Palacios-Morocho, ME.; Inca, S.; Monserrat Del Río, JF. (2023). Multipath Planning Acceleration Method With Double Deep R-Learning Based on a Genetic Algorithm. IEEE Transactions on Vehicular Technology. 72(10):12681-12696. https://doi.org/10.1109/TVT.2023.3277981

Multipath Planning Acceleration Method with Double Deep R-learning based on a Genetic Algorithm

Elizabeth Palacios-Morocho, *Student Member, IEEE,* Saúl Inca and Jose F. Monserrat, *Senior Member, IEEE,*

***Abstract*—Autonomous navigation is a well-studied field in robotics requiring high standards of efficiency and reliability. Many studies focus on applying AI techniques to obtain a high-quality map, a precise localization, or improve the proposed trajectory to be followed by the agent. As traditional planning methods need a high-quality map to obtain optimal trajectories, this paper addresses the problem of multipath map-less planning, and proposes a novel multipath planning algorithm (Double Deep Reinforcement Learning - Enhanced Genetic (DDRL-EG)) for mobile robots in an unknown environment. It combines Double Deep Reinforcement Learning (DDRL) with Heuristic Knowledge (HK), Experience Replay (ER), Genetic Algorithm (GA), and Dynamic Programming (DP), allowing the agent to reach its target successfully without maps. In addition, it optimizes the training time and the chosen path in terms of time and distance to the target. A hybrid method is also used in which Semi-Uniform Distributed Exploration (SUDE) is employed to determine the probability that the action is decided based on directed knowledge, hybrid knowledge, or autonomous knowledge. The performance of DDRL-EG is compared with two other algorithms in two different environments. The results show that DDRL-EG is a more robust and powerful algorithm since with less training, it can provide much smoother and shorter trajectories to the target.**

***Index Terms*—Reinforcement learning, dynamic programming, prioritized experience, heuristic knowledge, genetic algorithm.**

I. Introduction

THE Industry 4.0 transforms all sectors, especially robotics in the field of autonomous navigation. To achieve the efficiency and reliability requirements necessary for this field, Artificial Intelligence (AI) plays an essential role in the automation process of the agents [1].

Over the last few years, AI has moved from "Internet AI", whose learning is based on images, videos, and texts from the Internet, to "Embodied AI," whose knowledge is based on the agent's interaction with its environment [2], [3]. A clear example of this type of agent is the "Vecna Robots", a robot that combines vision systems with machine learning technologies, allowing it to navigate safely a ground vehicle [4]. The goal of equipping agents with AI is to mimic both humans' adaptive and intelligent capabilities. This means that agents will be able to perceive complex environments and make decisions quickly [5]. Traditional approaches to autonomous navigation of an agent include detailed mapping of the environment, precise localization, and optimal path planning [6]–[8], where path planning is an essential component of autonomous navigation.

Nowadays, the most popular planning methods use Simultaneous Localization and Mapping (SLAM) technology. There are different proposals of SLAM algorithms that allow the construction of a real-time or offline map from data obtained from other types of sensors [9], [10]. Vision sensors and laser or ultrasonic range finders combined with SLAM are widely used in service robots [11]. Typically, the next phase of route planning is executed once the location is obtained. Local and global planners are commonly used to determine the obstacle-free route. First, the global planner traces a global path from the starting point to the target point. Then, the local planner traces minor routes in real-time, trying to follow the global path but avoiding possible new obstacles not contemplated in the global route [12]. This method has the disadvantage of requiring high-end sensors to obtain a high-quality map. Also, the amount of time used to generate the map increases with the amount of data used or desired map quality. It is important to note that the map must be updated to cover all areas in which the robot must navigate if the environment changes [13].

An alternative to traditional methods is AI subsets, e.g., Machine Learning (ML), Reinforcement Learning (RL), and Deep Learning (DL) [14]. Within robotics, AI has demonstrated remarkable success since it has enabled robots to be equipped with machine learning capabilities. This learning allows the agents to make optimal decisions similar to those that a human would make. ML capability allows the robot to adapt to different scenarios and situations. This makes it ideal for tasks in map-free navigation, such as path planning [15], [16].

The techniques of ML are classified into four groups: supervised, unsupervised, semi-supervised, and reinforcement [17]. Supervised ML is one in which the output label is defined. This technique searches for a function that, given an input, assigns the appropriate label. Moreover, it can be divided into classification and regression [18]. Unsupervised ML allows the analysis of raw data sets. In other words, it can generate analytical knowledge from unlabeled data [19]. Semisupervised can build a model using both labeled and unlabeled training data. This technique reduces the cost of training since most of the data used are not labeled [20]. The RL class allows the agent to learn based on the data obtained through its interaction with the environment. It has been successfully applied in many fields, including robotics, especially in constructing autonomous systems [21].

Since large amounts of data are needed to achieve an optimal policy, many studies focus on the time-varying realistic channel optimization problem of next-generation wireless networks and the exchange of training data in a secure manner, ensuring privacy and energy consumption minimiza-

tion [22]–[24].

A. Related work

In recent years, path planning techniques applying ML have been a very studied topic. Therefore, this subsection presents a brief summary of some of the most relevant papers.

Hu et al. proposed an exploration strategy using a combination of Prioritized Experience Replay (PER) and Deep Deterministic Policy Gradients (DDPG) based on a map-free collision avoidance algorithm. The data used for training were collected by remotely driving the agent, i.e., a human operator controlled the agent during the exploration process. Moreover, the agent was able to regulate both angular velocity and linear velocity depending on the area in which it is located. Once the exploration process is completed, the data collected is sampled according to the priority of each state transition, and finally, it updates its policy [25].

Yan et al. proposed a Deep Reinforcement Learning (DRL) for trajectory planning based on global situation information. This approach combined the Dueling Double Deep Q-Networks (D3QN) algorithm with the $\epsilon - greedy$ strategy and heuristic search rules [26]. Xiang et al. proposed a method for continuous control of the navigation of mobile robots based on a Long Short-Term Memory (LSTM) neural network architecture with three neural networks. Two of them acted as the target Q network, but only the network with the lowest Q value was used at each step. In addition, this method used Soft Actor-Critic (SAC) to maximize the entropy in each visited state and improve the resulting policy [27].

Tai et al. presented an asynchronous extension of DDPG, where the sample collection process is performed in another thread, allowing collecting during each step almost four times more samples than the original DDPG. Working in threads made it possible to improve the obtained policy [28]. Liu et al. proposed a method with Decentralized-Recurrent Neural Network (DS-RNN). In this method, the network used spatial and temporal relationships for the agent's decision-making in crowd navigation [29].

Jiang et al. proposed a method based on Deep Q-Learning (DQL) with randomization, ER, and HK. The proposed algorithm divided the robot state into safe, unsafe, failure, and winning states. When the agent was in a safe state, it used the $\epsilon - greedy$ strategy to determine whether the action would be determined by target-directed knowledge or by a neural network. It used an obstacle avoidance method to get as far away as possible from the obstacle in an unsafe state [30].

Maoudj et al. proposed a variation to [30], in which the states were redefined as an obstacle, target, and last action. To determine the state in which the agent was located, he divided the sensor range into five regions that were classified into two groups safe and unsafe. In the case of the "obstacle state", it was only considered an obstacle if it was in the frontal region. In this proposal, the action to be executed was determined based on the information of the last step, the target, and the classification of the regions [31]. Garrido et al. proposed an algorithm based on DQL combined with ER and $\epsilon - greedy$ strategies in which actions were taken randomly. The learning policy was designed as a function of distance and angle to the target [32].

Peng et al. proposed a path planning algorithm to minimize agent energy consumption using Double Deep Q-Learning (DDQL) combined with $\epsilon - greedy$ and ER strategies [33]. Chen et al. proposed a method to improve the stability of reinforcement learning in autonomous driving by combining DDPG with imitation learning. The function of the imitation network is to act as empirical learning using the least amount of data, which is obtained manually [34]. He et al. proposed a novel framework focused on improving the performance of vehicular networks using deep reinforcement learning, experience replay buffering, dynamic orchestration of network resources, and caching [35].

B. Contribution of the paper

This paper focuses on the significant advances of AI in the subarea corresponding to RL that allows agent learning through its interaction with the environment. It is also based on ML applications in autonomous navigation and especially on the need to develop new methods to accelerate the real-time learning of an agent, using low computational capacity and reducing the time of training used in the methods proposed so far. In this framework, a novel algorithm (DDRL-EG) for multipath planning is presented, using DDRL based on HK, ER, GA, and DP.

DDRL-EG enables the agent to avoid obstacles and reach different destinations without using a preloaded map, as is the case with traditional navigation algorithms. This requires RL, which enables the agent to learn from its environment. In addition, the combination of HK, ER, GA, and DP allows the network convergence process to be accelerated. At each step, the algorithm decides based on sensor information, the current position of the agent, and the position of the target.

The contributions of this paper are:

1) An algorithm for multipath planning with obstacle avoidance based on DDRL and soft updating.
2) A hybrid strategy to accelerate learning using SUDE. It is used to calculate the action's probability based on directed knowledge, hybrid knowledge, or autonomous knowledge.
3) A map-free obstacle avoidance navigation algorithm to generate the data used as HK.
4) A strategy to accelerate network convergence by combining the advantages of ER and GA. This allowed modifying the batch of random samples with the best sample sequences based on the best reward obtained.
5) The code source for the algorithm proposed in this paper was made in open source and is available at https://github.com/ELIZABETH1611/ddrle_ge.

The remaining of the paper is organized as follows: Section II covers the technical background. Section III proposes the different phases to obtain a hybrid algorithm using RL, ER, HK, and GA that allows an agent to plan optimal routes to a target while avoiding obstacles. Section IV discusses and compares the results obtained from the proposed algorithm with the algorithms proposed by [25], [32]. Section V presents the conclusions and future lines.

II. Technical Background

This section presents a brief introduction to RL, DDQL, ER, HK, and GA to better illustrate their combined version in the proposed algorithm.

A. Reinforcement Learning

RL is based on the agent's learning by interaction with its environment (Fig. 1). In other words, at each interaction (action) of the agent with the environment, it receives a reward. This reward is used to build the agent's behavioral policy [36].

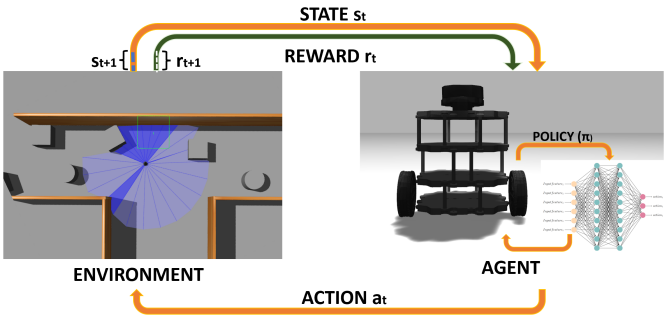


Fig. 1. The RL learning process starts when the agent observes its current state (s_t) in the environment at time (t). Then, the agent executes an action (a_t) that takes it to a new position (s_{t+1}) and receives a reward (r_{t+1}) for it. This cyclic process of receiving a positive or negative reward for each action performed is used to optimize the policy continuously.

As mentioned above, the agent's behavior is defined by its optimal policy, whose objective is to maximize the cumulative reward obtained by the agent. To construct this policy first the agent observes its current state (s_t) in the environment at time (t). Second, the agent executes an action (a_t) that takes it to a new position (s_{t+1}) in the environment. Third, it receives a reward (r_{t+1}), which is used to optimize the policy continuously [36].

The algorithm that will model the agent's policy is chosen depending on the complexity of the environment. In simple and small environments, the Q-Learning (Q-L) algorithm is used, where the policy is represented in a table. This table contains all possible states and actions that the agent can perform in that scenario.

However, in complex or larger scale environments, deep neural network algorithms are used, whose function is to approximate a function that models the agent's policy. This type of learning is known as DQL and is defined by the following properties:

1) A set of states $\boldsymbol{S}$.
2) A set of actions $\boldsymbol{A}$.
3) A set of reward $\boldsymbol{R}(s_{t+1}|s_t, a_t)$.
4) Discount factor $\gamma \in [0, 1]$.

The discount factor defines the importance of the reward within the policy used. When γ has values close to zero, the agent will give more importance to immediate rewards. This type of behavior is called myopic behavior since its goal is to perform the action that has the highest reward at the moment. On the other hand, the agent will prioritize future rewards when the values of γ are close to one. In this case, the agent will try to maximize the cumulative reward obtained until its target is reached [37], [38].

B. Double Deep Q-learning

The predecessor algorithms of DDQL are the standard DQL and Q-L algorithms. These two algorithms are susceptible to overfitting the generated policy because they use the same neural network to select and evaluate the action. That is, each of the actions executed by the agent will be based on a different policy as it is updated at each step.

The update method used by Q-L is defined in the literature as Eq. (1) [39].

$$Q(S_t, A_t) \leftarrow (1-\alpha)Q(S_t, A_t) + \alpha(R_{t+1} + \gamma\, b), \quad (1)$$

$$b = \max_{a \in A} Q(S_{t+1}, a), \quad (2)$$

where $Q(S_t, A_t)$ represents the value of the expected reward given an action A_t in current state S_t at time t and $\max_{a \in A} Q(S_{t+1}, a)$ represents the expected reward in the future (S_{t+1}) if the highest-valued action (a) is taken. The learning rate (α) allows to control the rate at which the model learns, as it regulates the weights assigned to the neural network. The lower the value, the longer it takes to converge, and the higher the value, the more sub-optimal the results may be.

The difference between Q-L and DDQL is that DDQL decouples the selection phase from the evaluation phase, thus avoiding overestimation and speeding up the convergence time of the network [40].

In DDQL, two neural networks with a completely identical architecture are used. One of them has the function of the primary network, i.e., it is in charge of choosing the action to be executed. The second neural network is the target network, whose function is to determine a target Q-value for the action selected by the first network. The primary network is trained at each time step, while the target network is only updated with the weights of the primary network every certain time. The reduction of constant updates results in a more stable policy [41].

Mathematically DDQL is expressed as shown in Eq. (3) [42].

$$\begin{aligned} Q_a(S_t, A_t) \leftarrow & (1-\alpha)Q_a(S_t, A_t) \\ & + \alpha(R_{t+1} + \gamma\, Q_b(S_{t+1}, a)), \end{aligned} \quad (3)$$

$$a = \max_{a \in A} Q_a(S_{t+1}, A_t), \quad (4)$$

as the agent initially has no information about the environment, it estimates the value of $Q(S_t, A_t)$ and updates it at each interaction. The update performed varies slightly from Q-L because DDQL employs two neural networks. Where the primary neural network that selects the best action to perform (Eq. (4)) is represented as Q_a and the secondary neural network that evaluates the selected action is represented as Q_b.

The use of two neural networks allows the values of Q_a and Q_b to be equally overestimated and can be seen as a uniform distribution. As a consequence, overestimates are no longer a problem, since noise does not affect the difference between $Q_b(S_{t+1}, a)$ and $Q_a(S_t, A_t)$ [43].

C. Experience Replay

Currently, ER is used in DRL to achieve network convergence acceleration. This method has been shown to improve the efficiency and stability of the network by storing the last transitions in a fixed-size batch called memory [44]. The storage of the samples allows them to be used several times instead of just once, resulting in more stability of the network in the training phase. In traditional approaches, the training of the network is performed using random batches of the data stored in memory [45].

D. Heuristic Knowledge

The performance of a neural network is defined by the amount of data or knowledge that the agent has about the environment. However, in traditional ML approaches, the agent has no prior knowledge. Therefore, knowledge is acquired by switching from the exploration strategy to the exploitation strategy. When the agent chooses exploration, it considers the whole environment as a target to be explored, as it seeks to maximize the knowledge obtained at the end of the exploration. Exploitation, on the other hand, assumes that the agent has some knowledge of the environment, which makes it capable of exploring promising local areas [21].

Of the exploration strategies, the most famous is $\epsilon - greedy$ strategies, also known as random strategies because the action to explore the environment is chosen randomly. This strategy is not the most efficient because the agent moves randomly within the environment, so it needs long time to complete the exploration and may explore unnecessary areas or the same areas several times. Another strategy is SUDE, which has demonstrated better performance against $\epsilon - greedy$ strategies. The difference between these two methods lies in the fact that actions are taken based on a probability distribution [46].

E. Genetic Algorithm

GA is a global algorithm whose objective is to find an optimal solution to optimization problems. In the field of AI, genetic algorithms present high performance since they maximize "fitness". Fitness in path planning is related to the path length since it represents the function that optimizes the process of finding the best path in known spaces. The optimization process by means of GA can be divided into two steps:

1) It selects a random population of possible solutions normally encoded in a string known as a "chromosome". In the field of path planning, this means that it selects random data from the training memory.
2) An operator is in charge of searching for promising areas, and the "crossover" and "mutation" operators are in charge of generating optimal possible solutions. In other words, it searches for the best routes and stores them in memory [47]–[49].

F. A* algorithm

A* algorithm is an algorithm based on heuristic searching used in path planning. It searches through all possible paths between the nearest cells but chooses the one with the lowest cost, i.e., the path with the lowest number of connections. The algorithm first divides the environment into cells, then evaluates each cell until it finds the shortest path, this means, the cell with the lowest $f(n)$ [50].

$$f(n) = g(n) + h(n), \tag{5}$$

where $g(n)$ is the length of the path from the origin point to the target and $h(n)$ represents the distance between the origin and the current cell.

In the field of robotics, A* has a high computational burden, especially in environments with a very large number of cells [51].

III. Double Deep Reinforcement Learning based on Genetic Algorithm

A. Observation space

The observation space of the environment (Fig. 2) is composed of: the observation state (s_t) (sensor data), the angle of orientation of the robot towards the target (heading), the distance to the nearest object (obstacle position), and the current distance relative to the target, calculated in polar coordinates.

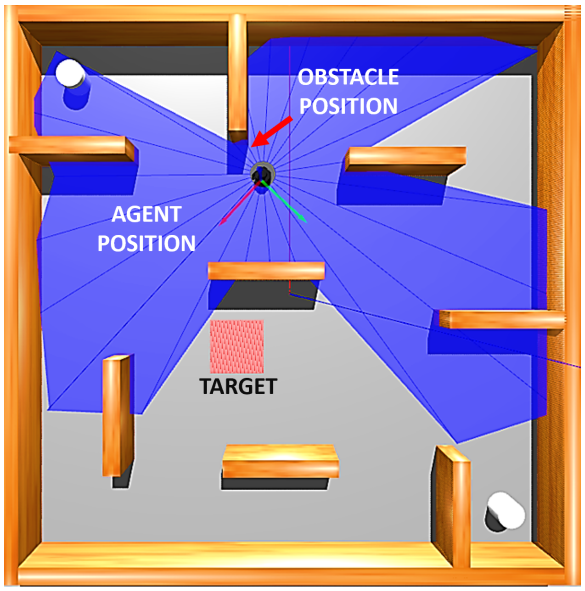


Fig. 2. In environment 2 ("Turtlebot3_plaza") the blue lines illustrate the sensor data corresponding to the 24 laser beams, the distance to the nearest object represents the nearest obstacle (wall). And the red square is the target to be reached by the agent.

The laser used for obstacle identification and positioning has a viewing angle of $360\,^\circ$ divided into 24 beams which were grouped into four regions, as shown in Fig. 3.

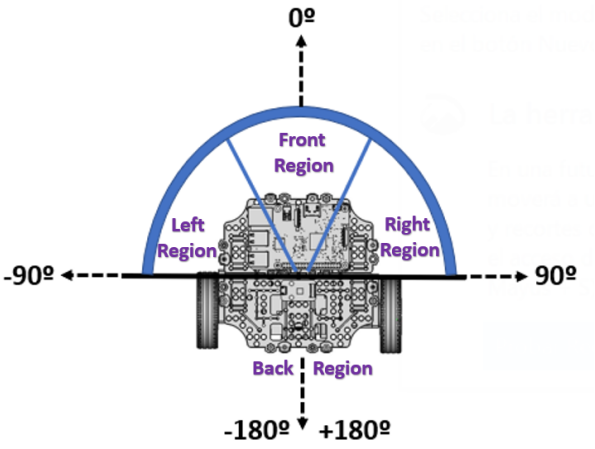


Fig. 3. The 24 laser beams are divided into 4 regions. The back region defined in the range of [-90 °,-180 °] and [+90 °,+180 °] corresponds to 12 of the 24 beams. The remaining 12 beams in the region defined in the range of [-90 °,0 °] and [+90 °,0 °] are divided between the front, left and right regions.

B. Action space

The agent is able to carry out six actions, as shown in Fig. 4, where each action corresponds to a given linear and angular velocity (Table. I).

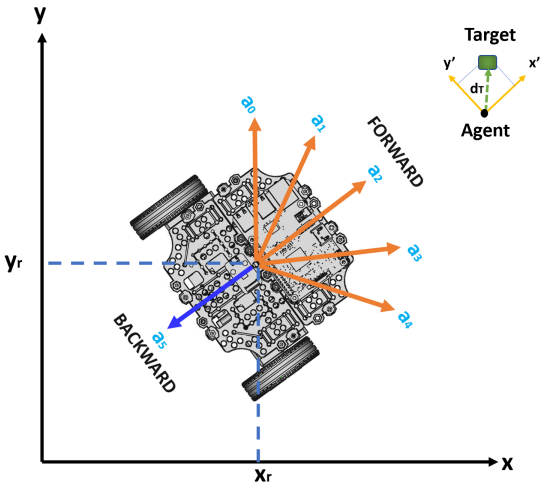


Fig. 4. The agent can perform six actions, go forward (a_2), backward (a_5), turn left (a_0 or a_1), and turn right (a_3 or a_4).

C. Reward space

The reward given to the agent at each step is defined in Eq. (6). It corresponds to the sum of six types of rewards assigned to the agent. These rewards depend on the route chosen by the agent, with higher rewards assigned to shorter routes in terms of distance traveled and time spent.

$$R_t = R_d + R_\theta + R_o + R_{p\_t} + R_g + R_c, \tag{6}$$

The reward obtained at each step by the agent consists of the reward corresponding to the distance traveled in that step

TABLE I
LINEAR VELOCITY (ν) AND ANGULAR VELOCITY(ω) CORRESPONDING TO EACH ACTION

type of action	# action	ν (m/s)	ω (rad/s)
turn	a_0	+1.2	+1.5
turn	a_1	+1.2	+0.75
forward	a_2	+1.5	0
turn	a_3	+1.2	-0.75
turn	a_4	+1.2	-1.5
backward	a_5	-1.5	0

(R_d), the reward received if the angle of rotation of the agent's path with respect to its target increased or decreased (R_θ), the reward assigned for avoiding an obstacle such as walls (R_o), the reward assigned to the time taken by the agent to reach a target (R_{p_t}), the reward given for reaching the target (R_g), and the reward (penalty) for collision (R_c). Each of these rewards is explained below.

The reward corresponding to the distance traveled at each step, R_d, is defined by Eq. (7). This defines the maximum reward that the agent could obtain when reaching its target.

$$R_d = \frac{(e^{-d_{t-1}} - e^{-d_t})}{(e^{-d_{t=0}} - 1)} \cdot m_r. \tag{7}$$

In order to obtain the maximum reward, a first approach was proposed in which only the difference between the last distance and the current distance to the goal was taken into account. The results showed that the agent tends to take longer routes to accumulate more rewards. Therefore, this approach does not work to achieve the goal of reducing the route distance. So, a new way of calculating the reward was proposed using negative exponentials and taking into account the initial distance at $t = 0$ ($d_{t=0}$) where the agent has not executed any action yet and the current distance (d_t) to the target. In addition, an auxiliary variable is used, which is assigned the value of the current distance in the previous step (d_{t-1}).

Negative exponentials allow the reward obtained by the agent to be more significant as it moves closer to the target and to have the most significant negative values as it moves away from it. Moreover, it cancels out the rewards received by the agent when it makes movements that form a looping trajectory.

Because the value of the reward obtained is in the range 10^{-1}, it was necessary to multiply it by a constant denoted as m_r. This constant sets the maximum cumulative reward as a function of the distance the agent will receive when it reaches its target.

As we can see in Fig. 5, there are different rings that represent the value of the accumulated reward that the agent will have according to the current distance to its target. For instance, at the origin point, the agent will have a reward of zero points because it has not performed any action that brings it closer or further away from its target. However, the closer it gets to its target, the more its rewards increase. In case it starts to move away from its target, the accumulated reward will be canceled. This is observed when the agent goes from 60 accumulated points to -11 points. But once it reaches the

target, it will have an accumulated reward equivalent to 300 points. Therefore, given any point of origin and destination, the agent can only have a maximum reward per distance traveled equivalent to 300 points.

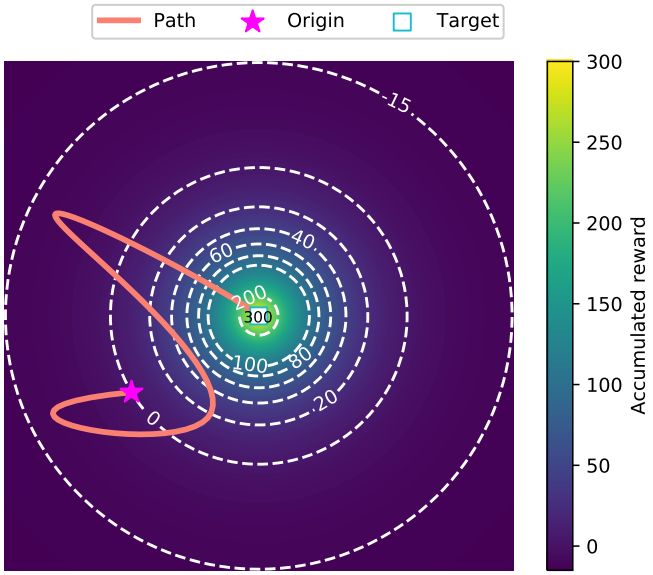


Fig. 5. The accumulated reward is displayed according to the heat map. Furthermore, an example of the agent's trajectory can be seen as the orange line. It starts with zero reward points and as it performs actions it accumulates or decreases rewards until it reaches the target with a maximum accumulated reward of 300 points.

The reward for the angle of rotation at each step, R_θ, is given by Eq. (8), where the initial heading is defined at $t = 0$, i.e., before the agent executed any action $(h_{t=0})$. The current heading is (h_t), and h_{t-1} corresponds to the value of the current heading in the previous step.

$$R_\theta = \frac{(e^{-|h_{t-1}|} - e^{-|h_t|})}{(e^{-|h_{t=0}-6|} - 1)} \cdot m_{\theta r}, \tag{8}$$

the reward was designed using the same approach as in the reward assigned to distance and the fact that the heading has a range from $-180\,^\circ$ to $+180\,^\circ$. Therefore, the absolute values of the different headings were used in order to assign the highest reward to the action that allows the agent to reach a heading close to zero (Fig. 6).

The behavior of the reward with respect to the relative angle between the agent's position with respect to the target (heading) is shown in Fig. 7. It is observed that the agent achieves a maximum cumulative reward equivalent to 100 points when its heading is zero. Before executing an action, the reward assigned to the agent is zero points, but as the heading decreases, the agent obtains a higher reward which may decrease if the heading increases.

The reward assigned for avoiding an obstacle, R_o, is specified in Eq. (9). It was defined based on a safety distance that the agent must maintain with respect to an obstacle. In this

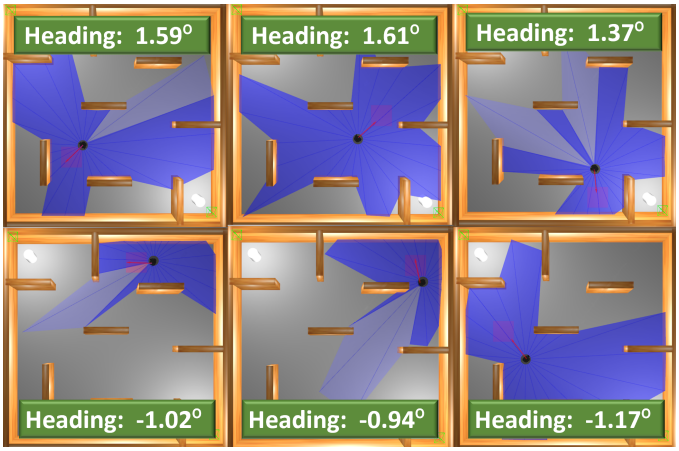


Fig. 6. The red arrow represents the direction of the agent, while the red box depicts the target. The panels show different situations when the agent's heading is directed towards the target. The respective direction in each case is written in the green boxes.

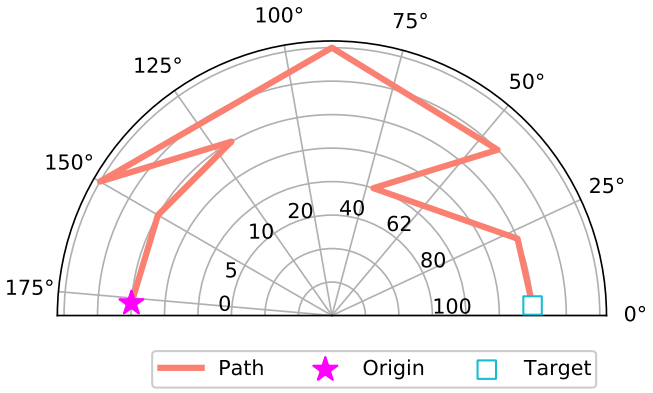


Fig. 7. The reward obtained by the agent as a function of the heading. The more the heading decreases, the higher the accumulated reward will be. The maximum reward equivalent to 30 points is reached when the heading is zero.

case, a safety distance of 0.25 m was defined.

$$\mathrm{R_o} = \begin{cases} \text{-5 points} & \text{distance to the obstacle} < \text{safe distance}, \\ \text{0 points} & \text{otherwise}. \end{cases} \tag{9}$$

The reward assigned to the time used by the agent to achieve an objective was calculated by Eq. (10). Where the Agent's threshold time (A_{th}) is calculated as the ratio between the estimated time the agent needs to reach the target (A_{tt}), and the time actually used (A_{tu}). The A_{tu} is calculated as the difference between the current time (t_t) and the time before it performs its first action $(t_{t=0})$. As the reward is related to the agent's success, it will only be calculated when the target is reached and will have a maximum value of 100 points (r_t),

otherwise, it will have a value of zero points.

$$R_{p_t} = \begin{cases} r_p & \text{if target achieved,} \\ 0 & \text{if target not achieved,} \end{cases} \tag{10}$$

where

$$r_p = \begin{cases} r_t \cdot 1 & A_{th} > 1, \\ r_t \cdot (A_{th} - 1) & A_{th} < 1, \end{cases} \tag{11}$$

$$A_{th} = \frac{A_{tt}}{A_{tu}}, \tag{12}$$

$$A_{tt} = \frac{d_{t=0}}{l_v}, \tag{13}$$

$$A_{tu} = t_t - t_{t=0}. \tag{14}$$

For a better understanding of the assigned rewards, on the one hand, Table. II summarizes the main parameters that influence the calculation of the rewards and their corresponding values. On the other hand, Table. III collects the values assigned to the actions performed by the agent from the origin $(0.0, 0.0)$ to the target $(-1.9, 1.1)$ represented in Fig. 8.

TABLE II
SUMMARY OF THE MAIN PARAMETERS INFLUENCING THE CALCULATION OF REWARDS

Description	Parameters	Values
Agent's threshold time	A_{th}	-
Agent time to target	A_{tt}	-
Agent time used	A_{tu}	-
Current time	t_t	-
Initial time	$t_{t=0}$	-
Reward for path	r_p	-
Maximun reward	m_r	300
Maximun θ reward	$m_{\theta r}$	30
Reward for time	r_t	100
Linear velocity	l_v	0.15 m/s
Reward for achieving the target	R_g	1000
Collision reward	R_c	-1000
Safety distance	-	0.25 m
Distance to reach the goal	-	0.30 m
Collision distance	-	0.18 m

D. Multipath planning algorithm

This paper proposes an algorithm that combines ER with the logic of Genetic Algorithms (GAs). The proposed algorithm is divided into four phases. In Phase 1, the use of three different memories was proposed.

The main memory (m_D) stores all the transitions (steps) that will be used for training the network. The second one $(m\_Optimal_t)$ is also part of the training but it only stores the trajectories whose target was reached in an optimal time. In other words, it stores the sequence of steps taken to reach that target. Finally, the third (m_T_{emp}) memory stores temporarily the transitions until the agent collides or reaches the target. Once one of these states occurs, the memory is cleared.

Since the policy is determined by how good or bad the input data is, this paper proposed to increase the probability of sampling the best data. Each sample represents the status of the agent at time t and is formed by state$_t$, action, reward, state$_{t+1}$, done. Where the state$_t$ is composed of scan data + [head$_t$, head$_{t=0}$, dist$_t$, distance to the obstacle] and done represents whether the action taken led the agent to a terminal state (collision) or not.

Phase 1 Reward-based data storage (Experience Replay)

```
Initialise max_r_t = 0, and memories: D, T_emp, Optimal_t
for each step do
    Save agent status in m_D and m_T_emp
    for each trajectory do
        if Process is not "done" and time is optimal then
            Save data once in m_Optimal_t
            Save data three times in m_D
        else
            Calculate maximum reward in the trajectory
(max_r_{t+1} = average reward of the trajectory)
            if max_r_{t+1} > max_r_t then
                max_r_t = max_r_{t+1}
                Copy three times from m_T_emp to m_D
                Reset m_T_emp
            else
                Reset m_T_emp
            end if
        end if
    end for
end for
```

The probability of training the neural network with specific data sequences $P(D)$ can be written as Eq. (15), where the set of all data stored in the main memory is defined as the sampling space (T), and the sequence of data with the highest reward by D.

$$P(D) = \frac{n(D)}{n(T)}. \tag{15}$$

In order to determine the best data, the third memory was used. In it, the average reward of a sequence of steps was calculated and the result was stored in a variable called max_r, which determines which sequences have the highest reward. Therefore, to increase n times the probability that the best data used for training will be sampled, it is necessary to copy n times in the main memory the sequences corresponding to the highest values of max_r_t.

In Phase 2, the batch of data to be used in each step during the training of the neural network is determined. This paper proposes to use the concept of GA, not in the classical sense applied to the weights, but focused on modifying the batch of data stored by the ER in Phase 1 to obtain a better batch of data to train the neural network. For this purpose, firstly, a batch with randomly chosen data ("chromosome") is obtained, then the last part of the batch is modified with a sequence of data corresponding to the optimal trajectory stored in the $m\_Optimal_t$ memory. This process is equivalent to the "crossover" and "mutation" operator, respectively. The combination of these data allowed the prioritization of the best data sequences while sampling each of them. The result was a faster convergence of the network.

On the other hand, in Phase 3, the data are obtained using routing and obstacle avoidance algorithm. This phase combines the A* algorithm with different processes that allow

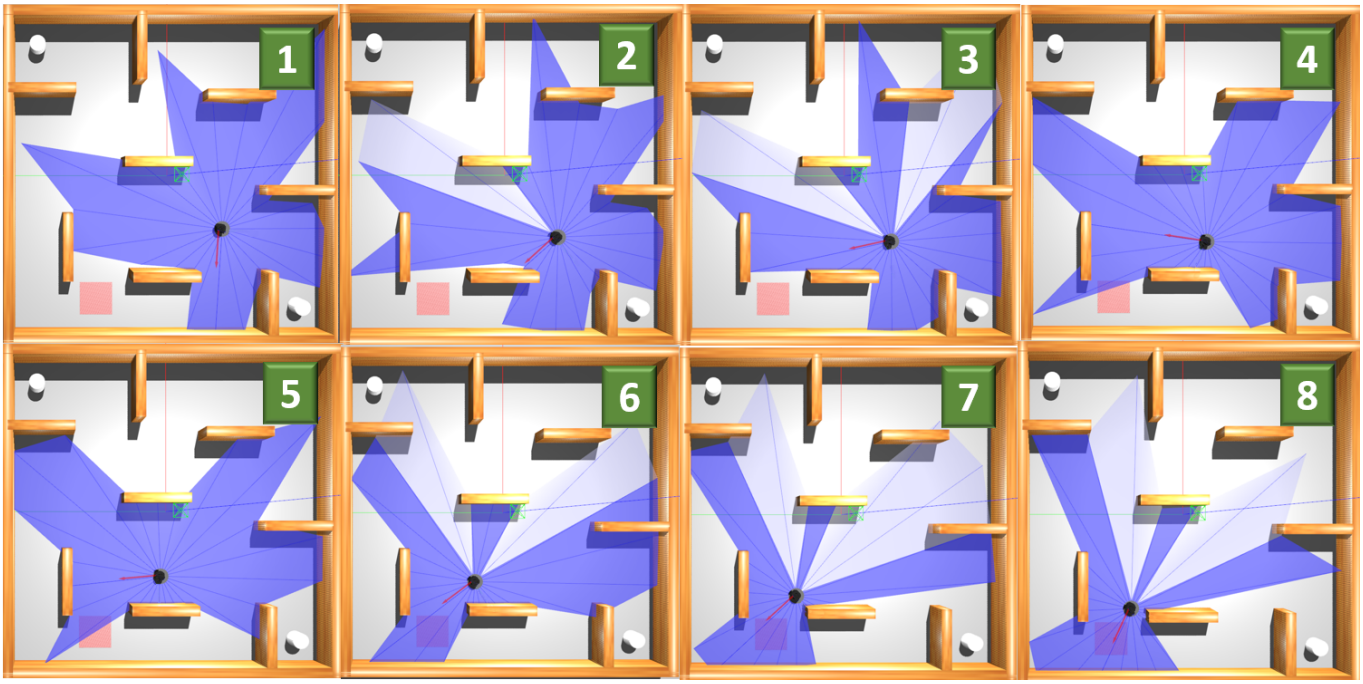


Fig. 8. Each of the subfigures shows the action executed by the agent from the origen position (0.0,0.0) to the target position (-1.9,1.1).

TABLE III
SUMMARY OF REWARD OBTAINED FOR EACH ACTION PERFORMED BY THE AGENT DURING THE ROUTE WITH ORIGIN (0,5,0,5) AND TARGET (0,7,1,9)

Number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
$d_{t=0}$ [m]	2.19	2.19	2.19	2.19	2.19	2.19	2.19	2.19	2.19
d_{t-1} [m]	2.19	2.13	2.01	1.69	1.35	1.06	0.75	0.60	0.33
d_t [m]	2.19	2.11	1.98	1.66	1.33	1.04	0.73	0.56	0.30
R_d	0.00	0.91	1.33	2.02	1.63	2.48	4.67	4.88	7.55
$h_{t=0}$ [°]	-58.27	-58.27	-58.27	-58.27	-58.27	-58.27	-58.27	-58.27	-58.27
h_{t-1} [°]	-58.27	-36.27	5.71	39.03	44.75	29.37	21.08	-5.31	-9.1
h_t [°]	-58.27	-28.06	13.31	39.99	46.96	24.16	21.85	-5.55	-9.2
R_θ	0.00	1.93 e -11	-0.098	-2.06	-9.81 e -19	9.61 e-10	-1.13 e -8	-0.03	-0.00
R_o	0.00	0.00	0.00	0.00	0.00	0.00	-5.00	-5.00	0.00
$Action$	-	3	2	2	1	1	1	2	2
A_{th}	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	19.28/17.86
R_p_t	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	100

Phase 2 Modification of the batch with the best sequence of data (Genetic Algorithm)

1: **for each** step **do**
2: $mini\_batch$ = $random\_sample$ from m_D
3: Choose random sequences from $m\_Optimal_t$ and save in $mini\_batch_1$
4: Mutate $mini\_batch$ with $mini\_batch_1$
5: **end for**

the agent to avoid collisions and go directly to its target when it has a direct vision of it.

In the Phase 4, a hybrid method was proposed using D_{pro} to determine the type of knowledge the agent would use in each episode. For this purpose, it was calculated P_a using SUDE (Eq. (16)), which defines the level of exploration or exploitation to be carried out by the agent taking into account the number of actions (n_a) and the values of P_{best}. Values of P_{best} close to zero correspond to an exploration method, and those close to one to an exploitation method.

$$P_a = P_{best} + \frac{(1 - P_{best})}{n_a} \tag{16}$$

In this paper the types of knowledge were classified as:

1) Directed knowledge: decisions were determined by Phase 3.
2) Hybrid knowledge: in this case, the actions were selected by directed knowledge or by a random choice (ρ_a). Where ρ_a (Eq. (17)) is determined by the probability associated with each of the Q-values (Q_p) (Eq. (18)).

$$\rho_a = P(a|Q_p), \tag{17}$$

where

$$Q_p = \frac{\mathbb{E}\left[Q(S_t|A_t)\right] - min(\mathbb{E}\left[Q(S_t|A_t)\right])}{\sum\left(\mathbb{E}\left[Q(S_t|A_t)\right] - min(\mathbb{E}\left[Q(S_t|A_t)\right])\right)}, \tag{18}$$

and $\mathbb{E}\left[Q\right]$ represents the expected value of Q.
3) Autonomous knowledge: decisions were made by the neural network.

Phase 3 Routing and obstacle avoidance algorithm

for each step **do**
  Get agent state and check the current process
  Choice of action according to the process in which the agent is involved
  if process is "follow_path" **then**
    The desired angle is determined by A* algorithm.
  else if process is "collision" **then**
    Go backward
    Change process to "follow_path"
  else if process is "orientation_heading" **then**
    Desired angle "zero "
    Turn to desired angle
    Change process to "follow_path"
  else if process is "driving_straight" **then**
    Go forward
    if free_distance $>$ target_distance **then**
      Keep going forward
    else
      Turn to the target
      Change process to "follow_path"
    end if
  else
    Error, the agent is not found in any of the possible processes
  end if
end for

Phase 4 Method for determining the type of knowledge to be used by the agent in each episode

The agent calculates P_a
if $P_a <= 0.25$ **then**
  Use directed knowledge
else if $0.25 < P_a < 0.95$ **then**
  Use hybrid knowledge
else
  Use autonomous knowledge
end if

In summary, to solve the map-free route planning problem, an off-policy DDRL-EG algorithm was proposed, which means that the policy that selects the action is different from the one that evaluates it.

E. Hardware Setup

The hardware used was the MSI-GL75 Leopard notebook, with an Intel(R) Core(TM) i7-10750H CPU at 2.60 GHz. The training of the different algorithms was employed only the Central Processing Unit (CPU).

F. Software Setup

Gazebo was used as the simulator and Robot Operating System (ROS) as the interface between the agent (Turtlebot3 Burguer) and the simulation interface. In addition, the tests were performed in two different environments one of them is Fig. 2 from Gazebo and the other one is Fig. 1.

Algorithm 1 Proposed DDQL-GE algorithm

Initialize learning rate (α), discount factor (γ), update rate (κ), increase factor (δ), tau(τ), time (t)
Initialize $q\_network$ and $target\_network$ with the same weights
for each episode **do**
  Initialize the environment and set the starting point and target of the agent
  Call phase 4 (Based on probability P_a choose among directed knowledge, hybrid knowledge and autonomous knowledge
  for each step **do**
    Call phase 3 (Routing and obstacle avoidance algorithm)
    Call phase 1 (Reward-based data storage)
    Call phase 2 (Modification of the batch with the best sequence of data)
    Network training
    if t % κ == zero **then**
      Update $target\_network$ weights from $q\_network$ using soft update method
      $\theta' \leftarrow \tau\theta$ + (1-τ)θ' with $\tau < 1$
    end if
  end for
end for

The sensor used was a laser distance with 24 beams, evenly distributed over a range of 360 ° and with a range distance from 0.12 m to 7 m.

G. Network architecture

The implemented network is a fully connected 5-layer deep neural network. First we have the input layer with 28 neurons corresponding to the agent state, followed by two hidden layers each with 528 neurons. Then a dropout layer was added at 20% to reduce the overfitting. And finally the output layer has 6 neurons with different Q values, which determine the action to be performed by the agent in each step.

TABLE IV
SUMMARY OF THE MAIN PARAMETERS OF THE PROPOSED ALGORITHM

Parameters	Values
Hidden Layers	2
Input layer neurons	28
Hidden layer neurons	526
Output layer neurons	6
Dropout	0.2
Replay memories	100000
Mini batch size	96
Update target	Soft Mode
Optimizer	RSMprop
Loss	MSE
α	0.0025
γ	0.9
κ	2000
δ	0.97
τ	0.1
P_a	0.00
P_{best}	0.01
P_{best_max}	0.95

IV. Results Discussion

In this section, the performance of the proposed algorithm is compared with two proposals mentioned in the section I-A using two different environments (see Fig. 1 and Fig. 2). The first algorithm presented in [32] proposed an approach using Deep Q-Network (DQN) and the second algorithm presented in [25] proposed an approach using DDPG. In this paper, these algorithms were named as DQN-Greedy and DDPG-PER, respectively.

The first subject of analysis is the area discovered by the agent in each of the algorithms, which is represented in Fig. 9a and Fig. 9b for environments 1 and environment 2.

A. Comparison of the area discovered in environment 1

As shown in Fig. 9a the area discovered by the agent using DQN-Greedy during 25 hours and 30 minutes does not cover the entire area of interest, i.e., the area where the targets are located. In addition, based on the colors shown in the heat map, it is inferred that the agent tends to explore the same area several times due to the fact that it used a random movement method to discover the environment. For instance, the central zone has a yellow color, which means that the agent has been in that zone at least 10^3 times. By contrast, DDPG-PER has a better knowledge of the environment in 12 hours and 50 minutes of training. As can be seen, the strategy uses a main path in the center of the environment, which leads the agent from the origin to all targets. Finally, the area discovered over 8 hours and 35 minutes hours using DDRL-EG shows that the agent has focused knowledge in the area of interest. The heat map shows that the agent knows several alternatives to reach the same goal and experienced a lower number of collisions, making it more robust compared to the others.

B. Comparison of the area discovered in environment 2

As shown in Fig. 9b the agent using DQN-Greedy was able to know the entire environment in 14 hours and 30 minutes. Furthermore, it is observed that random exploration has a better performance in small and delimited areas compared to large open areas, given that it does not perform explorations in areas that are not of interest. Using DDPG-PER, the agent needed 47 hours and 30 minutes of training to view the environment completely. Moreover, the heat map shows a large number of collisions throughout the environment because the agent tends to use high velocities and as it acquires knowledge of the environment it learns to regulate them. Whereas using DDRL-EG the agent only needs 6 hours and 30 minutes to have knowledge of all the areas of interest where the targets were located.

Therefore, based on the analysis performed for environments 1 and 2, the algorithm proposed in this work was able to have a better view of the areas of interest in both environments using less time than the other two algorithms. Furthermore, the obtained results show that DQN-Greedy and DDPG-PER require more time than DDRL-EG to obtain a clear view of the environment. Since in environment 1 they take a factor of 3.03 and 2.27 more time and in environment 2 a factor of 1.5 and 7.5, respectively.

The second subject of analysis is the time taken by the agent to reach his goal and the distance it has traveled. But before starting with the analysis, it is important to mention that this paper compares the first and the last time the agent reaches a specific target using the neural network. For that reason DDRL-EG does not take the value at position zero since at the beginning the algorithm uses only the directed knowledge. In addition, in order to have a frequency response as smooth as possible, a butterworth filter was applied.

C. Comparison of time and distance traveled to reach a target in environment 1

Of all possible targets, the longest path with coordinates (0.0,0.0)-(12.0,0.0) as origin and target was chosen. But since the agent using DQN-Greedy could not reach the chosen target during the 25 hours and 30 minutes, a new target with coordinates (3.0,1.0) was chosen. As shown in Fig. 10a the efficiency of DQN-Greedy has been decreasing over time. Since the distance and time to reach the target increased from 4.20 m in 29 s to 9.81 m in 66 s. This represents a low efficiency in the algorithm because the chosen route does not improve as the agent acquires more knowledge and uses more resources to reach the same objective.

DDPG-PER shows that it also did not improve its performance as the distance and time needed to reach the target was reduced from 3.74 m in 14 s to 3.97 m in 15 s. This is also indicated by Fig. 9a which shows the last path compared to the first one. In contrast, DDRL-EG showed better performance in terms of distance traveled than the other algorithms. Since it reduces the distance needed to reach the target from 4.21 m (31 s) to 3.64 m (26 s). Furthermore, the last trajectory shows that the agent offers direct paths to its target, avoiding unnecessary curves.

D. Comparison of time and distance traveled to reach a target in environment 2

The coordinates (0.0,0.0) were chosen as the origin and (0.5,1.8) as the target since it is the target with the farthest position reached more than once by DDPG-PER.

In the case of DQN-Greedy, the first time the target was reached, the agent traveled a distance of 5.06 m in 36 s but as it improved its policy it reduced the distance to 4.1 m in 28 s. Instead, the performance of DDPG-PER does not improve over time. Since the first time it reaches the target, it travels a distance of 3.58 m in 19 s and the last time it travels a distance of 4.17 m in 26 s. Finally, DDRL-EG proves to have the best performance in terms of time and distance traveled, as it is able to reduce them from 3.23 m in 23 s to 3.00 m in 21 s.

Therefore, the algorithm proposed in this paper performs better in both environments. Since it uses less training time, it can achieve all the objectives and is able to trace short and smooth paths. However, DDPG-PER obtains the best results in terms of time in the environment 1 even though the distance traveled is greater than DDRL-EG. The reason for this is that the maximum speed set in the DDPG-PER is 39% higher than in the other two strategies.

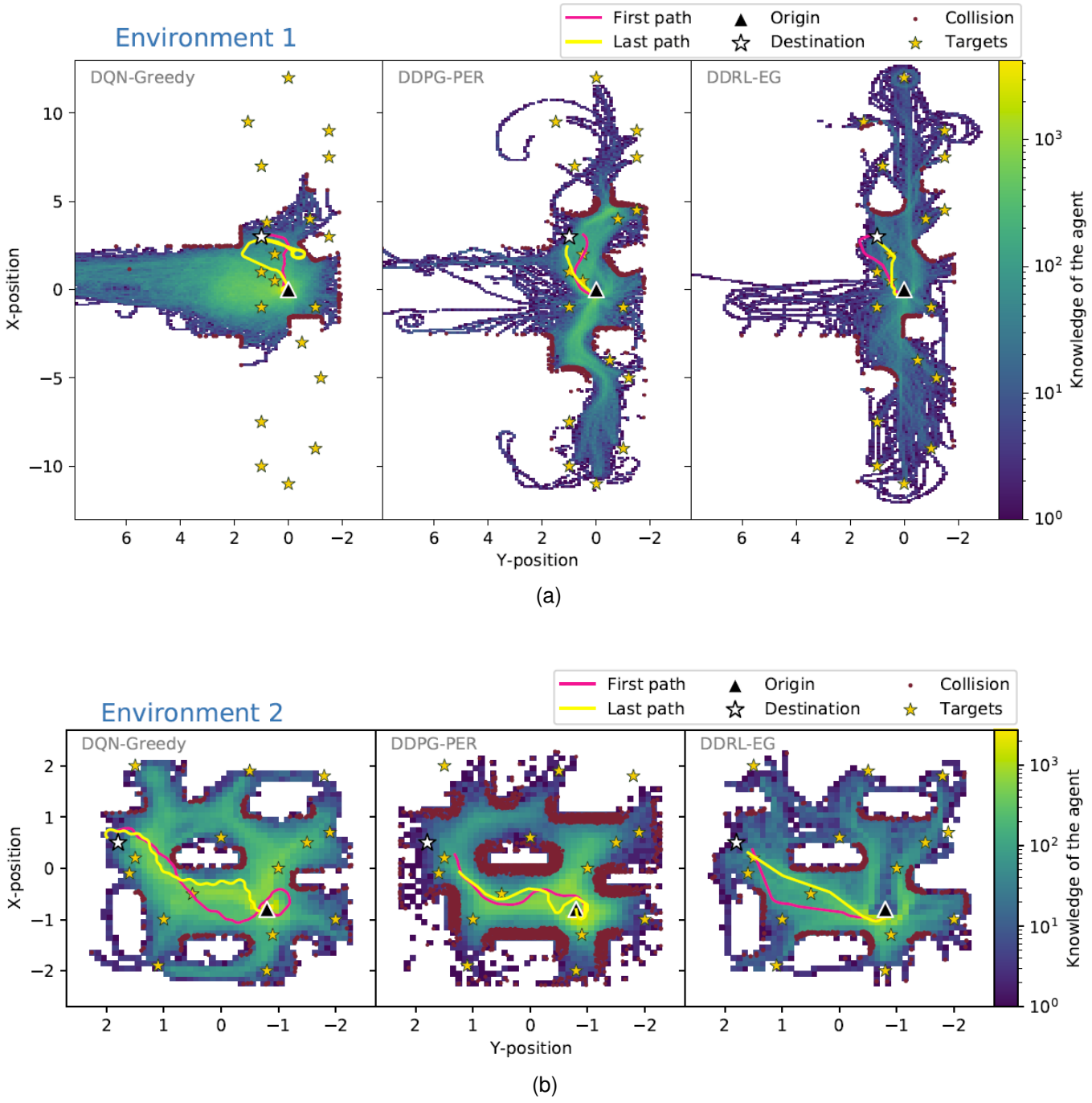
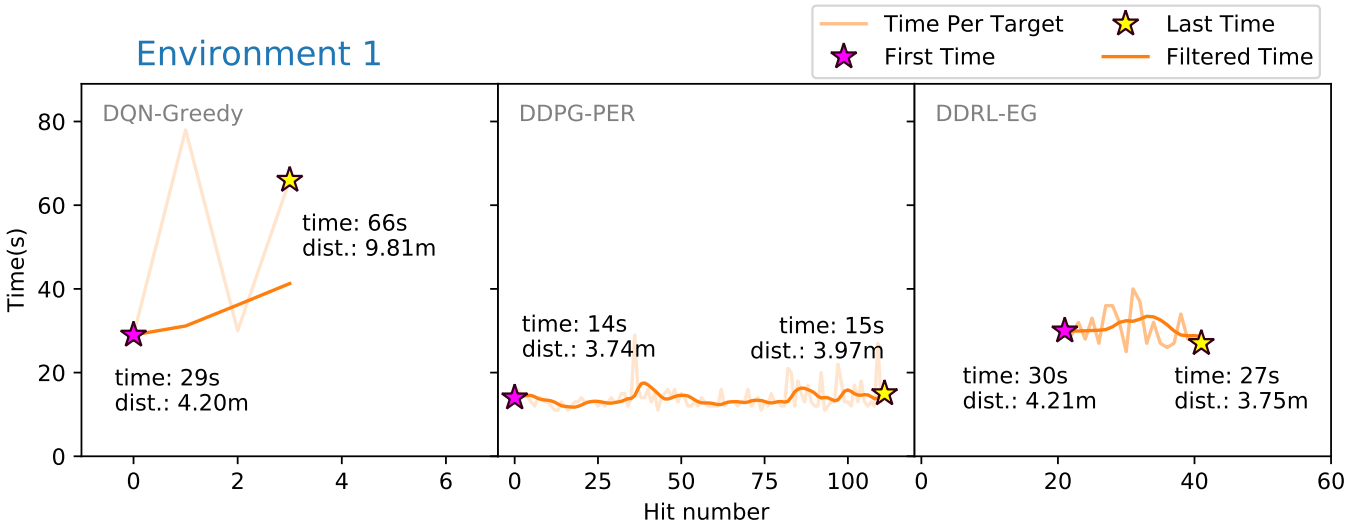


Fig. 9. In each subfigure two paths can be observed, one in magenta and the other in yellow, corresponding to the first and the last time the agent reached the target. The color bar on the right defines the heat map of each algorithm, which shows the number of times the agent was in each position within the environment. And the red dots define the positions where the agent had a collision. (a) Environment 1: DQN-Greedy, DDPG-PER, DDRL-EG. (b) Environment 2: DQN-Greedy, DDPG-PER, DDRL-EG.

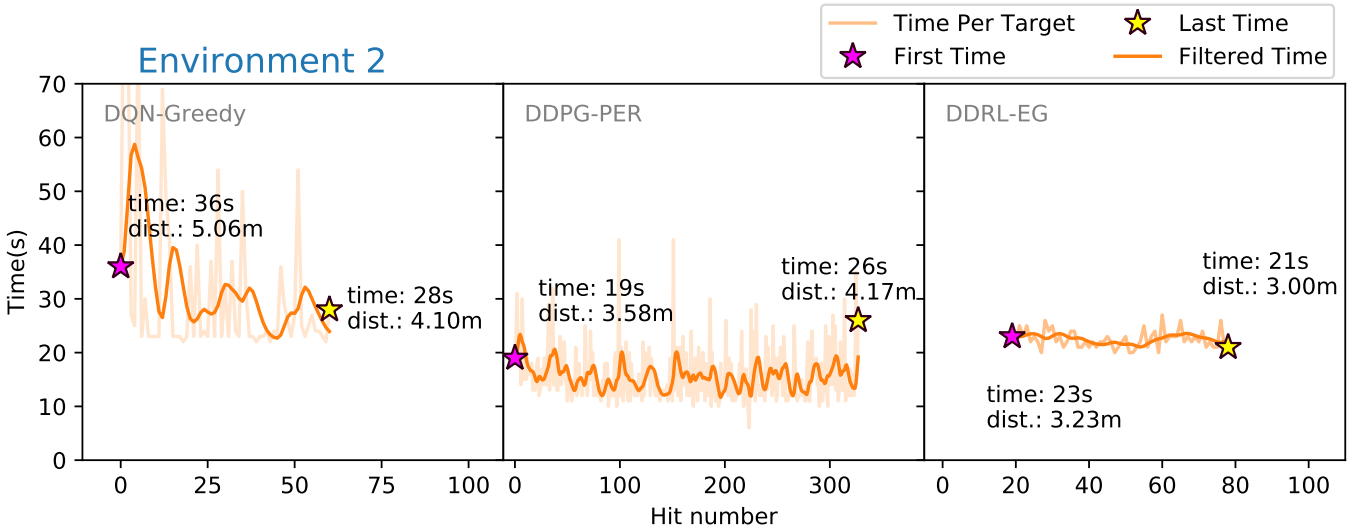
The third subject of analysis is the number of targets reached per episode (Fig. 11). An important factor to note prior to analysis is that since the speed in the DQN-Greedy and DDRL-EG algorithms is not variable, the maximum number of targets per episode will be limited to the space traveled by the agent. Where its maximum distance traveled per episode is directly proportional to the speed of the agent and the time duration of an episode. Another aspect to take into account is that the sequence in which the targets appear is random, so the farther away they are from the starting point, the lower the possible number of targets reached during the episode.

E. Comparison of number of targets reached per episode in environment 1

In this environment DQN-Greedy did not perform well, since the agent is only able to reach one target per episode



(a) Target position (3.0,1.0)



(b) Target position (0.5,1.8)

Fig. 10. Each subfigure shows the time used by the agent each time it reached a certain target (hit counter). A butterworth filter was applied to the time obtained in order to have a smoother frequency response. It also shows the distance traveled corresponding to the time spent by the agent when it reached the target for the first and last time. (a) DQN-Greedy, DDPG-PER, DDRL-EG. (b) DQN-Greedy, DDPG-PER, DDRL-EG.

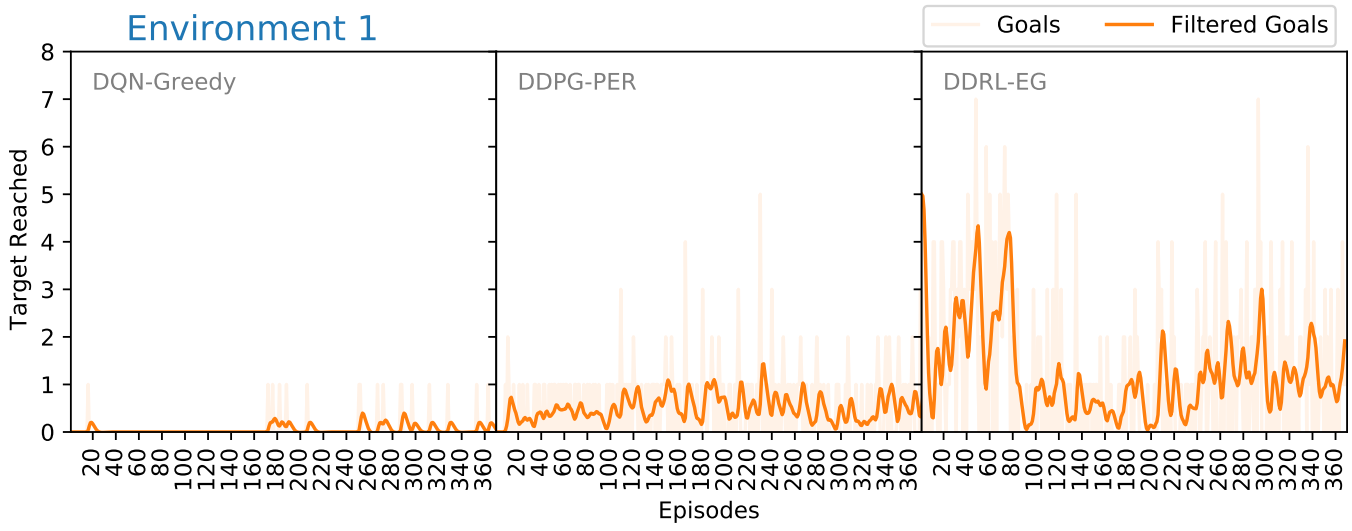
(500 s). Furthermore, it is only able to reach targets that are close to the agent's starting point. Whereas, DDPG-PER was able to achieve up to two targets per episode, showing better performance than the previous one. For last, DDRL-EG was able to achieve up to three targets per episode.

F. Comparison of the number of targets reached per episode in environment 2

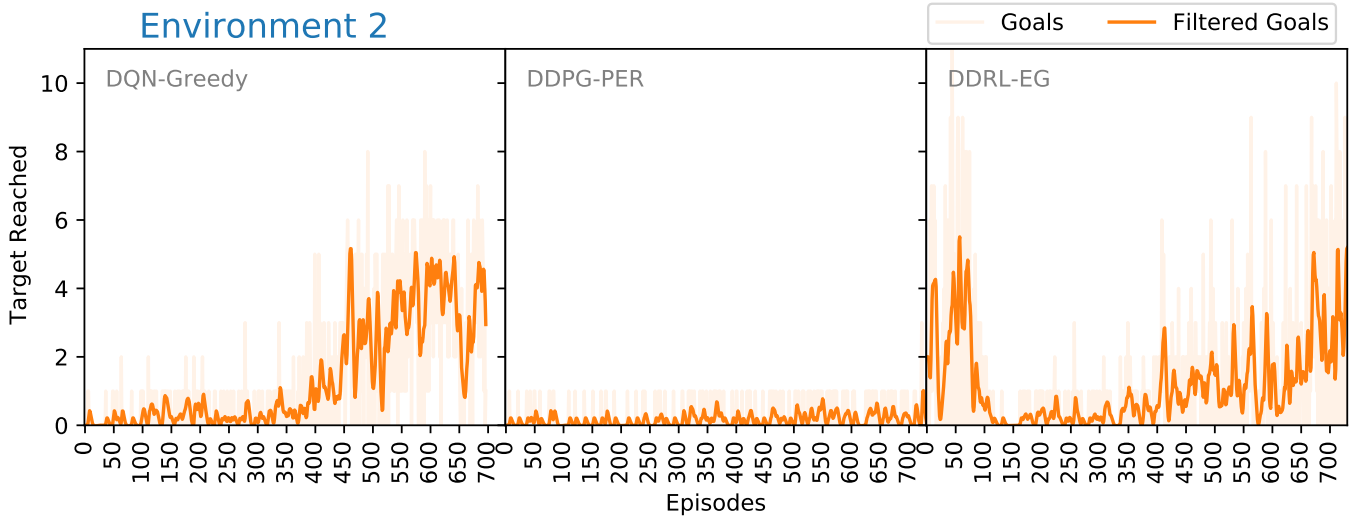
First, DQN-Greedy shows that it can improve its performance, as the number of targets reached per episode increased from one to five. DDPG-PER by contrast achieved one target per episode. The performance of this algorithm is not the most suitable in small environments with many obstacles because the agent tends to perform movements with high velocities, which causes it to collide constantly. And DDRL-EG can reach up to five targets per episode. It can be observed that the number of targets reached is initially high when using directed knowledge. Then it decreases when using hybrid knowledge and finally increases with autonomous knowledge.

Based on the upward curve of the number of goals achieved per episode presented by DDRL-EG in both environments, it can be inferred that the algorithm improves its performance over time.

The fourth subject of analysis focuses on the evolution of the performance of the algorithms, in which the percentage of achieved targets and collisions are compared throughout their learning process (Fig. 12).



(a)



(b)

Fig. 11. Number of targets achieved per episode (a) DQN-Greedy, DDPG-PER, DDRL-EG. (b) DQN-Greedy, DDPG-PER, DDRL-EG.

G. Evolution of the performance of the algorithms in environment 1

In large environments such as environment 1, in general, DQN-Greedy does not perform well. Of the 19 different targets, it was only able to reach eight of them. Furthermore, it had a success rate of 20 % only for targets whose position was close to the starting point, the rest of the targets had a success rate of less than 15 %. On the other hand, DDPG-PER showed better performance in large environments than DQN-Greedy, as it was able to reach each of the targets but with different success rates. It had a success rate of over 80 % in six of them, 60 % or more in two of them, above 40 % or more in three of them, 20 % or more in two of them, 10 % or more in one of them and less than 5 % in five of them. However, DDRL-EG showed better performance than the other two algorithms, as it was able to reach each of the targets with higher success rates using less training time. It had a success rate of 100 % in two of them, equal or higher than 100 % four of them, above 60 % in one of them, above 40 % or more in six of them and 20 % or more in six of them.

H. Evolution of the performance of the algorithms in environment 2

In small environments with several obstacles such as environment 2, DQN-Greedy performs well because out of the 16 different targets set, it was able to achieve all of them. With a success rate of 80 % or more in two of them, above 60 % in six of them, 40 % or more in six of them, above 20 % in one of them, and less than 10 % in one of them. On the other hand, DDPG-PER shows poor performance because it was only able to achieve seven objectives. The algorithm had a success rate of 100 % on the target that is close to the agent's starting point and less than 20 % on six of them. However, DDRL-EG has a success rate higher than 80 % in five of them, higher

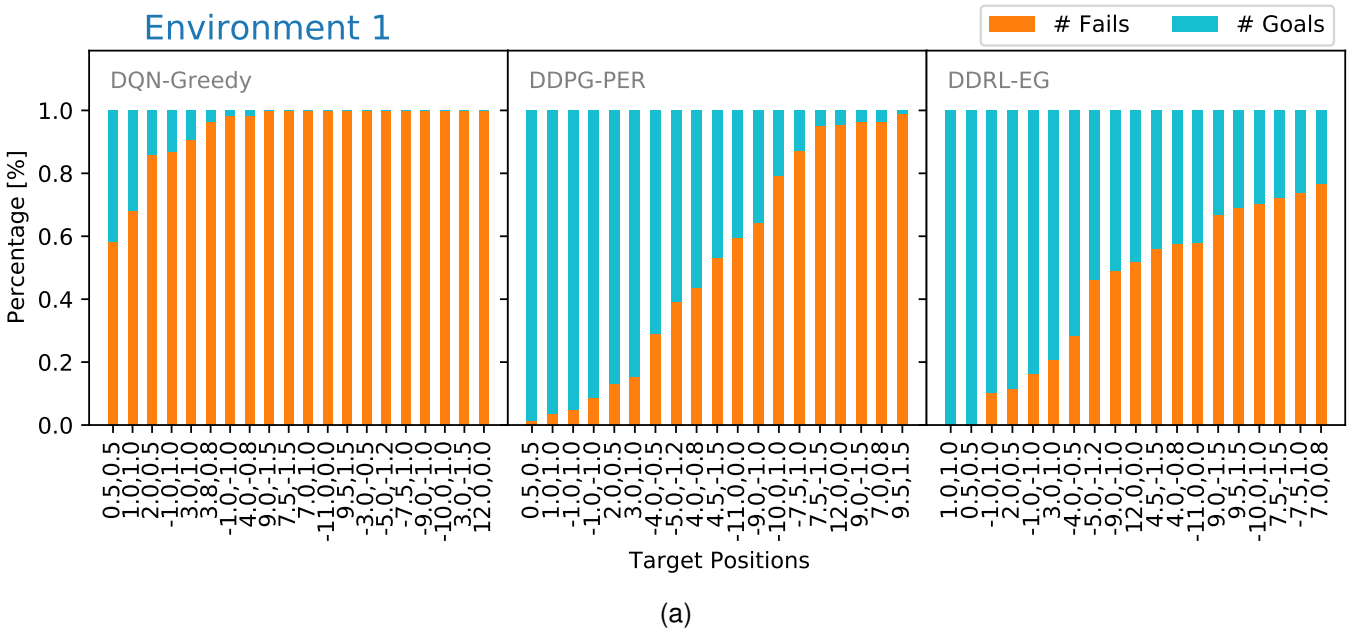


Fig. 12. Each subfigure shows the percentage of success vs. collisions of the agent during the training process of each of the algorithms. (a) DQN-Greedy, DDPG-PER, DDRL-EG. (b) DQN-Greedy, DDPG-PER, DDRL-EG.

than 60 % in four of them, and up to 20 % in seven of them. Therefore, the proposal proposed in this paper presents the best performance in both large and small environments with many obstacles.

To conclude the analysis, the training time and the number of episodes required to obtain the results mentioned above are summarised (Table V). Furthermore, once the training is completed, the success rate (s_r) of the models for three episodes has been calculated (Table VI). Each of the algorithms has a duration of 500 s per episode in addition to the same set of targets that were chosen at random.

TABLE V
SUMMARY OF TRAINING TIME AND NUMBER OF EPISODES USED IN THE ANALYSIS

Env	Algorithms	Episodes	Training Time
1	DQN-Greedy	1089	25:30
	DDPG-PER	1647	12:50
	DDRL-EG	396	8:35
2	DQN-Greedy	694	14:30
	DDPG-PER	7713	47:30
	DDRL-EG	690	06:30

TABLE VI
SUCCESS RATE CALCULATED OVER THREE EPISODES

Env	Algorithms	# Goals	# Fails	s_r [%]
1	DQN-Greedy	1	2	33
	DDPG-PER	5	3	63
	DDRL-EG	5	1	83
2	DQN-Greedy	4	1	80
	DDPG-PER	1	3	25
	DDRL-EG	12	2	86

V. CONCLUSIONS

In order to obtain smooth and short trajectories to a target without the use of a preloaded map, a novel multipath planning algorithm was proposed for mobile robots in an unknown environment. The proposed algorithm allowed the agent to optimize in terms of distance and time the traced paths to different targets. Another important feature is that the generalization capability of the algorithm can be guaranteed, since a large diversity of data, including multi-target and multi-source points, was used for training. Furthermore, the training process was performed using only the CPU, which demonstrates that the algorithm achieves high performance without requiring a high level of computational complexity.

The performance of the proposed algorithm (DDRL-EG) was evaluated in two different environments and compared with the DQN-Greedy and DDPG-PER algorithms. The obtained results showed that DDRL-EG provides short and smooth trajectories in any type of environment and also uses less training time and number of episodes. In the environment 1, DDRL-EG requires of DQN-Greedy and DDPG-PER $36.4\,\%$ and $24.0\,\%$ of the episodes, while training time requires $33.0\,\%$ and $66.8\,\%$ respectively. In the environment 2, DDRL-EG requires of DQN-Greedy and DDPG-PER $99.4\,\%$ and $8.9\,\%$ of the episodes, while training time requires $44.1\,\%$ and $13.3\,\%$ respectively. Therefore, it can be corroborated that the algorithm presented in this paper has a better performance.

Finally, the success rate of the final model of each algorithm was evaluated using random targets during three episodes. Each time a target was reached, a new target was launched until the end of the episode. The episode could end if the time was $500\,\mathrm{s}$ or also when the agent collided. The result obtained was a success rate of $33\,\%$, $63\,\%$, $83\,\%$ in environment 1 and $80\,\%$, $25\,\%$, $86\,\%$ in environment 2 for DQN-Greedy, DDPG-PER and DDRL-EG respectively. Although DDQN-Greedy has a high success rate in environment 2, it should be noted that it has a lower number of targets reached per episode since it focuses on obstacle avoidance at the expense of providing long trajectories. Therefore, it can be shown again that the proposed algorithm provides exemplary performance in any environment with fixed obstacles.

As a future line, it is proposed to transfer knowledge between different agents with different sensors incorporated to create a collaborative policy among agents.

ACKNOWLEDGMENTS

Elizabeth Palacios would like to thank the Research and Development Grants Program (PAID-01-19) of the Universitat Politècnica de València.

REFERENCES

[1] M. Martelli, A. Virdis, A. Gotta, P. Cassarà, and M. Di Summa, "An Outlook on the Future Marine Traffic Management System for Autonomous Ships," *IEEE Access*, vol. 9, pp. 157 316–157 328, 2021.

[2] J. Duan, S. Yu, H. L. Tan, H. Zhu, and C. Tan, "A Survey of Embodied AI: From Simulators to Research Tasks," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 6, no. 2, pp. 230–244, 2022.

[3] J. M. Faria and A. H. J. Moreira, "Implementation of an Autonomous ROS-Based Mobile Robot with AI Depth Estimation," in *IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society*, 2021, pp. 1–6.

[4] M. Kostrzewski, P. Varjan, and J. Gnap, *"Solutions Dedicated to Internal Logistics 4.0"*, 2020, pp. 243–262.

[5] H. He, J. Gray, A. Cangelosi, Q. Meng, T. McGinnity, and J. Mehnen, "The Challenges and Opportunities of Human-Centred AI for Trustworthy Robots and Autonomous Systems," *IEEE Transactions on Cognitive and Developmental Systems*, pp. 1–1, 2021.

[6] S. Sivashangaran and M. Zheng, "Intelligent Autonomous Navigation of Car-Like Unmanned Ground Vehicle via Deep Reinforcement Learning," *IFAC-PapersOnLine*, vol. 54, no. 20, pp. 218–225, 2021.

[7] A. Artuñedo, J. Villagra, J. Godoy, and M. D. d. Castillo, "Motion Planning Approach Considering Localization Uncertainty," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 6, pp. 5983–5994, 2020.

[8] W. Lim, S. Lee, M. Sunwoo, and K. Jo, "Hierarchical Trajectory Planning of an Autonomous Car Based on the Integration of a Sampling and an Optimization Method," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 2, pp. 613–626, 2018.

[9] I. Maurović, M. Seder, K. Lenac, and I. Petrović, "Path Planning for Active SLAM Based on the D* Algorithm With Negative Edge Weights," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 8, pp. 1321–1331, 2018.

[10] Z. Liu, "Implementation of SLAM and path planning for mobile robots under ROS framework," in *2021 6th International Conference on Intelligent Computing and Signal Processing (ICSP)*, 2021, pp. 1096–1100.

[11] S.-M. Lee, J. Jung, S. Kim, I.-J. Kim, and H. Myung, "DV-SLAM (Dual-Sensor-Based Vector-Field SLAM) and Observability Analysis," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 2, pp. 1101–1112, 2015.

[12] D. Soegiarto, B. R. Trilaksono, W. Adiprawita, E. M. Idris, and Y. P. Nugraha, "On-line Planning on Active SLAM-Based Robot Olfaction for Gas Distribution Mapping," in *2018 IEEE Asia-Pacific Conference on Geoscience, Electronics and Remote Sensing Technology (AGERS)*, 2018, pp. 1–7.

[13] H. Kanayama, T. Ueda, H. Ito, and K. Yamamoto, "Two-mode Mapless Visual Navigation of Indoor Autonomous Mobile Robot using Deep Convolutional Neural Network," in *2020 IEEE/SICE International Symposium on System Integration (SII)*, 2020, pp. 536–541.

[14] R. Trivedi and S. Khadem, "Implementation of artificial intelligence techniques in microgrid control environment: Current progress and future scopes," *Energy and AI*, vol. 8, p. 100147, 2022.

[15] J. Grönman, M. Saarivirta, T. Aaltonen, and T. Kerminen, "Review of Artificial Intelligence Applications in the ROS Ecosystem," in *2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO)*, 2021, pp. 1149–1153.

[16] I. Porres, S. Azimi, S. Lafond, J. Lilius, J. Salokannel, and M. Salokorpi, "On the Verification and Validation of AI Navigation Algorithms," in *Global Oceans 2020: Singapore – U.S. Gulf Coast*, 2020, pp. 1–8.

[17] T. Jo, *Machine Learning Foundations: Supervised, Unsupervised, and Advanced Learning*. Springer Nature, 2021.

[18] S. K. Chinnamgari, *R Machine Learning Projects: Implement supervised, unsupervised, and reinforcement learning techniques using R 3.5*. Packt Publishing Ltd, 2019.

[19] M. Usama, J. Qadir, A. Raza, H. Arif, K.-l. A. Yau, Y. Elkhatib, A. Hussain, and A. Al-Fuqaha, "Unsupervised Machine Learning for Networking: Techniques, Applications and Research Challenges," *IEEE Access*, vol. 7, pp. 65 579–65 615, 20219.

[20] T. Liu, Y. Yang, G.-B. Huang, Y. K. Yeo, and Z. Lin, "Driver Distraction Detection Using Semi-Supervised Machine Learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 7, no. 4, pp. 1108–1120, 2016.

[21] Z. Mammeri, "Reinforcement Learning Based Routing in Networks: Review and Classification of Approaches," *IEEE Access*, vol. 7, pp. 55 916–55 950, 20219.

[22] Y. He, Z. Zhang, F. R. Yu, N. Zhao, H. Yin, V. C. M. Leung, and Y. Zhang, "Deep-reinforcement-learning-based optimization for cache-enabled opportunistic interference alignment wireless networks," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 11, pp. 10 433–10 445, 2017.

[23] J. Du, F. R. Yu, G. Lu, J. Wang, J. Jiang, and X. Chu, "Mec-assisted immersive vr video streaming over terahertz wireless networks: A deep reinforcement learning approach," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9517–9529, 2020.

[24] ——, "Mec-assisted immersive vr video streaming over terahertz wireless networks: A deep reinforcement learning approach," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9517–9529, 2020.

[25] J. Hu, H. Niu, J. Carrasco, B. Lennox, and F. Arvin, "Voronoi-Based Multi-Robot Autonomous Exploration in Unknown Environments via Deep Reinforcement Learning," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 12, pp. 14 413–14 423, 2020.

[26] X. X. Yan C and W. C, "Towards Real-Time Path Planning through Deep Reinforcement Learning for a UAV in Dynamic Environments," *Journal of Intelligent & Robotic Systems*, vol. 98, no. 2, p. 297–309, 2020.

[27] J. Xiang, Q. Li, X. Dong, and Z. Ren, "Continuous Control with Deep Reinforcement Learning for Mobile Robot Navigation," in *2019 Chinese Automation Congress (CAC)*, 2019, pp. 1501–1506.

[28] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 31–36.

[29] S. Liu, P. Chang, W. Liang, N. Chakraborty, and K. Driggs-Campbell, "Decentralized Structural-RNN for Robot Crowd Navigation with Deep Reinforcement Learning," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 3517–3524.

[30] 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, no. 4, pp. 1179–1189, 2020.

[31] A. Maoudj and A. L. Christensen, "Q-learning-based navigation for mobile robots in continuous and dynamic environments," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, 2021, pp. 1338–1345.

[32] L. Garrido Alvarez, "DQN: Deep Q-Learning for Autonomous Navigation," *Conferencia de investigación de pregrado de la Universidad Estatal de Kansas*, 2019.

[33] Y. Peng, Y. Liu, and H. Zhang, "Deep Reinforcement Learning based Path Planning for UAV-assisted Edge Computing Networks," in *2021 IEEE Wireless Communications and Networking Conference (WCNC)*, 2021, pp. 1–6.

[34] S. Chen, M. Wang, W. Song, Y. Yang, Y. Li, and M. Fu, "Stabilization Approaches for Reinforcement Learning-Based End-to-End Autonomous Driving," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 5, pp. 4740–4750, 2020.

[35] Y. He, N. Zhao, and H. Yin, "Integrated Networking, Caching, and Computing for Connected Vehicles: A Deep Reinforcement Learning Approach," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 1, pp. 44–55, 2018.

[36] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep Reinforcement Learning: A Brief Survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[37] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. A. Sallab, S. Yogamani, and P. Pérez, "Deep Reinforcement Learning for Autonomous Driving: A Survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 4909–4926, 2022.

[38] S. Na, H. Niu, B. Lennox, and F. Arvin, "Bio-Inspired Collision Avoidance in Swarm Systems via Deep Reinforcement Learning," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 3, pp. 2511–2526, 2022.

[39] J. Rischke, P. Sossalla, H. Salah, F. H. P. Fitzek, and M. Reisslein, "QR-SDN: Towards Reinforcement Learning States, Actions, and Rewards for Direct Flow Routing in Software-Defined Networks," *IEEE Access*, vol. 8, pp. 174 773–174 791, 2020.

[40] V.-H. Bui, A. Hussain, and H.-M. Kim, "Double deep q -learning-based distributed operation of battery energy storage system considering uncertainties," *IEEE Transactions on Smart Grid*, vol. 11, no. 1, pp. 457–469, 2020.

[41] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, S. U. Khan, and P. Li, "A double deep q-learning model for energy-efficient edge scheduling," *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 739–749, 2019.

[42] C. E. Mariano and E. F. Morales, "DQL: A New Updating Strategy for Reinforcement Learning Based on Q-Learning," in *Machine Learning: ECML 2001*, 2001, pp. 324–335.

[43] H. Hasselt, "Double Q-learning," in *Advances in Neural Information Processing Systems*, vol. 23, 2010.

[44] W. Fedus, P. Ramachandran, R. Agarwal, Y. Bengio, H. Larochelle, M. Rowland, and W. Dabney, "Revisiting fundamentals of experience replay," in *Proceedings of the 37th International Conference on Machine Learning*, vol. 119, 2020.

[45] U. F. Siddiqi, S. M. Sait, and M. Uysal, "Deep Reinforcement Based Power Allocation for the Max-Min Optimization in Non-Orthogonal Multiple Access," *IEEE Access*, vol. 8, pp. 211 235–211 247, 2020.

[46] R. McFarlane, "A survey of exploration strategies in reinforcement learning," *McGill University*, 2018.

[47] A. Sehgal, H. La, S. Louis, and H. Nguyen, "Deep Reinforcement Learning Using Genetic Algorithm for Parameter Optimization," in *2019 Third IEEE International Conference on Robotic Computing (IRC)*, 2019, pp. 596–601.

[48] C.-C. Tsai, H.-C. Huang, and C.-K. Chan, "Parallel Elite Genetic Algorithm and Its Application to Global Path Planning for Autonomous Robot Navigation," *IEEE Transactions on Industrial Electronics*, vol. 58, no. 10, pp. 4813–4821, 2011.

[49] Q. Chen, M. Huang, Q. Xu, H. Wang, and J. Wang, "Reinforcement Learning-Based Genetic Algorithm in Optimizing Multidimensional Data Discretization Scheme," *Mathematical Problems in Engineering*, vol. 2020, 2020.

[50] F. Duchoň, A. Babinec, M. Kajan, P. Beňo, M. Florek, T. Fico, and L. Jurišica, "Path Planning with Modified a Star Algorithm for a Mobile Robot," *Procedia Engineering*, vol. 96, pp. 59–69.

[51] B. Fu, L. Chen, Y. Zhou, D. Zheng, Z. Wei, J. Dai, and H. Pan, "An improved A* algorithm for the industrial robot path planning with high success rate and short length," *Robotics and Autonomous Systems*, vol. 106, pp. 26–37, 2018.

VI. Biography Section

Elizabeth Palacios-Morocho Ing. Elizabeth Palacios-Morocho received her MSc. in Systems Technologies and Communication Networks from the Universitat Politècnica de València (UPV) in 2019, and her degree in Electronics and Telecommunications Engineering from the University National of Loja (UNL), in 2018. She is currently participating as a predoctoral researcher at the Universitat Politècnica de València at the Institute of Telecommunications and Multimedia Applications (iTEAM-UPV).

Saúl Inca Dr.-Ing. Saúl Inca received his MSc. in Systems Technologies and Communication Networks and his Ph.D. degree in Telecommunications Engineering from the Universitat Politècnica de València (UPV) in 2014 and 2019. He is currently participating as a postdoctoral researcher in European Projects at the Institute of Telecommunications and Multimedia Applications (iTEAM-UPV). His main research areas are focused on radio channel modeling and the development of visualization platforms for 5G and B5G networks.

Jose F. Monserrat Dr.-Ing. Jose F. Monserrat received his MSc. degree with High Honors and Ph.D. degree in Telecommunications Engineering from the Universitat Politècnica de València (UPV) in 2003 and 2007, respectively. He was the recipient of the First Regional Prize of Engineering Studies in 2003 for his outstanding student record, also receiving the Best Thesis Prize from the UPV in 2008. In 2009 he was awarded the best young researcher prize in Valencia. In 2016 he received the merit medal from the Spanish royal academy of engineering in the young researcher category. Jose Monserrat is a senior member of the IEEE, holds six patents, and has published more than 50 journal papers.