IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 12 August 2023, accepted 28 August 2023, date of publication 4 September 2023, date of current version 19 September 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3311519

RESEARCH ARTICLE

Mobile Service Robot Path Planning Using Deep Reinforcement Learning

A. A. NIPPUN KUMAAR[1] AND SREEJA KOCHUVILA[2]

[1]Department of Computer Science and Engineering, Amrita School of Computing, Amrita Vishwa Vidyapeetham, Bengaluru Campus, Bengaluru 560035, India
[2]Department of Electronics and Communication Engineering, Amrita School of Engineering, Amrita Vishwa Vidyapeetham, Bengaluru Campus, Bengaluru 560035, India

Corresponding author: A. A. Nippun Kumaar (aa_nippunkumaar@blr.amrita.edu)

ABSTRACT A mobile service robot operates in a constantly changing environment with other robots and humans. The service environment is usually vast and unknown, and the robot is expected to operate continuously for a long period. The environment can be dynamic, leading to the generation of new routes or the permanent blocking of old routes. The traditional path planner that relies on static maps will not suffice for a dynamic environment. This work is focused on developing a reinforcement learning-based path planner for a dynamic environment. The proposed system uses the deep Q-Learning algorithm to learn the initial paths using a topological map of the environment. In an environmental change, the proposed $\boldsymbol{\beta}$-decay transfer learning algorithm trains the agent in the new environment. This algorithm uses experience vs. exploration vs. exploitation-based training depending on the similarity of the old and new environments. The system is implemented on the Robotic Operating System framework and tested using Turtlebot3 mobile robot in the Gazebo simulator. The experimental results show that the reinforcement learning system learns all the routes based on the initial topological map of different service environments with an accuracy of over 98%. A comparative analysis of the $\boldsymbol{\beta}$-decay transfer learning and non-transfer learning agents is performed based on various evaluation metrics. The transfer learning agent converges twice faster than the non-transfer learning agent.

INDEX TERMS Path planning, mobile robots, service robot, navigation systems, reinforcement learning, deep Q-Learning, Q-learning, transfer learning, dynamic environment, machine learning, artificial intelligence, incremental learning, lifelong learning.

I. INTRODUCTION

The Mobile Robotics field is one of the most popular domains among robotics researchers. An essential mobile robot subsystem is a navigation system. A navigation system perceives the environment and enables the robot to navigate autonomously to perform autonomous tasks [1]. There is a vast range of applications in which mobile robots are being used, such as surveillance, automation in industries, museum guides, elderly care, hospital care, home assistance, servers in restaurants, and so on. Mobile robots are classified based on their form factor, such as wheeled, legged, flying, and so on [2]. Wheeled mobile robots, in particular, can be effective and simple to be used in flat indoor terrain.

The associate editor coordinating the review of this manuscript and approving it for publication was Yangmin Li.

A mobile service robot performs autonomous tasks in service environments like homes, restaurants, hospitals, etc. [3]. A service environment is unique, unlike other environments, based on a few features of the environment. To name a few, vast square footage, moving obstacles (robots and humans) [4], prone to environmental changes over some time, and augmenting (adding extra sensors) the environment to support the autonomous operation is not possible. A path planner is essential for an autonomous mobile robot to plan an efficient path given the source and destination.

A path planner is as good as its understanding of the environment. Initially, the main focus of the path-planning research community was to achieve optimal paths given different levels of environmental awareness. These works are categorized based on the environmental information available to the path planners [5]. One of the main concerns of a

service robot's path planner is environmental changes that will hinder its performance. A path planner uses the initial environment configuration to identify efficient paths, and this configuration will not be valid if the environment changes. Hence a path planner which can dynamically plan the path based on the latest configuration of the environment is necessary to tackle this problem. This type of path planner is classified as a dynamic path planner, which considers environmental changes throughout its lifetime.

A dynamic path planner continuously monitors environmental changes and updates itself to provide efficient paths. Using machine learning (ML) techniques for dynamic path planning has recently gained more traction among researchers. In particular, Reinforcement Learning (RL) [6] is a type of machine-learning technique that uses the Markov Decision Process (MDP) to learn to interact in an environment. The main reason to use RL for path planning is it can be modeled as an MDP intuitively, and the fact that it does not need massive labeled data for training. In most of the work in RL-based path planning, the agent's state space is considered as images or sensory information. While this is a reasonable approach to defining the state space on an RL agent, the shortcoming is the bigger state space. The bigger the state space, the more time to converge in the learning phase.

Lifelong learning [7] is a factor that is considered to keep any artificial intelligent agent learning and evolving continuously based on the various changes over a long period. Transfer Learning(TL) is an ML technique that uses previous experience to learn and perform well in a similar situation. A TL algorithm can efficiently make the RL agent dynamic with lifelong learning ability. Given the variations in the environment configuration, the transfer learning approach can be used to achieve lifelong learning. In this context, the source states are the old environment's topological information, and the target states are the new altered environment's topological information. There will be differences in the state space, and all the other RL-agent parameters will remain the same. TL approaches considering only the state difference between the source task, and target task is minimal. So a novel transfer learning technique that considers only state space difference is proposed in this article.

This work aims to design and develop a Deep Reinforcement Learning (DRL) based path planning framework. The proposed framework uses a Deep Q-Learning algorithm to learn the initial paths based on the topological map of the environment. A novel $\boldsymbol{\beta}$-decay TL algorithm is proposed to achieve lifelong learning. This algorithm uses an incremental learning approach to update the RL agent based on the changes in the environment dynamically for lifelong. The key contributions of the work are:

- Design and development of a scalable DQL path planning framework for a mobile service robot.
- Incorporating $\boldsymbol{\beta}$-decay TL algorithm to continuously evolve the RL agent based on environmental changes and achieve lifelong learning.
- The RL agent's scalability and the TL algorithm's learning efficiency are tested and proved.

The remainder of the paper is presented as follows; a summary of the related work in path planning is described in section II. The problem formulation is defined in section III. Section IV describes the proposed RL and TL framework in detail. Results and analysis of various test cases are depicted in section V. The article is concluded, and future directions are identified in the last section.

II. RECENT AND RELEVANT WORK

The path planning problem can be categorized into two classes of problems, global and local path planning [8]. Global path planning is planning a path from a source to a destination, given the environment map. Local planning is planning the path if an unforeseen scenario (obstacle) occurs while traversing the global path. Rapid development in artificial intelligence and machine learning techniques has led to using it to solve path planning problems [8], [9], [10]. Artificial intelligence techniques like fuzzy logic [11], neural networks [12], and hybrid solutions like neuro-fuzzy inference systems [13] are used for local path planning as there are very efficient in dynamic problem-solving. Evolutionary techniques like particle swarm optimization [14] and genetic algorithms [15] are used for global path planning to optimize the path length and planning time.

One of the recent advancements in path planning research is using Reinforcement learning-based techniques. Unlike other machine learning algorithms, an RL agent can be trained with a minimal dataset. This is one primary reason to use RL in the path planning domain. Q-learning(QL) and Deep Q-Learning (DQL) are two of the most used RL algorithms for path planning. In QL, the Q-value is calculated based on the Bellman equation. In [16] and [17] QL agent is used for planning the path. The drawback in Q-learning is scalability; as the environment size increases, the memory complexity also increases. In DQL, the Q-value is predicted using a neural network, which makes the system scalable. In [18], [19], and [20], planning a path in a grid world environment using DQL is presented. The environment information regarding the obstacle location is available, and the RL agent learns to avoid all the static obstacles and plan the path efficiently. DQL-based algorithm for dynamic environment path planning is proposed in [21] and [22]. A dynamic environment is considered to have moving obstacles, and the obstacle location is unknown. Another approach to explore in an unknown environment is to use sensor values to train the RL agent [23], [24], [25].

One critical factor in any machine-learning algorithm is convergence time, i.e., the time the agent takes to learn the task. There have been considerable efforts to reduce the convergence time of DRL-based path-planning algorithms. In [26], convergence time is reduced by pre-training the RL agent using a 2D simulator environment and later using this experience to train in a real-time 3D environment.

A Q-learning system with general information about the goal and current states is proposed in [27]. This knowledge is used efficiently while training to reduce the training time significantly. In article [28], the authors propose the use of RL and particle swarm optimization to increase the agent's convergence rate.

Researchers have made numerous attempts to solve lifelong learning in various domains. In [29], a notion similar to lifelong learning called concept drift is described, and a solution is proposed to use previous experience to learn new similar concepts. Correspondence learning is yet another attempt to use old experience in a new task efficiently. In [30], correspondence learning uses previously learned skills to solve new tasks in an Atari ping-pong game environment. Transfer learning(TL) is a technique that is applied to evolve machine learning models toward lifelong learning. A TL algorithm transfers the source task knowledge to a target model. In [31], RL and TL tracking in stationary environments is proposed. Key points to be considered while applying transfer knowledge are the source task and target task similarity, mapping between source states and target states, and the type of knowledge to be transferred [32], [33]. Applying TL between the source and target tasks with different states and actions, the challenge lies in mapping the source states to target states. In articles [34] and [35], authors proposed a simple approach to provide the mappings as subject experts manually. While this is a simple solution, there can be practical difficulties in obtaining human experts mapping in all the applications. Inter-task mapping can be considered as required or learned automatically to address this shortcoming. If no explicit mapping is required, the agent attempts to learn the abstraction of the MPD, which are invariant even though actions and state variables change [36], [37]. Another approach is to learn inter-task mapping automatically [38], [39] using novel mapping learning methods. Applying heuristics in transfer learning is proven to have reasonable acceleration in target task learning. In article [40], heuristics is obtained by exploring the environment. Obtaining the heuristics as different cases based on source task learning and using it in target learning is proposed in articles [41], [42].

III. PROBLEM DEFINITION

Consider a wheeled mobile robot performing autonomous tasks in an indoor service environment. To perform tasks autonomously, the robot should be capable of navigating from one location to another. A typical service robot is required to work in the environment for a long period, during which the environment configuration can change. Environment configuration is the location of all the static obstacles, doorways, pathways, etc. To cope-up with the environmental changes, the mobile robot should be able to perform lifelong learning. This will enable the robot to plan a path based on the current environment configuration rather than the old environment configuration. This can be achieved using Reinforcement learning.

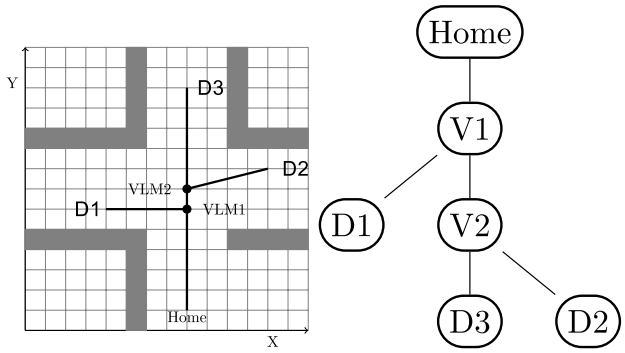


FIGURE 1. Sample environment and its topological map.

The RL agent will be the mobile robot, and the environment will be the service environment. The proposed path planner is based on the Deep Q-learning algorithm. The DQL agent will be pre-trained based on the topological map of the service environment. A topological map is a high-level connectivity tree with all the critical locations in the environment. The topological map can be generated manually or automatically as proposed in [43]. The techniques to generate a topological map are out of the scope of this work. There are many techniques in the literature to obtain a topological map of an environment. A few techniques to generate a topological map are explored in our previous works [43], [44], [45]. Fig. 1 depicts a sample environment with paths and its topological map. The Home nodes refer to the charging point for the robot, the "Vx" nodes are waypoint nodes, and the "Dx" nodes are the destination locations. The map is represented in the form of a ternary tree, each node in the map is a location, and all the destination locations are the leaf nodes. Each node in the tree can have a maximum of three branches (left, right, forward). The direction to reach from the parent node to a child node is encoded as the branch of a node. For example, if a parent node has a left child node, then reaching that node is by taking the left direction from the parent node.

Lifelong learning can be achieved using transfer learning in the proposed DQL agent. In a practical scenario, the changes in the environment will be gradual. For example, furniture can be added or removed, which might block a current path. Such changes will lead to environment configuration changes, and the topological map can be updated based on the new configuration [46]. The process of obtaining the new topological map is not in the scope of this article. Consider a permanent obstacle placed between node "home" and "V1" in the sample environment shown in Fig. 1; this will affect the path from these two nodes. The sample environment with obstacle and its topological map is depicted in Fig. 2 with dotted lines to indicate the change in path. Given this new topological map, the RL agent should re-learn the new environment. Training the RL agent to learn the new environment configuration from scratch will require more time than using the TL technique to jump-start the training process. This can be achieved by efficiently transferring the old knowledge and training the RL agent.

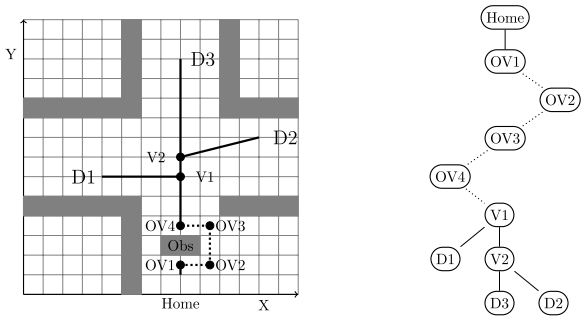


FIGURE 2. Sample environment and its topological map with obstacle.

IV. DESIGN OF THE FRAMEWORK

The proposed system is a path planner for a mobile service robot with lifelong learning ability. The overall system consists of two modules DRL module and the TL module. DRL module is used to train an RL agent with the initial environment configuration to plan the paths efficiently. TL module is used to re-train the RL agent if there is any configuration change in the environment.

A. DEEP REINFORCEMENT LEARNING FRAMEWORK

RL is a class of ML algorithms that uses an agent to learn a task by interacting with the environment through its actions. An agent with its current state s takes action a and reaches another state s'. The agent receives a reward of r for every action in the environment, which can be positive or negative. The reward will be based on the action and the agent's current state. The agent's ultimate goal is to maximize the reward that will indirectly train the agent to perform the task. A Deep Q-learning-based RL algorithm implements the path planner, and the mobile robot is the RL agent. DQL uses a neural network to predict the best action given the current state. The parameters of the DQL framework are initialized based on the topological map used to train the RL agent.

1) STATES - S

States are all the possible configurations of an RL agent. In the mobile robot domain, states can be the location of the mobile robot. The state space S of the agent can be defined set of all the nodes in the topological map. The current state can be defined as $s \in S$, where s is the robot's current location, a subset of all the possible locations.

2) ACTIONS - A

Actions are considered as the interaction of the agent in the environment. The result of such interaction is the agent's transition from one state to another. An action is defined as $a \in A$. The action space A is the set of all the possible actions a mobile robot can take, $A = (left, right, forward, backward)$.

3) REWARD - R

The reward system is the key ingredient of RL training that can make or break an agent. The reward space $R(S, A)$ is the set of all the possible rewards based on the current state and action. Equation (1) shows the reward function of the proposed RL agent. While the value of the reward is constant, it can be changed as long as the difference between the different rewards is maintained. A positive reward motivates the agent to perform a similar action in the future, and a negative reward is to avoid the action in the future. The highest reward r_1 is to reach the goal state $g \in G$ where G is the set of all the goal states $G \subset S$. The highest negative reward r_4 for reaching an unknown state $U \notin S$ is given. A reward r_3 is given to the agent to avoid looping. A reward r_2 is given as a step reward; this will converge the agent to find the shortest path possible.

$$r(s,a) = \begin{cases} r_1, & s = g \\ r_2, & s = \{x | x \notin (S \cap G)\} \\ r_3, & \text{s is previously visited location} \\ r_4, & s = U \end{cases} \tag{1}$$

4) Q-VALUE - Q(s, a)

Q-value is used to determine the quality of an action given the current state. The DQN neural network is trained to predict the Q-values for all the possible actions given a state $Q(s, A)$. The best action a is chosen based on $argmax(A)$.

5) DQN ARCHITECTURE

The DQN architecture consists of an input layer, two hidden layers, and an output layer with Adam optimizer, as shown in Fig. 3. The neural network configurations depend on the states and actions of the RL agent, as shown in (2) and (3). The total number of input neurons n depends on the number of states S and the number of goals G, i.e., the input to the neural network depends on the current state s and desired goal to reach g.The goal states $G \subset S$ denotes all the possible destination locations obtained from the topological map. Adding goal states as part of the input layer will enable the agent to find the optimal path to all the possible destinations. The output neurons m depend on the number of actions A. This represents Q-values for all the actions based on the current state. The best action is based on the output layer's maximum Q-value.

$$n = |S| + |G| \tag{2}$$

$$m = |A| \tag{3}$$

6) TRAINING

The neural network is trained using an experience replay buffer. Experience $e = (s, a, s', r)$ is stored after every iteration in the experience memory. A random experience batch is used for training. Batch size η is a hyper-parameter. Equation (4) shows the loss equation, which is the Bellman error derived from the Bellman equation for Q-Learning. $Q^*(s, a)$ is the predicted Q value. $maxQ'(s', a')$ is the maximum expected future reward based on the new state s' and all the possible actions. γ is the discount rate, a hyper-parameter controlling the weightage of the future

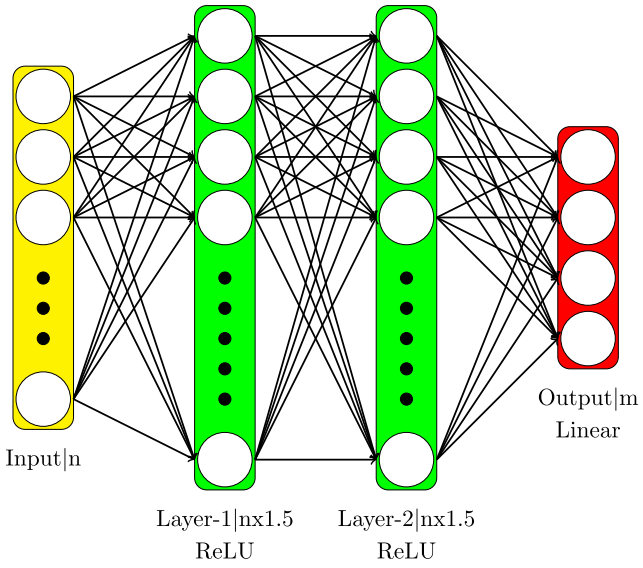


FIGURE 3. Proposed DQN architecture.

Algorithm 1 Experience Vs. Exploration Vs. Exploitation Pseducode

```
Initialize Source and Target DQN network
Initialize parameters - ϵ, β, episode_count
for episode <= episode_count do
    s = random(S)
    while s! = terminate_state do
        if random(0 − 1) < β then
            Q_value(s_source) = Source.Predict(s_source)
            a = a_max(Q_value(s_source))
        else if random(0 − 1) < ϵ then
            Q_value(s_target) = Target.Predict(s_target)
            a = a_max(Q_value(s_target))
        else
            a = randint(1 − 4)
        end if
    end while
end for
```

reward. An RL agent at the early stages of the training process should explore more to understand the environment better. After exploring enough, the agent should rely on the trained network, i.e., exploitation. The hyper-parameter ϵ manages the exploration and exploitation factor. To set the ϵ value high to explore initially and reduce it gradually to exploit, the ϵ-decay algorithm is used. The decay function in (5) is chosen to be an exponential function based on the ϵ, current training episode number, ϵ_min, ϵ_initial. The algorithm gradually reduces (decays) the ϵ value from ϵ_initial to ϵ_min. A training episode is terminated based on the current state s of the agent, (6) depicts all the terminating conditions.

$$Loss = (Q^*(s,a) - (r + \gamma max Q'(s',a')))^2 | e = (s,a,s',r) \tag{4}$$

$$\epsilon_{new} = \epsilon_{old} \cdot \exp^{current\_episode}$$

$$where, \quad \exp = total\_episodes \sqrt{\frac{\epsilon\_min}{\epsilon\_initial}} \tag{5}$$

$$s = \begin{cases} \text{terminal\_state}, & s = g \\ \text{terminal\_state}, & s = U \\ \text{terminal\_state}, & \text{total visited states} == |S| \\ \text{non-terminal\_state}, & \text{Otherwise} \end{cases} \tag{6}$$

B. β-DECAY LIFELONG LEARNING ALGORITHM

The proposed $\boldsymbol{\beta}$-decay algorithm is a transfer learning algorithm applied when a configuration change in the environment affects the trained DQL path planner. The Non-TL approach would be to retrain the new RL agent from scratch. The configuration changes are less likely to be major and will be gradual. This will affect the number of nodes in the topological tree, i.e., there will be only a change in the state space of the RL agent. Considering this fact, the Non-TL approach would be redundant and time-consuming as the old agent will have useful information that can be used to upgrade the new agent to incorporate the new environment configuration.TL approach will be used to re-train the new agent using the knowledge of the old agent hence reducing training time. TL approach considers the old agent path planner as the source task and the new agent path planner as the target task.

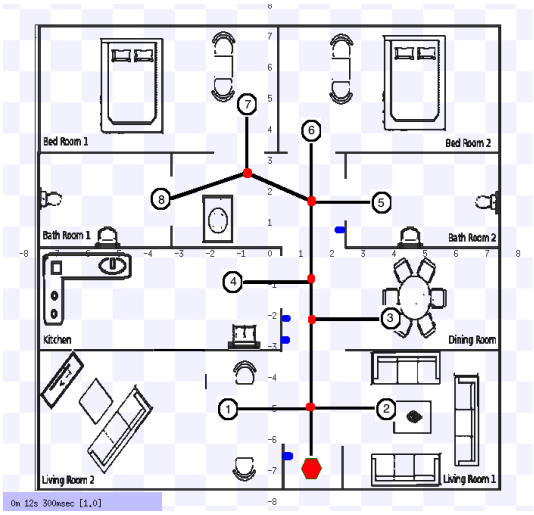
The proposed $\boldsymbol{\beta}$-decay TL algorithm transfers the knowledge of the source task to the target task based on the $\boldsymbol{\beta}$ factor. The target task agent will learn from the source task knowledge or ϵ-decay learning process described in IV-A6.

The higher the $\boldsymbol{\beta}$ value, the higher the possibility of transferring the source task knowledge. $\boldsymbol{\beta}$ is initialized based on (7). S_{target} is the state space of the target task and S_{source} is the state space of the source task. The $\boldsymbol{\beta}$ is initialized based on the probability of choosing a random state from the target state space that will be the same as the source state space. The decay factor is calculated as in (5). Based on the ϵ and $\boldsymbol{\beta}$, the proposed TL algorithm chooses between Experience vs. Exploration vs. Exploitation (EEE) as shown in algorithm 1.

$$\beta = \frac{|S_{target} \cap S_{source}|}{|S_{target}|} \tag{7}$$

V. RESULTS AND DISCUSSIONS

The DRL path planner is proposed to be implemented in two stages: pre-training and lifelong learning. The pre-training stage trains the RL agent to plan efficient paths based on the topological map. The lifelong learning stage uses the $\boldsymbol{\beta}$-decay Transfer Learning Algorithm to keep the RL agent in continuous learning mode to update its training based on environmental changes. The system is implemented in the ROS2 framework and tested using Turtlebot3 mobile in the Gazebo simulation environment. Two service environment



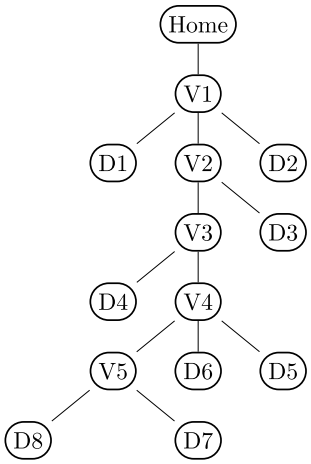
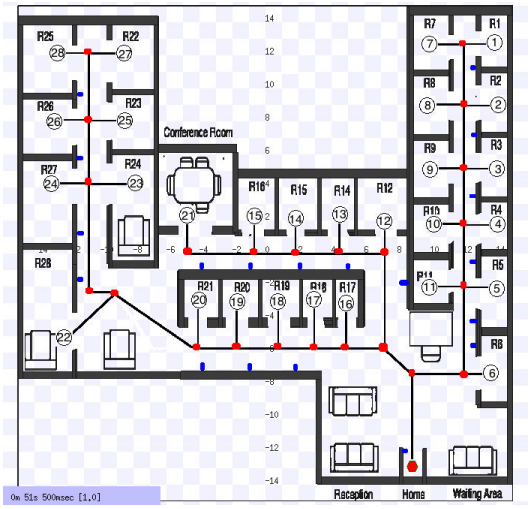


FIGURE 4. House layout and its topological map.



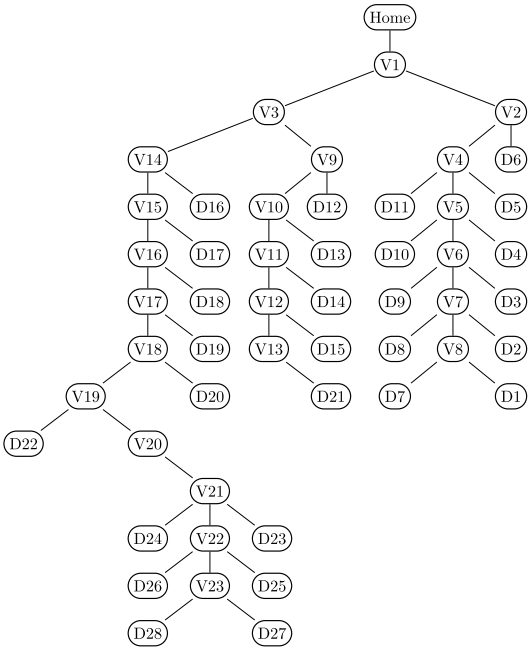


FIGURE 5. Hospital layout and its topological map.

and one benchmark maze environment that is different in terms of its size are chosen to implement the proposed path planner. The house layout has 8 destinations and 5 waypoints. The hospital layout has 28 destinations and 23 waypoints. The benchmark maze environment has 18 destinations and 88 waypoints. The environments and corresponding topological maps are depicted in Fig. 4 and 5. A virtual node, ''U'', is added to the topological map. This represents the unknown state of the agent that will be reached as a result of an action performed to reach a location not represented by the topological map. The performance of the DRL framework is measured based on its training and testing success rate. The TL module performance is measured based on three parameters, jumpstart, time-to-threshold, and time-to-converge.

A. DQL PATH PLANNER PRE-TRAINING

The DQL planner is pre-trained using the topological map of the environment. The system is trained and tested in two service environments to prove the path planner's scalability and generality.

1) SELECTION OF HYPER-PARAMETERS

One important step to train any RL agent is to tune the hyper-parameters, which enables the agent to learn the task efficiently. The hyper-parameters are α-learning rate, γ-discount factor, ϵ-range exploration factor, and replay memory batch size. The pace of the agent's learning is based on the learning rate. The learning rate is often chosen between values 0-1. A higher learning rate will result in less learning

TABLE 1. ϵ-range & batch size success rate.

Batch Size	ϵ-range			
	Destination-D1		Destination-D8	
	1-0.1	0.1-0.001	1-0.1	0.1-0.001
32	81.6%	48.4%	75.6%	43%
16	65.5%	47.8%	36.1%	31%

TABLE 2. Hyper-parameters.

Parameter	Values
α	0.001
γ	0.95
ϵ-range	(1-0.1)
Batch Size	32

time but at the cost of sub-optimal learning quality. A lower learning rate will result in optimal learning quality but at the cost of increased learning time. Since the proposed RL agent's trained for a service environment and is carried out offline, learning quality is chosen over learning time. Hence α is chosen as 0.001 to achieve stable and quality learning. The discount factor is used to evaluate the weight of future rewards. γ is chosen at 0.95 to give importance to future rewards while learning. The other factors, ϵ range and batch size are determined by performing single-goal training in the house layout.

Single goal DQL network differs only in the input layer size compared to the proposed DQL network. The input layer size $n = |S|$ for the single goal DQL equals the total number of nodes in the topological map. In the house layout, the RL agent is trained to reach two destinations, D1 and D8, for 500 episodes. Batch size values 32 and 16 were chosen as the NN is trained with numerical values. ϵ range is given as (initial-min), a range of numbers between min and max with exponential decay factor calculated based on (5). ϵ-ranges chosen are (1-0.1) and (0.1-0.001). The evaluation parameter is the training success rate which indicates the agent's convergence. It is calculated based on the ratio of successful and total training episodes. Table 1 shows the training success rates of all the training cases, which vary based on batch size and ϵ range.The test case with batch size 32 and ϵ range of (1-0.1) yielded higher success rates (marked in red). The final hyperparameters values are tabulated in Table 2.

2) HOUSE LAYOUT TRAINING

The RL agent is trained to learn paths to all the destinations in the house layout based on the topological map. The DQL architecture shown in Fig. 3 is used in the proposed system. The architecture has two hidden layers with some neurons twice the input layer size. This is determined by experimenting with different configurations. Table 3 shows the three different architectures considered to determine the ideal hidden layer size and its success rate after training for 2500 episodes. The difference in success rate is marginal; this can be because the agent is trained based on numerical data with fewer features. To gain more insight success rate is plotted and shown in Fig. 6. The success rate plot can be

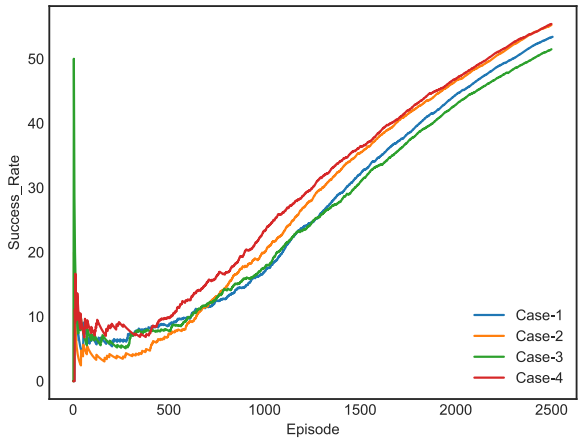


FIGURE 6. Different DQN architectures training success rate.

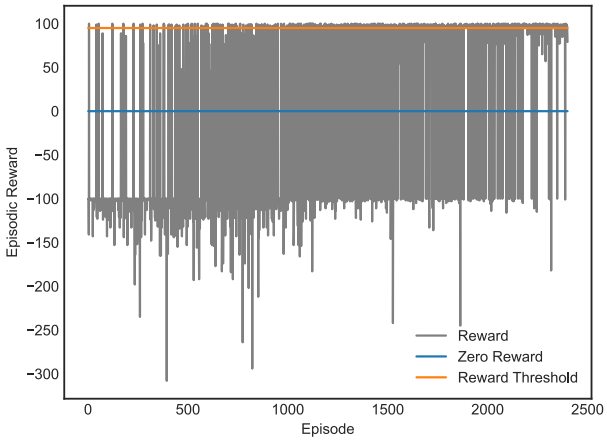


FIGURE 7. House layout case-2 episodic reward.

TABLE 3. Success rate of different DQL architecture configuration.

Test Case	Hidden Layer Size		Success Rate
	Neurons	Layers	
Case-1	24	2	53.3%
Case-2	48	2	55.2%
Case-3	24	4	51.4%

considered as the learning curve of the agent. From the plot, it can be concluded that case-2 performs better than other cases. Case-2 configuration is 48 neurons with two layers. Based on (2), input layer size n for house layout is calculated to be $n = 23(15(\text{states count}) + 8(\text{goal states count}))$, so the minimum hidden layer size is 1.5 times the input layer size, i.e., 34. This is the same as case-2, validating the architecture configuration proposed in IV-A5. The rewards used for the training are shown in (8).

$$r(s,a) = \begin{cases} 100, & s = g \\ -1, & s = \{x | x \notin (S \cap G)\} \\ -10, & \text{s is previously visited location} \\ -100, & s = U \end{cases} \tag{8}$$

The episodic reward collected during the case-2 training process is shown in Fig. 7 along with zero and threshold reward lines. The threshold reward is calculated based on the difference between the goal reward and the critical

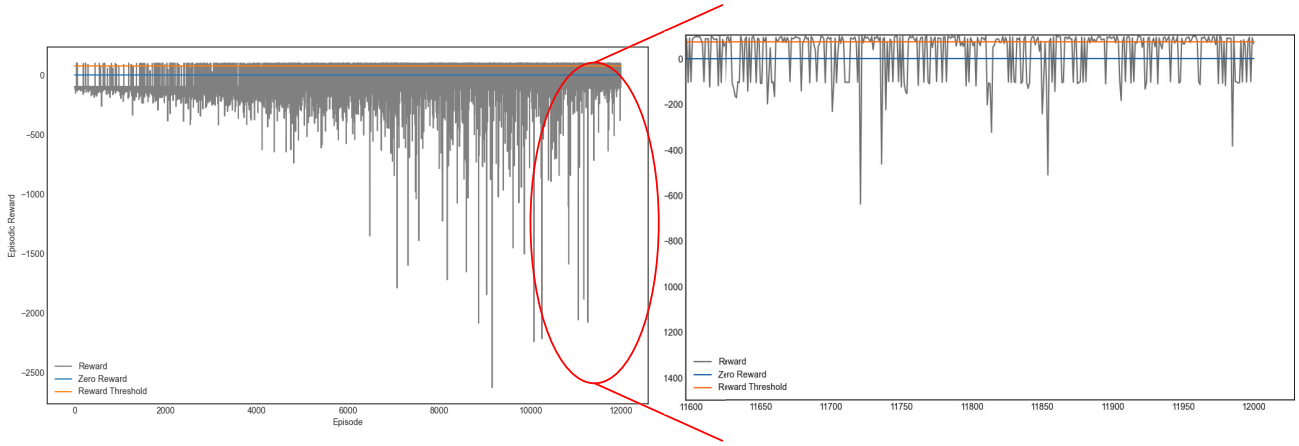


FIGURE 8. Hospital layout episodic reward[Left], Hospital layout episodic reward(last 2000 episodes)[Right].

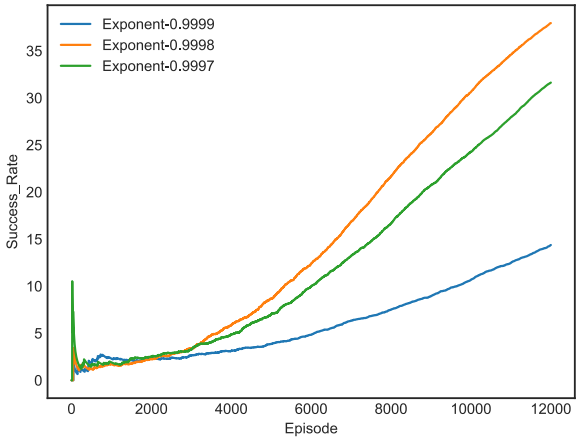


FIGURE 9. Reasoning for exponent.

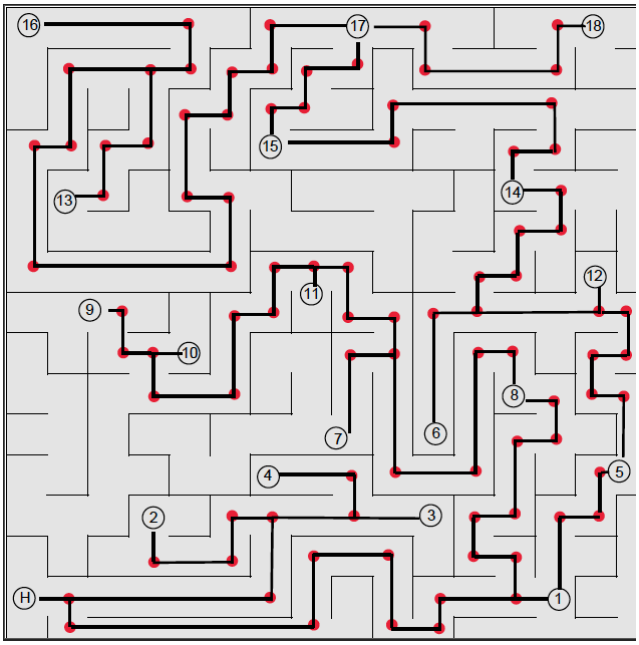


FIGURE 10. 2D pathfinding benchmark maze environment with paths.

path. A critical path is the biggest path (in terms of node count) among efficient paths to the destinations. Analyzing the reward plot reveals two major factors to evaluate the minimum number of training episodes required to train the agent effectively. First, the minimum number of episodes required for training is based on the frequency reduction of the negative rewards about the zero reward line. Second, training efficiency is based on the agent by comparing episodic reward about the threshold reward line. The negative reward frequency reduction and increased frequency of attaining episodic rewards about threshold reward are achieved in 2200 episodes. Hence, 2200 episodes are considered the minimum training episode to effectively train the RL agent in the house layout.

3) HOSPITAL LAYOUT TRAINING

To verify the scalability of the proposed DQL path planner, it is used to train the hospital layout, another service environment. The hospital layout is a bigger environment compared to the house layout. The DQL architecture is the same as the case-2 type in the house layout. The input layer size n for hospital layout is calculated to be $n = 81(53(\text{states count}) + 28(\text{goal states count}))$, two hidden layers with the size of 128 neurons, and output layer size $m = 4$ are chosen. The training hyper-parameters are the same as in Table 2. A plot on episodic reward during the training process is shown in Fig. 8. The total number of training episodes required is over 10,000, so the plot on the left is too small to analyze the information. A scaled version of the same plot with the last 600 episodes of training data is depicted on the right. The RL agent negative reward frequency reduction and frequency of attaining episodic rewards to threshold reward is achieved in 11850 episodes. Hence, the nearest upper-rounded number, 12000, is considered the minimum training episode to effectively train the RL agent in the hospital layout.

In both the service environment, the ϵ exponent decay factor is calculated based on (5). For a hospital layout with

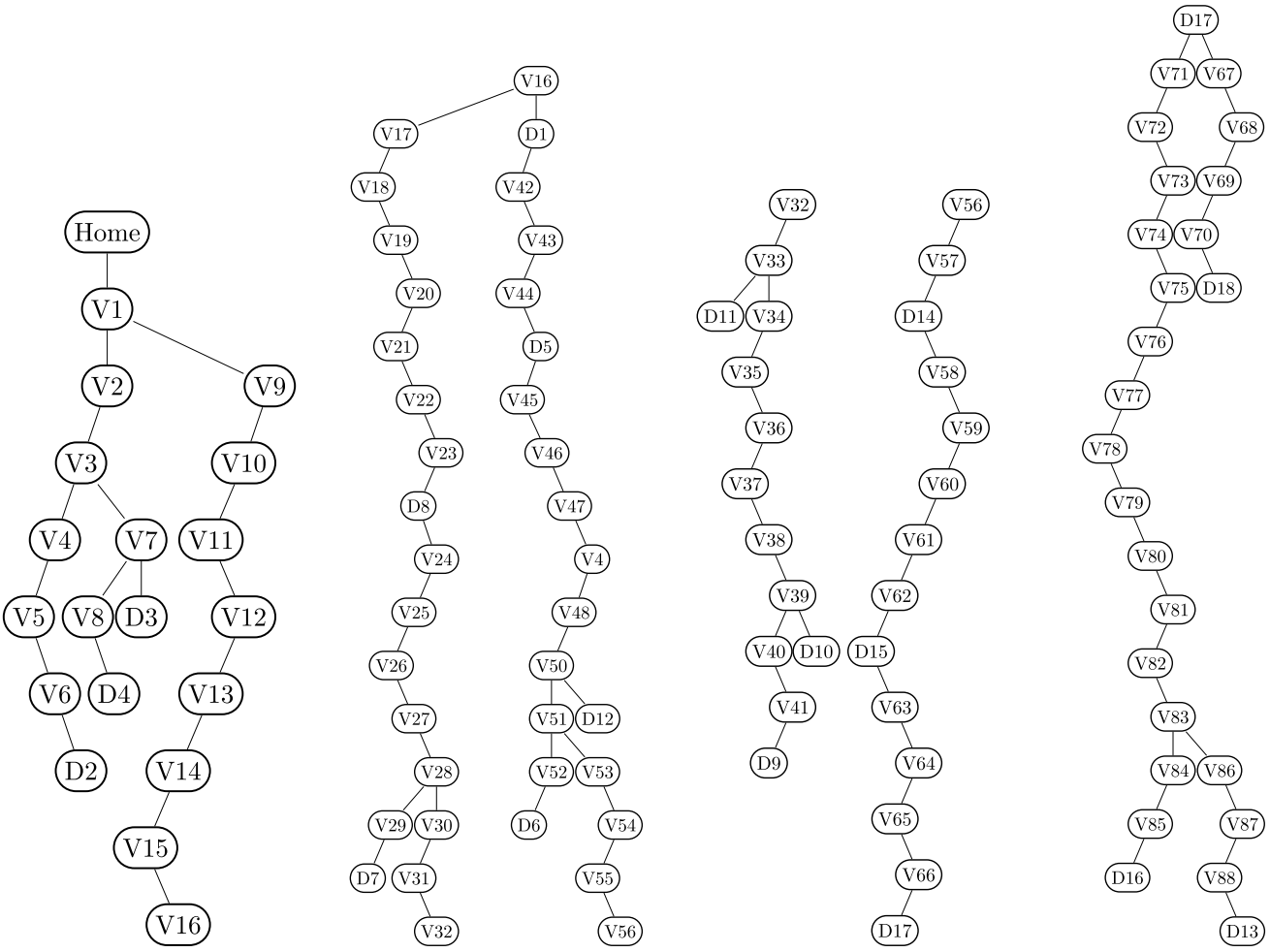


FIGURE 11. Benchmark maze environment path in four parts).

12000 episodes of training, the proposed exponent value is 0.9998. This theory is verified by training the agent in the hospital layout with various exponents close to the proposed exponent. Fig. 9 depicts the training success rate plot for three exponent values. From the plot, it can be concluded that the learning with an exponent value of 0.9998 is efficient compared to other values.

4) GENERALIZED PARAMETERS

The DQN architecture parameters are chosen based on the topological map properties for successfully training the RL agent in two different service environments. Table 4 tabulates all the model parameters, training success rate, and generalized parameters for both environment models. The generalized parameter for all the DQN parameters is derived as the function of topological map properties.

5) TESTING

The proposed DQL path planner is successfully trained in two service environments. To prove that the model is generic and scalable, it is essential to test the model for its ability to plan paths for various destinations given a source. The test cases considered are all the destination nodes in the topological map to be sources and destinations for path planning. The model is evaluated based on the testing success rate, calculated as the ratio of the successful and total number of test cases. The testing success rate is more than 98% for both environments.

TABLE 4. Proposed DQL model parameters and generalized parameters.

Parameter	House Layout	Hospital Layout	Generalized Parameters
Input Neurons	23	81	$\|S\| + \|G\|$
Output Neurons	4	4	$\|A\|$
Min. Hidden Layers x Neurons	2x48	2x128	1.5 x Input Neurons
Batch Size	32	32	Constant
ϵ-Range	1-0.1	1-0.1	Constant
ϵ-Decay Exponent	0.999	0.9998	Equation (5)
Testing Success Rate	100%	98.4%	-

To further verify the correctness of the generalized parameters mentioned in Table 4, the proposed RL agent is tested using a 2D pathfinding maze benchmark map from [47]. These benchmark maps have been used by researchers worldwide for evaluating path-planning algorithms [48], [49], [50], [51]. Fig. 10 depicts the benchmark maze environment with the paths and random destinations. The topological map for the benchmark maze is shown in Fig. 11

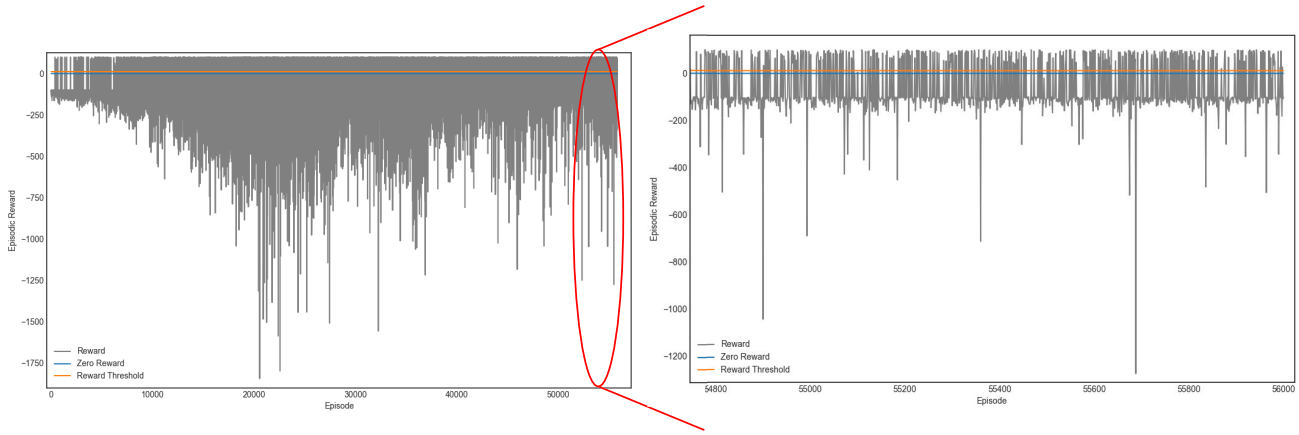


FIGURE 12. Benchmark maze layout episodic reward[Left], Benchmark maze layout episodic reward(last 1200 episodes)[Right].

with 18 destinations and 88 waypoints, including home and unknown nodes. There are 108 nodes in total, which is 7x and 2x times bigger compared to the house and hospital layouts, respectively. The topological map is too big to fit on a single page and split into three parts. The DQN architecture to train the RL agent for the maze environment is based on the generalized parameters tabulated in Table 4. The input layer size n for hospital layout is calculated to be $n = 126(108(\text{states count}) + 18(\text{goal states count}))$, two hidden layers with the size of 190 neurons, and output layer size $m = 4$ are chosen. The training hyper-parameters are the same as in Table 2. The RL agent was trained for 56000 episodes with the ϵ-decay factor as 0.999956 calculated based on (5). The episodic reward plot is shown in Fig. 12. The episodic reward was increasingly better as the training episodes reached 56000. The testing success rate is 98.3% for the maze layout. This is on par with the hospital and the house layouts. Based on the generalized parameters in Table 4, it can be concluded that the proposed DQL system is general and scalable.

In the work [48], a modified Q-learning approach is proposed to overcome the drawback of increased convergence time of the traditional Q-learning approach. In this article, the authors have used the benchmark maze environment to compare their proposed approach with traditional Q-learning and other path-planning approaches. The system was trained from a fixed starting point and fixed ending point for over 30 runs, and the averaged comparison metrics were evaluated. The starting point considered is the same as the Home location, and the ending point is D18. Even though in this work, a topological map with 19 different destinations is used to train the RL agent. The final path generated by the RL agent from Home to D18 is shown in Fig. 13, and it is the same as the modified Q-learning approach [48].

The house layout DQL agent is also tested in the mobile robot Turtlebot3 using the Robotic Operating System (ROS) framework and the Gazebo simulator. Fig. 14 depicts the two paths, Home to D5 and Home to D4, autonomously generated by the path planner agent.

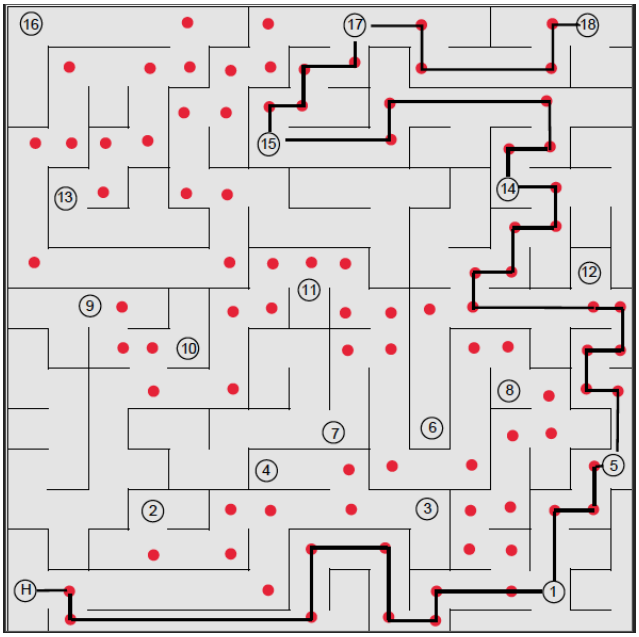


FIGURE 13. Path generated from Home to D18 in benchmark maze environment.

B. TRANSFER LEARNING AND TESTING

The proposed $\boldsymbol{\beta}$-decay transfer learning algorithm is tested by adding an obstacle in the house layout gazebo environment. The obstacle is placed between the V1 and V2 nodes of the house layout. Fig. 15 depicts the house layout with the cuboid obstacle in the Gazebo simulator environment and its topological map. The number of nodes in the topological map is increased, and all the new nodes are highlighted in red with dotted edges for their connections. This changes the environment's configuration, making the pre-trained DQL agent inefficient in this new environment.

The $\boldsymbol{\beta}$-decay transfer learning algorithm transfers knowledge of the source path planner (pre-trained agent) to the target path planner (new agent). To validate the proposed method of initializing the $\boldsymbol{\beta} = 0.78$ calculated based on (7) and its decay exponent for 1300 episodes $e = 0.9984$ based

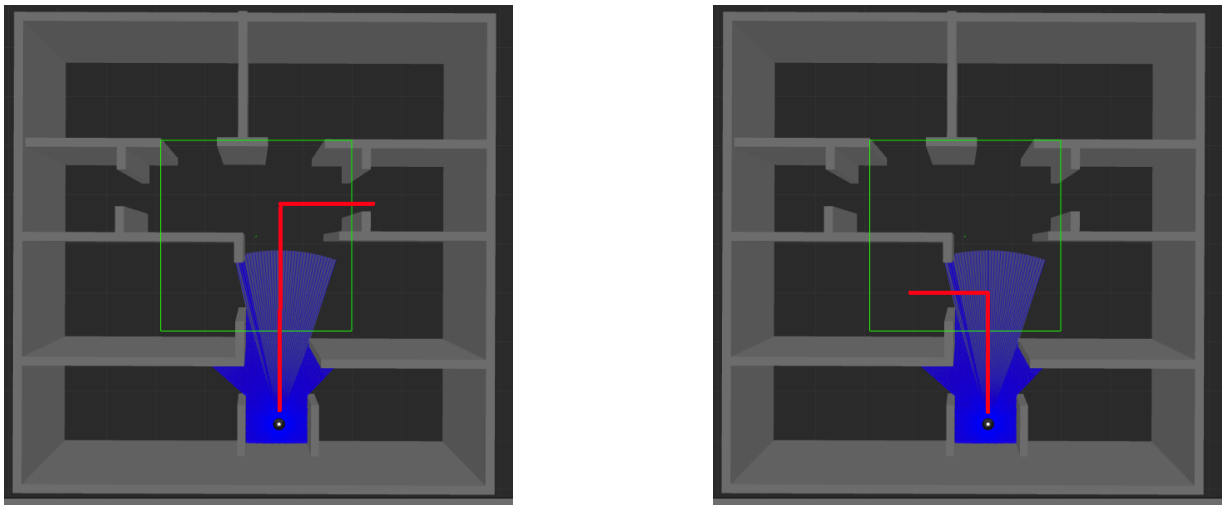


FIGURE 14. Simulator paths(in red) for home to D5 (left) and D4 (right).

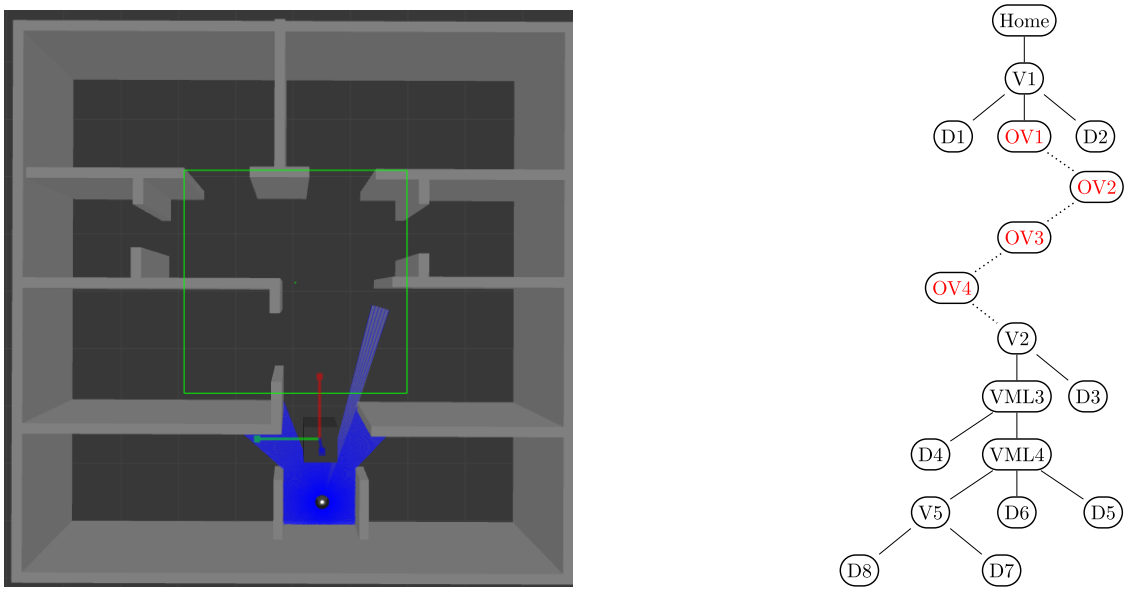


FIGURE 15. Obstacle added between V1 and V2 in house layout and its topological map.

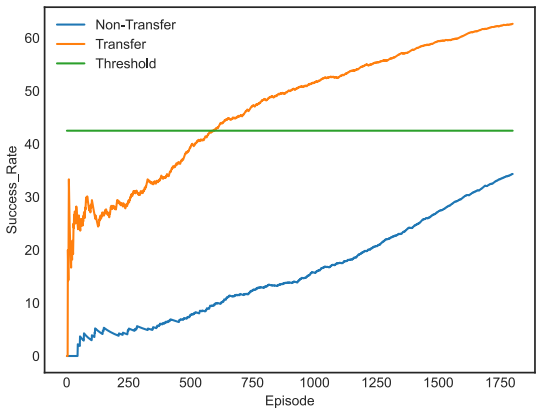


FIGURE 16. Training success rate for transfer vs. non-transfer training.

on (5) by replacing ϵ with $\boldsymbol{\beta}$. The DQL agent is trained using three different TL algorithms, without-$\boldsymbol{\beta}$, constant-$\boldsymbol{\beta}$, and proposed $\boldsymbol{\beta}$-decay. Table 5 shows the training success rate for the three TL algorithms after 500 training episodes. The proposed $\boldsymbol{\beta}$-decay TL algorithm outperforms the other two.

The $\boldsymbol{\beta}$-decay transfer learning algorithm transfers knowledge of the source path planner (pre-trained agent) to the target path planner (new agent). To validate the proposed method of initializing the $\boldsymbol{\beta} = 0.78$ calculated based on (7) and its decay exponent for 1300 episodes $e = 0.9984$ based on (5) by replacing ϵ with $\boldsymbol{\beta}$. The DQL agent is trained using three different TL algorithms, without-$\boldsymbol{\beta}$, constant-$\boldsymbol{\beta}$, and proposed $\boldsymbol{\beta}$-decay. Table 5 shows the training success rate for the three TL algorithms after 500 training episodes. The proposed $\boldsymbol{\beta}$-decay TL algorithm outperforms the other two.

The DQL agent is trained with and without transfer learning to quantitatively evaluate the proposed transfer learning approach. The evaluation metrics to evaluate the

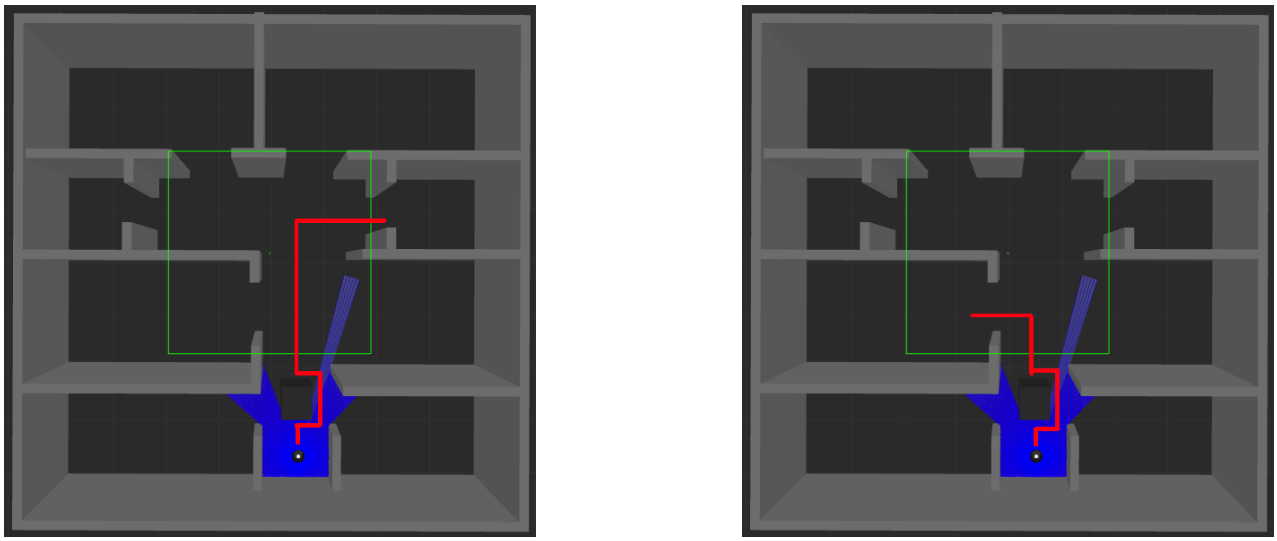


FIGURE 17. β-decay agent simulator paths(in red) for Home to D5 (left) and D4 (right) in new environment (with obstacle).

TABLE 5. Testing success rate comparison of TL algorithms.

Algorithm	Testing Success Rate
No-beta	7.4%
Constant-beta	41%
Proposed beta-decay	76%

efficiency of the TL algorithm are jumpstart, time-to-threshold, and time-to-converge. To evaluate the first two metrics success rate of the training for each training episode is analyzed. The success rate plot is shown in Fig. 16. The plot's overall view shows that the training with a transfer outperforms non-transfer training. Jumpstart is the agent's success rate in the initial few training episodes, and it can be calculated based on the average success rate of the first 100 training episodes. The time-to-threshold metric evaluates the learning performance of the agent by the time it takes to reach the threshold performance. The threshold value chosen is based on the agent training in the old environment. As there is only a small incremental change in the new environment compared to the old environment, this consideration is reasonable and logical. The threshold success rate value after 1800 training episodes is 42.5%. The time-to-converge metric will indicate the minimum number of training episodes required to learn the environment for path planning. This can be evaluated based on the testing success rate. Testing is performed by the agent's ability to plan the path with source and destinations as destination locations. Table 6 shows the success rate comparison between transfer and non-transfer learning-based learning. The transfer learning approach attains a 100% success rate after 1300 episodes, whereas non-transfer learning attains a 78.5% success rate after 2500 episodes.

Table 7 compares all the evaluation metrics between transfer and non-transfer DQL agents with performance improvement factors. In all the metrics, the DQL agent with proposed β-decay transfer learning outperforms the non-transfer agent. The convergence time of the transfer agent is more than two times faster than the non-transfer agent. The DQL agent with the proposed system is tested in the Gazebo simulator to navigate two paths, home to D5 and home to D4. Fig. 17 depicts the paths planned by the path planner by successfully avoiding the obstacles.

TABLE 6. Testing success rate of transfer and non-transfer training.

Episodes	TL	Non-TL
500	26.7%	21.4%
1000	79%	69%
1100	89%	71%
1200	91%	72%
1300	100%	73%
1400	100%	74%
1500	100%	75%
2000	100%	75%
2500	100%	79%

TABLE 7. Transfer learning evaluation matrices comparison.

Metric	Units	Non-Transfer	Transfer	Improvement Factor
Jumpstart	Training Success Rate	1%	20%	20x
Time to Threshold	Training Episodes	600	>1800	>3x
Time to Converge	Training Episodes	1300	>2500	>1.9x

VI. CONCLUSION AND FUTURE SCOPE

A generic and scalable RL-based mobile service robot path planner with transfer learning is proposed in this work. Once a service robot is deployed, it is expected to work for lifelong. A typical path planner will only consider the initial configuration of the environment to plan the path. But a service environment is prone to small gradual changes over some time. A deep Q-learning-based path planner

with novel $\boldsymbol{\beta}$-decay transfer learning for lifelong learning ability is proposed and implemented. The DQL path planner is initially pre-trained in a service environment using its topological map. If there are any changes in the environment's configuration, the proposed TL algorithm trains a new DQL agent by efficiently transferring the knowledge of the old agent. The DQL agent is trained in two service environments: house and hospital layouts, with a testing success rate of 100% and 98%, respectively. The hospital environment is almost three times larger than the house environment concerning the number of nodes and destinations. This proves that the proposed path planner is scalable. The DQL agent's generality factor is derived for all its architecture parameters and training hyper-parameters. They are the same or dependent on the environment's topological map, proving the system is generic. The path planner is successfully tested in the ROS framework with the Gazebo simulator using the turtlebot3 mobile robot.

The ability to lifelong learning is tested by adding an obstacle in the house environment. The proposed TL algorithm uses the Experience vs. Exploration vs. Exploitation (EEE) logic based on the β-decay factor. The correctness of the proposed TL algorithm's $\boldsymbol{\beta}$ initial value and decay factor is evaluated by comparing it with non-$\boldsymbol{\beta}$ TL and constant $\boldsymbol{\beta}$ TL. It is found to be better than the other two techniques. The efficiency of the TL algorithm is evaluated based on the metrics: jumpstart, time-to-threshold, and time-to-converge. The evaluation metrics are evaluated in comparison with the Non-TL based agent. The proposed TL method's speed-up factor is 20x more in jumpstart testing success rate, more than three times faster in achieving time-to-threshold, and converging more than two times faster.

Implementing and testing the proposed system in a real-time service robot is one future extension of this work. Methods to improve the efficiency of the TL algorithm can be explored. The $\boldsymbol{\beta}$ factor is a stochastic factor initialized based on the total number of similar states in the environment. The similarity is identified based on the state names obtained from the topological map. Identifying the similarity based on state variables can increase the efficiency of the TL algorithm.

REFERENCES

[1] A. Loganathan and N. S. Ahmad, "A systematic review on recent advances in autonomous mobile robot navigation," *Eng. Sci. Technol., Int. J.*, vol. 40, Apr. 2023, Art. no. 101343. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2215098623000204

[2] F. Rubio, F. Valero, and C. Llopis-Albert, "A review of mobile robots: Concepts, methods, theoretical framework, and applications," *Int. J. Adv. Robot. Syst.*, vol. 16, no. 2, pp. 1–22, 2019, doi: 10.1177/1729881419839596.

[3] A. A. S. Gunawan, B. Clemons, I. F. Halim, K. Anderson, and M. P. Adianti, "Development of e-butler: Introduction of robot system in hospitality with mobile application," *Proc. Comput. Sci.*, vol. 216, pp. 67–76, Jan. 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050922021901

[4] H. Kivrak, F. Cakmak, H. Kose, and S. Yavuz, "Social navigation framework for assistive robots in human inhabited unknown environments," *Eng. Sci. Technol., Int. J.*, vol. 24, no. 2, pp. 284–298, Apr. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2215098620308727

[5] D. Nakhaeinia, S. Tang, S. Noor, and O. Motlagh, "A review of control architectures for autonomous navigation of mobile robots," *Int. J. Phys. Sci.*, vol. 6, pp. 169–174, Jan. 2011.

[6] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[7] S. Thrun, "Is learning the n-th thing any easier than learning the first?" in *Advances in Neural Information Processing Systems*, vol. 8, D. Touretzky, M. Mozer, and M. Hasselmo, Eds. Cambridge, MA, USA: MIT Press, 1995.

[8] L. Liu, X. Wang, X. Yang, H. Liu, J. Li, and P. Wang, "Path planning techniques for mobile robots: Review and prospect," *Expert Syst. Appl.*, vol. 227, Oct. 2023, Art. no. 120254. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S095741742300756X

[9] S. Campbell, N. O'Mahony, A. Carvalho, L. Krpalkova, D. Riordan, and J. Walsh, "Path planning techniques for mobile robots a review," in *Proc. 6th Int. Conf. Mechatronics Robot. Eng. (ICMRE)*, Feb. 2020, pp. 12–16.

[10] J. R. Sánchez-Ibáñez, C. J. Pérez-del-Pulgar, and A. García-Cerezo, "Path planning for autonomous mobile robots: A review," *Sensors*, vol. 21, no. 23, p. 7898, Nov. 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/23/7898

[11] M. Hank and M. Haddad, "A hybrid approach for autonomous navigation of mobile robots in partially-known environments," *Robot. Auton. Syst.*, vol. 86, pp. 113–127, Dec. 2016.

[12] R. S. Nair and P. Supriya, "Robotic path planning using recurrent neural networks," in *Proc. 11th Int. Conf. Comput., Commun. Netw. Technol. (ICCCNT)*, Jul. 2020, pp. 1–5.

[13] A. Pandey, S. Kumar, K. K. Pandey, and D. R. Parhi, "Mobile robot navigation in unknown static environments using ANFIS controller," *Perspect. Sci.*, vol. 8, pp. 421–423, Sep. 2016.

[14] H. S. Dewang, P. K. Mohanty, and S. Kundu, "A robust path planning for mobile robot using smart particle swarm optimization," *Proc. Comput. Sci.*, vol. 133, pp. 290–297, Jan. 2018.

[15] W. Rahmaniar and A. E. Rakhmania, "Mobile robot path planning in a trajectory with multiple obstacles using genetic algorithms," *J. Robot. Control*, vol. 3, no. 1, pp. 1–7, Jun. 2021. [Online]. Available: https://journal.umy.ac.id/index.php/jrc/article/view/11024

[16] L. M. Zamstein, A. A. Arroyo, and E. M. Schwartz, "Koolio: Path planning using reinforcement learning on a real robot platform," in *Proc. Florida Conf. Recent Adv. Robot. (FCRAR)*, Miami, FL, USA, May 2006, pp. 1–4.

[17] A. A. N. Kumaar and S. Kochuvila, "Reinforcement learning based path planning using a topological map for mobile service robot," in *Proc. IEEE Int. Conf. Electron., Comput. Commun. Technol. (CONECCT)*, Jul. 2023.

[18] A. I. Panov, K. S. Yakovlev, and R. Suvorov, "Grid path planning with deep reinforcement learning: Preliminary results," *Proc. Comput. Sci.*, vol. 123, pp. 347–353, Jan. 2018.

[19] Y. Hu, L. Yang, and Y. Lou, "Path planning with Q-learning," *J. Phys., Conf. Ser.*, vol. 1948, no. 1, Jun. 2021, Art. no. 012038.

[20] G. Pan, Y. Xiang, X. Wang, Z. Yu, and X. Zhou, "Research on path planning algorithm of mobile robot based on reinforcement learning," *Soft Comput.*, vol. 26, no. 18, pp. 8961–8970, Sep. 2022, doi: 10.1007/s00500-022-07293-4.

[21] G. R. Pennock, X. Lei, Z. Zhang, and P. Dong, "Dynamic path planning of unknown environment based on deep reinforcement learning," *J. Robot.*, vol. 2018, Sep. 2018, Art. no. 5781591.

[22] H. Li, Q. Zhang, and D. Zhao, "Deep reinforcement learning-based automatic exploration for navigation in unknown environment," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 6, pp. 2064–2076, Jun. 2020.

[23] A. Balachandran, A. S. Lal, and P. Sreedharan, "Autonomous navigation of an AMR using deep reinforcement learning in a warehouse environment," in *Proc. IEEE 2nd Mysore Sub Sect. Int. Conf. (MysuruCon)*, Oct. 2022, pp. 1–5.

[24] K. Rajakani, Z. Liu, Q. Wang, and B. Yang, "Reinforcement learning-based path planning algorithm for mobile robots," *Wireless Commun. Mobile Comput.*, vol. 2022, May 2022, Art. no. 1859020, doi: 10.1155/2022/1859020.

[25] A. Buitrago-Martínez, R. F. De la Rosa, and F. Lozano-Martínez, "Hierarchical reinforcement learning approach for motion planning in mobile robotics," in *Proc. Latin Amer. Robot. Symp. Competition*, Oct. 2013, pp. 83–88.

[26] J. Gao, W. Ye, J. Guo, and Z. Li, "Deep reinforcement learning for indoor mobile robot path planning," *Sensors*, vol. 20, no. 19, p. 5493, Sep. 2020.

[27] A. Konar, I. G. Chakraborty, S. J. Singh, L. C. Jain, and A. K. Nagar, "A deterministic improved Q-learning for path planning of a mobile robot," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 43, no. 5, pp. 1141–1153, Sep. 2013.
[28] P. K. Das, H. S. Behera, and B. K. Panigrahi, "Intelligent-based multi-robot path planning inspired by improved classical Q-learning and improved particle swarm optimization with perturbed velocity," *Eng. Sci. Technol., Int. J.*, vol. 19, no. 1, pp. 651–669, Mar. 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2215098615001548
[29] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts," *Mach. Learn.*, vol. 23, no. 1, pp. 69–101, Apr. 1996, doi: 10.1007/BF00116900.
[30] M. Ruman and T. V. Guy, "Learning state correspondence of reinforcement learning tasks for knowledge transfer," 2022, *arXiv:2209.06604*.
[31] R. S. Sutton, A. Koop, and D. Silver, "On the role of tracking in stationary environments," in *Proc. 24th Int. Conf. Mach. Learn. (ICML)*. New York, NY, USA: Association for Computing Machinery, 2007, pp. 871–878, doi: 10.1145/1273496.1273606.
[32] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *J. Mach. Learn. Res.*, vol. 10, no. 56, pp. 1633–1685, 2009. [Online]. Available: http://jmlr.org/papers/v10/taylor09a.html
[33] Z. Zhu, K. Lin, A. K. Jain, and J. Zhou, "Transfer learning in deep reinforcement learning: A survey," 2020, *arXiv:2009.07888*.
[34] L. Torrey, J. Shavlik, T. Walker, and R. Maclin, "Relational macros for transfer in reinforcement learning," in *Inductive Logic Programming*, H. Blockeel, J. Ramon, J. Shavlik, and P. Tadepalli, Eds. Berlin, Germany: Springer, 2008, pp. 254–268.
[35] M. E. Taylor, N. K. Jong, and P. Stone, "Transferring instances for model-based reinforcement learning," in *Machine Learning and Knowledge Discovery in Databases*, W. Daelemans, B. Goethals, and K. Morik, Eds. Berlin, Germany: Springer, 2008, pp. 488–505.
[36] B. Banerjee and P. Stone, "General game learning using knowledge transfer," in *Proc. 20th Int. Joint Conf. Artif. Intell. (IJCAI)*. San Francisco, CA, USA: Morgan Kaufmann, 2007, pp. 672–677.
[37] G. Konidaris and A. Barto, "Building portable options: Skill transfer in reinforcement learning," in *Proc. 20th Int. Joint Conf. Artif. Intell. (IJCAI)*. San Francisco, CA, USA: Morgan Kaufmann, 2007, pp. 895–900.
[38] G. Kuhlmann and P. Stone, "Graph-based domain mapping for transfer learning in general games," in *Proc. 18th Eur. Conf. Mach. Learn. (ECML)*. Berlin, Germany: Springer-Verlag, 2007, pp. 188–200.
[39] E. Talvitie and S. Singh, "An experts algorithm for transfer learning," in *Proc. Int. Joint Conf. Artif. Intell.*, 2007, pp. 1065–1070.
[40] R. A. C. Bianchi, C. H. C. Ribeiro, and A. H. R. Costa, "Accelerating autonomous learning by using heuristic selection of actions," *J. Heuristics*, vol. 14, no. 2, pp. 135–168, Apr. 2008, doi: 10.1007/s10732-007-9031-5.
[41] R. A. C. Bianchi, L. A. Celiberto, P. E. Santos, J. P. Matsuura, and R. L. de Mantaras, "Transferring knowledge as heuristics in reinforcement learning: A case-based approach," *Artif. Intell.*, vol. 226, pp. 102–121, Sep. 2015.
[42] L. A. Celiberto Jr., J. P. Matsuura, R. L. de Màntaras, and R. A. C. Bianchi, "Using transfer learning to speed-up reinforcement learning: A cased-based approach," in *Proc. Latin Amer. Robot. Symp. Intell. Robot. Meeting*, Oct. 2010, pp. 55–60.
[43] A. A. N. Kumaar and T. S. B. Sudarshan, "Path mapping and planning with partially known paths using hierarchical state machine for service robot," in *Robot Intelligence Technology and Applications 3*. Cham, Switzerland: Springer, 2015, pp. 45–55.
[44] A. A. N. Kumaar and T. S. B. Sudarshan, "Mobile robot programming by demonstration," in *Proc. 4th Int. Conf. Emerg. Trends Eng. Technol.*, Nov. 2011, pp. 206–209.
[45] A. A. N. Kumaar, S. Kochuvila, and S. R. Nagaraja, "A scalable tree based path planning for a service robot," *J. Autom., Mobile Robot. Intell. Syst.*, vol. 16, no. 1, pp. 31–45, Mar. 2023.
[46] A. A. N. Kumaar and T. S. B. Sudarshan, "Learning from demonstration with state based obstacle avoidance for mobile service robots," *Appl. Mech. Mater.*, vol. 394, pp. 448–455, Sep. 2013.
[47] N. R. Sturtevant, "Benchmarks for grid-based pathfinding," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 2, pp. 144–148, Jun. 2012.
[48] E. S. Low, P. Ong, C. Y. Low, and R. Omar, "Modified Q-learning with distance metric and virtual target on path planning of mobile robot," *Expert Syst. Appl.*, vol. 199, Aug. 2022, Art. no. 117191.
[49] T. Lei, T. Sellers, C. Luo, and L. Zhang, "A bio-inspired neural network approach to robot navigation and mapping with nature-inspired algorithms," in *Proc. 13th Int. Conf. Adv. Swarm Intell. (ICSI)*, Xi'an, China. Berlin, Germany: Springer-Verlag, Jul. 2022, pp. 3–16.
[50] A. Toma, H. A. Jaafar, H. Hsueh, S. James, D. Lenton, R. Clark, and S. Saeedi, "Waypoint planning networks," 2021, *arXiv:2105.00312*.
[51] M. Zhao, H. Lu, S. Yang, and F. Guo, "The experience-memory Q-learning algorithm for robot path planning in unknown environment," *IEEE Access*, vol. 8, pp. 47824–47844, 2020.

A. A. NIPPUN KUMAAR received the Diploma degree in electronics and communication engineering from the Thiyagarajar Polytechnic College, Salem, Tamil Nadu, India, in 2003, the bachelor's and B.E. (Hons.) degrees in electronics and communication engineering from the Sona College of Technology, Salem, in 2006, and the M.Tech. degree (Hons.) in embedded systems from the School of Engineering, Amrita Vishwa Vidyapeetham, Bengaluru Campus. He is currently pursuing the Ph.D. degree in service robotics. He is also an Assistant Professor (Sr.Gr) with the Department of Computer Science, Amrita School of Computing. His research interests include robotics, swarm robotics, embedded systems, the Internet of Things (IoT), and machine learning.

SREEJA KOCHUVILA received the M.Tech. degree in instrumentation and control systems from the National Institute of Technology, Calicut, Kerala, India, in 2001, and the Ph.D. degree from the School of Engineering, Amrita Vishwa Vidyapeetham, Bengaluru Campus, India. She is currently an Assistant Professor with the Electronics and Communication Engineering Department, School of Engineering, Amrita Vishwa Vidyapeetham, Bengaluru Campus. Her research interests include robotics and control systems.