2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)
August 23-27, 2021. Lyon, France

# Q-learning-based navigation for mobile robots in continuous and dynamic environments

Abderraouf Maoudj and Anders Lyhne Christensen

2021 IEEE 17th International Conference on Automation Science and Engineering (CASE) | 978-1-6654-1873-7/21/$31.00 ©2021 IEEE | DOI: 10.1109/CASE49439.2021.9551641

***Abstract*— Autonomous collision-free navigation in dynamic environments is essential in many mobile robot applications. Reinforcement learning has the potential to automate the control design process and it has been successfully applied to path planning and mobile robot navigation. Obtaining effective navigation strategies with reinforcement learning is, however, still very time-consuming, especially in complex continuous environments where the robot can easily get trapped. In this paper, we propose a novel state space definition that includes information about the robot's most recent action. The proposed state variables for the target and nearby obstacles are binary and denote presence (or absence) within a corresponding region in the robot's frame of reference. In addition, we propose two heuristic algorithms that provide the robot with basic prior knowledge about a promising action in each state, which reduces the initial time-consuming blind exploration and thereby significantly shortens training time. We train a robot using our improved Q-learning approach to navigate in continuous environments in a high-fidelity simulator. In a series of experiments, we demonstrate the effectiveness of the proposed approach in terms of training time and solution quality compared to state-of-the-art approaches.**

## I. Introduction

Recent advances in the field of robotics mean that mobile robots have the potential to take over tasks currently carried out by humans. Consequently, mobile robots have become a focus of attention in industries, such as manufacturing and warehouse management [1], [2]. Manual design of control for autonomous robots can, however, be both challenging and time-consuming. In the light of this, there is significant interest, both in academia and in industry, to develop algorithms that can automate the design of control, and enhance robot efficiency and degree of autonomy.

Reinforcement learning is a machine learning paradigm in which an agent learns to map states to actions to maximize cumulative reward over time [3]. The reward is given by a task-specific reward function designed to reflect the quality of an agent's behavior. Q-learning (QL) is a reinforcement learning algorithm that iteratively updates a *Q-table* with estimates of the expected cumulative reward of taking different actions in the states visited during training. Research on path planning and obstacle avoidance using QL has recently received considerable attention [4], [5], and it has been successfully applied to mobile robot navigation, see [6], [7], [8] for examples. In [9], an approach based on an improved QL algorithm and heuristic search strategies is proposed for mobile robot path planning in dynamic environments. The approach combines $\epsilon$-greedy exploration with Boltzmann exploration. Another improved QL algorithm is proposed in [10] that uses inverse Euclidean distance-based Q-table initialization and an improved $\epsilon$-greedy algorithm which is combined with Boltzmann exploration and heuristic search strategies for action selection. Although these approaches were shown to speed up convergence, they were only applied to relatively simple discrete grid-world environments and to a simple action space that only enabled the agent to move to neighboring cells.

In [11], an approach for mobile robot navigation that combines QL and a *dynamic window approach* [12] is proposed. The dynamic window approach is used for local path planning and QL is applied to tune the weights of each evaluation function. A series of simulation-based experiments were used to assess the performance of the approach. However, the simulation environments were relatively simple and contained only static obstacles. In [7], the concept of *partially guided QL* was introduced wherein the *flower pollination algorithm* [13] was applied to initialize the Q-table with heuristic estimates. The results showed that the proposed QL algorithm outperforms the classic QL algorithm. However, the authors only considered discrete static environments, and the obtained navigation trajectories were not smooth since actions were limited to movements in a few predefined directions.

Although QL has shown prominent performance in several mobile robot applications, there are still significant challenges to address before it can be adopted as a general approach to design control for autonomous navigation. First, many of the state-of-the-art methods, such as those discussed above, have only been demonstrated in relatively simple discrete grid worlds [4], and may be too inefficient for large environments [14]. Second, for many algorithms, the state space only considers the spatial relation between the robot and nearby obstacles or the target, but ignores the robot's orientation, which may be insufficient to describe the robot's state [11]. In addition, the action space is relatively simple and generally limited to a few movements in predefined directions. Another challenge is *over-fitting*, where the robot is only able to display good performance in simple training environments and performs poorly in unseen complex environments [15]. Thus, the efficiency, generalizability, and scalability of existing QL-based approaches are still insufficient for many real-world applications [14].

In this paper, we make the following contributions to address the above-mentioned issues: (i) we proposed a novel

A. Maoudj and A. L. Christensen are with SDU Biorobotics, MMMI, University of Southern Denmark (SDU), Odense, Denmark, email: abma@mmmi.sdu.dk, andc@mmmi.sdu.dk. This work was supported by the Independent Research Fund Denmark under grant 0136-00251B.

and efficient definition of the state space that includes the previous action performed by the robot to enable it to navigate in complex continuous environments; (ii) we propose to give the robot prior knowledge in the form of two heuristic algorithms to significantly speed up learning; (iii) we demonstrate the efficiency of our approach in a high-fidelity simulator, namely CoppeliaSim [16]; and (iv) we compare the proposed approach to other state-of-the-art methods.

## II. Proposed improved Q-learning approach

In this section, we present our approach to improve the efficiency of QL for mobile robot navigation in continuous dynamic environments. We assume that the robot is a differential-drive mobile robot that is able to turn on the spot at any time. In addition, the robot is assumed to always know its position and orientation and the location of the target. To apply QL, four major components must be defined: (i) the discrete set of actions that the robot can perform (the action space), (ii) the set of states that the robot can be in (the state space), (iii) the reward function, and (iv) the action selection strategy which should ensure a good balance between exploration and exploitation. In the following subsections, each of these parts of our improved Q-learning (IQL) approach is explained in detail.

### A. Action space definition

In literature, actions often restrict the robot to relatively simple movements [17], [4], [7], for instance, turn right or left with a fixed angle or move in one of a limited number of directions when a grid-based representation of the environment is used, see [4] for an example. Such limited movements tend to make the robot's path jagged and longer than necessary. In this work, we define the action space, such that (i) the robot can move effectively and smoothly, (ii) the size of the action space is kept small enough for learning to be efficient, and (iii) the action space contains specific movements that allow a robot to safely and effectively avoid obstacles (see Fig. 1). More specifically, the action space must enable the robot to:

- Move forward and turn left/right.
- Rotate on the spot (around $z$-axis) to assume any orientation. Unlike several state-of-the-art approaches, such as [18], that do not consider the robot's orientation, the proposed IQL considers the robot's orientation to enable the robot to navigate in realistic environments with complex obstacles.
- Navigate around obstacles in smooth trajectories, as illustrated in Fig. 1.

To this end, we conducted a series of preliminary experiments using a model of the Pioneer 3-DX in CoppeliaSim to find actions that enable the robot to navigate effectively around complex obstacles. During the experiments, we identified a set of different speeds $\{v_{max}$, $v_{max}/2$, $v_{max}/3$ and $v_{max}/4\}$ by which the robot can perform the required actions. In this work, the action space $A$ contains eight actions that each is a pair of speeds, one for the right wheel and one for the left wheel $(V_R,V_L)$: $A = \{ (-v_{max}/3, v_{max}/3), (v_{max}/3, -v_{max}/3), (v_{max}/4, v_{max}/2), (v_{max}/3, v_{max}/4), (v_{max}/4, v_{max}/3), (v_{max}/2, v_{max}/4), (v_{max}/3, v_{max}/3), (v_{max}, v_{max})\}$. The first two actions enable the robot to rotate around the $z$-axis. The third to sixth action allow the robot to navigate round obstacles by following a trajectory with a small curvature or a large curvature to the left or right. The last two actions enable the robot to move forward at different speeds.

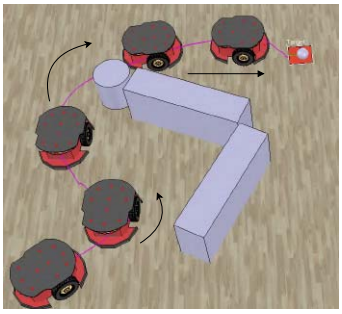


Fig. 1: Example of effective navigation around an obstacle.

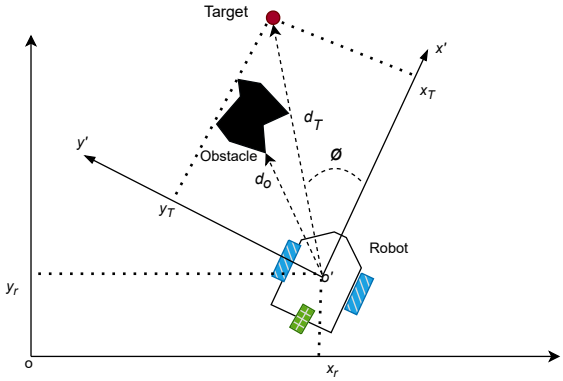
### B. State space definition

The state space definition is one of the most important aspect of QL [3]. Indeed, the states should have the Markov property and provide enough information to estimate the expected cumulative reward of taking actions in each state, without the need for any additional information. To this end, we define a state space that consists of three types of information or *state variables*: (i) *Obstacle:* the presence or absence of obstacles in the robot's immediate surroundings, (ii) *Target:* the location of the target within the robot's frame of reference, and (iii) *Last action:* the action selected by the robot in the previous time step.

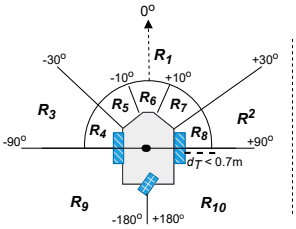
**Obstacle state variables:** The robot's immediate surroundings are divided into five regions covering the area in front of the robot, as shown in Fig. 2. The proposed state space only considers obstacles in front of the robot, and obstacles behind the robot are ignored. Each of the areas, *front region* (FR), *right region* (RR), *left region* (LR), *right side region* (RSR), and *left side region* (LSR) are assigned either *True* or *False* to denote whether or not an obstacle is present in the corresponding region. The regions extend to $d_o = 0.8\,\text{m}$ from the robot.

**Target state variable:** The target state variable consists of two parts, namely the region $R_i$ that contains the relative direction to the target from the robot's perspective, and a flag *T-in-SR* that indicates whether the target is located in a safe region (*SR*) or in an unsafe region (*USR*).

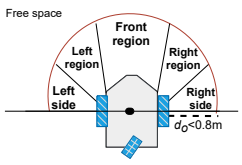
Determining whether the target is located in a SR or an USR is based on the target region $R_i$, the obstacle region flags that are *True*, and the distances $d_T$ and $d_o$. The target is located in an *USR* (i.e. *T-in-SR* = *False*) if there is at least one obstacle between the target and the robot (i.e. the obstacle flag in the direction of the target is True) and $d_T > d_o$.




(a) Coordinates




(b) Target regions




(c) Obstacle regions

Fig. 2: Coordinates and regions. (a) The system coordinates, (b) target regions according to distance $d_T$ and orientation with respect to the target $\Theta$, and (c) regions for nearby obstacles, detected by the robot's sensors (ultrasonic or laser scanner, for instance).

Otherwise, the target is located in *SR* which means that *T-in-SR = True*.

The target state variable is set based on the distance $d_T$, a threshold (0.7 m), and the angle $\Theta$ between the robot heading direction and target. Note that in each time step, the target position $(x_T, y_T)$ must be calculated in the robot base frame (frame $o'$). The equations of the distance $d_T$ and the angle $\Theta$ (deg) are given in the robot base frame as follows:

$$\begin{aligned} d_T &= \sqrt{x_T^2 + y_T^2} \\ \Theta &= \text{atan2}(Y_T, X_T) \cdot \frac{180}{\pi} \end{aligned}$$

The angular region $R_i$ where the target is located is calculated as follows:

$$R_i = \begin{cases} R_1 & \text{if } \Theta \in [-30, +30] \quad \wedge \quad d_T > 0.7\text{m} \\ R_2 & \text{if } \Theta \in ]+30, +90] \quad \wedge \quad d_T > 0.7\text{m} \\ R_3 & \text{if } \Theta \in [-90, -30[ \quad \wedge \quad d_T > 0.7\text{m} \\ R_4 & \text{if } \Theta \in [-90, -30[ \quad \wedge \quad d_T < 0.7\text{m} \\ R_5 & \text{if } \Theta \in [-30, -10[ \quad \wedge \quad d_T < 0.7\text{m} \\ R_6 & \text{if } \Theta \in [-10, +10] \quad \wedge \quad d_T < 0.7\text{m} \\ R_7 & \text{if } \Theta \in ]+10, +30] \quad \wedge \quad d_T < 0.7\text{m} \\ R_8 & \text{if } \Theta \in ]+30, +90] \quad \wedge \quad d_T < 0.7\text{m} \\ R_9 & \text{if } \Theta \in [-180, -90[ \\ R_{10} & \text{if } \Theta \in [+90, +180] \end{cases}$$

To complete the definition of the target state variable, it is necessary to define whether the target is located in a safe region or in an unsafe region (flag *T-in-SR*). According to the obstacle region, $d_o$, $d_T$ and target region $R$, the flag *T-in-SR* is defined as follows:

$$\textit{T-in-SR} = \begin{cases} \text{True} & \text{if } \exists obs \in Ri \forall (i = 1...10) \wedge d_T > d_o \\ \text{False} & \text{Otherwise} \end{cases}$$

**The complete definition of states:** Now that state variables for the target, and the presence or absence of nearby obstacles have been detailed, the complete state space can be defined as follows:

$$s_t = \{FR, RR, LR, RSR, LSR, R_i, \textit{T-in-SR}, \textit{Last Action}\}$$

### C. Reward function

The robot's aim is to reach the target as quickly as possible while avoiding colliding with obstacles. In the proposed approach, the reward function $R(s_t, a_t, s_{t+1})$, which gives the robot a reward after performing an action $a_t$ at time step $t$ that takes the robot from state $s_t$ to state $s_{t+1}$, is defined as follows:

$$R = \alpha \cdot (N_o + \beta) + \cos\Theta \cdot \left(\sigma + \frac{-\Delta d}{1 + N_o}\right) + \eta \quad (1)$$

where $\Theta$ is the robot's relative direction to the target and $\Theta \in [-90°, +90°]$ (in case the target is behind the robot, $\Theta$ is clipped to the nearest limit). The $\cos\Theta$ increases as the angle $\Theta$ decreases and thus encourages the robot to turn towards the target during its movement.

The variable $\Delta d$ denotes the difference in the robot's Euclidean distance to the target at $s_{t+1}$ and $s_t$. The variable $N_o$ is the number of the flags related to obstacle regions that are *True* in the state $s_{t+1}$. Finally, $\alpha$, $\beta$, $\sigma$ and $\eta$ are variables that are defined as follows:

$$\alpha = \begin{cases} 0 & \text{if the flag } \textit{T-in-SR} \text{ in } s_{t+1} \text{ is True} \\ -1 & \text{Otherwise} \end{cases}$$

$$\beta = \begin{cases} 0 & \text{if the flag } \textit{FR} \text{ in } s_{t+1} \text{ is True} \\ -1 & \text{Otherwise} \end{cases}$$

$$\sigma = \begin{cases} 0 & \text{if } \Delta d \neq 0 \\ 1 & \text{if } \Delta d = 0 \wedge \Theta_t - \Theta_{t+1} <= \Theta_t \\ -1 & \text{if } \Delta d = 0 \wedge \Theta_t - \Theta_{t+1} > \Theta_t \end{cases}$$

$$\eta = \begin{cases} +M & \text{if the robot reaches the target} \\ -M & \text{if the robot collides with an obstacle} \\ 0 & \text{Otherwise} \end{cases}$$

where $M$ is the largest reward value the robot can receive after performing an action $a \in A$ (in our case, $M$ is set to 50).

The reward function takes into account the target's relative location with respect to the robot, the obstacle regions in front of the robot, and the robot's orientation in the new state $s_{t+1}$. We use $\cos\Theta$ to make the reward proportional to

**Algorithm 1:** Heuristic algorithm for navigation when no obstacle is nearby

**input** : Current state: $S_t$
**output** : Best action $a$, where $a \in A$

**if** *Target in* $R_1$ **then**
  Move fast forward: $a$ = *A[8]*
**if** *Target in* $R_2$ **then**
  Turn right while making a small curvature: $a$ = *A[5]*
**if** *Target in* $R_3$ **then**
  Turn left while making a small curvature: $a$ = *A[4]*
**if** *Target in* $R_6$ **then**
  Move forward with small speed: $a$ = *A[7]*
**if** *Target in* $R_5 \vee R_4 \vee R_9$ **then**
  rotate around the $z$-axis to the left: $a$= *A[2]*
**if** *Target in* $R_7 \vee R_8 \vee R_10$ **then**
  rotate $z$-axis to the right: $a$= *A[1]*
**return** $a$

**Algorithm 2:** Heuristic algorithm for avoidance obstacles while moving to target

**input** : Current state: $s_t$
**output** : Best action $a$, where $a \in A$

**if** *Unsafe Front Region* **then**
  **if** *Safe Right and Safe Left* **then**
    **if** *Target in* $(R_7 \vee R_1 \vee R_6) \wedge$ *Safe Right Side* **then**
      rotate *around z-axis* to the right: $a$= *A[1]*
    **if** *Target in* $(R_5 \vee R_1 \vee R_6) \wedge$ *safe Left Side* **then**
      rotate *around z-axis* to the left: $a$= *A[2]*
    **else**
      $a$ = choose the $LastAction$
  **else if** *Unsafe Left and Unsafe Right* **then**
    **if** *Target* $R_7 \wedge$ *Safe Right Side* **then**
      rotate *around z-axis* to the right: $a$ = *A[1]*
    **if** *Target* $R_5 \wedge$ *Safe Left Side* **then**
      rotate *around z-axis* to the left: $a$ = *A[2]*
    **else**
      $a$ = choose the $LastAction$
  **else if** *Safe Right or Safe Left* **then**
    rotate *around z-axis* to the safe region
**else if** *Unsafe Right Region* **then**
  **if** *Safe Front* **then**
    Move forward with small speed: $a$ = *A[7]*
  **else if** *Safe Left* **then**
    rotate *around z-axis* to the left:$a$ = *A[2]*
  **else if** *Safe Right Side and* $LastAction \neq 2$ **then**
    rotate *around z-axis* to the right: $a$ = *A[1]*
  **else if** *Safe Left Side and* $LastAction \neq 1$ **then**
    rotate *around z-axis* to the left:$a$ = *A[2]*
  **else**
    $a$ = choose the $LastAction$
**else if** *Unsafe Left Region* **then**
  **if** *Safe Front* **then**
    Move forward with small speed: $a$ = *A[7]*
  **else if** *Safe right* **then**
    rotate *around z-axis* to the right: $a$ = *A[1]*
  **else if** *Safe Left Side and* $LastAction \neq 1$ **then**
    rotate *around z-axis* to the left: $a$ = *A[2]*
  **else if** *Safe right Side and* $LastAction \neq 2$ **then**
    rotate *around z-axis* to the right: $a$ = *A[1]*
  **else**
    $a$ = choose the $LastAction$
**return** $a$

the robot's alignment with respect to the target, thus pushing the robot to orient itself towards the target. Moreover, the distance minimization $\Delta d$ parameter is used such that the reward value is directly proportional to this parameter, which encourages the robot to move to the target direction and minimize the distance $d_T$. This means that the robot learns to condition its actions on the distance and orientation to the target, and on the presence or absence of obstacles in front of the robot.

### *D. Heuristic prior knowledge*

Using prior knowledge to initialize the Q-table can significantly speed up the learning process of QL algorithms [7]. In our approach, two heuristic algorithms are introduced to give the robot an effective learning foundation and provide it with basic prior knowledge about a promising action in each state, which reduces initial time-consuming blind exploration.

The proposed heuristic algorithms are based on simple rules that were adjusted empirically based on preliminary experiments conducted in the CoppeliaSim simulator. According to the adopted rules, at first, the robot adjusts its orientation to point toward the target, and then moves straight forward if no obstacle is detected ahead. If an obstacle is detected in front of the robot, the robot tries to avoid the obstacle by turning either left or right depending on the target location. Moreover, to avoid the oscillation between two states in situations where the robot could otherwise become trapped, such as inside U-shaped obstacles, the action selected in the last state is used. In these situations, the robot rotates on the spot (i.e. around $z$-axis) in the direction of the target until it finds an obstacle-free path. These basic principles are transformed into the two heuristic algorithms presented in Algorithm 1 and Algorithm 2.

### *E. Action selection strategy*

In reinforcement learning, the performance of the learning process is closely related to the trade-off between exploration and exploitation. For tackling this trade-off, we use an $\epsilon-$greedy-based algorithm (Algorithm 3). As shown in Algorithm 3, instead of always selecting the action with the highest Q-value in the Q-table or defined by heuristic Algorithm 1 and thereby maximizing the immediate reward, the algorithm randomly selects one action not tried before from $A$ with a small probability. This action will be selected from a set of actions having an uninitialized Q-value. In addition, to reduce exploration over time, in our proposed action selection strategy, $\epsilon$ is reduced by $\epsilon-decay$ every episode.

**The proposed improved Q-Learning algorithm:** Algorithm 4 describes the proposed IQL that returns an optimized policy for mobile robot navigation while using the proposed Algorithm 1 and 2 for initializing the $Q$-table and the action selection in Algorithm 3. The IQL constantly learns the policy by maintaining an estimate of the expected return for each state-action pair in the $Q$-table. This estimate is updated as the robot explores the environment.

**Algorithm 3:** Action Selection Strategy

**input** : Current state: $s_t$, $\epsilon$
**output** : An action $a$, where $a \in A$

$r$ = random$(0, 1)$
**if** $r < \epsilon$ **then**
  $B$ = Set of actions with uninitialized Q-value
  **if** $B \neq \varnothing$ **then**
    $a$ = Select a random action from the set $B$
  **else**
    $a$ = Select a random action from $A$
**else**
  $r'$= random$(0, 1)$
  **if** $r' < \epsilon$ **then**
    **if** *no obstacle nearby* **then**
      $a$ = Algorithm1$(s_t)$
    **else**
      $a$ = Algorithm2$(s_t)$
  **else**
    $a = \arg\max$(Q-table$(s_t, :)$)
**return** $a$

**Algorithm 4:** Proposed $IQL$

**input** : Environment, states $S$, actions $A$, reward function $R$: Eq. (1), discounting factor $\gamma$, learning rate $\delta$, $\delta - decay$, $\epsilon$ and $\epsilon - decay$
**output** : Learned $Q$-table

Initialization:
Initialize $Q(s, a)$, using *Algorithm 1 and 2*
**foreach** *episode* **do**
  Initialize the Target with new random position
  **foreach** *step in the current episode* **do**
    Calculate current state $s_t$
    $a \leftarrow$ *Algorithm3(s)* // Select an action $a$
    $s_{t+1} \leftarrow Execute(a)$ // Receive the new state
    $r \leftarrow R(s_t, a, s_{t+1})$ ; // Receive the reward
    $Q(s_{t+1}, a) \leftarrow$
      $(1-\delta) \cdot Q(s_t, a) + \delta \cdot (r + \gamma \cdot \arg\max_{a'} Q(s_{t+1}, a'))$
    $s_t \leftarrow s_{t+1}$
  $\delta \leftarrow \delta \cdot (1 - \delta$-decay$)$
  $\epsilon \leftarrow \epsilon \cdot (1 - \epsilon$-decay$)$
**return** $Q$

## III. Experimental results and performances analysis

In this section, we present the results of a series of experiments carried out in different scenarios to assess the performance of the proposed IQL. The experiments are conducted with a virtual model of the Pioneer 3-DX using the CoppeliaSim simulator. The values of the key parameters listed in Table I were selected based on preliminary experiments.

| *Parameter* | *value* |
|---|---|
| Discount factor $\gamma$ | 0.9 |
| Learning rate, $\delta$ | 0.3 |
| Learning rate decay, $\delta$-decay | 0.0001 |
| $\epsilon$ | 0.4 |
| $\epsilon$-decay | 0.001 |
| $v_{max}$ | 2.0 m/s |
| Control time step | 100 ms |
| Simulation time step | 50 ms |

TABLE I: Parameter values

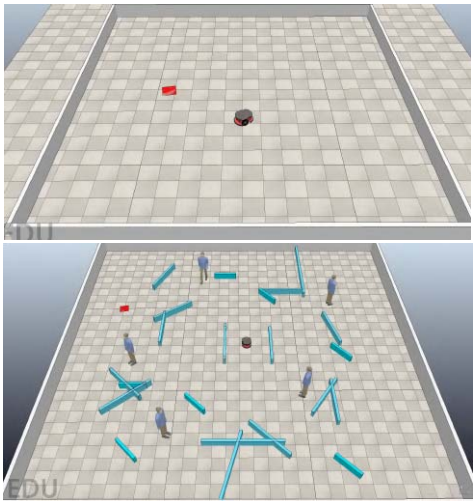


Fig. 3: Training environments.

### A. Training process

Training is conducted in two environments. The first environment, shown in Fig. 3(top), is obstacle-free and is used to train the robot to track the target. The second environment shown in Fig. 3(bottom) is more complex with static obstacles and simulated humans moving randomly in the environment as dynamic obstacles (humans are simulated by using *Walking Bill* objects from CoppeliaSim). The second environment is made so that the robot encounters many of the complex situations that it could face in a real-world application. Moreover, the target is placed at a random location at the beginning of each episode. The training finishes when the average number of steps required to reach the target converges and stays within a limited range.

Note that during the learning process, the value of $\epsilon$ and the learning rate, $\delta$, are subject to decay, see Table I. The $\epsilon$-decay causes exploration to be reduced over time, while the learning rate decay facilitates the combination of fast initial learning and fine tuning of the policy towards the end of the training process.

To assess the impact of the proposed heuristics shown by *Algorithm 1* and *Algorithm 2* on the convergence speed, we trained robots both with and without the proposed heuristic algorithms. The average number of steps (calculated every 20 episodes) to reach the target during training are shown in Fig. 4. In both experimental setups, the robot clearly learns to reach the target faster as training progresses. When the heuristic algorithms are used, however, learning is faster than if standard blind exploration through an $\epsilon$-greedy approach is used. Indeed, using the two heuristic algorithms speeds up learning significantly as it takes only $2,440$ episodes to reach the convergence point after which the average number of steps varies within a range of approximately 60-70 steps. Whereas without using the proposed heuristic algorithms, the IQL takes $6,900$ episodes to converge after which the average number of steps varies within a range of approximately 55-75 steps. Consequently, it can be concluded that with the proposed heuristics, the robot explores the environment

in a more effective manner and the convergence speed is significantly improved.

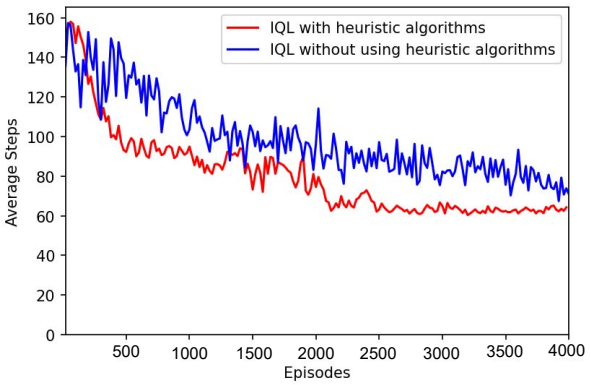


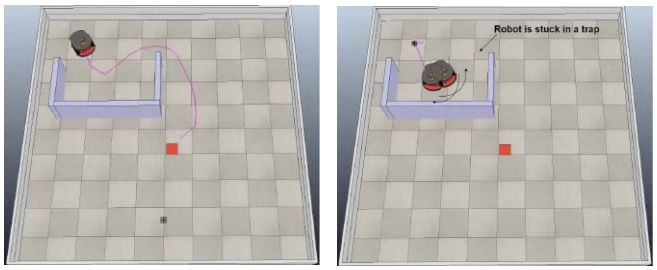

Fig. 4: Average number of steps to reach the target during training with and without using the proposed heuristic algorithms.

### B. Testing the trained policy

We designed different maps to assess the trained policy in a variety of situations. For each map, we ran 20 experiments with the robot's initial position and orientation randomly sampled from uniform distributions. In the first map, we used a U-shaped obstacle with the robot initially placed behind the obstacle to test if the inclusion of the *Last Action* state variable gives the robot the capacity to avoid getting trapped in dead ends. We ran 20 simulations with the robot trained with the state variable *Last Action* and 20 simulations with a robot trained without the *Last Action* state variable. The obtained results are shown in the first two rows in Table II. Through 20 simulations, the robot trained with the *Last Action* state variable successfully navigated around the U-shaped obstacle 19 times and it only failed once. The robot failed because the randomly sampled position of the robot was so close to the obstacle that the robot collided with the obstacle while trying to rotate on the spot (no action enables the robot to move backwards). Using the policy trained without the *Last Action* state variable, the robot was only able to surpass the obstacle in 7 of the 20 experiments. In all the 7 successful experiments, the robot had a favorable initial orientation and position, that allowed it to navigate safely around the obstacle. Fig. 5 shows two examples of the behavior displayed by the robot trained respectively with and without the *Last Action* state variable.

| *Scenario* | *Runs* | *Successes* | *Failures* |
|---|---|---|---|
| U-shaped obs. with *Last Action* | 20 | 19 | 1 |
| U-shaped obs. without *Last Action* | 20 | 7 | 13 |
| Map 1 | 20 | 20 | 0 |
| Map 2 | 20 | 20 | 0 |
| Map 3 | 20 | 20 | 0 |
| Map 4 | 20 | 20 | 0 |

TABLE II: Summary of successes and failures in different maps.




(a) (b)

Fig. 5: Trajectories around U-shape obstacles. (a) State space with *Last Action*, and (b) state space without using *Last Action*.

To assess the performance of the policy trained with the proposed IQL in complex environments, the four maps shown in Fig. 6 were used. The first three maps are adaptations of maps from [19], while the fourth map is a complex environment designed with dynamic obstacles. The maps and their arrangement of obstacles differ significantly and include different challenging situations. In addition, to assess the robustness of the trained policy, Map 4 was designed to contain situations that the robot could encounter in real-world applications, including dynamic obstacles. For the dynamic obstacles, we used eight dynamic *Walking Bill* models from CoppeliaSim and two static models to simulate humans in the environment.

The results obtained in 20 independent runs of each scenario are summarized in Table II. From these results it can be seen that on Map 1, Map 2 and Map 3, the robot was able to reach the target in all runs and was thus able to navigate safely around all obstacles encountered. Moreover, it is noteworthy that the robot successfully reached the target in Map 4 with dynamic obstacles in all runs.

Figure 6 illustrates examples of the paths taken by the robot navigating according to the learned policy. The efficiency of the proposed state space definition is reflected in the robot's ability to navigate safely around obstacles, follow walls, and reach the target. With regard to dynamic environments, a short and smooth path is achieved in Map 4 even when dynamic obstacles are present, see Fig. 6.

### C. Comparisons with other RL algorithms

We conducted experiments to compare our approach to the state-of-the-art algorithms studied in [18], namely neural Q-Learning (QL+NN) [20], Deep Q-learning with Experience Replay (DQL+ER), and Deep Q-learning with Experience Replay and a heuristic algorithm (DQL+ER+HR). To this end, we replicated the experimental environments presented in [18] in CoppeliaSim, as shown in Fig. 7, and conducted experiments with the robot trained with our approach.

Interesting results can be observed in Fig. 7, and the obtained trajectories hint at a more efficient policy trained with the proposed IQL compared to the RL algorithms studied in [18]. The policies learned with the different algorithms enable the robot to navigate to the target without

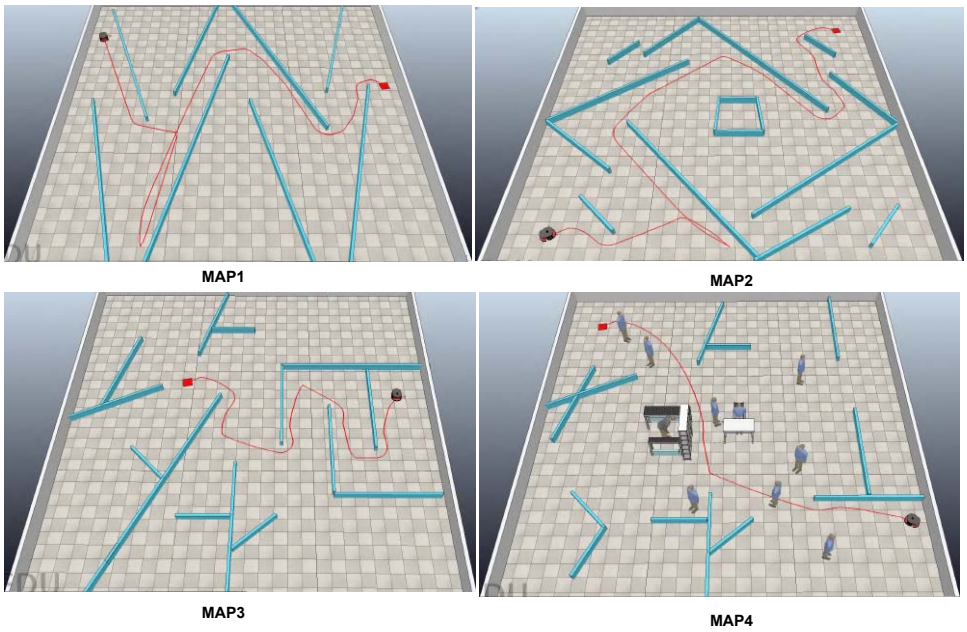



Fig. 6: Obtained paths on the proposed 3D maps.

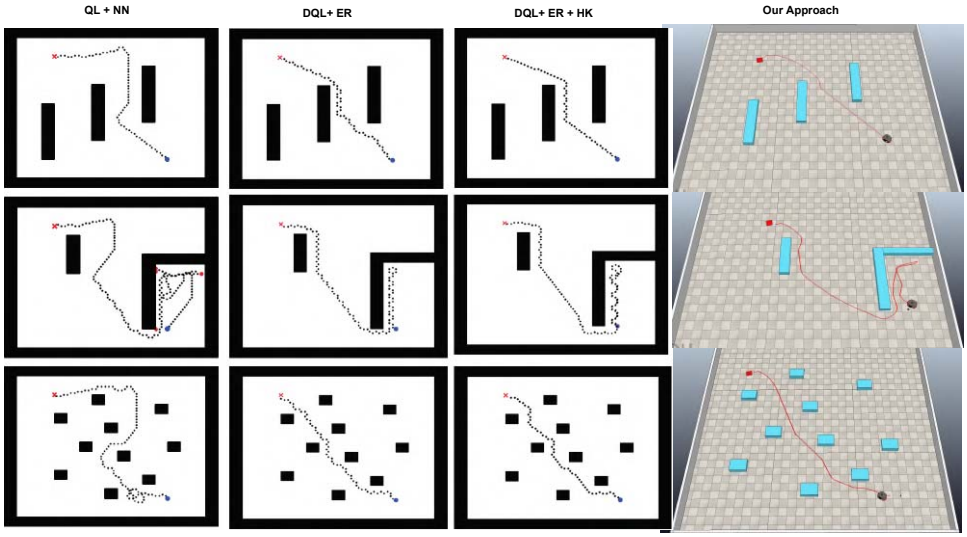



Fig. 7: Comparison with other reinforcement learning approaches. The figure on the left is a reprint from [18].

colliding with an obstacle. However, by visually comparing the trajectories, it can be observed that our approach yields shorter and smoother paths.

## IV. Conclusions

In this paper, we presented an improved Q-learning algorithm for mobile robot navigation in continuous and complex environments. We proposed a novel state space definition and two heuristic algorithms to generate prior knowledge. The state space definition included information about the robot's most recent action to help it avoid getting stuck when faced with challenging obstacles. The proposed heuristic algorithms were applied to initialize the Q-table with basic prior knowledge about a promising action in each state, which reduced the need for initial random exploration and thus shortened the training time significantly. In addition, the heuristic algorithms were integrated in the action selection strategy.

To comprehensively evaluate the proposed approach, we performed experiments to demonstrate its effectiveness and generalization ability to complex, continuous and dynamic environments. Through these experiments and by comparing the resulting trajectories to state-of-the-art algorithms, we conclude that the proposed approach is a promising step toward automatic design of control for efficient robot navigation in complex and dynamic environments.

## REFERENCES


[1] I. Nielsen, Q. Dang, G. Bocewicz, and Z. Banaszak, "A methodology for implementation of mobile robot in adaptive manufacturing environments," *Journal of Intelligent Manufacturing*, vol. 28, no. 5, pp. 1171–1188, 2017.

[2] H. Lee and J. Jeong, "Mobile robot path optimization technique based on reinforcement learning algorithm in warehouse environment," *Applied Sciences*, vol. 11, no. 3, p. 1209, 2021.

[3] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT Press, 2018.

[4] J. Jiang and J. Xin, "Path planning of a mobile robot in a free-space environment using Q-learning," *Progress in Artificial Intelligence*, vol. 8, pp. 133–142, 2019.

[5] T. T. Nguyen, N. D.Nguyen, and S. Nahavandi, "Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications," *IEEE Transactions on Cybernetics*, vol. 50, pp. 3826–3839, 2020.

[6] A. Maoudj and A. Hentout, "Optimal path planning approach based on Q-learning algorithm for mobile robots," *Applied Soft Computing*, vol. 97, p. 106796, 2020.

[7] E. S.Low, P. Ong, and K. C. Cheah, "Solving the optimal path planning of a mobile robot using improved Q-learning," *Robotics and Autonomous Systems*, vol. 115, pp. 143–161, 2019.

[8] R. Yuan, F. Zhang, Y. Wang, Y. Fu, and S. Wang, "A Q-learning approach based on human reasoning for navigation in a dynamic environment," *Robotica*, vol. 37, pp. 445–468, 2019.

[9] S. Li, X. Xu, and L. Zuo, "Dynamic path planning of a mobile robot with improved Q-learning algorithm," in *IEEE International Conference on Information and Automation*. IEEE Press, Piscataway, NJ, 2015, pp. 409–414.

[10] J. Wang, K. Hirota, X. Wu, Y. Dai, and Z. Jia, "An improved Q-learning algorithm for mobile robot path planning," in *Proceedings of the 9th International Symposium on Computational Intelligence and Industrial Applications (ISCIIA2020)*, 2020.

[11] L. Chang, L. Shan, and Y. C. Jiang, "Reinforcement based mobile robot path planning with improved dynamic window approach in unknown environment," *Autonomous Robots*, vol. 45, pp. 1–26, 2021.

[12] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[13] X.-S. Yang, "Flower pollination algorithm for global optimization," in *International Conference on Unconventional Computing and Natural Computation*. Springer, Berlin, Germany, 2012, pp. 240–249.

[14] B. Wang, Z.Liu, Q. Li, and A. Prorok, "Mobile robot path planning in dynamic environments through globally guided reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 5, pp. 6932–6939, 2020.

[15] X. Pan, W. Wang, X. Zhang, B. Li, J. Yi, and D. Song, "How you act tells a lot: Privacy-leaking attack on deep reinforcement learning," in *International Conference on Autonomous Agents and Multi-Agent Systems*. IFAAMAS, 2019, pp. 368–376.

[16] E. Rohmer, S. P. Singh, and M. Freese, "V-rep: A versatile and scalable robot simulation framework," in *Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE Press, Piscataway, NJ, 2013, pp. 1321–1326.

[17] S. Li, X. Xu, , and L. Zuo, "Dynamic path planning of a mobile robot with improved Q-learning algorithm," in *2015 IEEE International Conference on Information and Automation*. IEEE Press, 2015, pp. 409–414.

[18] L. Jiang, H.Huang, and Z. Ding, "Path planning for intelligent robots based on deep Q-learning with experience replay and heuristic knowledge," *IEEE/CAA Journal of Automatica Sinica*, vol. 7, 2020.

[19] J. Gomes, P. Mariano, and A. L. Christensen, "Devising effective novelty search algorithms: A comprehensive empirical study," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO)*. ACM Press, New York, NY, 2015, pp. 943–950.

[20] S. Parasuraman and S. Yun, "Mobile robot navigation: neural Q-learning," *International Journal of Computer Applications in Technology*, vol. 44, no. 4, pp. 303–311, 2013.