TENCON 2021 - 2021 IEEE Region 10 Conference (TENCON)
7-10 Dec 2021. Auckland, New Zealand

Autonomous Warehouse Robot using Deep Q-Learning

Ismot Sadik Peyas[1, 2], Zahid Hasan[1, 2], Md. Rafat Rahman Tushar[1, 2], Al Musabbir[2], Raisa Mehjabin Azni[2], Shahnewaz Siddique[3]
Department of Electrical and Computer Engineering
North South University
Dhaka, Bangladesh
{ismot.peyas, zahid.hasan04, rafat.tushar, al.musabbir, raisa.azni, shahnewaz.siddique}@northsouth.edu

***Abstract*—In warehouses, specialized agents need to navigate, avoid obstacles and maximize the use of space in the warehouse environment. Due to the unpredictability of these environments, reinforcement learning approaches can be applied to complete these tasks. In this paper, we propose using Deep Reinforcement Learning (DRL) to address the robot navigation and obstacle avoidance problem and traditional Q-learning with minor variations to maximize the use of space for product placement. We first investigate the problem for the single robot case. Next, based on the single robot model, we extend our system to the multi-robot case. We use a strategic variation of Q-tables to perform multi-agent Q-learning. We successfully test the performance of our model in a 2D simulation environment for both the single and multi-robot cases.**

***Index Terms*—Warehouse, Autonomous agent, Reinforcement learning, Multi-agent reinforcement learning, Deep Q-learning**

I. Introduction

The global warehouse robotics market is predicted to grow at a CAGR (Compound Annual Growth Rate) of 14.0%, from USD (United States Dollar) 4.7 billion in 2021 to USD 9.1 billion by 2026 [1]. According to Dubois and Hamilton [2] the need for warehouse robots is growing, and is expected to expand. In 2017, these warehouse robots assisted in the picking and packing of goods worth USD 394.8 billion.

The impact of COVID-19 on the market resulted in a massive increase in demand for warehouse robots [1]. The pandemic's supply chain disruption is impacting the market severely. Additionally, due to lockdown and travel restrictions, companies are not able to get the necessary workforce for their operations. Various warehouse operations, such as transportation, picking and placing, packaging, palletizing, and de-palletizing, are automated using warehouse robotics. The deployment of warehouse robots minimizes the need for human interaction and improves warehouse operations efficiency. Warehouse robots are used in a variety of fields such as online shopping, automotive, electrical, electronics, food and beverage, and pharmaceuticals to name a few.

For a sustainable supply chain system, these operations must be executed fast and efficiently. Both autonomous Unmanned Ground Vehicles (UGV) and Unmanned Aerial Vehicles (UAV) can be very efficient in such scenarios. Such warehouse agents can be utilized with autonomous algorithms to conduct operations that are challenging for human operators at low operating costs. Warehouse operations involve receiving, shipping and storing. Stacking loaded pallets in warehouses and storage facilities are critical for preventing accidents. Poorly stacked loaded pallets pose a severe risk to employee safety and can cause significant product damage and increase the total cost of business. Also, in many cases maintaining the health and safety of a human workforce becomes costlier than maintaining a fleet of robots.

The warehouse environment varies from place to place based on their construction and architectural design. Therefore, in many cases, a precise mathematical model of the underlying environment is unavailable or ambiguous. So, it is vital to build an efficient and accurate model to address these complicated tasks without human interference. Moreover, the search environment can change unexpectedly, and the objects can be placed anywhere in the warehouse. Hence, the agent's interaction with the environment should be autonomous, and the agent must have the capability to make decisions for itself.

On such occasions, reinforcement learning (RL) [3] proposes a unique approach to solve these issues. RL does not require any prior knowledge of the environment. Agents based on RL algorithms can navigate the environment autonomously without any explicit model of the environment. Rather, the RL agent frequently interacts with the environment and receives negative or positive rewards based on a predefined reward function. Through this process, it learns to function in an entirely new environment.

Our agent function consists of three major components: (1) autonomous navigation, (2) stacking products optimally, and (3) obstacle avoidance. The autonomous navigation and obstacle avoidance feature is based on Deep Q-learning. The agent has a set of forward, backward, left, and right actions to navigate and avoid collisions in the warehouse environment. The robot finds the maximum available space in the warehouse and then moves the product using the shortest path available to the destination point. The destination space is updated as soon as the product is place in the destination point (maximum available space). Discovering the maximum available space is implemented with the Q-learning algorithm.

Our system is first developed for the single robot case. Later, a multi robot system is also developed to operate in the warehouse environment. In the multi-agent system, all agents aim to maximize their cumulative reward. When an

[1] Equal Contribution.
[2] Undergraduate student.
[3] Assistant Professor, IEEE Member.

agent collides with an obstacle or another agent, their reward is deducted by a certain amount.

II. Related Work

Reinforcement learning is not widely used in warehouse robotics research. In warehouse operations, path finding and obstacle avoidance are challenging. The most popular approaches employed in path computing to meet this difficulty are deterministic, heuristic-based algorithms [4]. [4] compares and contrasts static algorithms (such as A*), re-planning algorithms (such as D*), anytime algorithms (such as ARA*), and anytime re-planning algorithms (such as AD*). Classical algorithms generate path planning for known static environments. In path planning, states are agent locations and transitions between states are actions the agent can do, each with a cost [4]. Later these are expanded and blended to work in a partially known or dynamic environment.

A path planning algorithm is required for the mobile robot to operate autonomously throughout the warehouse [5]. For the mobile robot, this path planning algorithm generates a collision-free path from the start point to the goal point. The location of all the shelves and the open space must be known to the algorithm in order for it to complete this task. In our study, we have used Reinforcement learning, which does not require this information. Once the algorithm has been given the start and destination points, it will evaluate all four nearby grids to see if they are shelves or free space. In works such as [5] the closest euclidean distance between all nearby free space grids and the objective point is considered after identifying the neighboring free space grids, whereas our agent is reward driven. This process is repeated until the distance between the goal and the present point reaches zero.

Reinforcement learning algorithms have already been utilized to develop algorithms for an autonomous aerial vehicle that can rescue missing people or livestock [6]. [6] used Deep Q learning for robot navigation. They used a cyclic approach of three tasks: Region Exploration, Target Search, and Target Selection. The DQN architecture explicitly separates the representation of state values and state-dependent action advantages via two separate streams.

In [7], the authors developed and trained a Deep Reinforcement Learning (DRL) network to determine a series of local navigation actions for a mobile robot to execute. The onboard sensors on the robot provided the sensory data.The results showed that using the DRL method the robot could successfully navigate in an environment towards the target goal location when the rough terrain is unknown.

A system for fast autonomy on a quadrotor platform showed its capabilities and robustness in high-speed navigation tasks [8]. As the speed rises, state estimation, planning, and control difficulties increase significantly. These issues are rectified based on the existing methods and demonstrate the whole system in various environments [8]. To avoid the obstacle, our model uses the deep learning method and object detection is crucial.

[9] presents a review of deep learning-based object detection frameworks. It initially focuses on typical generic object detection architectures and some modifications and valuable tricks to improve detection performance. As distinct particular detection tasks show various characteristics, [9] briefly survey numerous specific tasks, including salient object detection, face detection, and pedestrian detection. Experimental studies are also given to distinguish various methods. Finally, some promising directions and tasks are provided to serve as guidelines for future work in both object detection and relevant neural network-based learning systems.

III. Deep Q-Learning for Warehouse Agents

A. Deep Q-learning

Any discrete, stochastic environment can be described as Markov Decision Process (MVP). MVP is the mathematical formulation of intelligent decision-making processes. According to MVP, an actor or agent, given an environment, E, performs a task or takes action at time t and transits into a new state s_{t+1} of that environment at a time $(t+1)$. This can be written as,

$$f(S_t, A_t) = R_{t+1} \tag{1}$$

The reward can further be described as a discounted reward, where the agent takes action following a policy, which provides the agent with the future discounted reward of this present action. The discounted reward can be formulated as,

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \tag{2}$$

Here, γ is the discounted factor which is between 0 and 1. The maximum discounted reward depends on the optimal state-action value pair followed by the policy. Q-learning is based on this MVP paradigm. By following this process, the optimal q-function can be written as,

$$q^*(s,a) = max\ q(s,a) \tag{3}$$

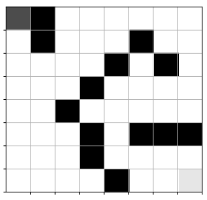
According to this q-function, the policy should choose the highest q-value to get the highest future overall reward. To get the optimal q-value, the Bellman Equation [10] must be satisfied. Therefore, we can write,

$$q^*(s,a) = E[R_{t+1} + \gamma max\ q^*(s', a')] \tag{4}$$

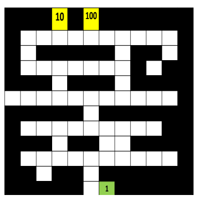
This equation means that the optimal q-value function for a given state-action pair (s,a) will be the expected reward R_{t+1} after taking that action plus the maximum discounted reward by following that optimal policy for the future state-action pair (s', a'). To find this q* value, sometimes a linear function approximator is used if the state space is simple. But in a complex environment, a non-linear function approximator, like the neural network, is used to approximate the optimal q-value.

B. Navigation and Obstacle Avoidance

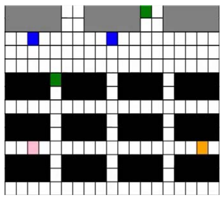
When constructing a warehouse environment agent, we first structure the warehouse upper-view as a 2D map divided into 8×8 equal regions. For simplicity, we assumed that our warehouse would only contain boxes of the same length and width. The warehouse agent has access to the upper view of the environment. That means we can train the agent on this 2D map array. We define the starting point, $s = (x_0, y_0)$, and the map's destination point, $d = (x_d, y_d)$. The warehouse 2D



(a) Navigation and obstacle avoidance system



(b) Maximum available space finding



(c) Multi-agent environment

Fig. 1. Environment design

Algorithm 1 Navigation and Obstacle Avoidance with Deep Q-learning

```
Require: Initialize the warehouse 2D environment, replay memory buffer $E$ with capacity $C_E$, q-function approximator neural network $Q_n$ with random weights, exploration probability $\epsilon$, discount factor $\gamma$
  for episode = 1 to M do
    Get starting state $s_0$
    while episode not terminated do
      Take random $\rho$ value between 0 and 1
      if $\epsilon > \rho$ then
        Take random action $a_t$ from action space
      else
        Action $a_t = argmax_a Q(s_t, a)$
      end if
      Decay exploration probability $\gamma$
      Perform action $a_t$ then get reward $r_t$ and next state $s_{t+1}$
      Store experience tuple $e_t = (s_t, a_t, r_t, s_{t+1})$ in replay memory $E$ in FIFO order
      Take sample minibatch $(s_k, a_k, r_k, s_{k+1})$ from $E$
      if episode terminates at $s_{k+1}$ then
        Set $y_k = r_k$
      else
        Set $y_k = r_k + \gamma max_a Q(s_{k+1}, a)$
      end if
      Calculate loss function $loss = (y_k - Q(s_k, a_k))^2$
      Execute optimization algorithm Adam [11] on loss function to update neural network $Q$ for back-propagation
      Set $s_t = s_{t+1}$
    end while
  end for
```

cells have values of one (1) and zero (0). The zero represents there is an obstacle, and one represents the open path. The visual representation of this warehouse environment is shown in Fig. 1(a). Here the starting point is the upper-left cell, and the destination cell for the agent is the lower-right cell. The black boxes are the walls or obstacles, and white boxes are the allowed moving paths. The agent can move freely with four action spaces: front, back, left, and right. The reward mechanism for the agent is simple, which is shown in Table I. We design a simple neural network, which is used as a function approximator for q-values. The architecture of the neural network is given in Fig. 2. We have trained our model up to 500 epochs, and if the agent can reach the destination before 500 epochs without hitting any walls, we terminate the training process. The whole process can be observed by looking at Algorithm 1. The hyper-parameters used in this training process are shown in Table II.

TABLE I
REWARD MECHANISM FOR TRAINING

Moves	**Rewards**	**Results**
Agent hit with wall/obstacle	-1	End of an episode
Agent in the free-way	0	Continue the episode
Agent reaches the destination	+1	End of an episode

TABLE II
LIST OF HYPERPARAMETERS

Hyperparameter	**Value**	**Description**
Discount Factor	0.90	γ-value in max Q-function
Initial Epsilon	1.0	Exploration epsilon initial value
Final Epsilon	0.1	Exploration final epsilon value
Batch size	32	Mini batch from replay memory
Learning Rate	0.0025	Learning rate for Adam optimizer
Experience Replay Memory	1000	Capacity of experience replay memory

C. *Finding the Maximum Available Space for Storing*

We developed a slightly different environment for this training process. Because this time, the agent has to know each cell's occupied and available space, the warehouse environment must contain that information. The visual design of this environment is shown in Fig. 1(b). The modified 2D view of the environment has cells containing five different values. The cells' values and their representations are shown in Table III. The goal for the agent is to learn the shortest possible path to reach the cell that has the most available space. Moreover, the agent has to learn to avoid any obstacle while reaching the optimal destination point. After arriving at the optimal destination, which is 100 in our environment, the available space for that cell is updated. For example,

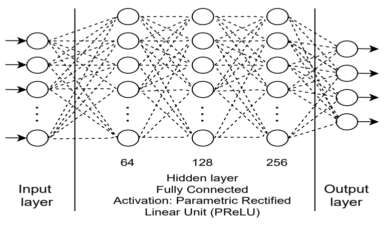


Fig. 2. Neural network structure

Algorithm 2 Maximum Available Space with Q-learning

```
Require: Initialize the warehouse 2D environment, Q table $Q$, exploration probability $\epsilon$, discount factor $\gamma$, q-value updating factor $\alpha$
  for episode = 1 to M do
    Initialize a random process and get the initial state information $s_0$
    while episode not terminated do
      Take random $\rho$ value between 0 and 1
      if $\epsilon > \rho$ then
        Take random action $a_t$ from action space
      else
        Action $a_t = arg\max_a Q(s_t, a_t)$
      end if
      Decay exploration probability $\epsilon$
      Execute action $a_t$ then observe reward $r_t$ and next state $s_{t+1}$
      Get Q-predict $Q_p = Q(s_t, a_t)$ from Q-table
      if episode terminates at $s_{t+1}$ then
        Target $Q_T = r_t$
      else
        Target $Q_T = r_t + \gamma max_a Q(s_{t+1})$
      end if
      Update Q-table $Q(s_t, a_t) += \alpha(Q_T - Q_P)$
      Decay updating factor $\alpha$
    end while
  end for
```

when the object reaches the maximum available space cell, which is 100, the available space for that cell becomes 99. We developed a 12×12 2D map array for training this model. We used the Q-learning algorithm for training our agent to navigate and identify the optimal path and destination through the warehouse environment. Through exploration, our agent can get to know the best possible action that can be taken given a state. The mathematical explanation of Q-learning can be found in Section III-A. For policy or action-selection strategy, we employed the ϵ-greedy [12] approach during training. Equation 4 shows the updating process of optimal Q-function. In section III-A, we describe that for q-function, sometimes a linear function approximator is used. In this model, we used a vector-based q-table for storing and retrieving the updated q-values. Algorithm 2 contains the detailed implementation of our model.

D. Multi-agent Exploration

The optimal system for the warehouse problem will be a multi-agent environment where more than one agent will interact with the warehouse environment cooperatively. We

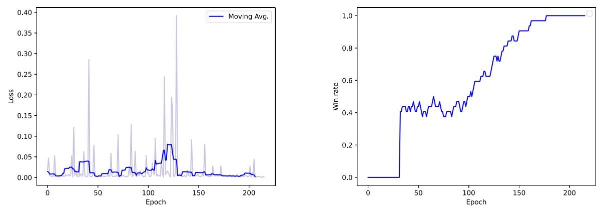
TABLE III
REWARD AND CELL'S VALUE REPRESENTATION

Values	Representation
-100	Wall or obstacle
-1	Open Path
100	One hundred available space
10	Ten available space
1	The object to be stored

Algorithm 3 Multi-agent Q-learning with Q-tables

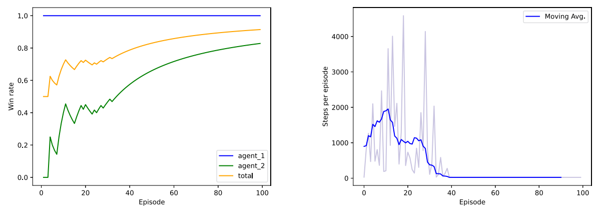
```
Require: Initialize the warehouse 2D environment, Q tables $Q_1, Q_2, ..., Q_n$ for n number of agents, exploration probability $\epsilon$, discount factor $\gamma$, Q-value updating factor $\alpha$
  for episode = 1 to M do
    Initialize a random process and get the initial state information $s_0$
    while episode not terminated do
      for agent i = 1 to n do
        With probability $\epsilon(i)$ select a random action $a_t(i)$; otherwise, select best available action from q-table
        Decay exploration probability $\epsilon(i)$
        Execute action $a_t(i)$ then observe reward $r_t(i)$ and next state $s_{t+1}(i)$
        Get Q-predict $Q_p = Q(s_t(i), a_t(i))$ from Q-table
        if episode terminates at $s_{t+1}$ then
          Target $Q_T = r_t(i)$
        else
          Target $Q_T = r_t(i) + \gamma max_a Q(s_{t+1}(i))$
        end if
        Update Q-table $Q(s_t(i), a_t(i)) += \alpha(Q_T - Q_P)$
        Decay updating factor $\alpha$
      end for
    end while
  end for
```

designed a multi-agent model for our warehouse environment where multiple autonomous actors can store and transport. Fig. 1(c) displays the visual representation of multi-agent environment. The two blue boxes act as two agents, while the green boxes represent human workers. The orange and pink boxes are the destination points, the black boxes are the obstacles, and the rest white areas are the free-moving path for the agents. We performed multi-agent Q-learning with a strategic variation of Q-tables. We initially create Q-tables for each agent and use these tables to store q-values for state-action pairs during training. We train our agents on these Q-tables containing q-values for every possible optimal navigation from the initial position to the destination in the warehouse environment. The q-values are stored and updated in the Q-tables by the factor of α, which we call the q-value update factor. This variable is used to control the impact of updating and storing q-values. Primarily, q-values are updated with much higher impact or higher factors in the Q-tables. As time passes, the q-value updating impact is reduced by using this q-value update factor α. Initially, we set α value 0.03. This value decays by the factor 0.002 times the current episode until it reaches 0.001. The idea of the q-value updating factor is that primarily our q-values in Q-tables contain values that can be sometimes noisy or wrong, and more impactful updates are needed to those values if any optimal state-action values are observed. But, after some training, the q-values in Q-tables often are more accurate, and it may cause harm to make significant changes to those accurate q-values. So, as time passes, the impact of updating the q-values needs to be reduced by a factor which is α. After successful training, given a state, the agents can predict the optimal action to be taken by exploring the respective Q-tables for each agent. The optimal action



(a) Loss vs. epoch graph (b) Win rate vs. epoch graph

Fig. 3. Navigation and obstacle avoidance results



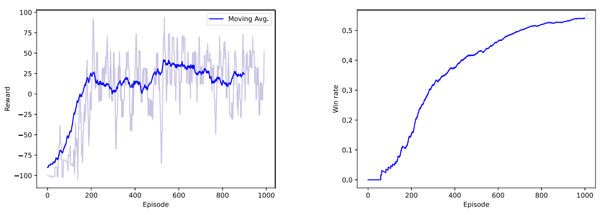
(a) Win rate vs. episode graph (b) Steps vs. episode graph

Fig. 5. Multi-agent RL results

refers to the action which provides the maximum reward among all possible actions that can be taken with a given state. The detailed procedure of our multi-agent model is provided in the Algorithm 3. Here, we have designed a warehouse environment with two autonomous agents, two moving humans, and some obstacles. Our autonomous agents have successfully learned optimal strategies for navigating and reaching a destination without collision with the other agent, the obstacles, and the humans.

IV. Results and Analysis

Fig. 3 shows the training results of the navigation and obstacle avoidance model described in Section III-B. During training, we determine that the training will occur up to 500 epochs. But if the agent can learn better policy before that, we stop the training process. We can call that situation an optimal policy when the agent gets a consistent win rate close to 1. In Fig. 3, our agent learned a decent score between epoch numbers 200 to 220. Fig. 3(a) represents the loss vs. epoch graph, and Fig. 3(b) represents the win rate vs. epoch graph of our navigation and obstacle avoidance system during training. Fig. 3(a) shows the line plot for the loss of the neural network during training. The light-blue line is the actual loss value, and the dark-blue line is the moving average of the loss value in this graph. The moving average is calculated according to Equation (5). The line plot graph, especially the moving average plot, shows that the model is able to train the neural network so that the loss reduces gradually. Fig. 3(b) is a line plot graph that shows that our model is becoming progressively better at reaching the destination without hitting anything. This graph shows that our agent is gradually increasing its winning rate to the point where the win rate becomes close to 1.



(a) Reward vs. episode graph (b) Win rate vs. episode graph

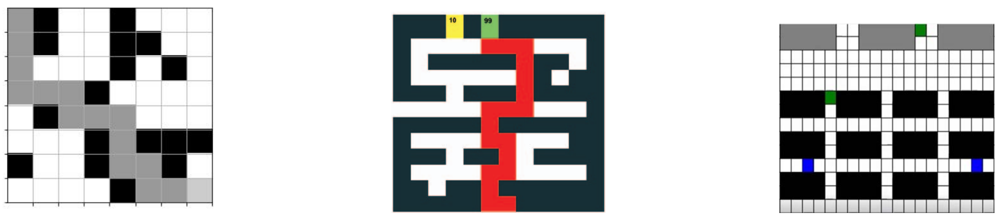
Fig. 4. Maximum available space finding result

$$\text{Moving Average}(k,n) = \frac{\sum_{i=k}^{k+n} v_i}{n} \tag{5}$$

The training result in the maximum space finding model described in Section III-C is shown in Fig. 4. We trained our agent for 1000 episodes. Fig. 4 indicates that the agent successfully learned a better policy by gradually increasing the scores during training. Fig. 4(a) represents the reward vs. episode graph and Fig. 4(b) represents the win rate vs. episode graph of our maximum available space model. Fig. 4(a) shows the line plot graph of gained rewards by the agent while taking action during training. The light-blue line is the actual reward value, and the dark-blue line is the moving average (according to Equation (5)) of the reward value in this graph. In Fig. 4(a), it appears that the agent gradually increases the rewards (according to Table III). Fig. 4(b) represents the line plot graph of win rate while training. Win rate is calculated according to Equation 6. This win rate is continuously increasing in this line graph. This graph is evidence that the agent is gaining optimal policy.

$$\text{Rate}(k) = \frac{\sum_{i=0}^{k} v_i}{k} \tag{6}$$

The results of multi-agent RL expressed in Section III-D are shown in Fig. 5. We trained this multi-agent system with two agents for 100 episodes and recorded the results. Fig. 5(a) displays the line plot graph of win rate vs. episode for every acting agent, and Fig. 5(b) displays the graph for steps per episode vs. episode graph. Fig. 5(a) is a multi-line plot graph that represents the win rate for two agents and the total rate. The win rate matrix is calculated by Equation (6). The three-line plots: blue, green, and orange constitute the win rate graph for the first agent, second agent, and total for both agents. Fig. 5(b) unveils the line plot for the steps needed for the agents to reach the destination points from starting points. The optimal model will take less time to reach the destination. The light-blue line draws the actual value, and the dark-blue line is the moving average value, which is calculated by Equation (5). This graph reveals that at first, the agents took many steps to reach the destination points, which is not convenient in these warehouse storing scenarios. But the agents gradually achieved a better policy to the point where they took the least number of steps to reach the destination because the line plot decreased afterward.



(a) Navigation and obstacle avoidance system (b) Maximum available space finding (c) Multi-agent environment

Fig. 6. Visual demonstration of autonomous agents' navigation during the testing phase

We evaluate each of our models in their respective developed environments to observe the performance. Fig. 6 unveils the visual representation of our agent navigating in the respective environments during the testing phase. Fig. 6(a) shows the path taken by the agent from starting point (upper-left) to destination point (lower-right). The agent's traversing area is the bold gray colored line. Fig. 6(b) displays the path taken by the agent described in Section III-C during the testing phase. The red line is the path taken by the agent from starting point (lower-middle) to the destination point (upper-middle) containing maximum available space (100), which becomes 99 upon the agent's arrival. Finally, Fig. 6(c) displays that the both agents (blue box) are at the destination point which is described Section III-D. By observing these graphs, we can safely say that our three designed models can navigate and reach the destination points by following the shortest possible path, enabling our models to become time-efficient and resource-efficient.

V. Conclusion

In this paper, we design three approaches to navigate the autonomous robots in warehouse systems by using reinforcement learning. The first approach is designed with deep Q-learning, and the second one is developed with traditional Q-learning algorithms with slight variation. Both of these designs are for a single-agent environment. As we know that the practical usage of these autonomous systems will be in a multi-agent environment where optimal navigation and storage for the warehouse will take place, we design a multi-agent RL system for those scenarios. After that, we test and evaluate our designs' results and establish that all of our designs are suitable for use in practical fields as they unveil an excellent performance score for each type of warehouse environment. The results also establish that the autonomous agents reach the destination points by taking the least actions needed so that the cost of navigation remains low. The use of RL in a warehouse environment is ideal because the environment of these systems is dynamic, and RL is suitable to perform well in those partially observable, dynamic states. Although the use of RL algorithms in warehouse navigation is still moderate because of the lack of satisfactory design, we believe the use of RL algorithms in the design process will increase the possibility of deploying an autonomous system in real-world scenarios. In future work, we intend to design a multi-agent system that takes complex and higher dimensional inputs to classify and train the autonomous agents to deal with more practical scenarios ensuring that little or no intervention is needed once deployed.

References

[1] "Warehouse robotics market." [Online]. Available: https://www.marketsandmarkets.com/Market-Reports/warehouse-robotic-market-128876258.html (accessed Jul. 21, 2021)

[2] L. Kolodny, "The team who created amazon's warehouse robots returns with a new robot named chuck," Jul 2017. [Online]. Available: https://www.cnbc.com/2017/07/26/6-river-systems-former-kiva-execs-build-warehouse-robot.html (accessed Aug. 7, 2021)

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

[4] D. Ferguson, M. Likhachev, and A. Stentz, "A guide to heuristic-based path planning," in *Proceedings of the Workshop on Planning under Uncertainty for Autonomous Systems at The International Conference on Automated Planning and Scheduling ICAPS*, 2005.

[5] N. V. Kumar and C. S. Kumar, "Development of collision free path planning algorithm for warehouse mobile robot," *Procedia Computer Science*, vol. 133, pp. 456–463, 2018, international Conference on Robotics and Smart Manufacturing (RoSMa2018). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050918310019

[6] A. Peake, J. McCalmon, Y. Zhang, B. Raiford, and S. Alqahtani, "Wilderness search and rescue missions using deep reinforcement learning," in *2020 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2020, pp. 102–107.

[7] K. Zhang, F. Niroui, M. Ficocelli, and G. Nejat, "Robot navigation of environments with unknown rough terrain using deep reinforcement learning," in *2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2018, pp. 1–7.

[8] K. Mohta, K. Sun, S. Liu, M. Watterson, B. Pfrommer, J. Svacha, Y. Mulgaonkar, C. J. Taylor, and V. Kumar, "Experiments in fast, autonomous, gps-denied quadrotor flight," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 7832–7839.

[9] Z.-Q. Zhao, P. Zheng, S.-T. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[10] R. Bellman, *Dynamic Programming*. Princeton, NJ, USA: Princeton University Press, 1957.

[11] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[12] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, UK, May 1989.