Journal of Robotics and Control (JRC)
Volume 6, Issue 5, 2025
ISSN: 2715-5072, DOI: 10.18196/jrc.v6i5.27579

Deep Q-Network-Based Path Planning in a Simulated Warehouse Environment with SLAM Map Integration and Dynamic Obstacles

Himandi Medagangoda [1*], Nilusha Jayawickrama [2], Rajitha de Silva [3],
U.U. Samantha Kumara Rajapaksha [4], Pradeep K.W. Abeygunawardhana [5]
[1] Faculty of Science and Engineering, Curtin University, Colombo, Sri Lanka
[2] School of Engineering, Aalto University, Espoo, Finland
[3] School of Engineering and Physical Sciences, University of Lincoln, Lincoln, United Kingdom
[4, 5] Faculty of Computing, Sri Lanka Institute of Information Technology, Malabe, Sri Lanka
Email: [1] himandi2003@gmail.com, [2] nilusha.jayawickrama@aalto.fi, [3] Odesilva@lincoln.ac.uk, [4] samantha.r@sliit.lk,
[5] pradeep.a@sliit.lk
*Corresponding Author

***Abstract*—With the rise of e-Commerce and the evolution of robotic technologies, the focus on autonomous navigation within warehouse environments has increased. This study presents a simulation-based framework for path planning using Deep Q-Networks (DQN) in a warehouse environment modeled with moving obstacles. The proposed solution integrates a prebuilt map of the environment generated using Simultaneous Localization and Mapping (SLAM), which provides prior spatial knowledge of static obstacles. The reinforcement learning model is formulated with a state space derived from grayscale images that combine the static map generated by SLAM and dynamic obstacles in real time. The action space consists of four discrete movements for the agent. A reward shaping strategy includes a distance-based reward and penalty for collisions to encourage goal-reaching and discourage collisions. An epsilon-greedy policy with exponential decay is used to balance exploration and exploitation. This system was implemented in the Robot Operating System (ROS) and Gazebo simulation environment. The agent was trained over 1000 episodes and metrics such as the number of actions executed to reach the goal and the cumulative reward per episode were analyzed to evaluate the convergence of the proposed solution. The results across two goal locations show that incorporating the SLAM map enhances learning stability, with the agent reaching a goal approximately 150 times, nearly double the success rate compared to the baseline without map information, which achieved only 80 successful episodes over the same number of episodes. This indicates faster convergence and reduced exploration overhead due to improved spatial awareness.**

Keywords*—Deep Q-Networks; Path Planning; Simultaneous Localization and Mapping; Robot Operating System; Gazebo Simulation.

I. INTRODUCTION

The rapid expansion of e-commerce in the recent years has driven a greater demand for automation in warehouse workflows to ensure that operations are carried out efficiently and securely [1]–[3]. As a result, automation technologies have rapidly evolved, contributing to benefits such as improved operational output and reducing costs through minimal labor [4]–[6]. The shared working space between humans and robots in modern warehouses requires careful management to avoid disruptions and ensure a smooth workflow [7], [8]. Therefore, mobile robots must be equipped with real-time path planning mechanisms that allow them to navigate dynamic environments with humans and operate autonomously.

Reinforcement Learning (RL) has emerged as a powerful technique for path planning due to its ability to learn from experiences. Deep Q-Networks (DQN), a RL algorithm, combines Q-learning with deep neural networks to approach problems with high-dimensional state spaces [9]. However, standard DQNs are known to have two main limitations. These are, the requirement for extensive exploration before meaningful learning occurs and slow convergence, especially in environments with large state spaces and sparse rewards [10], [11]. These limitations make standard DQNs less practical for real-world applications.

In this work, we aim to address these shortcomings by providing prior spatial knowledge in the form of a pre-built static map generated using simultaneous localization and mapping (SLAM) to the learning agent. The core hypothesis is that a SLAM-derived map can significantly reduce exploration overhead and accelerate convergence by allowing the agent to begin training with a structured understanding of the environment. While SLAM itself is widely used in robotic navigation, integrating it with a learning-based planner has not been thoroughly explored.

In this approach, the SLAM map is incorporated into the agent's state representation as a grayscale image. This image will include information on the availability of free space and the positions of static obstacles, allowing the agent to learn optimal paths with less exploration. Additionally, the reward function also makes use of the map to encourage proximity-based rewards which allows the DQN to associate environmental features more efficiently than in maples setups.

The evaluation is conducted entirely in a simulated environment with just two static goal positions due to hardware constraints that restricted extended training across more goal scenarios. The generalizability of the findings to real-world warehouse deployments and the system's performance to highly dynamic warehouse layouts and varied lighting remain to be validated. Despite the constraints, the setup was sufficient to demonstrate the effect of using SLAM maps on training efficiency and convergence. The results demonstrate that agents trained with the SLAM map achieved nearly double the number of successful goal completions compared to those without map input, enabling a comparison between mapped and maples path planning using a DQN framework.

The main contributions of this work are as follows;

- A DQN based path planning framework is implemented and evaluated for dynamic warehouse environments using a pre-built map generated via GMapping SLAM. While GMapping is a standard component in the ROS Navigation Stack, this study shows how prior knowledge in the form of a static map can reduce the exploration overhead in RL.
- A reward function tailored to indoor warehouse environments with dynamic agents is implemented, combining rewards for goal completion, proximity-based incentives, and collision penalties to guide the learning agent.

An experimental comparison of learning performance with and without SLAM map support in a simulated warehouse setting.

II. Ease of Use

A. Global and Local Planners

Path planning was categorized into two main types, global path planning and local path planning by Chik *et al.* [12]. Global path planners, such as Dijkstra [13], [14], and A* [15]–[18], compute the optimum route in static environments. Meanwhile, local path planners focus on avoiding dynamic hazards by allowing the robot to observe the environment and process real-time sensor inputs. This is also referred to as offline or online, respectively. Local planners include artificial potential field (APF), dynamic window approach (DWA), and reinforcement learning algorithms (RL). APF works with attraction and repelling theory, where it is attracted to the goal and repelled by an obstacle [19]–[21]. However, this method suffers from the local minima problem.

The DWA approach includes a set of feasible velocities. The algorithm selects a velocity from this set in the dynamic window and guides the agent to its goal. Although the agent reaches the goal faster using this approach, this method may fail in environments with high uncertainty or dynamic changes [22]–[25].

RL has emerged as a robust alternative for handling dynamic obstacles. The agent receives feedback for every action it performs. This feedback enables the robot to continuously learn which actions result in successful outcomes. Each positive action is rewarded while each negative or suboptimal actions are given a penalty [26]–[29]. Among the many RL techniques, Q- learning is one of the most studied algorithms. This method uses a Q-table for Q-values representation and hence is infeasible for high-dimensional states [30]–[32]. DQN was introduced in [33] to address this limitation by integrating artificial neural networks (ANN) with Q-learning. This could approximate Q-values with the use of the neural networks [34]–[36]. However, DQN suffers from overestimated Q-values and slow convergence [37]. To mitigate these problems, several DQN variants were introduced. Lei proposed Double DQN (DDQN) in [38] to overcome the problem of overestimation without any extra computational cost by decoupling action selection from value estimation [39]–[42]. Dueling DQN (D3QN) introduced in [43] enables the agent to estimate the more valuable state making it unnecessary to evaluate the impact of each action in all states. This accelerates the learning process and hence allows faster convergence [44]–[46]. Although the efficiency of this network is high, its implementation introduces additional complexity. Proximal Policy Optimization (PPO) offers training stability by using clipped objective functions and adaptive updates. PPO is suitable for complex navigation tasks due to its balance between sample efficiency and robustness [47]–[49]. Shuhuan Wen et al used D3QN together with active SLAM in [50] to improve navigation by creating the map of the environment using SLAM. However, this was limited to a static environment and was a highly complex algorithm. While these studies highlight the benefits of integrating SLAM with RL [51]–[53], most focus on mapping tasks rather than improving DQN convergence.

B. Research Gap

These limitations highlight a research problem where existing RL-based path planning solutions either require excessive exploration, suffer from slow convergence, or are highly complex algorithms that have not been extensively tested in dynamic warehouse environments with realistic constraints.

Our work builds on these efforts by demonstrating how incorporating a static SLAM-generated map as prior knowledge can reduce exploration overhead in a DQN-based navigation setup. This is evaluated in a dynamic warehouse simulation using ROS and Gazebo, with dynamic obstacles and two goal scenarios with a reward function specific for indoor environments.

III. Prepare Your Paper Before Styling

Using DQN, this research establishes a baseline for reinforcement learning with SLAM-generated maps. DQNs utilize a replay buffer to store their past experiences, allowing the agent to learn more effectively by sampling these [54]. Another notable advantage is that they can process high-dimensional inputs, including sensor data such as LiDAR [55]. This capability makes DQNs valuable in practical robotic systems. However, DQNs require significant computational power and often offer slow convergence. To mitigate this, the proposed solution incorporates a map generated via SLAM to allow the agent to navigate with prior spatial knowledge and avoid unnecessary exploration.

A. Scope and Assumptions

The objective of this study is to assess whether prior spatial knowledge via SLAM mapping can reduce exploration overhead in DQN-based learning. The simulation environment was selected to represent a structured warehouse scenario and includes dynamic agents but assumes a fixed static layout. Due to resource constraints, training was capped at 1000 episodes, which was empirically sufficient for convergence. This scope enables evaluation of map-assisted learning in a semi-realistic, reproducible context.

B. *ROS: Gazebo Warehouse Simulation*

ROS provides a collection of tools and libraries that are used in research and robust robot applications. Gazebo is a 3D robotic simulator that works alongside ROS, offering a versatile platform for testing and developing robot applications under realistic conditions [56], [57]. This solution was implemented and evaluated within the Gazebo simulation environment. For this study, a warehouse model was adopted from [58]. This environment offers a realistic warehouse setup, incorporating elements like shelving units, partitions, and dynamic elements. The robotic agent employed in this study is a Turtlebot3 (burger model) equipped with a 2D LiDAR sensor for environmental perception.

C. *Simultaneous Localization and Mapping (SLAM)*

SLAM is a primary technique in mobile robots that allows the robot to construct a map of its environment while simultaneously estimating its location within that space [59], [60]. In this study, SLAM is implemented by using the LiDAR on the Turtlebot3 robot. Fig. 1 shows the portable gray map as a result of SLAM. The white regions in Fig. 1 indicate the free navigable space. The gray area denotes the restricted area the robot should avoid. The black outlines correspond to the fixed static obstacles. To prepare this image in a format that can be processed by the DQN algorithm, the original map (.pgm) file of size 480×480 is first resized and then normalized to form a 200×200 grayscale image. This resized image is then fed into the neural network as input to the DQN.

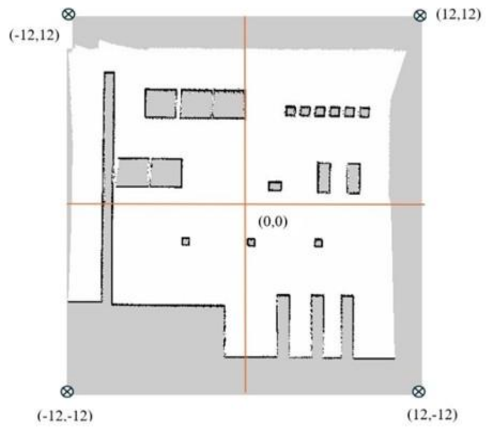

Fig. 1. Map of the warehouse created using 2D lidar using GMapping SLAM algorithm

D. *Coordinate System and Robot Localization*

To accurately interpret the occupancy grid map generated by SLAM and enable precise navigation, it is essential to establish a correspondence between the pixel-based map representation and real-world dimensions. Each pixel on the occupancy grid corresponds to a fixed real-world distance. The (1) shows the resolution of the SLAM-generated map.

$$Resolution = 0.05\, m/pixel \tag{1}$$

The total width and height of the occupancy grid are *480 pixels*, which is translated to real world dimensions using (2),

$$R\_D = N\_P \times RS \tag{2}$$

Where, R_D is the real-world dimensions of the warehouse, N_P is the number of pixels on the grid and RS is the resolution. Thus, the warehouse spans 24m×24m in real-world coordinates. The real-world coordinate corresponding to the *bottom-left* corner of the warehouse map is set at (–12, –12) meters. Therefore, the map can be interpreted as shown in Fig. 1.

Adaptive monte carlo localization (AMCL) was utilized to estimate the location of the agent in real-time. AMCL is a probabilistic localization algorithm that enables the robot to continuously estimate its pose in a known space using sensory data such as laser scans and odometry [61]–[63]. While the map server gives a static reference to localize the robot, AMCL actively updates the robot's estimated position within it. This ensures that, as the robot moves and senses the environment via LiDAR and odometry, its current position within the static map is continuously updated and corrected [63], [64]. Accurate localization is a critical requirement in autonomous navigation. Inaccurate position estimation will lead to ineffective decision- making [65], [66].

E. *Interpreting LiDAR Data and Detecting Obstacles*

Sensor readings are initially provided in polar form consisting of a distance and an angle relative to the robot's frame. These coordinates are transformed into Cartesian form for a grid-based representation. The ranges array containing the distance values has 720 elements. This suggests that the angle between consecutive readings is 0.25° (180/720 = 0.25). By applying trigonometric transformations to the measured angles and distances and using the angle increment from the LaserScan message, corresponding Cartesian (x, y) coordinate is calculated. The Cartesian coordinates representing the obstacles are scaled appropriately and mapped onto the 200×200 occupancy grid. Within the grid, the unoccupied space is denoted by 0, while the dynamic obstacles are denoted by 255. Overlaying the static map derived from SLAM with the dynamic obstacle map, a complete snapshot of the environment is created at that time. Every time the agent executes an action, this composite map will be updated and will serve as the current state input for the RL model.

F. *Deep Q-Network*

Mnih *et al.* introduced the concept of Deep Q-Leaning in [33]. This leverages deep neural networks to approximate Q val- ues in high-dimensional state spaces. In the proposed solution, the input to the Q-network is a combined occupancy map that represents static obstacles (from SLAM) and dynamic obstacles (from real-time LiDAR data). The DQN architecture consists of three primary components. These include a convolutional neural network (CNN), a replay buffer, and a target network.

1) Convolutional Neural Networks: CNNs are widely used in tasks involving image recognition and visual data processing [67]–[69]. In this study, a grayscale image of the map is used as the input, and hence a CNN with two convolutional layers is used to approximate the Q values. The first layer applies 32 filters of size 8×8 and a stride of 4 followed by a second layer consisting of 64 filters of size 4×4 and a stride of 2. This second layer captures additional abstract features and refines the output from the initial layer. Both layers use the Rectified Linear Unit (ReLu) for the activation function. The input to the DQN is the 200×200 grayscale image representing the presence of both static and dynamic obstacles, which also serves as the current state of the agent. The output layer of the network is fully connected and consists of neurons equal to the size of the action space. Each neuron outputs a Q-value for one possible action in the given state. The agent selects its next action depending on these Q-values. In this implementation, the action space includes four actions. Table I shows the actions allowed by the agent.

TABLE I. AGENT ACTIONS AND CORRESPONDING MOVEMENTS

Action	Description
Move Forward	Moves forward at a speed of 0.5 m/s
Stop	Sets both linear and angular speed to 0
Turn Right	Turns right by a random angle between 0° - 90°
Turn Left	Turns left by a random angle between 0° - 90°

G. Replay Buffer

The primary role of the replay buffer is to store the experiences/interactions of the agent with the environment enabling more accurate policy learning. Following an action, the experience is added to the buffer in the form of a tuple, *(s, a, r, s', t)* which includes the current state, chosen action, reward received, next state and termination flag, respectively [70], [71]. At each training step, a random set of past experiences of a predetermined size is sampled to mitigate temporal correlations in the data, enabling the agent to learn from a diverse set of previous experiences, stabilizing training [72], [73]. The replay buffer has limited capacity. Therefore, as experiences accumulate, the older experiences are overwritten with more refined experiences. This implementation employed a replay buffer with a capacity of holding 1000 experiences, which pro- vided a good trade-off between learning diversity and memory efficiency within the constraints of the simulation environment. A standard sampling batch size of 32 was used to balance convergence speed and stability.

H. Target Network

In addition to the main Q-network, a secondary target network is maintained to ensure stability in the target Q-value throughout the training phase. This network is updated at regular intervals [74]–[76]. Specifically, every 10 episodes, this network is updated with the primary network's weights to stabilize the training process. This frequency was chosen because more frequent updates could cause oscillations in Q- values, while less frequent updates can slow convergence within constraints of the simulation environment. The (3) [77], [78] shows how the target Q values are calculated using the Bellman equation,

$$Q(s,a) = r + (1 - t) \cdot \gamma \cdot max\, Q(s',a') \tag{3}$$

where, when the agent performs an action a in state s and receives a reward r, the corresponding Q value, $Q(s,a)$ for that state-action pair, reflects both the current reward r and the anticipated future reward in the subsequent state s'. Therefore, if an episode terminates and $t = 1$, the target Q value equals the immediate reward since no further state is encountered. Here, the discount factor, γ=0.99, controls the agent's emphasis on future rewards relative to immediate ones. Meanwhile, $maxQ(s',a')$ denotes the highest predicted Q-value for the next states estimated by the target network in the batch sampled. This value corresponds to the action that the agent identified as the one that will result in the highest future reward

1) Loss Function: To calculate the discrepancy between the predicted and target Q-values, the Huber loss function is used in this solution. This function combines the characteristics of the mean squared error (MSE) and the mean absolute error (MAE). It's low sensitivity to outliers makes it well-suited for stabilizing DQN models [79], [80]. The Huber loss function is defined as in (4) [81],

$$L_\delta\big(y, Q(s,a)\big) = \begin{cases} \frac{1}{2}\big(y-Q(s,a)\big)^2 & for\ |y-Q(s,a)| \le \delta \\ \delta|y-Q(s,a)|-\frac{1}{2}\delta^2 & otherwise \end{cases} \tag{4}$$

Where y denotes the target Q values, while $Q(s,a)$ represents the Q value predicted by the main network. The loss is minimized using the Adam optimizer with a learning rate of 1e-4. This learning rate is standard in DQN implementations and provides stable convergence for our solution without causing gradient instability or stagnation [82], [83].

2) Epsilon-Greedy Policy: This approach is used to maintain the balance between exploratory behavior and the exploitation of known actions during action selection. A random number is generated at each step. If this number is less than the current epsilon value (ϵ), a random action is selected (exploration). Otherwise, the action with the highest Q-value is chosen (exploitation) [84]–[86]. At the beginning of training, to understand the environment, the agent prioritizes exploration. As learning progresses, the agent relies on exploitation to exploit what it has learned. In this implementation, the epsilon is initially set to 0.999 and reduces exponentially to a lower limit of 0.01. The values were selected to prevent premature convergence and to accommodate the limited number of training episodes in this study. Once this minimum is reached, it is maintained to ensure that the agent continues to explore occasionally, even if the likelihood is low.

3) Reward Function: The reward function is designed to encourage the agent to take optimal path (least steps) to the goal by rewarding and discourage collisions and unsafe behavior by penalizing. In this solution, the reward function is designed such that, for each action taken, the agent receives a scalar reward based on one of three conditions: reaching the goal, approaching the goal, or colliding with an obstacle.

- Goal Reward: awarded when the agent is near the goal within a radius of 0.4m.
- Distance-based reward: awarded when the agent moves closer to the goal but has not reached it. The reward is scaled by how close the agent is to the goal.
- Collision penalty: awarded when the agent gets within a 0.3m radius of an obstacle.

These components are mutually exclusive, and only one reward type is applied per step. The reward for reaching the target is defined as shown in (5),

$$GR = MGR \times 1\frac{S\_T}{M\_S} + 100 \tag{5}$$

where GR is the reward for reaching the goal threshold, MGR=500, the maximum reward for reaching the goal, S_T is the number of steps from the initial position to the goal, and M_S=100, the maximum allowed steps. The reward is inversely scaled based on the step count used to achieve the target. This function encourages the agent to reduce the step count to the goal and promote efficiency in path selection. An offset of value of 100 ensures that the agent will receive a non-zero reward even when the target is achieved in the last step (100th).

If the goal has not yet been reached, the agent receives a distance-based reward that is defined as in (6)

$$R = 100 \times (1 - N\_D) \times 1\frac{S\_T}{M\_S} \tag{6}$$

where N_D is the normalized distance to the goal,

$$N_D = \frac{Dist}{max\ (Dist)} \tag{7}$$

Here, R is the reward, $Dist$ is the current Euclidean distance to the goal, and $max(Dist)$=12 m is the maximum distance in the environment.

A penalty of –1 is awarded when the agent comes within 0.3m of a static or dynamic obstacle, indicating a collision. This negative reward allows the agent learn to avoid unsafe/incorrect interactions with its environment.

The threshold values and reward functions were tuned empirically to achieve stable convergence in simulation and guided by the standard RL reward principles of goal proximity, efficiency, and penalty for wrong behavior of the agent.

I. An Episode

At the beginning of each episode, the agent is placed in the fixed initial position (0, 0) with a default orientation. The consistent initialization was intended to simplify early convergence and enable clearer comparisons between episodes and between the two scenarios (with and without map). The agent then begins selecting actions using the ϵ-greedy strategy described previously. The agent is allowed to execute a maximum of 100 actions in each episode. An episode terminates earlier if any of the following criteria are met:

- Goal Reached: If the agent successfully reaches the target within 100 actions or fewer, the episode terminates.
- Collision: In the event of a collision due to an action taken, the agent is penalized with a negative reward, and the episode terminates immediately.
- Action limit exceeds: If the agent reaches a maximum of 100 actions and has neither reached the goal nor encountered a collision, the episode ends.

Algorithm 1. DQN with SLAM Map Integration

```
Initialize: replay_buffer, Q-network $Q$, target
  network $Q_{target}$, exploration rate $\epsilon$
for each episode do
    Initialize robot position and environment
    static_map ← preprocess(SLAM _occupancy_grid)
    for each step in the episode do
        dynamic_map ← process_lidar()
        combined_map ← max(static_map, dynamic_map)
        s ← combined_map
        // Epsilon-greedy action selection
  With probability $\epsilon$, select a random action a
        Else, a ← arg max$_{a'}$ $Q(s, a')$
        Execute action a
        reward r ← reward_function
  Get next state s', and termination flag t
        Store (s, a, r, s', t) in replay_buffer
  Sample a mini-batch from replay_buffer
        target y ← Equation(3)
        Update Q
        s ← s'
        Every 10 episodes, update target network: $Q_{target}$ ←Q
        Decay $\epsilon$
        if goal reached or collision or max steps then
          break
        end if
    end for
end for
```

The algorithm 1 summarizes how the SLAM is integrated with the DQN in this solution and how each episode works. Fig. 2 visualizes a high-level workflow of the proposed solution using a flowchart. This shows what happens in a single episode.

During this training stage, a basic curriculum learning strategy was used to stabilize training [87], [88]. Here, the initial goal was placed closer to the agent's initial position. Once the agent consistently reached this goal five times, a more distant goal was introduced. This incremental learning strategy allowed the agent to first master simpler navigation tasks before attempting more complex ones. The threshold of using five successes was chosen to maintain a balance between the time taken to train and the learning confidence. A lower threshold resulted in the agent progressing on the basis of coincidental success rather than consistently learning. Meanwhile, a higher threshold risked overfitting to the current goal and significantly increased training time, which is a critical factor given our computational constraints.

Fig. 3 shows the target position as a red sphere and the agent as the base link as visualized in Rviz. The episodes, the cumulative reward, the average reward, the maximum loss, the average loss, the target coordinate, the goal reach count was recorded in a CSV file.

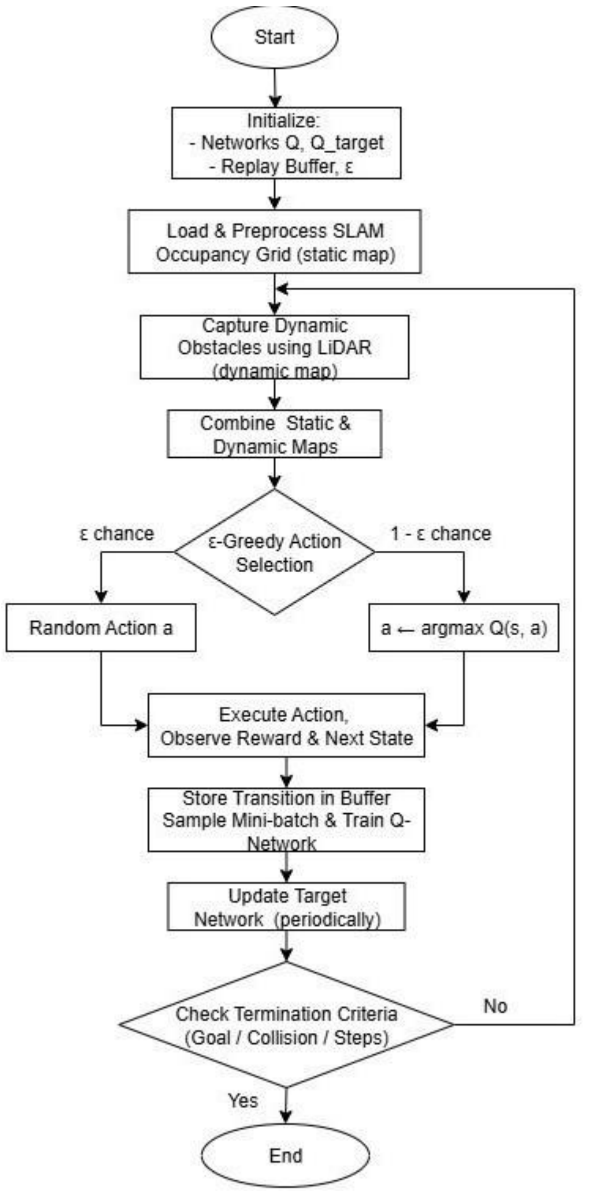
Start
Initialize:
- Networks Q, Q_target
- Replay Buffer, ε
Load & Preprocess SLAM Occupancy Grid (static map)
Capture Dynamic Obstacles using LiDAR (dynamic map)
Combine Static & Dynamic Maps
ε-Greedy Action Selection
ε chance
1 - ε chance
Random Action a
a ← argmax Q(s, a)
Execute Action, Observe Reward & Next State
Store Transition in Buffer Sample Mini-batch & Train Q-Network
Update Target Network (periodically)
Check Termination Criteria (Goal / Collision / Steps)
No
Yes
End

Fig. 2. Flowchart: A single episode

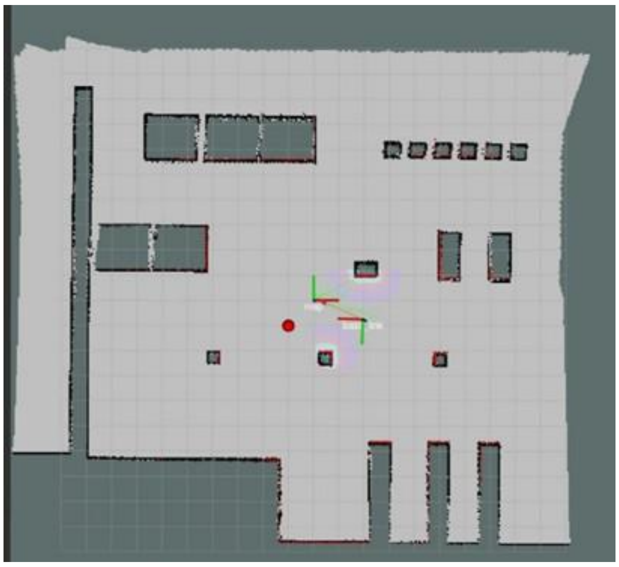

Fig. 3. Rviz Visualization of the agent navigating in the warehouse

J. Comparative Evaluation

To evaluate the impact of integrating a SLAM-generated map, a comparative analysis was conducted between two DQN models, one where the map was provided and other trained without any prior knowledge of the static layout. The results showed that the agent reached nearly twice as many goals (150 vs. 80) over the same training period, demonstrating faster convergence. The results will be discussed further in the next chapter. This comparative result provides preliminary evidence of the contribution of spatial priors. In future work, a detailed ablation analysis is expected to be performed to further quantify individual contributions to learning performance.

IV. EXPERIMENTAL CONSTRAINTS AND LIMITATIONS

The implementation of this solution requires a large amount of computational power and memory for several reasons. These include processing high-resolution maps in Gazebo, processing a large amount of LiDAR data, high-dimensional input states (200×200), processing a large amount of batch data during training from the replay buffer, and the replay buffer itself consumes a large amount of memory. Simultaneously running the simulation, neural network training, and ROS processes require a high-performance CPU and a high amount of RAM to maximize performance [89], [90]. A 64 GB RAM and a 12- core CPU might allow the system to run efficiently. Due to these computational constraints, the model was trained for 1000 episodes, and the experimental setup was restricted to a single warehouse layout with two goals. Although this limited the generalizability of the findings, the inclusion of both static and dynamic obstacles in this layout provided insights to the robot's learning behavior in realistic environments.

V. RESULTS AND DISCUSSION

Following the method discussed above, the model was trained for 1000 episodes. Using these results, we aim to assess if integrating a pre-built SLAM map improves convergence compared to a baseline of when a map is not used together with DQN to navigate to the goal.

A. Constructing the Slam Map and Localizing

Fig. 1 shows the map constructed using the GMapping SLAM algorithm and the LiDAR sensor. The odometry information gives the agent's current pose relative to the starting point at (0,0), enabling the agent to determine its location on the map.

B. Training the DQN Model

The DQN agent underwent training for a total of 1000 episodes, and the training parameters used are shown in Table II. The hyperparameter value choices and their empirical rationale are explained in Section 3.

TABLE II. PARAMETERS AND THEIR CORRESPONDING VALUES

Parameter	Value
Maximum number of steps/actions per episode	100
Initial epsilon	0.999
Minimum epsilon	0.01
Discount Factor	0.99
Learning rate	1e-4
Size of replay buffer	1000
Sample batch size	32

The learning process was evaluated based on two primary metrics:

- Evaluating step count to goal during successful episodes
- Evaluating cumulative rewards over total training episodes

1) Measuring steps taken to reach to goal:

This metric can be used to evaluate the training process. This offers a reliable indicator of the agent's learning progress during training. Fig. 4 generated using matplotlib presents the number of steps taken to reach the goal plotted against the corresponding successful episodes. In the initial training phase, the agent has very limited knowledge of the environment in which it is navigating. There- fore, the agent explores the environment by randomly executing actions. As a result, the agent will require more actions/steps to reach the target early on. As learning progresses, the agent becomes familiar with the environment, hence learning to take more purposeful actions.

This results in a reduced number of actions taken by the agent to reach the target. This deceasing trend in the number of steps taken indicates that the policy learned by the agent is becoming more efficient and optimized. During the training period, the agent successfully reached the target approximately 150 times. Initially, the agent required around 65 steps (high number) to reach the goal, reflecting its lack of environmental knowledge. The fluctuations seen following the sudden decrease suggest that the agent is continuing to explore, leading the agent to occasionally select random actions. The plot illustrates that the agent consistently reached the 1^{st} goal after about 101 successful attempts. A new goal is introduced only after the agent has reached the previous goal consecutively five times. This is to ensure that the behavior has been sufficiently learned. By the time the agent had mastered the first goal, it only took approximately 25 steps to reach that goal.

A sharp increase in the step count is seen around the 101^{st} successful episode, corresponding to the introduction of the second goal. Similarly to the initial target, the agent begins with a larger step count and gradually reduces as it improves the policy. Fluctuations in the step count have reduced over time as the agent progresses to the second goal, indicating the shift from exploration to exploitation. This behavior suggests that the agent is refining its policy and learning the optimal path. With extended training, the agent will also learn to reach the 2^{nd} goal using the optimal path. Therefore, Fig. 4 is an effective representation of the agent's learning trajectory.

2) Baseline comparison: DQN with vs. without SLAM Map:

To assess the impact of prior spatial knowledge on learning efficiency, the same DQN model was trained without access to the SLAM-generated map. Fig. 5 shows the steps taken by the robot to reach the goal when the map is not provided. In this plot, it can be seen that there were only about 80 successful episodes, while there were about 150 when the map was used. It can also be seen that the agent struggled to learn a coherent path-planning strategy, as it did not reliably reach even the first goal during the course of training, exhibiting slow convergence. This comparative experiment further validates the use of the SLAM map to improve convergence by accelerating the learning process. The results of the comparative experiment, with and without the SLAM map are presented in Table III.

TABLE III. COMPARISON OF DQN PERFORMANCE WITH AND WITHOUT SLAM MAP

Metric	With SLAM Map	Without Map
Average steps to reach goal	27.6	58.5
Total successful episodes (out of 1000)	150	85
Average cumulative reward	1700	3000
Average collisions per 100 episodes	~3.4	~17

The results for the Average cumulative reward and Average collisions per 100 episodes are explained in the 'Measuring Cumulative Reward' section.

3) Epsilon Decay plot:

Fig. 6 illustrates the exponential decay of the epsilon value during training. This decay follows the ϵ-Greedy Policy discussed in Section III, allowing more frequently exploration initially and gradually allow more exploitation as learning progresses by relying on the predictions of the Q-network. This decrease also accounts for the reduced fluctuations in Fig. 4, as the agent begins to favor more consistent, reward-driven behavior.

4) Measuring Cumulative Reward:

Fig. 7 shows the cumulative reward plotted against the number of episodes. In the proposed solution, the agent receives a significant reward upon reaching a goal, inversely scaled by the number of actions required to reach it, as shown in the (5). Additionally, the agent is also incrementally rewarded when it moves closer to the goal, as seen in the (6). As a result of this stepwise distance- based reward, episodes in which more steps are involved tend to accumulate a higher total reward. Therefore, the cumulative reward is greater in the initial training episodes when the agent takes longer paths to reach the goal. As training progresses, the agent understands to follow the optimal path, reducing the step count, and hence the stepwise intermediate reward. This results in a decrease in the cumulative reward, suggesting that the learned policy is becoming more efficient. The noticeable fluctuations in the beginning of the plot, likely caused by frequent collisions during initial exploration, become less prominent towards the end, indicating that the agent is learning progressively to avoid collisions. Comparing the scenarios, with and without the SLAM generated map, the average cumulative reward is higher in the without-map setup as a consequence of the agent taking more steps per episode to reach the goal. In contrast, with the SLAM map provided, the agent learns a more optimal policy earlier with the prior spatial knowledge and takes fewer steps, resulting in lower accumulated intermediate rewards per episode. This explains the values shown in Table III.

The downward spikes in Fig. 7 represent collisions. As soon as a collision occurs, the reward becomes minimal. With the map incorporated, the collisions gradually decrease to an average of 3.4 per 100 episodes within the training period.

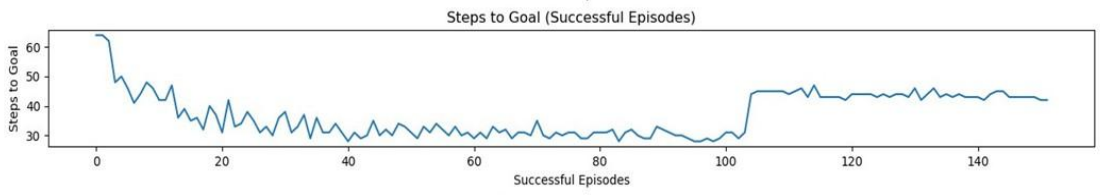

Fig. 4. Number of steps to reach the goal per successful episode (with SLAM map)

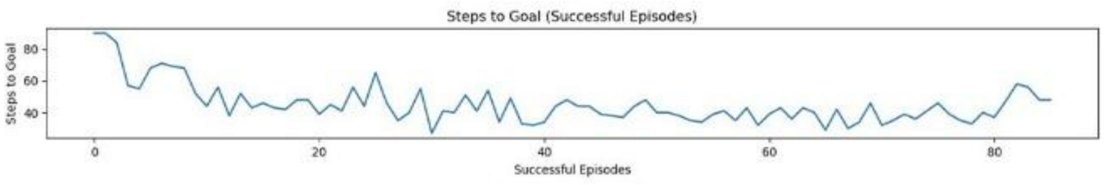

Fig. 5. Number of steps to reach the goal per successful episode (no SLAM map)

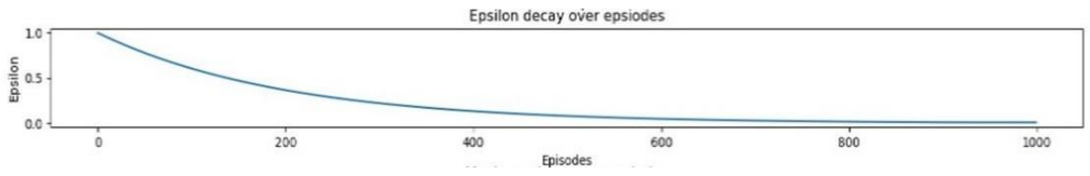

Fig. 6. Epsilon decay

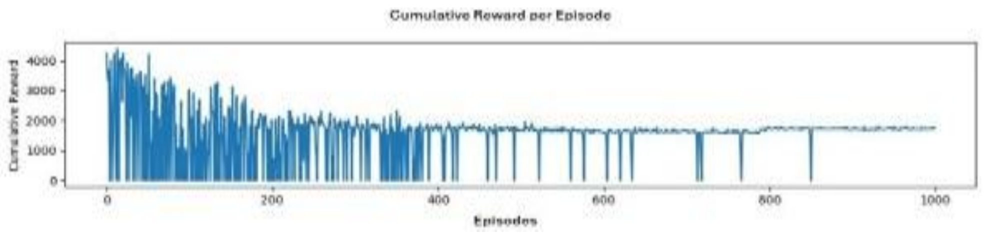

Fig. 7. Cumulative reward

C. Using the Trained Model

The trained DQN model was deployed in the same warehouse simulation to evaluate its real-time navigation behavior. The agent successfully learned to navigate to the first goal (5, -0.6). The action selection process during deployment followed the learned policy without exploration, that is, the action with the maximum predicted Q-value was selected at each step. Fig. 8 shows how the agent successfully reached the first target.

For instance, Moraes *et al.* [42] evaluated both DQN and DDQN with only low-dimensional Laser and goal-relative inputs and found that DDQN improved success rates but still took long to converge and stabilize. In contrast, our SLAM-integrated DQN achieved nearly double the number of successful goal completions (150 vs. 80) in the same training window, with a 40% reduction in steps to reach the goal.

The main findings of this study demonstrate that integrating SLAM-generated static maps with Deep Q-Networks significantly improves convergence speed and navigation efficiency in static warehouse environments with dynamic obstacles. The agent using prior spatial knowledge achieved nearly twice the success rate compared to the baseline DQN without map input. This study demonstrates that providing structured spatial context can enable faster learning in complex navigation tasks, making such methods more viable for real-time or resource-constrained robotic deployments. While the study shows promising results, limitations include the restricted number of goal positions and lack of dynamic layout testing, which limit the generalizability of the results and can be addressed in future work to strengthen the reproducibility and scalability of the proposed method.

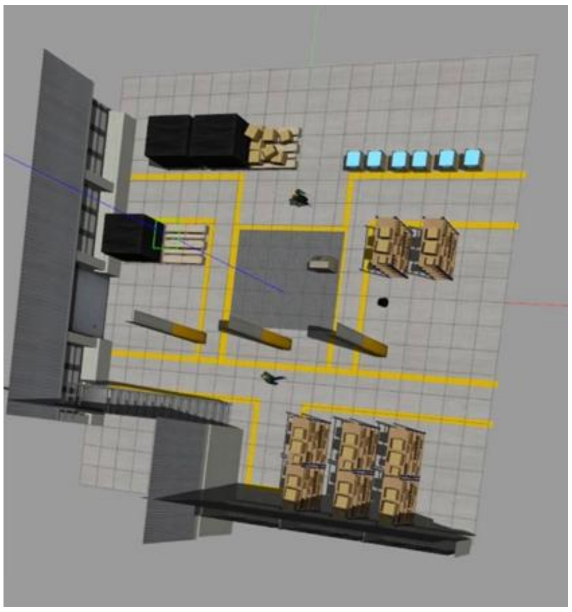

Fig. 8. Agent successfully reaching first goal

VI. Conclusion and Future Work

This study presented a Deep Q-Network (DQN)-based path planning framework for autonomous warehouse robots, integrating a SLAM-generated map to provide prior spatial knowledge aiming to reduce exploration overhead and contribute to more efficient policy learning.

The proposed approach was implemented and analyzed in a simulated Gazebo warehouse environment consisting of static and dynamic obstacles.

The research contribution lies in demonstrating that incorporating a SLAM-derived static map as a spatial prior converges faster to an optimal navigation policy by reducing exploration overhead while maintaining performance in dynamic conditions.

The scope of evaluation was limited to a relatively short training duration with two goal locations due to computational resource constraints affecting the generalizability of findings. Although a complete hyperparameter sensitivity analysis was beyond the scope of the current study due to computational constraints, we recognize this as an important direction for future work to further enhance reproducibility and robustness.

Furthermore, while the SLAM map helps reduce exploration in relatively stable environments, the system may under-perform if the environment diverges significantly from the static map, for example, in high-density human- robot interaction zones or environments with sudden structural changes. Real-world deployment would also require addressing the computational demands of combining high-dimensional SLAM maps with deep learning in real-time systems.

However, our study demonstrates promising results to improve the stability of policy learning in constrained warehouse settings using prior spatial knowledge. This method contributes new insights and highlights the potential of map-assisted RL in robotics.

Future work will include extending the model to handle multiple warehouse layouts and dynamic goal locations, incorporating domain randomization to improve robustness.

Author Contributions

Himandi Medagangoda led the research, implemented the methodology, performed the simulations, prepared, and reviewed the manuscript. Nilusha Jayawickrama provided technical guidance and critically reviewed the manuscript. Rajitha de Silva and U.U. Samantha Kumara Rajapaksha contributed to reviewing the manuscript. Pradeep K.W. Abeygunawardhana supervised the project and provided overall direction. All authors reviewed and approved the final version of the manuscript.

Acknowedgments

The authors thank the Sri Lanka Institute of Information Technology and the Arthur C. Clarke Institute for providing resources to train the model.

References

[1] Y. Li, R. Zhang, and D. Jiang, "Order-Picking Efficiency in E-Commerce Warehouses: A Literature Review," *Journal of Theoretical and Applied Electronic Commerce Research*, vol. 17, no. 4, pp. 1812–1830, 2022, doi: 10.3390/jtaer17040091.

[2] K. Ellithy, M. Salah, I. S. Fahim, and R. Shalaby, "AGV and Industry 4.0 in warehouses: a comprehensive analysis of existing literature and an innovative framework for flexible automation," *International Journal of Advanced Manufacturing Technology*, vol. 134, no. 1–2, pp. 15–38, 2024, doi: 10.1007/s00170-024-14127-0.

[3] A. A. Tubis and J. Rohman, "Intelligent Warehouse in Industry 4.0—Systematic Literature Review," *Sensors*, vol. 23, no. 8, p. 4105, 2023, doi: 10.3390/s23084105.

[4] A. R. Khairuddin, M. S. Talib, and H. Haron, "Review on simultaneous localization and mapping (SLAM)," in *Proceedings - 5th IEEE International Conference on Control System, Computing and Engineering, ICCSCE 2015*, pp. 85–90, 2016, doi: 10.1109/ICCSCE.2015.7482163.

[5] A. Jarašūnienė, K. Čižiūnienė, and A. Čereška, "Research on Impact of IoT on Warehouse Management," *Sensors*, vol. 23, no. 4, p. 2213, 2023, doi: 10.3390/s23042213.

[6] J. T. Licardo, M. Domjan, and T. Orehovački, "Intelligent Robotics—A Systematic Review of Emerging Technologies and Trends," *Electronics (Switzerland)*, vol. 13, no. 3, p. 542, 2024, doi: 10.3390/electronics13030542.

[7] F. Jacob, E. H. Grosse, S. Morana, and C. J. König, "Picking with a robot colleague: A systematic literature review and evaluation of technology acceptance in human–robot collaborative warehouses," *Computers and Industrial Engineering*, vol. 180, p. 109262, 2023, doi: 10.1016/j.cie.2023.109262.

[8] C. Scholz *et al*., "Sensor-enabled safety systems for human–robot collaboration: A review," *IEEE Sensors Journal*, vol. 25, no. 1, pp. 65–88, 2024, doi: 10.1109/JSEN.2024.3496905

[9] S. Ramesh, S. B. N, S. J. Sathyavarapu, V. Sharma, N. K. Nippun, and M. Khanna, "Comparative analysis of Q-learning, SARSA, and deep Q-network for microgrid energy management," *Scientific Reports*, vol. 15, no. 1, p. 694, 2025, doi: 10.1038/s41598-024-83625-8.

[10] C. Chen, J. Yu, and S. Qian, "An Enhanced Deep Q Network Algorithm for Localized Obstacle Avoidance in Indoor Robot Path Planning," *Applied Sciences (Switzerland)*, vol. 14, no. 23, p. 11195, 2024, doi: 10.3390/app142311195.

[11] X. Zhang, "Improving exploration efficiency of deep reinforcement learning in sparse-reward environments," *Expert Systems with Applications*, vol. 185, p. 115529, 2021.

[12] S. F. Chik, C. F. Yeong, E. L. M. Su, T. Y. Lim, Y. Subramaniam, and P. J. H. Chin, "A review of social-aware navigation frameworks for service robot in dynamic human environments," *Journal of Telecommunication, Electronic and Computer Engineering*, vol. 8, no. 11, pp. 41–50, 2016.

[13] S. Alshammrei, S. Boubaker, and L. Kolsi, "Improved Dijkstra Algorithm for Mobile Robot Path Planning and Obstacle Avoidance," *Computers, Materials and Continua*, vol. 72, no. 3, pp. 5939–5954, 2022, doi: 10.32604/cmc.2022.028165.

[14] X. Zhou, J. Yan, M. Yan, K. Mao, R. Yang, and W. Liu, "Path Planning of Rail-Mounted Logistics Robots Based on the Improved Dijkstra Algorithm," *Applied Sciences (Switzerland)*, vol. 13, no. 17, p. 9955, 2023, doi: 10.3390/app13179955.

[15] X. Xu, J. Zeng, Y. Zhao, and X. Lü, "Research on global path planning algorithm for mobile robots based on improved A*," *Expert Systems with Applications*, vol. 243, pp. 2321–2334, 2024, doi: 10.1016/j.eswa.2023.122922.

[16] Y. Bai, G. Li, and N. Li, "Motion Planning and Tracking Control of Autonomous Vehicle Based on Improved A∗ Algorithm," *Journal of Advanced Transportation*, vol. 2022, no. 1, p. 1675736, 2022, doi: 10.1155/2022/1675736.

[17] B. Fu *et al*., "An improved A* algorithm for the industrial robot path planning with high success rate and short length," *Robotics and Autonomous Systems*, vol. 106, pp. 26–37, 2018, doi: 10.1016/j.robot.2018.04.007.

[18] A. Chatzisavvas, M. Dossis, and M. Dasygenis, "Optimizing Mobile Robot Navigation Based on A-Star Algorithm for Obstacle Avoidance

in Smart Agriculture," *Electronics (Switzerland)*, vol. 13, no. 11, p. 2057, 2024, doi: 10.3390/electronics13112057.

[19] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 500–505, 1985, doi: 10.1109/ROBOT.1985.1087247.

[20] W. Zhang, N. Wang, and W. Wu, "A hybrid path planning algorithm considering AUV dynamic constraints based on improved A* algorithm and APF algorithm," *Ocean Engineering*, vol. 285, p. 115333, 2023, doi: 10.1016/j.oceaneng.2023.115333.

[21] J. Gao, X. Xu, Q. Pu, P. B. Petrovic, A. Rodic, and Z. Wang, "A Hybrid Path Planning Method Based on Improved A∗ and CSA-APF Algorithms," *IEEE Access*, vol. 12, pp. 39139–39151, 2024, doi: 10.1109/ACCESS.2024.3372573.

[22] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics and Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997, doi: 10.1109/100.580977.

[23] K. Qi, E. Li, and Y. Mao, "Dynamic Path Planning of Mobile Robot Based on Improved A* Algorithm and Adaptive DWA," *Shuju Caiji Yu Chuli/Journal of Data Acquisition and Processing*, vol. 38, no. 2, pp. 451–467, 2023, doi: 10.16337/j.1004-9037.2023.02.019.

[24] X. Bai, H. Jiang, J. Cui, K. Lu, P. Chen, and M. Zhang, "UAV Path Planning Based on Improved A ∗ and DWA Algorithms," *International Journal of Aerospace Engineering*, vol. 2021, no. 1, p. 4511252, 2021, doi: 10.1155/2021/4511252.

[25] W. Guan and K. Wang, "Autonomous Collision Avoidance of Unmanned Surface Vehicles Based on Improved A-Star and Dynamic Window Approach Algorithms," *IEEE Intelligent Transportation Systems Magazine*, vol. 15, no. 3, pp. 36–50, 2023, doi: 10.1109/MITS.2022.3229109.

[26] M. Naeem, S. T. H. Rizvi, and A. Coronato, "A Gentle Introduction to Reinforcement Learning and its Application in Different Fields," *IEEE Access*, vol. 8, pp. 209320–209344, 2020, doi: 10.1109/ACCESS.2020.3038605.

[27] M. Naeem, A. Coronato, Z. Ullah, S. Bashir, and G. Paragliola, "Optimal User Scheduling in Multi Antenna System Using Multi Agent Reinforcement Learning," *Sensors*, vol. 22, no. 21, p. 8278, 2022, doi: 10.3390/s22218278.

[28] W. Kumwilaisak, S. Phikulngoen, J. Piriyataravet, N. Thatphithakkul, and C. Hansakunbuntheung, "Adaptive Call Center Workforce Management With Deep Neural Network and Reinforcement Learning," *IEEE Access*, vol. 10, pp. 35712–35724, 2022, doi: 10.1109/ACCESS.2022.3160452.

[29] A. Alwarafy, M. Abdallah, B. S. Ciftler, A. Al-Fuqaha, and M. Hamdi, "The Frontiers of Deep Reinforcement Learning for Resource Management in Future Wireless HetNets: Techniques, Challenges, and Research Directions," *IEEE Open Journal of the Communications Society*, vol. 3, pp. 322–365, 2022, doi: 10.1109/OJCOMS.2022.3153226.

[30] Q. Zhou, Y. Lian, J. Wu, M. Zhu, H. Wang, and J. Cao, "An optimized Q-Learning algorithm for mobile robot local path planning," *Knowledge-Based Systems*, vol. 286, p. 111400, 2024, doi: 10.1016/j.knosys.2024.111400.

[31] Y. Lyu, A. Côme, Y. Zhang, and M. S. Talebi, "Scaling Up Q-Learning via Exploiting State–Action Equivalence," *Entropy*, vol. 25, no. 4, p. 584, 2023, doi: 10.3390/e25040584.

[32] N. Sutisna, A. M. R. Ilmy, I. Syafalni, R. Mulyawan, and T. Adiono, "FARANE-Q: Fast Parallel and Pipeline Q-Learning Accelerator for Configurable Reinforcement Learning SoC," *IEEE Access*, vol. 11, pp. 144–161, 2023, doi: 10.1109/ACCESS.2022.3232853.

[33] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015, doi: 10.1038/nature14236.

[34] J. Li, Y. Chen, X. N. Zhao, and J. Huang, "An improved DQN path planning algorithm," *Journal of Supercomputing*, vol. 78, no. 1, pp. 616–639, 2022, doi: 10.1007/s11227-021-03878-2.

[35] B. Varga, B. Kulcsár, and M. H. Chehreghani, "Deep Q-learning: A robust control approach," *International Journal of Robust and Nonlinear Control*, vol. 33, no. 1, pp. 526–544, 2023, doi: 10.1002/rnc.6457.

[36] T. Nakamura, M. Kobayashi, and N. Motoi, "Path Planning for Mobile Robot Considering Turnabouts on Narrow Road by Deep Q-Network," *IEEE Access*, vol. 11, pp. 19111–19121, 2023, doi: 10.1109/ACCESS.2023.3247730.

[37] R. Singh, J. Ren, and X. Lin, "A Review of Deep Reinforcement Learning Algorithms for Mobile Robot Path Planning," *Vehicles*, vol. 5, no. 4, pp. 1423–1451, 2023, doi: 10.3390/vehicles5040078.

[38] X. Lei, Z. Zhang, and P. Dong, "Dynamic Path Planning of Unknown Environment Based on Deep Reinforcement Learning," *Journal of Robotics*, vol. 2018, p. 10, 2018, doi: 10.1155/2018/5781591.

[39] X. Zhang, X. Shi, Z. Zhang, Z. Wang, and L. Zhang, "A DDQN Path Planning Algorithm Based on Experience Classification and Multi Steps for Mobile Robots," *Electronics (Switzerland)*, vol. 11, no. 14, p. 2120, 2022, doi: 10.3390/electronics11142120.

[40] A. Khlifi, M. Othmani, and M. Kherallah, "A Novel Approach to Autonomous Driving Using Double Deep Q-Network-Bsed Deep Reinforcement Learning," *World Electric Vehicle Journal*, vol. 16, no. 3, 2025, doi: 10.3390/wevj16030138.

[41] L. Chen, Q. Wang, C. Deng, B. Xie, X. Tuo, and G. Jiang, "Improved Double Deep Q-Network Algorithm Applied to Multi-Dimensional Environment Path Planning of Hexapod Robots," *Sensors*, vol. 24, no. 7, p. 2061, 2024, doi: 10.3390/s24072061.

[42] L. D. de Moraes *et al.*, "Double Deep Reinforcement Learning Techniques for Low Dimensional Sensing Mapless Navigation of Terrestrial Mobile Robots," in *Lecture Notes in Networks and Systems*, vol. 715 LNNS, pp. 156–165, 2023, doi: 10.1007/978-3-031-35507-3_16.

[43] J. Cao *et al.*, "Study on the Path Planning Algorithm Based on Dueling Deep Q Network," in *Journal of Physics: Conference Series*, vol. 1920, no. 1, 2021, doi: 10.1088/1742-6596/1920/1/012084.

[44] M. Gök, "Dynamic path planning via Dueling Double Deep Q-Network (D3QN) with prioritized experience replay," *Applied Soft Computing*, vol. 158, p. 111503, 2024, doi: 10.1016/j.asoc.2024.111503.

[45] D. A. Deguale, L. Yu, M. L. Sinishaw, and K. Li, "Enhancing Stability and Performance in Mobile Robot Path Planning with PMR-Dueling DQN Algorithm," *Sensors*, vol. 24, no. 5, p. 1523, 2024, doi: 10.3390/s24051523.

[46] W. Hu, Y. Zhou, and H. W. Ho, "Mobile Robot Navigation Based on Noisy N-Step Dueling Double Deep Q-Network and Prioritized Experience Replay," *Electronics (Switzerland)*, vol. 13, no. 12, p. 2423, 2024, doi: 10.3390/electronics13122423.

[47] J. Zhang, Z. Zhang, S. Han, and S. Lü, "Proximal policy optimization via enhanced exploration efficiency," *Information Sciences*, vol. 609, pp. 750–765, 2022, doi: 10.1016/j.ins.2022.07.111.

[48] C. C. Wong, K. D. Weng, and B. Y. Yu, "Multi-Robot Navigation System Design Based on Proximal Policy Optimization Algorithm," *Information (Switzerland)*, vol. 15, no. 9, p. 518, 2024, doi: 10.3390/info15090518.

[49] H. Taheri, S. R. Hosseini, and M. A. Nekoui, "Deep Reinforcement Learning with Enhanced PPO for Safe Mobile Robot Navigation," *arXiv preprint arXiv:2405.16266*, 2024.

[50] S. Wen, Y. Zhao, X. Yuan, Z. Wang, D. Zhang, and L. Manfredi, "Path planning for active SLAM based on deep reinforcement learning under unknown environments," *Intelligent Service Robotics*, vol. 13, no. 2, pp. 263–272, 2020, doi: 10.1007/s11370-019-00310-w.

[51] M. F. Ahmed, K. Masood, V. Fremont, and I. Fantoni, "Active SLAM: A Review on Last Decade," *Sensors*, vol. 23, no. 19, p. 8097, 2023, doi: 10.3390/s23198097.

[52] N. Botteghi, B. Sirmacek, R. Schulte, M. Poel, and C. Brune, "Reinforcement learning helps slam: Learning to build maps," *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, vol. 43, no. B4, pp. 329–336, 2020, doi: 10.5194/isprs-archives-XLIII-B4-2020-329-2020.

[53] Y. Huang, H. Zhao, and X. Peng, "A deep reinforcement learning-based framework for active SLAM in unknown environments," *Robotics*, vol. 12, no. 1, p. 8, 2023, doi: 10.3390/robotics12010008

[54] X. Zhang, X. Shi, Z. Zhang, Z. Wang, and L. Zhang, "A DDQN path planning algorithm based on experience classification and multi steps for mobile robots," *Electronics*, vol. 11, no. 14, p. 2120, 2022.

[55] M. F. R. Lee and S. H. Yusuf, "Mobile Robot Navigation Using Deep Reinforcement Learning," *Processes*, vol. 10, no. 12, p. 2748, 2022, doi: 10.3390/pr10122748.

[56] M. A. Chunab-Rodríguez, A. Santana-Díaz, J. Rodríguez-Arce, E. Sánchez-Tapia, and C. A. Balbuena-Campuzano, "A Free Simulation

Environment Based on ROS for Teaching Autonomous Vehicle Navigation Algorithms," *Applied Sciences (Switzerland)*, vol. 12, no. 14, p. 7277, 2022, doi: 10.3390/app12147277.

[57] R. Mengacci, G. Zambella, G. Grioli, D. Caporale, M. G. Catalano, and A. Bicchi, "An Open-Source ROS-Gazebo Toolbox for Simulating Robots With Compliant Actuators," *Frontiers in Robotics and AI*, vol. 8, p. 713083, 2021, doi: 10.3389/frobt.2021.713083.

[58] W. Han, "Robotics evaluation toolkits," *GitHub*. 2024. [Online]. Available: https://github.com/wh200720041/warehouse_simulation_toolkit/blob/master/README.md.

[59] Y. Raoui and M. Amraoui, "Simultaneous Localization and Mapping of a Mobile Robot with Stereo Camera Using ORB Features," *Journal of Automation, Mobile Robotics and Intelligent Systems*, vol. 18, no. 2, pp. 62–71, 2024, doi: 10.14313/jamris/2-2024/14.

[60] J. Qiao, J. Guo, and Y. Li, "Simultaneous localization and mapping (SLAM)-based robot localization and navigation algorithm," *Applied Water Science*, vol. 14, no. 7, p. 151, 2024, doi: 10.1007/s13201-024-02183-6.

[61] F. M. Rico, J. M. G. Hernández, R. Pérez-Rodríguez, J. D. Peña-Narvaez, and A. G. Gómez-Jacinto, "Open source robot localization for nonplanar environments," *Journal of Field Robotics*, vol. 41, no. 6, pp. 1922–1939, 2024, doi: 10.1002/rob.22353.

[62] H. Zhu and Q. Luo, "Indoor Localization of Mobile Robots Based on the Fusion of an Improved AMCL Algorithm and a Collision Algorithm," *IEEE Access*, vol. 12, pp. 67199–67208, 2024, doi: 10.1109/ACCESS.2024.3399192.

[63] M. Peavy, P. Kim, H. Oyediran, and K. Kim, "Integration of Real-Time Semantic Building Map Updating with Adaptive Monte Carlo Localization (AMCL) for Robust Indoor Mobile Robot Localization," *Applied Sciences (Switzerland)*, vol. 13, no. 2, p. 909, 2023, doi: 10.3390/app13020909.

[64] S. He, T. Song, P. Wang, C. Ding, and X. Wu, "An Enhanced Adaptive Monte Carlo Localization for Service Robots in Dynamic and Featureless Environments," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 108, no. 1, 2023, doi: 10.1007/s10846-023-01858-7.

[65] Y. Cao, K. Ni, T. Kawaguchi, and S. Hashimoto, "Path Following for Autonomous Mobile Robots with Deep Reinforcement Learning," *Sensors*, vol. 24, no. 2, p. 561, 2024, doi: 10.3390/s24020561.

[66] M. Usayiwevu, F. Sukkar, C. Yoo, R. Fitch, and T. Vidal-Calleja, "Continuous planning for inertial-aided systems," *Autonomous Robots*, vol. 48, no. 8, p. 24, 2024, doi: 10.1007/s10514-024-10180-6.

[67] X. Zhao, L. Wang, Y. Zhang, X. Han, M. Deveci, and M. Parmar, "A review of convolutional neural networks in computer vision," *Artificial Intelligence Review*, vol. 57, no. 4, 2024, doi: 10.1007/s10462-024-10721-6.

[68] G. Rangel, J. C. Cuevas-Tello, J. Nunez-Varela, C. Puente, and A. G. Silva-Trujillo, "A Survey on Convolutional Neural Networks and Their Performance Limitations in Image Recognition Tasks," *Journal of Sensors*, vol. 2024, 2024, doi: 10.1155/2024/2797320.

[69] A. Younesi, M. Ansari, M. Fazli, A. Ejlali, M. Shafique, and J. Henkel, "A Comprehensive Survey of Convolutions in Deep Learning: Applications, Challenges, and Future Trends," in *IEEE Access*, vol. 12, pp. 41180-41218, 2024, doi: 10.1109/ACCESS.2024.3376441.

[70] D. E. Neves, L. Ishitani, and Z. K. G. do Patrocínio Júnior, "Advances and challenges in learning from experience replay," *Artificial Intelligence Review*, vol. 58, no. 2, 2025, doi: 10.1007/s10462-024-11062-0.

[71] H. Hassani, S. Nikan, and A. Shami, "Improved exploration–exploitation trade-off through adaptive prioritized experience replay," *Neurocomputing*, vol. 614, p. 128836, 2025, doi: 10.1016/j.neucom.2024.128836.

[72] Y. Yang, M. Xi, H. Dai, J. Wen, and J. Yang, "Z-Score Experience Replay in Off-Policy Deep Reinforcement Learning," *Sensors*, vol. 24, no. 23, p. 7746, 2024, doi: 10.3390/s24237746.

[73] J. Ma, D. Ning, C. Zhang, and S. Liu, "Fresher Experience Plays a More Important Role in Prioritized Experience Replay," *Applied Sciences (Switzerland)*, vol. 12, no. 23, p. 12489, 2022, doi: 10.3390/app122312489.

[74] C. Kim, "Target-Network Update Linked with Learning Rate Decay Based on Mutual Information and Reward in Deep Reinforcement Learning," *Symmetry*, vol. 15, no. 10, 2023, doi: 10.3390/sym15101840.

[75] P. Wang, X. Li, C. Song, and S. Zhai, "Research on Dynamic Path Planning of Wheeled Robot Based on Deep Reinforcement Learning on the Slope Ground," *Journal of Robotics*, vol. 2020, no. 1, p. 7167243, 2020, doi: 10.1155/2020/7167243.

[76] C. Kim, "Temporal consistency-based loss function for both deep q-networks and deep deterministic policy gradients for continuous actions," *Symmetry*, vol. 13, no. 12, p. 2411, 2021, doi: 10.3390/sym13122411.

[77] S. Meyn, "The Projected Bellman Equation in Reinforcement Learning," *IEEE Transactions on Automatic Control*, vol. 69, no. 12, pp. 8323–8337, 2024, doi: 10.1109/TAC.2024.3409647.

[78] Z. Ben Hazem, "Study of Q-learning and deep Q-network learning control for a rotary inverted pendulum system," *Discover Applied Sciences*, vol. 6, no. 2, 2024, doi: 10.1007/s42452-024-05690-y.

[79] X. Xu, X. Li, N. Chen, D. Zhao, and C. Chen, "Autonomous Obstacle Avoidance with Improved Deep Reinforcement Learning Based on Dynamic Huber Loss," *Applied Sciences (Switzerland)*, vol. 15, no. 5, p. 2776, 2025, doi: 10.3390/app15052776.

[80] S. Mishra and A. Arora, "A Huber reward function-driven deep reinforcement learning solution for cart-pole balancing problem," *Neural Computing and Applications*, vol. 35, no. 23, pp. 16705–16722, 2023, doi: 10.1007/s00521-022-07606-6.

[81] S. Mishra and A. Arora, "Double Deep Q Network with Huber Reward Function for Cart-Pole Balancing Problem," *International Journal of Performability Engineering*, vol. 18, no. 9, pp. 644–653, 2022, doi: 10.23940/ijpe.22.09.p5.644653.

[82] M. Li, X. Gu, C. Zeng, and Y. Feng, "Feasibility Analysis and Application of Reinforcement Learning Algorithm Based on Dynamic Parameter Adjustment," *Algorithms*, vol. 13, no. 9, p. 239, Sep. 2020, doi: 10.3390/a13090239.

[83] Y. Zhang, W. Zhao, J. Wang, and Y. Yuan, "Recent progress, challenges and future prospects of applied deep reinforcement learning: A practical perspective in path planning," *Neurocomputing*, vol. 608, p. 128423, 2024.

[84] N. Khlif, K. Nahla, and B. Safya, "Reinforcement learning with modified exploration strategy for mobile robot path planning," *Robotica*, vol. 41, no. 9, pp. 2688–2702, 2023, doi: 10.1017/S0263574723000607.

[85] Y. Yin, Z. Chen, G. Liu, and J. Guo, "A Mapless Local Path Planning Approach Using Deep Reinforcement Learning Framework," *Sensors*, vol. 23, no. 4, p. 2036, 2023, doi: 10.3390/s23042036.

[86] A. de J. Plasencia-Salgueiro, "Deep Reinforcement Learning for Autonomous Mobile Robot Navigation," *Studies in Computational Intelligence*, vol. 1093, no. 17, pp. 195–237, 2023, doi: 10.1007/978-3-031-28715-2_7.

[87] E. Sayar, G. Iacca, and A. Knoll, "Curriculum Learning for Robot Manipulation Tasks With Sparse Reward Through Environment Shifts," *IEEE Access*, vol. 12, pp. 46626–46635, 2024, doi: 10.1109/ACCESS.2024.3382264.

[88] K. Or *et al.*, "Curriculum-reinforcement learning on simulation platform of tendon-driven high-degree of freedom underactuated manipulator," *Frontiers in Robotics and AI*, vol. 10, p. 1066518, 2023, doi: 10.3389/frobt.2023.1066518.

[89] F. J. Mañas-Álvarez, M. Guinaldo, R. Dormido, and S. Dormido-Canto, "Scalability of Cyber-Physical Systems with Real and Virtual Robots in ROS 2," *Sensors*, vol. 23, no. 13, p. 6073, 2023, doi: 10.3390/s23136073.

[90] J. Platt and K. Ricks, "Comparative Analysis of ROS-Unity3D and ROS-Gazebo for Mobile Ground Robot Simulation," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 106, no. 4, p. 80, 2022, doi: 10.1007/s10846-022-01766-2.