Uniosun Journal of Engineering and Environmental Sciences (UJEES), Vol. 7 No. 2, Sept. 2025
DOI: https//doi.org/10.64980/ujees.v7i2.433 **W. O. Adedeji**
Published: September 30, 2025

AUTONOMOUS MOBILE ROBOT PATH PLANNING USING REINFORCEMENT LEARNING IN MATLAB

W. O. ADEDEJI*

Department of Mechanical Engineering, Osun State University, Osogbo. Nigeria.
**Corresponding Author: wasiu.adedeji@uniosun.edu.ng*

Abstract
Path planning is a critical component of autonomous mobile robots, allowing them to navigate their environment efficiently while avoiding obstacles. Traditional algorithms such as A* and Dijkstra's provide optimal solutions based on predefined maps but struggle with adaptability in dynamic and unknown environments. These limitations necessitate the adoption of machine learning-based approaches such as Reinforcement Learning (RL). RL, particularly Q-learning and Deep Q-Networks (DQN), enables robots to learn optimal navigation strategies through continuous interaction with their surroundings, allowing them to adapt to environmental changes. This research implements and evaluates Q-learning and DQN in MATLAB, training autonomous robots in a simulated environment to navigate obstacles while optimizing path efficiency. The study compares RL-based path planning with traditional algorithms (A* and Dijkstra's) by assessing computational complexity, adaptability, and performance in dynamic scenarios. Key findings highlight the effectiveness of RL-based methods in improving realtime decision-making and adaptability, but also expose challenges such as high computational demands and convergence issues. The research provides insights into the applicability of RL for autonomous robot navigation and proposes potential optimizations for real-world deployment.

Keywords
Autonomous, Path Planning, Reinforcement, Navigate, Learning

1. INTRODUCTION

As autonomous robots increasingly find applications in industries such as logistics, surveillance, healthcare, and transportation, the demand for efficient, adaptive path-planning algorithms becomes more critical. Traditional methods, such as rule-based systems and heuristic approaches like A* and Dijkstra's algorithm, rely on predefined maps and deterministic rules (Sutton and Barto, 2018). While these methods guarantee optimal paths in static environments, they struggle to perform in dynamic and uncertain settings where obstacles and goals can change unpredictably. Reinforcement learning (RL) offers a promising alternative, enabling robots to learn optimal navigation strategies through interaction with their environment. Unlike traditional methods that depend on precomputed paths, RL-based algorithms adapt by continuously refining their decision-making policies based on experience. This adaptive approach proves especially beneficial in environments characterized by uncertainty, unexpected obstacles, and real-time decision-making needs. Among various RL techniques, Q-learning and Deep Q-Networks (DQN) have been widely explored due to their ability to handle discrete and continuous state spaces, respectively (Sutton and Barto, 2018).

This study investigates the implementation of RL-based path-planning algorithms in MATLAB, comparing their performance against conventional methods. By evaluating efficiency, adaptability, and computational complexity, the research aims to provide insights into the practical viability of RL for autonomous navigation in real-world scenarios. Traditional path-planning algorithms, such as A* and Dijkstra's, are effective in structured, static environments where obstacles and paths remain unchanged. These methods compute optimal paths based on predefined maps and deterministic rules, ensuring reliable navigation in well-defined settings. However, in dynamic or unpredictable environments where obstacles may appear, move, or change, these algorithms struggle to adapt, necessitating real-time decision-making (Wang et al.,2020). Reinforcement learning (RL)-based approaches offer a robust solution by enabling robots to learn navigation strategies through continuous interaction with their environment (Sutton and Barto, 2018). Unlike conventional algorithms, RL methods do not require prior knowledge of the environment, making them particularly suitable for real-world applications. Despite their potential, several challenges must be addressed to make RL-based path planning both practical and efficient in dynamic environments (Zhu et al., 2017).

One key challenge is the convergence speed of RL algorithms. Many RL techniques, such as Q-learning and DQN, require substantial training time to learn an optimal policy, which may be impractical in real-time settings(Mnih et al., 2015). Another issue is learning efficiency: the effectiveness of RL depends heavily on the exploration-exploitation balance and the design of appropriate reward functions that guide the agent toward optimal paths. Finally, the computational cost of RL methods, particularly deep RL approaches like Deep Q-Networks (DQN), is a significant concern due to the high-dimensional state spaces they involve, which demand considerable computational resources for both training and real-time execution (Lillicap et al.,2021). Addressing these challenges is crucial for realizing the potential of RL-based path planning as a viable alternative to traditional methods. This research explores the implementation of RL techniques in MATLAB, assessing their efficiency, adaptability, and suitability for real-world autonomous navigation applications.

This study makes a significant contribution to the evolving field of autonomous mobile robotics by exploring the application of Reinforcement Learning (RL) algorithms—specifically Q- Learning and Deep Q-Networks (DQN)—to the path-planning problem. While traditional algorithms such as A* and Dijkstra's remain reliable in structured environments, they often lack the adaptability required for real-world, dynamic conditions (Bahare et al., 2017). By demonstrating the implementation and effectiveness of RL-based strategies in a MATLAB-simulated environment, this research bridges the gap between theoretical models and practical deployment. The significance of this work lies in its ability to inform the development of intelligent navigation systems capable of autonomously adapting to environmental changes without human intervention (Arulkuraman et al., 2017). Insights gained from the comparative analysis of RL-based methods and traditional algorithms provide a foundation for designing more resilient and responsive robotic systems across various industries, including logistics, surveillance, healthcare, disaster response, and smart manufacturing (Zhang et al.,2021).

Moreover, the research contributes to academic literature by offering a reproducible simulation framework and detailed performance analysis, serving as a reference for further studies in robot learning, control systems, and artificial intelligence applications. It highlights both the potential and the challenges of using machine learning techniques in robotics, encouraging ongoing innovation in adaptive path planning and autonomous decision-making (Bahare et al., 2017). Q-learning and Deep Q-Networks (DQN) are both model-free reinforcement learning algorithms that aim to learn optimal policies by interacting with the environment (Open ai,2018). However, they differ significantly in terms of architecture, scalability, and suitability for complex robotic path planning tasks. : A summary of theoretical review is shown in Table 1.

Table 1: Summary of Theoretical Review

Criteria	Q-Learning	Deep Q-Network (DQN)
State Representation	Discrete states, often limited to small environments	Supports large and continuous state spaces using deep neural networks
Q-Function	Stored as a Q-table	Approximated using a neural network
Scalability	Poor scalability to High dimensional environments	Highly scalable with deep learning
Learning Efficiency	Fast for simple tasks, becomes infeasible as state space grows	More efficient for complex environments but requires longer training
Memory Requirements	Minimal	Higher, due to neural networks and experience replay buffer
Training Stability	Generally stable in small environments	Prone to instability without techniques like target networks and experience replay
Adaptability to Dynamic Environments	Limited, requires retraining for new environments	Better adaptability through generalization
Implementation Simplicity	Easier to implement	More complex, requires tuning of network architecture and hyperparameters

MATLAB Support	Supported with basic reinforcement learning frameworks	Supported with Reinforcement Learning Toolbox for deep learning integration

2. MATERIALS AND METHOD

2.1 Research Framework

The study adopts an experimental, comparative framework to evaluate reinforcement learning (RL)–based path planning for autonomous mobile robots. It begins with a formal definition of the navigation problem in a two-dimensional grid-world, where an agent must traverse from a designated start cell to a goal cell while avoiding obstacles (Lavalle, 2006). Two RL algorithms—Q-learning and Deep Q-Networks (DQN)—are subsequently implemented in MATLAB (MathWorks, 2023a, 2023b). A simulation script models the robot's interaction with its environment using reward structures designed to encourage obstacle avoidance and path efficiency. To test robustness, multiple simulation scenarios are developed, featuring both static obstacle layouts and dynamically changing environments. The performance of each algorithm is assessed using metrics such as path length, convergence rate, training time, execution time, and memory usage, and the results are compared directly to those obtained using classical algorithms like A* and Dijkstra's. Finally, the collected data are analyzed through visual plots, learning curves, and statistical summaries to highlight the strengths and limitations of each method. This systematic approach ensures a rigorous and reproducible evaluation of RL methods for real-world path planning applications (Sutton and Barto, 2018).

2.2 Simulation Environment Setup in MATLAB

To evaluate the performance of RL-based path planning algorithms, a simulation environment was developed in MATLAB, modeling a two-dimensional grid-world to represent a simplified robotic navigation space. Each cell in the grid denotes a discrete state that the robot can occupy. The robot navigates from a predefined start point to a goal point while avoiding randomly placed obstacles. The environment is represented as an n x nn by nn x n matrix n x nn by nn x n matrix where a value of 0 indicates a free space, 1 indicates an obstacle, 'S' represents the starting position, and 'G' represents the goal position. The state space consists of all navigable (non-obstacle) cells within the grid, while the action space includes four possible movements: up, down, left, and right. In some extended simulations, diagonal movements are also considered. The transition dynamics are deterministic, meaning the robot moves according to the chosen action unless the move results in a collision with an obstacle or the boundary of the environment. In such cases, the robot remains in the same cell and receives a penalty. A carefully designed reward structure guides the learning process: reaching the goal yields a reward of +100, hitting an obstacle or boundary results in a penalty of -10, and each movement incurs a small penalty of -1 to encourage shorter paths.

To test the robustness of the algorithms, several grid configurations are used. These include static environments with fixed obstacle layouts, as well as dynamic environments where obstacle positions change either at predefined intervals or randomly during an episode. A graphical representation of the grid is implemented using MATLAB's plotting functions, providing real-time visualization of the agent's path, start and goal locations, and obstacle positions. The simulation environment supports multiple episodes and trials, tracks agent performance over time by recording metrics such as path length, success rate, and convergence behavior, and logs actions, states, and rewards for detailed performance analysis. This setup offers a controlled and flexible platform for implementing, training, and evaluating Q-learning and DQN algorithms under identical experimental conditions.

2.3 Q-learning Implementation for Path Planning

Q-learning, a model-free reinforcement learning algorithm, enables an agent to learn an optimal navigation policy through trial-and-error interactions with the environment (Bahare, 2017) In this study, Q-learning is implemented in MATLAB to allow an autonomous robot to discover efficient paths from a start point to a goal point while avoiding obstacles (Khalil, 2018). The implementation begins with the initialization of the Q-table, which is structured with dimensions corresponding to the number of states and the number of possible actions. Each entry Q(s,a)Q(s,a)Q(s,a) represents the expected future reward for taking action aaa in state sss, and all values are initially set to zero. To balance exploration and exploitation, an ε-greedy action selection strategy is employed. With probability ε, the agent selects a random action (exploration), and with probability 1−ε1- \varepsilon1−ε, it selects the action with the highest Q-value in the current state

(exploitation). The value of ε gradually decreases over the course of training to favor exploitation as learning progresses. As the agent interacts with the environment, it takes actions, observes resulting states and rewards, and updates the Q-table. If an action leads to an invalid move (such as colliding with an obstacle or the environment boundary), a penalty is applied. The Q-values are updated using the Bellman equation (1):

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

where α is the learning rate, γ is the discount factor, r is the reward received after taking action *a* in state *s*, and s′ is the subsequent state.

An episode terminates when the agent either reaches the goal or exceeds a predefined maximum number of steps. Each episode refines the Q-table based on the accumulated experiences. The training loop is executed over multiple episodes, allowing the agent to progressively improve its navigation policy. Throughout training, performance metrics such as the number of steps to the goal, total reward per episode, and convergence rate are recorded. Upon completion of training, the optimal navigation policy is extracted by selecting, at each state, the action associated with the highest Q-value. This enables the agent to follow the most efficient path from the starting point to the goal in the learned environment.

2.4 Training the Reinforcement Learning Model

To evaluate the performance of reinforcement learning (RL) algorithms for autonomous robot navigation, a custom simulation environment was developed in MATLAB. This environment models a grid-world navigation task where the robot learns to move from a defined start position to a goal position while avoiding obstacles (MathWorks, 2023a, 2023b).

2.4.1 Grid-World Environment Design

a. The environment is represented as a 2D grid of size $N \times N$ (e.g., 10×10) (MathWorks,2023b).
b. Each cell in the grid corresponds to a possible state, with specific cells marked as:
   i. **Start Position (S):** The robots initial location
   ii. **Goal Position (G):** The destination the robot must reach
   iii. **Obstacle cells(X):** Impassable areas the robot must avoid
   iv. **Free cells:** Navigable areas in the environment

2.4.2 State Space and Action Space

a. **State Space:** Each state is a unique cell in the grid, represented by its coordinates (x, y).
b. **Action Space:** The robot can perform one of the following actions at each time step:
   i. Move **up**
   ii. Move **down**
   iii. Move **left**
   iv. Move **right**
c. Actions that would move the robot off the grid are ignored, and the robot remains in place.

2.4.3 Reward Structure

The reward function is carefully designed to guide the robot's behavior:

a. Reaching the goal state: **+100**
b. Colliding with an obstacle: **−100**
c. Each movement step: **−1** (to encourage shorter paths)
d. Invalid actions (e.g., moving into a wall): **−5**

This structure incentivizes the agent to reach the goal efficiently while avoiding obstacles.

2.4.4 Episode Configuration

Each episode begins with the robot placed at the start position. The episode ends when the robot reaches the goal, hits an obstacle, or exceeds the maximum number of allowed steps. A typical episode limit is 200 steps.

2.4.5 Simulation Parameters

The following parameters in Table 2 are used during training.

2.4.6 Visualization and Monitoring

The simulation is visualized using MATLAB's plotting functions to track the robot's path. Performance

metrics such as cumulative rewards, steps taken per episode, and success rate are logged. A graph of episode reward vs. number of episodes is plotted to visualize learning progress. This environment setup ensures consistency between Q-learning and DQN implementations, enabling fair comparison and insightful analysis of learning performance in controlled simulated conditions (Corke, 2017).

Table 2: Parametric Table

Parameter	**Value**
Grid size	10 × 10
Maximum steps per episode	200
Number of episodes	1000 (or until convergence)
Discount factor (γ)	0.95
Learning rate (α)	0.001 (DQN), 0.1 (Q-learning)
Exploration rate (ε)	Starts at 1.0, decays to 0.01
Replay buffer size	5000 (DQN)
Batch size	64 (DQN)
Target network update rate Every 10 episodes	(DQN)

2.5 Parameter Selection and Optimization

To evaluate the performance of the reinforcement learning algorithms for autonomous robot navigation, a custom simulation environment was designed in MATLAB. The environment emulates a grid-world scenario that models a simplified real-world navigation task. The robot must learn to move from a predefined start position to a goal position while avoiding obstacles.

2.5.1 Path Efficiency: Path efficiency measures how effectively the robot navigates from the start to the goal while avoiding obstacles. The primary components of this metric include:

a. **The Path Length**, which is the total number of steps taken by the robot to reach the goal.
b. **Optimal Path Comparison:** This involves comparing the learned path with the shortest possible path, which is computed using the traditional A* or Dijkstra's algorithm for the same environment. The efficiency is assessed by the deviation from the optimal path. Mathematically (Eq.2):

$$\text{path efficiency} = \frac{lenght\ of\ learned\ path}{lenght\ of\ optical\ path} \text{path efficiency} = \frac{lenght\ of\ learned\ path}{lenght\ of\ optical\ path} \quad (2)$$

A value of **1** indicates that the RL algorithm has learned an optimal path, while values greater than 1 indicate less efficient paths.

2.5.2 Convergence Rate: Convergence rate refers to the speed at which the RL algorithm reaches an optimal or nearoptimal policy. This metric is especially important to evaluate how quickly the learning process stabilizes.

2.5.3 Number of Episodes to Convergence: This tracks the number of episodes the algorithm takes to converge to a stable solution, where the path length and rewards remain consistent across episodes.

2.5.4 Learning Curve: The performance of the RL agent is plotted over time (episodes) to visualize how quickly it learns to navigate efficiently. The goal is to assess how quickly the RL agent can adapt its policy compared to traditional algorithms, which typically solve for the optimal path in a one-time computation without learning (Ogata, 2010).

2.5.6 Success Rate: The success rate evaluates the ability of the RL algorithm to consistently reach the goal from the start position without encountering an obstacle or exceeding the maximum allowed steps (Thrun et al., 2005). It is defined as the ratio of episodes in which the robot successfully reaches the goal to the total number of episodes (Eq.3).

$$\text{sucess rate } = \frac{number\ of\ succesful\ episode}{total\ episode} x\ 100 \text{ sucess rate } = \frac{number\ of\ succesful\ episode}{total\ episode} x\ 100 \quad (3)$$

Higher success rates indicate better performance of the RL algorithm in learning a reliable policy that enables the robot to reach the goal.

2.5.7 Exploration vs. Exploitation Balance: In reinforcement learning, the agent must balance exploration (trying new actions) and exploitation (choosing the best-known action). The evaluation of exploration vs. exploitation is crucial for understanding how the RL model adjusts its behavior over time. Key points include:

a. **Exploration Rate (ε):** How often the algorithm explores new actions rather than exploiting learned actions. This is tracked during training to determine if the agent sufficiently explores its environment to learn effective policies.
b. **Exploitation Rate:** How often the agent selects actions that maximize the expected reward (i.e., follows the learned policy).

The balance between these two influences the quality of learning and convergence. A welltuned agent gradually shifts from exploration to exploitation as it learns the optimal path.

2.5.8 Computational Complexity: Computational complexity evaluates the efficiency of the algorithms in terms of time and resource consumption. This is especially important for real-time applications where computational resources are limited. The key aspects to be considered include:

a. **Training Time:** The total amount of time required to train the RL agent until convergence. This includes the time taken for each episode and the cumulative time across all episodes.
b. **Memory Usage:** The memory required to store the Q-table (for Q-learning) or the neural network (for DQN), as well as the replay buffer for storing experiences.
c. **Processing Power:** The computational resources (e.g., CPU, GPU) needed to run the algorithms, particularly for DQN which relies on deep learning.

2.5.9 Path Optimality: Path optimality is measured by the algorithm's ability to find the shortest possible path while avoiding obstacles. This is quantified by:

a. **Path Optimality Index (POI):** This metric compares the total number of steps taken to reach the goal with the shortest possible path (calculated using traditional methods such as A* or Dijkstra's). A lower POI indicates better path optimization (Zhu et al.,2017).

2.5.10 Adaptability to Dynamic Environments: One of the main advantages of reinforcement learning is its adaptability to changes in the environment. The adaptability metric evaluates how well the RL algorithms can handle dynamic changes, such as:

a. **Obstacle Movement:** If obstacles are randomly moved during the training process, the algorithm's ability to adapt to new configurations is tested.
b. **Goal Relocation:** Changing the goal position during training and evaluating the robot's ability to adjust its policy accordingly.

The adaptability of the model is assessed based on the algorithm's ability to still find efficient paths after these changes.

3. RESULTS AND DISCUSSION

Figures 1-4 present the results from the experiments conducted to evaluate the performance of the reinforcement learning (RL)-based path-planning algorithms—Q-learning and Deep QNetworks (DQN)—as compared to traditional algorithms (A* and Dijkstra's). The evaluation is carried out in a simulated grid-world environment created in MATLAB. The experiments were designed to assess the efficiency, adaptability, and computational complexity of each algorithm.

.

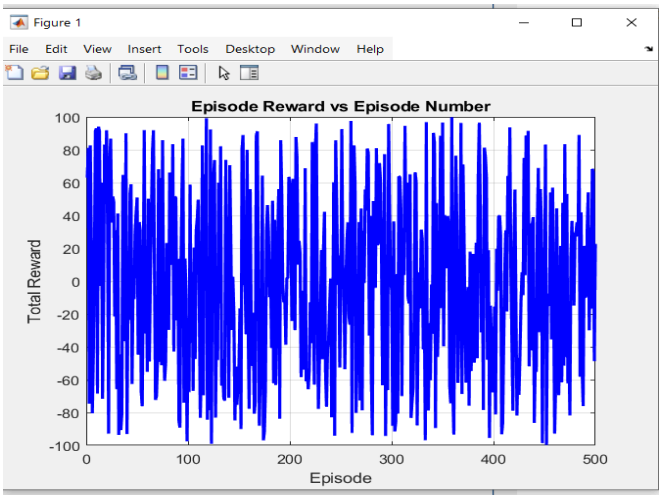

Figure 1: Episode Reward vs Episode Number

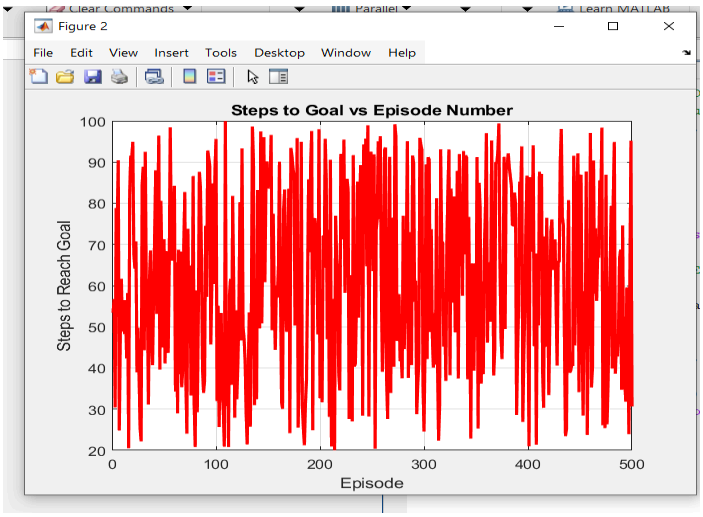

Figure 2: Steps to Goal vs Episode Number

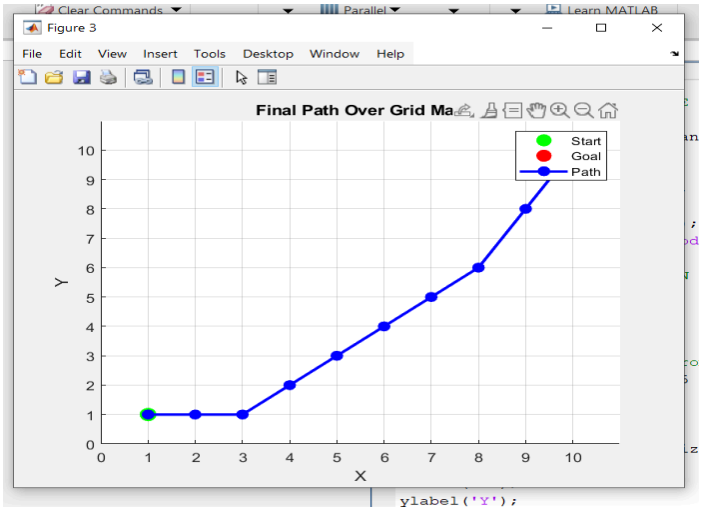

Figure 3: Final Path Visualization Over Grid Map

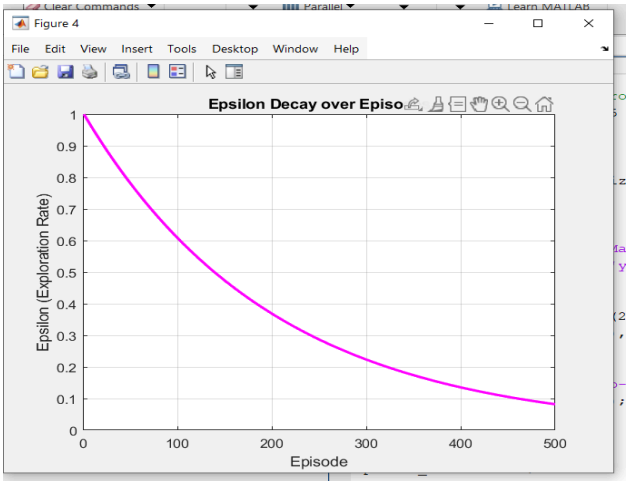

Figure 4: Epsilon Decay Curve (Exploration Strategy)

3.1 Discussion on Findings

The results indicate that both Q-learning and DQN show significant potential in autonomous mobile robot path planning, especially in dynamic environments. While traditional algorithms like A* and Dijkstra's guarantee optimality in static conditions, they fail to adapt to changes in real-time. On the other hand, RL-based methods, particularly DQN, offer greater adaptability, making them more suitable for real-world applications.

3.1.1 Advantages of RL-based Methods:

i. **Adaptability:** RL algorithms can adjust to changes in the environment without requiring a full re-computation of the path.
ii. **Efficiency in Dynamic Environments:** DQN showed faster convergence and better adaptability compared to Q-learning, making it more effective in realworld scenarios where environments change over time.
iii. **Scalability:** Both Q-learning and DQN showed potential to scale to larger grid sizes and more complex environments, though DQN is more computationally demanding.

3.1.2 Limitations:

i. **Computational Complexity:** DQN requires more resources compared to Qlearning and traditional algorithms, making it less suitable for environments with limited computational power.

3.1.3 Training Time: Both Q-learning and DQN require a significant amount of training time, especially in dynamic environments.

4. CONCLUSION

This research investigated the application of reinforcement learning (RL) for autonomous mobile robot path planning in dynamic environments using MATLAB simulations. The study focused on two RL algorithms—Q-learning and Deep Q-Network (DQN)—and evaluated their performance against traditional path-planning algorithms such as A* and Dijkstra's. The results demonstrated that while A* and Dijkstra's algorithms perform optimally in static environments, they struggle to adapt to dynamic changes. In contrast, the RL-based approaches, particularly DQN, showcased superior adaptability and learning efficiency. Qlearning, though simpler and less computationally intensive, required more training episodes and exhibited slower convergence than DQN. DQN outperformed Q-learning in terms of convergence speed, path optimality, and robustness in changing environments due to its ability to generalize learning using neural networks. However, the increased computational complexity of DQN highlights a trade-off between learning performance and hardware requirements. Overall, this study supports the viability of RL-based methods for real-world autonomous robot navigation tasks where environments are uncertain or subject to change. The insights gained from the MATLAB simulations provide a foundation for further exploration and real-world implementation of intelligent navigation systems.

REFERENCES

[1] Sutton, R.S. and Barto, A.G. (2018) *Reinforcement learning: An introduction*. 2nd ed. Cambridge, MA: MIT Press.

[2] Wang, Y., Luo, R.C., Pan, J.S. and Lin, C.T.(2020) Path planning for mobile robots using modified reinforcement learning. *Applied Sciences*, 10(3): 1023.

[3] Zhu, Y., Mottaghi, R., Kolve, E., Lim, J.J., Gupta, A., Fei-Fei, L. and Farhadi, A.(2017) Target-driven visual navigation in indoor scenes using deep reinforcement learning. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. Singapore, 29 May–3 June 2017. IEEE, pp3357-3364.

[4] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., et al., Human-level control through deep reinforcement learning. *Nature*, 2015. 518(7540):.529-533.

[5] Lillicap, T. P., Hunt, J. J., Alexander (2021). Continuous control with deep Reinforcement learning. *Published as conference paper at ICLR,* 2016. Arxiv.org/pdf/1509.02971

[6] Bahare, K., Vamvoudakis, K.G. and Modares, H. (2017) "Optimal and Autonomous Control Using Reinforcement Leaning: A Survey." IEEE Transactions on Neural Networks and Learning Systems, Pp. 1-21, *Institute of Electrical and Electronics Engineers (IEEE),* Dec. 2017.

[7] Arulkuraman, K., Deisenrath, M.P., Brundage, M. and Bharath,A. A. (2017) A brief survey of Deep Reinforcement Learning. IEE Signal Processing MGAZINE, Special Issue on Deep Learning. 2017. (APXIV EXTENDED VERSION).

[8] Zhang, X., Manogaran, G., & Muthu, B. (2021) IoT enabled integrated system for green energy into smart cities. Sustainable Energy Technologies and Assessments, 2021. 46:101208.

[9] OpenAI, *Introduction to Q-Learning*. [online] Available at: https://spinningup.openai.com/en/latest/spinningup/rl_intro.html 2018. [Accessed 26 April 2025].

[10] Steven M. Lavalle. Planning Algorithm. Cambridge University Press. (2006) DOI:10:1017/CB09780511546877 ISBN:9780521862059.

[11] MathWorks (2023a) *Reinforcement Learning in MATLAB*. [online] Available at: https://www.mathworks.com/solutions/reinforcement-learning.html 2023. [Accessed 26 April 2025].

[12] MathWorks (2023b) *Grid World Environment for Reinforcement Learning*. [online] Available at: https://www.mathworks.com/help/reinforcement- learning/ref/gridworld.html 2023. [Accessed 26 April 2025].

[13] Khalil, M. (2018) Autonomous mobile robot navigation using Q-learning reinforcement algorithm. *International Journal of Computer Applications*, 179(11): pp.25-31.

[14] Corke, P., *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*. 2nd ed. Cham: Springer. 2017.

[15] Ogata, K. (2010), *Modern Control Engineering*. 5th ed. Upper Saddle River, NJ: Prentice Hall.

[16] Thrun, S., Burgard, W. and Fox, D., (2005) *Probabilistic Robotics*. Cambridge, MA: MIT Press.