Phase 1 and Phase 2, respectively. The agent initially struggled with consistently negative rewards ranging from -6 to -8 during the first 60 episodes in Phase 1. This reflects the robot's exploratory behavior as it learned to navigate the environment through trial and error. A notable improvement occurred around episode 60, where the average reward dramatically increased to approximately 0, indicating a breakthrough in the agent's learning process as it began to exploit more effective policies. However, this improvement was not sustained, and the rewards fluctuated back to negative values in the final episodes, suggesting the need for extended training. Following the initial

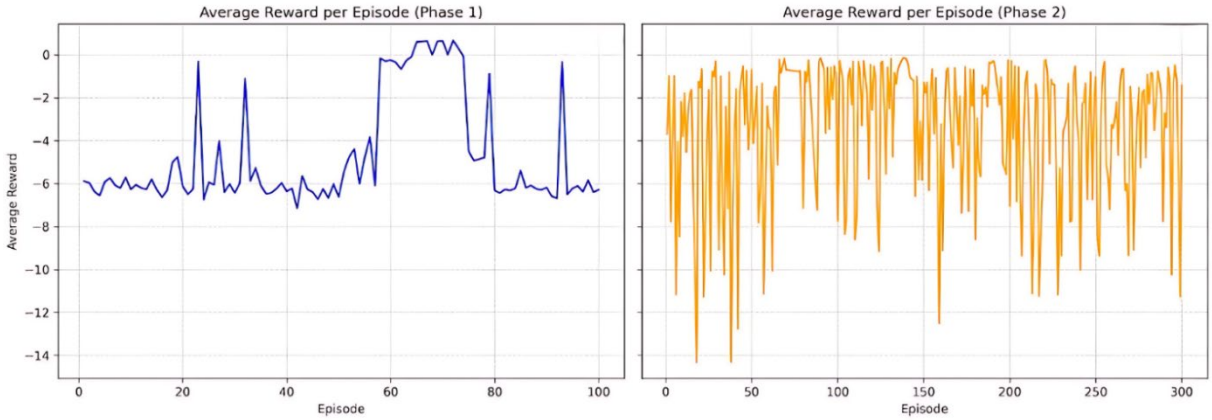
training phase, the robot underwent an extended training period of 300 episodes in Phase 2. The environment remained the same, but the hyperparameters were adjusted to promote exploitation of the knowledge gained in the previous phase and fine-tune the robot's obstacle avoidance behavior. Figure 4(b) demonstrates that Phase 2 rewards showed significantly different characteristics compared to Phase 1. The average rewards exhibited much higher variability, fluctuating between approximately -14 and -2 throughout the 300 episodes. While the rewards remained predominantly negative, the increased exploration range suggests the agent was actively learning and adapting to more complex scenarios or exploring different strategies for obstacle avoidance.

Figures 5(a) and 5(b) illustrate the steps per episode for Phase 1 and Phase 2, respectively. The step per episode in Phase 1 showed highly variable performance with most episodes requiring fewer than 100 steps. However, critical spikes occurred around episodes 60-70, where the number of steps reached up to 500 steps per episode, coinciding with the period of reward improvement. This indicates that while the agent was learning better policies, it was still taking longer paths to reach goals during this learning phase. Figure 5(b) shows the steps per episode in Phase 2 remained highly variable throughout the training period, with episodes ranging from fewer than 50 steps to over 400 steps. The persistent variability indicates that the agent continued to encounter diverse navigation challenges and had not yet converged to a consistently optimal policy. The frequent spikes in step count suggest the agent was still learning to handle complex obstacle configurations and required extended training to achieve stable performance.

4.2.2 DQN and DDQN

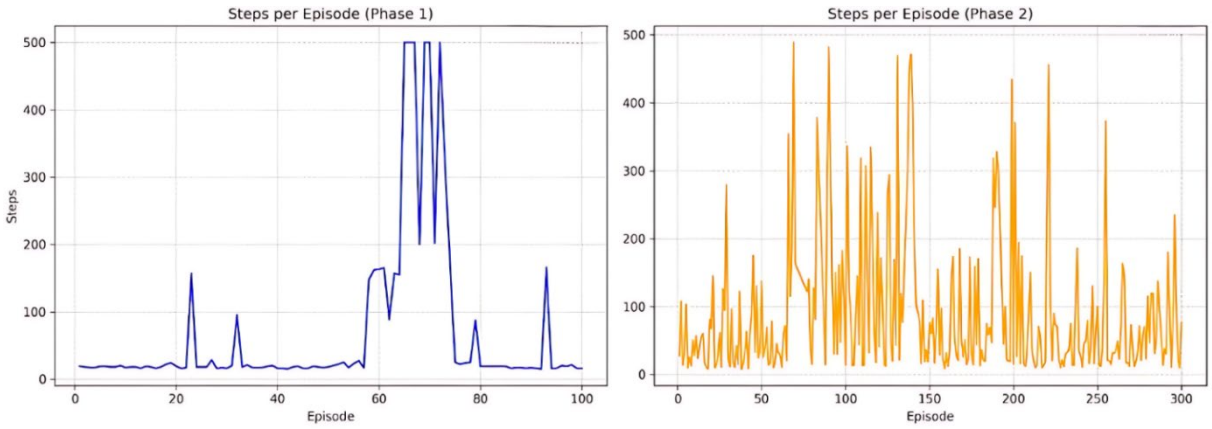
The performance of DQN and DDQN algorithms was evaluated across three progressive stages of increasing complexity (Stages 1 - 3), revealing distinct learning characteristics for each algorithm.

Stage1 Performance - Static obstacles with predefined goal. Analysis of Figure 6(a) showed that the average steps per episode decreased consistently throughout training for both algorithms, reflecting improved navigation efficiency as the agents learned more optimal paths to the goal, as depicted in Figures 6(b). By the end of training, DQN's average steps per episode stabilized at 125, while DDQN achieved superior efficiency with 84 steps per episode, demonstrating DDQN's enhanced pathfinding capabilities and representing a 33% improvement in navigation efficiency compared to DQN.



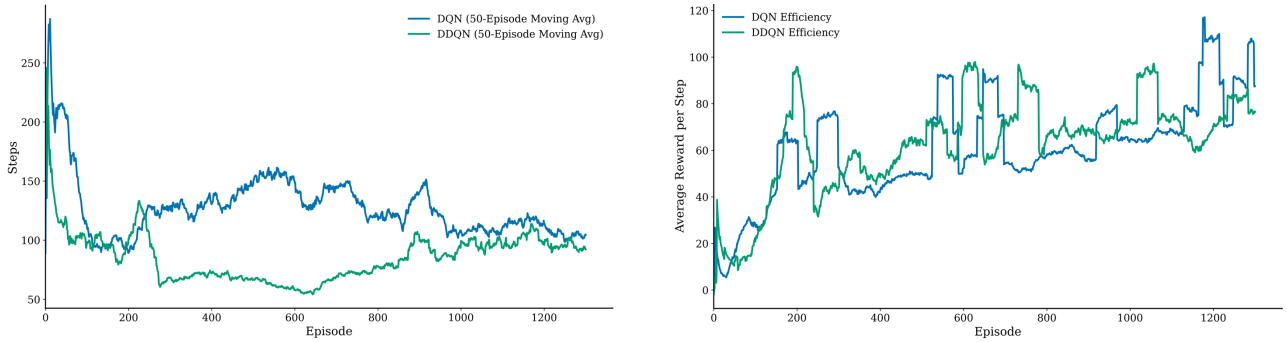
(a) (b)

Figure 4. Experiments study of Q-learning training in simulation environment: (a) Average rewards obtained during 100 episodes; (b) Average rewards obtained during 300 episodes.



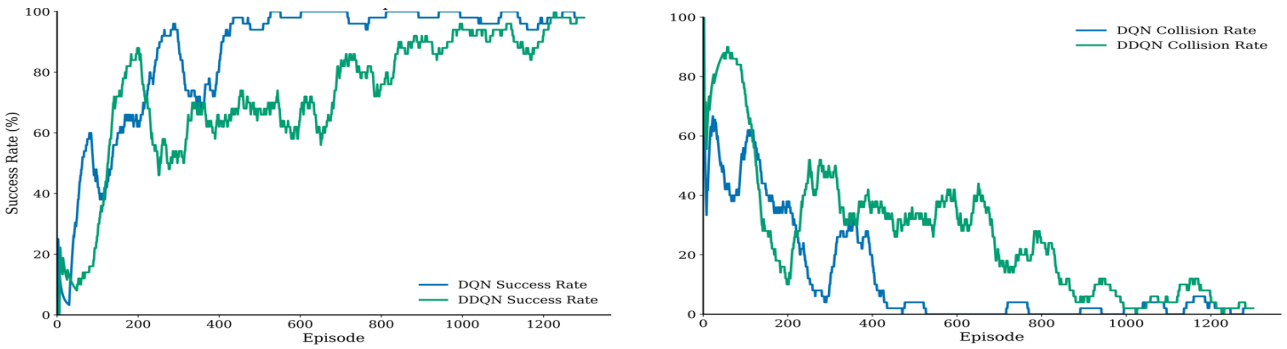
(a) (b)

Figure 5. Experiments study of Q-learning training in simulation environment: (a) Step per episode during 100 episodes; (b) Step per episode during 300 episodes.



(a) (b)

Figure 6. Simulation environment Stage 1: (a) Step per episode; (b) Learning efficiency.



(a) (b)

Figure 7. Simulation Environment Stage 1: (a) Success rate; (b) Collision rate.

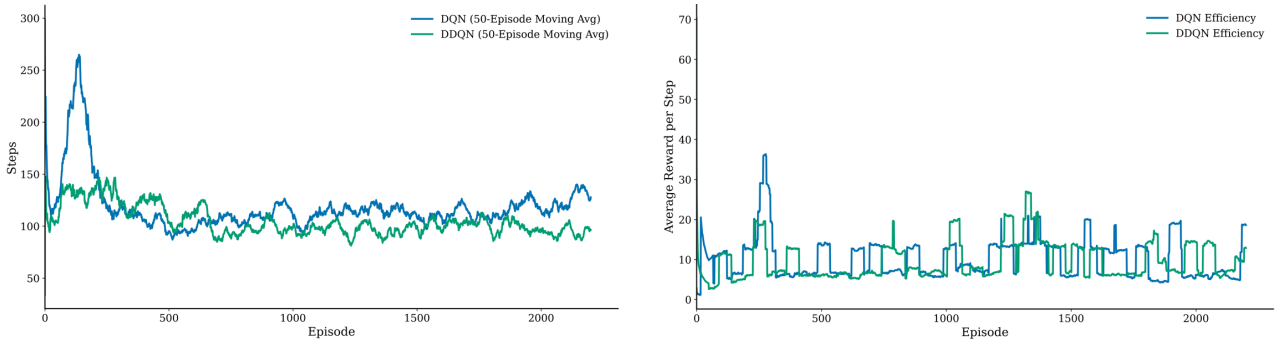
As shown in Figure 7(a), DQN's success rate started at 32.0% while DDQN began at 8.0%, both demonstrating steady improvement throughout the training process and reaching identical 98.0% success rates by the final episodes. This indicates that both agents effectively learned to navigate towards static goals with high consistency. Figure 7(b) shows the collision rates for both algorithms dropped significantly during the early stages of training, with DQN falling below 10% within the first 200 episodes and DDQN achieving this milestone within 300 episodes, both stabilizing at 2.0% by the end of training. Over the course of training, DQN completed 1,148 successful episodes while experiencing 140 collisions, whereas DDQN achieved 959 successful episodes with 328 collisions, highlighting DQN's superior safety record during the learning process.

The Stage 1 results reveal an important nuance in algorithm performance: DQN achieved a higher best reward (13,273.6) compared to DDQN (11,170.0), yet DDQN demonstrated superior overall efficiency with 33% fewer steps per episode (84 vs. 125 steps). This apparent contradiction reflects fundamental differences in the algorithms' learning strategies. DQN's higher peak reward stems from occasional episodes where the agent discovered particularly efficient paths early in training and exploited them aggressively, accumulating rewards rapidly. However, this aggressive exploitation came at the cost of consistency - DQN's reward variance remained higher throughout training, indicating less stable policy convergence.

In contrast, DDQN's lower peak reward but higher consistency reflects its bias-correction mechanism, which prevents overestimation and promotes more conservative policy updates. DDQN sacrifices occasional high-reward episodes for long-term stability, resulting in policies that are more reliable across diverse scenarios. This trade-off becomes critical in later stages: DDQN's conservative learning in Stage 1 translates to superior generalization in Stages 2 and 3, where environmental complexity increases. The 33% improvement in navigation efficiency (fewer steps) despite lower peak rewards demonstrates that DDQN learns more optimal long-term policies, even if individual episodes occasionally yield lower immediate rewards.

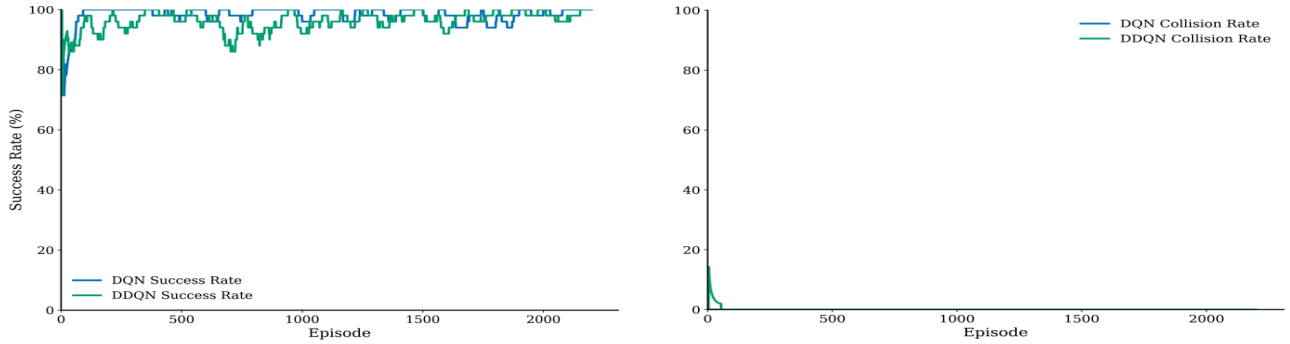
This finding has important implications for algorithm selection: environments requiring consistent performance (e.g., industrial applications) should favor DDQN despite its lower peak rewards, while applications tolerating higher variance for occasional optimal performance might benefit from DQN's aggressive exploration.

Stage 2 Performance - Randomized goals with static obstacles. The comparative performance of DQN and DDQN in this stage introduce more complexity compared to Stage1, where agents must navigate to randomized goal locations. Analysis of Figure 8(a) showed that the average number of steps per episode decreases gradually as training progresses for both algorithms, reflecting improvements in navigation efficiency, as depicted in Figure 8(b). DQN's average steps per episode stabilize at approximately 118 by the end of Stage 2, with occasional spikes in step count suggesting suboptimal navigation in certain randomized goal configurations, indicating that the agent sometimes struggles to generalize its learned policies across diverse scenarios. DDQN demonstrates superior efficiency with average steps per episode stabilizing quickly at approximately 103 steps, showing not only consistent goal-reaching behavior but also more efficient navigation compared to DQN. The absence of large spikes in step counts further underscores DDQN's optimized and stable navigation behavior across various randomized goal configurations, highlighting its ability to adapt seamlessly to dynamic environments while maintaining efficiency.



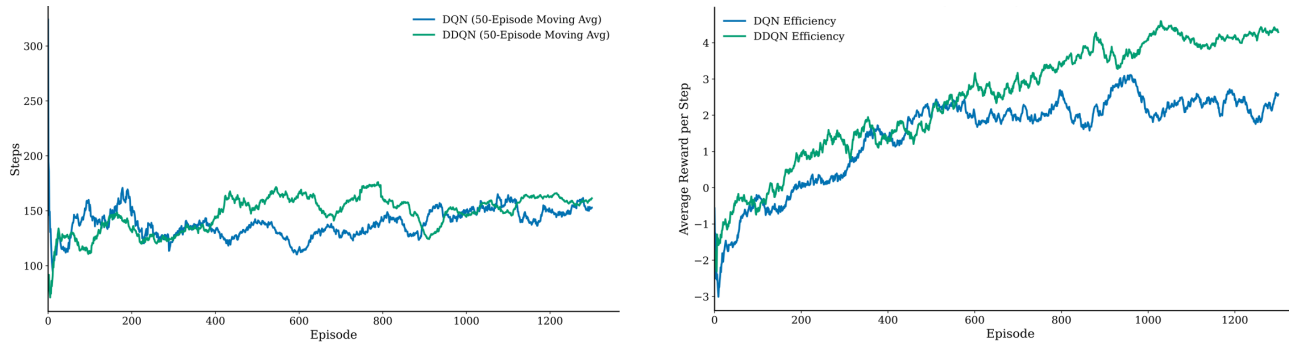
(a) (b)

Figure 8. Simulation environment Stage 2: (a) Step per episode; (b) Learning efficiency.



(a) (b)

Figure 9. Simulation environment Stage 2: (a) Success rate; (b) Collision rate.

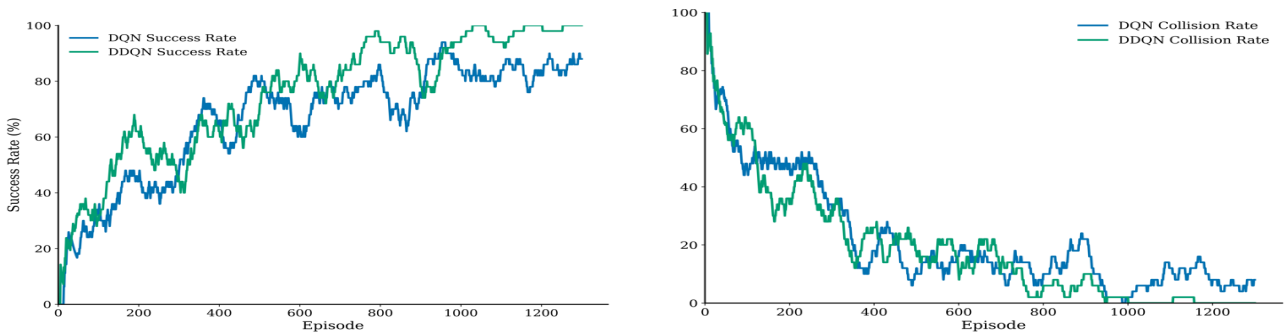


(a) (b)

Figure 10. Simulation environment Stage 3: (a) Step per episode; (b) Learning efficiency.

As shown in Figure 9(a), DQN's success rate improves steadily throughout the training process, reaching a final value of 96%, demonstrating effective learning to navigate towards randomized goals and avoid obstacles. However, the agent struggles to achieve perfect consistency in goal-reaching due to the variability in goal positions. DDQN achieves superior performance with a consistent 100% success rate by 600 episodes, reflecting its superior ability to generalize learned policies to new and diverse goal locations while maintaining effective obstacle avoidance. Both algorithms maintain consistently low collision rates after the initial learning phase, with DDQN's collision rate remaining near zero for most of the training processes presented in Figure 9(b), confirming that both learned policies prioritize collision-free navigation even in dynamic and challenging scenarios.

Stage 3 Performance - Dynamic obstacles with moving goals. The comparative performance of DQN and DDQN in this stage presents the most challenging scenario where agents must navigate to randomized goals while avoiding dynamic obstacles. The average number of steps per episode decreases gradually as training progresses for both algorithms, reflecting improvements in navigation efficiency as shown in Figures 10(a) and 10(b). The DQN's average steps per episode stabilize at 138 by the end of training, with occasional spikes in step count indicating difficulties in navigating around dynamic obstacles efficiently. The relatively high step count suggests that DQN prioritizes safety and collision avoidance over path length optimization, particularly in complex goal configurations. DDQN's average steps per episode stabilized at 148 by the end of training, reflecting efficient navigation strategies, although the higher step count compared to Stage 2 suggests a prioritization of safety and thorough obstacle avoidance over minimizing path length. The absence of significant spikes in step counts further indicates DDQN's ability to maintain consistent and optimized navigation behavior, even under complex conditions.



(a) (b)

Figure 11. Simulation Environment Stage 3: (a) Success rate; (b) Collision rate.

From Figure 11(a), DQN's success rate starts at 18.0% and gradually improves throughout the training process, stabilizing at 88.0%. This 12% performance gap compared to DDQN directly demonstrates the impact of Q-value over-estimation bias in dynamic environments. Analysis of failure cases revealed that DQN's over-estimation caused the agent to select paths that appeared optimal based on inflated Q-values but actually led to collisions with moving obstacles. The theoretical foundation of this behavior lies in DQN's use of the max operator for both action selection and evaluation, which amplifies estimation errors in states with high uncertainty precisely the conditions present in dynamic environments where obstacle positions change unpredictably [22]. DDQN achieves superior performance with a starting success rate of 30.0%, which steadily improved over the course of training, ultimately reaching 100.0% by the final episodes. This superior performance stems from DDQN's decoupled architecture: by using the online network to select actions and the target network to evaluate them, DDQN prevents the positive feedback loop that causes over-estimation [22]. In dynamic navigation scenarios, this architectural difference translates to more accurate risk assessment, DDQN correctly identifies high-risk states and selects conservative actions, while DQN's over-estimation causes it to underestimate collision probabilities. DDQN's zero collision rate versus DQN's 8% rate in Stage 3 provides empirical validation of the theoretical advantages of decoupled action selection and evaluation.

Following Tables 6 to 8 summarize a detailed numerical analysis of the experimental results presented in Figures 6 to 11 for three progressive stages of simulation environment. Analysis in Table 6 showed that the DDQN achieved the highest progressive performance in stages 2 and 3, maintaining a 100% success rate including the most challenging dynamic environment of Stage 3. The DQN showed declining performance in complex scenarios, with a success rate of 96% in Stage 2 and 88% in Stage 3. The Q-learning, while effective in static environments with a 98% success rate, struggled in dynamic scenarios, achieving only an 88% success rate in Stage 3. This highlights DDQN's superior adaptability to complex and dynamic environments compared to DQN and Q-learning. Table 7 presents the time required for agents and the time step required for each agent to reach an average success rate of training in all environments.

Regarding learning efficiency, the DDQN demonstrated faster convergence across all stages, stabilizing after approximately 400 episodes in Stage 1 and 800 episodes in Stage 3. Meanwhile, DQN required longer training periods, with convergence achieved after ~600 episodes in Stage 1 and ~1000 episodes in Stage 3 as shown in Table 8. It can be observed that the Q-learning showed the slowest convergence, requiring extensive exploration phases and exhibiting high variance in performance metrics. This indicates that DDQN not only performs better but also learns more efficiently than DQN and Q-learning. In terms of stability, DDQN demonstrated the highest reward stability, with consistent performance across all stages. DQN showed moderate stability with fluctuations, particularly in dynamic environments, while Q-learning exhibited high variance, reflecting its limitations in handling complex and dynamic scenarios. This underscores DDQN's robustness and reliability in diverse environments. Finally, in terms of computational efficiency, Q-learning required minimal computational resources, making it suitable for simple environments with limited hardware capabilities. DQN had moderate computational requirements, with CPU usage of 43.30% in Stage 1. DDQN, while more computationally demanding with CPU usage of 57.40% in Stage 3, justified its higher resource requirements with superior performance in complex environments. This trade-off between computational cost and performance must be considered in future.

Table 6. Average success rate of DQN/DDQN for all environments.

Algorithms	Stage 1	Stage 2	Stage 3
DQN	98%	96%	88%
DDQN	98%	100%	100%

Table 7. Time to reach average success rate of training in all environments.

Algorithms	Stage 1		Stage 2		Stage 3	
	Time (h)	Time Step	Time (h)	Time Step	Time (h)	Time Step
DQN	7.18	125	7.50	118	6.36	138
DDQN	5.20	84	7.15	103	7.13	148

Table 8. Other performance metrics.

Metric	Stage1		Stage 2		Stage 3	
	DQN	DDQN	DQN	DDQN	DQN	DDQN
Convergence Speed	~600 episodes	~400 episodes	~1000 episodes	~600 episodes	~1000 episodes	~800 episodes
Reward Stability	Moderate	High	Moderate	High	Moderate	High
Collision Rate	2%	2%	Low	Near Zero	8%	Near Zero
Avg. CPU Usage	43.30%	41.90%	47.65%	49.65%	52.00%	57.40%
Avg. Memory Usage	5.60%	5.40%	4.20%	4.05%	2.80%	2.70%

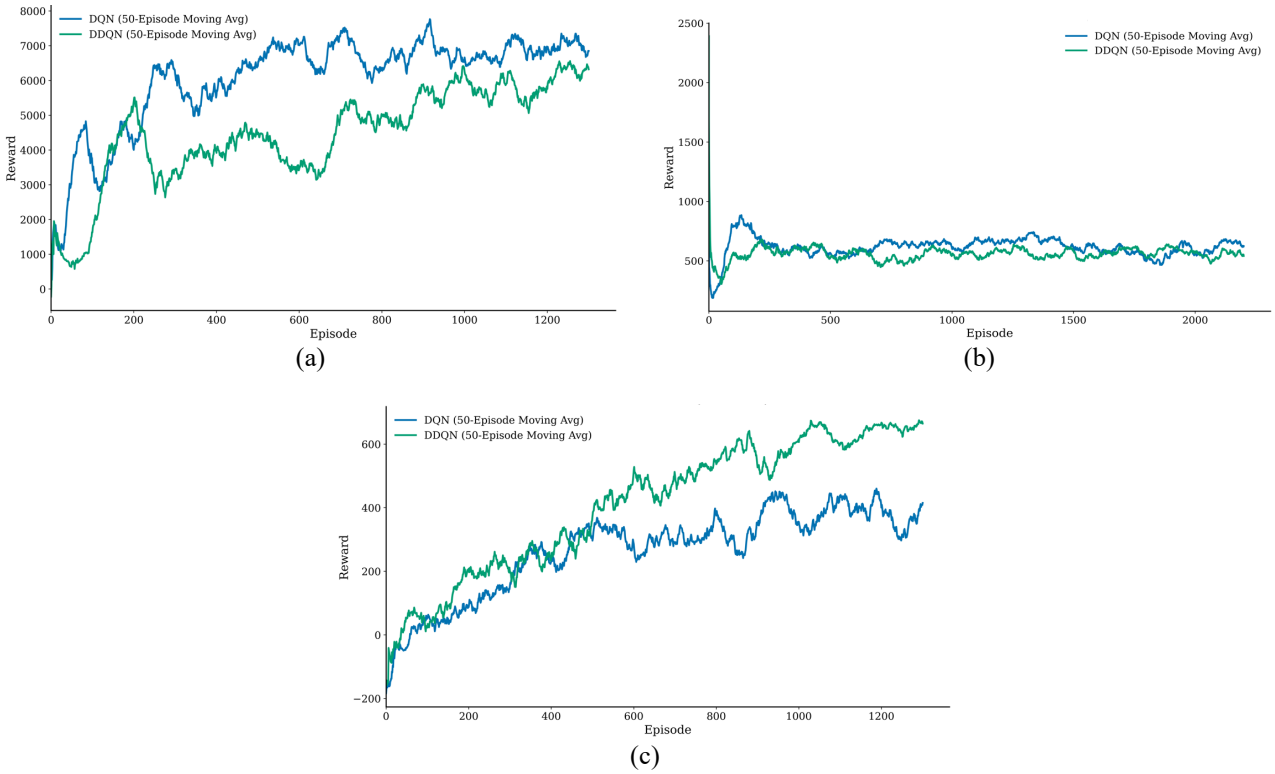


Figure 12. Rewards obtained during three different simulation environments of every 50 episodes for DQN and DDQN algorithms: (a) Stage 1; (b) Stage 2; (c) Stage 3.

Analysis of Figures 12(a) to 12(c) reveals that the Q-learning utilizing deeper networks like DQN/DDQN demonstrates significant enhancement in the average reward metric. In Stage 1 simulation environment, the performance of DQN and DDQN exhibited significant variability during the initial training phase, particularly within the first 300 episodes as shown in Figure 12(a). This variability is attributed to the exploration-heavy epsilon-greedy strategy employed during the early stages of training. DQN's 50-episode moving average reward began to stabilize after approximately 600 episodes, while DDQN demonstrated faster convergence, stabilizing after approximately 400 episodes, reflecting improved learning stability. By the end of training, DQN achieved a higher best reward of 13,273.6 compared to DDQN's 11,170.0, indicating that both algorithms successfully learned efficient navigation strategies with DQN showing superior peak performance.

To provide a more comprehensive evaluation beyond success and collision rates, the following complementary metrics and trade-offs are considered in the analysis:

a) Path Efficiency**:** Defined as the ratio of the shortest possible path (Euclidean distance) to the actual distance traveled by the agent. This metric is closely related to the time steps per episode recorded in Table 7. For instance, in Stage 3, DDQN takes slightly more steps (148) than DQN (138), indicating that DDQN may sacrifice minimal path efficiency to prioritize safety and collision avoidance, especially when navigating complex dynamic obstacles.

b) Energy Consumption: In mobile robotics, energy consumption is proportional to the total distance traveled and the frequency of angular changes (turning actions). Given the proportional relationship between travel time and path length, the step count (Table 7) serves as a robust proxy for energy consumption. The trade-off observed is that while DDQN is safer and more stable, its preference for conservative paths in complex dynamic environments suggests a potentially higher energy consumption per successful episode compared to an aggressive, less reliable path.

c) Response Latency: This metric represents the time delay between the robot sensing a change (e.g., an obstacle appearing) and the execution of the corrected action. While not explicitly measured in the current time metrics (Table

7), it is a critical factor for real-time applications. The computational overhead introduced by DRL algorithms, especially DDQN's dual-network architecture, inherently increases response latency compared to simpler, classical control loops. Although DDQN provides stability, future deployment would require minimizing this latency through hardware acceleration or optimized network pruning to ensure effective real-time control.

In Stage 2 simulation environment as shown in Figure 12(b), DQN exhibits substantial variability throughout the training process, particularly during the first 800 episodes. This variability suggests that DQN struggles to maintain consistent policy updates, likely due to Q-value overestimations in the more complex randomized environment. While the 50-episode moving average reward shows a gradual upward trend, it stabilizes after approximately 1000 episodes. In contrast, DDQN demonstrates significantly less variability and more stable learning, with the 50-episode moving average reward increasing rapidly during the initial training phase and stabilizing after approximately 600 episodes. This early stabilization demonstrates DDQN's ability to mitigate Q-value overestimations using a target network and dual-network updates, reflecting superior efficiency in learning optimal navigation strategies under randomized goal conditions.

Finally, in Stage 3 simulation environment as shown in Figure 12(c), DQN exhibits significant variability particularly during the initial 500 episodes, reflecting the challenges of learning optimal navigation strategies in dynamic environments. The 50-episode moving average reward shows a slow but steady upward trend, stabilizing after approximately 1000 episodes, achieving a best reward of 851.3. In contrast, DDQN demonstrates smooth and steady improvement throughout training, with the 50-episode moving average reward increasing consistently during the initial episodes and stabilizing after approximately 800 episodes, achieving a best reward of 846.4. The absence of significant reward variability in DDQN underscores the algorithm's robustness and its ability to mitigate Q-value overestimations, while DQN's overall reward trends highlight the difficulties it faces in consistently handling the added complexity of dynamic obstacles.

5. CONCLUSION

This paper presented the RL for autonomous navigation that can adapt to changing environments**.** Through three progressive training stages, the work addressed fundamental and advanced challenges in robot navigation, including static obstacle avoidance, randomized goal-reaching, and dynamic obstacle handling. The DDQN demonstrated superior performance, achieving a 100% success rate in dynamic environments with near-zero collisions, while DQN and Q-learning showed limitations in handling complex scenarios. The empirical analyses underscore the efficacy of the proposed algorithm.

Q-Learning demonstrated effective learning in basic navigation tasks, with clear state-action mapping through the Q-table and stable performance in static environments. Its low computational requirements make it suitable for simple applications. However, Q-learning struggles with scalability and dynamic scenarios, as it is limited to discrete state-action spaces and cannot handle continuous state representations effectively. Additionally, it requires extensive exploration phases, exhibits high variance in performance metrics, and converges slower compared to deep learning approaches. The DQN has successfully handled continuous state spaces, achieving 96-98% success rates in static environments and demonstrating effective goal-directed navigation. Its moderate computational requirements make it a balanced choice for moderately complex environments. However, DQN exhibits moderate reward variability and slower convergence, particularly in dynamic environments. It struggles with moving obstacles, achieving an 88% success rate in Stage 3, and requires longer training periods for stability. The DDQN outperformed both Q-learning and DQN, achieving a 100% success rate in all stages, including the most challenging dynamic environments. Its near-zero collision rates, faster convergence, and consistent performance across all environmental conditions highlight its robustness and adaptability. However, DDQN has higher computational demands, with longer training times and increased memory requirements for its dual-network architecture. Additionally, it prioritizes safety over path length optimization, resulting in higher average steps per episode (148 in Stage 3). Despite these limitations, DDQN's superior performance in complex environments makes it the most suitable algorithm for real-world applications requiring dynamic obstacle avoidance and goal-directed navigation.

The novel contributions of this work extend beyond algorithm comparison to provide actionable engineering insights: (1) Progressive Complexity Framework: The three-stage training methodology reveals that DDQN's architectural advantages only manifest under dynamic conditions (Stage 3), while simpler algorithms suffice for static environments, enabling cost-effective algorithm selection; (2) Quantified Breaking Points: We establish that Q-learning's discrete state representation fails beyond basic scenarios (achieving only 88% success in dynamic environments), DQN encounters stability issues with moving obstacles (88% success, 8% collision), while DDQN maintains 100% success with zero collisions; (3) Deployment Guidelines: The computational trade-off analysis (Table 8) provides practitioners with concrete metrics DDQN's 10% higher CPU usage is justified only when navigation scenarios involve dynamic obstacles and require zero-collision guarantees.

ACKNOWLEDGEMENT AND FUNDING

The authors would like to acknowledge the financial support from the Ministry of Higher Education (MOHE) Malaysia and Universiti Teknologi Malaysia under Matching Grant: *Safe and Flexible Manufacturing System* between Universiti Teknologi Malaysia and Telkom University, Indonesia (Q.J130000.3023.04M12 and R.J130000.7623.4B796).

DECLARATION OF CONFLICTING INTERESTS

The authors declare no potential conflicts of interest with respect to the research and publication of this article.

REFERENCES

[1] Q. Zou, Q. Sun, L. Chen, B. Nie and Q. Li, A comparative analysis of lidar slam-based indoor navigation for autonomous vehicles, *IEEE Transactions on Intelligent Transportation Systems*, 23, 202, 6907-6921.

[2] D. Küpper, L. Markus, K. Claudio, K. Kristian, M. Andreas, R. Lässig and T. Buchner, *Advanced Robotics in the Factory of the Future*, Boston Consulting Group, 2021.

[3] L. Chen, W. Zhan, W. Tian, Y. He and Q. Zou, Deep integration: A multi-label architecture for road scene recognition, *IEEE Transactions on Image Processing*, 28(10), 2019, 4883-4898.

[4] S. Kumar, V. Manjrekar, V. Singh and B. Kumar, Integrated yet distributed operations planning approach: A next generation manufacturing planning system, *Journal of Manufacturing Systems*, 54, 2020, 103-122.

[5] M. B. Alatise and G. P. Hancke, A review on challenges of autonomous mobile robot and sensor fusion methods, *IEEE Access*, 8, 2020, 39830-39846.

[6] V. R. F. Miranda, A. A. Neto, G. M. Freitas and L. A. Mozelli, Generalization in deep reinforcement learning for robotic navigation by reward shaping, *IEEE Transactions on Industrial Electronics*, 71, 2024, 6013-6020.

[7] J. Pak, J. Kim, Y. Park and H. I. Son, Field evaluation of path-planning algorithms for autonomous mobile robots in smart farms, *IEEE Access*, 10, 2022, 60253-60266.

[8] K. T. Song, Y. H. Chiu, L. R. Kang, S. H. Song, C. A. Yang, P. C. Lu and S. Q. Ou, Navigation control design of a mobile robot by integrating obstacle avoidance and lidar slam, *Proceedings of 2018 IEEE International Conference on Systems, Man, and Cybernetics*, Miyazaki, Japan, 2018, 1833-1838.

[9] M. N. A. Wahab, S. Nefti-Meziani and A. Atyabi, A comparative review on mobile robot path planning: Classical or meta-heuristic methods?, *Annual Reviews in Control*, 50, 2020, 233-252.

[10] S. Kadry, G. Alferov, V. Fedorov and A. Khokhriakova, Path optimization for D-Star algorithm modification, *AIP Conference Proceedings*, 2425(1), 2022.

[11] J. S. Smith, R. Xu and P. Vela, EgoTEB: Egocentric, perception space navigation using timed-elastic-bands*, Proceedings of IEEE International Conference on Robotics and Automation*, Paris, France, 2020, 2703-2709.

[12] J. Zhao, S. Liu and J. Li, Research and implementation of autonomous navigation for mobile robots based on SLAM algorithm under ROS, *Sensors*, 22(11), 2022, 4172.

[13] S. M. B. P. Samarakoon, M. A. V. J. Muthugala and M. R. Elara, Global and local area coverage path planner for a reconfigurable robot, *Proceedings of 2022 IEEE Congress on Evolutionary Computation*, Padua, Italy, 2022, 1-8.

[14] Q. Feng, K. Du, H. Wang, T. Chen, X. Meng, S. Wang and B. Lu, Path planning algorithm of mobile robot based on improved q-learning algorithm, *Proceedings of 2023 IEEE Information Technology, Networking, Electronic and Automation Control Conference*, Chongqing, China, 2023, 133-136.

[15] S. W. H. Teoh, K. Kamarudin, N. A. N. Ali, M. M. M. Zainal, M. R. Manan and S. M. Mamduh, Reinforcement learning for mobile robot's environment exploration, *Journal of Physics: Conference Series*, 2641, 2023, 012003.

[16] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed., Cambridge, MA: MIT Press, 2018.

[17] K. Zhu and T. Zhang, Deep reinforcement learning-based mobile robot navigation: A review, *Tsinghua Science and Technology*, 26, 2021, 674-691.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare and D. Hassabis, Human-level control through deep reinforcement learning, *Nature*, 518, 2015, 529-533.

[19] Y. Yuxiang, W. Shuting, X. Yuanlong, H. Yiming and L. Hu, Q-learning-based collision-free path planning for mobile robot in unknown environment, *Proceedings of 2022 IEEE International Conference on Industrial Electronics and Applications*, Chengdu, China, 2022, 1104-1109.

[20] C. Vishal and J. Amudha, Autonomous driving mobile robot using q-learning, *Proceedings of 2022 IEEE International Conference on Frontiers of Technology*, Belgaum, India, 2022, 1-8.

[21] Y. Wang and S. Yang, Navigation for mobile robots using reinforcement learning and dynamic obstacle detection, *IEEE Access*, 10, 2022, 18001-18010.

[22] H. Hasselt, A. Guez and D. Silver, Deep reinforcement learning with double q-learning, *Proceedings of the AAAI Conference on Artificial Intelligence*, Phoenix, Arizona, USA, 30(1), 2016.

[23] L. Tai, G. Paolo and M. Liu, Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation, *Proceedings of 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vancouver, Canada, 2017, 31-36.

[24] Z. Yue, Z. Liu, Y. Miao and H. Wang, Efficient reinforcement learning for 3d lidar navigation of mobile robots, *Proceedings of Chinese Control Conference*, Hefei, China, 2022, 3755-3760.