Engineering and Scientific International Journal (ESIJ)
Volume 13, Issue 2, April – June 2026
ISSN 2394-7187(Online)
ISSN 2394 - 7179 (Print)

RoboPath AI: Warehouse Robot Path Optimization using Tabular Q-Learning with Multi-Goal Task Decomposition

Thettu Mokshagna Theja[*1], M.Gowthami[2]

[1]*Student, Department of Computer Applications, Viswam Engineering College, Madanapalle, Andhra Pradesh.*
[2]*Assistant Professor, Department of Computer Applications, Viswam Engineering College, Madanapalle, Andhra Pradesh.*

***Abstract* —** The rapid growth of warehouse automation has increased the demand for intelligent robotic systems capable of efficient and safe navigation in dynamic environments. Traditional path-planning algorithms perform well in static conditions but are less effective in environments with moving obstacles and multi-step tasks. To address this challenge, this paper presents RoboPath AI, a reinforcement learning-based warehouse robot path optimization system using tabular Q-learning. The proposed system models the warehouse as a 12×12 grid environment where the robot learns to navigate, collect multiple target items, avoid obstacles and human workers, and return to its starting position. A structured reward function is designed to encourage efficient movement and safe navigation. To improve learning performance, a multi-Q-table architecture is implemented, enabling the agent to handle sequential subtasks effectively. An epsilon-greedy strategy is used to balance exploration and exploitation during training. The system also incorporates real-time visualization, policy representation, and reward analysis to monitor learning progress. Additionally, Optuna-based hyperparameter tuning is applied to optimize performance. Experimental results demonstrate that the agent successfully learns optimal navigation strategies over time, showing significant improvement in efficiency and convergence.

***Keywords*:** *Reinforcement Learning; Q-Learning; Warehouse Automation; Path Optimization; Grid-Based Navigation; Multi-Q-Table Architecture*.

1. Introduction

The growth of e-commerce and logistics has increased the need for intelligent and autonomous warehouse systems. Modern warehouses use robotic systems for navigation, item retrieval, and delivery in dynamic environments with obstacles such as humans and moving objects. Traditional path planning algorithms like Dijkstra and A* work well in static environments but are not suitable for real-time changes. To overcome this, reinforcement learning is used to enable agents to learn optimal behaviour through interaction with the environment. RoboPath AI is a reinforcement learning-based system that uses tabular Q-learning for warehouse navigation. It models the warehouse as a 12×12 grid environment where the agent learns to navigate efficiently, collect multiple targets, avoid obstacles, and return to the starting point. The system uses a multi-Q-table approach to handle sequential tasks, improving learning efficiency and overall performance.

2. Literature Survey

Reinforcement learning is a machine learning approach where an agent interacts with an environment to maximize cumulative reward, based on the framework of Markov Decision Processes (MDP) that define states, actions, rewards, and transitions [1]. Q-learning is a model-free reinforcement learning algorithm that learns optimal policies by updating Q-values using the Bellman equation [2] without requiring prior knowledge of the environment, making it suitable for discrete state spaces. Grid-based environments are commonly used to simulate navigation problems by representing them in discrete states, allowing easy visualization of obstacles, target locations, and agent movements [3]. For complex tasks involving multiple objectives, multi-goal reinforcement learning applies task decomposition, multi-Q-table architecture, and sequential learning to handle goals efficiently [7]. The epsilon-greedy strategy is used to balance exploration and exploitation, helping the agent learn effectively while gradually shifting toward optimal decisions [6]. Hyperparameters such as learning rate and discount factor are optimized using frameworks like Optuna [11] to improve training efficiency and convergence performance. Deep reinforcement learning approaches [4] have shown promise for larger state spaces, while multi-agent systems [7] enable cooperative warehouse operations.

3. System Architecture

RoboPath AI follows a modular architecture with four main components: Environment, Agent Training, Visualization, and Hyperparameter Tuning. The Environment module models the warehouse as a 12×12 grid containing obstacles, targets, human positions, and free spaces, and manages movement and rewards. The Agent Training module uses tabular Q-learning to update Q-values and learn optimal navigation policies through iterative learning. The Visualization module provides real-time tracking of the agent's movement using Tkinter and shows policy directions with arrows. The Hyperparameter Tuning module uses Optuna to optimize parameters like learning

DOI: **10.30726/esij/v13.i2.2026.1320038**

ISSN 2394-7187(Online)
ISSN 2394 - 7179 (Print)

rate, discount factor, and epsilon decay. The overall system architecture is shown in Fig. 1.

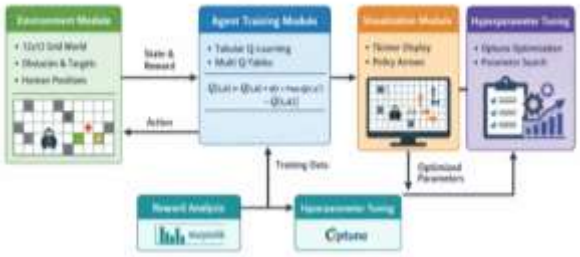

Fig. 1: System Architecture of RoboPath AI

3.1 System Flow

The workflow of RoboPath AI follows a sequential process as shown in Fig. 2. The system begins by initializing the environment and Q-tables, then runs training episodes where the agent selects actions using the epsilon-greedy strategy, performs actions in the environment, and updates Q-tables based on received rewards. After each episode, epsilon is decayed to shift from exploration to exploitation. The process continues until convergence is achieved.

Fig. 2: System Flow Diagram of RoboPath AI

3.2 UML Diagrams

3.2.1 Use Case Diagram

The Use Case Diagram in Fig. 3 shows how a Researcher/Engineer interacts with the system to configure the warehouse environment, set hyperparameters, and initiate agent training. The user defines grid size, obstacles, targets, and training parameters. The system trains the agent through reinforcement learning while providing real-time visualization and policy display, and supports performance analysis and automatic hyperparameter tuning using Optuna.

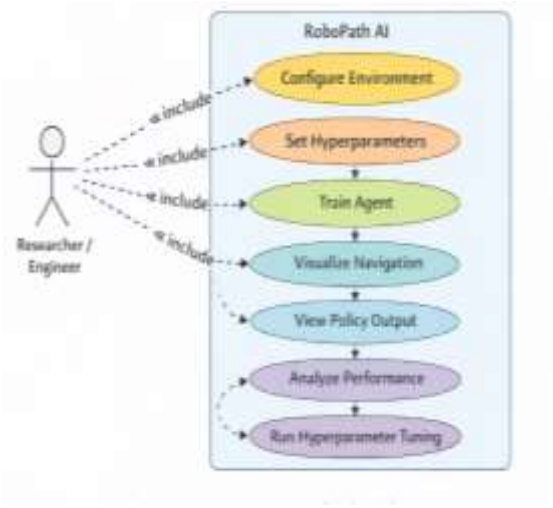

Fig. 3: Use Case Diagram of RoboPath AI

3.2.2 Class Diagram

The Class Diagram in Fig. 4 represents the system's modular structure with four main classes: Environment (models the warehouse grid, handles state transitions, rewards, and task completion), Agent Training (implements Q-learning with action selection and Q-value updates), Visualization (displays real-time agent movement and learned policies using Tkinter), and Hyperparameter Tuning (uses Optuna to optimize learning parameters). These classes work together to enable efficient and scalable warehouse navigation.

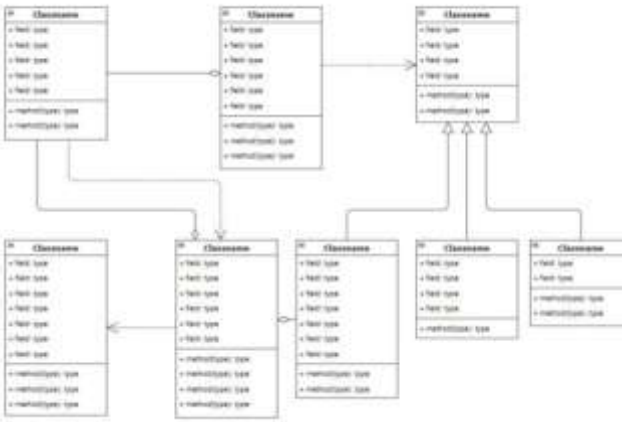

Fig. 4: Class Diagram of RoboPath AI

3.2.3 Sequence Diagram

The Sequence Diagram in Fig. 5 shows how system components interact during training. The Researcher

DOI: **10.30726/esij/v13.i2.2026.1320038**

ISSN 2394-7187(Online)
ISSN 2394 - 7179 (Print)

configures the environment and starts training. The Agent Training module initializes Q-tables and begins episodes where actions are selected using epsilon-greedy strategy, sent to the Environment which returns next state, reward, and status. The Agent updates Q-values while the Visualization module displays movement in real time. After training, Hyperparameter Tuning may be applied through multiple training runs.

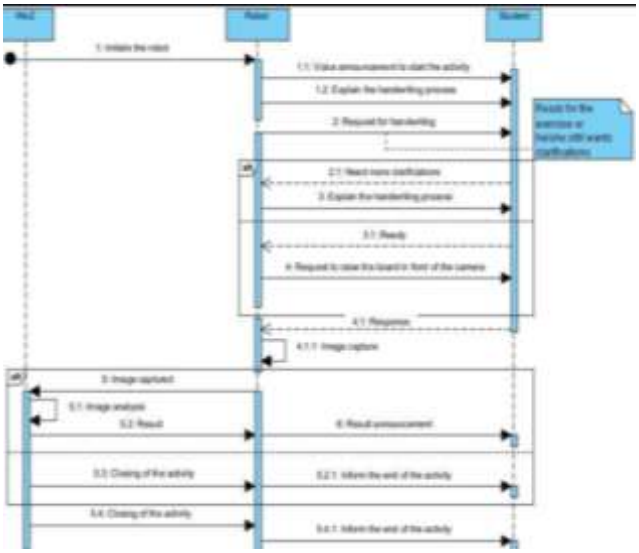

Fig. 5: Sequence Diagram of RoboPath AI

4. Implementation

4.1 Environment Module

The Environment module represents the warehouse as a 12×12 grid-world system. Each grid cell contains encoded values: empty space (0), obstacles (1), target items (2), human workers (3), and return position (4). This module handles agent movement, state transitions, and reward assignment. The take step () function determines the next state and assigns rewards based on agent actions.

4.2 Visualization and Tuning Modules

The Visualization module uses Tkinter to display agent movement, the grid environment, obstacles, and targets in real time. It also provides policy visualization where arrows indicate the best action for each state. The Hyperparameter Tuning module integrates Optuna for automatic optimization of learning rate, discount factor, and epsilon decay, improving training performance and convergence speed.

4.3 Reward Function Design

The reward system plays a crucial role in learning efficiency. The structured reward function assigns penalties and bonuses as summarized in Table 1: normal movement receives −1, hitting an obstacle −200, hitting a human −500, going out of bounds −300, collecting an item +200, collecting the final item +500, and returning to base +700. High penalties for human collisions prioritize safety, while step penalties encourage shorter paths.

Table 1: Reward Function Design

Action	Reward
Move (normal)	-1
Hit obstacle	-200
Hit human	-500
Out of bounds	-300
Collect item	+200
Final item	+500
Return to base	+700

4.4 Multi-Q-Table Architecture

To handle multiple goals, RoboPath AI uses multiple Q-tables: Q1 navigates to Item 1, Q2 to Item 2, Q3 to Item 3, and Q4 handles returning to start. This decomposition reduces learning complexity, allows each Q-table to specialize in a specific objective, and achieves faster convergence and improved accuracy compared to a single monolithic Q-table approach.

5. Results and Discussion

5.1 Performance Analysis

The performance of RoboPath AI was evaluated using a simulated 12×12 grid-world warehouse with fixed obstacles, dynamic human workers, and multiple target locations. The agent was trained over several thousand episodes using tabular Q-learning. After sufficient training, the agent demonstrates the ability to consistently complete the assigned task, successfully identifying optimal paths to reach target locations while avoiding obstacles and human workers. The learned policy ensures efficient routes with minimal unnecessary movements.

In the initial training phase, the agent operates under high exploration rate due to epsilon-greedy strategy, resulting in frequent collisions and negative rewards. As training progresses, Q-values are gradually updated, enabling the agent to distinguish between beneficial and unfavorable actions. The exploration rate decreases due to epsilon decay, leading to improved navigation behavior and reduced collisions. In later stages, total reward per episode becomes consistently positive, indicating successful task completion with minimal penalties.

DOI: 10.30726/esij/v13.i2.2026.1320038

ISSN 2394-7187(Online)
ISSN 2394 - 7179 (Print)

5.2 Reward Progression

The reward progression graph in Fig. 6 illustrates learning performance over training episodes. The early stage shows high fluctuation and negative rewards during exploration. The middle stage demonstrates gradual improvement as learning stabilizes. The final stage exhibits consistent positive rewards indicating convergence. A moving average curve smooths fluctuations and clearly demonstrates the upward learning trend, confirming successful convergence to an efficient navigation policy.

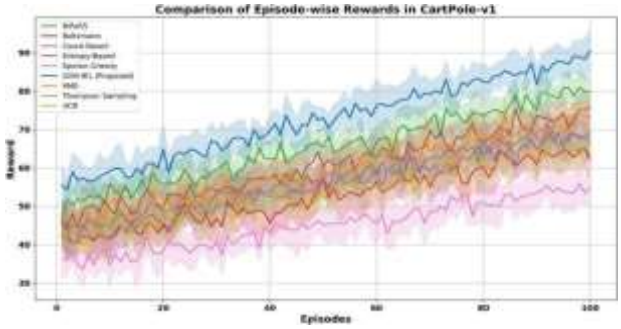

Fig. 6: Reward Progression Graph

5.3 Discussion

The results demonstrate that Q-learning effectively solves warehouse navigation problems and the multi-Q-table approach significantly improves performance through task specialization. The reward design strongly influences agent behavior, with the balance of penalties and bonuses enabling both efficient and safe navigation. The integration of Optuna for hyperparameter optimization further refines the learning process. However, scalability remains limited for very large environments due to the tabular representation, and training requires multiple episodes for convergence.

6. Conclusion

RoboPath AI presents an efficient and intelligent solution for warehouse robot path optimization using reinforcement learning. The system successfully demonstrates the application of tabular Q-learning in a structured grid-based warehouse environment. By incorporating a well-designed reward function, epsilon-greedy exploration strategy, and multi-Q-table architecture, the system enables the robot to navigate efficiently, avoid obstacles, collect multiple target items, and return to its starting position.The experimental results clearly indicate that the learning agent improves its performance over time, transitioning from random exploration to optimal decision-making. The integration of visualization tools enhances interpretability, while Optuna-based hyperparameter tuning improves convergence and overall efficiency.

7. Future Enhancements

The current system can be further enhanced in several directions. Replacing tabular Q-learning with Deep Q-Networks (DQN) would enable handling of larger and continuous environments. Introducing multiple robots for cooperative warehouse operations through multi-agent reinforcement learning would improve throughput. Simulating realistic human movement patterns would enhance safety optimization. Integration with ROS (Robot Operating System) would enable real-world deployment. Finally, extending the model to multi-level 3D warehouse environments would broaden applicability.

References

[1] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction. MIT Press, 2018.
[2] C. J. C. H. Watkins and P. Dayan, “Q-Learning,” Machine Learning, vol. 8, no. 3–4, pp. 279–292, 1992.
[3] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach. Pearson, 2021.
[4] V. Mnih et al., “Human-Level Control through Deep Reinforcement Learning,” Nature, vol. 518, pp. 529–533, 2015.
[5] D. Silver et al., “Mastering the Game of Go with Deep Neural Networks and Tree Search,” Nature, 2016.
[6] R. Bellman, Dynamic Programming. Princeton University Press, 1957.
[7] L. Busoniu et al., “Multi-Agent Reinforcement Learning: An Overview,” Innovations in Multi-Agent Systems, 2010.
[8] T. Schaul et al., “Prioritized Experience Replay,” ICLR, 2016.
[9] J. Schulman et al., “Proximal Policy Optimization Algorithms,” arXiv preprint, 2017.
[10] F. Hutter et al., “Sequential Model-Based Optimization for General Algorithm Configuration,” LION, 2011.
[11] T. Akiba et al., “Optuna: A Next-Generation Hyperparameter Optimization Framework,” KDD, 2019.
[12] M. L. Puterman, Markov Decision Processes. Wiley, 2014.
[13] S. Thrun et al., Probabilistic Robotics. MIT Press, 2005.
[14] P. Abbeel and A. Ng, “Apprenticeship Learning via Inverse Reinforcement Learning,” ICML, 2004.
[15] H. Kober et al., “Reinforcement Learning in Robotics: A Survey,” IJRR, 2013.

DOI: **10.30726/esij/v13.i2.2026.1320038**