2025 8th International Conference on Electronics, Materials Engineering & Nano-Technology (IEMENTech) | 979-8-3315-2076-2/25/$31.00 ©2025 IEEE | DOI: 10.1109/IEMENTECH65115.2025.10959559

Exploring Reinforcement Learning in Autonomous Robot Path Planning and Obstacle Navigation

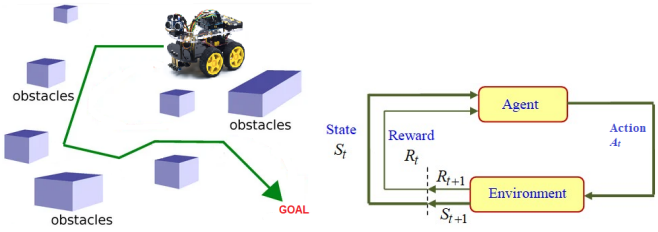
[1]Uday Sankar Mukherjee, [2]Mousumi Laha, and [2*]Piyali Datta
[1]*Department of CSE(AIML), Institute of Engineering & Management, University of Engineering and Management*
[2]*Department of CSE(AIML),CSBS, Institute of Engineering & Management, University of Engineering and Management, Kolkata, India*
{*udaysankar2003, lahamou, datta.piyali.in*}*@gmail.com*

Abstract*—The article deals with issues concerning the autonomous planning of trajectories and avoiding obstacles for mobile robots using reinforcement learning. The emphasis is on dynamic tasks performed by robots devoid of any prior knowledge or maps of the operating environment. In those conditions, conventional motion planning techniques do not work; however, Deep Reinforcement Learning (DRL) is a solution. DRL allows robots to learn and adapt independently while operating in real-time without human intervention, even in the most complicated environments. The research investigates different approaches including Q-learning, SARSA, and DQN algorithms, aiming to evaluate their differences based on the path length and the computational time in reaching that path. The findings confirm that DQN salvages SARSA and Q-learning in terms of efficiency and path optimization. The main focus of the research paper is the fusion of deep learning and reinforcement learning that enables the processing of the sensor to decision making, which in turn enhances the ability of the robot to navigate. The results of the experiments support the applicability of these methods in dynamic scenarios.

Mobile robot, reinforcement learning, path planning, Q learning, DQN, SARSA

I. Introduction

Mobile robots are rapidly involved in complex and danger-prone environments to perform various tasks that include exploring chaotic environments [1], searching and rescuing trapped victims [1], moving important materials [2], etc. Thus, mobile robots are irreplaceable in some crucial scenes [1], [3]. Many developed countries consider mobile robots to be a key industry in robotics [1], [3]. With the rapid development of artificial intelligence along with sensors, networks, and communication technologies, the intelligence of mobile robots improves significantly [4], [5]. The intelligent mobile robot should have high autonomy and thus could complete tasks quickly and accurately without human guidance, ignoring any environmental restrictions [5], [6]. These robots must navigate different environments autonomously (as exemplified in Figure 1(a)). Therefore, robot motion planning integrates diversified technologies like localization [1], mapping [5], perception [7], and path planning [8]. The motion planning problem is fundamentally a search problem for an optimal path avoiding collision from the current location to the goal location gaining information from the external environment deploying the sensors [8]. It is one of the most important benefits of intelligent robots. The optimization objective for motion planning is to explore the shortest planning path while minimizing system cost and training time [6]. Traditional motion planning algorithms can be categorized based on prior knowledge regarding the environment: (1) global motion planning [9], [10], (2) local motion planning [1], [10]. These conventional motion planning methods are generally incapable of executing the tasks in dynamically changing scenes where prior knowledge is not present and in cases of maps, like areas where earthquakes, fire, explosions, and other emergencies occur [10].



(a) (b)

Fig. 1: (a) Schematic diagram of a search space for a robot with obstacles, (b) Generic working principle of reinforcement learning-based approaches.

Consequently, motion planning strategies with self-learning capabilities arise as research directions with utmost importance [6]. Robotic AI technologies have been hugely deployed in all aspects of society. Therefore, Deep Learning (DL) and Reinforcement Learning (RL) approaches are considered the most efficient methods for solving motion planning problems in unknown/less-known dynamic environments [9], [10]. Deep learning combines low-level features to construct an abstract yet easily distinguishable high-level representation through its multilayer network structure along with nonlinear transformation. It discovers distributed feature representation of data while enabling mobile robots to possess strong environmental awareness [7], [8], [10]. RL is mostly involved in solving decision-making problems that are sequential in nature [9].

This work is supported by the grant-in-aid project of IEM-UEM Group allotted to the third author.

979-8-3315-2076-2/25/$31.00 ©2025 IEEE

Similar to animal learning, RL is inspired by a trial-and-error-based strategy. Hence, it involves the reward attained from the interaction between agents and the environment in terms of feedback signals for the sake of training the agents. It avoids the requisites of auxiliary means like artificial markers. Instead of providing instructions to the agent on how to perform the correct actions, the reinforcement signals are attained from the environment and are utilized to evaluate the effectiveness of the corresponding executions of the actions (Figure 1(b)). Hence, the approach differs from supervised learning where a mentor signal is associated. With easier availability of reinforcement signals for a variety of decision control problems, RL becomes a remarkably effective solution for solving decision problems in association with mobile robots [9-13].

Reinforcement Learning (RL) based motion planning methods can enable a robot to attain autonomous learning capability and therefore can find a way to reach the goal position [10]. It processes the input data to yield the output in an end-to-end manner [12]. Generally, sensor data, like laser scanning, color/depth image, etc. act as the input state while the robot motion parameters (e.g. displacement, coordinates, direction etc.) are designated as outputs. The training depends on the interactions between the environment and the agent (robot). Here, the optimization process does not rely on prior knowledge about the environment model, which may guide the robot in navigating toward the target avoiding collision in real-time scenarios. DRL is preferred over other algorithms for the following advantages: (i) since it does not require labeling of samples, solving special cases within the environment can be performed with higher efficacy; (ii) the point-to-point path planning strategy can deal with the system holistically making the system robust. These advantages possess a significant role in optimizing the decision control of mobile robots [9-13]. DRL-based motion planning has introduced significant breakthroughs in several tasks involving complex decision controlling systems and high-dimensional input data.

A. Problem Formulation

The fundamental motionâplanning problem is defined as follows. Given:

- S_{con} : configuration space;
- S_{obs} : obstacle region;
- S_{free}: free space ($S_{con} - S_{obs}$);
- $q(x_{init})$: configuration q allied with initial state x_{init};
- $q(x_{goal})$: configuration q allied with goal state x_{goal};

Compute a time τ and a set of controls $c : [0, \tau] \rightarrow C$ such that $x(\tau) = x_{goal}$ and $q(x(t)) \in S_{free}, \forall t \in [0, \tau]$. It can be depicted in integral form as follows:

$$\begin{aligned} x(\tau) &= x(0) + \int_0^{\tau-1} f(x(t), c(t))dt \\ x(t+1) &= x(t) + f(x(t), c(t)) \\ x(0) &= x_{init} \\ x(\tau) &= x_{goal} \\ q(x(t)) &\in S_{free}, \forall t \in [0, \tau] \end{aligned} \tag{1}$$

A motion planning framework following learning-based techniques comprises the following modules [6].

- Preprocessor module ($M : S_{con} \rightarrow S_{pro}$) considers the configuration space of recent time and the data extracted from sensors as its input and processes the configuration space. Its main objective is to extract the subspace from the already mentioned configuration space which, in turn, enhances the search efficacy. It also encodes the configuration space to a simpler one for conducting the planning and depicting the obstacles with low-dimensional data;
- Prediction module ($P : C \times X \rightarrow S_{pro}$) also aims for data preprocessing; however, this module is executed multiple times during the path planning;
- Execution module ($E : S_{pro} \times C \rightarrow X$) selects an action from action space C analyzing the current configuration S_{pro} and generates a new state;
- Collisionâchecker module ($O : S_{pro} \times X \rightarrow \{T, F\}$) examines whether the new state collides with the obstacles in the configuration space.

1) Formulation of the Problem using Markov Decision Process: A motion-planning problem can be formulated as a Markov Decision Process (MDP) using the concept of Reinforcement Learning. The key components of an MDP representation for motionâplanning can be depicted as follows. The fundamental MDP can be modeled as $\langle S, A, P, R, \gamma \rangle$.

- S represents a state space including the configuration space of the robot along with the observable environment (i.e. a 2D Cartesian space and the sensor information). A state $s_t \in S$ maintains the Markov property;
- A denotes the action space, $\forall a_t \in A$, similar to the control space C;
- $P(s_{t+1}|a_t, s_t)$ refers to the probability of state transition with which the robot executes an action a_t from the action set and the state s_t transits to the state s_{t+1}.
- $R(s_t, a_t)$ designates a reward function that is offered to the robot after action a_t which helps the robot know whether an action is 'suitable' at a particular state s_t. This has a similarity with the cost function deployed in classical motion planning which describes the desired behavior;
- γ represents a discount factor, $\gamma \in [0, 1]$ which adjusts the reward value.

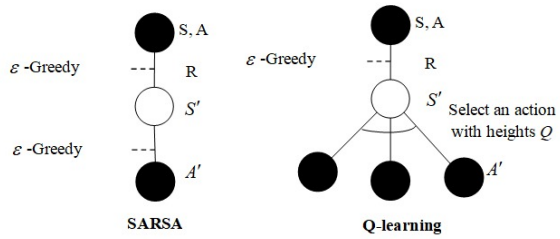


Fig. 2: Pictorial representation of SARSA and Q-Learning working principle.

When a robot takes an action $a_t \in A$ in the state $s_t \in S$ at time instant t, the robot will move to state s_{t+1} at time instant $t+1$ following the state transition probability $P(s_{t+1}|a_t, s_t)$, while it will gain an immediate reward $R(s_t, a_t)$. The key objective of Reinforcement learning algorithms is to learn a policy $\pi(a|s)$ which can choose actions that maximize the longâterm reward, referred to as the accumulated reward.

II. Methodology

A. Mathematical modeling of the optimization problem

A robot needs to move from its initial position *Init* (x_i, y_i) to its goal position *Goal* (x_g, y_g) with an objective to find an optimized path from *Init* to *Goal*. The produced path comprises a set of n nodes $(N_k, k = 1...n)$ and $(n\hat{a}1)$ segments defining two consecutive nodes $(\{Seg_k(N_k, N_{k+1})\})$. Let us assume a robot moving in an environment with m number of static obstacles, where information about the obstacles are known apriori. As shown in Fig. 1, we consider the obstacles in rectangular prism shape for simplicity. If the obstacles are in some irregular form, we approximate them in a cuboid form taking four extreme points horizontal and vertical. Similarly, the robot is also approximated by a cuboid. Their coordinates are altered according to the recent position of the robot (Pos_{curr}). The mathematical formulation for each such segment depicted by two points (P_i, P_j) of a rectangular prism enveloping an irregularly shaped obstacle is defined as:

$$Seg(P_i, P_j) = \begin{cases} y - y_k = \dfrac{y_l - y_k}{x_l - x_k}(x - x_k) \\ MIN(x_l, x_k) \leq x \leq MAX(x_l, x_k) \end{cases} \quad (2)$$

The following depict the parameters and indices in the mathematical model:

- n: node count of the generated path,
- m: obstacle count in the environment,
- $i(i \in 1...n)$: index of the path segment defined by *i*th and (*i* + 1)th nodes,
- $j(j \in 1...m)$ identifies the *j*th obstacle in the environment,
- *k*, *l* $(k, l \in 1...4)$ depicts the point defining an obstacle,

B. Problem Description

Environment Representation: A 2D discrete environment $E \subseteq R^2$ (size $H \times W$) with a set of N_s number of static obstacles $P_s = \{p_1, ..., p_{N_s}\}$, where $p_i \subset E$ denotes the ith static obstacle. The free space $E \backslash P_s$ is depicted by a roadmap $M = \langle P_f, \epsilon \rangle$, where $P_f = \{p_1, ..., p_{N_f}\} = E \backslash P_s$ refers to the free cells' set and $e_{xy} = (p_x, p_y) \subset \epsilon$ denotes the available space for traversal between two free cells p_x and p_y without crossing any other cell, i.e. the smallest road segment. $P_d(t) = \{d_1(t), ..., d_{N_d}(t)\}$ represents the locations of N_d number of dynamic obstacles at a time instant t, where $\forall k, l, t, d_k(t) \subset P_f, (d_k(t), d_k(t+1)) \subset \epsilon$ or $d_k(t) = d_k(t+1)$, and $d_k(t) \neq d_l(t)$. Moreover, if $d_k(t+1) = d_l(t)$, then $d_l(t+1) \neq d_k(t)$, to ensure the avoidance of any motion conflict.

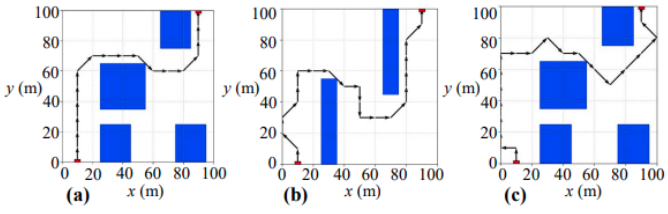


Fig. 3: Exemplifying the pathfinding by Q-learning, SARSA, and DQN for a grid of sizes 30×30 (in (a)) and 40×40 (in (b)), respectively.

Algorithm 1 *Q-Learning Algorithm*

Parameters: step size = $\beta \in (0, 1], \epsilon > 0$
$Q(s, a) \forall s \in S^+, a \in A(s).\ Q(goal, .) = 0$
for each episode e_i **do**
  Initialize S
  for each step of an episode e_i **do**
    Select A from S deploying policy π belonged to Q
    Take action A, observe O, S'
    $Q(S, A) \leftarrow Q(S, A) + \beta[O + \gamma max_a Q(S', a) - Q(S, a)]$
    $S \leftarrow S'$
  end for
  Until S is a *goal*
end for=0

C. Q-learning algorithm

In the Q-Learning algorithm, the idea of achieving a reward as well as a penalty while exploring the environment is based on suitable policy (as depicted in Figure 2). Following the policy, it then executes that action. The robot attains a state (s) and a reward (r) from the navigation environment in return. While optimizing the path, a state comprises the current position of the robot within its workspace and an action refers to the movement the robot takes while moving from one state to the next state. To achieve this goal, the action strategies are adjusted by the robot itself through the reward value r attained from the environmental feedback. This value assesses the action quality. Algorithm 1 shows the stepwise description of Q-Learning algorithm. The agent starts at an initial state S belonged to the environment. It selects an action A based on the current state and policy that guides the decision of an agent by specifying the probability of choosing each action at the given state. The popular policy strategies that are followed include ϵ-Greedy (where the action with the highest Q-value is explored; however, sometimes other actions are explored randomly) and Softmax (in which actions with probabilities proportional to their Q-values are prioritized).

D. State-Action-Reward-State-Action (SARSA)

The State-Action-Reward-State-Action (SARSA) RL algorithm enables the agents for learning and taking decisions in an environment while maximizing the cumulative rewards over time. An iterative process is followed to interact with the environment and learning from its experiences followed by the refinement of its decision-making policy. This is accomplished by modifying the policy to an ϵ-greedy manner at each step.

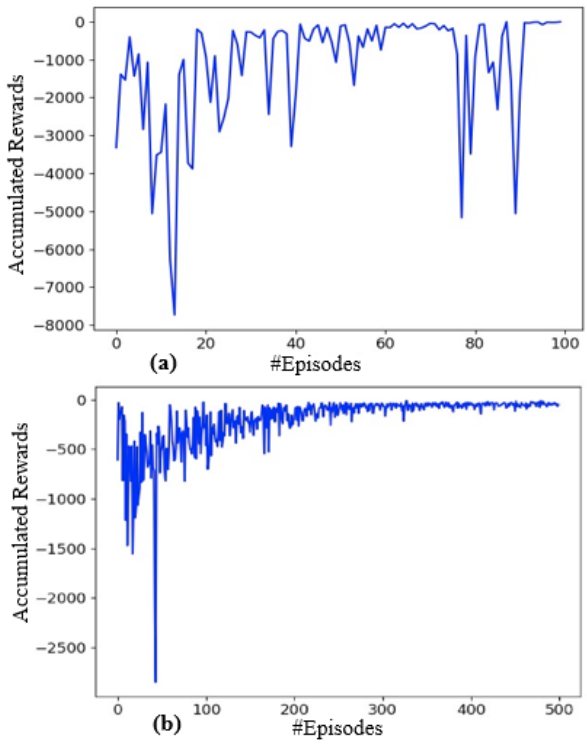


Fig. 4: Variation in the accumulated rewards with respect to the number of episodes (#Episodes) for (a) SARSA and (b) DQN.

1) Initialization: At the start, the Q-values for all state-action pairs are set to zeros or any other arbitrary value following an appropriate initialization method. The expected cumulative reward that an agent can attain by taking a particular action in a particular state is estimated by the Q-values.

2) Receiving Reward and Action Taking: After selecting an action, when the agent accomplishes it at its current state, it receives rewards r as the environmental response showing the immediate gain or penalty concerning the selected action. Consequently, the agent transits to a new state S' and the environment also evolves.

3) Next Action Selection: Reaching the new state (S'), the agent follows the same policy while selecting the next action (A'), which ensures SARSA is an on-policy algorithm, i.e. it learns and updates the Q-values depending on the current policy, where the action can be selected based on some past experiences or predefined exploration strategies. The *Temporal Difference* (TD) error is computed by the agent. TD depicts the difference between the estimated Q-value of the recent state-action pair ($Q(S, A)$) and the cumulative value of the already observed reward and the estimated Q-value of the successive state-action pair ($Q(S', A')$). The Q-value of $Q(S, A)$ is then updated referring to the TD error, a learning rate (α), and the already observed reward. Following depicts the update rule:

$$Q(S, A) \leftarrow Q(S, A) + \beta[r + \gamma Q(S', a') - Q(S, a)] \quad (3)$$

- s, a, and r denote the current state, the action taken in state s, the immediate reward attained, respectively.
- s' and a' represent the next state, the action chosen in state s' based on the recent policy, respectively.
- β is the learning rate ($0 < \beta \leq 1$) and γ depicts the discount factor ($0 \leq \gamma \leq 1$).

The learning rate decides the extent to which the Q-value is modified depending on the current information. Though a higher learning rate generally result in quicker learning, it may lead to an unstable condition. On the other hand, a lower learning rate accomplishes a slower but more steady learning. γ determines the significance of future rewards; if the value is close to 0, it gives priority to immediate rewards, while a value close to 1 emphasizes more on the long-term rewards.

E. Deep Q-learning (DQN) algorithm

Deep Q-Learning is a variant of reinforcement learning which deploys a deep neural network for approximating the Q-function. Q-function is fundamentally used to decide the optimal action that needs to be taken at a given state. It denotes the expected cumulative reward of accomplishing a specific action at a particular state while following a certain policy. Q-Learning updates the Q-function iteratively when the agent has an interaction with the environment. Deep Q-Learning has been being deployed in a wide range of applications like robotics, autonomous vehicles, game playing, and so on.

Instead of a simple Q-table of values, this variant of Q-Learning integrates a deep neural network to represent the Q-function, which allows the algorithm to tackle environments possessing a large number of actions and states while learning from high-dimensional inputs from sensor data. The key challenge issue in implementing DQN is that the Q-function is precisely non-linear and can possess multiple local minima, for which difficulty can arise for the neural network in converging to the appropriate Q-function. Various techniques like experience replay, target networks, etc. can be involved to address this. In experience replay, a subset of the agent's experiences is stored in terms of $\langle state, action, reward, nextstate \rangle$ in a buffer and samples are taken from this buffer while updating the Q-function. This helps in decorrelating the data while making the learning more stable. On the other hand, target networks are deployed for stabilizing the Q-function updates, where a separate network is built to calculate the target Q-values. The Q-function network is updated depending on that value. Figure 3 depicts the working principle of DQN.

III. Experimental Results

In this experiment, we conducted a comparative study of three reinforcement learning algorithmsâSARSA, Q-Learning, and Deep Q-Network (DQN)âapplied to autonomous path planning problems. The results were evaluated on multiple grid environments of varying sizes (ranging from 10Ã10 to 40Ã40) under different observation counts. The key performance metrics considered were path length (*Len*) and computation time (*Time*), both critical factors in evaluating

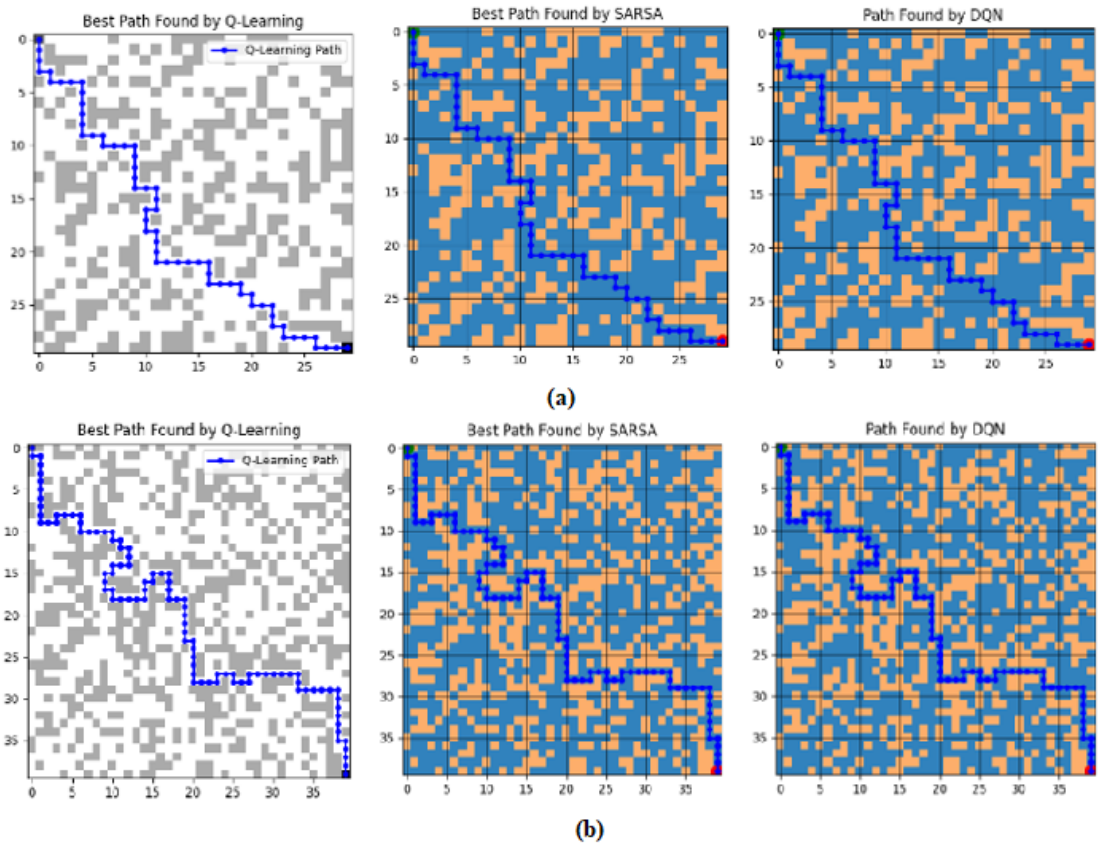


Fig. 5: Exemplifying the path finding by Q-learning, SARSA, and DQN for grid of sizes 30×30 (in (a)) and 40×40 (in (b)), respectively.

the efficiency of the path planning algorithms. The following are insights derived from the tabulated results:

A. Performance on Small Grids (10x10, 15x15)

Across smaller grid sizes (T1 and T2), the path lengths for SARSA, Q-Learning, and DQN were generally identical across trials, indicating all three algorithms are similarly effective in finding optimal paths for small environments. In terms of computation time, DQN significantly outperforms SARSA and Q-Learning. For instance, in the T1 grid with 29 observations, DQN completed the task in 6.29 seconds compared to SARSA's 22.68 seconds and Q-Learningâs 21.17 seconds. This trend holds true for other tests in T1 and T2, where DQN consistently shows faster performance, highlighting its efficiency in smaller grids.

B. Performance on Medium Grids (20×20, 25×25)

On the medium grid environments (T3 and T4), SARSA and Q-Learning began to show slight differences in path lengths. However, DQN still maintained equivalent or slightly shorter path lengths compared to the other two methods. The computation time for SARSA and Q-Learning increased significantly in the T3 and T4 tests, whereas DQN maintained its rapid performance. For instance, in the T4 grid with 153 observations, SARSA took 90.43 seconds while Q-Learning took 113.92 seconds, whereas DQN only required 9.99 seconds. This highlights that as the grid size and complexity increase, the time taken by SARSA and Q-Learning escalates dramatically, whereas DQN remains stable and efficient.

C. Performance on Large Grids (30×30, 40×40)

In large grid environments like T5 and T6, DQN continued to outperform SARSA and Q-Learning significantly. In the T6 grid with 676 observations, SARSA took 455.68 seconds to complete, Q-Learning took 344.36 seconds, while DQN completed the task in just 16.39 seconds. DQN also maintained relatively shorter path lengths, demonstrating its ability to navigate large grids while minimizing both the path length and the computational time required.

DQN outperforms SARSA and Q-Learning in all grid sizes and observation counts, particularly in terms of computational time. The deep neural network's capacity to handle large action and state spaces allows it to converge faster, making it ideal for large and complex environments. Q-Learning shows better time efficiency than SARSA in smaller grids but suffers from increased computation time as the grid size and complexity grow. SARSA, while effective in smaller grids, becomes computationally expensive in larger environments,

TABLE I: Comparative Study among the Path Finding Algorithms

Test Scenario			SARSA		Q-Learning		DQN	
Name	**Spec.**	**# Obs**	***Len***	***Time*(s)**	***Len***	***Time*(s)**	***Len***	***Time*(s)**
T1	10×10	29	19	22.68	19	21.16	19	6.28
		14	18	22.58	18	18.15	18	6.18
		24	19	22.51	19	19.16	19	5.98
T2	15×15	44	28	50.07	28	40.39	28	6.98
		39	29	49.98	29	43.45	29	6.98
		58	31	50.34	31	45.01	31	6.99
T3	20×20	118	39	89.31	39	78.20	39	8.36
		97	36	94.89	36	73.14	36	8.37
		90	34	94.89	34	71.50	34	8.51
T4	25×25	153	46	90.43	46	113.91	46	9.99
		135	49	92.80	49	112.04	49	9.86
		114	44	90.43	44	120.92	44	9.96
T5	30×30	324	75	203.28	78	164.17	68	11.98
		289	61	204.27	61	217.27	61	11.86
		267	53	199.94	51	291. 11	49	11.77
T6	40×40	676	88	455.68	91	344.35	82	16.39

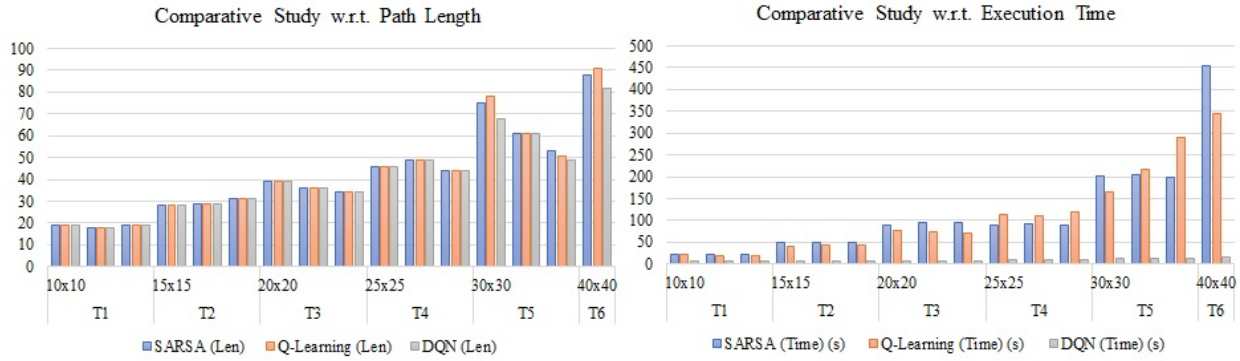


Fig. 6: Comparative study among the performances of SARSA, Q Learning, and DQN with respect to Path Length and Execution time.

making it less suitable for real-time applications where speed is critical. These results suggest that DQN is the best choice for real-time autonomous navigation, particularly in large, complex environments, due to its faster convergence and shorter computational times. SARSA and Q-Learning may be more suited for smaller-scale applications where computational resources are limited, but their performance diminishes as the environment grows in complexity.

REFERENCES

[1] Kobayashi, Masato, et al. “Local path planning: Dynamic window approach with Q-learning considering congestion environments for mobile robot.” IEEE Access 11 (2023): 96733-96742.

[2] H. A. M. Tran, H. Q. T. Ngo, T. P. Nguyen, and H. Nguyen, “Develop of AGV platform to support the arrangement of cargo in storehouse,” in Proc. 24th Int. Conf. Autom. Comput. (ICAC), Sep. 2018, pp. 1â5.

[3] N.Yamamoto, “The strategic plan of the Japan pharmaceutical association, the aspect of a super aged society,” YAKUGAKU ZASSHI, vol. 142, no. 9, pp. 951â963, 2022.

[4] C. Nam, S. Lee, J. Lee, S. H. Cheong, D. H. Kim, C. Kim, I. Kim, and S.-K. Park, “A software architecture for service robots manipulating objects in human environments,” IEEE Access, vol. 8, pp. 117900â117920, 2020.

[5] R. Kaneko, Y. Nakamura, R. Morita, and S. Ito, “Point cloud data map creation from factory design drawing for LiDAR localization of an autonomous mobile robot,” Artif Life Robot., vol. 28, pp. 1â9, Jan. 2022.

[6] Sun, Huihui, et al.“Motion planning for mobile robotsâFocusing on deep reinforcement learning: A systematic review.” IEEE Access 9 (2021): 69061-69081.

[7] H.-W. Chae, J.-H. Choi, and J.-B. Song, “Robust and autonomous stereo visual-inertial navigation for non-holonomic mobile robots,” IEEE Trans. Veh. Technol., vol. 69, no. 9, pp. 9613 9623, Sep. 2020.

[8] A. Muhammad, M. A. H. Ali, and I. H. Shanono, “Path planning methods for mobile robots: A systematic and bibliometric review,” ELEKTRIKA-J. Elect. Eng., vol. 19, no. 3, pp. 14 34, 2020.

[9] Wang, Binyu, et al., “Mobile robot path planning in dynamic environments through globally guided reinforcement learning.” IEEE Robotics and Automation Letters 5.4 (2020): 6932-6939.

[10] R. Smierzchalski and Z. Michalewicz, Path Planning in Dynamic Environments. Springer Berlin Heidelberg, 2005.

[11] L. Cohen, T. Uras, S. Jahangiri, A. Arunasalam, S. Koenig, and T. K. S. Kumar, “The fastmap algorithm for shortest path computations,â in Int. Joint Conf. Artif. Intell., 2018, pp. 1427â1433.

[12] Hammoud, Moahmmed, Ola Haydar, and Sergey Lupin, “Experimental Study on Path Planning Algorithms for Warehouse Mobile Robot Based on Reinforcement Learning.” 2024 Conference of Young Researchers in Electrical and Electronic Engineering (ElCon). IEEE, 2024.

[13] Aung, Zayar, et al., “Experimental Study of Algorithms for Planning the Trajectory of a Warehouse Mobile Robot Based on Reinforcement Learning.” 2024 IEEE Conference on Computer Applications (ICCA). IEEE, 2024.

[14] Zaghbani, Imen, Raja Jarray, and Soufiene BouallÃšgue, “Comparative Study of Q-Learning and SARSA Algorithms for UAV Path Planning in 3D Environments.” 2024 IEEE 28th International Conference on Intelligent Engineering Systems (INES). IEEE, 2024.