IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 15 March 2023, accepted 27 March 2023, date of publication 6 April 2023, date of current version 13 April 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3265207

METHODS

2D LiDAR Based Reinforcement Learning for Multi-Target Path Planning in Unknown Environment

NASR ABDALMANAN[1], KAMARULZAMAN KAMARUDIN[1,2], MUHAMMAD AIZAT ABU BAKAR[1], MOHD HAFIZ FAZALUL RAHIMAN[1], AMMAR ZAKARIA[1,2], SYED MUHAMMAD MAMDUH[2], AND LATIFAH MUNIRAH KAMARUDIN[2]

[1]Faculty of Electrical Engineering and Technology, Universiti Malaysia Perlis (UniMAP), Arau 02600, Malaysia
[2]Centre of Excellence for Advanced Sensor Technology (CEASTech), Universiti Malaysia Perlis (UniMAP), Arau 02600, Malaysia

Corresponding author: Kamarulzaman Kamarudin (kamarulzaman@unimap.edu.my)

This work was supported in part by the Fundamental Research Grant Scheme (FRGS) through the Ministry of Higher Education of Malaysia (MOHE) under Grant FRGS/1/2022/TK08/UNIMAP/03/13, and in part by the Malaysian Technical University Network (MTUN) Research Grant by MOHE under Grant 9002-00094/9028-00002.

ABSTRACT Global path planning techniques have been widely employed in solving path planning problems, however they have been found to be unsuitable for unknown environments. Contrarily, the traditional Q-learning method, which is a common reinforcement learning approach for local path planning, is unable to complete the task for multiple targets. To address these limitations, this paper proposes a modified Q-learning method, called Vector Field Histogram based Q-learning (VFH-QL) utilized the VFH information in state space representation and reward function, based on a 2D LiDAR sensor. We compared the performance of our proposed method with the classical Q-learning method (CQL) through training experiments that were conducted in a simulated environment with a size of 400 square pixels, representing a 20-meter square map. The environment contained static obstacles and a single mobile robot. Two experiments were conducted: experiment A involved path planning for a single target, while experiment B involved path planning for multiple targets. The results of experiment A showed that VFH-QL method had 87.06% less training time and 99.98% better obstacle avoidance compared to CQL. In experiment B, VFH-QL method was found to have an average training time that was 95.69% less than that of the CQL method and 83.99% better path quality. The VFH-QL method was then evaluated using a benchmark dataset. The results indicated that the VFH-QL exhibited superior path quality, with efficiency of 94.89% and improvements of 96.91% and 96.69% over CQL and SARSA in the task of path planning for multiple targets in unknown environments.

INDEX TERMS RL, path planning, Q-learning, mobile robot.

I. INTRODUCTION

Mobile robots have been widely utilized in a variety of fields, including industry, military, and healthcare [1]. One of the essential capabilities of these robots is their ability to navigate through an environment, using path planning algorithms to find optimal or near-optimal routes between locations [2]. Path planning can be divided into two categories: global and local. Global path planning is used when the environment is fully known and static, while local path planning is necessary for uncertain, partially known, or changing environments [2]. In the past, researchers have proposed a range of methods for global path planning, such as artificial potential fields [3] and visibility graphs [4], as well as heuristic approaches like A*, D*, and Dijkstra [2]. However, as the tasks assigned to mobile robots become more complex, local path planning methods, including neural networks, fuzzy logic, and reinforcement learning, are becoming increasingly necessary [2], [5].

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

Reinforcement learning is a powerful machine learning paradigm that enables agents to learn how to behave in an environment to achieve a specific goal, based on feedback in the form of rewards, without any prior knowledge of the environment or training data [6]. This makes it particularly well-suited for local path planning tasks. Q-learning is a classic reinforcement learning algorithm that has been widely used for path planning applications [5]. However, previous work has largely focused on using Q-learning in grid map state spaces with fixed target locations, leading to slow convergence in complex environments and a limited ability to handle changes in the target position or the environment without re-training. Several approaches have been proposed to improve the performance of Q-learning, such as using a more sophisticated grid state space or even a sensory state space, but these still rely on training with fixed target positions. An alternative approach is to utilize Deep Q-Networks (DQN), which has the potential to address these limitations but also involves the added computational cost of training deep neural networks [5].

In this work, path planning for multiple targets in unknown environments is investigated. It is often the case that complex mobile robot tasks, such as mapping, item delivery, or surveillance, require path planning to multiple targets within a mission. The hypothesis of this study is that classical Q-learning (CQL) is not capable of solving the problem of path planning for multiple targets due to the convergence of Q-values based on the location of states on the map. As a result, CQL will learn to plan for the first target and will be unable to plan for subsequent targets without de-learning and re-learning the Q-values, that is time-consuming and inefficient. Although many researchers have modified Q-learning to reduce convergence time, there is no research to our knowledge that has addressed the path planning of multiple targets in unknown environments using Q-learning. Further elaboration on the CQL algorithm will be presented subsequently.

The study proposes that by redefining the concept of states and rewards in the CQL algorithm, a modified Q-learning algorithm can be developed that is both faster and capable of learning and solving the problem of path planning for multiple targets in unknown environments. specifically for mobile robots equipped with 2D LiDAR. The effectiveness of this approach, called Vector Field Histogram based Q-learning (VFH-QL), is believed to be superior to the previous classical Q-learning (CQL) path planning algorithm in terms of convergence time and training time. While CQL has a long training time for path planning to a single target and is unable to handle multiple target scenarios, VFH-QL has been shown to successfully solve for both single and multiple targets with a notable improvement in training time.

The remainder of the paper is structured as follows: related work is introduced in Section II, the proposed algorithm is described in Section III, the results of the proposed algorithm are analyzed and compared to those of CQL, SARSA and the heuristic method A* in Section IV, and the paper is concluded in Section V.

II. RELATED WORK

Nature-inspired algorithms, such as genetic algorithms (GAs) and ant colony optimization (ACO), have shown promise in solving complex optimization problems [7]. GAs use principles of Darwinian evolution to produce a population of possible solutions, which are evaluated using a fitness function. The fittest individuals are then selected for reproduction using a crossover operation and mutation is used to ensure diversity in the population [8]. ACO is based on the behavior of ants and the use of pheromone trails to reinforce the shortest and most efficient paths between the starting and ending points [9].

In recent years, several optimization techniques have been proposed for mobile robot path planning in grid-based environments. Laminia et al. [10] developed a fitness function to optimize robot energy by reducing the number of turns in the path. Li et al. [11] combined a genetic algorithm with the dynamic window approach to improve obstacle avoidance, resulting in path length and running time reductions compared to traditional algorithms. Ajeil et al. [12] proposed an aging-based ant colony optimization algorithm that considers the age of ants and demonstrated its superior performance in finding the shortest and most collision-free path. Zhang et al. [13] proposed an adaptive improved ant colony algorithm, AIACSE, that incorporates non-uniform distribution initial pheromone, pheromone diffusion model, adaptive parameter adjusting strategy, and novel pheromone updating mechanism to enhance the optimization ability and efficiency of the ant colony system. Experimental results and statistical tests showed that AIACSE outperforms other algorithms in terms of performance metrics and can adapt to different scale maps.

Genetic algorithms (GAs) and ant colony optimization (ACO) have emerged as promising solutions for path planning problems. However, their effectiveness is hindered when applied to real-time path planning due to certain limitations. Both GAs and ACO are population-based algorithms that can be computationally expensive, particularly for large search spaces, making them unsuitable for real-time applications that require swift decision-making. Furthermore, the population-based approach adopted by these algorithms may lead to suboptimal solutions and slow convergence rates, especially in unknown environments [14].

Q-learning is a widely adopted algorithm in the domain of reinforcement learning that aims to learn the optimal sequence of actions for a given policy. The algorithm is noted for its simplicity, speed, and efficacy in addressing challenging problems in complex and unknown environments. Many previous researchers have employed the Q-learning algorithm in path planning applications. These approaches typically begin by simulating a real-world scenario by discretizing it into a state space. They differ in the design of the desired behavior for a specific task and the representation of the state space. One of the earliest such approaches was by D. Tamilselvi et al. [15], in which CQL was applied to a grid state space simulation environment with a size of 10 by

10 grids. The agent had four action directions (north, west, south, and east) and a Q-table with a total of 400 Q-values. Obstacles and a fixed target were introduced in the simulation, and the results demonstrated that the agent was able to learn how to reach the target. The study also found that there is a positive correlation between the learning rate value and the Q-values, with an inverse correlation between the learning rate and the convergence time.

Given the slow convergence of CQL, Das et al. [16] introduced an Improved Q-learning (IQL) algorithm that is characterized by its faster convergence. Unlike CQL, which computes Q-values from the previous state only, IQL obtains Q-values with the aid of all potential next states [17]. The training took place in a 20 by 20 grid state space with a four-action agent, and the Q-table had a total of 1600 Q-values. The desired behavior focused on minimizing the mobile robot's energy consumption by reducing the number of 90-degree turns in the path. This was supported by successful real-world experiments. Konar et al. [17] also used IQL to optimize the energy consumption of a mobile robot in a 20 by 20 grid state space simulation environment with a four-action agent and a Q-table of 1600 Q-values. The results showed a significant reduction in path planning time compared to the A*, Dijkstra, CQL, and Extended Q-learning (EQL) methods. EQL is a modified Q-learning algorithm that stores only the best action for each state in the Q-table to reduce its size [18], and this was also supported by real-world experiments.

Li et al. [19] implemented IQL with a heuristic search reward function to improve convergence. The training took place in a 20 by 20 grid state space simulation environment with an 8-action agent (north, northwest, west, etc.). The Q-table had a total of 3200 Q-values. The simulation contained both static and dynamic obstacles, and the proposed method performed better than previous CQL, IQL, EQL, and A* approaches in terms of path safety. Babu et al. [20] used CQL for path planning in an environment discretized using image processing. A bird's eye view camera was used to observe the map with obstacles and generate a 9 by 12 grid state space. The agent had 8 actions and the Q-table had 864 Q-values, and the approach was supported by real-world experiments. Peng and Li [21] successfully implemented a simple CQL in a 20 by 20 grid state space simulated environment with a four-action agent and a Q-table of 1600 Q-values.

Qianqian and Li [22] used IQL to achieve energy-saving behavior in a 20 by 20 grid state space simulation environment with an 8-action agent and a Q-table of 3200 Q-values. The results were compared to those of CQL, State-Action-Reward-State-Action (SARSA), and Neural Network Q-learning (NNQL), and the proposed IQL method performed better in terms of energy saving on 45-degree and 90-degree turns. Zhang et al. [23] proposed a Self-Adaptive Reinforcement-Exploration Q-learning (SARE-Q) method in various grid state space environments and compared the results to those of previous CQL and Self-Adaptive Q-learning (SA-Q). Jin et al. [24] and Gao et al. [25] both successfully implemented CQL for path planning in grid state space simulation environments, and both agents were able to learn how to find the desired path. However, their results were not compared to any other path planning methods. Khriji et al. [26] simulated a mobile robot with 15 range sensors divided into 6 directions. The state was represented by a combination of three variables: the distances between the robot and obstacles, the distance between the robot and the target, and the difference between the desired angle and the robot's angle in absolute value. The agent was able to perform path planning successfully in both simulated and real-world experiments.

Dewantara [27] implemented Q-learning for path planning to achieve social interaction behavior, with states defined as a local situation composed of several features such as the robot's heading direction, the relative position of static obstacles to the robot, and the relative position of dynamic obstacles to the robot. The reward function was designed as a combination of several rewards to optimize the robot's behavior. Pham et al. [28] configured states as a combination of discretized sensory inputs for a fixed altitude drone in a 2D grid environment. The agent was able to reach the target state while avoiding obstacles in both simulation and real-world experiments. Chen et al. [29] applied Q-learning for path planning of cargo ship steering behavior in a simulated environment. The results were compared to the heuristic methods RRT and A*, and the Q-learning approach obtained a path that was suitable for heavy cargo ships, rather than simply computing the shortest path that would not be feasible for a cargo ship to navigate in the real world. This work demonstrates that it is possible to achieve custom desired behaviors with Q-learning.

Cardona et al. [30] used a sensory-based state space for a simulated mobile robot with three range sensors on the front of the virtual robot, which were discretized into five levels. As a result, the state space contained 125 states. The agent had three actions (move forward, turn left, turn right) and a Q-table of 375 Q-values. Zhang et al. [31] used a multi-sensor mobile robot with Q-learning for path planning, defining states as a combination of the distance from the robot to the target and the distance from the robot to obstacles, including dynamic obstacles. Li et al. [32] implemented Q-learning in a simulation environment with an agent equipped with 8 sonar sensors, where the state was the discretization of sonar data. Ribeiro et al. [33] used Q-learning for mobile robot path planning by obtaining states with two discrete infrared sensors and an agent with 14 actions (i.e., three different speeds forward, three backward, three right turns, three left turns, two self-rotations CW and CCW). They developed two methods, A and B, with method A alternating actions during episodes and method B maintaining the same action until the Q-table was updated. In general, method B was more efficient. Maoudj and Hentout [34] used image processing to obtain points around obstacles as node states. While this

method was effective in minimizing the state space size, the agent was unable to explore a wider area, as it was restricted to certain paths only.

Previous Q-learning approaches for path planning can be broadly classified into two categories: grid state-space based, and sensory state-space based. Grid state representation leads to relatively large Q-values being processed, resulting in slow convergence in complex environments and high memory usage. While IQL and EQL approaches can address the issue of slow convergence time in Q-tables, they are limited to single fixed targets and require re-training every time the target location changes. On the other hand, sensory state representation approaches may be more effective in handling complex environments without excessively increasing the size of the Q-table, but to the best of our knowledge, they have not been applied to the problem of path planning for multiple or dynamic targets.

III. METHODS

A. CLASSICAL Q-LEARNING ALGORITHM

Q-learning is a type of reinforcement learning algorithm, first introduced by Watkins and Dayan [35]. Reinforcement learning (RL) is a machine learning paradigm that aims to optimize an agent's behavior in an environment through the use of reward signals [36]. The goal of RL is to maximize the cumulative reward received by the agent through its actions, as shown in (Figure 1). Q-learning is a well-known algorithm in the field of RL and has been widely applied to a variety of problems involving decision-making under uncertainty.

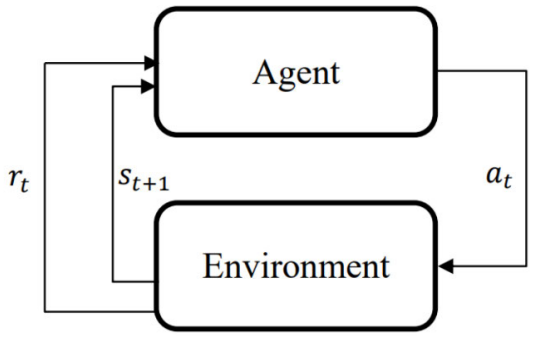

FIGURE 1. Reinforcement learning framework.

In the interaction of an agent within an environment, Q-learning algorithm consist of episodes, steps (T), set of states (S), set of actions (A), set of rewards (R), and a policy (π). Episodes are the iterations where the agent learns through. Steps (T) is the maximum number of total actions the agent can perform in a single episode, where each step (t) represents a time interval. When the agent takes an action in a state s_t it leads the agent to the next state s_{t+1}. States (S) are the state space or the set of all possible positions, locations, or conditions the agent can be in the environment. Actions (A) is the set of all posable actions the agent can perform in a state (s). Rewards (R) is the reward function or the set of all posable rewards the agent may receive after performing an action (a) in a state (s) at a time step (t), known as state-action pair s_t, a_t. The size of state-action pairs is the number of all posable states $s_1, s_2, s_3 \ldots s_n \in S$ multiplied by the number of all posable actions $a_1, a_2, a_3 \ldots a_m \in A$, or $(n \times m)$.

Let an agent be a mobile robot and the environment is a 2D grid-map sized 5 by 5 units (Figure 2), let the mobile robot be able to navigate a single unit at the time towards north, west, south, or east, and let the reward function be as the following.

$$r_{t+1} = f(s_t, a_t) = \left\{ \begin{array}{l} \mathbf{100}, \ \ (s_{t+1}) = \textbf{Target State} \\ \mathbf{-50}, \ \ (s_{t+1}) = \textbf{Obstacle State} \\ \mathbf{-1}, \ \ (s_{t+1}) = \textbf{Free State} \end{array} \right\} \tag{1}$$

where $(100, -50, -1) \in R$

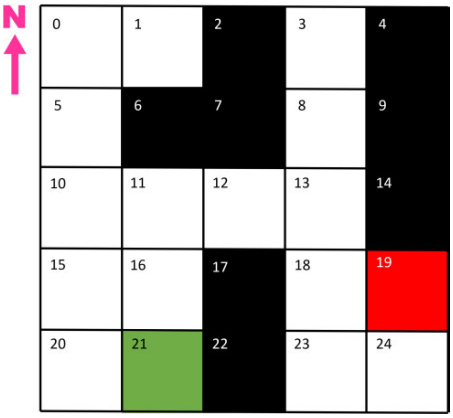

FIGURE 2. (5 × 5) 2D grid-map environment with robot, target, obstacles, and free states as green, red, black, and white respectively.

The robot receives an immediate reward of (100) points if it took any action (a_t) at any state (s_t) that leads to the target state in $(t+1)$, the robot receives an immediate reward of (-50) points if it took any action (a_t) at any state (s_t) that leads to an obstacle state in $(t+1)$, the robot receives an immediate reward of (-1) point if it took any action (a_t) at any state (s_t) that leads to a neither target or obstacle state in $(t+1)$. The reward function is responsible for shaping the behavior of the agent will inherit with time, in this case the reward function is reinforcing a behaviour of obtaining the shortest way to the target while avoiding the obstacles.

The size of the state-space (S) will be 25, this is known as grid state-space, where each grid represents a state the mobile robot land on. For each state (s_t) the mobile robot can perform an action (a) out of four actions (A). therefore, the size of state-action pairs is $(25 \times 4 = 100)$. The state-action pairs are stored in a lookup multi-dimensional table known as Q-table where each state-action pair will hold a value known as Q-value or $Q(s, a)$, the objective of the Q-learning algorithm is to obtain the optimal Q-values $[Q(s, a)]^*$ of all the state-action pairs in a process known as convergence (Figure 3), where the Q-table is completely

filled with optimum Q-values at the end of episodes. The Q-values are updated in the Q-table every step (t) by solving the Bellman equation as follows.

$$Q(s,a) = (1-\alpha) \times Q(s_t, a_t) + \alpha \times [r_{t+1} + \gamma \times \max(Q(s_{t+1}, a_{t+1}))] \quad (2)$$

where the Q-value of an action (a) preformed on a state (s) at time step (t) is the expected reward of that state-action pair combined with the discounted maximum Q-value available in the next state-action pair at ($t+1$). Alpha (α) is the learning rate that controls the rate of Q-values updates in the Q-table. The discount factor gamma (γ) determines the impact of the reward value at the current state (s_t). Eventually, for the robot to autonomously decide the optimal action (a^*) from the set of actions (A), there must be a policy (π) to govern the decision-making. The policy (π) states that the robot should select the action which carries the maximum Q-value among the other actions in a state (s). This approach of action decision is known as "Exploitation". However, at the beginning of the training, the robot might not have any experience of the environment, and the Q-table is empty. Therefore, the exploitation is to be neglected in the early training stages and replaced with random or semi-random action decisions to gain experience. This approach of action decision is known as "Exploration" [36].

The balance between exploration times and exploitation times is very important. If the agent over utilized exploration, it would lead to slow and random convergence of the Q-table. If the exploitation method is overutilized, it will lead to quick and poor convergence of the Q-table. One of the strategies to control the ratio between exploration and exploitation in Q-learning is (ε-greedy) algorithm [36].

$$a_t = f(\varepsilon, P) = \begin{cases} a^*, & \varepsilon < P \\ \text{rand}(A), & \varepsilon \geq P \end{cases} \quad (3)$$

The parameter epsilon (ε) represents an exploration probability. In every step (t) the epsilon value is to be compared with a random probability (P). If epsilon is greater than (P), a random action will be selected (Exploration). And if the epsilon is less than (P), the optimal action will be selected according to the policy (π) (Exploitation). Usually, epsilon is initiated as a value close to 1, at the end of each episode the epsilon value is updated by multiplying it with a decay factor (a positive number between 0 and 1) so the epsilon approaches zero with time. Both the initial epsilon value and decay factor value along with the number of episodes control the rate and transition of exploration to exploitation in the training process, at the end of training in the example of the mobile robot, the Q-table converges to the optimal Q-values for each state-action pair. The Q-values guide the robot toward the target state and prevent it colliding with obstacles from any empty state in the map (Figure 3).

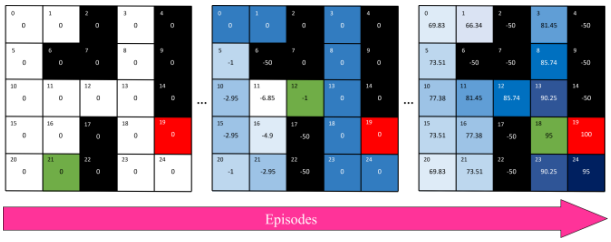

FIGURE 3. Q-table convergence through episodes.

Algorithm 1 Classical Q-Learning (CQL) for Path Planning

Initialization Define: State-space (S), Actions (A), Rewards (R), policy (π) Set values of: No of Episodes (E), No of Steps (T), Learning Rate (α), Discount Factor (γ), Epsilon (ε), Epsilon Decay Factor (ν)
Set Q-table to zeros
For $e = 0$ *to* E **do**
  For $t = 0$ *to* T **do**
    Observe the current state (s_t)
    Exploration vs Exploitation
    $P = rand\ (0, 1)$;
    If $\varepsilon < P$
      $a_t = \arg\max\ [Q(s_t, A)]$;
    Else
      $a_t = \text{rand}\ (A)$;
    Observe the new state (s_{t+1})
    Reward Function
    If (s_{t+1}) *is the target state*
      $r_{t+1} = 100$;
    Else
      if (s_{t+1}) is obstacle state
      $r_{t+1} = -50$;
    Else
      $r_{t+1} = -1$;
    Solve for Q-value
    $Q(s,a) = (1-\alpha) \times Q(s_t, a_t) +$
    $\alpha \times [r_{t+1} + \gamma \times \max(Q(s_{t+1}, a_{t+1}))]$
    Update Q-value to Q-table
    Q-table$[s_t, a_t] = Q(s,a)$;
  End
  Epsilon Decay
  $\varepsilon = \varepsilon \times \nu$;
End

B. SARSA ALGORITHM

SARSA and Q-learning are reinforcement learning algorithms that are commonly employed to determine optimal policies in Markov decision processes. Despite their shared objective, there exist crucial disparities between the two techniques. The fundamental divergence pertains to their respective methods of updating Q-values. Q-learning is categorized as an off-policy algorithm, whereby it enhances its Q-values by considering the optimal action to take in the present state, even if it is inconsistent with the existing policy. Conversely, SARSA is classified as an on-policy algorithm, where it modifies its Q-values based on the policy being pursued in the current state [37]. Thus, the modified Bellman equation

for SARSA can be articulated as follows.

$$Q(s,a) = (1-\alpha) \times Q(s_t, a_t) + \alpha \times [r_{t+1} + \gamma \times Q(s_{t+1}, a_{t+1})] \quad (4)$$

The SARSA algorithm is outlined as follows:

Algorithm 2 SARSA for Path Planning

Initialization
Define: State-space (S), Actions (A), Rewards (R), policy (π) Set values of: No of Episodes (E), No of Steps (T), Learning Rate (α), Discount Factor (γ), Epsilon (ε), Epsilon Decay Factor (ν)
Set Q-table to zeros
For $e = 0$ *to* E **do**
  For $t = 0$ *to* T **do**
    Observe the current state (s_t)
    Exploration vs Exploitation 1
    $P = rand\ (0, 1)$;
    If $\varepsilon < P$
      $a_t = \arg\max\ [Q(s_t, A)]$;
    Else
      $a_t = \text{rand}\ (A)$;
    Observe the new state (s_{t+1})
    Reward Function
    If (s_{t+1}) *is the target state*
      $r_{t+1} = 100$;
    Else
      if (s_{t+1}) is obstacle state
      $r_{t+1} = -50$;
    Else
      $r_{t+1} = -1$;
    Exploration vs Exploitation 2
    $P = rand\ (0, 1)$;
    If $\varepsilon < P$
      $a_{t+1} = \arg\max\ [Q(s_{t+1}, A)]$;
    Else
      $a_{t+1} = \text{rand}\ (A)$;
    Solve for Q-value
    $Q(s,a) = (1-\alpha) \times Q(s_t, a_t) + \alpha \times [r_{t+1} + \gamma \times Q(s_{t+1}, a_{t+1})]$
    Update Q-value to Q-table
    Q-table$[s_t, a_t] = Q(s,a)$;
  End
  Epsilon Decay
  $\varepsilon = \varepsilon \times \nu$;
End

In this paper a modified Q-learning algorithm have been proposed to perform the path planning of multiple targets in unknown environments for mobile robot equipped with a 2D LiDAR module, the proposed algorithm was mainly based on the processed 2D LiDAR data, as it considered sensory state-space. The proposed Q-learning algorithm was trained and tested in simulation environment, where it consists of a virtual omnidirectional mobile robot equipped with a 2D LiDAR scanner to allow map exploration and obstacle detection. The simulation environment was created in Python with the aid of OpenCV package. The process of creating the environment, designing the Q-learning algorithm, and its implementation is introduced in the next subsections.

C. SIMULATION ENVIRONMENT

The environment is a 2D map represented as an image of 400 by 400 pixels, where each pixel is equivalent to 5 centimeters in real-world dimensions. There are two maps, the reference map, and the robot map. The reference map is the fully known map used for training, which is a black and white image with a size of 400 by 400 pixels, where black and white areas represent obstacles and free spaces respectively. The mobile robot has no comprehension of the reference map; therefore, it has to create its own robot map. The robot map is a colored image size of 400 by 400 pixels, where the colors black, grey, and white represents obstacles, unknown, and free-space areas respectively, while the virtual mobile robot and targets are represented with green and red colors respectively.

The virtual mobile robot is assumed to be equipped with a 2D LiDAR to collect information about the unknown environment. A method of Lidar emulating was done in order to introduce this feature in the simulated environment. In the beginning, the robot map is initialized with all pixels in grey color which indicates that the map is an unknown area to the robot. Other initialization variables like range ($\boldsymbol{K}$), field of view angle ($\boldsymbol{\Theta_F}$), and segment angle ($\boldsymbol{\phi}$) are shown in (Figure 4).

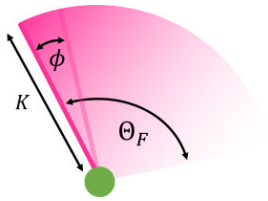

FIGURE 4. LiDAR variables for simulation.

The range ($\boldsymbol{K}$) determines the maximum reading distance of a laser ray at any angle. In this work the maximum range is 5 meters or 100 pixels in the simulated environment. The field of view ($\boldsymbol{\Theta_F}$) determines the angular scanning range with value of 360° degrees. The segment angle ($\boldsymbol{\phi}$) determines the angular discretization of the field of view and was set to 1° degree. In other words, there will be a LiDAR reading each degree of the 360° degrees, outputting 360 readings in total. The LiDAR simulator starts from the virtual robot as the origin point with nested loops A and B. loop A is the field of view loop from 0 to 359 where each step in the loop represents a segment. In loop A there is another loop B from 0 to 99 represents the range of 100 pixels. For each iteration in loop B, the algorithm grabs the x, and y coordinates of the next pixel in the direction of the angle obtained from loop A with respect to the robot location. The pixel coordinates are compared with the identical coordinates in the reference map image. If it is an obstacle (black pixel), loop B stops. If it is

Algorithm 3 2D LiDAR Scanner Simulator

Initialization
Set values of: Range (K), Field of View Angle (Θ_F), Segment Angle (ϕ), Robot Position (x_R, y_R)
Input: Reference Map (RFM), Robot Map (RBM)
Loop (A)
For $\theta_F = [0\ to\ \Theta_F] \times \phi$ **do**
  Loop (B)
  For $k = 0\ to\ K$ **do**
    Observe the current pixel $(x_{\theta k}, y_{\theta k})$
    $x_{\theta k} = x_R + int[k \times \cos(\theta_F)]$;
    $y_{\theta k} = y_R + int[k \times \sin(\theta_F)]$;
    Update the robot map
    If RFM $[x_{\theta k}, y_{\theta k}] = [0]$
      RBM $[x_{\theta k}, y_{\theta k}] = [0, 0, 0]$;
      Break
    Else
      RBM $[x_{\theta k}, y_{\theta k}] = [255, 255, 255]$;
  End
End

a free space (white pixel), loop B carries on to the next pixel in the same direction until reaching the maximum range or finds an obstacle. At the end of loop, A, a complete scan is performed, and the scanned information is passed to the robot map.

In the real-world, the mobile robot is omnidirectional. Therefore, to discretize the robot movement in the simulated environment, the virtual mobile robot is assumed to be able to move in eight directions (North, north-west, west, south-west, south, south-east, east, and north-east). For each direction, the virtual mobile robot is assumed to travel 9 pixels i.e., approximately (0.5) meters in the real world.

To further discreet the environment, the concept of tiled map is used (Figure 5), where the robot map is segmented into matrixed tiles. Each tile is set to 9 by 9 pixels sized. When the robot performs a movement step, it travels from the center of one tile to the center of one of the available 8 neighbor tiles. The tiles are generated after each laser scanning operation, where the first tile is at the initial virtual robot position. To generate the next surroundings tiles, the scanned area of the tile must be completely obstacle-free and known, and any pixel within the future generated tiles are considered unknown (gray) or occupied with obstacle (black). The method will not generate a tile in that location

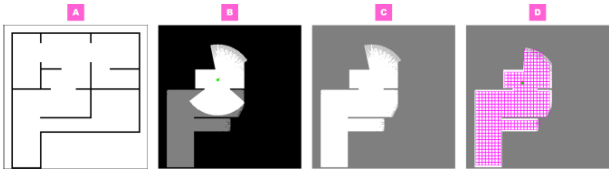

FIGURE 5. Example of reference map, LiDAR scanner map, robot map and tiles map as A, B, C and D respectively.

to prevent any physical collision of the virtual mobile robot with obstacles.

D. PROPOSED VFH-QL ALGORITHM

1) STATE CONFIGURATION

State is defined as a set of three elements based on processed lidar data. The first element is the determination of whether there is an open way or an obstacle around the robot. To visualize the approach, consider a mobile robot equipped with a 2D LiDAR performing laser scanning on a map. The readings of the LiDAR could be plotted as a two-dimensional graph of field of view ($\boldsymbol{\Theta_F}$) versus range ($\boldsymbol{K}$). For each angle ($\boldsymbol{\theta_F}$), the laser ray either bounces back from an obstacle at distance ($\boldsymbol{k}$) or reach maximum range $\boldsymbol{k} = \boldsymbol{K} = 100$ pixels (Figure 6).

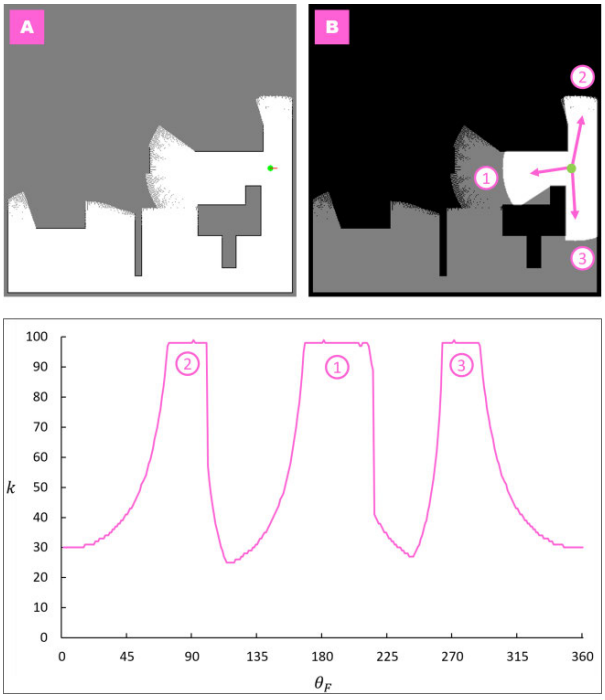

FIGURE 6. Robot map and LiDAR scanner map in A and B respectively, with LiDAR data graph.

In the graph, each mountain represents an ''Open way'' on the map with respect to the angle, ($\boldsymbol{\theta_F}$), whereas each valley represents an obstacle in that direction. A single dimension convolutional average filter is applied to smooth the data, eliminating the tiny noise mountains, and emphasizing the bigger ones according to the size of the sliding average window. In this work, a window of size 1 by 11 was used. The next step is to find the peaks of mountains, as it will determine the ($\boldsymbol{\theta_F}$) directions of the open ways on the map. The peaks finder algorithm is shown as follows.

The angles taken from the peak finder are classified into the eight angles of the robot movement directions. The output is an array of eight binary values representing direction of 0°, 45°, 90°, 135°, 180°, 225°, 270°, 315° where obstacle or open way are represented as zero or one respectively.

Algorithm 4 Peaks Finder

Initialization
Input: LiDAR Data
Set: New Array as (X)
$c = 0$;
For $\theta_F = [0\ to\ \Theta_F] \times \phi$ **do**
  If $k\,[\theta_F] \geq k\,[\theta_F - 1]$ ***and*** $k[\theta_F] \geq k[\theta_F + 1]$
  $X\,[c] = \theta_F$;
  $c+ = 1$;
End
Return X;

The second element is the determination of the target angle with respect to the robot position. The obtained angle is classified into one of the eight directions angles (i.e., 0°, 45°, 90°, 135°, 180°, 225°, 270°, 315°).

The third element is the determination of whether the target is visible to the robot or not, by calculating if the target point is inside the laser-scanned area. Let $(\boldsymbol{d_T})$ be the chessboard distance between the target and the robot and let $(\boldsymbol{\theta_T})$ be the angle of the target with respect to the robot direction. By referring to the LiDAR data, let $(\boldsymbol{k_T})$ be the lidar reading of the ray emitted at an angle $(\boldsymbol{\theta_T})$. If $\boldsymbol{k_T}$ is greater than $\boldsymbol{d_T}$, then the target point is inside the laser-scanned area, and if d_T is greater than $\boldsymbol{k_T}$, then the target point is located outside the laser-scanned area. The result is a binary value where 0 represents a situation of an invisible target, and 1 represents a visible target.

Finally, the three elements are combined in an array of 10 values to define a state, where the first 8 values describe the open ways and obstacles in 8 directions. Followed by the target's angle and target's visibility in the nineth and tenth values respectively. The theoretical size of the state-space is $(2^9 \times 8 = 4096)$ possible states.

2) REWARD FUNCTION

The reward function was designed to obtain a behavior of moving towards the target while avoiding the obstacles, where it is divided into positive rewards, major punishments, and minor punishments. The agent receives an immediate reward (250) when it reaches the target location. To encourage obstacles avoidance, the agent receives a major punishment (−50) in case of obstacle collision attempt. The agent receives a minor punishment (−1) when it chooses the action direction that fit two criteria:

1) the action direction must be one of the "Open way" directions in the state configuration and
2) the action direction must have the lowest action to target angle difference $(\boldsymbol{\Delta\theta_{aT}})$, where $\boldsymbol{\Delta\theta_{aT}} = \boldsymbol{\min}[(\boldsymbol{\theta_T} - \boldsymbol{a}), (\boldsymbol{a} - \boldsymbol{\theta_T})]$.

This minor punishment was made to encourage the agent to reach the target in the shortest time. If the agent chooses an action that failed criteria 2 and fitted criteria 1, it receives a minor punishment $(-2 \geq r \geq -5)$ depending on the action to target angle difference $\boldsymbol{\Delta\theta_{aT}}$ value. This minor punishment was made to encourage the prevention of using other open ways than the closest to the target.

IV. EXPERIMENTAL SETUP AND RESULTS

The proposed Q-learning path planning algorithm was developed to achieve faster convergence and better performance than the classical Q-learning algorithm for path planning (CQL). VFH-QL and CQL methods were trained in computer simulation via two experiments. Experiment 1: path planning for a single and fixed target on the map, where the virtual mobile robot is initiated at the start point and the goal is to obtain the optimal path to a fixed single target point (Figure 7). Experiment 2: path planning for multiple targets on the map, where the virtual mobile robot initiated at the start point with (10) target points scattered on the map. The goal is to obtain the optimal path to the locked target. Each time the robot reaches the locked target, it will be the new starting point for the next target to be locked in a defined order (Figure 7). The performance results of the proposed Q-learning path planning algorithm were qualitatively and quantitatively compared to the CQL results with identical Q-learning parameters, in the same unknown environment and conditions for both experiments. The obtained path optimality levels were estimated via comparison with the heuristic global path planning method A* and smoothed with Spline algorithm.

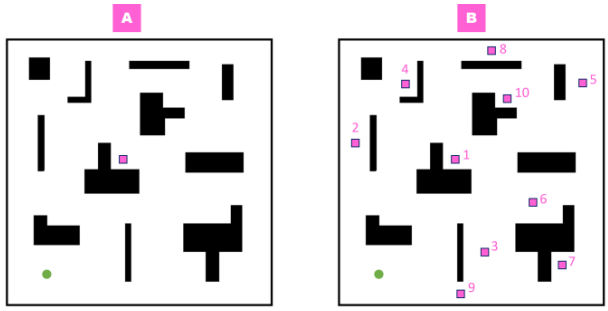

FIGURE 7. Training maps for A) experiments 1 and B) experiment 2, where the robot start point, targets, obstacles, and free space are represented as green, magenta, black, and white.

A. EXPERIMENT 1: PATH PLANNING FOR SINGLE FIXED TARGET

In this experiment, both VFH-QL and CQL methods were trained for 60,000 episodes to find the optimal path from the start point to the fixed target point on the map. The proposed method showed a faster convergence where it managed to find the solution faster than CQL (Figure 8). The "solution" stands for achieving the target in the shortest path within one episode such that the reward reaches the global maximum value. For example, in an episode, if the target is three steps far from a starting point, where each step costs (−1) reward and the target return (250) reward, the optimal path will cost $[(-1 \times 3) + 250 = 247]$ as a global maximum.

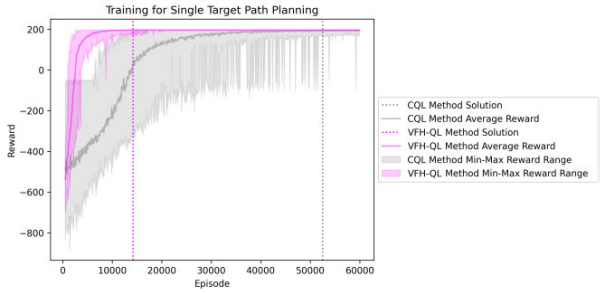

FIGURE 8. Learning curves of the proposed method compared to CQL after training for single target path planning.

Due to the differences in state-space configuration between the proposed method and CQL, it allows the proposed method to minimize unnecessary exploration, where if the virtual mobile robot learns from an obstacle in some locations, it no longer needs to learn again the next time it faces a similar obstacle in another location. Unlike CQL, it considers every locational state on the map as a unique state to learn even if it was similar in condition to a registered state in a previous location. Therefore, the episode reward range (i.e., minimum to maximum reward in the episode) of CQL is wider than the proposed method, which indicates the large number of Q-values to be processed.

The results of the final paths are shown in (Figure 9), where for CQL, the obtained path was identical to the path generated by A* algorithm if the map is assumed to be known. This path is the shortest possible path from the starting point to the fixed target. However, it is dangerous for mobile robots to maneuver close to obstacles in an unknown environment. VFH-QL method sacrificed some of the path's length for safety (better obstacle avoidance) due to the effect of the proposed reward function on the path planning behavior. In (Table 1), the proposed method showed a 52.7% improvement in the overall training time for 60,000 episodes compared to CQL and 72.95% less episodes required than CQL for the training process. Therefore, the proposed method required 87.06% less training time than CQL to find the optimal path in experiment A. The proposed method was 99.98% better at obstacles avoidance than CQL method with 75.52% less exploration (step-timeout).

B. EXPERIMENT 2: PATH PLANNING FOR MULTIPLE TARGETS

In this experiment, both VFH-QL and CQL methods were trained for 60,000 episodes to obtain the optimal journey path from the start point to a dynamic target point on the map. Each time the virtual mobile robot reaches the target, the environment will relocate the target to the next pre-determined location (refer Figure 7) which is the next target for the remaining episodes. The CQL method managed to find the first target after (17454) training episodes. However, when the target position was relocated in the environment, the robot was forced to navigate to the first target location due to the convergence being strongly attached to the map, leading to

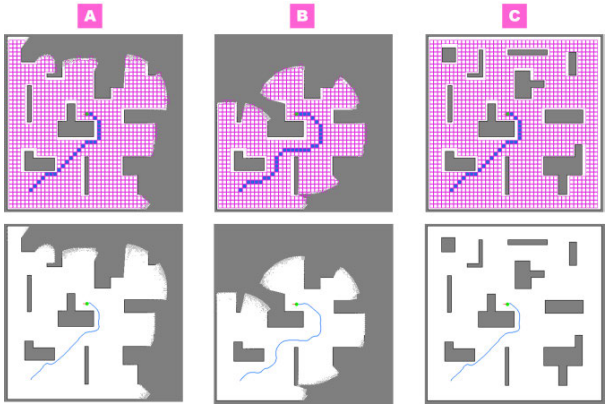

FIGURE 9. Experiment 1 final path results in tiles map and after path smoothing for A) CQL method, B) proposed method, and C) A* method.

TABLE 1. Training results of single fixed target path planning.

	CQL Method		VFH-QL Method		A*
Training Time	3.89 Hours		1.84 Hours		-
Solution Episode	After (52500) episodes/(3.4) hours		After (14200) episodes/(0.44) hours		-
Target Hit Times	**Before finding solution**	**After finding solution**	**Before finding solution**	**After finding solution**	1
	39434	7498	12003	45800	
Obstacle Hit Times	44422	2	11	0	0
Step-timeout Times	8525	0	2087	0	-
Final Path Length	23 Tiles		31 Tiles		23 Tiles

reward loss. CQL will need more time to unlearn the previous target through punishments until it finds the locked target, and this process of unlearning and relearning will occur again each time the virtual mobile robot reaches the next target in a sawtooth learning curve pattern (Figure 10). On the other hand, the VFH-QL method showed a faster convergence than the CQL and managed to successfully learn to achieve all targets throughout the episodes. The time taken for the subsequent targets is decaying and the average reward is increasing, showing optimization (Figure 10). The VFH-QL and CQL final path planning results are shown in (Figures 11, 12), and in (Table 2). The proposed method found the path to the first target in 56.92% less training time and 63.78% fewer episodes than CQL. For the next 9 targets, the proposed method showed a 24.61% average time decrease between targets in the training process, while CQL failed to solve for multiple targets. Therefore, the average performance of the proposed method was better than CQL by $[(56.92 + 900) \div 10] = 95.69\%$ in training time and by 83.99% in

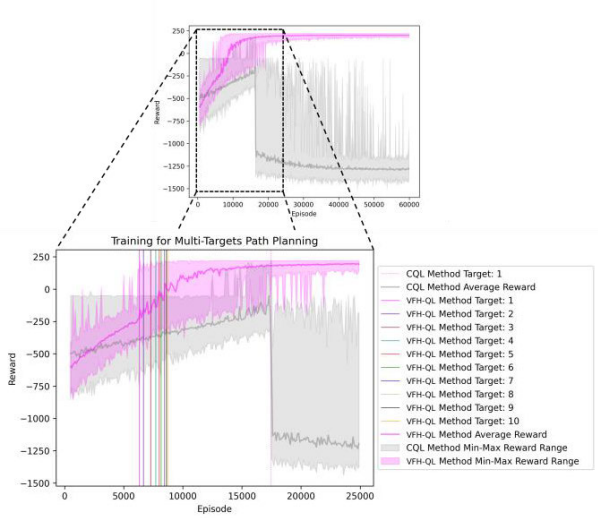

FIGURE 10. Learning curves of the proposed method compared to CQL after training for multiple targets path planning.

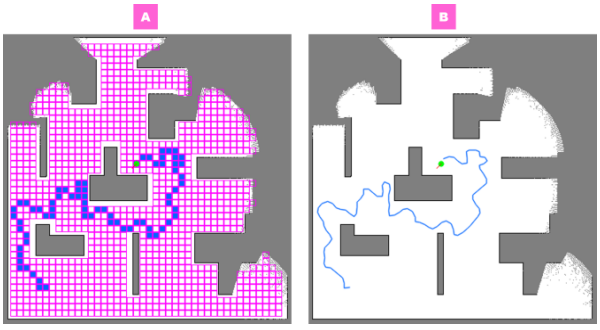

FIGURE 11. Experiment 2 final path results of CQL in A) tiles map and B) after path smoothing for the first target.

path optimality, where the average path length error (Δ Path Length) between CQL and A* was 30.6 tiles compared to 4.9 tiles for the VFH-QL method.

C. EVALUATION OF VFH-QL PERFORMANCE

To evaluate the stability and efficiency of the VFH-QL algorithm in multiple targets path planning, we conducted a series of experiments on three different grid maps of varying complexity. The "Artificial Benchmarks" proposed by N. Sturtevant [38] were chosen as the grid maps for this study. The first map used was "empty-32-32" map, which was a simple grid map with no obstacles, comprising 32 by 32 states with uncrossable outer borders. The second map was "random-32-32-10" map, which was a grid map with size 32 by 32 with randomly placed obstacles, creating 922 free states with uncrossable outer borders. The third map used was "random-32-32-20" map, which was a grid map of the same size as the previous one but with relatively denser random obstacles, creating a total of 819 free states. These maps are shown in (Figure 13).

The benchmark set provided a variety of scenarios for each map, each containing multiple problems. These problems involved randomly selecting two points on the map and finding the optimal path between them. The optimal path length was calculated using diagonals with a cost of $\sqrt{2}$ and cardinal movements with a cost of 1. The problems were categorized into larger "buckets", with the bucket value for a path of length (l) being $\lfloor \frac{l}{4} \rfloor$. Thus, short path problems had a lower bucket value compared to longer path problems. For each set of experiments, 10 problems were randomly selected to be solved as a multi-target case. These problems had bucket values ranging from 0 to 9, testing different path lengths. The robot was tasked with finding the optimal path for all 10 problems. If the robot completed all 10 problems before the maximum number of episodes was reached, the problems were repeated until all the episodes were used. We compared the performance of VFH-QL with CQL and SARSA algorithms, in terms of path length and the number of episodes required to find the shortest path.

TABLE 2. Training results of multiple targets path planning.

	CQL Method			A*	
Training Time	6.64 Hours			-	
	Episodes	**Time (H)**	**Path Length (Tiles)**	**Path Length (Tiles)**	**Δ Path Length (Tiles)**
Target: 1	17454	1.3	71	23	48
Target: 2	UNABLE TO CONVERGE WITHIN 60,000 EPISODES			19	19
Target: 3				27	27
Target: 4				29	29
Target: 5				32	32
Target: 6				22	22
Target: 7				17	17
Target: 8				38	38
Target: 9				42	42
Target: 10				32	32
	VFH-QL Method			**A***	
Training Time	2.83 Hours			-	
	Episodes	**Time (H)**	**Path Length (Tiles)**	**Path Length (Tiles)**	**Δ Path Length (Tiles)**
Target: 1	6322	0.56	33	23	10
Target: 2	336	0.05	23	19	4
Target: 3	609	0.03	38	27	11
Target: 4	444	0.02	35	29	6
Target: 5	278	0.01	34	32	2
Target: 6	163	0.02	28	22	6
Target: 7	277	0.01	18	17	1
Target: 8	11	0.008	43	38	5
Target: 9	162	0.005	46	42	4
Target: 10	99	0.005	32	32	0

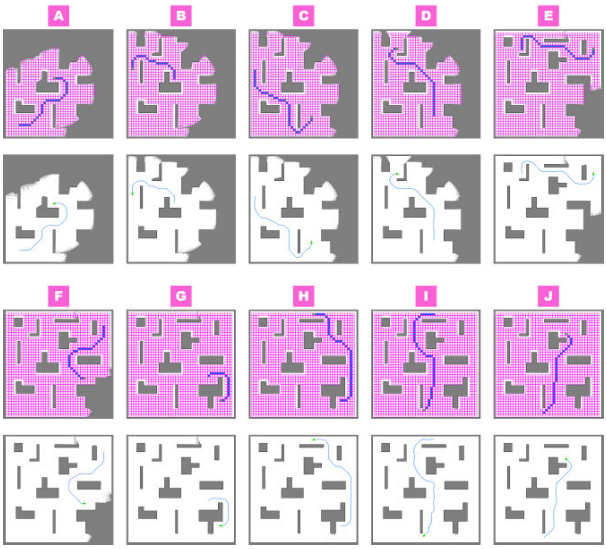

FIGURE 12. Experiment 2 final path results of proposed method in tiles maps and after path smoothing for targets 1 to 10 represented in (A) to (J) respectively.

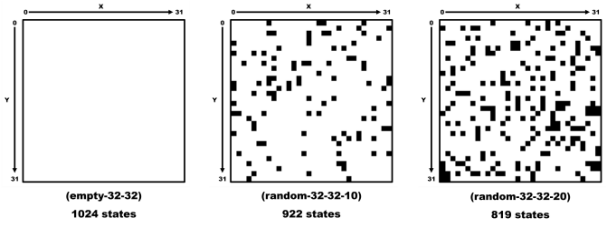

FIGURE 13. Benchmark 2D grid maps sized 32 by 32 grids with different levels of complexity. The obstacles and free space are represented as black and white respectively.

The training parameters were consistent for all three algorithms and across all three experiment sets. The algorithms were trained for 500 episodes, with each episode having a maximum of 100 action steps. The maximum epsilon value was set to 0.9, and the minimum epsilon was set to 0.05, with an epsilon decay value of 0.99 to ensure a smooth decay from exploration to exploitation over the 500 episodes. The learning rate was set to 0.1 to ensure long-term learning with discount factor of 0.95.

In analyzing the performance of the (empty-32-32) map (Figure 14), it is observed that CQL exhibits the lowest average reward rate. Furthermore, the learning curve pattern demonstrates that the algorithm is not learning and, in fact, is de-learning. In (Figure 15) CQL only achieved the first 4 targets, with the first target attained perfectly in the first episode. It is highly probable that CQL stumbled upon the optimal path by chance, given that the epsilon value was 0.9, signifying that 90% of the robot's actions were random. In the second and third paths, CQL finds the second target but with poor path quality. When attempting to find the third path, the robot exhibits a tendency to move toward the previous target location instead of the current target (refer to red arrow with label 1). This behavior results in a loss of reward during training until the Q-table corrects the previous experience, and the

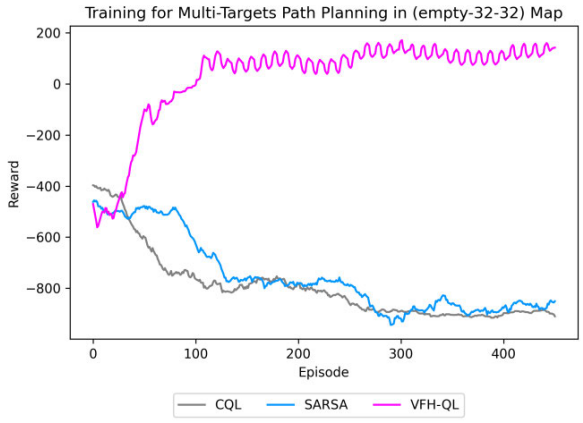

FIGURE 14. Learning curves of the proposed method compared to CQL and SARSA after training for multiple targets path planning in map "empty-32-32".

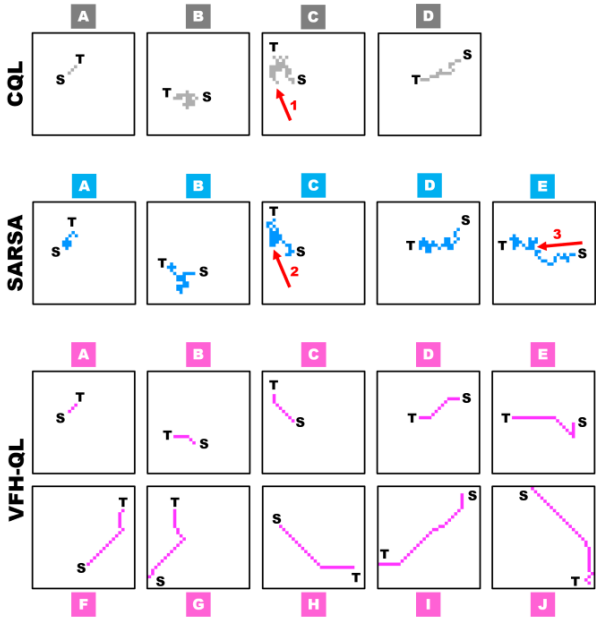

FIGURE 15. Best path results of CQL, SARSA and the VFH-QL method in "empty-32-32"' map, targets 1 to 10 represented in (A) to (J) respectively. The start and target points labeled as (S) and (T). Further details and coordinates in (Table 3).

robot learns to navigate to the correct target. Consequently, the overall path length is poor.

Similar behavior is exhibited by SARSA, as demonstrated by its learning curve pattern (Figure 14), which is analogous to that of CQL. In (Figure 15), the path of the third target illustrates that the robot was pulled towards the second target by the Q-table, and likewise, the robot was drawn towards the fourth target during the search for the fifth target (refer to red arrows with labels 2 and 3).

It is worth noting that CQL and SARSA failed to find the fifth and sixth targets respectively, and remained stuck in the process until all episodes were completed.

Upon examining the learning curves of (empty-32-32) map presented in (Figure 14), it is evident that VFH-QL exhibits a learning behavior, as the average reward steadily

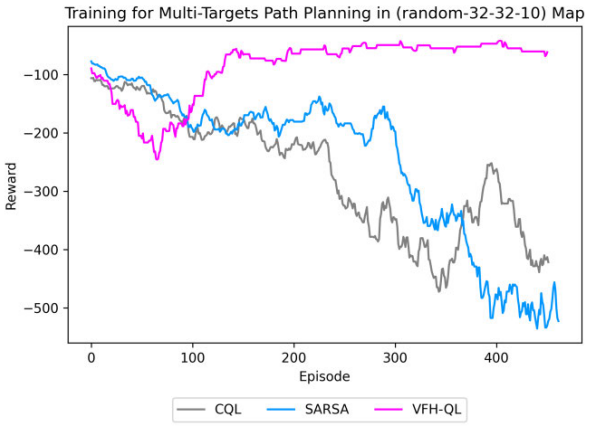

FIGURE 16. Learning curves of the proposed method compared to CQL and SARSA after training for multiple targets path planning in map "random-32-32-10".

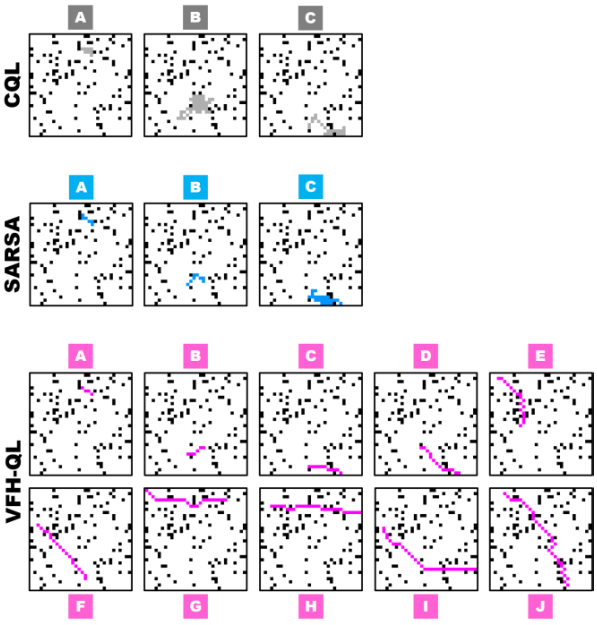

FIGURE 17. Best path results of CQL, SARSA and the VFH-QL method in "random-32-32-10" map, targets 1 to 10 represented in (A) to (J) respectively. Further details and coordinates in (Table 3).

increases throughout the episodes. Moreover, VFH-QL displays a repentant wavy pattern in the learning curve, particularly after approximately 100 episodes, indicating that the algorithm successfully completed the task of identifying all ten target paths and continues to repeat the task while optimizing the path lengths, leading to a subsequent increase in average reward. (Figure 15) showcases the 10 best paths obtained by VFH-QL during training, with (Table 3) detailing their respective path lengths and the episodes at which they were acquired. The results illustrate the efficacy of the VFH-QL algorithm in accomplishing the planning task in an unknown environment, with the path lengths being the shortest or nearly the shortest in most cases.

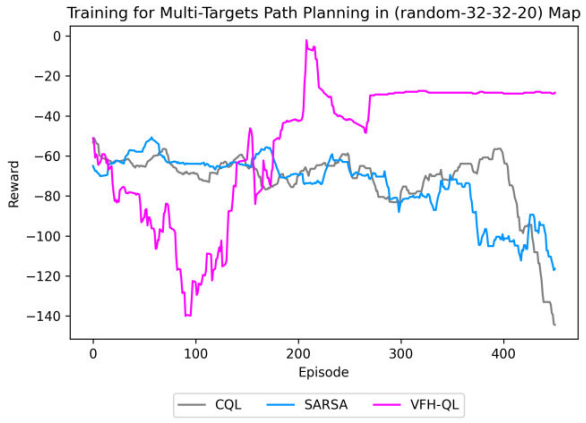

FIGURE 18. Learning curves of the proposed method compared to CQL and SARSA after training for multiple targets path planning in map "random-32-32-20".

In the conducted experiment that aimed to train for multiple target path planning within (random-32-32-10) map, (Figure 16) illustrates that both the CQL and SARSA methods exhibited a decreasing trend in their average reward values across episodes, thereby indicating that they failed to acquire the ability to learn for multiple targets path planning. In contrast, the VFH-QL method exhibited an increasing trend in its average reward values, indicating that it was successful in learning the task.

In (Figure 17), it is observed that the CQL and SARSA methods successfully generated paths for the first 3 targets only, albeit with significant path length errors, and were unable to obtain paths for the remaining 7 targets. Conversely, the VFH-QL method was able to generate paths for all ten targets with an average path length error of less than 5%. Further details on the paths generated by the three methods are presented in (Table 3).

In the context of the experiment aimed at training for multiple targets path planning in (random-32-32-20) map, (Figure 18) illustrates that the average rewards for both CQL and SARSA decrease with each episode. This trend suggests that these methods have not been successful in learning the task of multiple target path planning. On the other hand, the VFH-QL method exhibits an increasing trend in its average reward, indicating its proficiency in learning the task. The slightly rougher appearance of VFH-QL's learning curve can be attributed to the increased complexity of the map with a greater number of obstacles as compared to the previous two maps.

As illustrated in (Figure 19), both CQL and SARSA were successful in obtaining paths for only the first two targets, with path length errors, resulting in their failure to acquire the remaining eight targets.

Conversely, VFH-QL was able to successfully obtain all ten paths, with an average path length error of less than 5%. Detailed information pertaining to the paths acquired by all three methods can be found in (Table 3).

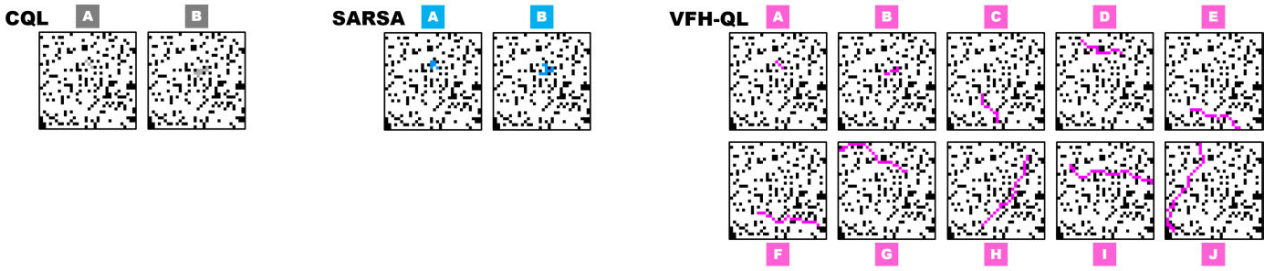

FIGURE 19. Best path results of CQL, SARSA and the VFH-QL method in "random-32-32-20" map, targets 1 to 10 represented in (A) to (J) respectively. Further details and coordinates in (Table 3).

TABLE 3. Training results of multiple targets path planning in benchmark maps.

Map	Start (x, y)	Target (x, y)	Best path episode			Best path length			Optimal path length	Error		
			CQL	SARSA	VFH-QL	CQL	SARSA	VFH-QL		CQL	SARSA	VFH-QL
empty-32-32	(11, 12)	(13, 10)	1	4	343	2.83	19.90	2.83	2.83	0.00	17.07	0.00
	(14, 22)	(8, 20)	34	13	122	40.04	30.04	6.83	6.83	33.21	23.21	0.00
	(9, 15)	(3, 7)	64	50	210	57.11	53.36	10.49	10.49	46.62	42.87	0.00
	(25, 8)	(13, 14)	76	127	292	23.31	54.77	14.49	14.49	8.82	40.28	0.00
	(25, 16)	(6, 14)	N/A	289	260	0	57.77	25.49	19.83	19.83	37.94	5.66
	(17, 24)	(28, 7)	N/A	N/A	215	0	0	22.38	21.56	21.56	21.56	0.82
	(0, 28)	(8, 7)	N/A	N/A	437	0	0	27.80	24.31	24.31	24.31	3.49
	(5, 12)	(28, 25)	N/A	N/A	274	0	0	28.38	28.38	28.38	28.38	0.00
	(26, 2)	(0, 24)	N/A	N/A	385	0	0	37.46	35.11	35.11	35.11	2.35
	(12, 0)	(30, 30)	N/A	N/A	462	0	0	38.28	37.46	37.46	37.46	0.82
						Average percentage error				159.15%	232.10%	5.56%
random-32-32-10	(19, 6)	(16, 4)	16	14	116	12.66	10.07	3.83	3.83	8.83	6.24	0.00
	(18, 23)	(13, 25)	49	76	119	84.36	8.66	5.83	5.83	78.53	2.83	0.00
	(25, 31)	(15, 29)	247	325	164	35.80	51.46	11.83	10.83	24.97	40.63	1.00
	(14, 23)	(26, 31)	N/A	N/A	316	0	0	15.90	15.31	15.31	15.31	0.59
	(9, 16)	(2, 1)	N/A	N/A	279	0	0	22.56	19.07	19.07	19.07	3.49
	(2, 11)	(17, 28)	N/A	N/A	127	0	0	23.80	23.80	23.80	23.80	0.00
	(0, 0)	(25, 3)	N/A	N/A	246	0	0	27.49	26.83	26.83	26.83	0.66
	(31, 7)	(3, 5)	N/A	N/A	456	0	0	29.66	28.83	28.83	28.83	0.83
	(2, 12)	(31, 25)	N/A	N/A	296	0	0	36.38	34.97	34.97	34.97	1.41
	(24, 30)	(4, 1)	N/A	N/A	229	0	0	42.53	39.04	39.04	39.04	3.49
						Average percentage error				250.81%	121.85%	4.97%
random-32-32-20	(15, 9)	(17, 11)	5	64	121	2.83	7.41	2.83	2.83	0.00	4.58	0.00
	(15, 13)	(19, 11)	163	204	239	8.83	14.83	5.41	5.41	3.42	9.42	0.00
	(16, 29)	(11, 20)	N/A	N/A	392	0	0	12.66	11.66	11.66	11.66	1.00
	(8, 2)	(21, 6)	N/A	N/A	255	0	0	15.83	15.83	15.83	15.83	0.00
	(8, 25)	(24, 31)	N/A	N/A	386	0	0	19.90	19.07	19.07	19.07	0.83
	(29, 27)	(9, 23)	N/A	N/A	432	0	0	24.31	21.66	21.66	21.66	2.65
	(22, 9)	(1, 1)	N/A	N/A	215	0	0	27.73	24.90	24.90	24.90	2.83
	(11, 27)	(26, 4)	N/A	N/A	444	0	0	30.38	30.38	30.38	30.38	0.00
	(4, 7)	(31, 13)	N/A	N/A	262	0	0	33.73	32.07	32.07	32.07	1.66
	(2, 29)	(11, 0)	N/A	N/A	453	0	0	38.38	36.14	36.14	36.14	2.24
						Average percentage error				86.32%	109.20%	4.79%

The results of the experiment, as illustrated in (Figures 14 to 19) and (Table 3), demonstrate that the efficiency of the VFH-QL approach in multiple targets path planning remains consistently superior across the three maps tested. The efficiency of each method can be determined by computing the average percentage error, which is derived from the difference between the obtained path length and the optimal path length, divided by the optimal path length and then multiplied by 100. Alternatively, the efficiency of each method can be expressed as a negative value, with 0 representing perfect efficiency. The CQL method demonstrated an average efficacy of −159.15%, −250.81%, and −86.32% for the (empty-32-32), (random-32-32-10), and (random-32-32-20) maps, respectively, resulting in an average overall efficiency of 165.43% less efficient than the optimal benchmark results. SARSA demonstrated efficiencies of −232.10%, −121.85%, and −109.20%, resulting in an average overall efficiency of 154.38% less efficient than optimal benchmarks. In contrast,

the VFH-QL approach demonstrated efficiencies of −5.56%, −4.97%, and −4.79%, resulting in an average overall efficiency of only 5.11% less efficient than optimal benchmarks. These results indicate that the VFH-QL approach achieved a 96.91% improvement over CQL and a 96.69% improvement over the SARSA algorithm in path planning for multiple targets.

V. CONCLUSION

In summary, this research has presented a modified Q-learning method called Vector Field Histogram based Q-learning (VFH-QL) that addresses the limitations of classical Q-learning algorithm (CQL) in unknown and dynamic environments with multiple targets. The VFH-QL algorithm utilizes a 2D LiDAR sensor for sensory state-space representation and reward function, which results in a more efficient and effective path planning solution. The performance of the proposed method was evaluated through simulation experiments, where it was demonstrated that the VFH-QL algorithm achieved significant improvements in terms of training time and path quality compared to CQL and SARSA methods, with improvements of 96.91% and 96.69% respectively. Furthermore, the proposed method was able to handle multiple targets in unknown environments with an efficiency of 94.89%, making it a suitable solution for mobile robots equipped with 2D LiDAR sensors. Overall, this research has made a valuable contribution to the field of path planning, providing a new and efficient solution for multiple targets in unknown and dynamic environments, which will have potential applications in various fields such as robotics, autonomous vehicles, and drones, particularly for map exploration and mapping tasks.

REFERENCES

[1] D. R. Poduval and P. Rajalakshmy, "A review paper on autonomous mobile robots," in *Proc. AIP Conf.*, 2022, Art. no. 030005.

[2] H.-Y. Zhang, W.-M. Lin, and A.-X. Chen, "Path planning for the mobile robot: A review," *Symmetry*, vol. 10, no. 10, p. 450, Oct. 2018, doi: 10.3390/sym10100450.

[3] I. Iswanto, A. Ma'Arif, O. Wahyunggoro, and A. Imam, "Artificial potential field algorithm implementation for quadrotor path planning," *Int. J. Adv. Comput. Sci. Appl.*, vol. 10, no. 8, pp. 1–11, 2019.

[4] 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.

[5] L. C. Garaffa, M. Basso, A. A. Konzen, and E. P. de Freitas, "Reinforcement learning for mobile robotics exploration: A survey," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Nov. 12, 2021, doi: 10.1109/TNNLS.2021.3124466.

[6] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, Nov. 1996. [Online]. Available: https://www.jair.org/index.php/jair/article/view/10166/24110

[7] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: Past, present, and future," *Multimedia Tools Appl.*, vol. 80, pp. 8091–8126, Oct. 2020.

[8] S. Mirjalili, "Genetic algorithm," in *Evolutionary Algorithms and Neural Networks* (Studies in Computational Intelligence), 2018, pp. 43–55.

[9] M. Dorigo and T. Stützle, "Ant colony optimization: Overview and recent advances," in *Handbook of Metaheuristics* (International Series in Operations Research & Management Science), 2018, pp. 311–351.

[10] C. Lamini, S. Benhlima, and A. Elbekri, "Genetic algorithm based approach for autonomous mobile robot path planning," *Proc. Comput. Sci.*, vol. 127, pp. 180–189, Jan. 2018.

[11] Y. Li, J. Zhao, Z. Chen, G. Xiong, and S. Liu, "A robot path planning method based on improved genetic algorithm and improved dynamic window approach," *Sustainability*, vol. 15, no. 5, p. 4656, Mar. 2023.

[12] F. H. Ajeil, I. K. Ibraheem, A. T. Azar, and A. J. Humaidi, "Grid-based mobile robot path planning using aging-based ant colony optimization algorithm in static and dynamic environments," *Sensors*, vol. 20, no. 7, p. 1880, Mar. 2020.

[13] S. Zhang, J. Pu, and Y. Si, "An adaptive improved ant colony system based on population information entropy for path planning of mobile robot," *IEEE Access*, vol. 9, pp. 24933–24945, 2021.

[14] B. K. Patle, A. Pandey, D. R. K. Parhi, and A. Jagadeesh, "A review: On path planning strategies for navigation of mobile robot," *Defence Technol.*, vol. 15, pp. 582–606, Aug. 2019.

[15] D. Tamilselvi, S. M. Shalinie, and G. Nirmala, "Q learning for mobile robot navigation in indoor environment," in *Proc. Int. Conf. Recent Trends Inf. Technol. (ICRTIT)*, Jun. 2011, pp. 324–329, doi: 10.1109/ICRTIT.2011.5972477.

[16] P. K. Das, S. C. Mandhata, H. S. Behera, and S. N. Patro, "An improved Q-learning algorithm for path-planning of a mobile robot," *Int. J. Comput. Appl.*, vol. 51, no. 9, pp. 40–46, Aug. 2012, doi: 10.5120/8073-1468.

[17] 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, doi: 10.1109/TSMCA.2012.2227719.

[18] I. Goswami, P. K. Das, A. Konar, and R. Janarthanan, "Conditional Q-learning algorithm for path-planning of a mobile robot," in *Proc. Int. Conf. Ind. Electron., Control Robot.*, Dec. 2010, pp. 23–27, doi: 10.1109/IECR.2010.5720165.

[19] S. Li, X. Xu, and L. Zuo, "Dynamic path planning of a mobile robot with improved Q-learning algorithm," in *Proc. IEEE Int. Conf. Inf. Autom.*, Aug. 2015, pp. 409–414, doi: 10.1109/ICInfA.2015.7279322.

[20] V. M. Babu, U. V. Krishna, and S. K. Shahensha, "An autonomous path finding robot using Q-learning," in *Proc. 10th Int. Conf. Intell. Syst. Control (ISCO)*, Jan. 2016, pp. 1–6, doi: 10.1109/ISCO.2016.7727034.

[21] L. Peng and J. Li, "Grid path planning for mobile robots with improved Q-learning algorithm," in *Proc. LISS*, 2020, pp. 583–594, doi: 10.1007/978-981-15-5682-1_42.

[22] H. Qianqian and K. Li, "An improved genetic algorithm for mobile robot path planning in grid environment," *Proc. SPIE*, vol. 11574, p. 17, Oct. 2020, doi: 10.1117/12.2576111.

[23] L. Zhang, L. Tang, S. Zhang, Z. Wang, X. Shen, and Z. Zhang, "A self-adaptive reinforcement-exploration Q-learning algorithm," *Symmetry*, vol. 13, no. 6, pp. 1–16, 2021.

[24] C. Jin, Y. Lu, R. Liu, and J. Sun, "Robot path planning using Q-learning algorithm," in *Proc. 3rd Int. Symp. Robot. Intell. Manuf. Technol. (ISRIMT)*, Sep. 2021, pp. 202–206, doi: 10.1109/ISRIMT53730.2021.9596694.

[25] H. Gao, Y. Liu, S. Su, and W. Yao, "A simple implement of Q-learning in robot path planning," *J. Phys., Conf. Ser.*, vol. 2294, no. 1, Jun. 2022, Art. no. 012034, doi: 10.1088/1742-6596/2294/1/012034.

[26] L. Khriji, F. Touati, K. Benhmed, and A. Al-Yahmedi, "Mobile robot navigation based on Q-learning technique," *Int. J. Adv. Robot. Syst.*, vol. 8, no. 1, pp. 45–51, Jan. 2011, doi: 10.5772/10528.

[27] B. S. B. Dewantara, "Building a socially acceptable navigation and behavior of a mobile robot using Q-learning," in *Proc. Int. Conf. Knowl. Creation Intell. Comput. (KCIC)*, Nov. 2016, pp. 88–93, doi: 10.1109/KCIC.2016.7883630.

[28] H. X. Pham, H. M. La, D. Feil-Seifer, and L. Van Nguyen, "Reinforcement learning for autonomous UAV navigation using function approximation," in *Proc. IEEE Int. Symp. Saf., Secur., Rescue Robot. (SSRR)*, Aug. 2018, pp. 1–6, doi: 10.1109/SSRR.2018.8468611.

[29] C. Chen, X.-Q. Chen, F. Ma, X.-J. Zeng, and J. Wang, "A knowledge-free path planning approach for smart ships based on reinforcement learning," *Ocean Eng.*, vol. 189, Oct. 2019, Art. no. 106299, doi: 10.1016/j.oceaneng.2019.106299.

[30] G. A. Cardona, C. Bravo, W. Quesada, D. Ruiz, M. Obeng, X. Wu, and J. M. Calderon, "Autonomous navigation for exploration of unknown environments and collision avoidance in mobile robots using reinforcement learning," in *Proc. SoutheastCon*, Apr. 2019, pp. 1–7, doi: 10.1109/SoutheastCon42311.2019.9020521.

[31] Y. Zhang, X. Wei, and X. Zhou, "Dynamic obstacle avoidance based on multi-sensor fusion and Q-learning algorithm," in *Proc. IEEE 3rd Inf. Technol., Netw., Electron. Autom. Control Conf. (ITNEC)*, Mar. 2019, pp. 1569–1573, doi: 10.1109/ITNEC.2019.8729554.
[32] S. Li, X. Wang, L. Hu, and Y. Liu, "Mobile robot path planning based on Q-learning algorithm," in *Proc. WRC Symp. Adv. Robot. Autom.*, Aug. 2019, pp. 160–165, doi: 10.1109/WRC-SARA.2019.8931944.
[33] T. Ribeiro, F. Goncalves, I. Garcia, G. Lopes, and A. F. Ribeiro, "Q-learning for autonomous mobile robot obstacle avoidance," in *Proc. IEEE Int. Conf. Auto. Robot Syst. Competition (ICARSC)*, Apr. 2019, pp. 1–7, doi: 10.1109/ICARSC.2019.8733621.
[34] A. Maoudj and A. Hentout, "Optimal path planning approach based on Q-learning algorithm for mobile robots," *Appl. Soft Comput.*, vol. 97, Dec. 2020, Art. no. 106796, doi: 10.1016/j.asoc.2020.106796.
[35] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992, doi: 10.1023/A:1022676722315.
[36] J. Clifton and E. Laber, "Q-learning: Theory and applications," *Annu. Rev. Statist. Appl.*, vol. 7, pp. 279–301, Mar. 2020, doi: 10.1146/annurev-statistics-031219-041220.
[37] V. N. Sichkar, "Reinforcement learning algorithms in global path planning for mobile robot," in *Proc. Int. Conf. Ind. Eng., Appl. Manuf. (ICIEAM)*, Mar. 2019, pp. 1–5.
[38] N. R. Sturtevant, "Benchmarks for grid-based pathfinding," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 2, pp. 144–148, Jun. 2012.

NASR ABDALMANAN received the B.Eng. degree in mechatronic engineering from Universiti Malaysia Perlis (UniMAP), Perlis, Malaysia, in 2021, where he is currently pursuing the Ph.D. degree in mechatronic engineering with the Faculty of Electrical Engineering and Technology. He is a Graduate Research Assistant with the Faculty of Electrical Engineering and Technology, UniMAP. His research interest includes reinforcement learning for mobile robot path planning, specifically for mapping applications.

KAMARULZAMAN KAMARUDIN received the first degree in mechatronics engineering from the University of Canterbury, New Zealand. He is currently pursuing the Ph.D. degree in mobile robot olfaction with Universiti Malaysia Perlis (UniMAP). He is an Associate Professor with the Faculty of Electrical Engineering and Technology, UniMAP. His research interests include robotics, reinforcement learning, SLAM, image and 3D point cloud processing, and odor sensing and mapping.

MUHAMMAD AIZAT ABU BAKAR received the master's degree in mechatronic engineering from Universiti Malaysia Perlis (UniMAP), Malaysia, in 2018.

From 2015 to 2018, he was a Research Fellow with the School of Mechatronic Engineering, UniMAP. He is a University Lecturer teaching bachelor's degree in robotic and automation engineering technology program with the Faculty of Electrical Engineering and Technology (FKTE), UniMAP. He is the author of more than 40 articles and more than ten inventions. His research interests include electronic noses, sensors, and mobile robots.

Mr. Bakar was a recipient of the Professional Technologist authorized by the Malaysia Board of Technologists (MBOT), in 2021, and the Win International Special Award from the Taiwan Invention Products Promotion Association (TIPPA) in Perlis International Engineering Invention and Innovation Exhibitions, in 2021.

MOHD HAFIZ FAZALUL RAHIMAN received the B.Eng. degree in electrical (control and instrumentation) and the M.Eng. and Ph.D. degrees in electrical engineering from Universiti Teknologi Malaysia, in 2003, 2005, and 2013, respectively. In 2006, he joined Universiti Malaysia Perlis (UniMAP), Perlis, Malaysia, as a Faculty Member, where he is currently an Associate Professor with the Faculty of Electrical Engineering and Technology. His research interests include process tomography, sensors, and instrumentation.

AMMAR ZAKARIA received the bachelor's degree in electric and electronic engineering from Portsmouth University, U.K., and the Ph.D. degree in mechatronic engineering from Universiti Malaysia Perlis (UniMAP). He is currently the Manager of the Centre of Excellence for Advanced Sensor Technology (CEASTech). His current research interests include sensor technology, artificial intelligence, and the Internet of Things (IoT).

SYED MUHAMMAD MAMDUH received the bachelor's degree (Hons.) in mechatronics engineering from the University of Canterbury, New Zealand, in 2010, and the Ph.D. degree in mechatronic engineering from Universiti Malaysia Perlis (UniMAP), in 2017. He has been a Senior Lecturer with UniMAP. His current research interests include mobile robots, machine intelligence, and chemical sensing.

LATIFAH MUNIRAH KAMARUDIN is currently an Associate Professor with the Faculty of Electronic Engineering Technology (FTKEN), Universiti Malaysia Perlis (UniMAP), where she leads the Wireless Communication Research Group, Centre of Excellence for Advanced Sensor Technology (CEASTech). Her research interests include the Internet of Things (IoT), cloud analytics, radio tomographic imaging, cognitive radio, and wireless communication and networks.